

TLX DR PL7 13E Reference manual	Description of PL7 software	A
Detailed description of instructions and functions		B
TLX DM PL7 J13E Operating modes	General	C
Configuration and programming		D
Debugging, Adjustment, Documentation and Appendices		E
TLX DS 57 PL7 13E Application-specific functions Volume 1/2	Common features of application-specific functions	F
Discrete I/O		G
Counting		H
Analog		I
PID control		J
TLX DS 57 PL7 13E Application-specific functions Volume 2/2	Man-machine interface	K
<div style="text-align: right;">Reserved</div>		L
Axis control		M
Stepper control		N

Communication : see TLX DS COM PL713E manual

Section	Page
1 General	1/1
1.1 Presentation of PL7 software	1/1
1.1-1 Presentation	1/1
1.1-2 Single task structure	1/3
1.1-3 Multitask structure	1/3
1.1-4 Symbolic programming	1/4
1.1-5 PL7 instructions	1/5
1.2 Addressable objects	1/6
1.2-1 Definition of main Boolean objects	1/6
1.2-2 Addressing TSX 37 I/O module objects	1/7
1.2-3 Addressing TSX 57 I/O module objects	1/9
1.2-4 Addressing words	1/11
1.2-5 Function block objects	1/15
1.2-6 Structured objects	1/16
1.2-7 Grafcet objects	1/18
1.2-8 Symbolization	1/19
1.3 User memory	1/20
1.3-1 General	1/20
1.3-2 Saving / retrieving internal words %MWi	1/22
1.3-3 Bit memory	1/23
1.3-4 Word memory	1/25
1.3-5 TSX 37-10 PLCs	1/26
1.3-6 TSX 37-21/22 PLCs	1/27
1.3-7 TSX 57-10 PLCs	1/28
1.3-8 TSX 57-20 PLCs	1/29
1.4 Operating modes	1/30
1.4-1 Processing on outage and power return	1/30
1.4-2 Warm restart processing	1/31
1.4-3 Cold restart processing	1/32
1.5 Single task software structure	1/32
1.5-1 Presentation of the master task	1/33
1.5-2 Cyclic execution	1/34
1.5-3 Periodic execution	1/35

Section	Page
1.5-4 Monitoring scan time	1/37
1.6 Multitask software structure	1/38
1.6-1 Description	1/38
1.6-2 Master task	1/39
1.6-3 Fast task	1/39
1.6-4 Assigning I/O channels to the master and fast tasks	1/40
1.6-5 Event-triggered tasks	1/41
2 Ladder language	2/1
2.1 Presentation of Ladder language	2/1
2.1-1 Principle	2/1
2.1-2 Graphic elements	2/2
2.2 Structure of a rung	2/4
2.2-1 General	2/4
2.2-2 Labels	2/5
2.2-3 Comments	2/5
2.2-4 Rungs	2/6
2.2-5 Rungs with function and operation blocks	2/9
2.3 Rules for executing rungs	2/11
2.3-1 Principle for executing a rung	2/11
3 Instruction list language	3/1
3.1 Presentation of Instruction list language	3/1
3.1-1 Principle	3/1
3.1-2 Instructions	3/2
3.2 Program structure	3/4
3.2-1 General	3/4
3.2-2 Comments	3/4
3.2-3 Labels	3/4
3.2-4 Using parentheses	3/5
3.2-5 MPS, MRD and MPP instructions	3/7
3.2-6 Principles for programming predefined function blocks	3/8
3.3 Rules for executing Instruction list programs	3/9

Section	Page
4 Structured Text language	4/1
4.1 Presentation of Structured Text language	4/1
4.1-1 Principle	4/1
4.1-2 Instructions	4/2
4.2 Program structure	4/6
4.2-1 General	4/6
4.2-2 Comments	4/6
4.2-3 Labels	4/7
4.2-4 Instructions	4/7
4.2-5 Control structures	4/8
4.3 Rules for executing a Structured Text program	4/14
5 Grafcet language	5/1
5.1 Presentation of Grafcet language	5/1
5.1-1 Reminder of principles of Grafcet	5/1
5.1-2 Graphic symbols specific to Grafcet language	5/2
5.1-3 Objects specific to Grafcet	5/4
5.1-4 Grafcet chart representation	5/5
5.1-5 Actions associated with steps	5/11
5.1-6 Conditions associated with transitions	5/14
5.2 Organization of the master task	5/17
5.2-1 Description of the master task	5/17
5.2-2 Preprocessing	5/18
5.2-3 The use of system bits in preprocessing	5/19
5.2-4 Sequential processing	5/21
5.2-5 Post-processing	5/23

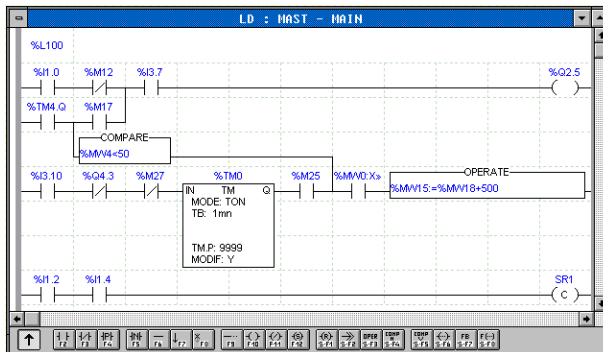
1.1 Presentation of PL7 software

1.1-1 Presentation

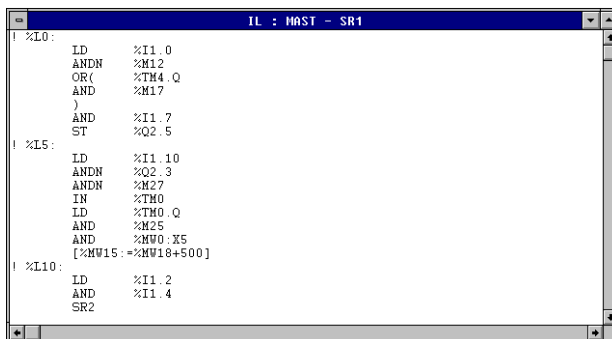
PL7 Junior software is the programming software for TSX 37 and TSX 57 PLCs operating under Windows. PL7 Micro software can only be used to program TSX 37 PLCs.

PL7 software offers :

- A graphic language, Ladder language, for transcribing relay diagrams, which is especially suitable for combinational processing and offers basic graphic elements, that is, contacts and coils. Numeric calculations can be written within operation blocks.



- A Boolean language, Instruction list language, which is a "machine" language for writing logical and numerical processing operations.



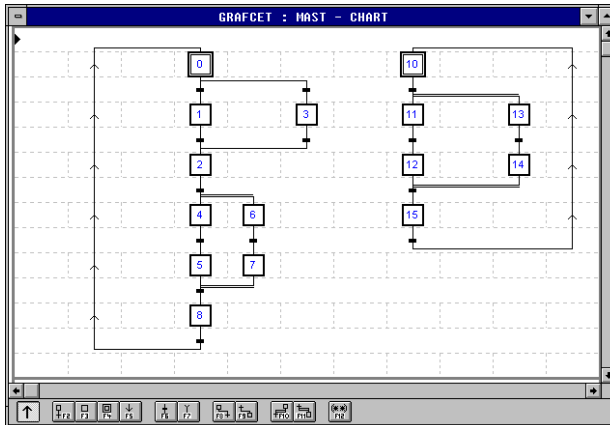
- A Structured text language which is a "data processing" type language enabling the structured writing of logical and numerical processing.

```

ST : MAST - POST
IF %M3 THEN
  FOR %MW99:=0 TO 31 DO
    IF %MW100[%MW99]<>0 THEN
      %MW10:=%MW100[%MW99];
      %MW11:=%MW99;
      %M1:=TRUE;
      EXIT;
    ELSE
      %M1:=FALSE;
    END_IF;
  END_FOR;
ELSE
  %M1:=FALSE;
END_IF;

```

- A Grafcet language which is used to represent the operation of a sequential control system in a graphic and structured way.



These languages include predefined function blocks (timers, counters, etc), which can be supplemented by application-specific functions (analog, communication, counting, etc) and specific functions (time management, character strings, etc).

The language objects can be symbolized.

PL7 software conforms to standard IEC 1131-3. The tables of conformity are provided in the Appendix : part B section 6.

1.1-2 Single task structure

This is the default structure of the software. It comprises a single task, the master task.

Master task

This task can either be a periodic, said to be cyclic (the default choice), or periodic. For cyclic operation, the tasks are linked one to the other, without pausing. For periodic operation, tasks are linked at a period fixed by the user.

1.1-3 Multitask structure

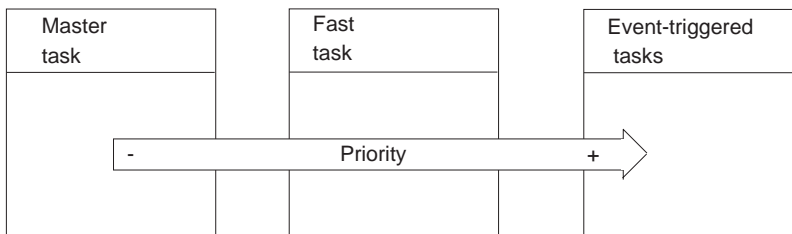
The multitask structure of TSX 37 and TSX 57 PLCs enables better use which gives high-performance real-time applications by associating a specific program with each application. Each of these programs is controlled by a task.

These tasks are independent and executed in "parallel" by the main processor which manages their priority as well as their execution.

The aim of this type of structure is to :

- Optimize use of processing power.
- Simplify design and debugging. Each task is written and debugged independently of the others.
- Structure the application. Each task has a unique function.
- Optimize availability.

The multitask system offers a master task, a fast task and from 8 to 64 event-triggered tasks depending on the processor.



Fast task

The fast task (optional), which is periodic, is used to perform short processing operations with a higher priority than in the master task. When it is programmed, it is automatically launched by the system during start-up. It can also be stopped and then restarted by activating a system bit.

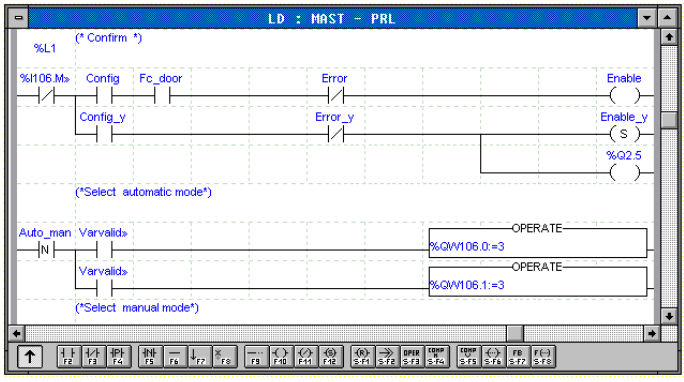
Event-triggered tasks

Unlike the tasks described above, these tasks are not linked to a period. They are triggered by calls originating from certain modules. These tasks have the highest priority. The processing they perform is deliberately short so that they do not interfere with the execution of other tasks.

1.1-4 Symbolic programming

Using PL7 software, the user can choose to enter or display objects :

- either by their address (for example : %Q2.5),
- or by a character string (32 characters max) known as a symbol (for example Fc_door).



Note

Addresses and symbols can be displayed simultaneously in Ladder language.

Symbols used can be entered beforehand or while editing the program.

This symbol database, which is managed by the software VARIABLES editor, is global to the PLC station.

Variables					
Parameters		I/O	Module Address	3	<input checked="" type="checkbox"/> Entry Field
Sensor_3					
Address	Type	Symbol	Comment		
%I3.1	EBOOL	Sensor_2	Sensor for identifying the type of item (0stype2, 1stype 1)		
%Mw3.1.2	WOPRD				
%I3.2.ERR	BOOL				
%I3.2	EBOOL	Sensor_3	sensor for detecting clamp open/clamp closed		
%Mw3.2.2	WOPRD				
%I3.3.ERR	BOOL				
%I3.3	EBOOL	Auto_man	SWITCH for selecting AUTOMATIC (=0) or MANUEL (=1) mode		
%Mw3.3.2	WOPRD				
%I3.4.ERR	BOOL				
%I3.4	EBOOL	Start_cycle	pushbutton to START automatic cycle		
%Mw3.4.2	WOPRD				
%I3.5.ERR	BOOL				
%I3.5	EBOOL	Stop_cycle	pushbutton to STOP automatic cycle		

1.2 Addressable objects

1.2-1 Definition of main Boolean objects

Input/output bits

These bits are the "logical images" of the electrical state of the I/O. They are stored in the data memory and are updated on each scan of the task in which they are configured.

Internal bits

Internal bits %Mi are used to store intermediate states during execution of the program.

Note : Unused I/O bits cannot be used as internal bits.

System bits

System bits %S0 to %S127 monitor correct operation of the PLC as well as progression of the application program. The role and use of these bits are described in detail in section 3.1 of part B.

Function block bits :

Function block bits correspond to the outputs of blocks. These outputs can be either wired directly, or used as objects.

Word extract bits :

Using PL7 software it is possible to extract one of the 16 bits from a word object.

Grafcet step status bits

Grafcet step status bits %Xi are used to identify the status of Grafcet step i.

List of bit operands

The following table gives a list of all types of Boolean operands.

Type	Address (or value)	Max number (2)		Access in write mode (1)	See	Part
		TSX 37	TSX 57		Sctn.	
Immediate value	0 or 1 (False or True)	–	–	–	1.2-4	A
Input bits	%Ix.i or %IXx.i	328	1024	no	1.2-2	A
Output bits	%Qx.i or %QXx.i			yes	1.2-3	
Internal bits	%Mi or %MXi	256	4096 (3)	yes	–	
System bits	%Si	128	128	depending on i	3.1	B
Function block bits	eg : %Tmi.Q %DRi.F.....	–	–	no	1.2-5	A
Step bits	%Xi	96	128	yes	5.1-3	A
Word extract bits	eg : %MW10:X5	–	–	depdg on word type	1.2-4	A

(1) Written via the program or in adjust mode via the terminal

(2) Depends on the processor being used.

(3) The maximum total of the bits (I/O + internal + system) is 4096.

1.2-2 Addressing TSX 37 I/O module objects

Addressing of the main word and bit objects in I/O modules is defined by the following characters :

%	I or Q	X, W or D	x	.	i
Symbol	Type of object I = input Q = output	Format X = Boolean W = word D = double word	Position x= Position number in the rack		Channel No. i= 0 to 127 or MOD

- **Type of object**

I and Q : the physical inputs and outputs of modules exchange this information implicitly on each scan of the task to which they are attached.

