Reference manual

PL7 Micro/Junior/Pro

Detailed description of Instructions and Functions

TLX DR PL7 40E eng V4.0

35002664 00

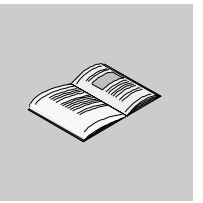
Related Documentation

Presentation

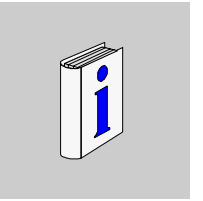
This manual is in 3 volumes:

- Volume 1: Description of the PL7 software
  - General points
  - Ladder language
  - Instruction list language
  - Structured text language
  - Grafcet language
  - DFB function blocks
  - Function modules
- Volume 2:
  - Basic instructions
  - Advanced instructions
  - Bit objects and system words
- Volume 3: Appendices
  - Differences between PL7-2/3 and PL7-Micro/Junior
  - Memory aids
  - Reserved words
  - Compliance with IEC standard 1131-3
  - OLE Automation Server
  - Performances

Table of Contents



About the book



At a Glance

Document Scope

This manual describes the instructions and objects that can be addressed in programmable Micro, Premium and Atrium PL7 programming languages.

Validity Note

The updating of this publication takes into account the functions of PL7V4 . Nevertheless it can be used to set up earlier PL7 versions.

Revision History

Rev. No.	Changes
1	first creation
2	This version include modifications on the maps : - D-SA-0001074, D-SA-0001076, D-SA-0004462

Related Documents

Title of Documentation	Reference Number
Title of related document	Reference to related document

Product Related Warnings

Contents

User Comments

We welcome your comments about this document. You can reach us by e-mail at TECHCOMM@modicon.com

Description of PL7 software

I

Presentation

What's in this spacer

This spacer introduces the PL7 software. It describes the basic elementary ides for programming Micro and Premium PL7s.

What's in this part?

This Part contains the following Chapters:

Introducing PL7 software

1

Presentation

Subject of this chapter

This chapter introduces the main characteristics of PL7 software.

What's in this Chapter?

This Chapter contains the following Maps:

Presenting PL7 software

General points

Designing and setting up applications for Micro and Premium PL7s is done using Pl7 software.

3 types of PL7 software are available:
- PL7 Micro
- PL7 Junior
- PL7 Pro

PL7 software

The following table shows the differences between the 3 types of software.

Services		PL7 Micro	PL7 Junior	PL7 Pro
Programming/ Debugging/Using		M	M/P	M/P
User function blocks	Creating	-	-	P
	Use	-	P	P
Operating screens	Creating	-	-	M/P
	Use	-	M/P	M/P
Function modules		-	-	P
Diagnostic DFB function block		-	-	P

Key:
M = Micro PL7s
P = Premium PL7s
- = not available

Write conventions

In the continuation of the document:
- the notation PL7 or PL7 software is used to name any of the 3 types of PL7 software, Micro, Junior and Pro.
- the Premium notation is used to name any of the processors TSX 57, PMX 57 and PCX 57.

Presenting PL7 languages

General points

The PL7 software has 4 programming languages:
- ladder
- instruction list
- structured text
- Grafcet

The following table gives the possible uses of the languages according to the type of PL7.

Language	Micro PL7	Premium PL7
Ladder	X	X
Instruction list	X	X
Structured text	X (only with Junior and Pro PL7 software)	X
Grafcet	X (except for macro steps)	X

Key:
X = available
- = not available

These languages can be mixed within the same application. One section of program can be written in ladder, another in text…

These languages set up:
- pre-defined function blocks (Timing, counters,….),
- application specific functions (analogue, communication, counting…),
- specific functions (time management, character strings…),

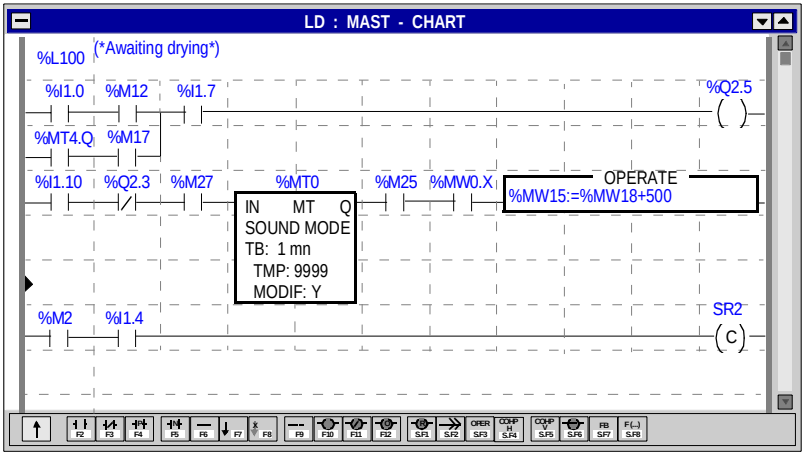
Language objects can be symbolized using the variables editor or on-line in the program editors.

The PL7 software complies with the IEC 1131-3 standard (see (See Reference Manual, Volumes 2 and 3)).

Ladder

Ladder (LD) is a graphic language. It is used to transcribe diagrams to relays. It is adapted in the combination process.

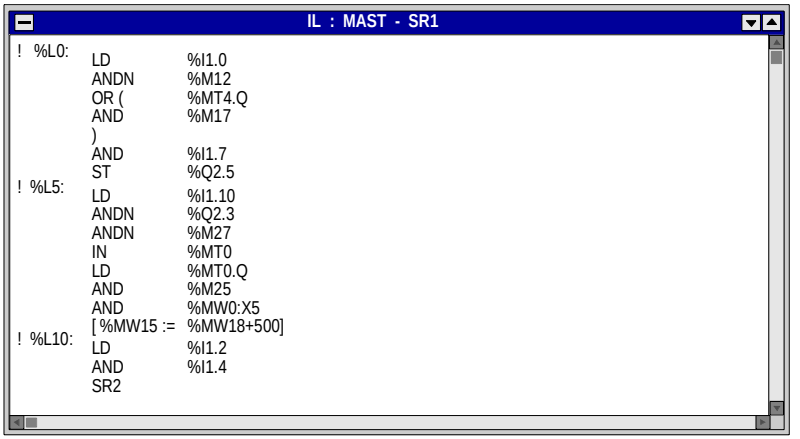
It offers basic graphic symbols: contacts, coils, blocks.
Writing numerical calculations is possible within the operation blocks.
Example of a contact network



Instruction list language

Instruction list language (IL) is a Boolean "machine" language used to write logic and numeric processes.

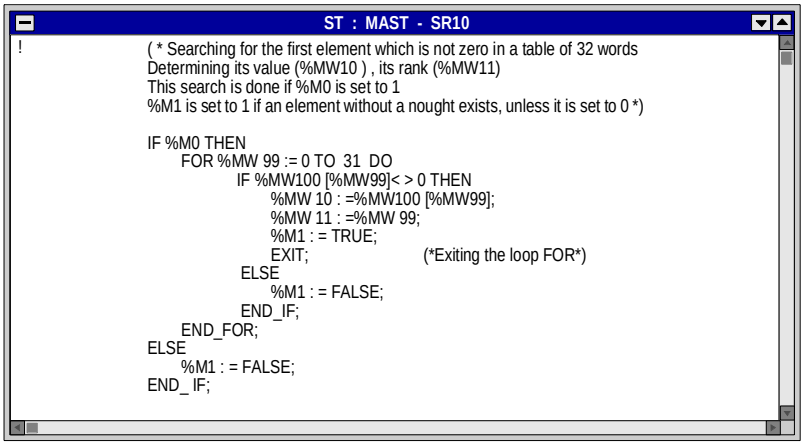
Example of how to program in instruction list language



Structured text language

Structured text language (ST) is an "IT" type language used to write structured logic and numeric processes.

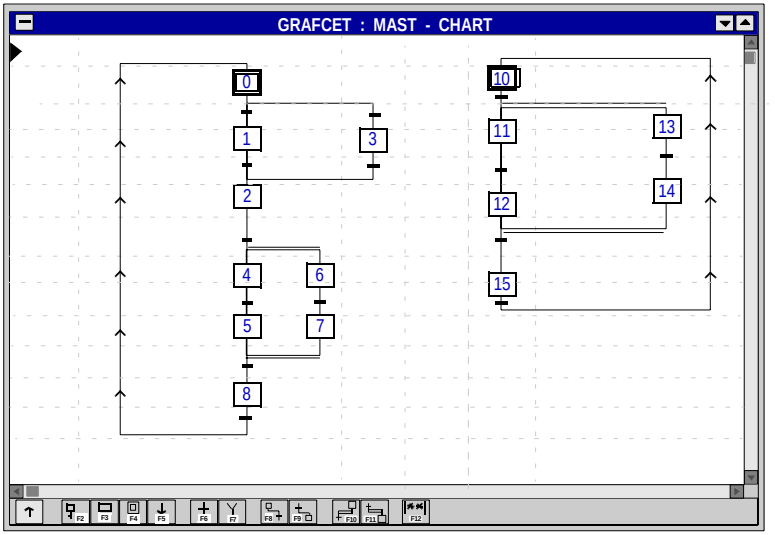
Example of how to program in structured text language



Grafcet language

Grafcet language is used to represent the operation of a sequential automatic system in a structured and graphic form.

Example of how to program in Grafcet language.



PL7 software structure

General points

PL7 software has two types of structure:

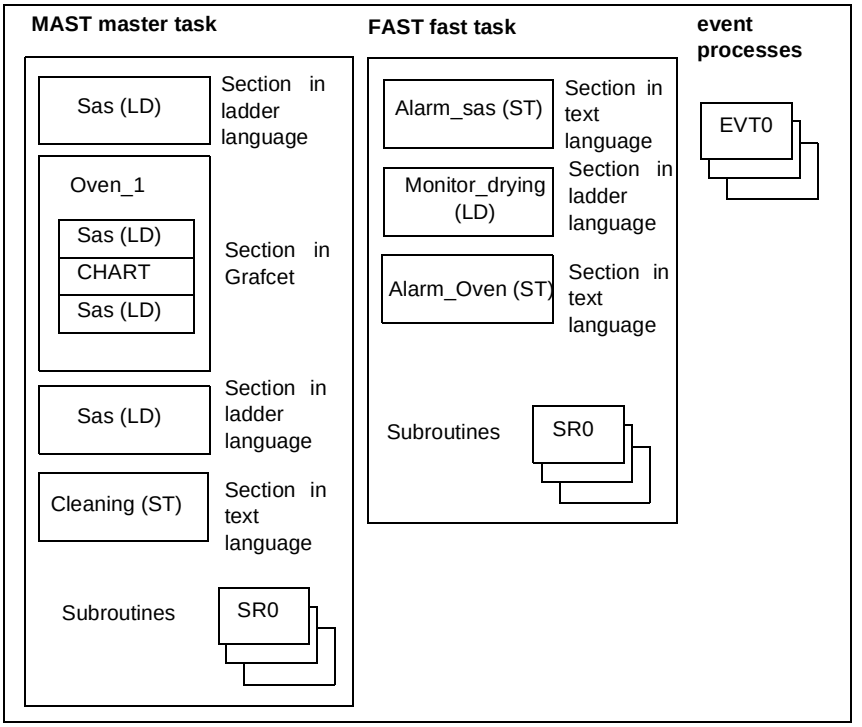
- **Mono task** this is the simplified default structure, where a single master task made up of a program of several sections and sub-programs is carried out.
- **Multi-task:** this structure, which is better suited for applications running in real time, is made up of a master task, a fast task and priority event processes.

Principle

PL7 master and fast program tasks are made up of several parts called sections and subroutines.

Each of these sections can be programmed in a language appropriate to the process to be performed.

The following illustration shows an example of dividing a PL7 program.



This division into sections is used to create a structured program and to generate or incorporate program modules easily.

Sub-routines can be called up from any section of the task to which they belong or from other sub-routines in the same task.

Function modules

General points

The PL7 Pro software is used to structure an application for the Premium PL7 in function modules.

A function module is a regrouping of program elements to carry out an automatic system function.

You can define a multi level tree structure in the automatic system application independently of the PL7 multitask structure.

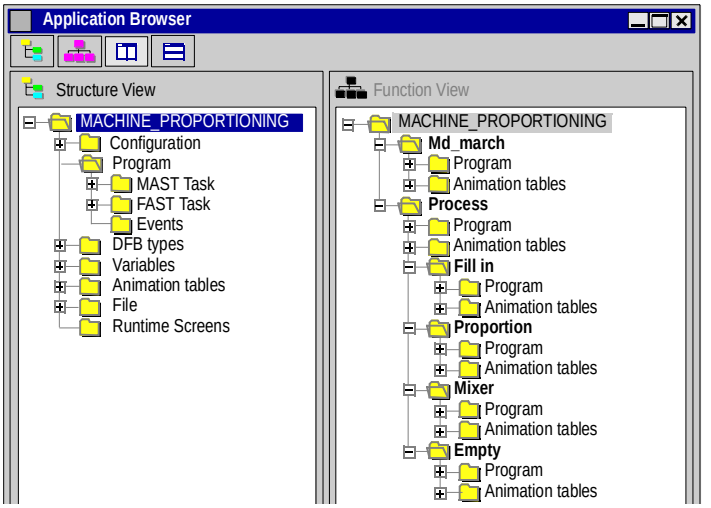
At each level you can attach program sections written in ladder, text, instruction list or Grafcet language, as well as animation tables and operating screens.

Function view

The function view in modules enables you to have a view of coherent functions with regard to the process to be ordered.

The structure view gives a view of the running order for program sections on the PL7.

The following illustration shows the 2 possible views for an application.



Services associated with the function view	The operating services are available in one view or the other. In particular, with a single command, it is possible to force a function module to run or not. In this case all sections attached to the function module are automatically forced.
Exporting/ importing function modules	You can export all or part of the tree structure in function modules. In this case all program sections on different module levels are exported.

Description of PL7 object language

2

Presentation

Subject of this chapter

This chapter describes all the PL7 language objects. These objects are used as operands in the instructions.

What's in this Chapter?

This Chapter contains the following Maps:

Definition of the main boolean objects

Description The following table describes the main boolean objects.

Bits	Description	Examples	Write access
Immediate values	0 or 1 (False or True)	0	_
Inputs/outputs	These bits are the "logic images" of the electrical states of the inputs/outputs. They are stored in the data memory and updated each time the task in which they are configured is polled. **Note:** The unused input/output bits may not be used as internal bits.	%I23.5 %Q51,2	No Yes
Internal	The internal bits are used to store the intermediary states during execution of the program.	%M200	Yes
System	The system bits %S0 to %S127 monitor the correct operation of the PLC and the running of the application program.	%S10	According to i
Function blocks	The function block bits correspond to the outputs of the function blocks or DFB instance. These outputs may be either directly connected or used as an object.	%TM8.Q	No
Word extracts	With the PL7 software it is possible to extract one of the 16 bits of a word object.	%MW10:X5	According to the type of words
Grafcet steps and macro-steps	The Grafcet status bits of the steps, macro-steps and macro-step steps are used to recognize the Grafcet status of step i, of macro-step j or of step i of the macro-step j.	%X21 %X5.9	Yes Yes

Definition of main word objects

Description The following table describes the main word objects.

Words	Description	Examples	Write access
Immediate values	These are algebraic values with the same format as single and double length words (16 or 32 bits), which are used to assign values to these words.	2542	_
Inputs/ outputs	These are the "logic images" of input/output electric values (e.g.: analogue inputs/outputs). They are stored in the data memory and are updated every time the task in which they are configured is scanned.	%IW23.5 %QW51.1	no yes
Internal	They are used to store values during the program. They are arranged inside the data space in the same memory field.	%MW10 %MD45	yes yes
Constants	They store constants or alphanumeric messages. Their content can only be written or modified by the terminal. They are stored in the same place as the program. They can therefore have the FLASH EPROM memory as a support.	%KW30	yes (only by the terminal)
System	These words ensure several functions: ● some find out about the state of the system (system and application operating time,…). ● others are used to act on the application (running modes,…).	%SW5	according to i
Function blocks	These words correspond to current parameters or values of standard function blocks or DFB instances.	%TM2.P	yes
Common	They are meant to be exchanged automatically on all stations connected to the communications network.	%NW2.3	yes
Grafcet	Grafcet words are used to find out the activity time of steps.	%X5,T	yes

Formatting values

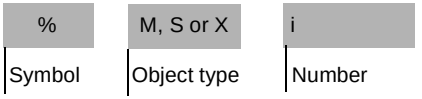
Word values can be coded in the following formats:

Type	Size	Example of value	Lower limit	Upper limit
Whole base 10	Single length.	1506	-32768	+32767
	Double length	578963	-2 147 483 648	2 147 483 647
Whole base 2	Single length.	2#1000111011111011011	2#10...0	2#01...1
	Double length	2#10001110111110110111111111011111011111	2#10...0	2#01...1
Whole base 16	Single length.	16#AB20	16#0000	16#FFFF
	Double length	16#5AC10	16#000000000	16#FFFFFFFF
Floating		-1.32E12	-3.402824E+38 (1) 1.175494E-38 (1)	-1.175494E-38 (1) 3.402824E+38 (1)
Key				
(1)	excluded limits			

Addressing bit objects

Presentation

Addressing internal, system and step bits observes the following rules:



Syntax

The table below describes the different elements that make up addressing.

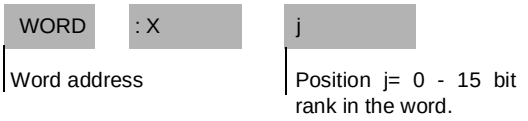
Family	Element	Values	Description
Symbol	%	-	-
Object type	M	-	**Internal bits** used to store intermediate states while the program is running. They are arranged inside the data space in the same memory field.
	O	-	**System bits** (See Reference Manual, Volume 2), these bits fulfil several functions : ● some find out about the status of the system by reading bits %Si (watch dog overflow,…). ● others are used to act on the application (initializing Grafcet,…).
	X	-	**Step bits**, step bits (See *Grafcet objects, p. 51*) give the status of step activities.
Number	i	-	The maximum number value depends on the number of objects configured.

Examples:

- %M25 = internal bit number 25
- %S20 = system bit number 20
- %X6 = step bit number 6

Bits extracted from words

PL7 software is used to extract one of the 16 bits from single length words. The address of the word is then completed by the bit row extracted according to the syntax below:



Examples:

- %MW10:X4 = bit number 4 of internal word %MW10
- %QW5.1:X10 = bit number 10 of output word %QW5.1

Note: Extracting bits from words can also be done on indexed words.

Addresing input/output module objects for the TSX 37

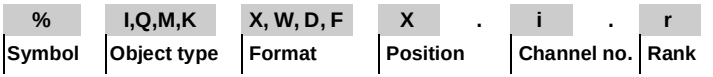
Presentation

Addressing input/output module bit and word principal objects is done geographically. That means that it depends:

- on the number (address) of the rack,
- the physical position of the module in the bac,
- the module channel number.

Illustration

Addressing is defined in the following way:



Syntax

The table below describes the different elements that make up addressing.

Family	Element	Values	Description
Symbol	%	-	-
Object type	I Q	- -	Picture of the physical input of the module, Picture of the physical output of the module, This information is exchanged implicitly each cycle of the task to which it is attached.
	M	-	Internal variable This reading or writing information is exchanged at the request of the application.
	K	-	Internal constant This configuration information is available as read only.
Format (size)	X	-	Boolean For Boolean objects the X can be omitted.
	W	16 bits	Single length.
	D	32 bits	Double length.
	F	32 bits	Floating. The floating format used is the IEEE Std 754-1985 standard (equivalent to IEC 559).

Family	Element	Values	Description
Module position	x	0 - 8 0 - 10	TSX 37 -10 TSX 37-21/22 **Note:** a module in standard format (taking up 2 positions) is addressed as 2 modules in the superimposed 1/2 format (see explanations below).
Channel no.	i	0 - 31 or MOD	Module channel number MOD: channel reserved for managing the module and parameters common to all the channels.
Row	r	0 - 127 or ERR	Position of the bit in the word. ERR: indicates a module or channel fault.

Examples

The table below shows some examples of addressing objects.

Object	Description
%I1.5	See input number 5 of the input/output module in position 1.
%MW2.0.3	Status word of row 3 of channel 0 of the input/output module on position 2.
%I5.MOD.ERR	Information on input/output module fault on position 5.

Standard format modules

They are addressed as 2 modules in superimposed 1/2 format.

For example, a module with 64 I/O occupying positions 5 and 6 is seen as 2 1/2 format modules:

- a 1/2 module of 32 inputs on position 5,
- a 1/2 module of 32 inputs on position 6,

The table below describes coding for the channel position/number according to the module.

Module	1/2 format			Standard format			
	4O	8I	12I	28I/O	32I	32O	64I/O
Channel number	0 - 3	0 - 7	0 - 11	0 - 15 (I)	0 - 15 (I)	0 - 15 (O)	0 - 31 (I)
				0 - 11 (O)	0 - 15 (I)	0 - 15 (O)	0 - 31 (O)
Addressing: Channel position/ number (x=position)	x.0 to x.3	x.0 to x.7	x.0 to x.11	x.0 to x.15	x.0 to x.15	x.0 to x.15	x.0 to x.31
				(x+1).0 to (x+1).11	(x+1).0 to (x+1).15	(x+1).0 to (x+1).15	(x+1).0 to (x+1).31

Examples The table below shows two examples of addressing standard 28 I/O module objects occupying positions 3 and 4.

Object	Description
%I3.6	Input channel module number 6
%Q4.2	Output channel module number 2

Addressing of language objects for modules remoted on the FIPIO bus

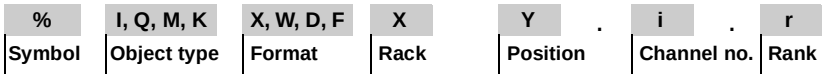
Presentation

Addressing for the main bit and word objects for modules remoted on the FIPIO bus is geographical. This means that it depends on:

- the connection point,
- the module type (base or extension),
- the channel number.

Illustration

Addressing is defined as follows:



Syntax

The table below shows the different elements which constitute addressing.

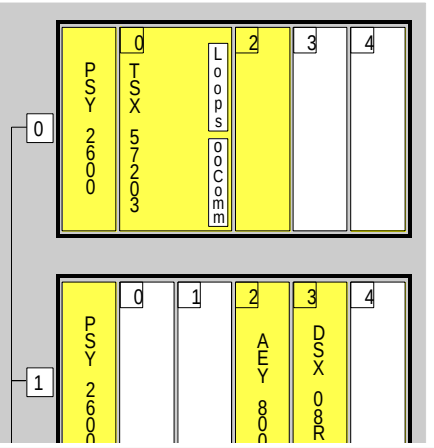
Family	Element	Values	Meaning
Symbol	%	-	-
Object type	I Q	- -	Image of the module's physical input, Image of the module's physical output, This information is exchanged automatically on each cycle of the task to which it is connected.
	M	-	Internal variable This read or write information is exchanged at the request of the application.
	K	-	Internal constant This configuration information is only accessible in read-only.
Format (size)	X	-	Boolean For boolean-type objects, the X may be omitted.
	W	16 bits	Single length.
	D	32 bits	Double length
	F	32 bits	Floating. The floating format used is that of IEEE standard 754-1985 (equivalent IEC 559).
Module/channel address and connection point	p	0 or 1	Number of the processor's position in the rack.
	2	-	Channel number of the processor's built-in FIPIO link.
	c	1 to 127	Number of the connection point.

Family	Element	Values	Meaning
Module position	m	0 or 1	0 : base module, 1: extension module.
Channel no.	i	0 to 127 or MOD	MOD: channel reserved for management of the module and the parameters shared by all channels.
Position	r	0 to 255 or ERR	ERR: indicates a module or channel fault.

Examples

The table below gives some examples of object addressing.

Object	Meaning
%MW\0.2.1\0.5.2	Position 2 status word for the image bit of input 5 of the remote input base module located at connection point 1 of the FIPIO bus.
%I\0.2.1\0.7	image bit of input 7 of the remote input base module located at connection point 1 of the FIPIO bus.
%Q\0.2.1\1.2	image bit of output 2 of the remote output extension module located at connection point 1 of the FIPIO bus.
%I\0.2.2\0.MOD.ERR	Fault information for the Momentum module located at connection point 2 of the FIPIO bus.
%Q\1.2.3\0.0.ERR	Fault information for channel 0 of module CCX17 located at connection point 3 of the FIPIO bus.



Addressing of language objects for modules remoted on the FIPIO bus

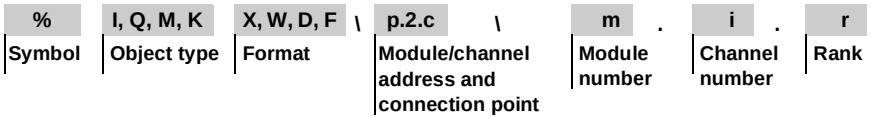
Presentation

Addressing for the main bit and word objects for modules remoted on the FIPIO bus is geographical. This means that it depends on:

- the connection point,
- the module type (base or extension),
- the channel number.

Illustration

Addressing is defined as follows:



Syntax

The table below shows the different elements which constitute addressing.

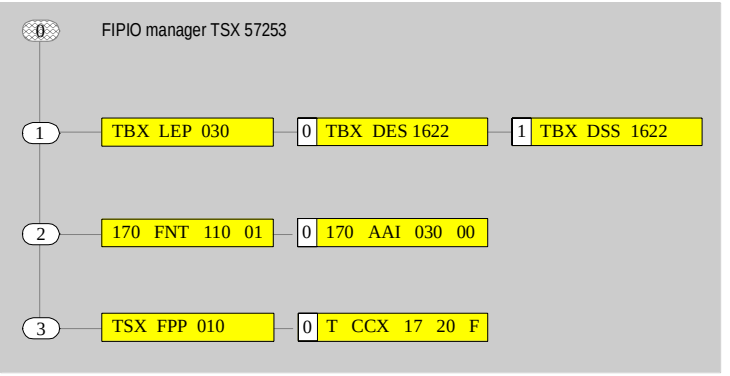
Family	Element	Values	Meaning
Symbol	%	-	-
Object type	I Q	- -	Image of the module's physical input, Image of the module's physical output, This information is exchanged automatically on each cycle of the task to which it is connected.
	M	-	Internal variable This read or write information is exchanged at the request of the application.
	K	-	Internal constant This configuration information is only accessible in read-only.
Format (size)	X	-	Boolean For boolean-type objects, the X may be omitted.
	W	16 bits	Single length.
	D	32 bits	Double length
	F	32 bits	Floating. The floating format used is that of IEEE standard 754-1985 (equivalent IEC 559).
Module/channel address and connection point	p	0 or 1	Number of the processor's position in the rack.
	2	-	Channel number of the processor's built-in FIPIO link.
	c	1 to 127	Number of the connection point.

Family	Element	Values	Meaning
Module position	m	0 or 1	0 : base module, 1: extension module.
Channel no.	i	0 to 127 or MOD	MOD: channel reserved for management of the module and the parameters shared by all channels.
Position	r	0 to 255 or ERR	ERR: indicates a module or channel fault.

Examples

The table below gives some examples of object addressing.

Object	Meaning
%MW\0.2.1\0.5.2	Position 2 status word for the image bit of input 5 of the remote input base module located at connection point 1 of the FIPIO bus.
%I\0.2.1\0.7	image bit of input 7 of the remote input base module located at connection point 1 of the FIPIO bus.
%Q\0.2.1\1.2	image bit of output 2 of the remote output extension module located at connection point 1 of the FIPIO bus.
%I\0.2.2\0.MOD.ERR	Fault information for the Momentum module located at connection point 2 of the FIPIO bus.
%Q\1.2.3\0.0.ERR	Fault information for channel 0 of module CCX17 located at connection point 3 of the FIPIO bus.



Addressing of language objects associated with AS-i bus

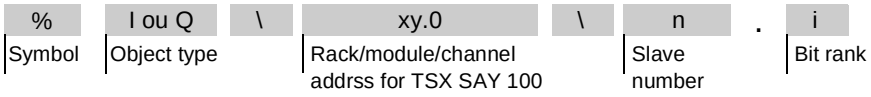
Presentation

Addressing for the main bit and word objects associated with the AS-i bus is geographical. This means that it depends on:

- the number (address) of the rack where the interface card is positioned,
- the physical position of the interface card in the rack,
- the number (address) of the slave device on the AS-i bus.

Illustration

Addressing is defined as follows:



Syntax

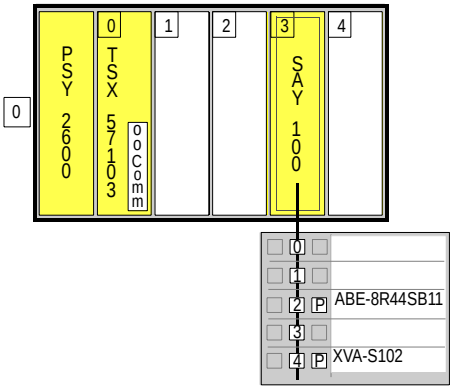
The table below describes the different elements which constitute addressing.

Family	Element	Values	Description
Symbol	%	-	-
Object type	I Q	- -	Image of the module's physical input, Image of the module's physical output, This data is exchanged automatically on every cycle of the task to which it is connected.
Rack address	x	0 or 1 0 to 7	TSX 5710/102/103/153, PMX 57102, PCX 571012). Other processors
Module position	y	00 to 14 (1)	Rack position number. When the rack number (x) is other than 0, the position (y) has a 2 digit code: 00 to 14; however, if the rack number (x) = 0, the non-meaningful zeros are deleted (from the left) from "y" ("x" does not appear and "y" takes 1 digit for values of less than 9).
Channel no.	0	-	The interface card TSX SAY 100 only has one channel.
Slave no.	n	0 to 31	Physical address of slave.
Position	i	0 to 3	Position of output or input image bit.
(1) : The maximum number of slots requires an extension rack to be used.			