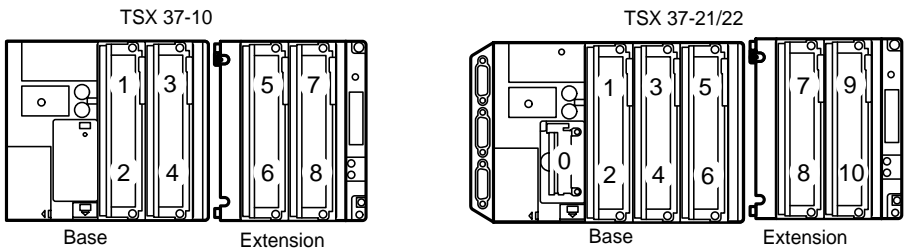
Note : Other types of data (status, command words, etc) can also be exchanged if requested by the application (see application-specific functions : part F).

- **Format (Size)**

For objects in Boolean format, the X can be omitted. Other types of format such as byte, word and double word are defined in section 1.2-4.

- **Channel position and number**

The base modularity of the TSX 37 is 1/2 format. The positions for each type of TSX 37 PLC (base and extension) are shown in the diagrams below.



Standard format modules are addressed as two superposed 1/2 format modules (see table below).

For example, a 64 I/O module is viewed as two 1/2 format modules : a 32 input 1/2 module located in position 5, and a 32 output 1/2 module located in slot 6.

Module	1/2 format			Standard format			
	4 Q	8 Q	12 I	28 I/O	32 I	32 Q	64 I/O
Channel number : i	0 to 3	0 to 7	0 to 11	0 to 15 0 to 11	0 to 15 0 to 15	0 to 15 0 to 15	0 to 31 0 to 31
Position and ch. no. (x = position)	x.0 to x.3	x.0 to x.7	x.0 to x.11	x.0 to x.15 (x+1).0 to (x+1).11	x.0 to x.15 (x+1).0 to (x+1).15	x.0 to x.15 (x+1).0 to (x+1).15	x.0 to x.31 (x+1).0 to (x+1).31

Note :

The channel number can be replaced by "MOD" to access data which is general to the module.

- **Suffix :** an optional suffix can be added after the channel number. It is used to distinguish different objects of the same type associated with the same channel (see applications, part F).

ERR : indicates a module or channel fault.

Examples : %I4.MOD.ERR : information on fault in module 4

%I4.3.ERR : information on fault in channel 3 of module 4.

Note :

For an addressing operation across the network or remote I/O, the full access path to the station is added to the position number in the rack.

Examples

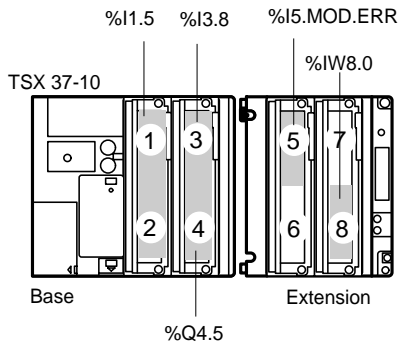
%I1.5 input channel no. 5 of the module located at position no. 1.

%I3.8 input channel no. 8 of the normal format module located in position no. 3 and 4.

%Q4.5 output channel no. 5 of the standard format module located in position no. 3 and 4.

%I5.MOD.ERR Information on module fault, of the module located in position no. 5.

%IW8.0 input channel no. 0 of the 1/2 format module located in position no. 8.



1.2-3 Addressing TSX 57 I/O module objects

Addressing of the main word and bit objects in I/O modules is defined as follows :

%	I or Q	X, W or D	x	y	•	i
Symbol	Type of object I = input Q = output	Format X = boolean W = word D = double word	Rack address x = 0 to 7	Module position y = 00 to 10		Channel No. i = 0 to 127 or MOD

- **Type of object**

I and Q : the physical inputs and outputs of modules exchange this information implicitly on each scan of the task to which they are attached.

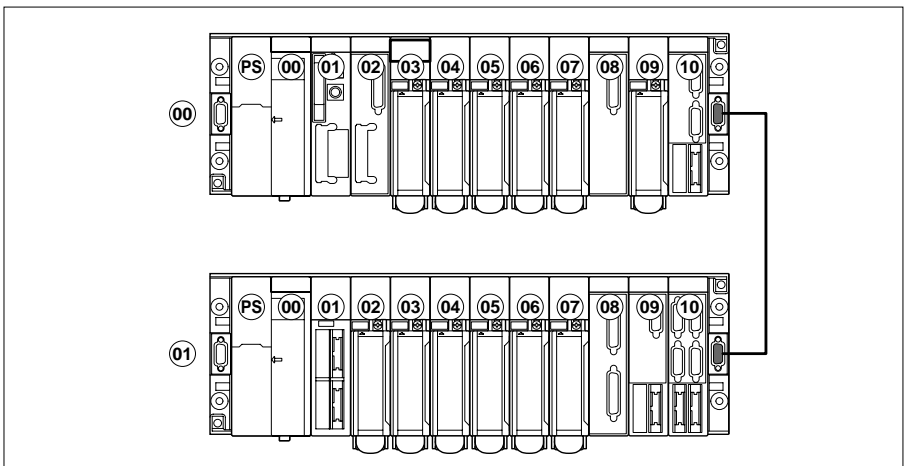
Note : Other types of data (status, command words, etc) can also be exchanged if requested by the application (see application-specific functions : part F).

- **Format (Size)**

For objects in Boolean format, the X can be omitted. Other types of format such as byte, word and double word are defined in section 1.2-4.

- **Channel addressing**

Channel addressing depends on the rack address, the physical location of the module in the rack and the channel number.



Rack addresses (x) and module positions (y)

TSX racks	RKY 6	RKY 8	RKY 12	RKY 6E	RKY 8E	RKY 12E
Rack address : x	0	0	0	0 to 7	0 to 7	0 to 7
Module position : y	00 to 04	00 to 06	00 to 10	00 to 04	00 to 06	00 to 10

Note :

The address of the rack supporting the processor is always 0.

Channel number (i)

Modules TSX DEY/DSY	64 I/O	32 I/O	16 I/O	8 I/O
Channel number : i	0 to 63	0 to 31	0 to 15	0 to 7

Note :

The channel number can be replaced by "MOD" to access general module information.

- **Suffix** : an optional suffix can be added after the channel number. It is used to distinguish between different objects of the same type which are associated with a single channel (see application-specific functions, part F).

ERR : indicates a module or a channel fault.

Examples :

- %I104.MOD.ERR : information on fault in the module in position 4 in rack at address 1.
- %I104.3.ERR : information on fault in channel 3 of the module in position 4 in rack at address 1.

Note :

For an addressing operation across the network or addressing remote I/O, the full access path to the station is added to the channel address.

Examples :

- **%I102.5** : input channel 5 of the module located in position 2 in rack at address 1.
- **%Q307.2** : output channel 2 of the module located in position 7 in rack at address 3.
- **%I102.MOD.ERR** : information on module fault, of the module located in position 2 in rack at address 1.

1.2-4 Addressing words

Addressing I/O module words is defined in section 1.2-2 or 1.2-3. Other words used in PL7 language (except network words and function block words) are addressed in the following way :

%	M, K or S	B, W, D or F	i
Symbol	Type of object	Format	Number
	M = internal K = constant S = system	B = byte W = word D = double word F = floating point	

• **Type of object**

M internal words which store values during execution of the program. They are stored in the data zone within a single memory zone.

K constant words which 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 use the FLASH EPROM memory as their support.

S system words. These words perform several functions :

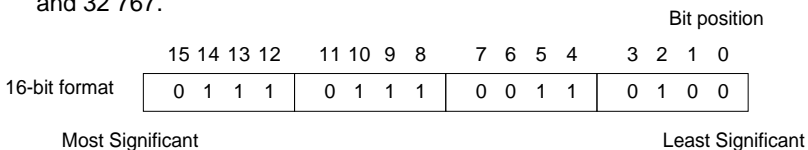
- some provide information on the status of the system by reading %SWi words (system and application operating time, etc).
- others are used to perform operations on the application (operating mode, etc). System words are described in section 3, part B.

• **Format**

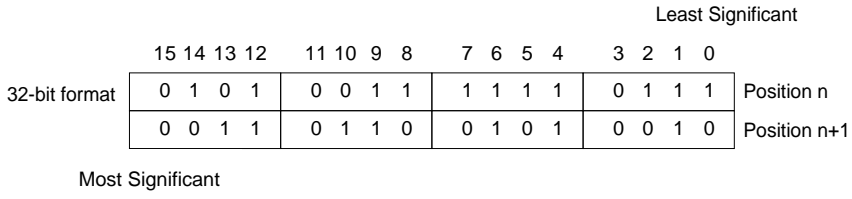
Word objects can be addressed by PL7 software using four different formats :

B byte : this format is used exclusively for operations on character strings.

W single length : these 16-bit words can contain an algebraic value between -32 768 and 32 767.



D 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.



F floating point : the floating point format used is that of IEEE Std 754-1985 (equivalent to IEC 559). Words are 32 bits long, which corresponds to single length floating point numbers.

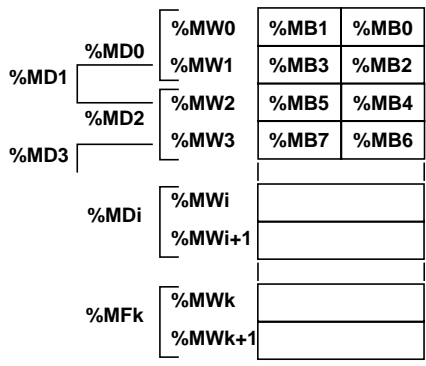
Example of floating point values :
 1285.28
 12.8528E2

Overlay between objects :

Bytes, single, double length and floating point words are stored within the data area in the same memory zone.

Thus, there is overlay between :

- Double length word %MDi and single length words %MWi and %MWi+1 (word %MWi containing the least significant bits and word %MWi+1 containing the most significant bits of word %MDi).
- Single length word %MWi and bytes %MBj and %MBj +1 (where j=2.i).
- Between the floating point %MFk and single length words %MWk and %MWk+1.



Examples :

- %MD0 corresponds to %MW0 and %MW1.
- %MW3 corresponds to %MB7 and %MB6.
- %KD543 corresponds to %KW543 and %KW544.
- %MF10 corresponds to %MW10 and %MW11.

Immediate values

These are algebraic values whose format is similar to that of single and double length words (16 or 32 bits), which allow assignment of values to these words. They are stored in the program memory and can take the following syntax :

Type	Syntax	Lower limit	Upper limit
Boolean	0 or 1 (FALSE or TRUE)		
Base 10 integer	single length	1506	+32767
	double length	578963	2 147 483 647
Base 2 integer (binary)	single length	2#1000111011111011011	2#01...1
	double length	2#1000111011111011011111111011111011111	2#01...1
Base 16 integer (hexadecimal)	single length	16#AB20	16#FFFF
	double length	16#5AC10	16#FFFFFFFF
Floating point	-1.32E12	-3.402824E+38 1.175494E-38	-1.175494E-38 3.402824E+38
Character string	'aAbBcC'		

Addressing common words on the network

These are single length word objects (16 bits) common to all stations connected to the communication network.

Addressing :

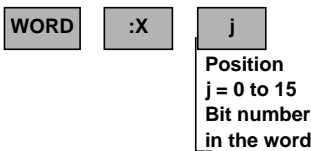
- on network n° 0 : %NW{j}k

where :

- j = 0 to 31 station n°
- k = 0 to 3 word n°

Word extract bits

PL7 software can be used to extract one of the 16 bits from single length words. The number of the bit extracted is then added to the word address, following the syntax below :



Examples :

%MW10:X4 = bit n° 4 of internal word %MW10

%QW5.1:X10 = bit n° 10 of output word %QW5.1

Summary list of main word objects and associated bits

The notations used are R for read, and W for write.

Associated words and bits	Type	Addressing	Limits	Possibilities
Internal words	single length	%MWi	(1)	R/W
	double length	%MDi	(1)	R/W
	floating point	%MFi	(1)	R/W
	byte (2)	%MBi	(1)	R/W
Constant words	single length	%KWi	(1)	R/W (3)
	double length	%KDi	(1)	R/W (3)
	floating point	%KFi	(1)	R/W (3)
	byte (2)	%KBi	(1)	R/W (3)
I/O module words	Input single length	%IWxy.i	$0 \leq i \leq 127$	R
	Input double length	%IDxy.i	$0 \leq i \leq 126$	R
	Output single length	%QWxy.i	$0 \leq i \leq 127$	R/W
	Output double length	%QDxy.i	$0 \leq i \leq 126$	R/W
Common words	on network	%NW{j}k	$0 \leq j \leq 31$ $0 \leq k \leq 3$	R/W
System words	single length	%SWi	$0 \leq i \leq 127$	R/W (4)
	double length	%SDi	$0 \leq i \leq 126$	R/W (4)
Word extract bits	bit j of internal word	%MWi:Xj	$0 \leq j \leq 15$	R/W
	bit j of constant word	%KWi:Xj	$0 \leq j \leq 15$	R/W (3)
	bit j of input word	%IWi:Xj	$0 \leq j \leq 15$	R
	bit j of output word	%QWi:Xj	$0 \leq j \leq 15$	R/W
	bit j of system word	%SWi:Xj	$0 \leq j \leq 15$	R/W (4)
	bit j of network 0 common word	%NW{j}k:Xm	$0 \leq m \leq 15$	R/W

- (1) The maximum limit depends on the memory size available and the number of words declared during software configuration.
- (2) This object only exists in the start address of a character string %MBi:L or %KBi:L (see section 2.8-1, part B).
- (3) Write by terminal only.
- (4) Write depending on i.

1.2-5 Function block objects

Function blocks use specific bit and word objects.

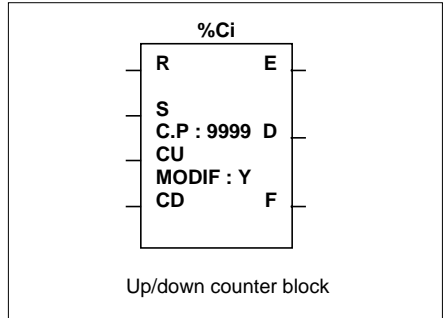
- Bit objects :**

These correspond to the outputs of blocks. These bits are accessible using Boolean test instructions.

- Word objects :**

These correspond :

- to the configuration parameters of the block. The program may (eg : preset parameter) or may not (eg : time base) be used to access these parameters,
- to the current values (eg : %Ci.V current counter value).