Example

The table below gives some examples of object addressing.

Object	Description
%I3.0\2.2	Input 2 of slave 2, the module TSX SAY 100 being positioned at slot 3 of rack 0.
%Q3.0\4.3	Output 3 of slave 4, the module TSX SAY 100 being positioned at slot 3 of rack 0.



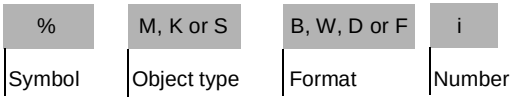
Addressing word objects

Presentation

Addressing words (except for input/output module and function block words) follows the same syntax described below.

Illustration

Addressing internal, constant and system words observes the following rules:



Syntax

The table below describes the different elements that make up addressing.

Family	Element	Values	Description
Symbol	%	-	-
Object type	M	-	**Internal words** used to store values during the program. They are arranged inside the given space in the same memory field.
	K	-	**Constant words** store constant values or alphanumeric messages. Their content can only be written or modified by the terminal. They are stored in the same place as the program. They can therefore have the FLASH EPROM memory as a support.
	S	-	**System words** (See Reference Manual, Volume 2), these words fulfil several functions: ● some find out about the state of the system by reading the %SWi words (system and application operating time,…). ● others are used to act on the application (running modes,…).

Family	Element	Values	Description
Format	B	8 bits	**Octet**, this format is used exclusively for operations on character strings.
	W	16 bits	**Single length.**: these 16 bit words can contain an algebraic value between - 32 768 and 32 767, Rang du bit 15 14 13 12 | 11 10 9 8 | 7 6 5 4 | 3 2 1 0 0 1 1 1 | 0 1 1 1 | 0 0 1 1 | 0 1 0 0 Poids fort … Poids faible
	D	32 bits	**Double length**: these 32 bit words can contain an algebraic value between -2 147 483 648 and 2 147 483 647. These words are stored in the memory on two consecutive single length words. Poids faible 15 14 13 12 | 11 10 9 8 | 7 6 5 4 | 3 2 1 0 0 1 1 1 | 0 1 1 1 | 0 0 1 1 | 0 1 0 0 0 0 1 1 | 0 1 1 0 | 0 1 0 1 | 0 0 1 0 Poids fort
	F	32 bits	**Floating**: the floating format used is the IEEE Std 754-1985 standard (equivalent to IEC 559). The length of the words is 32 bits, which corresponds to single precision floating numbers. Examples of floating values: 1285.28 12.8528E2
Number	i	-	The maximum number value depends on the number of objects configured.

Examples:

- %MW15 = single length internal word number 15
- %MF20 = floating internal word number 20
- %KD26 = constant double word number 26
- %SW30 = system word number 30

Addressing words on the network

Addressing words on the network is described in the manual Application communication.

Otherwise networks use specific objects: common words. These are single length object words (16 bits) common to all stations connected on the communications network.

Addressing: %NW{i.j}k

with: i = 0 - 127 network number, j = 0 - 31 station number and k= 0 - 3 word number

Overlay rules

Principles

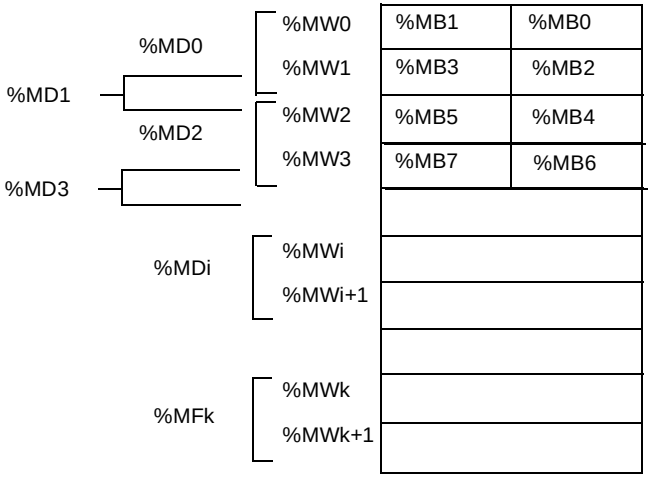
Bytes, single, double length and floating words are arranged inside the space given in the same memory field.

Thus overlay is possible between:

- the double length word %MDi and the single length words %MWi and %MWi+1 (the word %MWi being the least significant and the word %MWi+1 the most significant of the word %MDi),
- the single length word %MWi and the bytes %MBj and %MBj+1 (with j=2 x i),
- the floating %MFk and the single length words %MWk and %MWk+1.

Illustration

This illustration shows overlay of internal words.



Examples

- %MD0 corresponds to %MW0 and %MW1 (see illustration above).
- %MW3 corresponds to %MB7 and %MB6 (see illustration above).
- %KD543 corresponds to %KW543 and %KW544.
- %MF10 corresponds to %MW10 and %MW11.

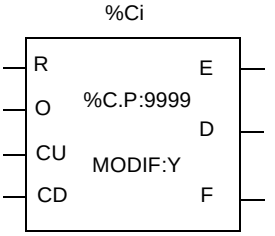
Function block objects

General points

Function blocks set up bit objects and specific words that can be accessed by the program.

Example of a function block

The following illustration shows a counter/count down function block.



Counter/count down block

Bit objects

They correspond to block outputs. These bits can be accessed by Boolean test instructions.

Word objects

They correspond:

- to block configuration parameters. These parameters can be accessed (e.g.: pre-selection parameters) or not (e.g.: base time) by the program,
- to current values (e.g.: %Ci.V counting value in progress).

List of function block objects that can be accessed by the program

The following table describes all the function block objects.

Function blocks	Symbol	No. of Maxi Micro	No. of Maxi Premium	Types of objects	Description	Address	Write access
Timer	%TMi	64	255 (128 default)	Word	Current value	%TMi.V	no
					Preset value	%TMi.P	yes
				Bit	Timer output	%TMi.Q	no
Counter/Down counter	%Ci	32	255 (64 default)	Word	Current value	%Ci.V	no
					Preset value	%Ci.P	yes
				Bit	Overflow output (empty)	%Ci.E	no
					Pre-selection output reached	%Ci.D	no
					Overflow output (full)	%Ci.F	no
Monostable	%MNi	8	255 (32 default)	Word	Current value	%MNi.V	no
					Preset value	%MNi.P	yes
				Bit	Overflow output (empty)	%MNi.R	no
Word register	%Ri	4	255 (4 default)	Word	Access to register	%Ri.I	yes
					Register output	%Ri.O	yes
				Bit	Register output full	%Ri.F	no
					Register output empty	%Ri.E	no
Drum	%DRi	8	255 (8 default)	Word	Number of step in progress	%DRi.S	yes
					Status of step j	%DRi.Wj	no
					Activity time of step	%DRi.V	no
				Bit	Last defined step in progress	%DRi.F	no
Series 7 timeer	%Ti	64	255 (0 default)	Word	Current value	%Ti.V	no
					Preset value	%Ti.P	yes
				Bit	Output in progress	%Ti.R	no
					Timer output elapsed	%Ti.D	no

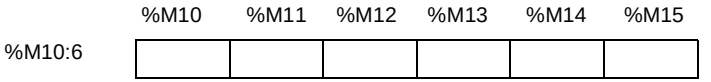
Note: the total number of timers %TMi + %Ti is limited to 64 for a Micro, and 255 for a Premium.

Table type PL7 objects

Bit table

Bit tables are sets of adjacent bit objects of the same type and defined length: L

Example of bit tables : %M10:6



This table defines bit objects which can be put into the form of a bit table.

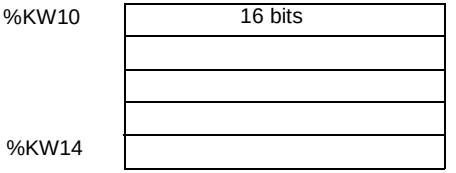
Type	Address	Example	Write access
Discrete input bits	%Ix.i:L	%I25.1:8	No
Discrete output bits	%Qx.i:L	%Q34.0:16	Yes
Internal bits	%Mi:L	%M50:20	Yes
Grafcet bits	%Xi:L, %Xj.i:L	%X50:30	No

Note: The maximum length of the tables depends on the object type
- **For discrete input/output bits**: the maximum size depends on the modularity (number of module inputs/outputs).
- **For internal or Grafcet bits**: the maximum size depends on the sized defined when configuring.

Word tables

Word tables are sets of adjacent bit objects of the same type and defined length: L

Example of word tables: %KW10:5



This table defines word objects which can be put into the form of a word table.

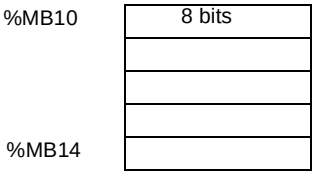
Type	Format	Address	Example	Write access
Internal words	Single length.	%MWi:L	%MW50:20	Yes
	Double length	%MDi:L	%MD30:10	Yes
	Floating point	%MFi:L	%MF100:20	Yes
Constant words	Single length.	%KWi:L	%KW50:20	No
	Double length	%KDi:L	%KD30:10	No
	Floating point	%KFi:L	%KF100:20	No
Grafcet words	Grafcet words	%Xi.T:L, %Xj.i.T:L	%X12.T:8	No
System words	System words	%SWi:L	%SW50:4	Yes

Note: The maximum lengths of the tables depend on the object type.
- **For internal, constant or Grafcet words**: the maximum size depends on the sized defined when configuring.
- **For system words**: only the words %SW50 to 53 can be set out in the form of a table.

Character strings

Character strings are sets of adjacent bytes of the same type and defined length: L

Example of character string: %MB10:5



This table defines bit objects which can be put into the form of a character string.

Type	Address	Example	Write access
Internal words	%MBi:L	%MB10:8	Yes
Constant words	%KBi:L	%KB20:6	Yes

Note: the index i must be even.

Indexed objects

Direct addressing

Addressing objects is called direct when the address of these objects is fixed and defined when the program was written.

Example: %MW26 (internal word with address 26)

Indexed addressing

In indexed addressing, the object's direct address is completed with an index: the contents of the index is added to the object address.

The index is defined either by:
- an internal word %MWi
- a constant word %KWi
- an immediate value

There is no limit to the number of "index words".

This type of addressing is used to run through a set of objects of the same type (internal words, constant words…), successively: the contents of the index is added to the object address.

Example:
MW108[%MW2] : direct address word 108 + contents of word %MW2.
If the word %MW2 has the value 12 for its content, writing %MW108[%MW2] is therefore equivalent to writing %MW120.

Describing objects that can be indexed

The following table defines the objects that can be indexed.

Type	Format	Address	Example	Write access
Input bits	Boolean	%Ixy.i[index]	%I21.3[%MW5]	No
Output bit	Boolean	%Qxy.i[index]	%Q32.4[%MW5]	Yes
Internal bit	Boolean	%Mi[index]	%M10[%MW5]	Yes
Grafcet bit	Boolean	%Xi[index]	%X20[%MW5]	No
		%Xj.i[index]	%X2.3[%MW5]	No
Internal words	Single length.	%MWi[index]	%MW30[%MW5]	Yes
	Double length	%MDi[index]	%MD15[%MW5]	Yes
	Floating point	%MFi[index]	%MF15[%MW5]	Yes
Constant word	Single length.	%KWi[index]	%KW50[%MW5]	No
	Double length	%KDi[index]	%KD50[%MW5]	No
	Floating point	%KFi[index]	%KF50[%MW5]	No

Type	Format	Address	Example	Write access
Grafcet words	Single length.	%Xi .T[index]	%X20 .T[%MW5]	No
		%Xj.i .T[index]	%X2.3 .T[%MW5]	No
Word table		%MWi[index]:L	%MW50[%MW5]:10	Yes
		%MDi[index]:L	%MD40[%MW5]:10	Yes
		%KWi[index]:L	%KW70[%MW5]:20	No
		%KDi[index]:L	%KD80[%MW5]:10	No

Note: The maximum values of the indexes depend on the types of object indexed.
- **For discrete input/output bits**: 0<i+index<m (m being the maximum number of module inputs/outputs).
- **For all other objects** (except double length or floating objects): 0<i+index<Nmax , Nmax = maximum size depends on the size defined in the configuration.
  For double length or floating words: 0<i+index<Nmax-1.

Indexing double words

The real address = direct address of the indexed double word + twice the content of the index word.

Example: %MD6[%MW100]
Si %MW100=10, le word addressed will be 6 + 2 x 10 -->%MD26.

Index overflow

The index will overflow as soon as the address of an indexed object exceeds the limits of the field containing the same type of object, i.e. when:
- object address + index content lower than zero,
- object address + index content greater than the maximum limit configured

If the index overflows, the system resets the system bit %S20 to 1 and the object is assigned with an index value of 0.

The following table gives the conditions for setting the system bit %S20 to 1 and 0.

Set to 1	Reset to 0
• set to 1 by the system when the index overflowed	• set to 0 by the user after modifying the index

Grafcet objects

Bit objects

The following table summarizes all the Grafcet bit objects available and describes their role.

Type	Description
%Xi	status of step i of the main graph (Chart).
%XMj	status of the Grafcet macro step j.
%Xj.i	status of the i step of the Grafcet j macro step
%Xj.IN	status of the input step of the macro step
%Xj.OUT	status of the output step of the macro step

These bits are set to 1 when the step or the macro step is active, to 0 when it is inactive.

Word objects

The following table summarizes all the Grafcet word objects available and describes their role.

Type	Description
%Xi.Ti	activity time for Grafcet step i.
%Xj.i.T	activity time for the i step of the Grafcet j macro step
%Xj.IN.T	activity time for step i of macro step j which allows it to find out about the status of step i of the Grafcet macro step j.
%Xj.OUT.T	activity time for the input step of the macro step
%Xj.OUT	activity time for the output step of the macro step

These words are incremented every 100 ms and take a value of between 0 and 9999.

Symbolizing

Role

Symbols are used to address PL7 language objects by name or customized mnemonics.

Syntax

A symbol is a string of a maximum of 32 alphanumeric characters the first character of which is alphabetic.

A symbol begins with a capital letter, the others are in lower case (e.g.: `Burner_1`).

When it is being entered the symbol can be written in capitals or lower case (e.g.: `BURNER_1`), the program automatically puts the symbol in the correct form.

Characters that can be used

The following table provides the characters that can be used when creating symbols.

Type	Description
alphabetic capitals	"A - Z" and the following letters "ÀÁÂÃÄÅÆÇÈÉÊËÌÍÎÏDÑÒÓÔÕÖØÙÚÛÜYþ"
alphabetic lower case	"a - z" and the accented letters àáâãäåæçèéêëìíîïñòóôõöØùúûüypßÿ
numerical	figures from 0 - 9 (they cannot be in first place of the symbol).
the character "_"	it cannot be either at the beginning of the symbol nor at the end.

A certain number of words are reserved by the language and cannot be used as symbols, see (See Reference Manual, Volume 3).

Editing symbols

Symbols are defined and associated with language objects by the variables editor. A comment of 508 characters can be associated with each symbol.

Symbols and their comments are stored on the terminal hard disk and not in the PL7.

Objects which can be made into symbols

All PL7 objects can be symbolized except for table type structured objects and indexed objects, but if the base object or index is symbolized the symbol is used in the structured object.

Examples:

- if the word %MW0 has "Temperature" for a symbol, the word table %MW0:12 is symbolized by `Temperature:12`,
- the word %MW10 has`Oven_1` for a symbol, the indexed word %MW0[%MW10] is symbolized by `Temperature[Oven_1]`.

Object bits extracted from words, bits or function block words can be symbolized but if they are not symbolized they can inherit the symbol from the base object.

Examples:

- if the word %MW0 has `Pump_state` for a symbol and if the bit extracted from the word %MW0:X1 is not symbolized, it inherits the symbol from the word, %MW0:X1 has as its symbol: `Pump_ state:X1`,
- if the function block %TM0 has for its symbol `Time_oven1` and if the output %TM0.D is not symbolized, it inherits the block symbol, %TM0.D has as its symbol: `Time_oven.D`.

Object which are only symbolic

DFB function block parameters can only be accessed in the form of symbols. These objects are defined by the following syntax:

Name_DFB.Name_parameter

The elements have the following meaning and characteristics.

Element	Maximum number of characters	Description
Name_DFB	32	name given to the DFB function block used.
Name_parameter	8	name given to the output parameter or public variable.

Example: `Gap.check` for the gap `output` of the DFB instance named `Check`.

Presymbolized objects

Role

Certain application specific functions (example: counting, axes request, …) support an automatic symbolization of the objects which are linked to them.

If you give the generic symbol of the module's %CHxy.i channel, all of the symbols of the objects linked to this channel can then be automatically generated on request.

Syntax

These objects are symbolized with the following syntax:

PREFIX_USER_SUFFIX_MANUFACTURER

The elements have the following meaning and characteristics:

Element	Maximum number of characters	Description
PREFIX_USER	12	generic symbol given to the channel by the user
SUFFIX_MANUFACTURER	20	part of the symbol which corresponds to the bit object or word of the channel given by the system

Note: As well as the symbol, a manufacturer's comment is automatically generated, this comment recalls succinctly the object's role.

Example

This example shows a counting module situated in slot 3 of the automatic tray.

If the generic symbol (prefix-user) given to channel 0 is `COMPT_PIECES`, the following symbols are automatically generated.

Address	Type	Symbol	Comment
%CH3.0	CH		
%ID3.0	DWORD	`COMPT_PIECES_CUR-MEAS`	Counter current value
%ID3.0.4	DWORD	`COMPT_PIECES_CAPT`	Counter captured value
%I3.0	EBOOL	`COMPT_PIECES_ENAB_ACTIV`	Counter enable active
%I3.0.1	EBOOL	`COMPT_PIECES_PRES_DONE`	Preset done

User memory

3

Presentation

Subject of this chapter

This chapter describes the memory structure of Micro and Premium PL7s.

What's in this Chapter?

This Chapter contains the following Maps:

Structure of Micro PL7 memory

General

Micro PL7 memory that can be accessed by the user is divided into two distinct sets:
- bit memory
- words memory

Bit memory

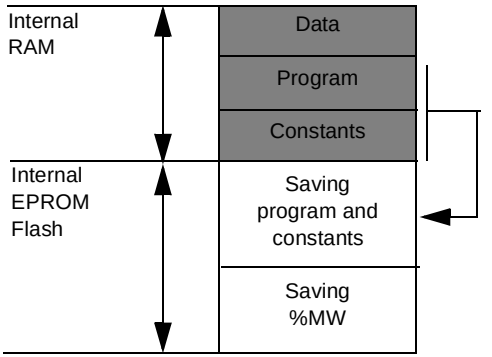
The bit memory is in the RAM memory that is integrated into the processor module. It contains the map of 1280 bit objects.

Role of the words memory

The words memory (16 bits) supports:
- **data**: dynamic application data and system data,
- **the program**: descriptors and executable code for tasks,
- **constants**: constant words, initial values and input/output configuration.

Structure without extension memory card

Data, program and constants are supported by the internal RAM memory in the processor module.
The following diagram describes the memory structure.

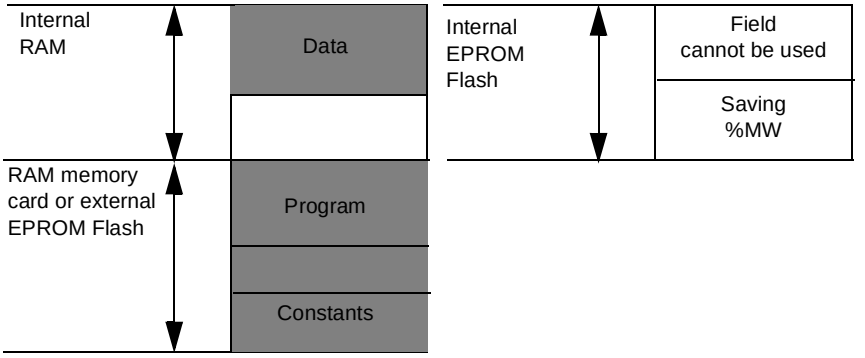


The EPROM FLASH 16 Kword memory integrated into the processor module can be used for saving:
- the application program (15Kwords)
- 1000 internal words %MWi

Structure with extension memory card

Data is supported by the processor module internal RAM memory.
Programs and constants are supported by the extension memory card.

The following diagram describes the memory structure.



The EPROM FLASH 16 K word memory integrated into the processor module can be used to save 1000 internal words %MWi.

Saving the memory

The RAM memory can be backed up by nickel cadmium batteries:
- supported by the processor module for the bit memory and internal RAM,
- inserted onto a card for the RAM memory card.

Transferring the application from the internal EPROM FLASH memory to the RAM memory is done automatically when the application is lost in RAM (if it has not been saved or if there is no battery).
Manual transfer can also be requested through a programming terminal.

Special characteristics of memory cards

The following table describes the different types of cards available.

Memory card	Description	TSX references	Application saving	Data storage
RAM	Contains the application programs and constants. They can be backed up by nickel cadmium batteries.	MRP 032P	32 K words	-
		MRP 064P	62 K words	-
RAM + storage	As well as the program and constants, these cards have a field for storing data which can be accessed by the PL7 instructions for reading/writing to files (See Reference Manual, Volume 2).	MRP 232P	32 K wordsP	128 K words
		MRP 264P	64 K words	128 K words
Eprom Flash	Contains the application programs and constants.	MFP 032P	32 K words	-
		MFP 064P	62 K words	-
Eprom Flash + storage	As well as the program and constants, these cards have a field for storing data which can be accessed by the PL7 instructions for reading/writing to files (See Reference Manual, Volume 2).	MFP 232P	32 K wordsP	128 K words
		MFP 264P	64 K words	128 K words
Back up	A back up EPROM FLASH card (not shown in the diagrams) can also be used for updating an application in the processor's internal RAM. This card contains the program section and the constants but not the data.	MFP BAK 032P	32 K words	-

Memory structure for Premium PL7s

General

Premium PL7 memory space comprises only one set.
The bit memory is integrated into the word memory (in the data field). It is limited to 4096 bits.

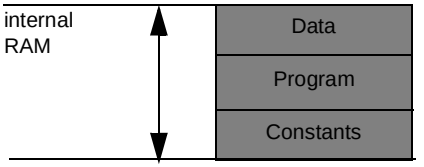
Role of the words memory

The words memory (16 bits) supports:

- **data**: dynamic application data and system data (the system reserves a RAM memory field of at least 5 K words)
- **the program**: descriptors and executable code for tasks,
- **constants**: constant words, initial values and input/output configuration.

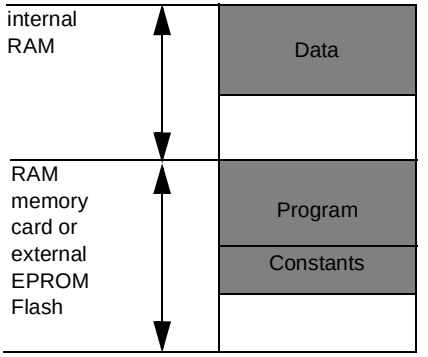
Structure without extension memory card

Program, data and constants are supported by the internal RAM memory in the processor module.
The following diagram describes the memory structure.



Structure with extension memory card

Data is supported by the processor module internal RAM memory.
Programs and constants are supported by the extension memory card.
The following diagram describes the memory structure.



Saving the memory

The bit memory and internal RAM are backed up by the nickel cadmium battery supported by the processor module.
The internal RAM memory card is backed up by a nickel cadmium battery.

Special characteristics of memory cards

The following table describes the different types of cards available.

Memory card	Description	TSX references	Application saving	Data storage	Symbol storage
RAM	Contains the application programs and constants. They can be backed up by nickel cadmium batteries. **Note:** TSX MRP 256P K word memory cards are paged cards. One 128 K word page receiving the executable code, the other 128 K word page receiving graphic information.	MRP 032P	32 K words	-	-
		MRP 064P	64 K words	-	-
		MRP 128P	128 K words	-	-
		MRP 256P	256 K words	-	-
RAM + file storage	As well as the program and constants, these cards have a field for storing data which can be accessed by the PL7 instructions for reading/writing to files (See Reference Manual, Volume 2).	MRP 232P	32 K words	128 K words	-
		MRP 264P	64 K words	128 K words	-
RAM + file + symbol storage	These cards contain an extra field for storing application symbols (and their comments).	MRP 2128P	128 K words	128 K words	128 K words
		MRP 3256P	256 K words	640 K words	128 K words
		MRP 3384P	384 K words	640 K words	-
		MRP 0512P	512 K words	-	256 K words
Eprom Flash	Contains the application programs and constants.	MFP 032P	32 K words	-	-
		MFP 064P	62 K words	-	-
		MFP 128P	128 K words	-	-
		MFP 256P	256 K words	-	-

Memory card	Description	TSX references	Application saving	Data storage	Symbol storage
Eprom Flash + file storage	As well as the program and constants, these cards have a field for storing data which can be accessed by the PL7 instructions for reading/writing to files (See Reference Manual, Volume 2).	MFP 032P	32 K words	128 K words	-
		MFP 064P	62 K words	128 K words	-
		MFP 128P	128 K words	128 K words	-
Back up	A back up EPROM FLASH card (not shown in the diagrams) can also be used for updating an application in the processor's internal RAM. This card contains the program section and the constants but not the data.	MFP BAK 032P	32 K words	-	-

Description of bits memory

General points

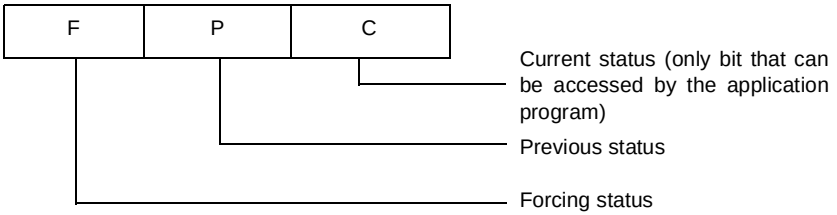
For Micro PL7s: this memory contains 1280 bit objects whatever the type of PL7.

For Premium PL7s: this bit memory does not exist and its contents are in the word memory in the application data field.

PL7 bit object coding is used to test the rising or falling edges on:
- input/output bits,
- internal bits.

Operation

Each bit object contained in the bit memory is stored using 3 bits allocated in the following way:



When updating the bit memory the system maintains:

Phase	Description
1	Transferring the map from the current status to the past status.
2	Re-updating the current status by the program, the system or the terminal (by forcing a bit).

Forcing

When forcing is requested by the terminal:
- Forcing status F is set to 1
- current status C is set to :
  - 1 if forcing to 1 is requested
  - 0 if forcing to 0 is requested

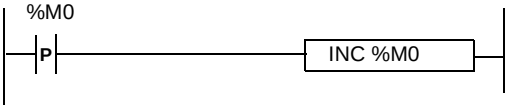
These states do not develop any more until:
- forcing is stopped and the bit involved updated,
- reverse forcing is requested, only the current status is modified.

Advice for using rising or falling edges

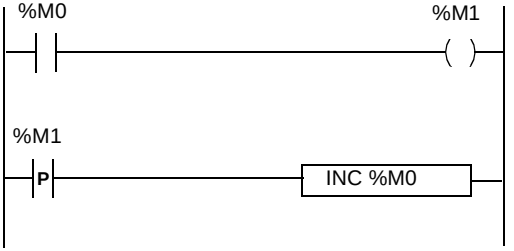
These rising or falling edge contact instructions only operate correctly if you follow the rules below:

- in each case, to process the same object:
  - input bit: the edge contact in the task or input module is exchanged,
  - output or internal bit: process reading and writing to it within the same task.
- Any bit object tested on an edge must be written only once using normal coils -( )- or negated coils -(/)- (and/or equivalent in Instruction List language). Do not use -(S)- or -(R)- coils. When an output is declared in an event processing exchange list, it triggers the exchange of the group of channels assigned to it; this disrupts the management of the edges in the task which normally manages this group of channels.
- do not SET or RESET an object whose edge you are testing because even if the result of the equation that sets SET/RESET is 0, the SET/RESET action is not carried out but the object history is updated (edge is lost).
- do not test the edge of inputs/outputs used in the event task, a master task or a fast task
- for internal bits: detecting an edge is not dependent on the task cycle. An edge on internal bit %Mi is detected when its status has changed between 2 readings. **This edge remains detected as long as this internal bit is not scanned in the action field.**

Example : And so in the example below, if you force bit %M0 to 1 in an animation table, the edge remains permanent.



So that the edge is only detected once, you must use an intermediate internal bit. In this case the %M1 history is updated, therefore the edge is only present once.



Description of word memory

General points

This 16 bit word memory is made up of 3 logic spaces:
- Data,
- Program,
- Constants.

the size of which is defined by configuration.

Note: Symbols and comments associated with the objects are not recorded in the PL7 memory but stored in the local application (terminal hard disk).

Application data memory

The data memory comprises the following different fields:

Word type	Description
System	Fixed number
Function blocks	Corresponds to the input/output words of these blocks (current values, adjustment...). The number of each type of function block is set in configuration
Internal	Size defined by the number declared in configuration.
Inputs/outputs	Corresponds to the words associated with each module. Their number depends on the modules configured.
Network commons	4 common words per PL7 station (only available if the communications module is present and configured in the common words exchange).

Application program memory

This field contains the executable program code, graphic information (contact network) and program comments.

Application constant memory

This field contains the function block and input/output module parameters defined in configuration and constant words %KW.

Characteristics of TSX 37 PL7 memory

Size of bit memory

The following table describes the bit object memory division.

TSX Processor		37 05/08/10	37 21/22
Size available on processor		1280	1280
Type of objects	system bits %Si	128	128
	input/output bits %I/O (?)x.i	(1)	(1)
	internal bits %Mi	256	256
	step bits %Xi	96	128
Key			
(1)	depends on the hardware configuration declared (input/output modules, devices on AS-i bus)		

Size of the words memory

The following table describes the word object memory division.

TSX Processor	3705/08	3710	3721			37 22		
Cartridge	-	-	-	32 K words	64 K words	-	32 K words	64 K words
internal RAM	9 K words	14 K words	20 K words	52 Kwords	84 Kwords	20 K words	52 Kwords	84 Kwords
Data (%MWi)	0.5 Kwords (1)	0.5 Kwords (1)	0.5 Kwords (1)	17.5 Kwords	17.5 Kwords	0.5 Kwords (1)	17.5 Kwords	17.5 Kwords
100% Boolean program								
• LD language	1.6 Ki	3.8 Ki	6.6 Ki	13.7 Ki	28.5 Ki	6.3 Ki	13.6 Ki	28.4 Ki
• IL language	2 Ki	4.9 Ki	8.4 Ki	17.5 Ki	36.3 Ki	8.1 Ki	17.3 Ki	36.1 Ki
• ST language	1.3 Ki	3.3 Ki	5.6 Ki	11.7 Ki	24.2 Ki	5.4 Ki	11.5 Ki	24.1 Ki
90% Boolean program								
• LD language	1.1 Ki	3.1 Ki	5.4 Ki	11.8 Ki	24.7 Ki	5.2 Ki	11.6 Ki	24.5 Ki
• IL language	1.4 Ki	3.8 Ki	6.6 Ki	14.3 Ki	30.0 Ki	6.3 Ki	14.2 Ki	29.8 Ki
• ST language	1.1 Ki	2.9 Ki	5.1 Ki	11.1 Ki	23.3 Ki	4.9 Ki	11.0 Ki	23.2 Ki
65% Boolean program								
• LD language	0.9 Ki	2.2 Ki	4.0 Ki	9.1 Ki	18.9 Ki	3.9 Ki	8.9 Ki	18.8 Ki
• IL language	1.0 Ki	2.5 Ki	4.6 Ki	10.3 Ki	21.3 Ki	4.4 Ki	10.1 Ki	21.2 Ki
• ST language	1.0 Ki	2.5 Ki	4.6 Ki	10.3 Ki	21.3 Ki	4.4 Ki	10.1 Ki	21.2 Ki

TSX Processor	**3705/08**	**3710**	**3721**			**37 22**		
Constants (1)	128 words	128 words	128 words	256 words	512 words	128 words	256 words	512 words
Key								
(1)	Default size, can be extended at the expense of the application program size.							
Ki	K instructions (1024 instructions)							

Note: The PL7 AP/memory usage command is used to find out about the application memory division in the PL7 memory.

Characteristics of TSX/PCX 57 10/15/20/25 PL7 memories

Size of bit memory

This table describes the memory division of word objects for TSX 57-103, TSX 57-153, TSX 57-203, PCX 57-203 and TSX 57-253 PL7s.

Processor		**TSX 57 103/153 et PCX 57 203**	**TSX57 203/253**
Type of objects	system bits %Si	128	128
	input/output bits %I/Qx.i	(1)	(1)
	internal bits %Mi (max. no.)	3962	8056
	step bits %Xi (max. no.)	1024	1024
Key			
(1)	depends on the hardware configuration declared (input/output modules, devices on AS-I bus)		

Size of the words memory

The table describes the memory division of word objects for TSX 57-103, TSX 57-153, TSX 57-203, PCX 57-203 and TSX 57-253 PL7s.

Processor	**TSX 57-103 - TSX 57 153**			**TSX-PCX 57 203**	**TSX- 57 253**	**TSX-PCX 57 203/ TSX 57 253**	**TSX-PCX 57 203/ TSX 57 253**	**TSX-PCX 57 203/ TSX 57 253**
Cartridge	-	32K	64K	-	-	32K	64K	128K
internal RAM	32K	32K	32K	48K/64K	48K/64K	48K/64K	48K/64K	48K/64K
Data (%MWi)	0,5 K (1)	26 K	26 K	1K (1)	1K (1)	30,5K	30,5K	30,5K
100% Boolean program								
• Langage LD	8,8 Ki	12,3 Ki	26,9 Ki	15,5 Ki	22,8 Ki	12,3 Ki	26,6 Ki	565,2 Ki
• IL language	11,2 Ki	15,6 Ki	34,3 Ki	19,7 Ki	29,1 Ki	15,6 Ki	33,9 Ki	71,6 Ki
• ST language	7,6 Ki	10,5 Ki	22,9 Ki	13,1 Ki	19,4 Ki	10,4 Ki	22,6 Ki	47,8 Ki
90% Boolean program								
• LD language	5,2 Ki	8,6 Ki	21,4 Ki	11,0 Ki	17,4 Ki	8,6 Ki	21,1 Ki	46,9 Ki
• IL language	6,2 Ki	10,3 Ki	25,6 Ki	13,1 Ki	20,7 Ki	10,3 Ki	25,2 Ki	56,0 Ki
• ST language	5,0 Ki	8,3Ki	20,5 Ki	10,5 Ki	16,6 Ki	8,3 Ki	20,2 Ki	44,9 Ki
65% Boolean program								
• LD language	3,6 Ki	6,7 Ki	16,7 Ki	8,1 Ki	13,1 Ki	6,6 Ki	16,4 Ki	36,6 Ki
• IL language	3,7 Ki	6,8 Ki	17,0 Ki	8,3 Ki	13,4 Ki	6,8 Ki	16,8 Ki	37,5 Ki
• ST language	4,2 Ki	7,9 Ki	19,7 Ki	9,6 Ki	15,5 Ki	7,8 Ki	19,4 Ki	43,3 Ki
Constants (1)	128 words	256 words	512 words	256 words	256 words	256 words	512 words	512 words

Processor	**TSX 57-103 - TSX 57 153**	**TSX-PCX 57 203**	**TSX- 57 253**	**TSX-PCX 57 203/ TSX 57 253**	**TSX-PCX 57 203/ TSX 57 253**	**TSX-PCX 57 203/ TSX 57 253**
Key						
(1)	Default size, can be extended at the expense of the application program size.					
Ki	Kinstructions					
K	Kwords					

Note:
- when this table mentions as a characteristic 2 values separated by a "/", they are associated with each type of processor respectively (separated by a "/" in the table heading).
- The PL7 AP/memory usage command is used to find out about the application memory division in the PL7 memory.

Characteristics of TSX/PCX 57 30/35 PL7 memories

Size of bit memory

This table describes the memory division of word objects in TSX 57-303, TSX 57-353, and PCX 57-353 PL7s.

TSX/PCX processor		**57 303/353**
Type of objects	system bits %Si	128
	input/output bits %I/Qx.i	(1)
	internal bits %Mi (max. no.)	16250
	step bits %Xi (max. no.)	1024
Key		
(1)	depends on the hardware configuration declared (input/output modules, devices on AS-I bus)	

Size of the words memory

This table describes the memory division of word objects in TSX 57-303, TSX 57-353, and PCX 57-353 PL7s.

Processor	**TSX 57 303**	**TSX/ PCX57 353**	**TSX 57 303 / TSX/PCX57 353**	**TSX 57 303 / TSX/PCX57 353**	**TSX 57 303 / TSX/PCX57 353**	**TSX 57 303 / TSX/PCX57 353**	**TSX 57 303 / TSX/PCX57 353**
Cartridge	-	-	32K	64K	128K	256K	384K
internal RAM	64K/80K	64K/80K	80K/96K	80K/96K	80K/96K	80K/96K	80K/96K
Data (%MWi)	1K (1)	1K (1)	30,5K	30,5K	30,5K	30,5K	30,5K
100% Boolean program							
• LD language	28,8 Ki	30,1 Ki	12,3 Ki	26,6 Ki	56,2 Ki	115,3 Ki	150,5 Ki
• IL language	36,7 Ki	38,4 Ki	15,6 Ki	33,9 Ki	71,6 Ki	147,1 Ki	150,5 Ki
• ST language	24,5 Ki	25,6 Ki	10,4 Ki	22,6 Ki	47,8 Ki	98,0 Ki	148,3 Ki
90% Boolean program							
• LD language	22,6 Ki	23,8 Ki	8,6 Ki	21,1 Ki	46,9 Ki	98,4 Ki	149,9 Ki
• IL language	27,1 Ki	28,4 Ki	10,3 Ki	25,2 Ki	56,0 Ki	117,5 Ki	157,6 Ki
• ST language	21,7 Ki	22,7 Ki	8,3 Ki	20,2 Ki	44,9 Ki	94,2 Ki	142,9 Ki
65% Boolean program							
• LD language	17,4 Ki	18,2 Ki	6,6 Ki	16,4 Ki	36,6 Ki	77,0 Ki	117,4 Ki
• IL language	17,8 Ki	18,6 Ki	6,8 Ki	16,8 Ki	37,5 Ki	78,8 Ki	120,1 Ki
• ST language	20,5 Ki	21,5 Ki	7,8 Ki	19,4 Ki	43,3 Ki	91,1 Ki	138,8 Ki
Constants (1)	256 words	256 words	256 words	1024 words	1024 words	1024 words	1024 words
Key							

Processor	**TSX 57 303**	**TSX/ PCX57 353**	**TSX 57 303 / TSX/PCX57 353**	**TSX 57 303 / TSX/PCX57 353**	**TSX 57 303 / TSX/PCX57 353**	**TSX 57 303 / TSX/PCX57 353**	**TSX 57 303 / TSX/PCX57 353**
(1)	Default size, can be extended at the expense of the application program size.						
Ki	Kinstructions						
K	Kmots						

Note:

- when this table mentions as a characteristic 2 values separated by a "/", they are associated with each type of processor respectively (separated by a "/" in the table heading).
- The PL7 AP/memory usage command is used to find out about the application memory division in the PL7 memory.

Characteristics of TSX 57 453 PL7 memory

Size of bit memory

This table describes the memory division of word objects in TSX 57-453 PL7s.

Processor TSX		**57 453**
Type d'objets	system bits %Si	128
	input/output bits %I/Qx.i	(1)
	internal bits %Mi (max. no.)	32634
	step bits %Xi (max. no.)	1024
Key		
(1)	depends on the hardware configuration declared (input/output modules, devices on AS-I bus)	

Size of the words memory

The following table describes the memory division of word objects in TSX 57-453 PL7s.

Processor	**TSX 57 453**						
Cartridge	-	32K	64K	128K	256K	384	512K
internal RAM	96K	176K	176K	176K	176K	176K	176K
Data (%MWi)	1K (1)	30,5K	30,5K	30,5K	30,5K	30,5K	30,5K
100% Boolean program							
• LD language	37,5 Ki	12,3 Ki	26,6 Ki	56,2 Ki	115,3 Ki	150,5Ki	150,5 Ki
• IL language	47,8 Ki	15,6 Ki	33,9 Ki	71,6 Ki	147,1 Ki	150,5 Ki	150,5 Ki
• ST language	31,9 Ki	10,4 Ki	22,6 Ki	47,8 Ki	98,0 Ki	148,3 Ki	150,7 Ki
90% Boolean program							
• LD language	30,2 Ki	8,6 Ki	21,1 Ki	46,9 Ki	98,4 Ki	149,9 Ki	157,6 Ki
• IL language	36,0 Ki	10,3 Ki	25,2 Ki	56,0 Ki	117,5 Ki	157,6 Ki	157,6 Ki
• ST language	28,9 Ki	8,3 Ki	20,2 Ki	44,9 Ki	94,2 Ki	142,9 Ki	157,8 Ki
65% Boolean program							
• LD language	23,2 Ki	6,6 Ki	16,4 Ki	36,6 Ki	77,0 Ki	117,4 Ki	157,8 Ki
• IL language	23,7 Ki	6,8 Ki	16,8 Ki	37,5 Ki	78,8 Ki	120,1 Ki	161,3 Ki
• ST language	27,4 Ki	7,8 Ki	19,4 Ki	43,3 Ki	91,1 Ki	138,8 Ki	171,3 Ki
Constants (1)	256 words	256 words	1024 words	1024 words	1024 words	1024 words	1024 words
Key							
(1)	Default size, can be extended at the expense of the application program size.						
Ki	Kinstructions						
K	Kwords						

Note:

- when this table mentions as a characteristic 2 values separated by a "/", they are associated with each type of processor respectively (separated by a "/" in the table heading).
- The PL7 AP/memory usage command is used to find out about the application memory division in the PL7 memory.

Operating modes

4

Presentation

Subject of this chapter

This chapter deals with the behavior of the user program on a warm restart and cold start.

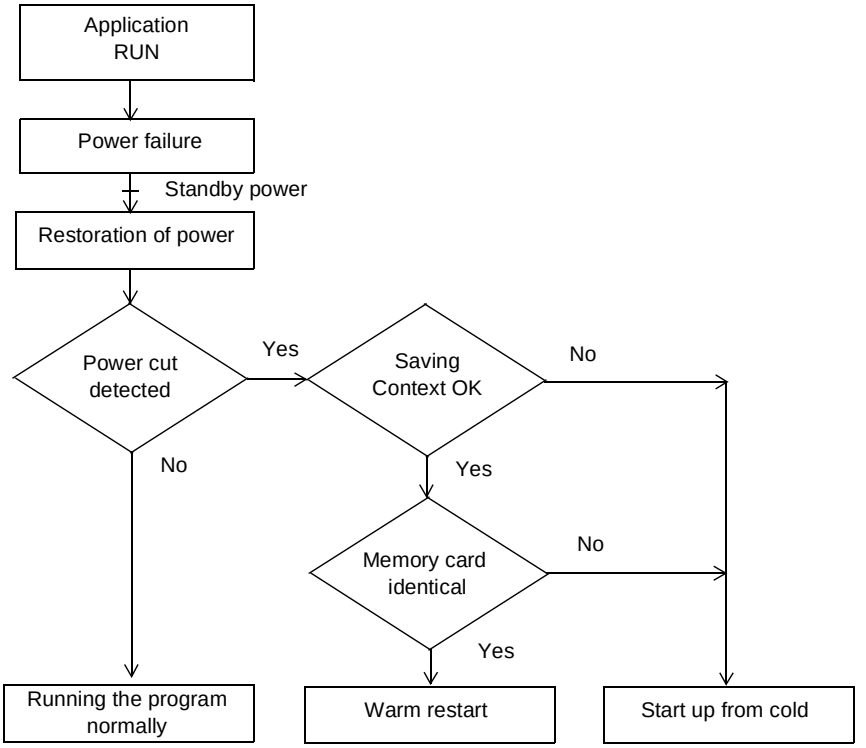
What's in this Chapter?

This Chapter contains the following Maps:

Dealing with power cuts and power restoration

Illustration

The illustration shows the various power restarts detected by the system. If the duration of the cut is less than the power supply filtering time (about 10 ms for an alternating current supply or 1 ms for a direct current supply), this is not noticed by the program which runs normally.



Operation

The table below describes the processing phases for power cuts.

Phase	Description
1	In the event of a power cut the system stores the application context and the time of the cut.
2	It sets all outputs to fallback status (status defined in configuration).
3	When power is restored, the context saved is compared with the one in progress which defines the type of start to run: ● if the application context has changed (loss of system context or new application), the PL7 initializes the application: start up from cold, ● if the application context is the same, the PL7 restarts without initializing data: warm restart.

Supply failure on a rack other than rack 0

All the channels on this rack are seen as in error by the processor but the other racks are not affected. The values of the error inputs are no longer updated in the application memory and are set to 0 in the case of a discrete input module unless they have been forced in which case they are kept at the forcing value.

If the cut lasts less than 10 ms for alternating current supplies or up to 1 ms for a direct current supply, these are not noticed by the program which runs normally.

Dealing with a warm restart

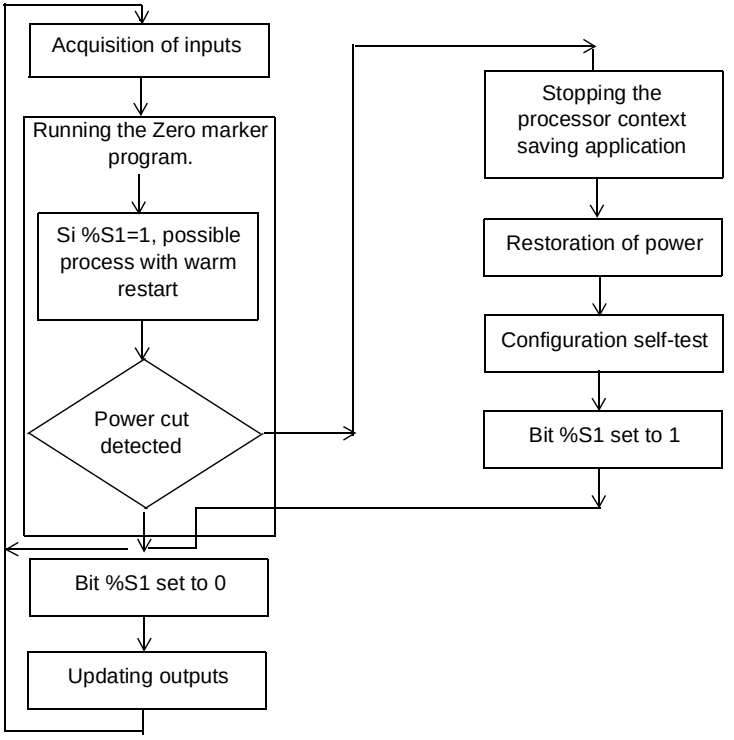
Cause of a warm restart

A warm restart can occur:

- when power is restored without loss of context,
- when the system bit %S1 is set to 1 by the program,
- from the PL7 by the terminal,
- by pressing the RESET button on the supply module on rack 0 (except on a station with a PCX 57 processor).

Illustration

The drawing below describes a warm restart operation.



Operation

The table below describes the restart phases for running a program after a warm restart.

Phase	Description
1	The program starts up again from the element where the power cut took place, without updating the outputs.
2	At the end of the restart cycle the system: ● initializes message and event files ● sends configuration parameters to all discrete and application specific input/output modules, ● deactivates the fast task and event processes (until the end of the first master task cycle).
3	The system carries out a restart cycle in which it: ● takes into account again all the input modules, ● relaunches the master task with the bits %S1 (warm restart) and %S13 (first cycle in RUN) set to 1, ● resets bits %S1 and %S13 to 0 at the end of this first master task cycle, ● reactivates the fast task and event processes at the end of this first master task cycle.

Warm restart processes per program

In the event of a warm restart, if you require a particular application process, you have to write the corresponding program by testing %S1 at 1 at the beginning of the master task program.

Developing outputs

As soon as a power failure is detected the outputs are set to fallback position:
- either they take the fallback value,
- or they keep the current value,

depending on the choice made at configuration.

When power is restored, outputs are at zero until they are updated again by the task.

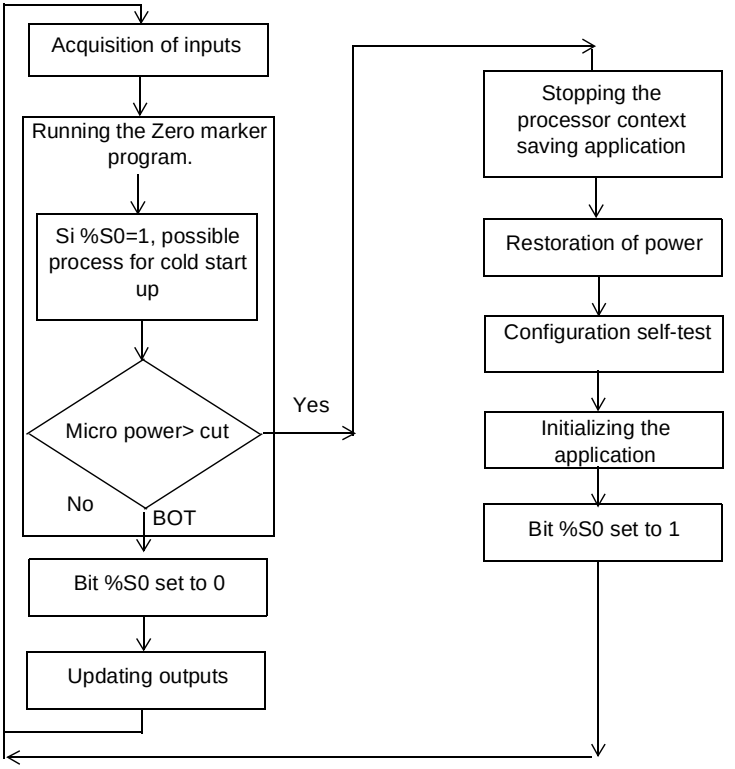
Dealing with a cold start

Cause of a cold start

The following table describes the different possible causes for a cold start.

Causes	Characteristics of the start
Loading an application	Cold start forced to STOP
Pressing the processor RESET button	Cold start forced to STOP or RUN according to the definition at configuration
Pressing the processor RESET button after a blocking fault	Cold start forced to STOP
Manipulating the prehensile or inserting/extracting a PCMCIA memory card	Cold start forced to STOP or RUN according to the definition at configuration
Initializing from a Junior or Pro PL7 Forcing the system bit %S0	Cold start forced to STOP or RUN according to the definition at configuration, without initializing the discrete and application specific input/output modules.
Restart after a power failure with loss of context	Cold start forced to STOP or RUN according to the definition at configuration

Illustration The drawing below describes a cold restart operation.



Operation

The table below describes the restart phases for running a program after a cold restart.

Phase	Description
1	Start up is in RUN or in STOP depending on the parameter `Automatic start up in RUN` defined at configuration or if this is used according to the status of the RUN/STOP input. The program run restarts at the beginning of the cycle.
2	The system: ● resets bits, the I/O map and internal words to zero (if the %MW reset to zero option on restart from cold is checked in the processor Configuration screen). If the %MW reset is not active and if internal words %MWi are saved in the internal EPROM Flash memory (TSX 37), these are restored in the event of a cold start. ● initializes system bits and words. ● initializes function blocks from configuration data. ● deactivates tasks, other than the master task, up till the end of the first master task cycle. ● sets Grafcet to initial steps. ● cancels forcings. ● initializes data declared in the DFBs: either to 0 or to the initial value declared in the code, i.e. with the saved value from the SAVE function ● initializes message and event files ● sends configuration parameters to all discrete and application specific input/output modules,
3	For this first restart cycle the system: ● relaunches the master task with bits %S0 (warm restart) and %S13 (first cycle in RUN) set to 1, the word %SW10 (detecting cold restart on the first revolution of a task) is set to 0, ● resets bits %S0 and %S13 to 0 and resets to 1 each word bit %SW10 to the end of this first master task cycle, ● activates the fast task and event processes at the end of this first master task cycle.

Dealing with cold start for each program

To carry out an application process after the PL7 has started from cold, it is possible to test the bit %SW10:X0 per program (if %SW10:X0=0, there is a cold restart).

Developing outputs

As soon as a power failure is detected the outputs are set to fallback position:

- either they take the fallback value,
- or they keep the current value,

depending on the choice made at configuration.

When power is restored, outputs are at zero until they are updated again by the task.

Software structure

5

Presentation

Subject of this chapter

This chapter describes the tasks and how they run in the PL7.

What's in this Chapter?

This Chapter contains the following Sections:

5.1 Description of tasks

Presentation

Introduction to this section

This section describes the role and content of each of the tasks that can make up a PL7 program.

What's in this Section?

This Section contains the following Maps:

Presenting the master task

General points

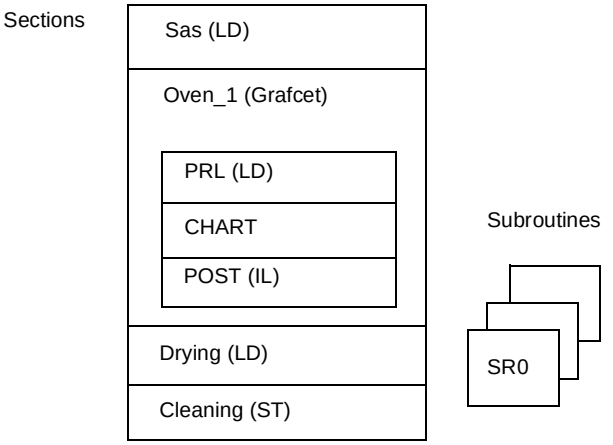
The master task represents the main program. It is compulsory whatever the mono-task or multitask structure adopted.

The master task program (MAST) is made up of several program modules called sections (See *Description of sections and subroutines, p. 86*), and subroutines.

How the master task is run can be chosen (in configuration). It can be cyclical (See *Cyclic run, p. 95*) or periodic (See *Periodic run, p. 97*).

Illustration

The following illustration shows an example of a master task made up of 4 sections and 3 subroutines.



Description of sections and subroutines

Presenting the sections

The sections are autonomous programming entities. Instruction line and contact network location labels … belong to the section (no program jump possible to another section).

They are programmed either in:
- ladder language,
- instruction list language,
- structured text language,
- Grafcet.

Sections are run in programming order tin the browser window (structure view).

Sections are linked to a task. One section cannot belong to several tasks at the same time.

Presenting subroutines

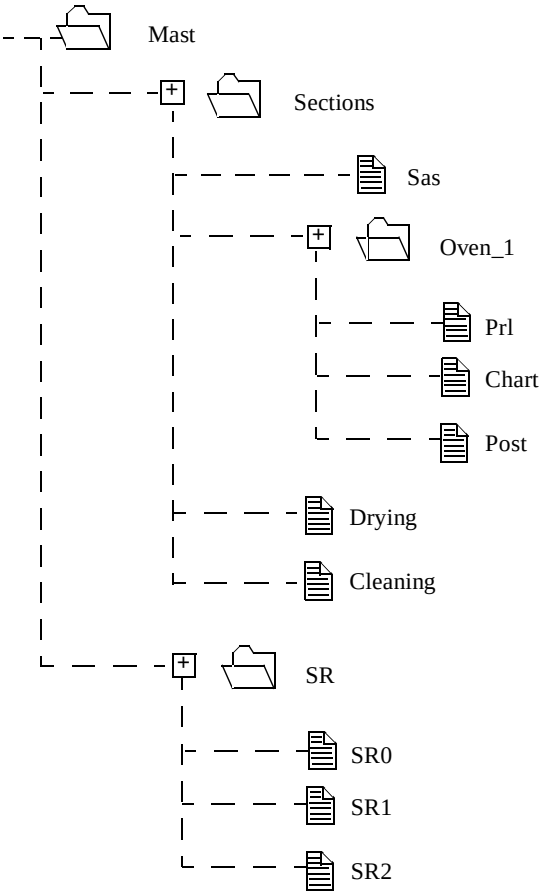
Subroutine modules are also programmed as separate entities either in:
- ladder language,
- instruction list language,
- structured text language,

Calling up subroutines is done in the sections or from another subroutine (a maximum of 8 overlay levels).

Subroutines are also linked to a task. One subroutine cannot be called up by several tasks.

Example

The following drawing gives an example of the structure of a task in sections and subroutines.



Characteristics of a section

The following table describes the characteristics of a section.

Characteristic	Description
Name	24 characters maximum
Language	Ladder, instruction list , structured text or Grafcet language
Task	Master or fast
Condition (optional)	Objects allowed as a condition: • %M,%S,%X • indexed bits, bits extracted from words • %I , %Q All these objects can be forced from the terminal except for %S bits, indexed bits, extracted bits, %Ixy.i.ERR,and %I xy.MOD.ERR. The condition must be status 1 for the section to be run.
Comments	250 characters maximum.
Protection	Write protection, read/write protection. Protection can be global or partial.

Note: on a cold start run conditions are at 0. All sections associated with a condition are disabled.

Grafcet section

The following table describes program elements for a Grafcet section.

Processing	Name	Characteristics
Preliminary	PRL	Programmed in ladder language LD, instruction list language IL or structured text language ST. It is run before Grafcet.
Grafcet	CHART	Transition conditions associated with transitions and actions associated with steps or macro step steps are programmed in the Grafcet pages.
Subsequent	POST	Programmed in ladder language LD, instruction list language IL or structured text language ST. It is run after Grafcet.

Characteristics of a subroutine

The following table describes the characteristics of an Sri subroutine.

Characteristic	Description
Number	0 à 253
Language	Ladder, instruction list , structured text
Task	Master or fast
Comments	250 characters maximum.

Presenting the fast task

General points

This task which has a higher priority than the master task MAST is periodic so that tasks that have a lower priority have time to run.

Also, processes associated with it must therefore be short in order not to hinder the master task. As with the master task, the associated program is made up of sections and subroutines.

Fast task period

The fast task period FAST is set at configuration from 1 - 255 ms. This may be defined as greater than the master task MAST so that it can be adapted for slow but priority periodic processes.

The program run, however, must remain short to avoid exceeding tasks with lower priority.

The fast task is checked by a watch dog which is used to detect an abnormal period in the application program. In the case of overflow, system bit %S11 is set at 1 and the application is declared as having a PL7 blocking fault.

Fast task check

The system word %SW1 contains the period value. It is initialized when starting from cold by the value set in the configuration. It can be modified by the user by the program or the terminal.

System bits and words, are used to check the running of this task:

- %S19: indicates that the period has been exceeded. It is set to 1 by the system when the cycle time is greater that the task period.
- %S31: is used to confirm or disable the fast task. It is set to 0 by the system when the application is on cold start, at the end of the first master task cycle. It is set to 1 or 0 to confirm or disable the fast task.

Displaying fast task running time

The following system words are used for information on the cycle time:

- %SW33 contains the running time for the last cycle.
- %SW34 contains the running time for the longest cycle,
- %SW35 contains the running time for the shortest cycle.

Presenting event processing

General points

Event processes are used to reduce the software reaction time for command events coming from certain application specific modules.

These processes take priority over any other task. They are therefore suitable for processes which require very short reaction times in relation to the arrival of the event.