List of function block bit and word objects accessible by program

Predefined function blocks	Associated words and bits	Address	Access in write mode	See Part B	
Timer %Tmi (i=0 to 63) (1)	Word	Current value	%Tmi.V	no	sn1.3-2
		Preset value	%Tmi.P	yes	
	Bit	Timer output	%Tmi.Q	no	
Up/down counter %Ci (i=0 to 31)	Word	Current value	%Ci.V	no	sn1.3-3
		Preset value	%Ci.P	yes	
	Bit		Underflow output (empty)	%Ci.E	no
			Preset reached output	%Ci.D	no
			Overflow output (full)	%Ci.F	no
Monostable %Mni (i=0 to 7)	Word	Current value	%Mni.V	no	sn2.2-1
		Preset value	%Mni.P	yes	
	Bit	Current monostable output	%Mni.R	no	
Register %Ri (i= 0 to 3)	Word	Register input	%Ri.I	yes	sn2.2-2
		Register output	%Ri.O	yes	
	Bit	Register full output	%Ri.F	no	
		Register empty output	%Ri.E	no	
Drum controller %DRi (i=0 to 7)	Word	Current step number	%DRi.S	yes	sn2.2-3
		Status of step j	%DRi.Wj	no	
		Active time of the step	%DRi.V	no	
	Bit	Last current step defined	%DRi.F	no	
Series 7 timer %Ti (i=0 to 63) (1)	Word	Current value	%Ti.V	no	sn2.2-4
		Preset value	%Ti.P	yes	
	Bit	Timer running output	%Ti.R	no	
		Timer done output	%Ti.D	no	

(1) The total number of timers %Tmi and %Ti is limited to 64 for a TSX 37, 255 for a TSX 57.

(2) The maximum number is given for a TSX 37, for a TSX 57 i = 0 to 254 for all function blocks.

1.2-6 Structured objects

Bit tables

Bit tables are sequences of adjacent bit objects of the same type and of a defined length, L.

%M10 %M11 %M12 %M13 %M14 %M15

Example of bit tables : %M10:6

--	--	--	--	--	--

Type	Address	Maximum size	Access in write mode
Discrete input bits	%Ix.i:L	$1 \leq L \leq m$ (1)	No
Discrete output bits	%Qx.i:L	$1 \leq L \leq m$ (1)	Yes
Internal bits	%Mi:L	$i+L \leq n$ (2)	Yes
Step bits	%Xi:L	$i+L \leq n$ (2)	No

(1) m = modularity of the module (eg : 8 for an 8 input or 8 output module),

(2) n varies according to the size defined during configuration.

Word tables

Word tables are sequences of adjacent words of the same type and of a defined length, L.

%KW10	16 bits
%KW14	

Example of word tables : %KW10:5

Type	Format	Address	Maximum size	Access in write mode
Internal words	Single length	%MWi:L	$i+L \leq N_{max}$ (2)	Yes
	Double length	%MDi:L	$i+L \leq N_{max}-1$ (2)	Yes
	Floating point	%MFi:L	$i+L \leq N_{max}-1$ (2)	Yes
Constant words	Single length	%KWi:L	$i+L \leq N_{max}$ (2)	No
	Double length	%KDi:L	$i+L \leq N_{max}-1$ (2)	No
	Floating point	%KFi:L	$i+L \leq N_{max}-1$ (2)	No
System words	Single length	%SW50:4 (3)		Yes

Character strings

Character strings are sequences of adjacent bytes of the same type and of a defined length, L.

%MB10	8 bits
%MB14	

Example of character string : %MB10:5

Type	Address	Maximum size	Access inwrite mode
Internal words	%MBi:L (5)	$1 \leq i+L \leq N_{max}$ (4)	Yes
Constant words	%KBi:L (5)	$1 \leq i+L \leq N_{max}$ (4)	Yes

(3) Only words %SW50 to %SW53 can be addressed in table form.

(4) Nmax = maximum number defined during software configuration.

(5) i must be even.

Indexed objects

• Direct addressing

Addressing of objects is said to be direct when the address of these objects is fixed and defined when the program is written.

Example : %MW26 (internal word at address 26)

• Indexed addressing

In indexed addressing, an index is added to the direct address of the object : the content of the index is added to the address of the object. The index is defined by an internal word %MWi. The number of "index words" is unlimited.

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

Type	Format	Address	Maximum size	Access in write
Input bit	Boolean	%Ii[%MWj]	$0 \leq i + \%MWj \leq m$ (1)	No
Output bit	Boolean	%Qi[%MWj]	$0 \leq i + \%MWj \leq m$ (1)	Yes
Internal bit	Boolean	%Mi[%MWj]	$0 \leq i + \%MWj \leq N_{max}$ (2)	Yes
Internal words	Single length	%MWi[%MWj]	$0 \leq i + \%MWj \leq N_{max}$ (2)	Yes
	Double length	%MDi[%MWj]	$0 \leq i + \%MWj \leq N_{max} - 1$ (2)	Yes
	Floating point	%MFi[%MWj]	$0 \leq i + \%MWj \leq N_{max} - 1$ (2)	Yes
Constant words	Single length	%KWi[%MWj]	$0 \leq i + \%MWj \leq N_{max}$ (2)	No
	Double length	%KDi[%MWj]	$0 \leq i + \%MWj \leq N_{max} - 1$ (2)	No
	Floating point	%KFi[%MWj]	$0 \leq i + \%MWj \leq N_{max} - 1$ (2)	No
Word table	<Object> [%MWj]:L	%MWi[%MWj]:L	$0 \leq i + \%MWj + L \leq N_{max}$ (2)	Yes

(1) m = modularity of the I/O module (eg : 8 for an 8 input or 8 output module). Indexation is only possible for discrete I/O modules.

(2) Nmax = maximum number defined during software configuration.

This type of addressing is used to run through a series of objects of the same type (internal words, constant words etc) in succession. The content of the index is added to the object address.

Note :

Indexing double words (or floating points).

Example : %MD6[%MW100] direct address double word 6 + 2 times the content of word %MW100.
If %MW100=10, the word addressed will be 6 + 2 x 10 -->%MD26.

- **Index overrun, system bit %S20**

Index overrun occurs as soon as the address of an indexed object exceeds the limits of the zone including this same type of object, that is, when :

- object address + index content is lower than the value zero.
- object address + index content is higher than the maximum limit configured (see table on the previous page).

In the event of index overrun, the system sets system bit %S20 to 1 and the object is assigned an index value of 0.

Monitoring of overrun is the responsibility of the user : bit %S20 must be read by the user program for any processing. Resetting it is the responsibility of the user.

%S20 (initial state = 0) :

- on index overrun : set to 1 by the system,
- overrun acknowledgment : set to 0 by the user after modification of the index.

1.2-7 Grafcet objects

Bit objects

The user has bit objects %Xi associated with the steps which enable him to identify the status of Grafcet step i. This bit is set to 1 when the step is active, and 0 when it is inactive.

Word objects

One word is associated with each step : %Xi.T. It indicates the activity time of Grafcet step i. It is incremented every 100ms and takes a value between 0 and 9999.

1.2-8 Symbolization

Symbols

A symbol is a string of up to 32 alphanumeric characters, of which the first character is alphabetical. A symbol starts with a capital letter, followed by letters in lower case (for example : the symbol Burner_1). When it is entered, the symbol can be written in upper or lower case (for example : BURNER_1). The software automatically puts the symbol into its correct format.

The following characters can therefore be used :

- upper case alphabetical :
"A to Z" and accented letters "ÀÁÂÃÄÅÆÇÈÉÊËÌÍÎÏÐÑÒÓÔÕÖØÙÚÛÜÝÞ"
- or lower case alphabetical :
"a to z", and accented letters : `àáâãäåæçèéêëìíîïðñòóôõöøùúûüýþÿ"
- numerical : digits 0 to 9 (they cannot be placed at the start of the symbol).
- the character "_" (this cannot be placed either at the beginning or end of the symbol).

A number of words are reserved by the language and cannot be used as symbols. See full list in section 5 part B.

The symbols are defined and associated with language objects by the variables editor (see section 5, part D). A comment of 508 characters can be associated with each symbol. The symbols and their comments are saved to the terminal disk and not in the PLC.

Objects which can be symbolized

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

Examples :

- if word %MW0 has "Temperature" as its symbol, the word table %MW0:12 is symbolized by "Temperature:12"
- if word %MW10 has "Oven_1" as its symbol, the indexed word %MW0[%MW10] is symbolized by "Temperature[oven_1]".

Word extract bit objects and function block bits or words can be symbolized, but if they are not symbolized, they can take on the symbol of the base object.

Examples :

- if word %MW0 has "Pump_status" as its symbol and if the word extract bit %MW0:X1 is not symbolized, it takes on the symbol of the word. %MW0:X1 has as its symbol, "Pump_status:X1".
- if function block %TM0 has "Oven1_timer" as its symbol and if output %TM0.D is not symbolized, it takes on a symbol of the block. %TM0.D has as its symbol, "Oven_timer.D".

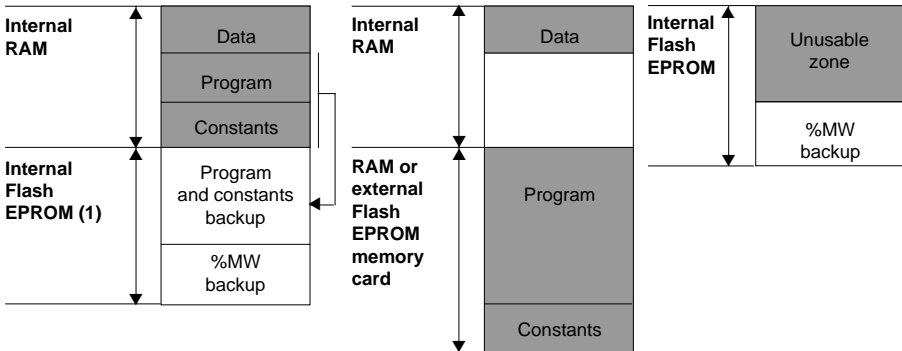
1.3 User memory

1.3-1 General

The memory space of **TSX 37 PLCs**, which is accessible to the user, is divided into two distinct units :

- **Bit memory :**
RAM integrated in the processor module containing the image of 1280 bit objects.
- **Word memory :**
16-bit words (program, data and constants) supported by a RAM memory in the processor module. This memory can be extended by a 32 or 64 K word RAM or FLASH EPROM user memory card (on TSX 37-21/22).
A 16 K word FLASH EPROM memory integrated in the processor module can be used to back up the application program (15 Kwords) and 1000 internal words (1 Kword) (see paragraph 1.3-2).
A 32 K word FLASH EPROM backup card can also be used to update an application in the internal RAM of the processor. This card contains the program part and the constants but not the data.
The word memory can be organized in 2 different ways depending on whether or not a memory card (PCMCIA) is used :

TSX 37-10 or TSX 37-21/22 (without PCMCIA card) **TSX 37-21/22 (with PCMCIA card)**



- Data** : application and system dynamic data.
- Program** : descriptors and executable code of tasks.
- Constants** : constant words, initial values and I/O configuration.

Note

RAM memories can be protected by Cadmium-nickel batteries, supported by the processor module for the bit and internal RAM memory.

(1) The application is transferred automatically from the FLASH EPROM memory to the RAM memory if the application in the RAM is lost (backup fault or absence of battery). A manual transfer can also be requested, via a programming terminal.

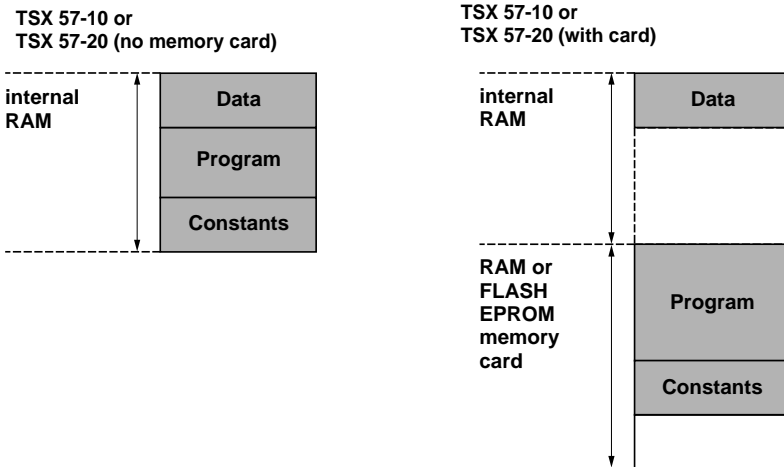
The memory space of **TSX 57 PLCs** only has one single unit. The bit memory, which is separate on the TSX 37, is integrated into the word memory (in the data zone). It is limited to 4096 bits.

• **Word memory :**

16-bit words (program, data and constants) supported by a RAM memory in the processor module. This memory can be extended by a 32 or 64 K word (on TSX 57-10) and by a 32, 64 or 128 K word (on TSX 57-20) RAM or FLASH EPROM user memory card.

A 32 K word FLASH EPROM backup card can also be used for updating an application in the internal RAM of the processor. This card contains the program part and the constants but not the data.

The memory can be organized in 2 different ways depending on whether or not a memory card (PCMCIA) is present and how it is used :



Data : dynamic application data and system data (the system reserves a RAM zone of 5 Kwords minimum : see part B, section 8).

Program : descriptors and executable code of tasks.

Constants : constant words, initial values and I/O configuration.

There is no possibility either of data overflow on the memory card or of having a program on the internal RAM and on the cartridge at the same time.

Note

RAM memories can be protected by nickel-cadmium batteries.

1.3-2 Saving / retrieving internal words %MWi

Saving internal words %MWi

In order to save adjustment data in the event of a power outage, when the processor battery is faulty or missing, TSX 37 PLCs can copy 1000 internal words (%MW) maximum to the internal Flash EPROM memory. This backup zone can be used at all times, even if the PLC is fitted with a PCMCIA memory card (TSX 37-21/22).

To save internal words to the Flash EPROM, **the application must be stopped**. It is triggered according to the choice made during configuration :

- by setting the discrete input %I1.9 to 1,
- from an adjustment panel, by setting bit 0 of %SW96 to 1.

The value of system word %SW97 defines the number of %MWi to be saved (1000 maximum).

At the end of the backup, the display block displays OK or NOK depending on the result of the operation.

The internal words %MWi are always saved when the application program is saved.

If system word %SW97 is initialized to 0, only the application program contained in the internal RAM is transferred to the Flash EPROM (equivalent to a Backup program).

Warning : any saved %MWi are still erased.

Retrieving internal words %MWi

Saved %MWi are transferred from the internal Flash EPROM memory to the RAM memory on a **cold restart** caused by :

- loss of the internal RAM contents. In this case, if the application program backup is valid, this is also transferred to the internal RAM (TSX 37-10 or TSX 37-20 without PCMCIA application cartridge),
- pressing the RESET button, on the front panel of the PLC,
- setting bit %S0 to 1, in adjust mode,
- clicking on the "Cold restart" button of the PL7 processor debug screen,
- transferring a program to the PLC (via terminal port, FIPWAY, etc),
- plugging in a PCMCIA application cartridge.

For the saved %MW to be retrieved to the internal RAM, the "Reset %MWi on cold restart" check box must not be checked in the processor configuration screen.

For further information, refer to part A of the TSX Micro installation manual.

1.3-3 Bit memory

Composition

This memory contains 1280 bit objects, for all types of TSX 37 PLC. For the TSX 57, this bit memory does not exist and its contents are found in the word memory in the application data zone.