The number of event processes that can be programmed depends on the type of processor.

PL7 type	Number of processes	Name
Micro TSX 37-05/08/10	8	EVT1 - EVT8
Micro TSX 37-21/22	16	EVT0 - EVT15
Premium TSX/PCX 57-1•	32	EVT0 - EVT31
Premium TSX/PCX 57-2•/3•/4•	64	EVT0 - EVT63

Operation

The appearance of an event diverts the application program to the process that is associated with the input/output channel which has caused the event.

Inputs (%I, %IW, %ID) associated with the I/O channel which triggered the event are updated by the system before calling up the event process.

Association between a channel and an event number is made in the channel configuration screen.

Command events

These are external events linked to application specific functions.

On Micro PL7s event processes can be triggered by:
- inputs 0 - 3 of position 1 module, on rising or falling edge,
- the counting module counting channel(s),
- module 1 counting channels (if this is configured in the counter),
- receiving telegrams in a TSX 37-21/22 equipped with a TSX FPP20 module.

On Premium PL7s event processes can be triggered by:
- inputs from modules DEY 16 FK, DMY 28 FK, DMY 28 RFK
- counting module channels,
- channels for axis command modules TSX CAY •,
- channels for step by step command modules TSX CFY •,
- "FPP20" communication channels.
- ...

Managing event processes

Event processes can be confirmed or disabled globally by the application program using system bit %S38.
If one or more events occur while they are disabled, the associated processes are lost.

Two PL7 language instructions, MASKEVT( ) and UNMASKEVT( ), used in the application program are also used to mask or unmask event processes.
If one or more events occur while they are masked, they are stored by the system and the associated processes will only be carried out when they have been unmasked.

Process priority

Micro TSX 37-05/08/10 PL7s
The 8 possible command events all have the same priority level, therefore, one event process cannot be interrupted by another.

Micro TSX 37-21/22 or Premium PL7s
There are 2 priority levels for command events: event 0 (EVT0) has a higher priority than the other events

5.2 Mono task structure

Presentation

Introduction to this section

This section describes how a mono task application runs.

What's in this Section?

This Section contains the following Maps:

Mono task software structure

Description

The mono task application program is associated with a single user task, the master task MAST (see *Presenting the master task, p. 85*).

The program associated with the master task (MAST) is made up of several sections and subroutines.

Running the master task can be selected (at configuration) as
- cyclic (See *Cyclic run, p. 95*)
- or periodic (See *Periodic run, p. 97*)

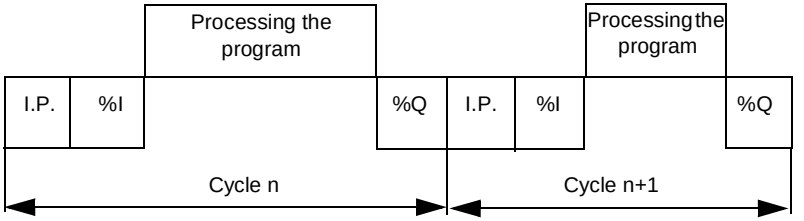
Cyclic run

Description

This type of operation corresponds to a normal PL7 cycle run (default selection).
It consists of linking master task cycles (MAST) one after the other.
After updating the outputs, the system carries out the appropriate processes then links another task cycle.

Operation

The following drawing shows the running phases of the PL7 cycle.



Description of the various phases

The table below describes the operating phases.

Address	Phase	Description
I.P.	Internal processing	The system implicitly monitors the PL7 (managing system bits and words, updating current timer values, updating status lights, detecting RUN/STOP switches,…) and processes requests from the terminal (modifications and animation). In the case of the Premium PL7 internal processing is done in parallel with input and output processes.
%I	Acquisition of inputs	Writing to the memory the status of information on discrete and application specific module inputs associated with the task,
-	Program processing	Running the application program written by the user,
%Q	Updating outputs	Writing output bits or words associated with discrete and application specific modules associated with the task according to the status defined by the application program.

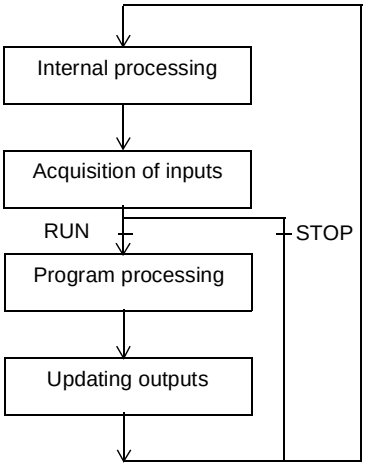
Operating mode

PL7 in RUN, the processor carries out the internal processing order, acquiring inputs, processing the application program and updating outputs.
PL7 in STOP, the processor carries out:

- internal processing,
- acquisition of inputs,
- and depending on the selected configuration:
  - fallback mode: outputs are set to "fallback",
  - maintenance mode: outputs are maintained at their last value.

Illustration

The following illustration shows the operating cycles.



Check cycle

The check cycle is carried out by watch dog (See *Checking cycle time, p. 100*).

Periodic run

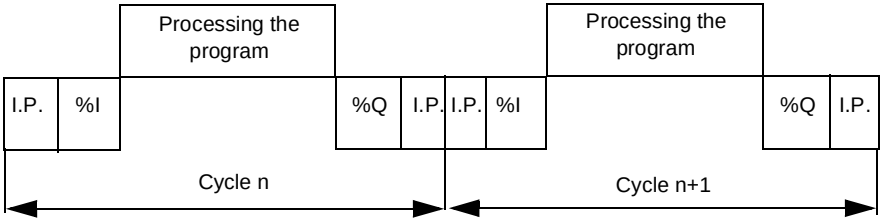
Description

In this operating mode, acquiring inputs, processing the application program and updating outputs are done periodically according to the time defined at configuration (from 1 - 255 ms).

At the beginning of the PL7 cycle, a timer, the value of which is initialized at the period defined at configuration, starts to count down.
The PL7 cycle must end before the timer has finished and relaunches a new cycle.

Operation

The following drawing shows the running phases of the PL7 cycle.



Description of the various phases

The table below describes the operating phases.

Address	Phase	Description
I.P.	Internal processing	The system implicitly monitors the PL7 (managing system bits and words, updating current timer values, updating status lights, detecting RUN/STOP switches, ...) and processes requests from the terminal (modifications and animation) In the case of the Premium PL7 internal processing is done in parallel with input and output processes.
%I	Acquisition of input	Writing to the memory the status of information on discrete and application specific module inputs associated with the task,
-	Program processing	Running the application program written by the user,
%Q	Updating outputs	Writing output bits or words associated with discrete and application specific modules associated with the task according to the status defined by the application program.

Operating mode

PL7 in RUN, the processor carries out the internal processing order, acquiring inputs, processing the application program and updating outputs.

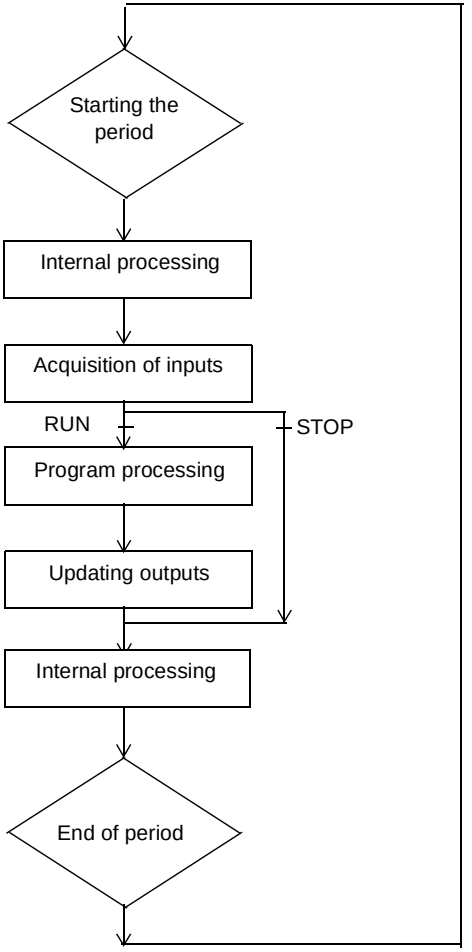
- If the period has not yet finished, the processor completes it operating cycle until the end of the internal processing period.
- If the operating time is longer than that allocated to the period, the PL7 indicates that the period has been exceeded by setting the task system bit %S19 to 1. The process continues and is run completely (however, it must not exceed the watch dog time limit). The following cycle is linked in after writing the outputs of the cycle in progress implicitly.

PL7 in STOP, the processor carries out:

- internal processing,
- acquisition of inputs,
- and depending on the selected configuration:
  - fallback mode: outputs are set to "fallback",
  - maintenance mode: outputs are maintained at their last value.

Illustration

The following illustration shows the operating cycles.



Check cycle

Two checks are carried out :

- period overflow (See *Checking cycle time, p. 100*),
- par watch dog (See *Checking cycle time, p. 100*),

Checking cycle time

General points

The duration of the master task operation when running cyclically or periodically, is controlled by the PL7 (watch dog) and must not exceed the value set out in the T max configuration (250ms default, 500ms maximum).

Software watch dog (periodic or cyclic operation)

If this is exceeded, an error is declared in the application which causes the PL7 to stop immediately:

- **on the Micro** setting the %Q2.0 alarm output to 0 if it has been configured,
- **on the Premium**, setting the power supply alarm relay to 0

The bit %S11 is used to check the running of this task. It indicates that the watch dog has been exceeded. It is set to 1 by the system when the cycle time is greater than the watch dog.

Note: On the Premium the watch dog value must be greater than the period.

Check on periodic operation

In periodic operation an additional check is used to detect the period being exceeded:

- %S19: indicates that the period has been exceeded. It is set to 1 by the system when the cycle time is greater that the task period.
- %SW0 : this word contains the period value (in ms). It is initialized when starting from cold by the value set in the configuration. It can be modified by the user.

Using master task running time

The following system words are used for information on the cycle time:

- %SW30 contains the running time for the last cycle.
- %SW31 contains the running time for the longest cycle,
- %SW32 contains the running time for the shortest cycle.

Note: This different information can also be accessed from the configuration editor explicitly.

5.3 Multi task structure

Presentation

Introduction to this section

This section describes how a multi task application runs.

What's in this Section?

This Section contains the following Maps:

Multitask software structure

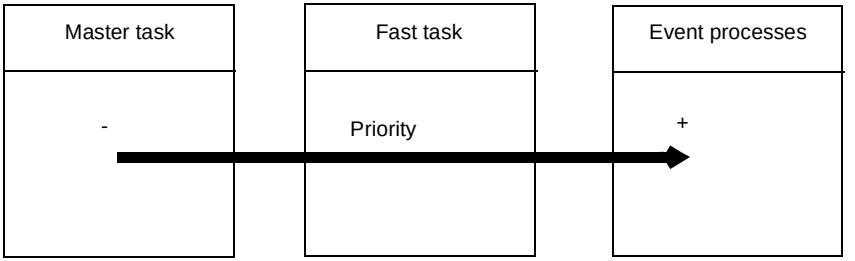
Description

Task structure for such an application is as follows:

Task	Name	Description
Master	MAST	Always present which can be cyclic or periodic.
Fast	FAST	Optional which is always periodic.
Event	EVTi	Called up by the system when an event appears on an input/output data module. These processes are optional and are used for applications which need short response times to act on inputs/outputs.

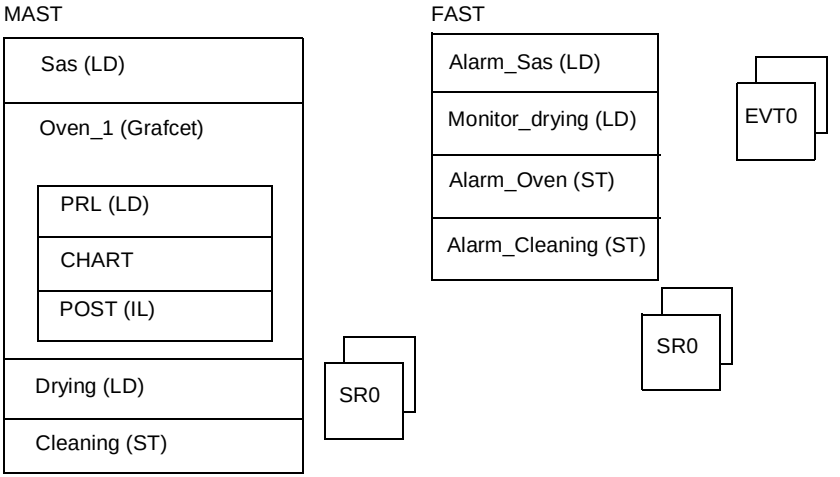
Illustration

The following drawing shows multitask structure tasks and their priority level.



Example

The following example shows a multitask structure made up of a master task MAST, a fast task FAST and 2 event processes EVTO and EVTI.



Sequencing tasks in a multitask structure

General points

The master task default is active.
The fast task default is active if it is programmed.
The event process is activated when the event associated with it occurs.

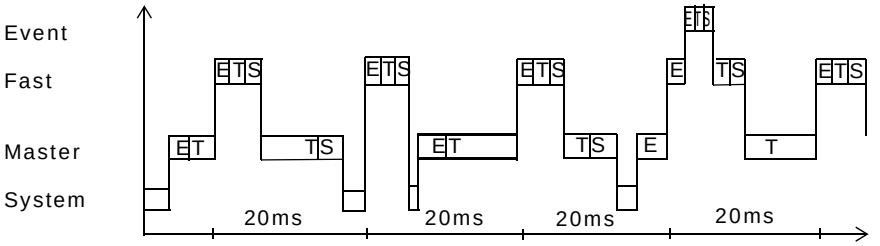
Operation

The following table describes running priority tasks.

Phase	Description
1	Arrival of an event or beginning of the fast task cycle.
2	Stopping the running of tasks in progress that have a lower priority,
3	Running the priority task.
4	The interrupted task takes over again when the priority task processes have finished.

Description of sequencing tasks

The following drawing illustrates task sequencing for a multitask process including a cyclical master task, a fast 20ms task and an event process.



Key:
I: Acquisition of inputs
P: program processing
O: updating outputs

Checking tasks

Running fast and event tasks can be checked by the program by using system bits:

- %S30 is used to activate or not the master task MAST.
- %S31 is used to activate or not the fast task FAST.
- %S38 is used to activate or not event tasks EVTi.

Assigning input/output channels to master and fast tasks

General points

As well as the application program master MAST and fast FAST tasks run system functions linked to managing implicit inputs/outputs associated with them.

Associating a channel or group of channels to a task is defined in the configuration screen of the corresponding data module, the associated default task being the MAST task.

Discrete modules

As the modularity of discrete modules is 8 successive channels (channels 0 - 7, channels 8 - 15,…), inputs/outputs can be assigned by groups of 8 channels either to the MAST task or the FAST task.

Example: it is possible to assign channels of a 28 input/output module in the following way:

- inputs 0 - 7 assigned to the MAST task,
- inputs 8 -15 assigned to the FAST task,
- outputs 0 - 7 assigned to the MAST task,
- outputs 8 - 15 assigned to the FAST task.

Counting modules

Each counting module channel can be assigned either to the MAST task or the FAST task.

Example: for a 2 channel counting module it is possible to assign :

- channel 0 to the MAST task
- channel 1 to the FAST task

Analogue modules

Micro analogue input module channels must be assigned to the MAST task. On the other hand it is possible to assign analogue output channels or groups of channels either to the MAST task or the FAST task with a 2 channel modularity.
Example: for a 4 analogue output module it is possible to assign:

- channels 0 and 1 to the MAST task and,
- channels 2 and 3 to the FAST task.

The Premium analogue input and output module channels can be assigned to the MAST task or the FAST task. This assigning is individual for each of the isolated analogue input or output module channels (4 isolated channels) and with a modularity of 4 channels for the other modules.

Note: In order to achieve the best performance, it is preferable to regroup the channels of a module into the same task.

Exchanging inputs/outputs in event processes

General points

It is possible to use input/output channels other than those relating to the event for each process.

Exchanges are then made implicitly by the system before (%I) and after (%Q) in the process to be applied.

These exchanges can be related to a channel (e.g. counting module) or to a group of channels (discrete module). In the second case, if, for example, the process modifies outputs 2 and 3 of a discrete module, the map of outputs 0 - 7 will be transferred to the module.

Operation

The following table describes the exchanges and processes carried out.

Phase	Description
1	The appearance of an event diverts the application program to the process that is associated with the input/output channel which has caused the event.
2	All the inputs associated with the channel that has caused the event are acquired automatically.
3	All the inputs used by the user in the EVTi process are acquired.
4	The event process is carried out. It must be as short as possible.
5	All the outputs used by the user in the EVTi process are updated. The outputs associated with the channel that caused the event must also be used, so that they are updated.

Programming rules

General rule:
The inputs exchanged (and the associated group of channels) when the event process is carried out are updated again (loss of historic values and therefore edges). You must therefore avoid testing edges on these inputs in master tasks (MAST) or fast tasks (FAST).

In the case of modules TOR TSX DEY16FK, TSX DMY28FK or TSX DMY28RFK:
The input which triggered the event must not be tested in the event process (the value is not updated).
Testing the edge which triggered the event must be done on the status word:

- %IWxy.i:X0 = 1 --> rising edge
- %IWxy.i:X0 = 1 --> rising edge

On Micro PL7s:
- analogue input modules which can only be used in the MAST task must not be exchanged in an event process.
- for each event process, it is possible to declare at the most the exchanges for 2 input modules (before the event process) and 2 output modules (after the event process).

Performance

On Premium PL7s, according to the processor used, the number of exchanges used is limited:

Number of exchanges that can be used in event processes by processor	P57-1•	P57-2• /3• /4•
Discrete inputs/outputs	32 exchanges	128 exchanges
Analogue inputs/outputs	8 exchanges	16 exchanges
Other application specific	4 exchanges	16 exchanges

For discrete inputs/outputs an exchange involves a group of 8 channels. It is generated when using inputs from a group of 8 channels (other than the group of channels that generated the event) and when writing the output for a group of 8 channels.

For analogue inputs/outputs or another application specific, an exchange is generated when using the inputs from one channel (other than the channel which generated the event and when writing from channel outputs).

Note:
- As input/output exchanges in the EVTi task are done by channel (for some analogue and application specific modules) or by group of channels (for discrete modules and some analogue modules), if the process modifies outputs 2 and 3 of a discrete module for example, the map (automatic memory) of outputs 0 - 7 will be transferred to the module.
- Any exchange of an input/output in an event task can cause the loss of edge information with regard to the process carried out on this channel (or group of channels), in the task where it was declared. MAST or FAST

Displaying the number of events processed

The system word %SW48 gives the number of events processed.
This word is initialized at 0 on starting from cold, then incremented by the system when an event is launched.
This word can be modified by the user.

The system bit %S39 indicates the loss of an event.

5.4 Function modules

Structuring in function modules

General points

A function module is a regrouping of program elements to carry out an automatic system function.

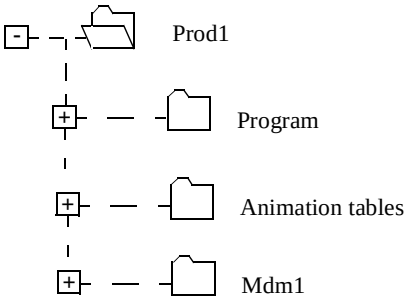
Structure

A function module is defined by the following attributes:

- short name: 8 characters (e.g.: TR371)
- long name: 16 characters (e.g.: Continue/Withdraw for BT371)
- a descriptive form (with no limit to the number of characters) not stored in the PL7 but stored in the .STX file of the application.

Illustration

The illustration below shows how a function module is made up:



Description of function module elements

The table describes the role of each of the elements:

Element	Composition
Program	One or more code modules: ● sections ● events ● macro steps ● animation tables ● ...
Animation tables	One or more animation tables.
Mdm1	Lower level function modules. These modules take on one or more automatic system sub functions in relation to the main function.

Limitations of use

Only the PRO PL7 can be used to set up function modules on Premium PL7s.

Description of PL7 languages

II

Presentation

What's in this spacer

This spacer describes the programming languages for Micro and Premium PL7s.

What's in this part?

This Part contains the following Chapters:

Contact language

6

Presentation

Subject of this chapter

This chapter describes programming in contact language.

What's in this Chapter?

This Chapter contains the following Maps:

General presentation of contact language

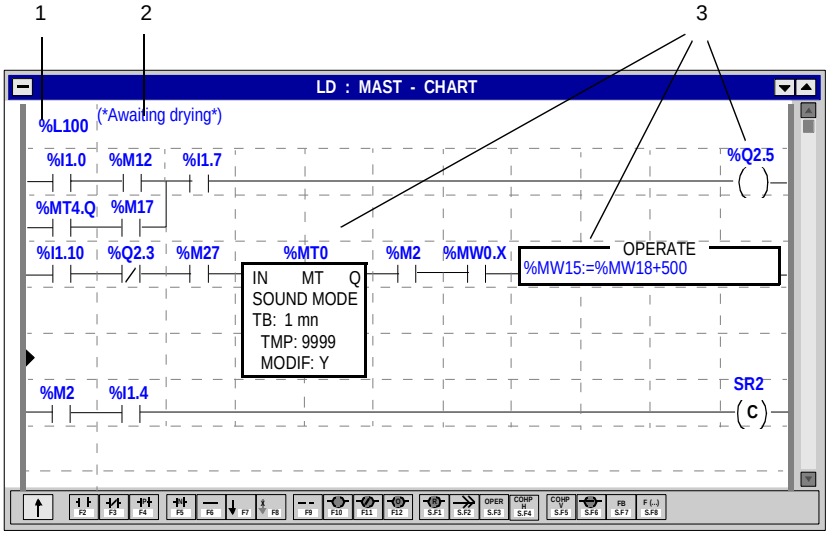
General points

A section of program written in contact language is made up of a suite of network contacts run one after the other by the PL7.

A contact network diagram is similar to an electrical circuit diagram.

Illustration of a contact network

The following screen shows a PL7 contact network.



Composition of a contact network

This table describes how a contact network is made up.

Address	Element	Function
1	Label	Contact network address (optional):
2	Comments	Gives information on a network address (optional):
3	Graphic elements	They represent: • the PL7 inputs and outputs (push buttons, detectors, relays, indicators..) • automatic system functions (timers, counters…,), • arithmetic, logic and specific operations, • the PL7 internal variables.

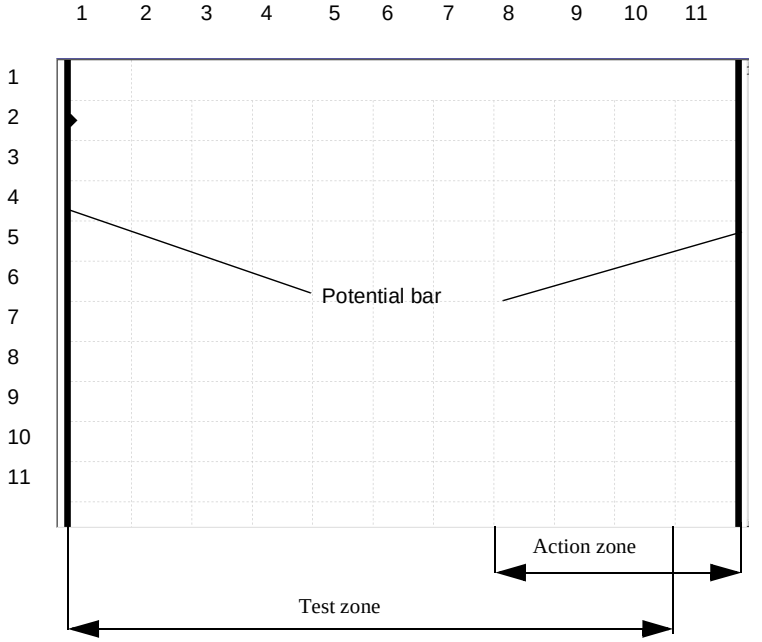
Structure of a contact network

Introduction

A network is entered between two potential bars. The current goes from the left potential bar to the right potential bar.

Illustration

The drawing below describes the structure of a contact network.



Description of a contact network

A contact network is made up of a group of graphic elements placed on a grid of:

- 16 lines maximum and 11 columns (for Premium PL7s),
- 7 lines maximum and 11 columns (for Micro PL7s),

It is divided into two areas:

- the test area, in which the necessary conditions for an action appear
- the action area which applies the consequent result to a test link.

Contact network label

General points

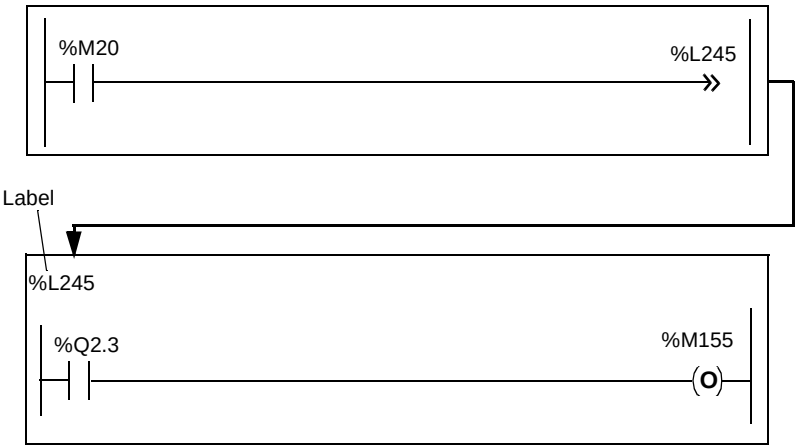
The label is used to locate a network in a program entity (main program, sub-program,...). It is optional.

Syntax

This label has the following syntax: %Li with i between 0 - 999.
It is in the top left section in front of the potential bar.

Illustration

The following contact networks illustrate how a label is used.



Rules

A label address can only be allocated to a single network within the same program entity.

It is necessary to label a network in order to allow a connection after a program jump (see illustration below).

The order of label addresses does not matter, (it is the order of entering the networks that is taken into account by the system when scanning).

Contact network comments

General points

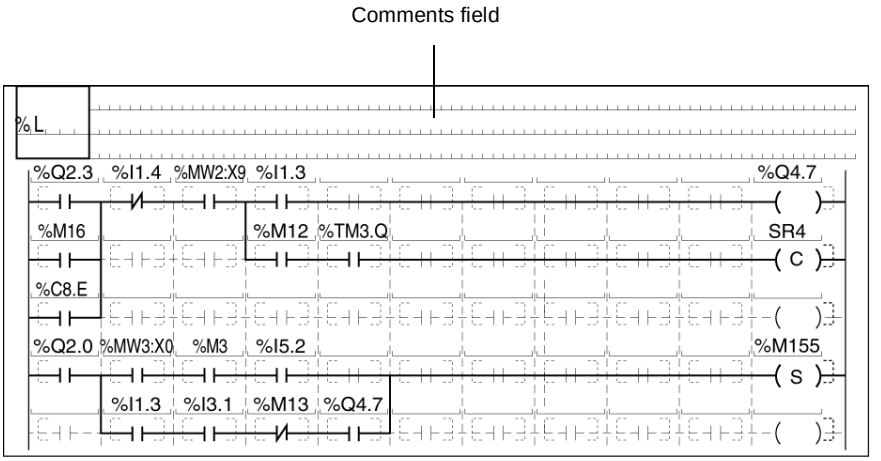
The comments are used to interpret the network to which they are assigned, but they are not obligatory.

Syntax

The comments are integrated into the network and are made up of a maximum of 222 alphanumeric characters, on either side of which are the characters (* and *).

Illustration

The drawing below shows the position of the comments.



Rules

The comments are displayed in the reserved field in the upper part of the contact network.

If a network is deleted, the comments with it are also deleted.

The comments are stored in the PL7 and can be accessed by the user at any time. In this case they take up program memory.

Contact language graphic elements

General points

Graphic elements are the contact language instructions.

Contacts

The contacts graphic elements are programmed in the test area and take up one cell (1 line high and 1 column wide).

Name	Computer art	Functions
Normally open contact		Passing contact when the bit object which controls it is at state 1.
Normally closed contact		Passing contact when the bit object which controls it is at state 0.
Contact for detecting a rising edge		Rising edge: detecting the change from 0 to 1 of a bit object which controls it.
Contact for detecting a falling edge		Falling edge: detecting the change from 1 to 0 of a bit object which controls it.

Link elements

The graphic link elements are used to connect the test and action graphic elements.

Name	Computer art	Functions
Horizontal connection		is used to link in series the test and action graphic elements between the two potential bars.
Potential vertical connection		is used to link the test and action graphic elements in parallel.
Short circuit by pass		is used to link 2 objects through several connections.

Coils

The coil graphic elements are programmed in the test area and take up one cell (1 line high and 1 column wide).

Name	Computer art	Functions
Direct coil		The associated bit object takes the value of the test field result.
Negated coil		The associated bit object takes the negated value of the test field result.
Set coil		The associated bit object is set to 1 when the result of the test field is 1.
Reset coil		The associated bit object is set to 0 when the result of the test field is 1.
Conditional jump to another network (JUMP)	->>%Li	is used to connect to a labeled network, upstream or downstream. Jumps are only made within the same programming entity (main program, sub-program,...). Making a jump causes: ● scanning of a network in progress to stop, ● running of the required labeled network, ● the part of the program between the jump action and the designated network not to be scanned.
Transition condition coil		provided in Grafcet language, used when the programming of the transition conditions associated with the transitions causes a changeover to the next step.
Coil calling up a sub-program (CALL)		is used to connect at the start of a sub-program when the result of the sub-program test field is at 1. Calling up a sub-program means that: ● scanning of the network in progress stops, ● the sub-program runs, ● the scanning of the network that was interrupted resumes.
Return of the sub-program	<RETURN>	Reserved for SR sub-program and allows the calling module to return when the result of the test field is at 1.
Stop program	<HALT>	stops the program running when the result of the test field is at 1.