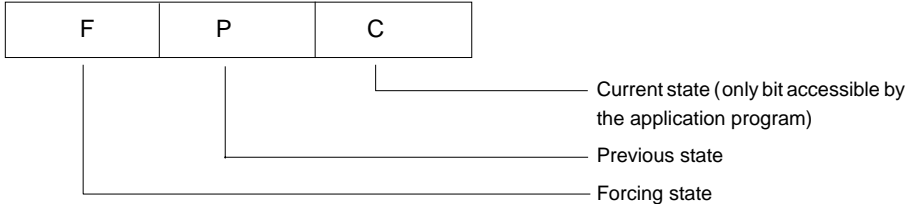
		TSX 37-10	TSX 37-21/22	TSX 57-10	TSX 57-20
System bits	%SI	128	128	128	128
I/O bits	%I/Qx	408 (1)	472 (1)	512	1024
Internal bits	%Mi	256	256	4096 (2)	4096 (2)
Step bits	%Xi	96	128	128	128

(1) With AS-i bus

(2) The number of internal bits can be set in the configuration. The default value (256 to 2048) varies depending on the processor being used and on the presence of a memory cartridge. The rest of the memory is available for application-specific functions. The total number of bits (system bits + I/O bits + internal bits) must not exceed 4096.

Structure

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



When the bit memory is updated, the system performs the following :

- the transfer of the image of the current state to the previous state,
- the updating of the current state via the program, the system or the terminal (when a bit is forced).

Rising or falling edge

This structure of the bit memory is used to test for a rising or falling edge on :

- I/O bits,
- internal bits.

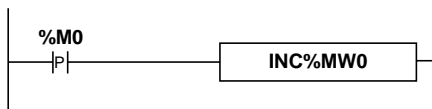
Recommendations for the use of rising or falling edges

The rising or falling edge contact instructions only operate correctly in the following conditions :

- in all cases, for a single object :
 - input bit : process the edge contact in the task to which the input module has been assigned,
 - output or internal bit : process reading and writing of it within the **same task**.

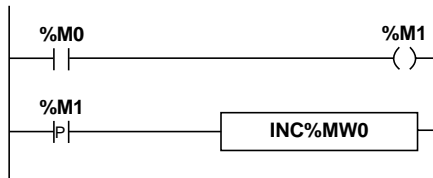
- Write the coil of an object **once only** when an edge contact of this object is used in a program.
- Do not perform a SET or RESET on an object where the edge is tested, because even if the result of the equation conditioning the SET/RESET equals 0, the SET/RESET action is not performed, but the object log is updated (loss of the edge).
- Do not test the edge of an I/O used in a event-triggered task, in a master or fast task.
- For internal bits : the detection of an edge is separate from the task scan. An edge on internal bit %Mi is detected when it changes state between 2 read operations. **This front remains detected as long as this internal bit has not been scanned in the action zone.**

Thus, in the example opposite, if bit %M0 is forced to 1 in an animation table, the edge is permanently on.



In order that the edge is only detected once, an intermediate internal bit must be used.

In this case the %M1 log is updated, therefore the edge only occurs once.



Forcing states

When there is a forcing request via the terminal :

- forcing state F is set to 1
- current state C is set to :
 - 1 if forcing to 1 is requested
 - 0 if forcing to 0 is requested.

These states remain unchanged until :

- forcing is deactivated and the bit in question updated,
- inverse forcing is requested, in which case only the current state is modified.

1.3-4 Word memory

This 16-bit word memory is structured into 3 logical areas :

- Data
- Program
- Constants

Application data
Application program
Application constants

of which the size is defined by configuration.

Application data memory

The data memory contains the following zones :

- **System words** : fixed number
- **Function blocks** : correspond to the words and I/O of these blocks (current, adjustment values, etc).
The number of each type of function block is fixed during configuration.
- **Internal words** : size defined by the number declared during configuration.
- **I/O** : correspond to the words associated with each module. Their number depends on the modules configured.
- **Network common words** : 4 common words per PLC station (only available if communication module present and configured for exchange of common words).

In the case of the TSX 57, the data memory also includes the information bits detailed in the preceding paragraph.

Application program memory

This zone contains the executable program code, graphic data (Ladder language rungs) and program comments.

Application constant memory

This zone contains the parameters of the function blocks and I/O modules defined during configuration, and constant words %KW.

Note :

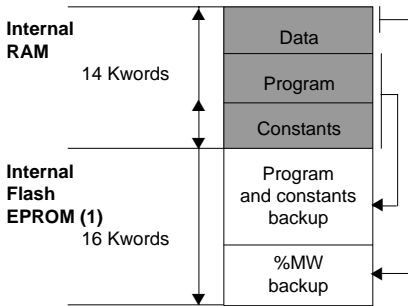
The symbols and comments associated with objects are not recorded in the PLC memory but stored in the local application (hard disk on the terminal).

1.3-5 TSX 37-10 PLCs

Structure of the bit memory

Bit memory		Size
Size available on processor		1280
Type	system bits %Si	128
of objects	I/O bits %I/Qx.i	408 (with AS-i)
	internal bits %Mi	256
	step bits %Xi	96

Structure of the word memory



Word memory	Size
Total size available on processor	14 Kwords
Data (%MWi)	0.5 Kwords (2)
Program (3)	
• Ladder language	4.0/3.0/2.1 Kinstructions
• Instruction list language	5.1/3.7/2.4 Kinstructions
• Structured text language	3.4/2.8/2.4 Kinstructions
Constants	128 words (2)

- (1) Of which 15K words are available for the program and the application constants and 1 Kword is reserved for the %MWi backup.
- (2) Default size can be extended. However, this will affect the size of the application program.
- (3) Sizes are given in Kinstructions for applications which are 100% Boolean, 90% Boolean, 65% Boolean respectively.

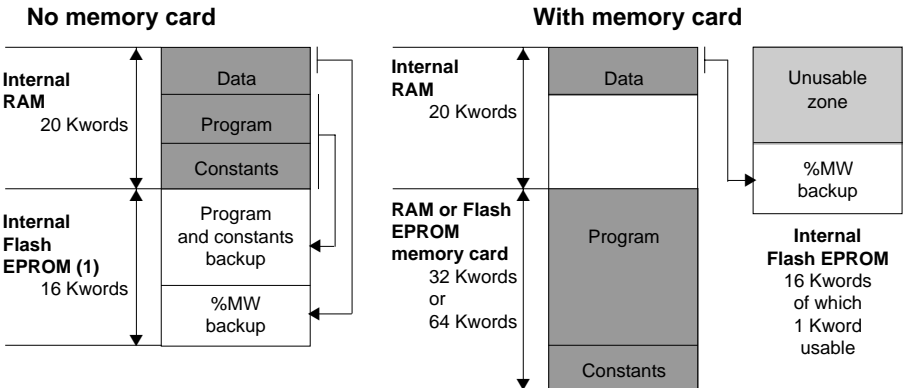
Note : the **PLC/Memory Usage** command in PL7 software provides the memory mapping of the application in the PLC memory.

1.3-6 TSX 37-21/22 PLCs

Structure of the bit memory

Bit memory	Size
Size available on processor	1280
Type system bits %Si	128
of objects I/O bits %I/Qx.i	472 (with AS-i)
internal bits %Mi	256
step bits %Xi	128

Structure of the word memory



(1) Can be used if the Program/Constants together are less than 15 K words

Word memory	No card	With 32 K card	With 64 K card
Total size available	20 Kwords	52 Kwords	84 Kwords
Data (%MWi)	0.5 Kwords	17.5 Kwords	17.5 Kwords
Program (3)			
• Ladder language	6.6/5.3/3.9 Kinstr.	13.5/11.6/8.8 Kinstr.	28.1/24.3/18.6 Kinstr.
• List language	8.5/6.5/4.4 Kinstr.	17.2/14.1/10.0 Kinstr.	35.9/29.6/21.0 Kinstr.
• Structured text language	5.6/5.0/4.4 Kinstr.	11.5/10.9/10.0 Kinstr.	23.9/23.0/21.0 Kinstr.
Constants	128 words (2)	256 words (2)	512 words (2)

(1) Of which 15K words are available for the program and the application constants and 1 Kword is reserved for the %MWi backup.

(2) Default size can be extended. However, this will affect the size of the application program.

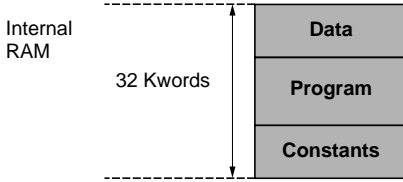
(3) Sizes are given in Kinstructions for applications which are 100% Boolean, 90% Boolean, 65% Boolean respectively.

Note : the **PLC/Memory Usage** command in PL7 software provides the memory mapping of the application in the PLC memory.

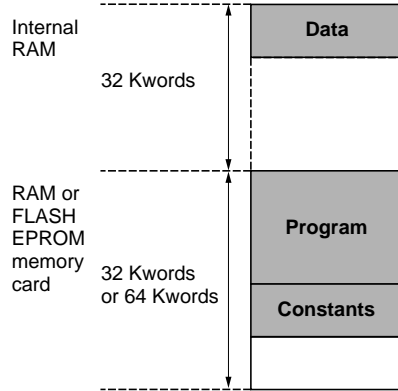
1.3-7 TSX 57-10 PLCs

Structure of the word memory

No memory card



With memory card



Bit memory	Size
System bits %Si	128
I/O bits %I/Qx.i	512
Internal bits %Mi	4096
Step bits %Xi	128

The number of internal bits can be set in the configuration. The values indicated are maximum values and they depend on the presence of a memory card.

Word memory	No card	With 32 K card	With 64 K card
Total size available	32 Kwords	64 Kwords	96 Kwords
Data (%MWi)	0.5 Kwords (1)	26 Kwords	26 Kwords
Program (2)			
• Ladder language	8.5/5.0/3.4 Kinstr.	11.9/8.3/6.4 Kinstr.	26.6/21.0/16.4 Kinstr.
• List language	10.9/5.9/3.5 Kinstr.	15.1/9.9/6.5 Kinstr.	33.9/25.2/16.8 Kinstr.
• Structured text language	7.2/4.8/4.0 Kinstr.	10.1/7.9/7.6 Kinstr.	22.6/20.1/19.4 Kinstr.
Constants (1)	128 words	256 words	512 words

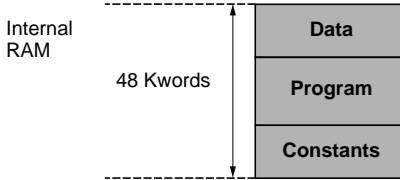
- (1) Default size can be extended. However, this will affect the size of the application program.
- (2) Sizes are given in Kinstructions for applications which are 100% Boolean, 90% Boolean, 65% Boolean respectively.

Note : the **PLC/Memory usage** command in PL7 software provides the memory mapping of the application in the PLC memory.

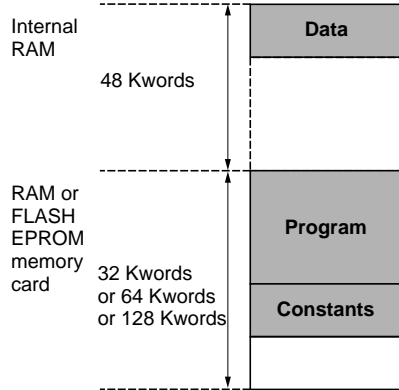
1.3-8 TSX 57-20 PLCs

Structure of the word memory

No memory card



With memory card



Bit memory	Size
System bits %Si	128
I/O bits %I/Qx.i	1024
Internal bits %Mi	4096
Step bits %Xi	128

The number of internal bits can be set in the configuration. The values indicated are maximum values and depend on the presence of a memory card.

Word memory	No card	With 32 K card	With 64 K card	With 128 K card
Total size available	48 Kwords	80 Kwords	112 Kwords	176 Kwords
Data (%MWi)	1 Kword (1)	30.5 Kwords	30.5 Kwords	30.5 Kwords
Program (2)				
• Ladder language	14.1/10.0/7.2 K	11.0/7.6/5.8 K	25.4/20.1/15.6 K	55.2/45.8/35.8 K
• List language	18.0/11.9/7.4 K	14.0/9.0/6.0 K	32.3/24.0/16.0 K	70.3/54.7/36.6 K
• Structured text language	12.0/9.6/8.6 K	9.3/7.2/6.9 K	21.5/19.2/18.5 K	46.9/43.9/42.4 K
Constants (1)	256 words	256 words	1024 words	1024 words

- (1) Default size can be extended. However, this will affect the size of the application program.
- (2) Sizes are given in Kinstructions for applications which are 100% Boolean, 90% Boolean, 65% Boolean respectively.

Note : the **PLC/Memory Usage** command in PL7 software provides the memory mapping of the application in the PLC memory.

1.4 Operating modes

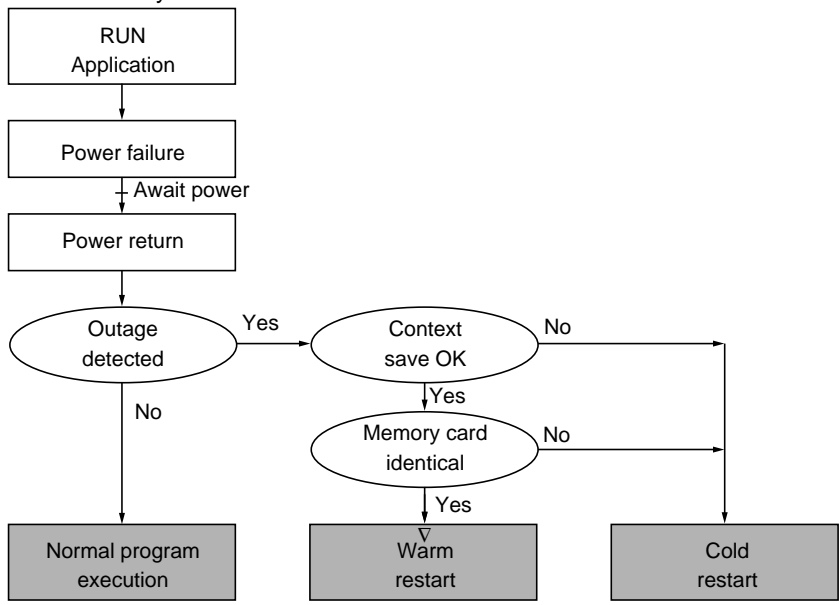
1.4-1 Processing on outage and power return

When power outage occurs, the system saves the application context and time of the outage and then sets all outputs to fallback state (state defined by configuration).

On power return, the context saved is compared to the current one. This defines the type of restart to be performed :

- If the application context has changed (loss of the system context or new application), the PLC initializes the application, that is, performs a cold restart.
- If the application context is identical, the PLC performs a restart without data initialization, that is, a warm restart.