Standard function blocks

The graphic elements of DFB function blocks are programmed in the test field and take up a maximum of 16 lines in height and 3 columns wide.

Name	Computer art	Functions
Timer blocks, counter, monostable, register, cyclical programmer		Each of the standard function blocks uses inputs, outputs, inputs/outputs which enable links to the other graphic elements.

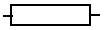
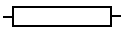
DFB function blocks

The graphic elements of DFB function blocks are programmed in the test field and take up a maximum of 16 lines in height and 3 columns wide.

Name	Computer art	Functions
Programmable blocks		Each of the DFB function blocks uses inputs, outputs, inputs/outputs which enable links to other graphic elements for bit objects or which can be assigned to numeric or table objects

Operation blocks

Operation block graphic elements are programmed in the test field and take up the space mentioned below.

Name	Computer art	Functions
Vertical comparison block		is used to compare 2 operands, according to the result the corresponding output changes to 1. Size: 2 columns/ 4 lines
Horizontal comparison block		is used to compare 2 operands, the output changes to 1 when the result is checked (one block can contain up to 4096 characters). Size: 2 columns/ 1 line
Operation block		carries out the arithmetic and logic operations…calls up the structured text language syntax. (One block can contain up to 4096 characters). Size: 4 columns/ 1 line

Rules for programming a contact network

General points

Programming a network contact is done using graphic elements, observing the following programming rules.

Programming rules

Single test and action graphic elements each take up one cell within a network.

Each contact line begins on the left potential line and must finish on the right potential line.

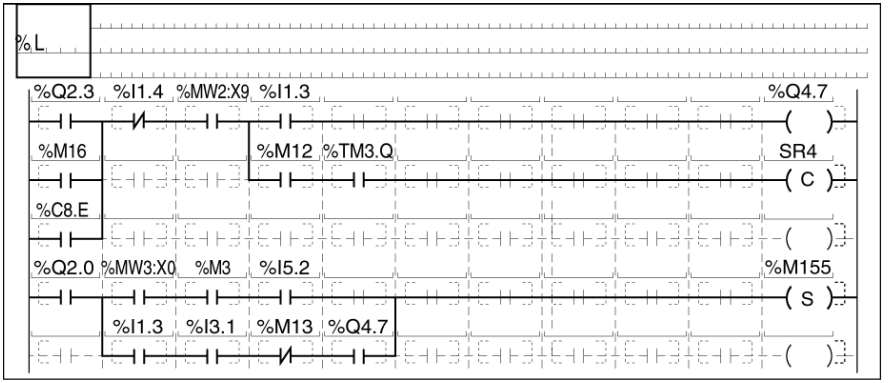
Tests are always in columns 1 - 10.
Actions are always in column 11.

The direction of the current is as follows:

- for horizontal links from **left to right**,
- for vertical links, in both directions.

Example of a contact network

The following screen shows an example of a contact network.



Rules for programming function blocks

General points

Standard function blocks are found in the test field of the contact networks.

Rules for programming function blocks

Whatever type of function block used, it must be linked at the input to the left potential bar, directly or through other graphic elements.

- **outputs "in the air":** it is not necessary to link function block outputs to other graphic elements,
- **outputs that can be tested:** function block outputs can be accessed by the user in the form of a bit object.

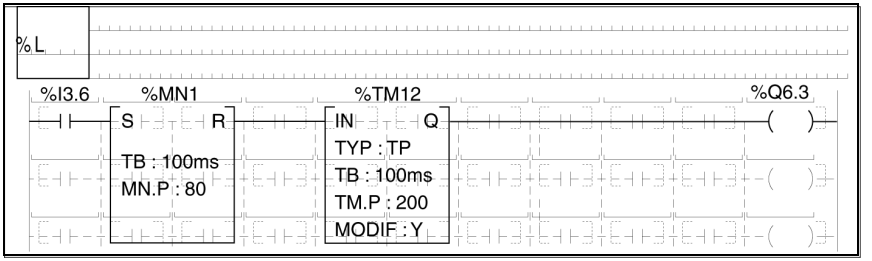
Internal block variables and graphic outputs are objects that can be used remotely from another part of the program.

Non hardwired standard function block inputs are set to 0.

Just as with the contact type graphic elements, it is possible to have combinations of function blocks.

Example of a contact network

The following illustration shows an example of a contact network containing 2 function blocks.



Rules for programming operation blocks

General points

Comparison blocks are in the test field and operation blocks are in the action field.

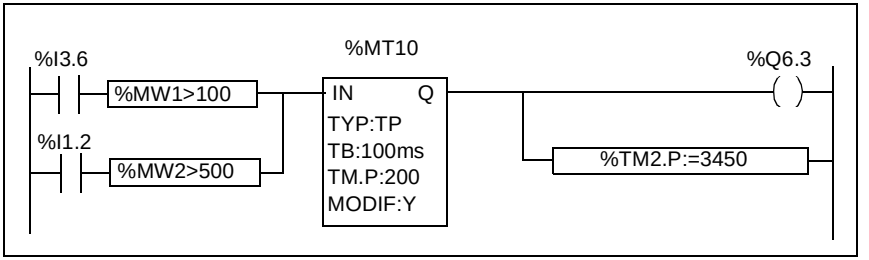
Rules for programming operation blocks

Whatever type of operation block used, it must be linked at the input to the left potential bar, directly or through other graphic elements.

Just as with the contact type graphic elements, it is possible to have combinations of function and operation blocks.

Example of operation blocks

The following illustration shows an example of a contact network containing 2 comparison blocks and one operation block.



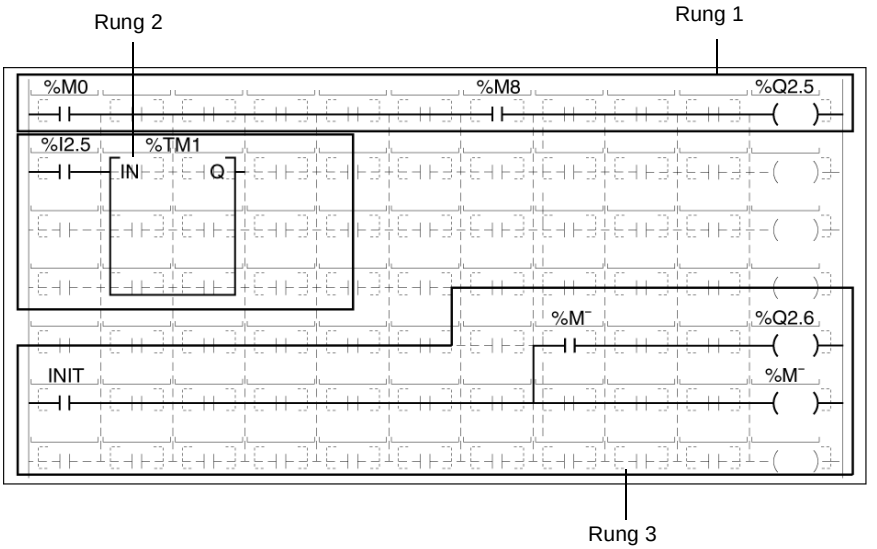
Running a contact network

Rung

A rung contains graphic elements which are all linked to each other by link elements (except potential bar), but independent of the other network graphic elements (no vertical links to the top or bottom within the rung).

Illustration of rungs

The following contact network is made up of 3 rungs.



Rules for running rungs

The first rung evaluated is the one with the left corner in the top left.

A rung is evaluated in the direction of the equation: evaluating the network from top to bottom, line by line and each line from left to right.

When a vertical convergence line is encountered, the sub network associated with it is evaluated (according to the same logic) before continuing evaluating the network surrounding it.

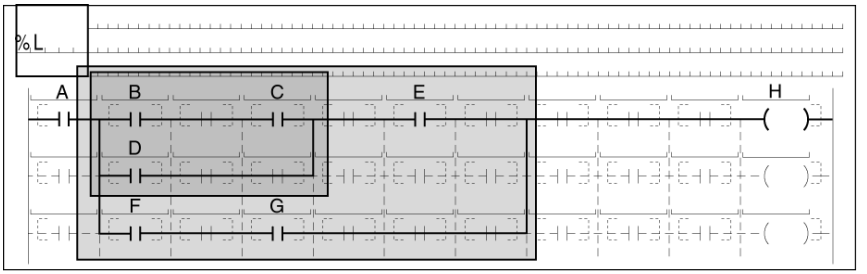
Running elements in a rung

The following table describes the running order for elements in a rung.

Phase	Description
1	The system evaluates the logic state of each contact according to: ● the current value of application internal objects, ● the state of input/output module entries from the beginning of the cycle
2	The system runs the processes associated with the functions, block functions and sub programs,
3	The system updates the bit objects associated with the coils (updating the input/output module outputs is done at the end of the cycle),
4	The system disconnects and goes to another labeled network on the same program module (jump to another %Li ->>network), goes back to a calling module <RETURN>, or program stop<HALT>,

Example 1: illustration

The following drawing displays the running order for graphic elements.



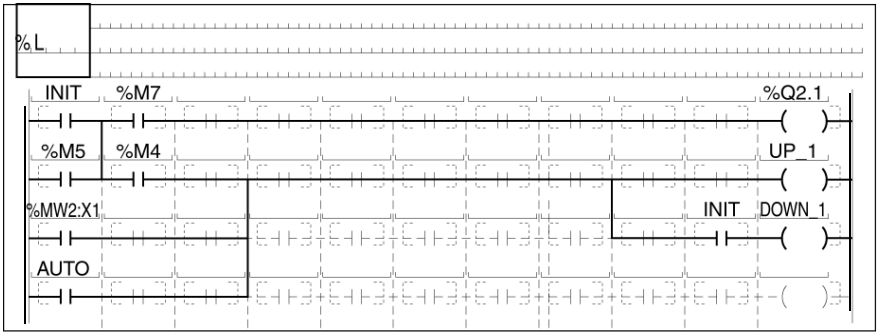
Example 1: operation

The following table describes the running of graphic elements in the network illustrated above.

Phase	Description
1	Evaluating the network up to the first vertical convergence link: contacts A, B, C.
2	Evaluating the first sub network: contact D,
3	Continuing the evaluation of the network up to the second vertical convergence line: contact E,
4	Evaluating the 2nd sub network: contacts F and G,
5	Evaluating coil H.

Example 2: illustration

The following drawing displays the running order for graphic elements.



Example 2: operation

The following table describes the running of graphic elements in the network illustrated above.

Phase	Description
1	coil 1: INIT, %M5, %M7, %Q2.1,
2	coil 2: %M4, %MW2:X1,AUTO, UP_1,
3	Operation block

Instruction list language

7

Presentation

Subject of this chapter

This chapter describes the rules for programming in instruction list language.

What's in this Chapter?

This Chapter contains the following Maps:

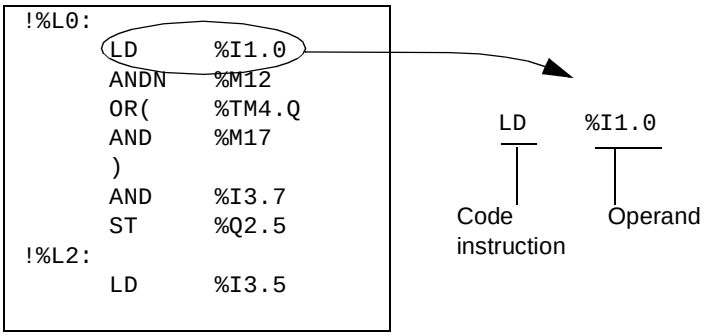
General presentation of instruction list language

General points

A section written in instruction list language is made up of a suite of instructions run in sequence by the PL7.

Illustration of a program

The following illustration shows a PL7 instruction list program and an instruction detail.



Composing an instruction

This tables describes what an instruction is made up of.

Element	Function
Instruction code	The instruction code determines the operation to be run. There are 2 types of instruction code: ● **test**, in which the necessary conditions for an action appear (e.g.: LD, AND, OR...), ● **action**, which applies the consequent result to a test link. (e.g.: ST, STN, R, ...).
Opérand	An instruction acts on an operand. This operand can be: ● a PL7 input/output (push buttons, detectors, relays, indicators...), ● automatic system functions (timers, counters…), ● an arithmetic logic operation or a transfer operation, ● a PL7 internal variable.

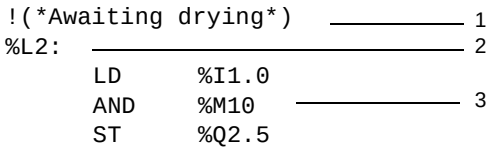
Structure for an instruction list program

General points

Just as with contact language, instructions are organized in instruction sequence (equivalent to a contact network) called a sequence.

Example of a sequence

The following illustration shows a PL7 instruction list sequence.



Description of a sequence

Each sequence begins with an exclamation mark (generated automatically). It includes the following elements.

Address	Element	Function
1	Comments	Enter a sequence (optional).
2	Label	Locate a sequence (optional).
3	Instructions	One or more test instructions, the result of these instructions being applied to one or more action instructions. One instruction takes up a maximum of one line

Label for a sequence in instruction list language

General points

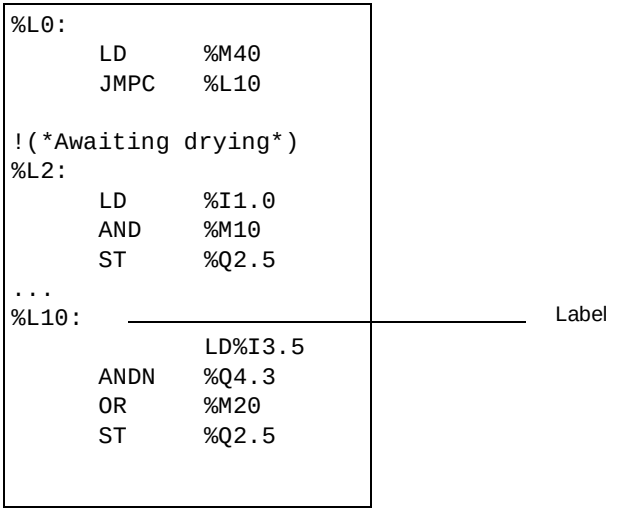
The label is used to locate a sequence in a program entity (main program, sub-program,...). It is optional

Syntax

This label has the following syntax: %Li with i between 0 and 999. It is at the beginning of a sequence.

Illustration

The following program shows how the label is used.



Rules

The same label can only be allocated to a single sequence within the same program entity.

It is necessary to label a sequence so that connection can be made after a program jump.

The order of label addresses does not matter, (it is the order of entering the sequences that is taken into account by the system when scanning).

Comments on a sequence in instruction list language

General points

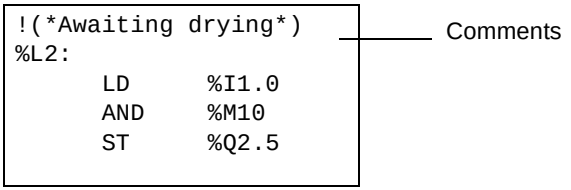
The comments make interpreting a sequence to which they are assigned easier. They are optional.

Syntax

The comments can be integrated at the beginning of a sequence and can take up up to 3 lines (i.e. 222 alphanumeric characters), on either side of which are the characters (* and *).

Illustration

The following illustration locates the position of the comments in a sequence.



Rules

The comments are only displayed from the first line of the sequence.

If a sequence is deleted, the comments with it are also deleted.

The comments are stored in the PL7 and can be accessed by the user at any time. In this case they take up program memory.

Presenting instructions in instruction list language

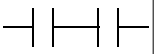
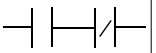
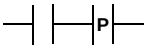
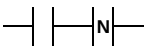
General points

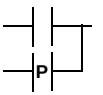
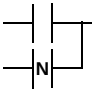
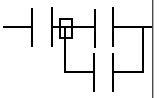
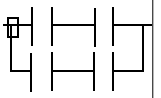
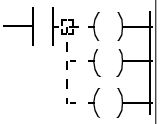
The instruction list language is made up of the following :

- test instructions
- action instructions
- on a function block
- numerical

Test instructions

The following table describes test instructions in instruction list language.

Name	Equivalent computer art	Functions
LD		The Boolean result is the same as the status of the operand.
LDN		The Boolean result is the same as the reverse status of the operand.
LDR		The Boolean result changes to 1 on detection of the operand (rising edge) changing from 0 to 1.
LDF		The Boolean result changes to 1 on detection of the operand (falling edge) changing from 1 to 0.
AND		The Boolean result is equal to the And logic between the Boolean result of the previous instruction and the status of the operand.
ANDN		The Boolean result is equal to the And logic between the Boolean result of the previous instruction and the reverse status of the operand.
ANDR		The Boolean result is equal to the And logic between the Boolean result of the previous instruction and the detection of the operand's rising edge (1 = rising edge).
ANDF		The Boolean result is equal to the And logic between the Boolean result of the previous instruction and the detection of the operand's falling edge (1 = falling edge).
OR		The Boolean result is equal to the Or logic between the Boolean result of the previous instruction and the status of the operand.
ORN		The Boolean result is equal to the Or logic between the Boolean result of the previous instruction and the reverse status of the operand.

Name	Equivalent computer art	Functions
ORR		The Boolean result is equal to the Or logic between the Boolean result of the previous instruction and the detection of the operand's rising edge (1 = rising edge).
ORF		The Boolean result is equal to the Or logic between the Boolean result of the previous instruction and the status of the operand and the detection of the operand's falling edge (1 = falling edge).
AND(		Logic And (8 parenthesis levels)
OR(		Logic Or (8 parenthesis levels)
XOR, XORN, XORR, XORF	-	Exclusive Or
MPS MRD MPP		Switching to the coils.
N	-	Negation

Action instructions

The following table describes test instructions in instruction list language.

Name	Computer art	Functions
ST		The associated operand takes the value of the test field result.
STN		The associated operand takes the reverse value of the test field result.
S		The associated operand is set to 1 when the result of the test field is 1.
R		The associated operand is set to 0 when the result of the test field is 1.
JMP	-	is used to connect unconditionally to a labeled sequence, upstream or downstream.

Name	Computer art	Functions
JMPC	-	is used for a conditioned connection to a Boolean result at 1, to a labeled sequence upstream or downstream.
JMPCN	-	is used for a conditioned connection to a Boolean result at 0, to a labeled sequence upstream or downstream.
SRn	-	Connection at the beginning of a sub program.
RET	-	Return of the sub-program
RETC	-	Return of the conditioned sub program to a Boolean result at 1.
RETCN	-	Return of the conditioned sub program to a Boolean result at 0.
END	-	End of program.
ENDC	-	End of the conditioned program at a Boolean result of 1.
ENDCN	-	End of the conditioned program at a Boolean result of 0.

Instruction on a function block

The following table describes test instructions in instruction list language.

Name	Computer art	Functions
Timer blocks, counter, monostable, register, cyclical programmer		For each of the standard function blocks, there are instructions for controlling the block. A structured form is used to hardwire the block inputs and outputs directly.

Numeric instructions

The following table describes test instructions in instruction list language.

Name	Instructions	Functions
Test element	LD[.....] AND[.....] OR[.....]	is used to compare 2 operands. The output goes to 1 when the result is checked. **Example** : LD[%MW10<1000] Result to 1 when %MW10<1000.
Action element	[.....]	carry out arithmetic logic operations… use the structured text language syntax. **Example** : [%MW10:=%MW0+100] The result of the %MW0+100 operation is placed in the internal word %MW10.

Rule for using parentheses in instruction list language

General points

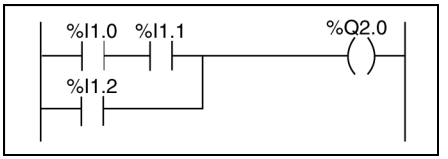
The instructions AND and OR can use parentheses.
These parentheses are used to make up simple contact diagrams.

Principle

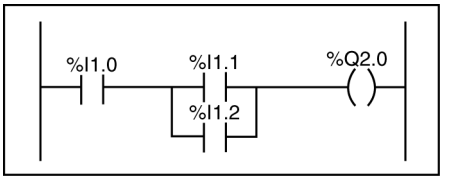
Opening parentheses is associated with the instruction AND or OR.
Closing parentheses is an instruction. It must be done for each open parenthesis.

Example: AND(

The 2 following programs show how parentheses are used.



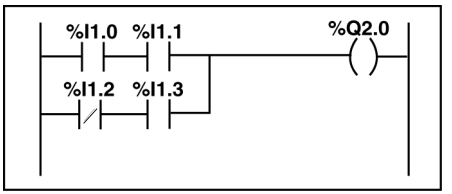

```
LD      %I1.0
AND     %I1.1
OR      %I1.2
ST      %Q2.0
```




```
LD      %I1.0
AND(    %I1.1
OR      %I1.2
)
ST      %Q2.0
```

Example: OR(

The following program shows how the parenthesis is used.




```
LD      %I1.0
AND     %I1.1
OR(N    %I1.2
AND     %I1.3
)
ST      %Q2.0
```

Associating parentheses to modifiers

The following "modifiers" can be associated with parentheses.

Code	Role	Example
N	Negation	AND(N
F	Falling edge	AND(F
R	Rising edge	OR(R
[	Comparison	OR([%MW0>100]

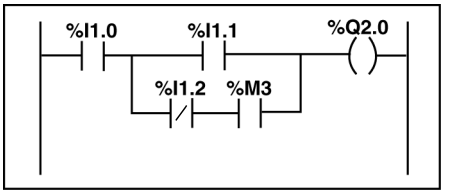
Overlapping of parentheses

It is possible to overlap up to 8 parenthesis levels.
The following rules must be observed:

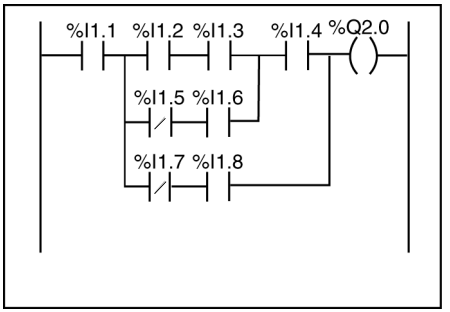
- Each open parenthesis must be closed
- The labels `%Li:` must not be put in expressions in parentheses, or the jump instructions `JMP` and calling up sub program instructions `SRi`,
- Assigning instructions `ST`, `STN`, `S` and `R` must not be programmed in parentheses.

Example:

The following programs show how to use overlapping of parentheses.




```
LD      %I1.0
AND(    %I1.1
OR(N    %I1.2
AND     %M3
)
)
ST      %Q2.0
```




```
LD      %I1.1
AND(    %I1.2
AND     %I1.3
OR(N    %I1.5
AND     %I1.6
)
AND     %I1.4
OR(N    %I1.7
AND     %I1.8
)
)
ST      %Q2.0
```

Description of the MPS, MRD and MPP instructions

General points

The 3 instruction types are used to switch to the coils.

These instructions use an intermediate memory called a stack which can store up to 3 Boolean instructions…

Note: These instructions cannot be used in an expression in parentheses

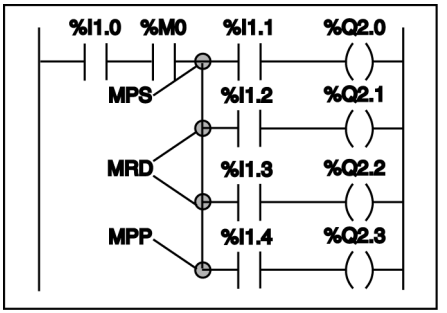
Role

The following table describes the role of each of the instructions

Instruction	Role
MPS (Memory PuSh)	This instruction stores the results of the last test instruction at the top of the stack and moves the other values to the bottom of the stack.
MRD (Memory ReaD)	This instruction reads the top of the stack.
MPP (Memory PoP)	This instruction reads, draws down the top of the stack and moves the other values towards the top of the stack.

Example 1

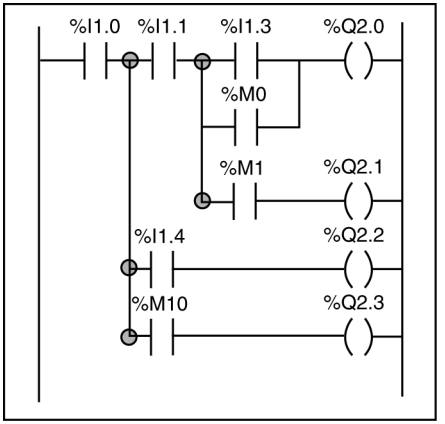
This example shows how to use the MPS, MRD and MPP instructions.




```
LD      %I1.0
AND     %M0
MPS
AND     %I1.1
ST      %Q2.0
MRD
AND     %I1.2
ST      %Q2.1
MRD
AND     %I1.3
ST      %Q2.2
MPP
AND     %I1.4
ST      %Q2.3
```

Example 2

This example shows how the MPS, MRD and MPP instructions operate.




```
LD    %I1.0
MPS
AND   %I1.1
MPS
AND(  %I1.3
OR    %M0
)
ST    %Q2.0
MPP
AND   %M1
ST    %Q2.1
MRD
AND   %I1.4
ST    %Q2.2
MPP
AND   %M10
ST    %Q2.3
```

Principles of programming pre-defined function blocks

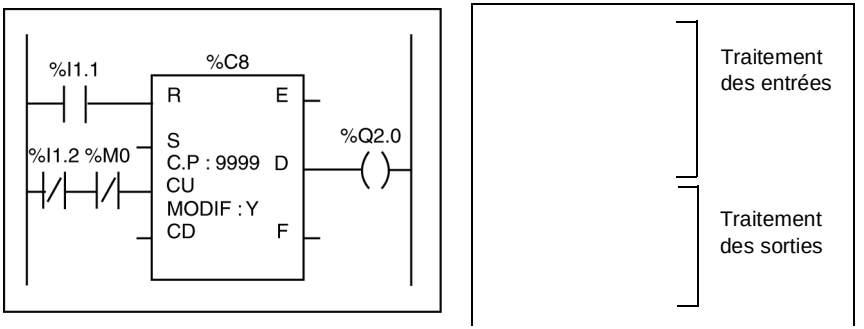
General points

Automatic system function blocks can be programmed in 2 different ways:
- with specific instructions for each function block (e.g.: `CU %Ci`), this method is the simplest and the most direct,
- with block structuring instructions `BLK` ,`OUT_BLK`, `END_BLK`.

Principle of direct programming

The instructions control the block inputs (e.g.: `CU`). The outputs can be accessed in bit form (e.g.: `%C8.D`).

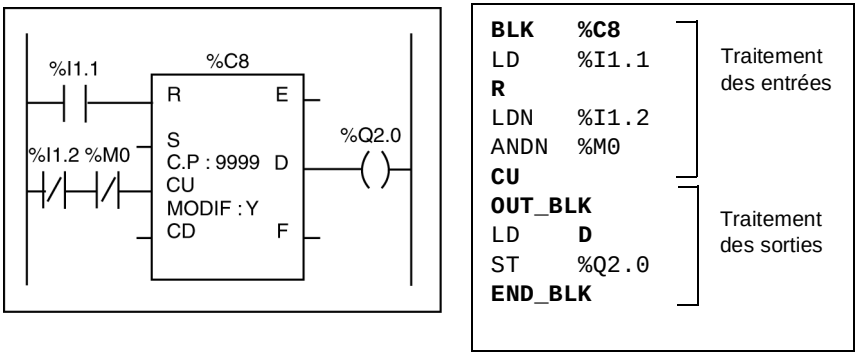
Example:
This example shows direct programming of a counter function block.



Principle of structured programming

This type of programming uses a set of instructions with instructions round them:
- `BLK` indicates the beginning of the block
- `OUT_BLK` is used to hardwire the block outputs directly
- `END_BLK` indicates the end of the block

Example:
This example shows structured programming of a counter function block.



Note: This principle of structured programming which needs the additional instructions BLK, OUT_BLK and END_BLK requires a greater amount of memory than direct programming. It must, however, be used if you want to keep the similarity between reversible programs for nano TSX 07 PL7s.

Rules for running an instruction list program

Principle

Running an instruction list program is done sequentially instruction by instruction.

The first instruction for an instruction sequence must always be an LD instruction, i.e. an unconditional instruction (e.g.: JMP).

Each instruction (except for LD and unconditional instructions) uses the Boolean result of the previous instruction.

Example 1

The program below describes the complete run of a sequence.

```
LD      %I1.1    résultat = état du bit %I1.1
AND     %M0      résultat  = ET du résultat booléen précédent et de l'état du bit %M0
OR      %M10     résultat  = OU du résultat booléen précédent et de l'état du bit %M10
ST      %Q2.0    %Q2.0 prend l'état du résultat booléen précédent
```

Example 2

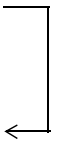
The parentheses are used to modify the order that the Boolean results are taken into account:

```
LD      %I1.1    résultat = état du bit %I1.1
AND     %M0      résultat  = ET du résultat booléen précédent et de l'état du bit %M0
OR(     %M10     résultat  = état du bit %M10
AND     %I1.2    résultat  = ET du résultat booléen précédent et de l'état du bit %M10
)
ST      %Q2.0    %Q2.0 prend l'état du résultat booléen précédent
```

Example 3

Sequencing instructions can be modified by jump instructions JMP for calling up a sub program.

```
!       LD      %M0
        JMPC    %L10
!       LD      %I1.1
        AND     %M10
        ST      %Q2.0
!       %L10:
        LD      %I1.3
        AND     %M20
        ......
```



Saut à l'étiquette %L10 si %M0=1

Structured text language

8

Presentation

Subject of this chapter

This chapter describes the rules for programming in structured text language.

What's in this Chapter?

This Chapter contains the following Maps:

Presentation of structured text language

General points

Structured text language is a developed algorithmic language that is specially adapted for programming complex arithmetic functions, manipulating tables and managing messages.

It is used to make up programs by writing programming lines made up of alphanumeric characters.

Limits of use

This language can be used with Junior and Pro PL7 software on Premium and Micro PL7s.
In the Pro PL7 version, this language is used to create DFB user function blocks on Premium PL7s.

Illustration of a program

The following illustration shows a program in PL7 structured language.

```
!     (* Searching for the first element that is not zero
in a
      table of 32 words, determining its value
      (%MW10), its rank (%MW11). This search
      is done if %M0 is set to 1, %M1 is set to 1 if
      an element which is not zero exists unless it is
set to 0*)

      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;
```

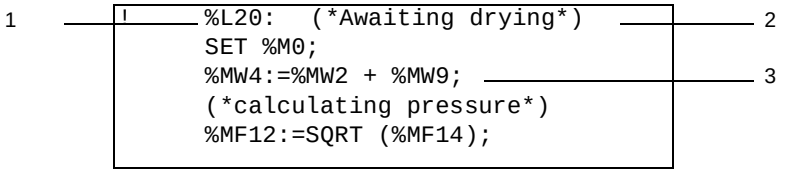
Structuring a program in structured text language

General points

A section of text program is organized into sequences.
A text sequence is the equivalent of a contact network in contact language.

Example of a sequence

The following illustration shows a sequence in PL7 structured language.



Description of a sequence

Each sequence begins with an exclamation mark (generated automatically). It includes the following elements.

Address	Element	Function
1	Label	Locates a sequence.
2	Comments	Fill in a sequence.
3	Instructions	One or more instructions separated by ";".

Note: Each of these elements is optional, that is it is possible to have an empty sequence, a sequence made up only of comments or only of a label.

Label for a sequence in structured text language

Role

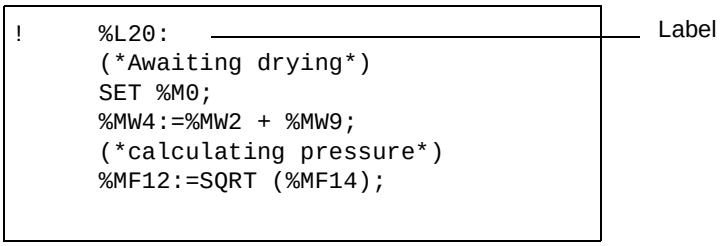
The label is used to locate a sequence in a program entity (main program, sub-program,...). It is optional.

Syntax

This label has the following syntax: %Li : with i between 0 and 999. It is at the beginning of a sequence.

Illustration

The following program shows how the label is used.



Rules

The same label can only be allocated to a single sequence within the same program entity.

It is necessary to label a sequence so that connection can be made after a program jump.

The order of label addresses does not matter, (it is the order of entering the sequences that is taken into account by the system when scanning).

Comments on a sequence in structured text language

Role

The comments make interpreting a sequence to which they are assigned easier. They are optional.

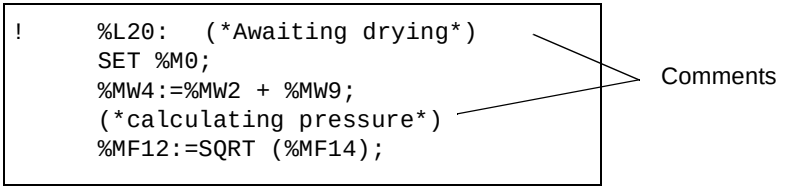
Syntax

The comments can be integrated anywhere in the sequence and the number of comments per sequence is not limited.

A comment is surrounded on both sides by the characters (* and *).

Illustration

The following illustration locates the position of the comments in a sequence.



Rules

- Any character is allowed in a comment.
- The number of characters is limited to 256 per commentary.
- Comments may not overlap.
- A comment can contain several lines.

The comments are stored in the PL7 and can be accessed by the user at any time. In this case they take up program memory.

Bit object instructions

Bits instructions

The following instructions apply to bit objects.

Designation	Function
:=	Assignment of a bit
OR	boolean OR
AND	boolean AND
XOR	boolean exclusive OR
NOT	Inversion
RE	Rising edge
FE	Falling edge
SET	Set to 1:
RESET	Set to 0:

Bits table instructions

The following instructions apply to bits table objects.

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

Arithmetic and logic instructions

Whole arithmetic on words and double words

The following instructions apply to word and double word objects…

Name	Function
+, -, *, /	Addition, Subtraction, Multiplication, whole Division
REM	Remainder of whole division
SQRT	Whole square root
ABS	Absolute value
INC	Incrementation
DEC	Disincrementation

Arithmetic on floating points

The following instructions are applied to floating objects...

Name	Function
+, -, *, /	Addition, Subtraction, Multiplication,Division
SQRT	Square root
ABS	Absolute value
TRUNC	Whole part
LOG	Base 10 logarithm
LN	Napierian logarithm
EXP	Natural exponential
EXPT	Exponentiation of an actual by a whole
COS	Cosine of a value in radians
SIN	Sine of a value in radians
TAN	Tangent of a value in radians
ACOS	Cosine arc (result between 0 and 2 p)
ASIN	Sine arc (result between -p/2 and +p/2)
ATAN	Tangent arc (result between -p/2 and +p/2)
DEG_TO_RAD	Converting degrees to radians
RAD_TO_DEG	Converting radians to degrees

Logic instructions on words and double words

The following instructions are applied to word and double word objects.

Name	Function
AND	logic AND
OR	logic OR
XOR	exclusive logic OR
NOT	Logic complement
SHL	Logic shift to left
SHR	Logic shift to right
ROL	Circular logic shift to left
ROR	Circular logic shift to right

Numerical comparisons on words, double words and floating points

The following instructions are applied to word, double word and floating objects.

Name	Function
<	Clearly less than
>	Clearly more than
<=	Less or equal to
>=	More or equal to
=	Equal to
<>	Different from

Instructions for tables and character strings

Instructions for word tables and double words

The following instructions are applied to word tables and double words.

Name	Function
Table: = Table	Assigning between two tables
Table: = Word	Initializing a table
+, -, *, /, REM	Arithmetic operations between tables
+, -, *, /, REM	Arithmetic operations between expressions and tables
SUM	Totaling the elements in a table
EQUAL	Comparing two tables
NOT	Logic complement for a table
AND, OR, XOR	Logic operations between two tables
AND, OR, XOR	Logic operations between expressions and tables
FIND_EQW, FIND_EQD	Finding the first element equal to a value
FIND_GTW, FIND_GTD	Finding the first element greater than a value
FIND_LTW, FIND_LTD	Finding the first element less than a value
MAX_ARW, MAX_ARD	Finding the maximum value in a table
MIN_ARW, MIN_ARD	Finding the minimum value in a table
OCCUR_ARW, OCCUR_ARD	Number of times a value occurs in a table
SORT_ARW, SORT_ARD	Sorting a table in ascending or descending order
ROL_ARW, ROL_ARD	Shifting a table to the left in a circular movement
ROR_ARW, ROR_ARD	Shifting a table to the right in a circular movement
FIND_EQWP,FIND_EQDP	Finding the first element from a rank equal to a value
LENGTH_ARW, LENGTH_ARD	Calculating the length of a table

Instructions for floating point tables

The following instructions are applied to floating point tables.

Name	Function
Table: = Table	Assigning between two tables
Table: = Floating point	Initializing a table
SUM_ARR	Totaling the elements in a table
EQUAL_ARR	Comparing two tables
FIND_EQR	Finding the first element equal to a value
FIND_GTR	Finding the first element greater than a value
FIND_LTR	Finding the first element less than a value
MAX_ARR	Finding the maximum value in a table
MIN_ARR	Finding the minimum value in a table
OCCUR_ARR	Number of times a value occurs in a table
SORT_ARR	Sorting a table in ascending or descending order
ROL_ARR	Shifting a table to the left in a circular movement
ROR_ARR	Shifting a table to the right in a circular movement
LENGTH_ARR	Calculating the length of a table

Instructions for character strings

The following instructions are applied to character strings.

Name	Function
STRING_TO_INT	Converting ASCII to binary (single word format)
STRING_TO_DINT	Converting ASCII to binary (double word format)
INT_TO_STRING	Converting binary (single word format) to ASCII
DINT_TO_STRING	Converting binary (double word format) to ASCII
STRING_TO_REAL	Converting ASCII to floating point
REAL_TO_STRING	Converting floating point to ASCII
<, >, <=, >=, =, <>	Alphanumeric comparison
FIND	Position of a sub-chain
EQUAL_STR	Position of the first different character
LEN	Length of a character string
MID	Extracting a sub-chain
INSERT	Inserting a sub-chain
DELETE	Deleting a sub-chain
CONCAT	Joining two strings

Name	Function
REPLACE	Replacing a string
LEFT	Start of string
RIGHT	End of string

Instructions for numerical conversions

Instructions for numerical conversions

These instructions convert bits, words, double words and floating points.

Name	Function
BCD_TO_INT	Converting BCD to binary
INT_TO_BCD	Converting binary to BCD
GRAY_TO_INT	Converting Gray to binary
INT_TO_REAL	Converting a single whole format into floating
DINT_TO_REAL	Converting a double whole format into floating
REAL_TO_INT	Converting a floating point into single whole format
REAL_TO_DINT	Converting a floating point into a double whole format
DBCD_TO_DINT	Converting a 32 bit BCD number into a 32 bit whole
DINT_TO_DBCD	Converting a 32 bit whole number into a 32 bit BCD number
DBCD_TO_INT	Converting a 16 bit BCD number into a 32 bit whole
INT_TO_DBCD	Converting a 16 bit whole number into a 32 bit BCD number
LW	Extracting the least significant word from a double word
HW	Extracting the most significant word from a double word
CONCATW	Joining two single words

Instructions for programs and specific instructions

Program instructions

The following instructions do not affect language objects but the running of the program.

Name	Function
HALT	Stopping the running of the program
JUMP	Jumping to a label
SRi	Calling up a sub program
RETURN	Return of the sub-program
MASKEVT	Masking events in the PL7
UNMASKEVT	Unmasking events in the PL7

Instructions on time management

The following instructions carry out operations on dates, times and durations…

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

"Orpheus" instructions

The following instructions are specific to the Orpheus language.

Name	Function
WSHL_RBIT, DSHL_RBIT	Shifting a word to the left with recovery of shifted bits
WSHR_RBIT, DSHR_RBIT	Shifting a word to the right with sign extension and recovery of shifted bits
WSHRZ_C, DSHRZ_C	Shifting a word to the right with filling in with 0 and recovery of shifted bits
SCOUNT	Counting/counting down with indication of overrun
ROLW,ROLD	Circular shift to the left
RORW,RORD	Circular shift to the right

Timing instructions

These instructions are timing functions to be used for programming DFB codes..

Name	Function
FTON	Time until actuation
FTOF	Time until reset
FTP	Pulse time
FPULSOR	Generating rectangular signals

Conditional check structure IF...THEN

Role

This check structure takes one or more actions if one condition is true. In its general form there can be a number of conditions.

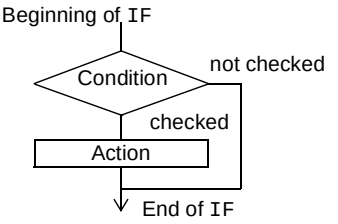
Simple form

In its simple form the check structure has the following syntax and operation.

Syntax

```
IF condition THEN
    actions ;
END_IF;
```

Operation



Example:

```
! (*Conditional action IF (simple form)*)
  IF %M0 AND %M12 THEN
       RESET %M0;
       INC %MW4;
       %MW10:=%MW8+%MW9;
  END_IF;
```

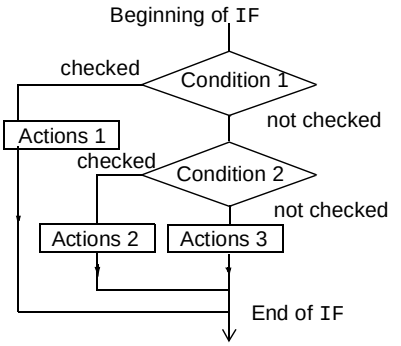
General form

In its general form the check structure has the following syntax and operation.

Syntax

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

Operation



Example:

```
! (*Conditional action IF (simple form)*)
  IF %M0 AND %M1 THEN
       %MW5:=%MW3+%MW4;
       SET %M10;
  ELSEIF %M0 OR %M1 THEN
       %MW5:=%MW3-%MW4;
       SET %M11;
  ELSE
       RESET %M10;
       RESET %M11;
  END_IF;
```

Programming rule

- There can be a number of conditions.
- Each action represents an instruction list.
- Several IF check structures can be overlapped.
- There is no limit to the number of ELSIF.
- There is a maximum of one ELSE.

Conditional check structure WHILE...END_WHILE

Role

This check structure carries out a repetitive action as long as a condition is verified.

Description

The check structure has the following syntax and operation.

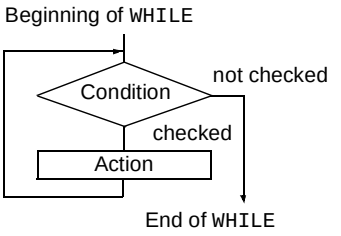
Syntax

```
WHILE condition DO

    action ;

END_WHILE;
```

Operation



Example:

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

Programming rule

- It can be a multiple condition.
- The action represents an instruction list.
- The test on the condition is done before the action is carried out. If when the condition is first evaluated its value is wrong, then the action is never carried out.
- Several WHILE check structures can be overlapped.

Note: The instruction EXIT (See *Role, p. 162*) is used to stop the loop running and to continue to the instruction following the END_WHILE.

Conditional check structure REPEAT...END_REPEAT

Role

This check structure carries out a repetitive action until a condition is verified.

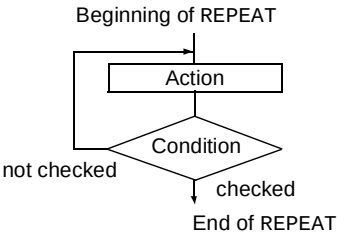
Description

The check structure has the following syntax and operation.

Syntax

```
REPEAT
    action ;
UNTIL condition END_REPEAT;
```

Operation



Example:

```
! (*REPEAT conditional repeated action*)
  REPEAT
       INC %MW4;
       SET %M25[%MW4];
  UNTIL %MW4>12 END_REPEAT;
```

Programming rule

- It can be a multiple condition.
- The action represents an instruction list.
- The test on the condition is done after the action is carried out. If when the condition is first evaluated its value is wrong, then the action is carried out once.
- Several REPEAT check structures can be overlapped.

Note: The instruction EXIT (See *Role, p. 162*) is used to stop the loop running and to continue to the instruction following the END_REPEAT.

Conditional check structure FOR...END_FOR

Role

This check structure carries out a process a certain number of times by adding 1 to each loop index.

Description

The check structure has the following syntax and operation.

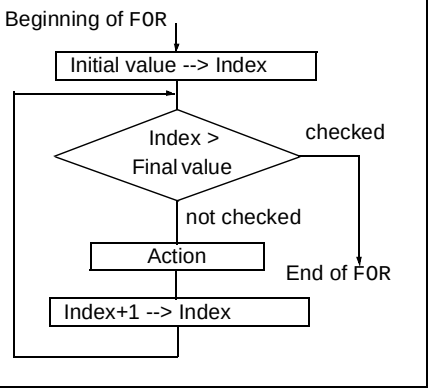
Syntax

```
FOR index:=initial value TO final
value DO

    action ;

END_FOR;
```

Operation



Example:

```
! (*Repeated action FOR*)
  FOR %MW4=0 TO %MW23+12 DO
       SET %M25[%MW4];
  END_FOR;
```

Programming rule

- When the index is clearly greater than the final value, the operation continues on the instruction following the END_FOR.
- Increasing the index is done automatically and is therefore not under your control.
- The action represents an instruction list.
- The initial value and the final value must be numeric word type expressions.
- The index must be a word type object with write access.
- Several FOR check structures can be overlapped.

Note: The instruction EXIT (See *Role, p. 162*) is used to stop the loop running and to continue to the instruction following the END_FOR.

Output instruction for the EXIT loop

Role

This instruction is used to stop the loop running and to go on to the instruction following the end of loop instruction.

Programming rule

- This instruction can only be used in the actions of one of 3 loops WHILE, REPEAT or FOR.
- This instruction is linked to the nearest incorporated loop, i.e. it does not stop all the loops which incorporate it from running.

Example

In this example the instruction EXIT is used to stop the REPEAT loop but never the WHILE loop.

```
! (*Instruction for exiting the loop EXIT*)
  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;
```

Rules for running a structured text program

General points

Running a text program is done sequentially, instruction by instruction observing the check structures.

In the case of arithmetic or Boolean expressions made up of several operators, priority rules are defined between the various operators.

Operator priority rule

The table below gives the priorities for evaluating an expression with a greater or lesser priority.

Operator	Symbol	Priority
Parentheses	(expression)	Greater
Logic complement Inversion - on operand + on operand	NOT NOT - +	
Multiplication Division Rollover	* / REM	
Addition Subtraction	+ -	
Comparisons	<,>,<=,>=	
Equality comparison Inequality comparison	= <>	
logic AND Boolean AND	AND AND	
logic exclusive OR Boolean exclusive OR	XOR XOR	
logic OR Boolean OR	OR OR	Lesser

Note: When there is a conflict between two operators at the same priority level, the first operator has priority (evaluation is done from left to right).

Example 1

In the example below the NOT is applied on the %MW3 then the result is multiplied by 25. The sum of %MW10 and %MW12 is then calculated then the logic AND from the result of the multiplication and the addition.

```
NOT %MW3 * 25 AND %MW10 + %MW12
```

Example 2

In this example multiplying%MW34 by 2 is done first then the result is used to carry out the rollover.

```
%MW34 * 2 REM 6
```

Using parentheses

Parentheses are used to modify the order in which operators are evaluated to allow, for example, an addition to be carried out before a multiplication.

You can overlap parentheses and there is no limit to the level of overlap.

Parentheses can also be used in order to prevent the program being wrongly interpreted.

Example 1

In this example, addition is done first, then multiplication:

```
(%MW10+%MW11)*%MW12
```

Example 2

This example shows that parentheses can be used to avoid any misinterpretation of the program.

```
NOT %MW2 <> %MW4 + %MW6
```

Using these operator priority rules, the interpretation is as follows :

```
((NOT %MW2) <> (%MW4 + %MW6))
```

But you may think that the operation is as follows:

```
NOT (%MW2 <> (%MW4 + %MW6))
```

Therefore parentheses serve to clarify the program.

Implicit conversions

Implicit conversions are about words and double words.
The operators that you use in arithmetic expressions, in comparisons and operator allocation carry out these implicit conversions (which are therefore not under the user's control).

For an instruction in the form: <operand 1> <operator> <operand 2>, the possible conversion cases are :

Operand 1 type	Operand 2 type	Conversion Operand 1	Conversion Operand 2	Operation in type
Word	Word	No	No	Word
Word	Double word	Double word	No	Double word
Double word	Word	No	Double word	Double word
Double word	Double word	No	No	Double word

To assign the form <left operand> := <right operand>, the left operand prescribes the type expected to carry out the operation, which means that the right operand must be converted if necessary according to the table :

Operand type left	Operand type right	Conversion right operand
Word	Word	No
Word	Double word	Word
Double word	Word	Double word
Double word	Double word	No

Note: Any operation between two adjacent values is carried out in double length.

Grafcet

9

Presentation

Subject of this chapter

This chapter describes programming rules in Grafcet.

What's in this Chapter?

This Chapter contains the following Sections:

9.1 General presentation of Grafcet

Presentation

Subject of this section

This section describes the basic Grafcet elements.

What's in this Section?

This Section contains the following Maps:

Presenting Grafcet

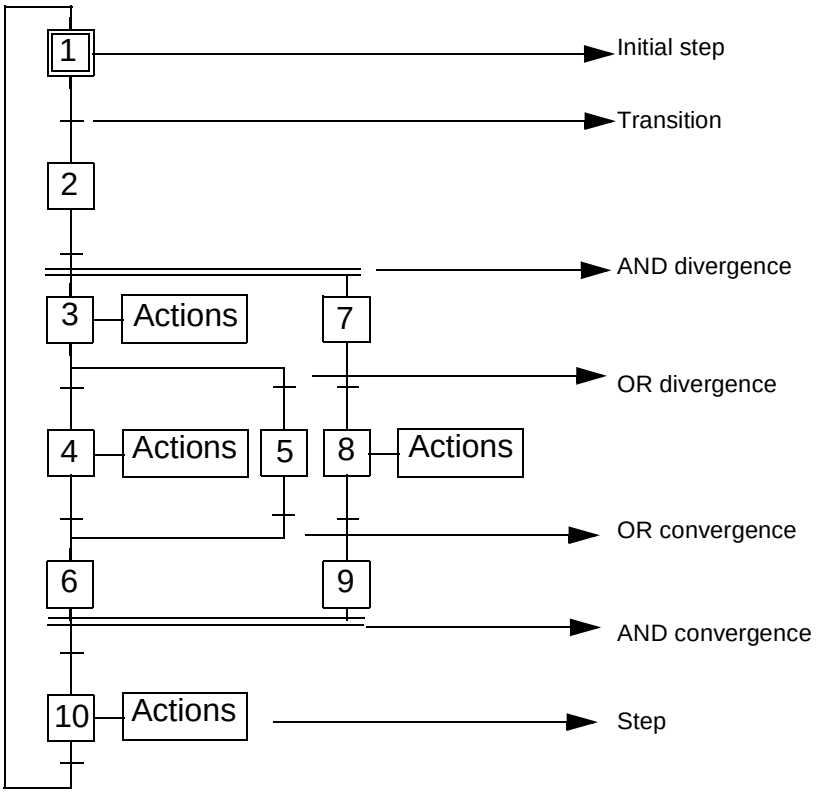
General points

The Grafcet language complies with the "Sequence function chart" (SFC) language of the IEC 1131-3 standard.

Grafcet is used to represent the operation of a sequential automatic system in a structured and graphic form.

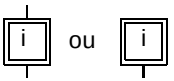
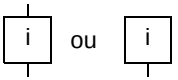
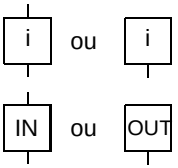
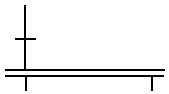
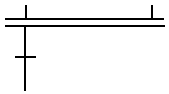
Presentation

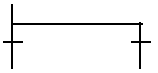
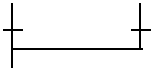
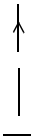
This graphic description of the sequential behavior of the automatic system and of the various situations that emanate from it is done with the help of simple graphic symbols.



Description of Grafcet graphic symbols

Description The following table describes the basic Grafcet elements.

Name	Symbol	Functions
Initial steps (		symbolize the initial active steps at the beginning of the cycle after initialization or re-start from cold.
Simple steps (		show that the automatic system is in a stable condition. The maximum number of steps (including the initial steps) can be configured from: ● 1 - 96 for a TSX 37-10, ● 1 - 128 for a TSX 37-20, ● 1 - 250 for a TSX 57. The maximum number of active steps at the same time can be configured.
Macro steps		Symbolize a macro step: a single group of steps and transitions. The maximum number of macro steps can only be configured from 0 - 63 for the TSX 57.
Stage of Macro steps		Symbolizes the stages of a macro step. The maximum number of stages for each macro step can be configured from 0 - 250 for the TSX 57. Each macro step includes an IN and OUT step.
Transitions		allow the transfer from one step to another. A transition condition associated with this condition is used to define the logic conditions necessary to cross this transition. The maximum number of transitions is 1024. It cannot be configured. The maximum number of valid transitions at the same time can be configured.
AND divergences		Transition from one step to several steps: is used to activate a maximum of 11 steps at the same time.
AND convergences		Transition of several steps to one: is used to deactivate a maximum of 11 steps at the same time.

Name	Symbol	Functions
OR divergences		Transition from one step to several steps: is used to carry out a switch to a maximum of 11 steps.
OR convergences		Transition of several steps to one: is used to end switching from a maximum of 11 steps.
Source connectors		"n" is the number of the step "it comes from" (source step).
Destination connector		"n" is the number of the step "it's going to" (target step).
Links directed towards: • top • bottom • right or left		These links are used for switching, jumping a step, restarting steps (sequence).

Note: The maximum number of steps (main graph steps + macro step steps) in the Grafcet section must not exceed 1024 on the TSX 57.

Description of specific Grafcet objects

General points

Grafcet uses bit objects associated in steps, specific system bits, word objects which show the activity time of the steps and specific system words.

Grafcet objects

The following table describes all the objects associated with Grafcet.

Name		Description
Bits associated with the steps (1 = active step)	%Xi	Status of the i step of the main Grafcet
		(i from 0 - n) (n depends on the processor)
	%XMj	Status of the j macro step (j from 0 - 63 for TSX/PMX/PCX 57)
	%Xj.i	Status of the i step of the j macro step
	%Xj.IN	Status of the input step of the j macro step
	%Xj.OUT	Status of the output step of the j macro step
System bits associated with Grafcet	%S21	Initializes Grafcet
	%S22	Grafcet resets everything to zero
	%S23	Freezes Grafcet
	%S24	Resets macro steps to 0 according to the system words %SW22 - %SW25
	%S25	Set to 1 when: • tables overflow (steps/transition), • an incorrect graph is run (destination connector on a step which does not belong to the graph).
Words associated with steps	%Xi.T	Activity time for main Grafcet step i.
	%Xj.i.T	Activity time for the i step of the j macro step
	%Xj.IN.T	Activity time for the input step of the j macro step
	%Xj.OUT.T	Activity time for the output step of the j macro step
System words associated with Grafcet	%SW20	Word which is used to inform the current cycle of the number of active steps, to be activated and deactivated.
	%SW21	Word which is used to inform the current cycle of the number of valid transitions to be validated or invalidated.
	%SW22 à %SW25	Group of 4 words which are used to indicate the macro steps to be reset to 0 when bit %S24 is set to 1.

Bits associated with steps

The bits associéated with steps %Xi, with macro steps %XMi, and macro step steps %Xj.I , %Xj.IN and %Xj.OUT have the following properties:

- They are at 1 when the steps are active.
- They can be tested in all the tasks, but can only be written in the preliminary process of the master task (pre-positioning of graphs). These tests and actions are programmed either in contact language, instruction list language or in text language.
- They can be indexed.

Activity time

The activity time words of the steps %Xi.T and macro step steps %Xj.I, %Xj.IN and %Xj.OUT have the following properties:

- They are incremented every 100 ms and have a value from 0 - 9999.
- Incrementation of the word: while the associated step is active.
- When the step is deactivated the contents are frozen.
- When the step is activated the contents are reset to zero and then incremented.
- The number of activity time words cannot be configured. One word is reserved for each step.
- These words can be indexed.

Grafcet possibilities

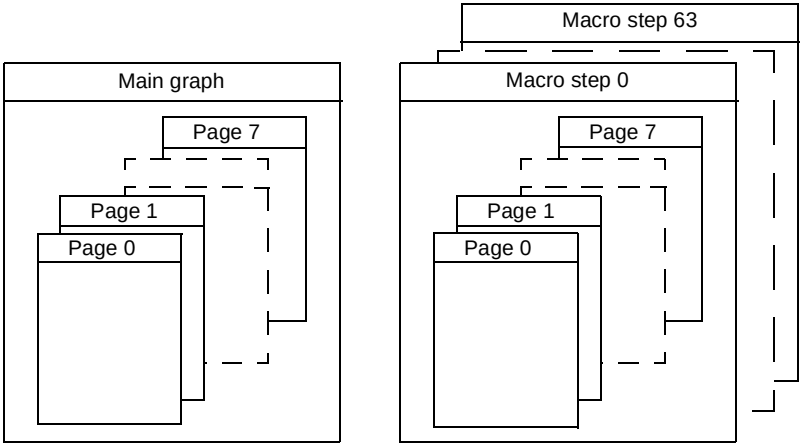
General points

The sequential process is structured in:
- 1 sub set Main graph,
- 64 sub sets: Macro steps,

These sub sets are themselves divided into 8 pages.

Illustration

The following illustration describes the general structure of a Grafcet page.



Characteristics

They depend on the processor to be programmed. They are summarized in the table below.

Number	**TSX 37 -10**		**TSX 37 -20**		**TSX 57**	
	Default settings	Maxi-mum	Default settings	Maxi-mum	Default settings	Maxi-mum
Main graph steps	96	96	128	128	128	250
Macro steps	0	0	0	0	8	64
Macro step steps	0	0	0	0	64	250
Step total	96	96	128	128	640	1024
Steps active at the same time	16	96	20	128	40	250
Transitions valid at the same time	20	192	24	256	48	400

The number of synchronous transitions (or number of AND convergences) must not exceed 64. The total number of transitions is always 1024.

9.2 Rules for constructing Grafcet

Presentation

Subject of this section

This section describes the basic rules for constructing Grafcet graphs.

What's in this Section?