If the period of the outage is shorter than the supply filter time (approximately 10 ms for an AC power supply or 1 ms for DC power supply), it is not seen by the program, which proceeds normally.



Note :

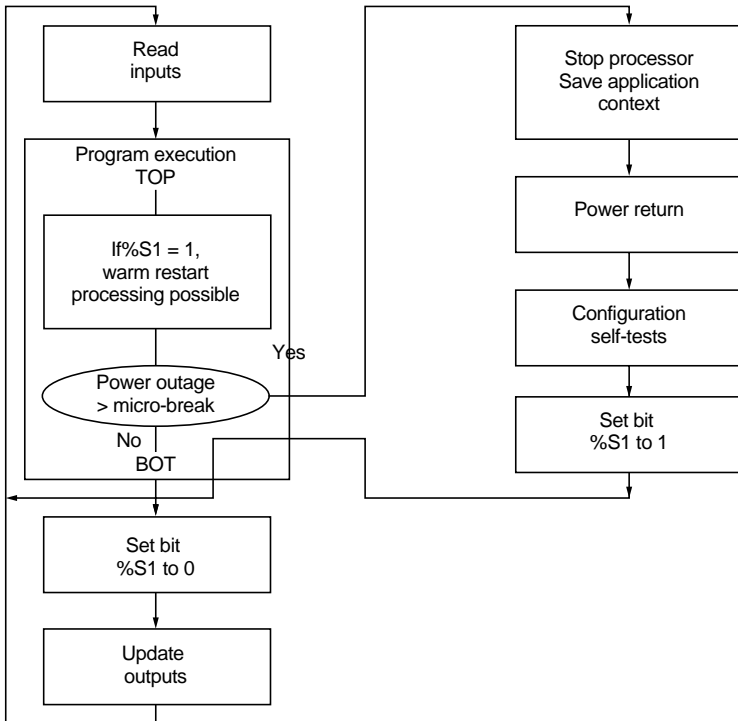
A cold restart can be activated :

- On power return with loss of context (example : processor backup battery not operating).
- During the first execution of an application.
- By pressing the RESET button on the processor.
- By the program setting system bit %S0 to 1.
- Through initialization from PL7 by the terminal.
- Then the PCMCIA memory card is inserted in its slot (or when the handle is manipulated).

A warm restart can be activated :

- On power return without loss of context.
- By the program setting system bit %S1 to 1,
- From PL7 by the terminal.

1.4-2 Warm restart processing



Restart program execution

Program execution restarts from the element at which power outage took place, without updating outputs. The system then performs a restart cycle in which it takes into account again all the input modules, relaunches the master task with bit %S1 set to 1 during one scan of the task, and updates the outputs.

The system deactivates the fast and event-triggered tasks until the end of the first scan of the master task.

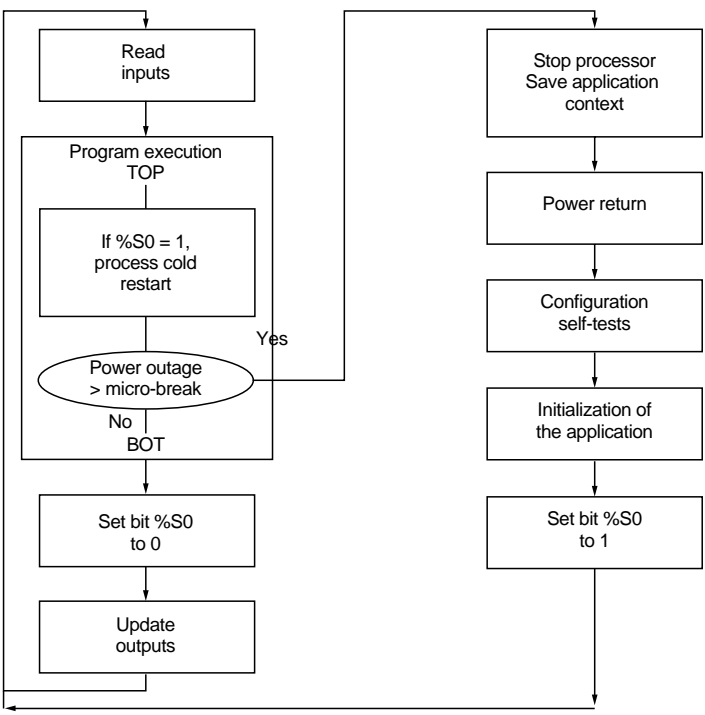
Warm restart processing

In the event of warm restarts users who require a particular processing operation in relation to the application must write the corresponding program by testing for %S1 at 1 at the beginning of the program of the master task.

Change in outputs

- As soon as power outage is detected, the outputs are set to fallback state : either set to fallback value or retain their current value, depending on the choice made during configuration.
- On power return, the outputs remain at zero until they are updated by the task.

1.4-3 Cold restart processing



Initialization of the data and the system, which corresponds to :

- Resetting the bits, the image of the I/O and the internal words (if the option to reset internal words on a cold restart is selected in the PLC Configuration screen, see section 1.3, part D). If the Reset option for %MW is not active and the %MWi are saved in the Flash EPROM internal memory (TSX37), they are retrieved on a cold restart.
- Initialization of system bits and words.
- Initialization of function blocks based on configuration data.
- The tasks, other than the master task, are deactivated until the end of the first scan of the master task.
- Positioning of the Grafcet on the initial steps.

Cold restart processing

In the event of a cold restart, if the user requires a particular processing operation in relation to the application, bit %S0 can be tested (if %S0 = 1, then cold restart processing occurs). On a cold restart, **the PLC either does or does not continue execution**, depending on the choice of the user defined during configuration (RUN AUTO parameter).

Change in outputs

- As soon as a power outage is detected, the outputs are set to fallback state. They are either set to fallback value, or the current value is maintained, depending on the choice made during configuration.
- On power return, the outputs remain at zero until they are updated by the task.

1.5 Single task software structure

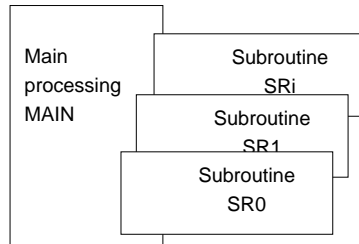
1.5-1 Presentation of the master task

The program of a single task application is associated with a single user task : the master task, MAST.

The program associated with the master task is structured into several program modules. Depending on whether or not Grafcet is used, there are two alternatives :

No Grafcet:

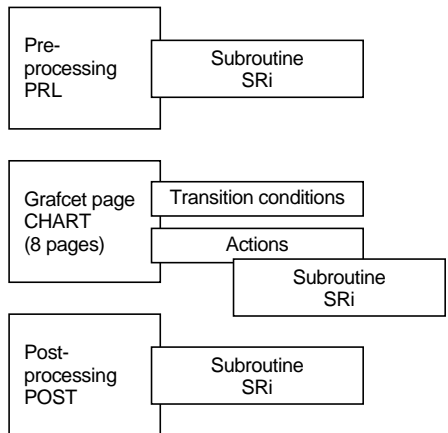
- main processing (MAIN)
- subroutine SR_i (i=0 to 253)
The subroutine modules are programmed as separate entities, calls to subroutines being performed during main processing or from other subroutines (8 levels of subroutines are possible).



Either cyclic or periodic execution of the master task can be chosen (during configuration).

With Grafcet :

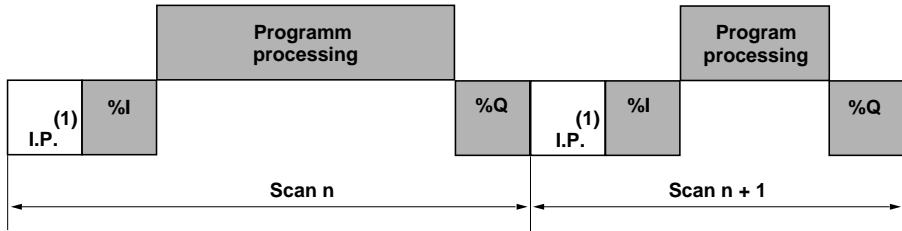
- preprocessing (PRL). This is executed before the Grafcet chart,
- Grafcet (CHART) : transition conditions associated with the transitions and actions associated with the steps are programmed in the Grafcet pages,
- post-processing (POST). This is executed after the Grafcet chart,
- subroutine SR_i (i = 0 to 253). The subroutine modules are programmed separately, calls to subroutines being made during preprocessing or post-processing, in the actions associated with the steps or from other subroutines (8 levels maximum).



Either cyclic or periodic execution of the master task can be selected (during configuration).

1.5-2 Cyclic execution

This type of operation corresponds to normal execution of the PLC scan (default option). It consists of consecutively linking the cycles of the master task (MAST). After updating the outputs, the system performs its specific processing operations and then links to another task scan.



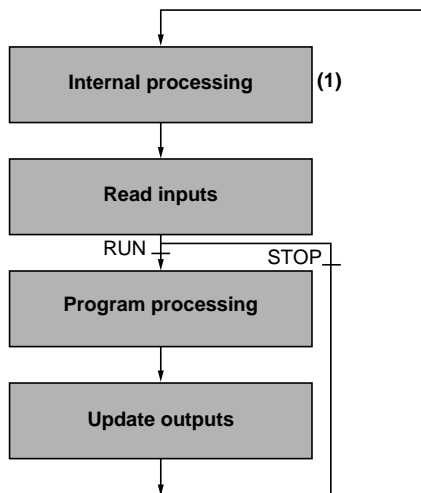
I.P. Internal processing : the system implicitly monitors the PLC (management of system bits and words, updating current values of the real-time clock, updating status indicator lamps, detection of RUN/STOP changes, etc) and processes requests from the terminal (modifications and animation),

%I Read inputs : writes to memory the status of the information on the inputs of discrete and application-specific modules associated with the task,

Program processing : execution of the application program, written by the user.

%Q Update outputs : writes output bits or words associated with discrete and application-specific modules associated with the task depending on the status defined by the application program.

Operating scan cycle/cycle monitoring



PLC in RUN : the processor performs, in order, internal processing, reads the inputs, processes the application program and updates the outputs.

PLC in STOP : the processor performs

- internal processing,
- reads the inputs,
- and depending on the configuration chosen :

- **fallback mode** : the outputs are set to "fallback" position,
- **maintain mode** : the outputs maintain their current values.

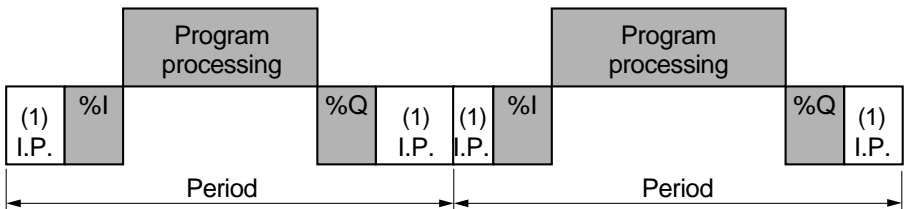
Cycle monitoring : the cycle is monitored by a watchdog. See section 1.5-4.

(1) In the case of the TSX 57, internal processing is carried out in parallel with I/O processing.

1.5-3 Periodic execution

In this operating mode, reading inputs, application program processing and updating output is performed periodically according to a period of time defined during configuration (from 1 to 255 ms).

At the start of the PLC scan, a timer, whose current value is initialized to the period defined during configuration, starts to count down. The PLC scan must finish before expiry of this timer, which then launches a new scan.



I.P. Internal processing : the system implicitly monitors the PLC (management of system bits and words, updating current values of the real-time clock, updating status indicator lamps, detection of RUN/STOP changes, etc) and processes requests from the terminal (modifications and animation).

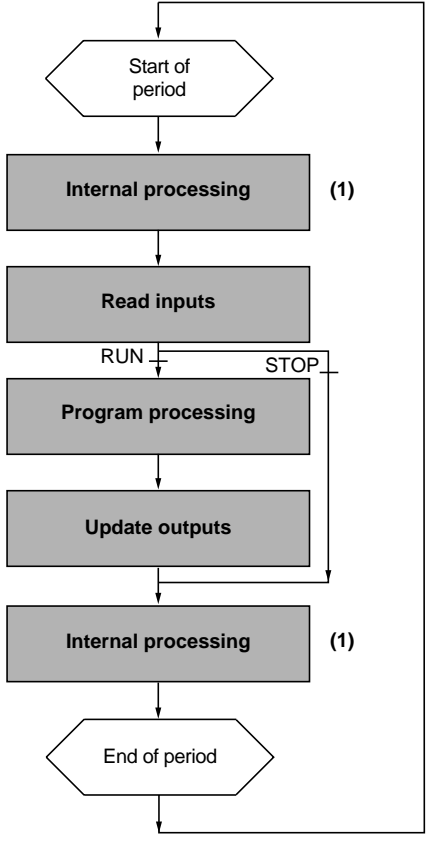
%I Read inputs : writes to memory the status of the information on the inputs of discrete and application modules associated with the task.

Program processing : execution of the application program, written by the user.

%Q Update outputs : writes output bits or words associated with discrete and application-specific modules, according to the status defined by the application program.

(1) In the case of the TSX 57, internal processing is carried out in parallel with I/O processing.

Operating scan cycle/monitoring



PLC in RUN : the processor performs, in order, internal processing, reads inputs, processes the application program and updates outputs.

If the period is not yet over, the processor completes its operating cycle until the end of the period, via internal processing.

If the operating time becomes longer than that assigned to the period, the PLC indicates a period overrun by setting system bit %S19 of the task to 1, processing continues and is fully executed (it must not, however, exceed the time limit of the watchdog). The following scan cycle is linked, after implicit writing of the outputs in the current cycle.

PLC in STOP : the processor performs

- internal processing,
- reads inputs,
- and depending on the configuration chosen :
 - **fallback mode** : the outputs are set to "fallback" position,
 - **maintain mode** : the outputs maintain their current values.

Cycle monitoring :

2 controls are performed :

- period overrun,
- via the watchdog,
(see section 1.5-4).

(1) In the case of the TSX 57, internal processing is carried out in parallel with I/O processing.

1.5-4 Monitoring scan time

Software watchdog (periodic or cyclic operation)

The execution time of the master task, whether in cyclic or periodic operation, is controlled by the PLC (watchdog) and should not exceed the value Tmax defined during configuration (250 ms default, 500 ms maximum).

In the event of overrun, the application is declared faulty, which stops the PLC immediately (on the TSX 37 the alarm output %Q2.0 is set to 0 if it has been configured, on the TSX 57, the alarm relay on the supply is set to 0).

Bit %S11 is used to monitor execution of this task.

It indicates a watchdog overrun. It is set to 1 by the system when the scan time becomes longer than the watchdog.

On the TSX 57, the watchdog value must be longer than the period.