This Section contains the following Maps:

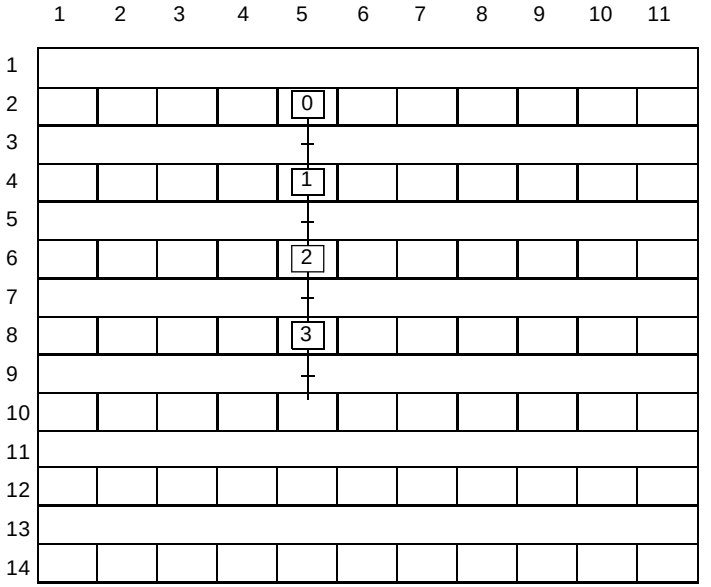
Ilustration of Grafcet

General points

The main graph and each of the macro steps are programmed on 8 pages (pages 0 - 7).
A Grafcet page is made up of 14 lines and 11 columns which define 154 cells.
It is possible to enter a graphic element in each cell.

Illustration

The drawing below illustrates the division of a Grafcet page.



Writing rules

- The first line is used to enter the source connectors.
- The last line is used to enter the destination connectors.
- The even lines (from 2 - 12) are step lines (for destination connector steps),
- The odd lines (from 3 - 13) are transition lines (for transitions and source connectors).
- Each step is located by a different number (0 - 127) in any order.
- Different graphs can be displayed on one page.

Using OR divergences and convergences

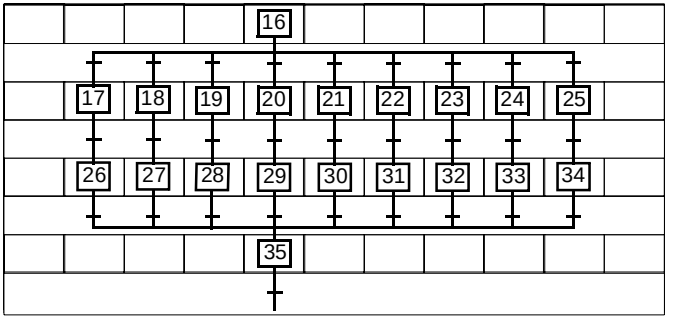
Role

An OR divergence is a switch from one step to several steps.

An OR convergence stops the switch.

Illustration

The drawing below shows an OR divergence of one step to 9 steps and an OR convergence.



Rules for use

- The number of transitions upstream of a switching end (OR convergence) or downstream of a switching (OR divergence) must not exceed 11.
- Switching can be to the left or to the right.
- Switching must general finish with switching end.
- To avoid crossing several transitions at the same time, the associated transition conditions must be exclusive.

Using AND divergences and convergences

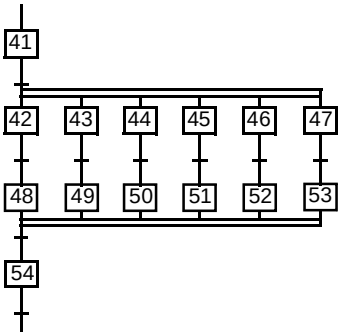
Role

An AND divergence is used to activate several steps simultaneously.

An AND convergence is used to deactivate several steps simultaneously.

Illustration

The drawing below shows an AND divergence and convergence of 6 steps.



Rules for use

- The number of steps downstream from a simultaneous activation (AND divergence) or upstream from a simultaneous deactivation (AND convergence) must not exceed 11.
- Simultaneous activation of steps must usually end with a simultaneous deactivation of steps.
- Simultaneous activation is always shown from left to right.
- Simultaneous deactivation is always shown from right to left.

Using connectors

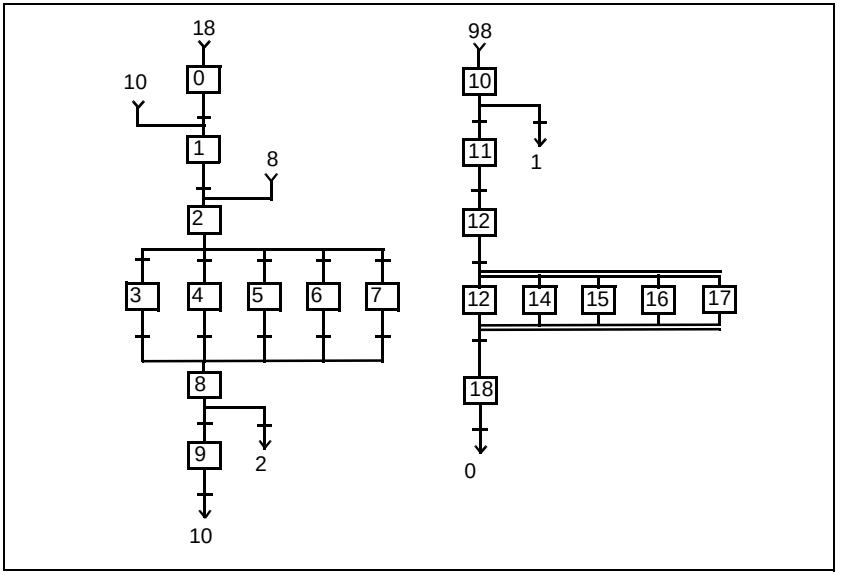
Role

Connectors ensure the continuity of a Grafcet when the direct outline of a directed link cannot be made either within a page or between two consecutive pages or not.

This continuity is maintained thanks to a destination connector which has a corresponding system source connector.

Example

The following illustration shows examples of connectors.

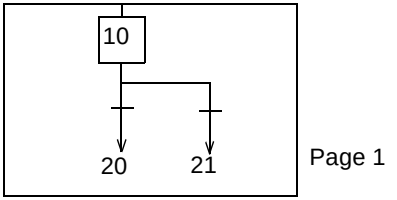
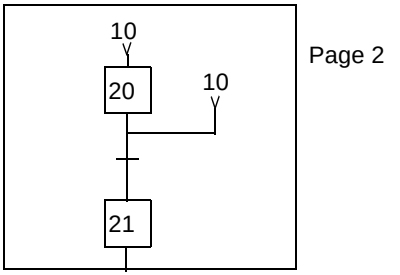
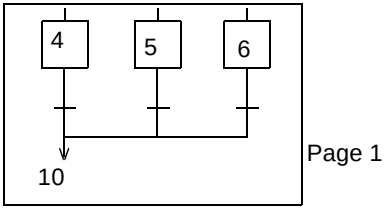
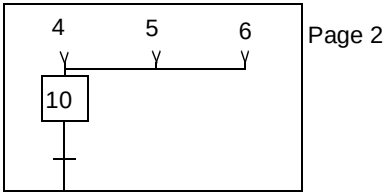


The table below explains how to use the connectors in the example.

Use	Example
Re-connecting a graph can be done using connectors.	Reconnecting step 18 to step 0.
Re-starting a sequence can be done using connectors.	Step 10 to step 1 or step 8 to step 2.
Using connectors when a branch of the graph is longer than the page.	Step 9 to step 10.

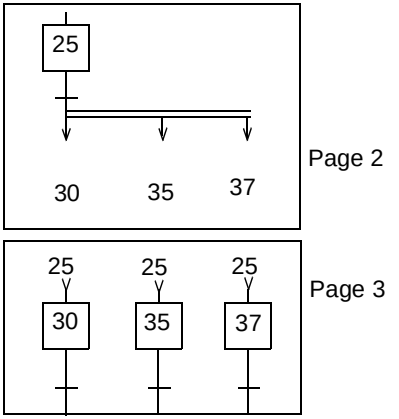
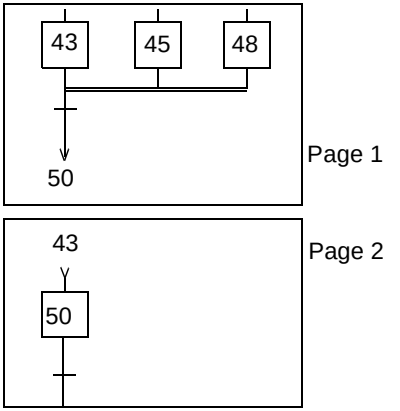
Connectors used in OR divergences and convergences

The following table gives the rules for using connectors in the case of OR divergence or convergence.

Rule	Illustration
For switching, transitions and destination connectors must be entered on the same page.	
To end switching, the source connectors must be entered on the same page as the destination step.	
For an end to switching followed by a return to destination, there must be as many source connectors as steps before the end of switching.	 

Connectors used in AND divergences and convergences

The following table gives the rules for using connectors in the case of AND divergence or convergence.

Rule	Illustration
To activate steps simultaneously, the destination connectors must be on the same page as the divergence step and transition.	
To deactivate simultaneously, the convergence steps and transition must be on the same page as the destination connector. When several steps converge onto one transition, the source connector has the number of the furthest upstream step on the left.	

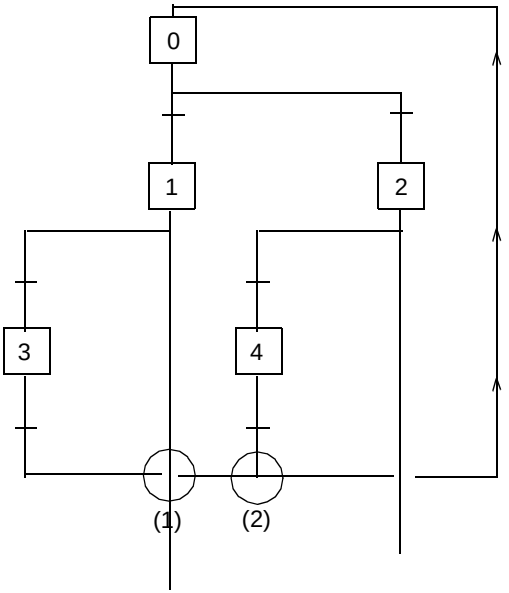
Using directed links

Role

Directed links join a step to a transition or a transition to a step. They can be vertical or horizontal.

Illustration

The following diagram shows an example of how to use a directed link.



Rules

Directed links can:

- cross (1), they are then different types,
- meet (2), they are then the same type.

It is not possible for a link to cross and for steps to be activated or deactivated at the same time.

Grafcet comments

General points

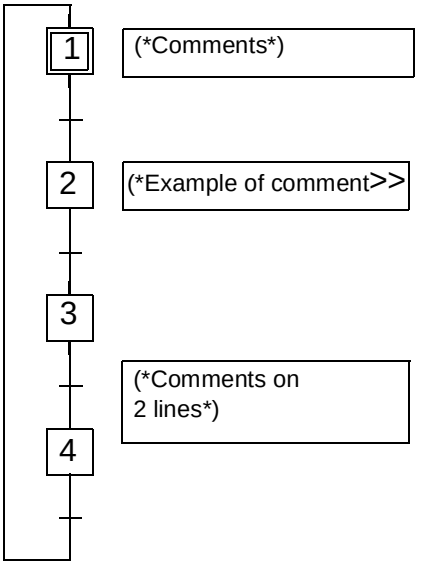
Comments can give information on Grafcet steps and transitions. They are optional.

Syntax

The text of the comments is within (* to the left and *) to the right. Its maximum size is 64 characters.

Illustration

The following illustration shows examples of comments.



Rules

- In a Grafcet page it is possible to enter a comment in any cell.
- A comment takes up two cells side by side on a maximum of two lines.
  If the display field is too small, the comment is truncated on the display but when the document is printed the comments are shown in their entirety.
- The comment entered on a Grafcet page is stored in the graphic information embedded in the PL7. In this case they take up program memory.

9.3 Programming actions and conditions

Presentation

Subject of this section

This section describes the programming rules for Grafcet actions and conditions.

What's in this Section?

This Section contains the following Maps:

Programming actions associated with steps

General points

Actions associated with steps describe the orders to be transmitted to the operative part (process to be automated) or to other automated systems.

The actions can be programmed either in contact language, instruction list language or in structured text language.
These actions are only scanned if the step with which they are associated is active.

3 types of action

The PL7 software allows three types of action:

- **actions for activation** : actions carried out once when the step with which they are associated passes from the inactive to the active state.
- **actions for deactivation** : actions carried out once when the step with which they are associated passes from the active to the inactive state.
- **continuous actions** : these actions are carried out for as long as the step with which they are associated is active.

Note: One action can include several programming elements (sequences or contact networks).

Locating actions

These actions are located in the following manner:

MAST - <Grafcet section name> - CHART (or MACROk)- PAGE n %Xi x
with
x = P1 for Activation, x = N1 Continuous, x = P0 Deactivation
n = Page number
i = Step number

Example: MAST - Paint - CHART - PAGE 0 %X1 P1 Action for activating step 1 of page 0 of the Paint section

Rules for use

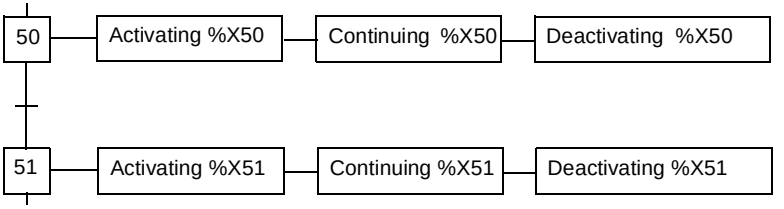
- All actions are considered stored actions, from where: an action paced to the duration of the Xn step must be reset to zero when step Xn is deactivated or when step Xn+ 1 is activated.
  An action for maintained effect on several steps is set to one when step Xn is activated and reset to zero when step Xn+m is deactivated.
- All actions can be paced to logic conditions, therefore can be conditional.
- Actions paced to indirect safeguards must be programmed in subsequent processing (See *Description of subsequent processing, p. 210*) (process carried out on each scan)

Running order for actions

For the following example, for one cycle revolution, the running order for the actions is as follows. When step 51 is activated, the actions are carried out in the following order:

1. actions for deactivating step 50,
2. actions for activating step 51,
3. continuous actions for step 51.

Example:



As soon as step 51 has been deactivated the associated continuous actions are no longer scanned.

Programming actions for activating or deactivating

Rules

These actions are carried out once when the step with which they are associated passes from the inactive to the active state.

These actions are in pulses and are carried out **in a single scanning revolution**. They are used to call up a sub-program, incrementing a counter, etc.

Example 1

In this example this action calls up a sub-program

Example 2

In this example, this action increments the word %MW10 and resets the words %MW0 and %MW25 to 0.

```
%L1:
INC %MW10;%MW0:=0;%MW25:=0;
```

Programming continuous actions

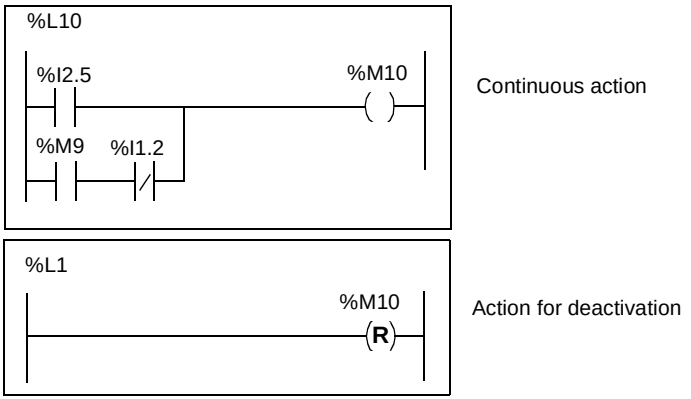
Rules

These actions are carried out for as long as the step with which they are associated is active. They can be:

- **Conditional actions** : the action is carried out if a condition is fulfilled,
- **Timed actions** : this is a special case, as the time is like a logic condition. This pacing can be done simply by testing the activity time associated with the step.

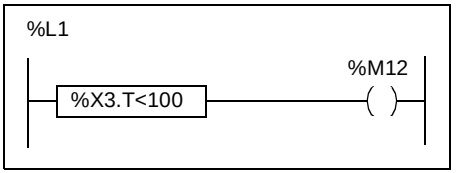
Example of a conditional action

In this example, the bit %M10 is paced to the input %12.5 or to the internal bit %M9 and to the input %I1.2.
As long as step 2 is active and these conditions apply, %M10 is set at 1.
The last state read for deactivation is stored since the associated actions are no longer scanned. It is therefore necessary to reset bit %M10 to 0 in the step deactivation action for example.
Illustration of the example.



Example of a timed action

In this example, bit %M12 is controlled as long as the activity time for step 3 is less than 10 seconds (time base : 100 ms).



These actions can also be unconditional.

Programming transition conditions associated with transitions

General points

A transition condition associated with a condition is used to define the logic conditions necessary to cross this transition.

The maximum number of transitions is 1024. It cannot be configured.
The maximum number of valid transitions at the same time can be configured.

Rules

- Associated with each transition is a transition condition which can be programmed either in ladder language, or in instruction list language or in text language.
- A transition condition is only scanned if the transition with which it is associated is valid.
- A transition condition corresponds to a contact network or an instruction list or a text expression, including a series of tests on bits and/or words.
- A transition condition which has not been programmed is always an incorrect transition condition.

Locating the transition condition

The transition conditions are located in the following manner:

`MAST - <Grafcet section name> - CHART(or MACROk) - PAGE n %X(i) --> % X(j)` with :
n = Page number
i = Upstream step number
j = Downstream step number

Example: `MAST - Paint - CHART - PAGE 0 %X(0) --> %X(1)`
Transition condition associated with the transition between step 0 and step 1 on page 0 of the Paint section graph.

Note: When steps have been activated or deactivated at the same time, the address indicated is that of the step in the column furthest to the left.

Transition condition using activity time

In some applications, actions are controlled without a check on return information (end of run, detector…). The duration of a step is governed by a time. The PL7 language enables the activity time associated with each step to be used.

Example: `! X3.T>=150`
This transition condition programmed in structured text language allows the process to remain in step 3 for 15 seconds.

Programming transition conditions in ladder

Programming rules

The transition condition associated with the condition is programmed in the form of a contact network comprising a test field and an action field.

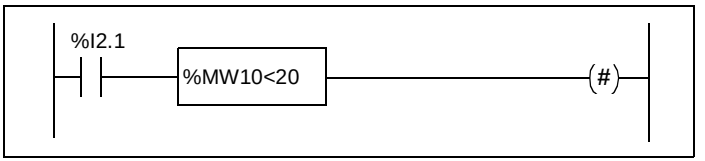
The structure of the contact network is similar to that of a network programmed in a program module.

Only the following elements can be used:

- **test graphic elements** : contacts (%Mi, %I, %Q, %TMi.D ...), comparison blocks,
- **action graphic elements** : "transition condition" coil only (the other coils are not significant in this case).

Example

This example illustrates programming for a transition condition in ladder .



Programming transition conditions in instruction list language

Programming rules

The transition condition associated with the transition is programmed in the form of an instruction list which only includes test instructions.

The instruction list for writing a transition condition differs from a normal instruction list in:

- the general structure: no label (%L).
- the instruction list:
  - no action instructions (bit, word or function block objects),
  - no jump, calling up of sub-program.

Example

This example illustrates programming for a transition condition in instruction list language.

```
!          LD   %I2.1
           AND  [%MW10<20]
```

Programming transition conditions in structured text language

Programming rules

The transition condition associated with the transition is programmed in the form of a Boolean expression or an arithmetic expression or an association of both.

The expression used for writing a transition condition differs from a programming line in text language in :

- the general structure:
  - no label (%L).
  - no action sequence, condition sequence or iterative sequence.
- the instruction list:
  - no action on bit object,
  - no jump, calling up of sub-program,
  - no transfer, no action instruction on blocks.

Example

This example illustrates programming for a transition condition in structured text language.

```
! %I2.1 AND [%MW10<20]
```

9.4 Macro steps

Presentation

Subject of this section

This section describes how to program macro steps.

What's in this Section?

This Section contains the following Maps:

Presenting macro steps

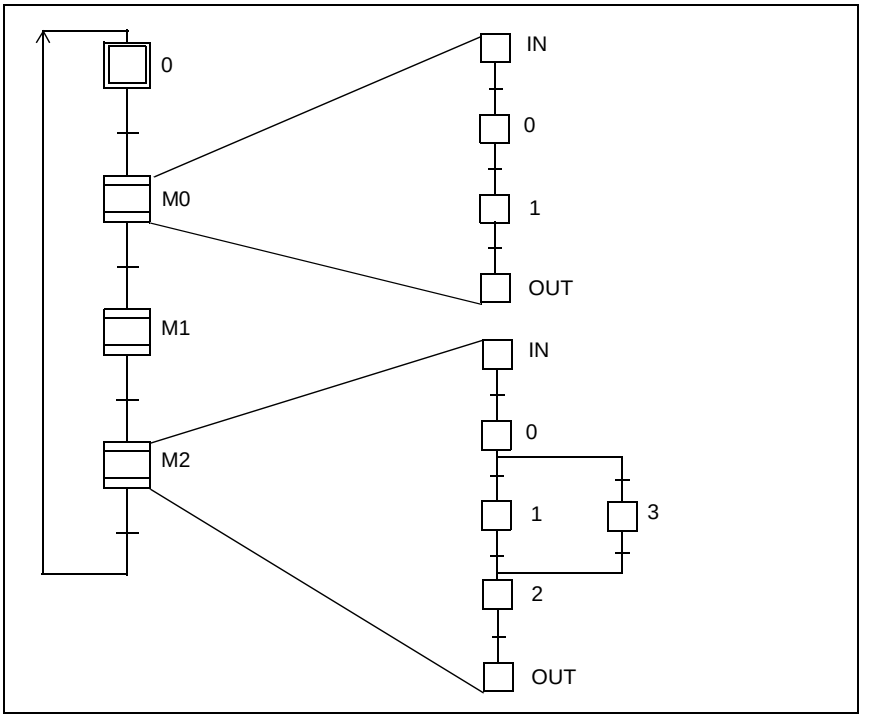
General points

A macro step is a single condensed representation of a set of steps and transitions. A macro step is inserted into a graph like a step and observes the step development rules.

Macro representation

A first level Grafcet which describes the linking of sequences is used to explain the structuring of the command part better.
Each sequence is associated with particular step symbols: the macro step.

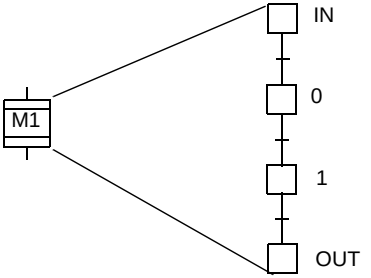
This idea of "macro representation" is used to put the analysis into a hierarchy. Each level can be completed, modified, without involving the other levels again.
Macro steps are available for TSX57 PL7s.
The following illustration shows a Grafcet made up of 3 macro steps.



Making up a macro step

Description

The graphic representation of a macro step is distinguished from a step by two horizontal lines.
The following illustration shows a macro step and its expansion.



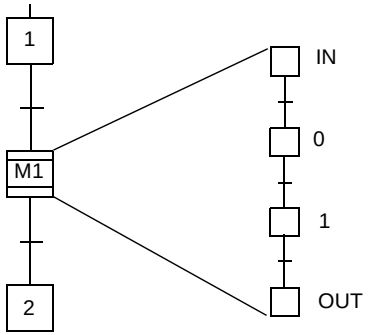
The expansion of a macro step is characterized by 2 specific steps:

- an input step observing the same rules as the other steps,
- an output step which cannot have any associated actions.

Development

When the macro step is active, development of the macro step observes the general Grafcet development rules).

Example:



Macro step M1 is activated when step 1 is active and its downstream transition condition is correct.
It is deactivated when its output step is active and the transition condition M1>2 is correct. Step 2 is then activated.

Characteristics of macro steps

General characteristics

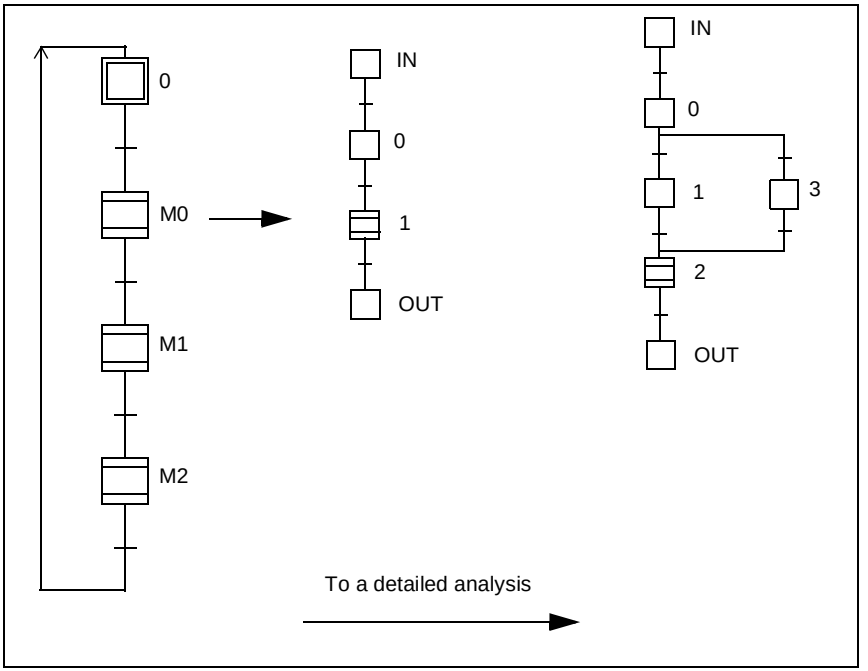
The PL7 Grafcet language allows programming of 64 macro steps from M0 to M63.

Expanding a macro step, made up of one or more sequences, can be programmed on 8 pages at the most and includes a maximum of 250 steps plus the IN step and the OUT step.

A macro step can contain one or more macro steps. This hierarchy is possible up to 64 levels.

Illustration

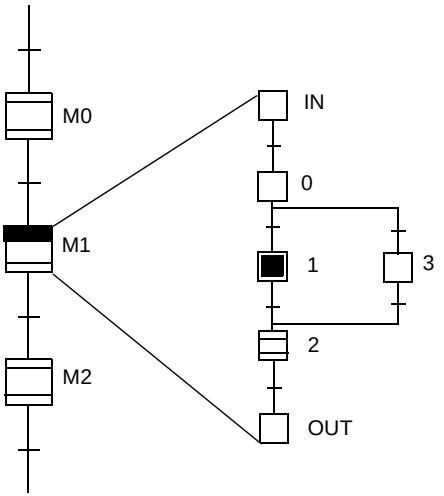
The analysis of an application can be structured in order to provide a more detailed global approach of the different operations to be performed.



Initial steps

The expansion of a macro step can contain one or more initial steps.
These initial steps are activated when the machine is switched on or when a program is initialized. The macro step is then displayed in the active state.

In the example below initial step 1 of the expansion is activated when the program is initialized. The macro step is then in an active state.



9.5 Grafcet section

Presentation

Subject of this section

This section shows how a Grafcet section is made up. It describes the use and programming rules for each process.

What's in this Section?

This Section contains the following Maps:

Structure of a Grafcet section

How a section is made up

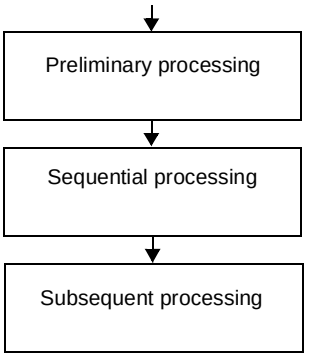
A section of program written in Grafcet is made up of three consecutive processes:

- the preliminary process,
- the sequential process,
- the subsequent process.

The Grafcet section is programmed in the MAST task.

Illustration

The following drawing shows the order the processes are scanned.



Role of the processes

The following table describes the role of each of the processes and the language with which they can be programmed.

Processing	Role	Language
Preliminary	This is used to process: ● initializations on a restart after a power or mechanical failure, ● initializations on a restart after a power or mechanical failure, ● the input logic.	Contact, instruction list or text language
Sequential	This is used to process the sequential framework of the application and gives access to transition conditions and actions directly associated with the steps.	Grafcet
Subsequent	This is used to process: ● the output logic, ● monitoring and indirect safeguards specific to outputs.	Contact, instruction list or text language

Note: The macro steps are carried out in their scanning order in sequential processing.

Description of preliminary processing

General points

Entered in contact language, instruction list language or structured text language, preliminary processing is scanned in its entirety from top to bottom.

It is run before the sequential and subsequent processes and is used to process all the events that have an effect on the following:

- managing power re-starts and re-initializations,
- resetting to zero or pre-positioning graphs.

Therefore it is only in the preliminary process that it can affect the bits associated with the steps (step bits %Xi or %Xi.j set to 0 or to 1 by the instructions SET and RESET).

System bits

Pre-positioning, initialization, freezing operations are done using system bits %S21 to %S24.

The system bits associated with Grafcet are classified numerically in order of priority (%S21 to %S24), and so when several of them are simultaneously set to 1 in the preliminary processing, they are dealt with one by one in ascending order (only one is used per scanning revolution).

This bits are used at the beginning of sequential processing.

Processing cold re-starts

On a new application or when a system context has been lost, the system restarts from cold.

The bit %S21 is set to 1 by the system before calling up the preliminary processing and Grafcet is positioned on the initial steps.

If you want a particular process for the application when it is re-starting from cold, it is possible to test % S0 which remains at 1 during the first cycle of the master task (MAST).

Processing cold re-starts

Following a power failure without changing the application, the system re-starts from cold. It starts again from where it was before the power failure.

If you want a particular process for the application in the case of a cold re-start, you can test %S1 in the preliminary process and call up the corresponding program.

Pre-setting the Grafcet

Role

Pre-setting the Grafcet can be used when switching from a normal running operation to a specific running operation or when an incident appears (e.g.: fault causing an imperfect run).

This operation takes place during the normal run of an application cycle. Therefore it must be done carefully.

Pre-setting the Grafcet

The setting can have an effect on all or part of the sequential process:

- by using the instructions SET, RESET,
- by a general reset to zero (%S22) then, in the following cycle, setting the steps to 1.

Note: Where a step is reset to zero, the deactivation actions for this are not carried out.

Example

In this example setting bit %M20 to 1 causes steps %X12 to be pre-set to 1, steps %X10 and %X11 to 0.

Initializing the Grafcet

Role

Initializing the Grafcet is done by the system bit %S21.
Normally set at state 0, setting %S21 to 1 causes:

- active steps to deactivate,
- initial steps to activate.

Initializing the Grafcet

The following table gives the different possibilities for setting to the system bit %S21 to 1 and 0.

Set to 1	Reset to 0
• By setting %S0 to 1 • By the user program • By the terminal (in debugging or animation table)	• By the system at the beginning of the process • By the user program • By the terminal (in debugging or animation table)

Rules for use

When it is managed by the user program, %S21 must be set to 0 or 1 in the preliminary process.

Resetting Grafcet to zero

Role

The system bit %S22 resets Grafcet to 0.

Normally set at 0, setting %S22 to 1 causes active steps in the whole of the sequential process to deactivate.

Note: The RESET_XIT function used to reinitialize via the program the step activity time of all the steps of the sequential processing. (See (See Reference Manual, Volume 2)).

Resetting Grafcet to zero

The following table gives the different possibilities for setting to the system bit %S22 to 1 and 0.

Set to 1	Reset to 0
• By the user program • By the terminal (in debugging or animation table)	• By the system at the end of the sequential process

Rules for use

- this bit must be written to 1 in the preliminary process,
- resetting %S22 to 0 is managed by the system. It is therefore pointless to have the program or terminal reset to 0.

To start up the sequential process in a given situation, you must carry out an initialization procedure or pre-set the Grafcet according to the application.

Freezing Grafcet

Role

Freezing the Grafcet is done by the system bit %S23.

Normally set at 0, setting %S23 to 1 causes the Grafcet state to be maintained. Whatever the value of the transition conditions downstream of the active steps, the Grafcets do not develop. The freeze is maintained as long as bit %S23 is set to 1.

Freezing Grafcet

The following table gives the different possibilities for setting to the system bit %S23 to 1 and 0.

Set to 1	Reset to 0
• By the user program • By the terminal (in debugging or animation table)	• By the user program • By the terminal (in debugging or animation table)

Rules for use

- managed by the user program, this bit must be set to 1 or 0 in the preliminary process,
- bit %S23 associated with bits %S21 and %S22 are used to freeze the sequential treatment at the initial state or at 0. In the same way the Grafcet can be set and then frozen by %S23.

Resetting macro steps to zero

Role

Freezing the Grafcet is done by the system bit %S24.

Normally set to 0, setting %S24 to 1 causes the selected macro steps to be set to zero in a table of 4 system words (%S22 - %S25).

Note: The RESET_XIT function used to reinitialize via the program the step activity time of the macro step. (See (See Reference Manual, Volume 2)).

Resetting macro steps to zero

The following table gives the different possibilities for setting the system bit %S24 to 1 and 0.

Set to 1	Reset to 0
• By the user program	• By the system at the beginning of the process

Rules for use

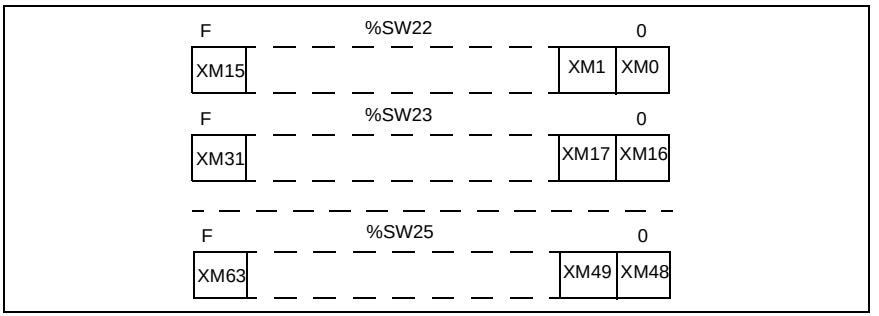
- this bit must be written to 1 only in the preliminary process,
- resetting %S24 to 0 is managed by the system. It is therefore pointless to have the program or terminal reset to 0.

Table of words %SW22 to %SW25

A macro step corresponds to each bit in this table. Using it is as follows:

- loading the table with the words %SW22 - %SW25 (bit to be set to 1 when the corresponding macro step must not be set to zero),
- validation by %S24.

The following illustration show coding for words %SW22 - %SW25.



Example:

```
! IF %I4.2 AND %T3.D THEN
%SW22:=16#AF8F;
%SW23:=16#F3FF;
%SW24:=16#FFEF;
%SW25:=16#FFFF;
SET %S24
```

These four words are initialized at 16#FFFF if %S21 = 1.

Running sequential processing

General points

This process is used to program the sequential framework of the application. Sequential processing includes:

- the main graph organized into 8 pages,
- up to 64 macro steps each from 8 pages.

In the main graph, several unconnected Grafcets can be programmed and run simultaneously.

The Grafcet is made up in 3 large phases.

Phase 1

The following table describes the operations carried out in the first phase.

Phase	Description
1	Evaluating the transition conditions of validated transitions.
2	Request to deactivate the associated upstream steps.
3	Request to activate the downstream steps concerned

Phase 2

Phase 2 corresponds to developing the Grafcet situation according to the transitions crossed:

Phase	Description
1	Deactivating the upstream steps of the transitions crossed.
2	Activating the downstream steps of the transitions crossed.
3	Invalidating the transitions crossed.
4	Validating the transitions downstream of the new activated steps. **Result:** The system updates two tables dedicated respectively to the activity of the steps and the validity of the transitions: • the step activity table stores the active steps, the steps to be activated and the steps to be deactivated for the current cycle, • the transition validity table stores the transitions downstream of the steps affected by the previous table for the current cycle

Phase 3

The actions associated with active steps are carried out in the following order:

Phase	Description
1	Actions to deactivate the steps to be deactivated.
2	Actions to activate the steps to be activated.
3	Actions to continue active steps.

Exceeding capacity

The number of elements in a steps activity table and transition validity table can be configured.
Exceeding the capacity of one or the other leads to:

- the PL7 switching to STOP (stopping the application running),
- the system bit %S26 switching to 1 (exceeding the capacity of one of the two tables),
- the ERR light on the PL7 flashing.

The system provides the user with two system words:

- %SW20: a word which is used to inform the current cycle of the number of active steps, to be activated and deactivated.
- %SW21: a word which is used to inform the current cycle of the number of valid transitions to be validated or invalidated.

Diagnostics

In the case of a blocking fault, the system words % SW125 - %SW127 are used to determine the nature of the fault.

%SW125	**%SW126**	**%SW127**	
DEF7	0	= 0	Exceeding the step table (steps/transitions)
DEF7	= 0	0	Exceeding the transition table
DEFE	Step no.	Macro step no. or 64 for the main graph	Running the wrong graph (transition problem with unresolved destination connector).

Description of subsequent processing

General points

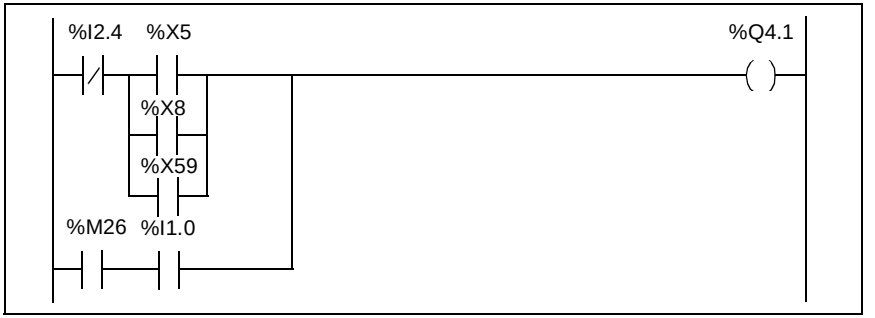
Entered in contact language, instruction list language or structured text language, subsequent processing is scanned from top to bottom.
This process is the last one run before activating the outputs and is used to program the output logic.

Actions associated with Grafcet

Subsequent processing is used to complete the set points issued by the sequential process by integrating the running and stopping methods and the indirect safeguards specific to the action with the output equation.
It is also used to process an output activated several times in the sequential process.

Generally it is recommended that actions which have a direct effect on the process should be programmed in the subsequent process.

Example:



- 12.4 = indirect safeguard specific to controlling output %Q4.1.
- %M26 = internal bit resulting from the input logic dealing with the running and stopping modes.
- %I1.0 = push button.

Output %Q4.1 is activated by steps 5, 8 and 59 of the sequential process.

Actions independent of Grafcet

Subsequent processing is also used to program outputs that are independent of sequential processing.

Checking the running of Grafcet

It may turn out to be necessary to check the proper running of Grafcet by testing the activity time of certain steps. Testing this time is done by comparing either with a minimum value or with a maximum value determined by the user.

Using the default is left up to the user (indications, special operating procedure, editing messages).

Example:

```
! IF (%X2.T > 100 AND %X2) THEN SET %Q4.0 ;END_IF ;
```

DFB function blocks

10

Presentation

Subject of this chapter

This chapter describes how to program DFB user function blocks.

What's in this Chapter?

This Chapter contains the following Maps:

Presenting DFB function blocks

Role

Pro PL7 software offers the user the possibility of creating his own function blocks corresponding to the specific needs of his applications.

These user function blocks are designed for structuring an application. They will be used when a programming sequence is repeated several times within an application, or to freeze standard programming (e.g.: command algorithm for a motor including taking local safeguards into account).

They can be transmitted to all the programmers and used in the same application or in any other application (export/import function).

Examples of use

Using a DFV function block in an application allows you to:
- simplify the program design and entries,
- increase the readability of the program,
- enable debugging (all the variables changed by the DFB type are identified on its interface),
- reduce the amount of code generated (the code corresponding to the DFB is only changed once, however many calls there are on the DFB in the program).

Comparison with the sub-program

In relation to the sub-program they enable:
- the parameters of the process to be set more easily,
- the internal variables belonging to the DFB and therefore independent of the application to be used,
- testing independent of the application.

They offer a graphic display of the block in ladder language, making programming and debugging easier.
Also, DFB function blocks use residual data.

Areas of use

The table below describes the DFB areas of application.

Function	Area
PL7s the DFBs can be used with.	Premium
DFB creation software	PL7 Pro
Software the DFBs can be used with.	Pro PL7 or Junior PL7
Programming language used to create DFB code.	structured text language and ladder
Programming languages the DFBs can be used with.	ladder, structured text and instruction list

How to set up a DFB function block

Procedure

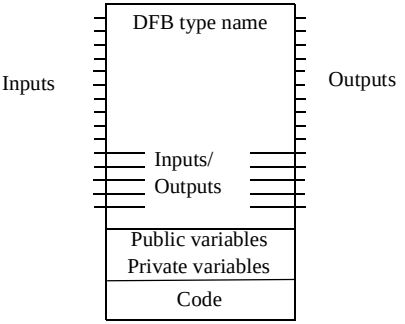
Setting up a DFB function block is done in 3 main steps:

Step	Action
1	Designing the DFB model (called: DFB Type).
2	Creating an image of this block called instance for using each time in the application.
3	Using the instance in the PL7 program.

Designing the DFB type

Consists of defining and coding all the elements that make up the DFB model, using the DFB editor.

The following illustration shows how a DFB model is made up.



A DFB type function block is made up:

- of a name,
- parameters:
  - inputs,
  - outputs,
  - inputs/outputs,
- variables:
  - public variables,
  - private variables,
- code in structured text language or ladder,
- comments,
- a descriptive form.

Creating a DFB instance

Once the DFB type is designed the user defines a DFB instance using a variables editor or when calling up the function in the program editor.

Using the DFBs

This block instance is used next as a standard function block in ladder language or as an elementary function in structured text language or instruction list language.

It can be programmed in the various tasks (except in event tasks) and sections of the application.

Defining DFB type function block objects

General characteristics of DFB objects

These objects are internal DFB data. They can be purely symbolic (no addressing under address form).

DFBs use 2 types of object:
- parameters
- variables

Syntax

For each parameter or variable used, the designer of the DFB type function block defines:
- a name with a maximum of 8 characters (non-accented letters, figures, the character "_", are allowed. The first character must be a letter; key words and symbols are not allowed),
- an object type (see table below),
- an optional comment with up to 80 characters,
- an initial value (except for Input/Output parameters).

Types of objects

The table below describes the list of different object types possible when declaring the parameters and variables for the DFB type.

Action on...	Type	Name	Examples
Bits	BOOL	Boolean	The BOOL type does not manage the edges. If edge management is not necessary in the process it is preferable to use the BOOL type. Example of a BOOL type object in PL7 language: %MWi:Xj which does not manage the edges but which takes up less memory than the EBOOL type.
	EBOOL	Extended Boolean	The EBOOL type manages the edges. Therefore it is possible to run RE and FE edge instructions on this type of parameter or variable. if you wish to associate an EBOOL type with an input/output parameter when it is being used, it must be an EBOOL type in the DFB. Example of a EBOOL type object in PL7 language: %Mi,%Ixy.i,%Qxy.i.
Words	WORD	All 16 bits	Example of a WORD type object in PL7 language: %MWi, %KWi,
	DWORD	All 32 bits	Example of a DWORD type object in PL7 language: %MDi, %KDi,
	REAL	Real	Example of a REAL type object in PL7 language: %MFi, %KFi

Action on…	Type	Name	Examples
Tables	AR_X	Bit table	Example of an AR_X type object in PL7 language: %Mi:L, %Ix.i:L
	AR_W	Table of all 16 bits	Example of an AR_W type object in PL7 language: %MWi:L, %KWi:L
	AR_D	Table of all 32 bits	Example of an AR_D type object in PL7 language: %MDi:L, %KDi:L
	AR_R	Table of reals	Example of an AR_R type object in PL7 language: %MFi:L, %KFi:L
	STRING	Character string	Example of a STRING type object in PL7 language: %MBi, %KBi

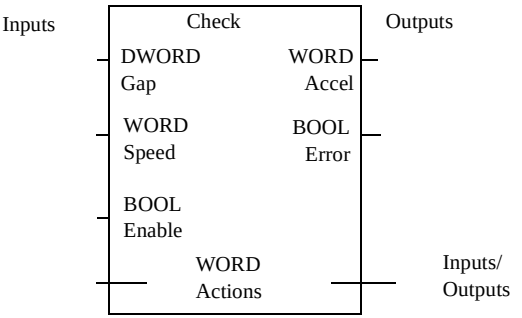
Note:

- Tables: the length of a table must be mentioned for outputs and public and private variables, on the other hand it is not necessary to define them for input parameters and input/output parameters.
- The initial values can be defined for inputs (if they are not table type), for outputs and for public and private variables.

Definition of DFB parameters

Illustration

The following illustration shows examples of parameters



Description of parameters

The table below describes the role of each type of parameter.

Parameter	Maximum number	Role
Inputs	15 (1)	This is data to be supplied to the DFB by the application program. These read only parameters cannot be modified in the DFB code.
Outputs	15 (2)	This is data calculated by the DFB to be sent to the application program.
Inputs/outputs	15	These are input parameters which can be modified in the DFB code.

Key:
(1) Number of inputs + Number of inputs/outputs less than or equal to 15
(2) Number of outputs + Number of inputs/outputs less than or equal to 15

Note:
- All the DFB block must have at least one Boolean input.
- It is only possible to modify a DFB interface (public variables or parameters) if it is not instanced and used in the application.

Definition of DFB variables

Description of variables

The table below describes the role of each type of variable.

Variable	Maximum number	Role
Public	100	Internal variables used in the process which can be accessed by the user for adjustment or by the application program outside the DFB code (for a DFB instance public variable, see below: Accessing public variables.
Private	100	Internal variables in function block code. These variables are calculated and used right inside the DFB but do not have any link with the outside of the DFB. These variables are useful for programming the block but are of no interest for the user of the block (e.g.: intermediate variable for sending back one combination expression to another, result of an intermediate calculation…).

Note: It is only possible to modify a DFB interface (public variables or parameters) if it is not instanced and used in the application.

Accessing public variables.

Only output parameters and public variables can be accessed as objects in the application program and outside the function block body.
Their syntax is as follows :

Name_DFB.Name_parameter

Or **Name_DFB** is the name given to the DFB instance used (32 characters maximum)

and **Name_parameter** is the name given to the output parameter or the public variable (8 characters maximum).

Example: `Gap.check` for the gap `output` of the DFB instance named `Check`.

Saving and restoring public variables

Public variables, modified by the program or by adjustment, can be saved in the location and sets initialization values (defined in the DFB instances) by setting system bit $S94 to 1.
Replacement only takes place if authorization for it has been given at the level of each DFB type variable.

These saved values are re-applied by setting system bit %S95 to 1 or by re-initializing the PL7.

It is possible to disable the «**Save/Restore**» function globally for all DFBest possible function blocks (dialogue box**Properties of DFB types**).

Coding rules for DFB types

General points

The code defines the processing the DFB block must carry out depending on the parameters declared.
The DFB function block code is programmed in text or ladder language.
As far as text language is concerned, the DFB is made up of a single text sequence of unlimited length.

Programming rules

All language instructions and advanced functions are permitted except for:

- calling up standard function blocks,
- calling up other DFB function blocks,
- connecting to a JUMP label,
- calling up a sub-program,
- the HALT instruction,
- instructions using input/output module variables (e.g.: READ_STS, SMOVE...).

The code uses DFB parameters and variables defined by the user.

The DFB block function code cannot use either input/output objects (%I,%Q...), or global application objects (%MW,%KW...) except for the system bits and words %S and %SW.

Note: it is not possible to use a label

Specific functions

The table below describes functions that have been specifically adapted to be used in the code.

Functions	Role
FTON, FTOF, FTP, FPULSOR	These timing functions are meant to be used instead of standard timing function blocks.
LW, HW, COCATW	These instructions are used to manipulate words and double words.
LENGTH_ARW, LENGTH_ARD, LENGTH_ARR	These instructions are used to calculate the length of tables.

Example of the code

The following program gives an example of text code

```
CHR_200:=CHR_100;
CHR_114:=CHR_104;
CHR_116:=CHR_106;
RESET START;
(*CHR_100 is incremented 80 times*)
FOR CHR_102:=1 TO 80 DO
            INC CHR_100;
            WHILE((CHR_104-CHR_114)<100)DO
            IF(CHR_104>400) THEN
EXIT;
            END_IF;
            INC CHR_104;
            REPEAT
            IF(CHR_106>300) THEN
EXIT;
            END_IF;
            INC CHR_106;
            UNTIL ((CHR_100-CHR_116)>100)
            END_REPEAT;
            END_WHILE;
            (*Loop continues as far as CHR_106)
            IF (CHR_106=CHR_116)
            THEN EXIT;
            ELSE
               CHR_114:=CHR_104;
               CHR_116:=CHR_106;
            END_IF;
            INC CHR_200;
END_FOR;
```

Creating DFB instances

General points

A DFB instance is a copy of the DFB type:
- it uses the DFB type code (there is no duplication of the code),
- it creates a field for specific data for this instance which is a copy of the parameters and variables of the DFB type. This field is in the given space in the application.

Each DFB instance is located by a name of no more than 32 characters defined by the user.
The characters allowed are identical to those allowed for symbols, i.e. the following are allowed:
- non-accented letters,
- figures,
- the character "_".

The first character must be a letter; key words and symbols are not allowed.

Rules

It is possible to create as many instances as necessary (only limited by the PL7 memory size) from the same type of DFB.

The initial values of the public variables defined for DFB type function blocks can be modified for each instance.

Rules for using DFBs in a program

General points

DFB instances can be used in all languages (ladder, text and instruction list) and in all parts of the application: sections, sub-programs, Grafcet modules (except in event tasks).

General rules for use

The following rules must be observed whatever language used:

- all the table type input parameters as well as input/output parameters must be entered.
- input parameters which have not been hardwired keep the value of the previous call or the initialization value if the block has never been called before this input has been made or hardwired.
- all the objects assigned to input, output and input/output parameters must be of the same type as those defined when the DFB type was set up (e.g.: if the WORD type is defined for the "speed" input parameter, the double words %MDi, %KDi are not allowed to be allocated to it).
  The only exceptions are the BOOL and EBOOL types for input or output parameters (not for input/output parameters) which can be mixed.
  Example : the "Validation" input parameter can be defined as BOOL and can be associated with an internal bit %Mi which is an EBOOL type. On the other hand, in the internal code of the DFB type, the input parameter will have BOOL type properties. It cannot manage edges.

Allocating the parameters

The following table summarizes the different parameter allocation possibilities in the various programming languages:

Parameter	Type	Parameter allocation	Allocation
Inputs	Boolean	Hardwired (1)	optional (2)
	Numeric	Object or expression	optional
	Table	Object	compulsory
Inputs/ Outputs	Boolean	Object	compulsory
	Numeric	Object	compulsory
	Table	Object	compulsory
Outputs	Boolean	Hardwired (1)	optional
	Numeric	Object	optional
	Table	Object	optional

(1) hardwired in ladder, or object in Boolean or text language
(2) in ladder, any DFB block must have at least one hardwired Boolean (binary) input.

Using a DFB in a ladder language program

Principle

There are two ways of calling up a DFB function block:

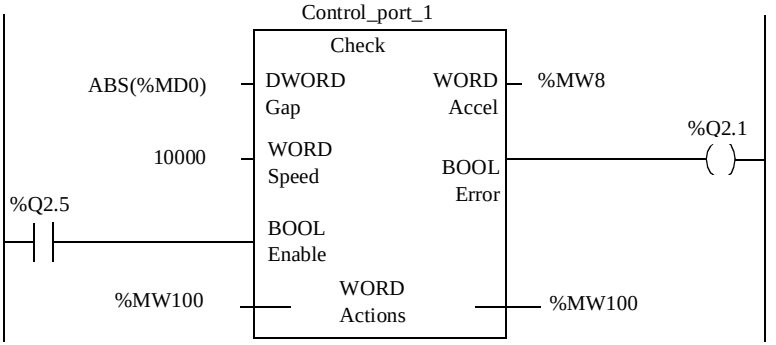
- a text call in an operation block, the syntax and the constraints on the parameters are the same as those in text language.
- a graphic call, see the example below.

Graphic DFB function blocks have inputs/outputs which are assigned directly by objects or expressions. These objects or expressions take up one cell in the graphic network.

2 DFB function blocks connected in series must be separated by at least 2 columns.

Example

The following illustration shows a simple example of how to program a DFB.



The table below locates the different elements of the DFB.

Address	Role
1	Name of the DFB
2	Name of the DFB type
3	Effective parameter for the first input
4	Input parameters (name and type)
5	Output parameters (name and type)
6	Input/output parameters (name and type)

Note:

- A DFB function block must have at least one hardwired Boolean input.
- The numerical inputs, outputs or inputs/outputs of the block are not hardwired. The objects mentioned in the cell opposite the contact are associated with these contacts.

Using a DFB in a program in instruction list or text language

General points

Calling up the DFB function block constitutes an action which can be put in a sequence just like any other action in the language.

General syntax

The DFB programming syntax is as follows:

```
DFB_Name
(E1,...,En,ES1,...,ESn,S1,...,Sn);
```

The following table describes the role of the instruction parameters.

Parameters	Role
E1, ..., En	Expressions (except for BOOL/EBOOL type objects), objects or immediate values using effective parameters for input parameters.
ES1, ..., ESn	Effective parameters corresponding to inputs/outputs. They are always read/write language objects.
S1, ..., Sn	Effective parameters corresponding to outputs. They are always read/write language objects.

Text syntax

Instructions in text language have the following syntax:
Name_DFB (E1,...,En,ES1,...,ESn,S1,...,Sn);

Example: Cpt_bolts(%I2.0,%MD10,%I2.1,%Q1.0);

Instruction list syntax

Instructions in instruction list language have the following syntax:
[Name_DFB (E1,...,En,ES1,...,ESn,S1,...,Sn)]

Example: [Cpt_bolts(%I2.0,%MD10,%I2.1,%Q1.0)]

Running a DFB instance

Operation

Running a DFB instance is done in the following order:

Step	Action
1	Loading the input and input/output parameters using the actual parameters. Any input left free takes on the initialization value defined in the DFB type when being initialized or starting from cold, then the current value of the parameter.
2	Going through input parameter values (except for table type).
3	Going through input/output parameter addresses.
4	Running the text code.
5	Writing the output parameters.

Debugging tools

PL7 software offers several debugging tools for the PL7 program and the DFBs:

- animation table: all parameters and public variables are displayed and animated in real time. It is possible to modify and force required objects,
- break point, step by step and diagnostic program,
- operating screens: for unit debugging.

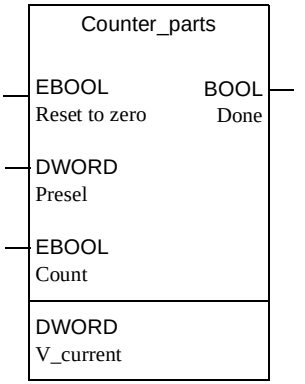
Example of how to program DFB function blocks

General points

This example is given in a technical capacity. The DFB to be programmed is a counter.

DFB type characteristics

The counter is made up from the following DFB type:



The table below describes the characteristics of the DFB type to be programmed.

Characteristics	Values
Name	**Counter_parts**
Inputs	• **Reset to zero**: resetting the counter to zero • **Presel**: value of counter preselection • **Count**: input count
Outputs	**Done**: output value of the preselection
Public variable	**V_cour**: Current variable incremented by the input **Count**

Counter operation

The following table describes the functions that the counter must have.

Phase	Description
1	This block counts the rising edges on the input **Count**.
2	The result is put in the variable **V_curr**, this value is reset to zero by a rising front on the input **Reset to zero**.
3	Counting is done up to the preselected value. When this value is reached the output**Done** is set to 1, it is reset to 0 on the rising edge on the input **Reset to zero**.

DFB code

Programming a DFB type code is given below.

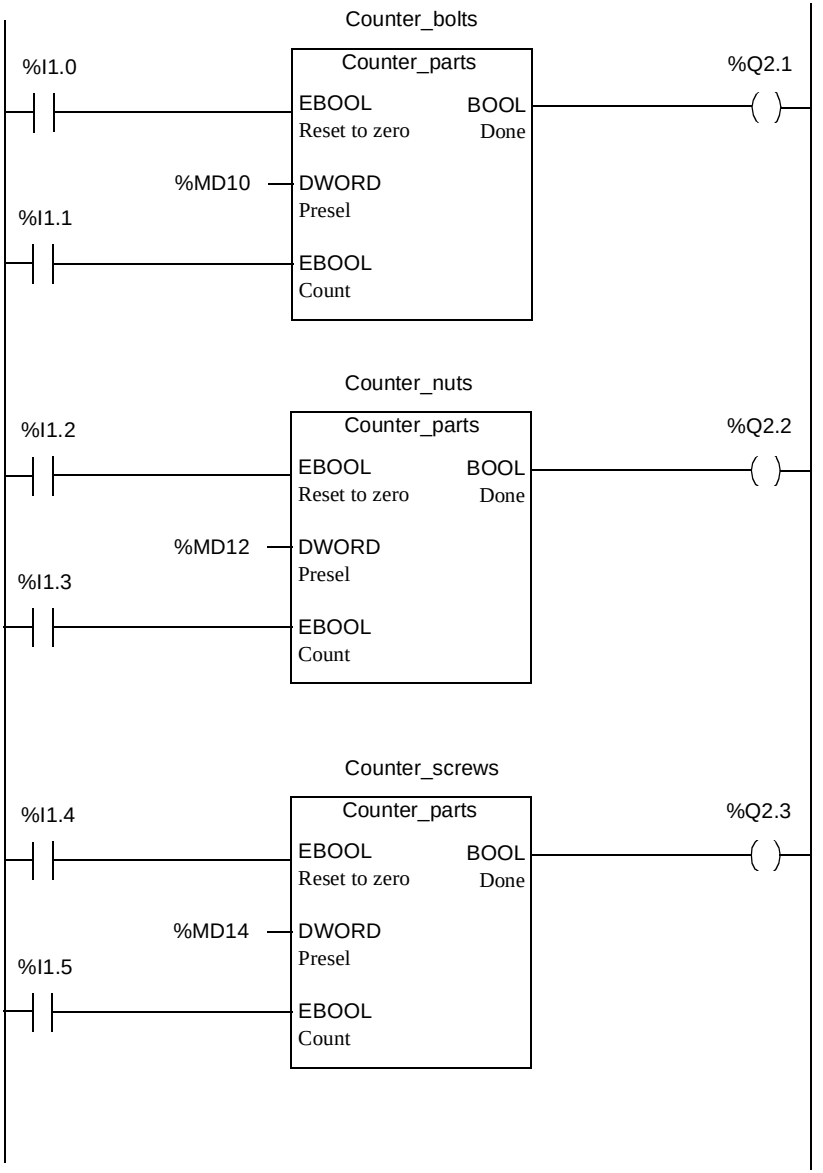
```
!(*Programming DFB counter_parts*)
IF RE Reset to zero THEN
      V_curr:=0;
END_IF;
IF RE Count THEN
      V_curr:=V_curr+1;
END_IF;
IF(V_curr>=Presel) THEN
      SET Done;
ELSE
      RESET Done;
END_IF;
```

Example of use

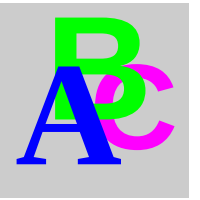
In this example, the DFB type created is used 3 times (3 DFB instances) for counting 3 types of parts.

When the number of parts programmed (in words %MD10, %MD12, and %MD14) is reached, the counter output stops the supply system of the corresponding parts.

The following program uses 3 instances of DFB type Counter_parts: Counter_bolts, Counter_nuts, Counter_screws.



Index



G

I

L

M

O

P

R

S

T

W