In periodic operation, an additional control is used to detect period overrun :

- %S19 : indicates a period overrun. It is set to 1 by the system when the scan time becomes longer than the task period.
- %SW0 : this word contains the value of the period (in ms). It is initialized at a cold restart by the value defined during configuration. It can be modified by the user.

Using the execution time of the master task

The following system words provide data on the scan time :

- %SW30 contains the execution time of the last scan.
- %SW31 contains the execution time of the longest scan.
- %SW32 contains the execution time of the shortest scan.

Note :

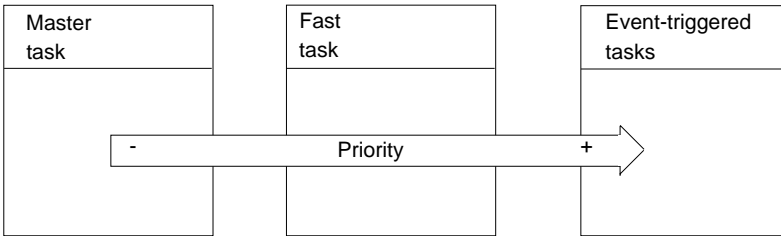
This data can also be accessed from the configuration editor if requested.

1.6 Multitask software structure

1.6-1 Description

The structure in tasks of such an application is as follows :

- The master task, MAST, which is always present and can be cyclic or periodic.
- The fast task, FAST, which is optional and always periodic.
- Event-triggered processing operations EVT_i, called up by the system when an event appears on an I/O module. This processing is optional and is used by applications requiring short response times to perform operations on I/O.



Management of tasks :

The master task is active by default. The fast task is active by default if it is programmed. The event-triggered task is activated on appearance of the event with which it has been associated.

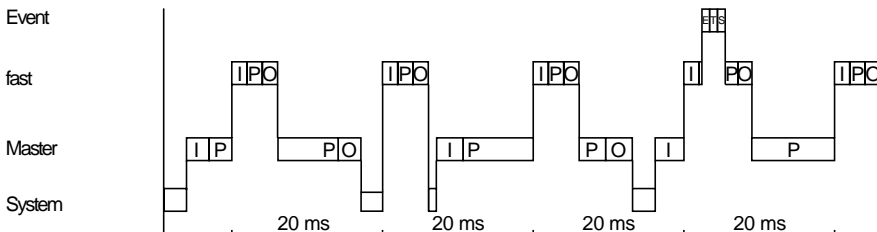
If an event occurs, or at the start of the fast task cycle, the task stops current execution of lower priority tasks in order to execute its own processing. The interrupted task takes over when processing of the priority task is completed.

The execution of fast and event-triggered tasks can be controlled by the program using system bits :

- %S30 is used to activate or deactivate the master task, MAST.
- %S31 is used to activate or deactivate the fast task, FAST.
- %S38 is used to activate or deactivate event-triggered tasks EVT_i.

Example of multitask processing

- cyclic master task
- fast task with 20 ms period
- event-triggered task.



1.6-2 Master task

This task, which has the lowest priority, manages the majority of the application program. The MAST task is organized according to the model described in the previous section : implicit reading of inputs, execution of the application program and implicit writing of outputs.

Whether the operating mode is periodic or cyclic, the task is monitored by a watchdog which is used to detect an abnormal duration of the application program. In the event of an overrun, system bit %S11 is set to 1 and the application is declared as faulty, which stops the PLC.

The system bit %S30 enables or inhibits the master task.

1.6-3 Fast task

This task, which is higher priority than the master task MAST, is periodic in order to leave time for execution of the lower priority task.

In addition, the processing operations which are associated with it should therefore be short, to avoid adversely affecting the master task. Like the master task, the associated program consists of a main module and subroutines.

The period of the fast task, FAST, is set during configuration, from 1 to 255 ms. This can be defined as longer than that of the master task, MAST, in order to adapt it to periodic processing operations which are slow but have higher priority. The program executed should however remain short to avoid overrun of lower priority tasks.

The fast task is monitored by a watchdog which is used to detect an abnormal duration of the application program. In the event of an overrun, system bit %S11 is set to 1 and the application is declared as faulty, which stops the PLC.

Control of the fast task

System word %SW1 contains the value of the period. It is initialized at a cold restart by the value defined during configuration and can be modified by the user via the program or the terminal.

System bits and words are used to monitor execution of this task :

- %S19 : indicates a period overrun. It is set to 1 by the system when the scan time becomes longer than the task period.
- %S31 : is used to enable or inhibit the fast task. It is set to 0 by the system at a cold restart of the application, at the end of the first scan of the master task. It is set to 1 or 0 to enable or inhibit the fast task.

Display of the fast task execution times

The following system words provide data on scan times :

- %SW33 contains the execution time of the last scan.
 - %SW34 contains the execution time of the longest scan.
 - %SW35 contains the execution time of the shortest scan.
-

1.6-4 Assigning I/O channels to the master and fast tasks

In addition to the application program, the MAST and FAST tasks execute system functions linked to the management of implicit I/O which are associated with them.

Associating a channel or a group of channels with a task is defined in the configuration screen for the corresponding module. The default associated task is the MAST task.

Since the modularity of discrete modules is 8 consecutive channels (channels 0 to 7, channels 8 to 15, etc), I/O can be assigned in groups of 8 channels, either to the MAST or FAST task. For example, it is possible to assign the channels of a 28 I/O module in the following way :

- Inputs 0 to 7 assigned to the MAST task.
- Inputs 8 to 15 assigned to the FAST task.
- Outputs 0 to 7 assigned to the MAST task.
- Outputs 8 to 11 assigned to the FAST task.

Each channel of a counter module can be assigned either to the MAST or FAST task. For example, for a 2 channel counter module, it is possible to assign :

- channel 0 to the MAST task and,
- channel 1 to the FAST task.

The channels of TSX 37 analog input modules must be assigned to the MAST task. However, it is possible to assign analog output channels to either the MAST or FAST task, with 2 channel modularity. For example, for a module with 4 analog outputs, it is possible to assign :

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

The channels of TSX 57 analog I/O modules can be assigned to the MAST or FAST task. Each channel of the isolated analog I/O modules (4 isolated channels) is assigned on an individual basis. A 4-channel modularity is used for other modules.

1.6-5 Event-triggered tasks

Event-triggered processing is used to reduce the response time of the software on command events.

Command events

These are **external events** associated with applications. The appearance of such an event diverts the application program to the processing operation associated with the I/O channel which caused the event. The inputs (%I, %IW, %ID) associated with the I/O channel which triggered the event are updated by the system before calling up event processing. It is possible to configure :

- 8 events in a TSX 37-10 PLC (EVT0 to EVT7).
- 16 events in a TSX 37-21/22 PLC (EVT0 to EVT15).
- 32 events in a TSX 57-10 PLC (EVT0 to EVT31).
- 64 events in a TSX 57-20 PLC (EVT0 to EVT63).

The association between a channel and an event number is created in the channel configuration screen.

On the TSX 37, event processing can be triggered by :

- Inputs 0 to 3 of module in position 1, on a rising or falling edge.
- The counter channel(s) of counter modules.
- The counter channels of module 1 (if this is configured as a counter).
- Reception of a telegram in a TSX 37-21/22 fitted with a TSX FPP20 module.

On the TSX 57, event processing can be triggered by :

- The 16 inputs of DEY 16 FK modules.
- The counter channels.
- The channels of CAY axis control modules.
- The channels of CFY stepper axis control modules.
- Communication channels "FPP20".

Management of event-triggered tasks

Event-triggered processing can be globally enabled or inhibited by the application program, using system bit %S38. If one or more events occur while they are inhibited, the associated processing operations are lost.

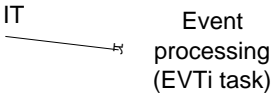
Two PL7 language instructions MASKEVT() and UNMASKEVT(), used in the application program, also allow masking or unmasking of event-triggered processing. If one or more events occur while they are masked, they are saved by the system and the associated processing will only be carried out after unmasking.

The 8 possible command events with a TSX 37-10 PLC all have the same level of priority. Thus, one event processing operation cannot be interrupted by another. In a TSX 37-21/22 or a TSX 57 PLC, there are 2 levels of priority for command events : event 0 (EVT0) is higher priority than the other events.

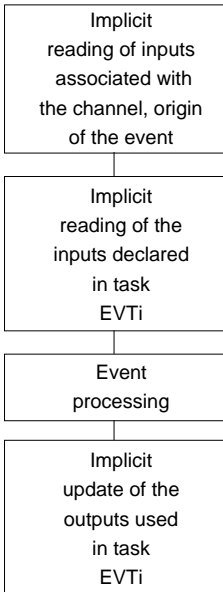
I/O exchanges in event-triggered tasks

With each event-triggered task it is possible to use I/O channels other than those relating to the event. Exchanges are then performed implicitly by the system before (%I) and after (%Q) application processing. These exchanges can relate to a single channel (example of counter module) or to a group of channels (discrete module). In the second case, if the processing modifies, for example, outputs 2 and 3 of a discrete module, it is the image of outputs 0 to 7 which will be transferred to the module.

Summary of exchanges and processing operations



The appearance of such an event diverts the application program towards the processing operation which is associated with the I/O channel which caused the event :



- All the inputs associated with the channel which caused the event are read automatically.
- All the inputs declared by the user in task EVTi are read.
- Processing should be as short as possible.
- All the outputs used by the user in task EVTi are updated. The outputs associated with the channel which caused the event should also be declared so they can be updated.

Notes

On the TSX 37, analog input modules, which can only be used in the MAST task, should not be exchanged in event processing.

The exchange of I/O, associated with the EVTi task and used by the program, is performed channel by channel (for counter modules) or in groups of channels (for discrete modules). For this reason, if processing modifies, for example, outputs 2 and 3 of a discrete module, it is the image of outputs 0 to 7 which will be transferred to the module.

On the TSX 37, for each event-triggered processing operation, it is possible to declare a maximum of exchanges for 2 input modules (before processing of the event) and 2 output modules (after processing of the event).

The inputs exchanged (and the group of associated channels) during event-triggered processing are updated (loss of log values and, consequently, edges). Thus, edges should not be tested on these inputs in the master (MAST) or fast (FAST) tasks.

On the TSX 57, depending on the processor used, the number of exchanges used is limited :

No. of exchanges which can be used in EVT's by processor	P57-10 (32 EVT's)	P57-20 (64 EVT's)
Max. no. of discrete exchanges	32 exchanges	128 exchanges
Max. no. of analog exchanges	8 exchanges	16 exchanges
Max. no. of other application-specific exchanges	4 exchanges	16 exchanges

For discrete I/O, an exchange involves a group of 8 channels. It is generated when the inputs of a group of 8 channels are used (other than the group of channels which generate the event) and when writing the outputs of a group of 8 channels.

For analog or other application-specific I/O, an exchange is generated when the inputs of a channel are used (other than the channel which generates the event) and when writing the outputs of a channel.

Display of the number of events processed

Word %SW48 shows the number of events processed. This word is initialized to 0 at a cold restart, then incremented by the system when an event is launched. This word can be modified by the user.

%S39 indicates loss of event.

Note :

A summary of the operations to be performed to program events is given in part F, section 5.2.

2.1 Presentation of Ladder language

2.1-1 Principle

Programs written in Ladder language consist of a series of rungs which are executed sequentially by the PLC.

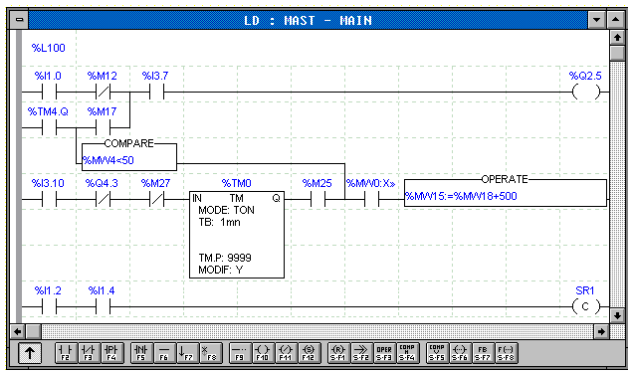
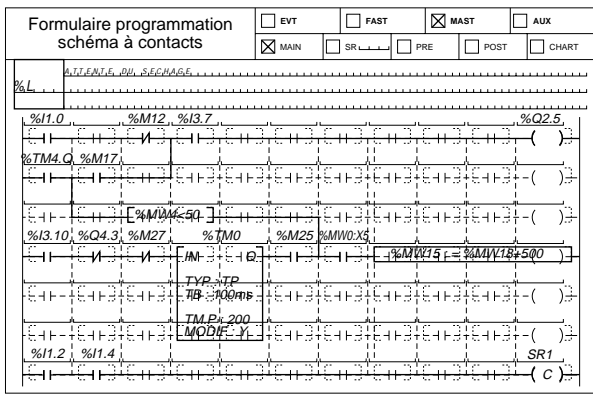
A rung consists of a set of graphic elements bounded on the left and right by power rails. They represent :

- The PLC I/O (pushbuttons, sensors, relays, indicator lamps, etc).
- Standard control system functions (timers, counters, etc).
- Arithmetic, logic and specific operations.
- The internal variables of the PLC.

The graphic elements are interconnected by horizontal and vertical links.

Each rung consists of a maximum of 7 lines and 11 columns, which are divided into 2 zones :












- The test zone which contains the conditions necessary to execute the actions.
- The action zone which contains the actions to be executed according to the results of the test zone.



2.1-2 Graphic elements


Basic elements

They all occupy a single cell (1 line high, 1 column wide).


Designation	Symbol	Function	
Test elements	• Normally open contact		Contact closed when the bit object which controls it is at 1.
	• Normally closed contact		Contact closed when the bit object which controls it is at 0.
	• Edge detection contacts		Rising edge : contact closed when the bit object which controls it changes from 0 to 1.
			Falling edge : contact closed when the bit object which controls it changes from 1 to 0.
Link elements	• Horizontal links		Used to link test and action graphic elements between the two power rails in series.
	• Vertical links		Used to link test and action graphic elements in parallel.
Action elements	• Direct coil		Sets the associated bit object to the value of the result of the test zone.
	• Negated coil		Sets the associated bit object to the inverse value of the result of the test zone.
	• Latch coil		Sets the associated bit object to 1 when the result of the test zone is at 1.
	• Unlatch coil		Resets the associated bit object to 0 when the result of the test zone is at 1.
	• Conditional JUMP to another rung	->> %Li	Allows connection to a labelled rung, either upstream or downstream. Jumps are only effective within the same programming entity (main program, subroutine, etc).
	• Transition condition coil		Offered in Grafcet language, used when programming conditions associated with transitions, to move to the next step.

If a jump is activated :


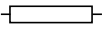
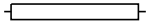
- Scanning of the current rung is interrupted.
- The requested labelled rung is executed.
- The part of the program between the jump action and the designated rung is not executed.

Designation		Symbol	Function
Action elements (continued)	<ul style="list-style-type: none"> Subroutine call coil (CALL) 		<p>Allows connection at the beginning of subroutines when the result of the test zone is at 1.</p> <p>If a subroutine is called :</p> <ul style="list-style-type: none"> Scanning of the current rung is interrupted. The subroutine is executed. Scanning of the interrupted rung is resumed.
	<ul style="list-style-type: none"> Subroutine return 	<RETURN>	Reserved for subroutines SR, allows return to the calling module when the result of the test zone is at 1.
	<ul style="list-style-type: none"> Stop program 	<HALT>	Stops execution of the program when the result of the test zone is at 1.

Function blocks

Designation		Symbol	Function
Test elements	<ul style="list-style-type: none"> Blocks : Timer Counter Monostable Register Drum controller 		<p>Each of the standard function blocks uses I/O which allow them to be linked to other graphic elements.</p> <p>The functions of each block are described in part B.</p> <p>Size : see section 2.2-5.</p>

Operation blocks

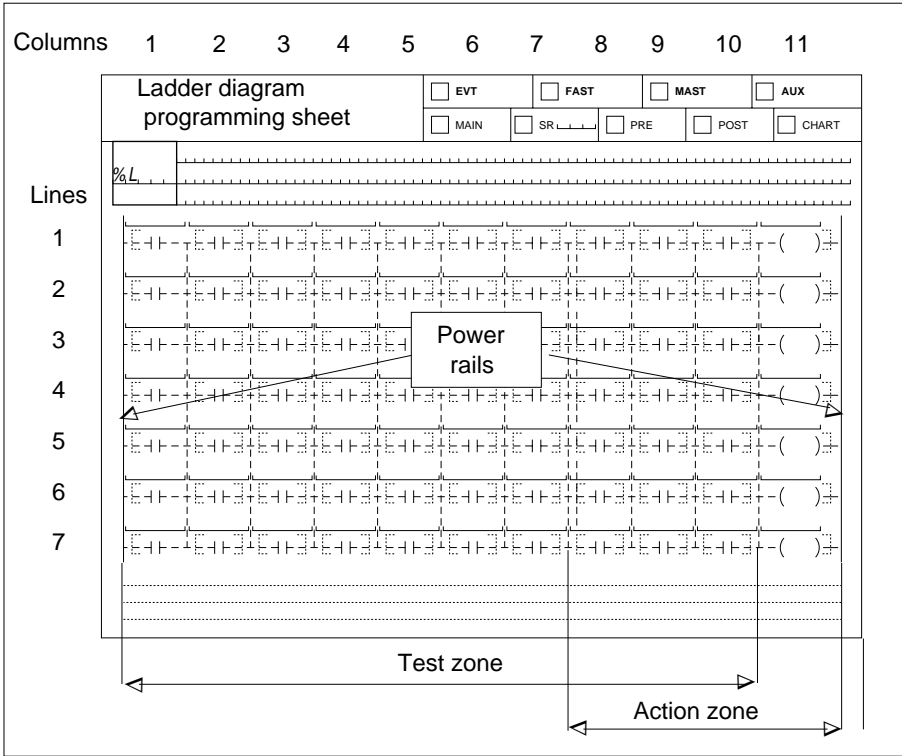
Designation		Symbol	Function
Test elements	<ul style="list-style-type: none"> Vertical comparison block 		<p>Allows comparison of 2 operands. Depending on the result, the corresponding output changes to 1.</p> <p>Size : 2 columns/4 lines.</p>
	<ul style="list-style-type: none"> Horizontal comparison block 		<p>Allows comparison of 2 operands. The output changes to 1 when the result is checked. (A block can contain up to 4096 characters).</p> <p>Size : 2 columns/1 line.</p>
Action elements	<ul style="list-style-type: none"> Operation block 		<p>Performs arithmetic, logic operations etc and uses Structured text language syntax. (A block can contain up to 4096 characters).</p> <p>Size : 4 columns/1 line.</p>

2.2 Structure of a rung

2.2-1 General

A rung is located between two power rails and consists of a set of graphic elements which are interconnected by horizontal or vertical links.

A rung contains up to 7 lines and 11 columns divided into two zones, the test zone and the action zone.



Each rung can be identified by a label and headed by a comment.

2.2-2 Labels

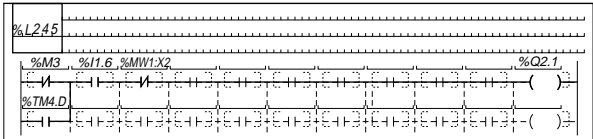
Labels are used to identify a rung within a program entity (main program, subroutine, etc), but are not compulsory.

Labels take the syntax %Li (i being 0 to 999) and are located at the top left before the power rail.

Each label can only be assigned to one rung within the same program entity.



However, a rung must be labelled to allow connection after a program jump.



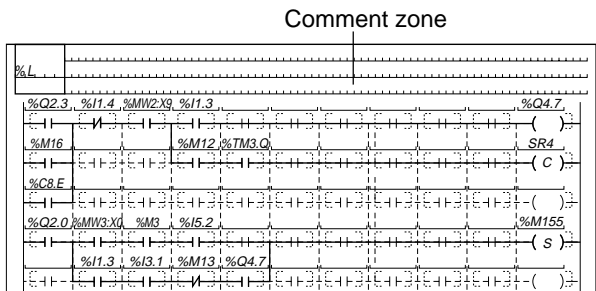
The system scans the rungs in the order in which they were entered, irrespective of the order of the label numbers.

2.2-3 Comments

The comment is integrated into the rung and contains up to 222 alphanumeric characters, framed at either end by the characters (* and *). It facilitates interpretation of the rung to which it is assigned, but is not compulsory.

Comments are displayed in the reserved zone in the upper part of the rung.

If a rung is deleted, the comment associated with it is also deleted.



Comments are stored in the PLC and can be accessed at all times by the user. They therefore use program memory.

2.2-4 Rungs

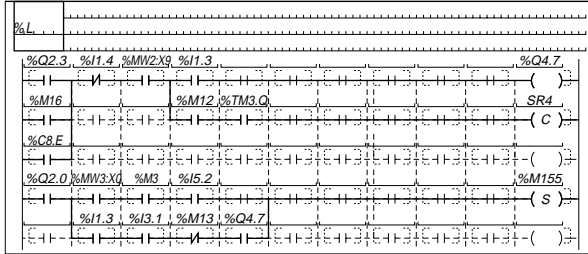
The representation of a rung is similar to that of a relay diagram.

Simple test and action graphic elements each occupy a single line and column within a rung.

All lines of contacts start on the left power rail and must finish on the right power rail.

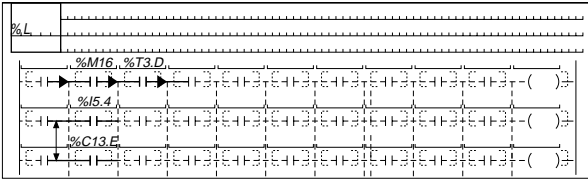
Tests are always located on columns 1 to 10.

Actions are always located on column 11.



The direction of current is the following :

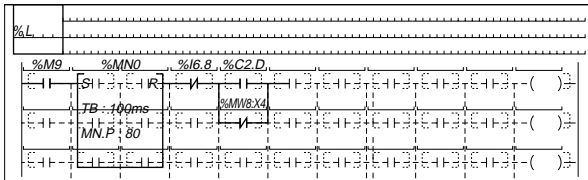
- for horizontal links, from **left to right**,
- for vertical links, in both directions.



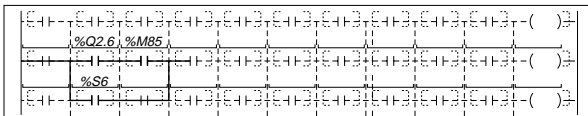
Test zone

This zone contains :

- contacts, to which all the bit objects defined above can be assigned,
- function blocks,
- comparison blocks.



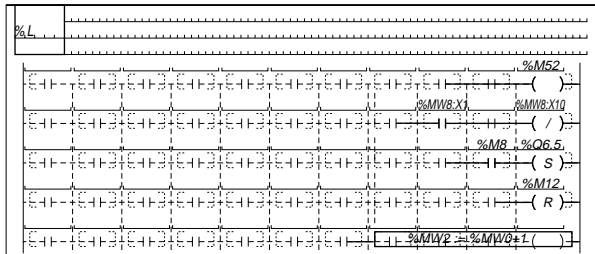
Rising and falling edges can only be associated with I/O bit objects and internal bits.



Action zone

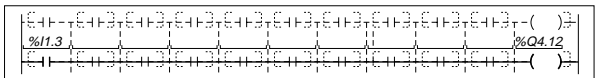
This zone contains :

- direct, negated, latch and unlatch coils, which can be assigned to any bit object which can be written by the user.
- operation blocks.
- the other "coils" :
Call, Jump, Halt, Return.

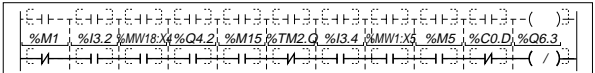


Simple rungs

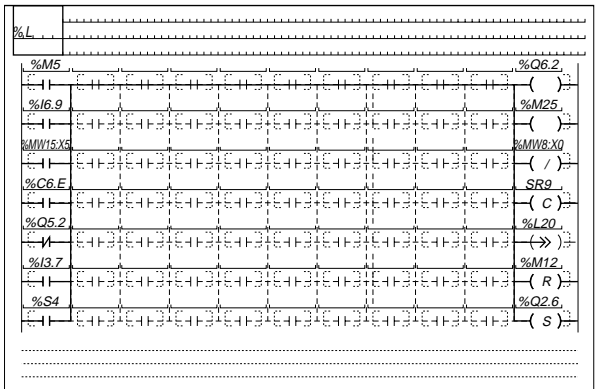
Control of a coil conditioned by the state of a contact.



Use of up to 10 contacts in series on one line.



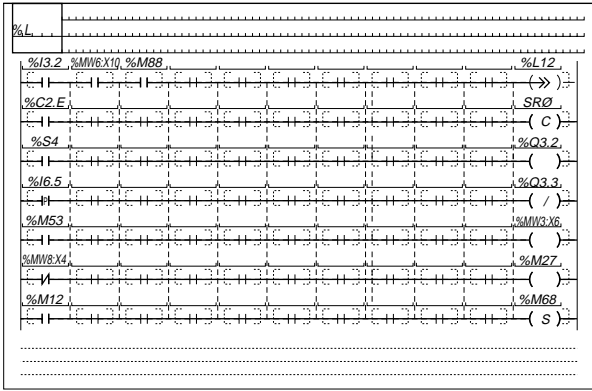
A maximum of 7 contacts can be tested in parallel on one column, and 7 coils placed in parallel.



Rung using several lines of contacts

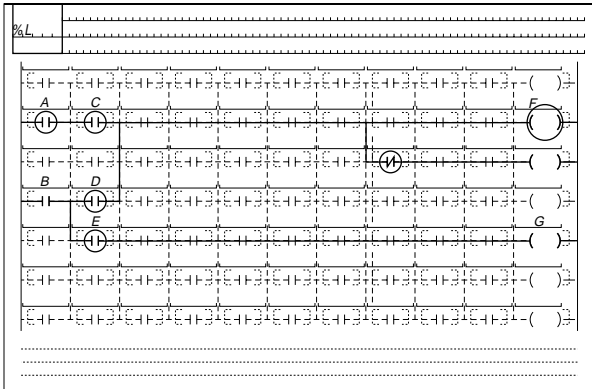
A rung can be divided into several independent lines of contacts controlling independent coils.

7 independent lines of contacts.



Rungs using the various principles which have been described.

Symbols at logic state 1 are circled. The current can pass from symbols A and C to coil F. It cannot pass from symbol C to symbols D and E. Coil G is therefore not activated.

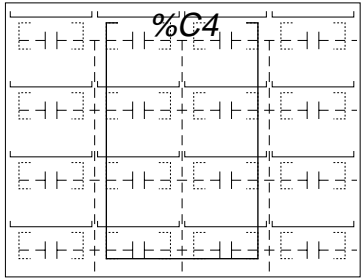


2.2-5 Rungs with function and operation blocks

- Function blocks are located in the test zone and are inserted in a rung. Four sizes of graphic are used to represent all the other function blocks of the PL7 Micro language.

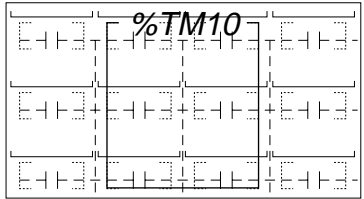
Up/down counter
"Vertical" comparison block

2 columns
4 lines



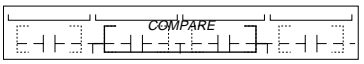
Timers
Monostable
Register
Drum controller

2 columns
3 lines

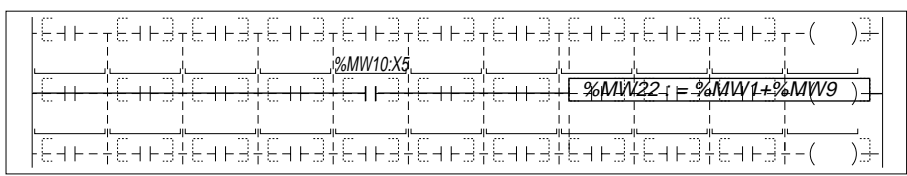


"Horizontal" comparison block

2 columns
1 line



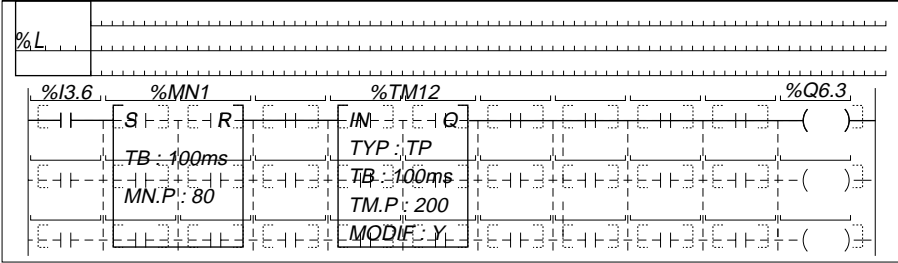
- Operation blocks are always located in the action zone. They are 1 line deep and 4 columns wide, are written in Structured text language and are always directly linked to the right power rail.



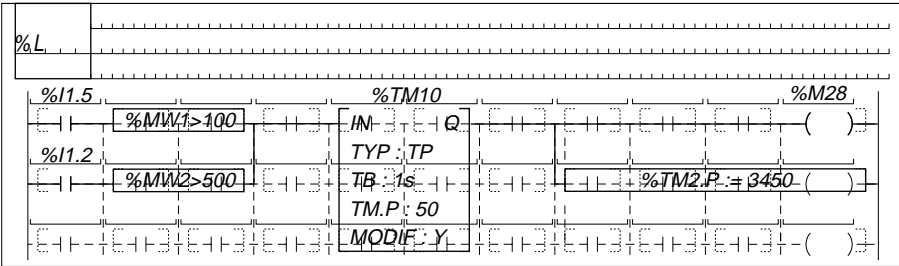
Function blocks can be "cascaded"

Like the contact type graphic elements, it is possible to combine function blocks.

Connecting function blocks in series :



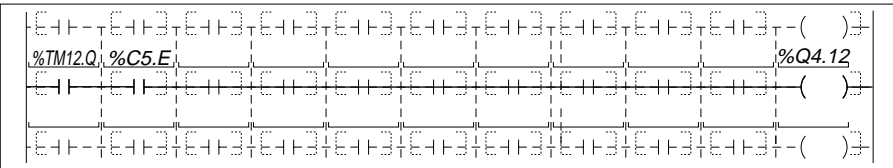
Function blocks and operation blocks can be mixed



Others possible uses of function blocks

Irrespective of the type of function block used, its input must be linked to the left power rail, either directly or via other graphic elements.

- **Outputs left open** : it is not necessary to link the outputs of function blocks to other graphic elements.
- **Testable outputs** : the outputs of function blocks are accessible to the user in the form of bit objects :



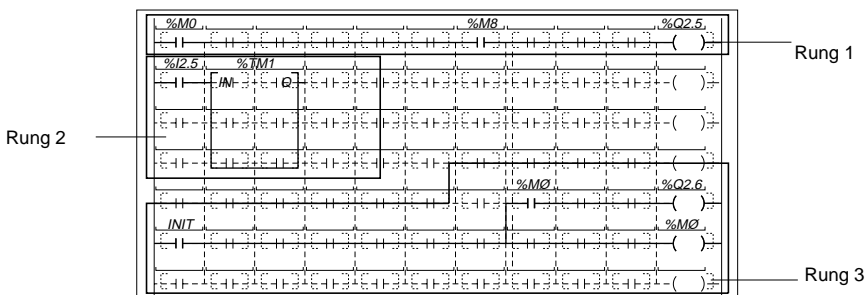
Internal variables of blocks and graphic outputs can be used remotely from another part of the program.

2.3 Rules for executing rungs

2.3-1 Principle for executing a rung

Rungs are executed rung by rung, and each rung is executed from left to right.

A rung contains graphic elements which are interconnected by horizontal and vertical links (apart from the power rail), but are independent from other graphic elements in the rung (no vertical links to the top or bottom edges of the rung).



The rung in the top left-hand corner is the first rung to be evaluated.

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

In cases where a vertical convergence link is found, the sub-rung associated with it is evaluated (following the same logic) before continuing evaluation of the rung in which it is included.

Following this order of execution, the system :

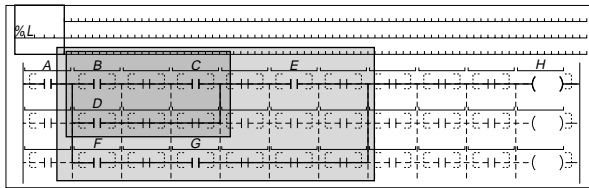
- Evaluates the logic state of each contact, according to the current value of the internal objects of the application or of the state of the inputs of the I/O modules read at the beginning of the scan.
- Executes the processing operations associated with the functions, function blocks and subroutines.
- Updates the bit objects associated with coils, (outputs of I/O modules are updated at the end of the scan).
- Goes to another labelled rung in the same program module (jumps to another rung ->>%Li), returns to the calling module <RETURN>, or stops the program <HALT>.

Note :

A rung must not contain nested rungs.

Elements in this rung are executed in the following order :

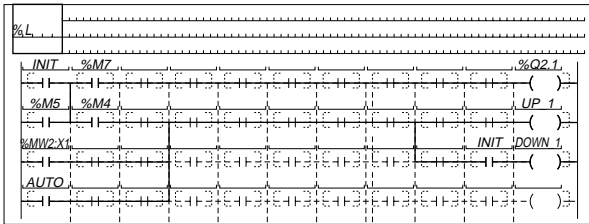
- evaluation of the rung until the 1st vertical convergence link is found, contacts A, B and C,
- evaluation of the 1st sub-rung, contact D,
- continued evaluation of the rung until the 2nd vertical convergence link is found, contact E,
- evaluation of the 2nd sub-rung, contacts F and G,
- evaluation of coil H.



Example of a "Boolean" rung

Order of evaluation :

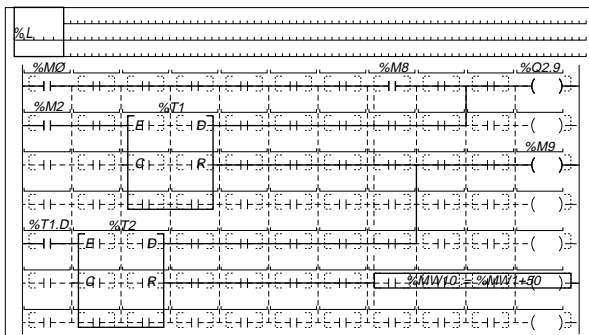
- coil 1 : INIT, %M5, %M7, %Q2.1,
- coil 2 : %M4, %MW2:X1,AUTO, UP_1,
- coil 3 : INIT, DOWN_1.



Example of rung containing blocks

Order of evaluation :

- coil 1 : %M0, %M8, %M2, %T1, %Q2.9,
- coil 2 : %T1.R, %T2, %M9,
- operation block.



3.1 Presentation of Instruction list language

3.1-1 Principle

A program written in Instruction list language is composed of a series of instructions executed sequentially by the PLC.

```

IL: FAST - MAIN
LD %I1.0
ANDN %M12
OR( %TM0 Q
AND %M17
)
AND %I3.7
ST %Q2.5

I%L1:
LD %I3.5
ANDN %Q2.3
ANDN %M2.7
IN %TM0
LD %TM0 Q
AND %M25
[%MW15 := %MW18 + 500]

I%L10:
LD %I1.2
AND %I1.4
ST %Q2.2

I%L13:
LD %I1.3
AND %M25
ST %Q2.1

```

Example of an instruction : LD %I1.0

└───┘
└───┘
 Instruction code Operand

Each instruction is composed of an instruction code and an operand.

These instructions act on :

- The I/O of the PLC (pushbuttons, detectors, relays, indicator lights etc).
- Standard control system functions (timers, counters etc).
- Arithmetic and logic operations and transfer operations.
- The internal variables of the PLC.

There are two types of instruction :

- Test instructions which contain the conditions necessary to execute an action, eg : LD, AND, OR etc.
- Action instructions which validate the result following a test sequence, eg : ST, STN, R, etc.

3.1-2 Instructions

Basic instructions

(For further information on each instruction, see part B).

Designation	Instructions	Equivalent functions
Test instructions	• LD, LDN, LDR, LDF	
	• AND, ANDN, ANDR, ANDF	
	• OR, ORN, ORR, ORF	
	• AND(, OR((8 levels of parentheses)	
	• XOR, XORN, XORR, XORF exclusive OR	
	• MPS MRD MPP	
• N	Negation	
Action instructions	• ST, STN, S, R	
	• JMP, JMPC, JMPCN	Used to jump (unconditional, conditional on a Boolean result at 1 or conditional on a Boolean result at 0) to a labelled instruction, either upstream or downstream.
	• SRn RET, RETC, RETCN	Used to jump to the beginning of a subroutine. Subroutine return (unconditional, conditional on a Boolean result at 1 or conditional on a Boolean result at 0).
	• END, ENDC, ENDCN	End of program (unconditional, conditional on a Boolean result at 1 or conditional on a Boolean result at 0).
	• HALT	Execution of the program is stopped.

Instructions on function blocks (see part B, section 1.3)

Designation	Instructions	Functions
Test elements	<ul style="list-style-type: none"> • Blocks : Timer Counter Monostable Register Drum controller 	There are instructions for controlling each of the standard function blocks. A structured form is used to directly "wire" the I/O of the function blocks.

Numeric instructions (see part B)

Designation	Instructions	Functions
Test elements	<ul style="list-style-type: none"> • LD[.....] AND[.....] OR[.....] <p>Example : LD[%MW10<1000]</p>	Used to compare two operands (see part B, section 1.4-2). The output changes to 1 when the result is checked. Result at 1 when %MW10<1000.
Action elements	<ul style="list-style-type: none"> • [.....] <p>Example : [%MW10:=%MW0+100]</p>	Perform arithmetic, logic operations etc. Use Structured text language syntax (see part B). The result of the operation %MW0+100 is placed in internal word %MW10.

3.2 Program structure

3.2-1 General

Like Ladder language, instructions are organized into sequences of instructions (equivalent to a rung), called a sequence. Each sequence is composed of one or more test instructions. The result of these instructions is applied to one or more action instructions.

An instruction occupies up to one line.

Each sequence starts with an exclamation mark (generated automatically). It can include a comment and be identified by a label.

```
!           (*Waiting for drying*)
           %L2:
           LD      %I0.1
           AND     %M10
           ST      %Q2.5
```

3.2-2 Comments

Comments can be integrated at the beginning of a sequence and can occupy up to 3 lines (ie. 222 alphanumeric characters), framed at either end by the characters (* and *). They facilitate interpretation of the sequence to which they are assigned but are not compulsory.

Comments are displayed only from the first line of the sequence.

If a sequence is deleted, its associated comment is also deleted.

Comments are stored in the PLC and can be accessed at all times by the user. They therefore use program memory.

3.2-3 Labels

Labels are used to identify a sequence in a program entity (main program, subroutine, etc) but are not compulsory.

Labels take the syntax %Li (where i is 0 to 999) and are located at the beginning of a sequence.

A label can only be assigned to a single sequence within the same program entity.

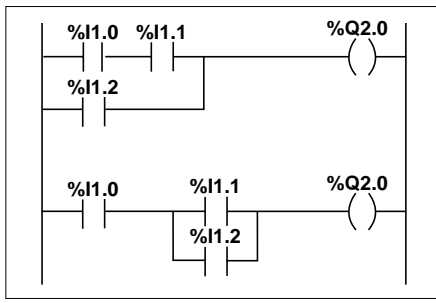
However, a sequence must be labelled to achieve a connection following a program jump.

The system scans the sequences in the order in which they were entered, irrespective of the order of the label numbers.

3.2-4 Using parentheses

It is possible to use parentheses with the instructions AND and OR. These parentheses are used for simple creation of Ladder diagrams. An opening parenthesis is associated with the instruction AND or OR. A closing parenthesis represents an instruction and is compulsory for each opening parenthesis.

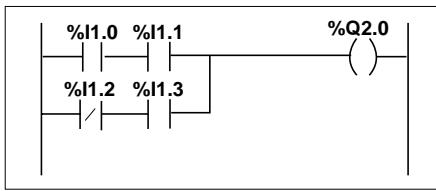
Example : AND(



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

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

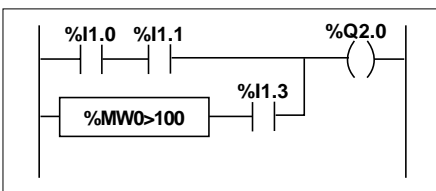
Example : OR(



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

The following modifiers can be associated with parentheses :

- N negation, eg : AND(N or OR(N
- F Falling edge, eg : AND(F or OR(F
- R Rising edge, eg : AND (R or OR (R
- [comparison

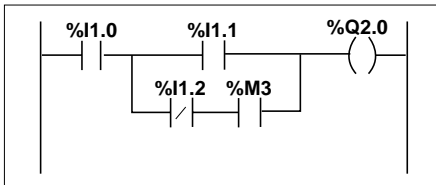


```
LD      %I1.0
AND     %I1.1
OR(     [%MW0>100]
AND     %I1.3
)
ST      %Q2.0
```

Nesting parentheses

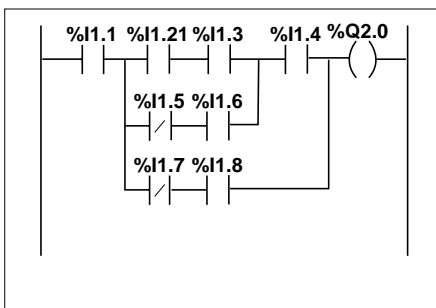
Up to eight levels of parentheses can be nested.

Example



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

Example



```
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
```

Note :

- Each opening parenthesis **must** be followed by a closing parenthesis.
- The labels %Li: must not be placed in expressions between parentheses. This also applies to jump instructions, JMP, and call subroutine instructions, SRI.
- Assignment instructions, ST, STN, S and R must not be programmed between parentheses.

3.2-5 MPS, MRD and MPP instructions

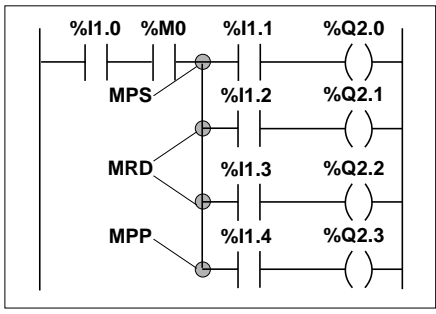
These three types of instruction are used to process the routing to the coils. They use a buffer known as a stack which is capable of storing up to 3 Boolean data bits.

The instruction MPS (Memory PuSh) stores the result of the last test instruction at the top of the stack and shifts the other values towards the bottom of the stack.

The instruction MRD (Memory ReaD) reads the top of the stack.

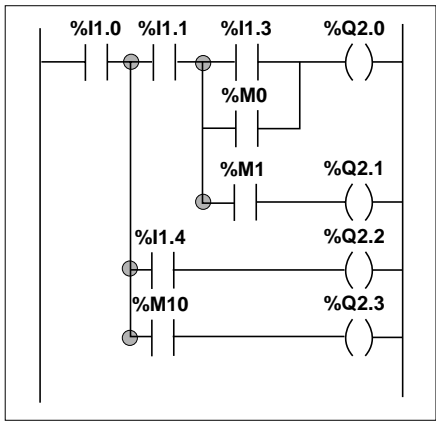
The instruction MPP (Memory PoP) reads and retrieves the top of the stack, and shifts the other values towards the top of the stack.

Examples :



```

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
    
```



```

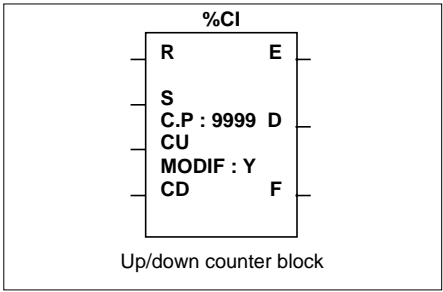
LD      %I1.0
MPS
AND    %I1.1
MPS
AND(
OR
)
ST     %Q2.0
MPP
AND    %M1
ST     %Q2.1
MRD
AND    %I1.4
ST     %Q2.2
MPP
AND    %M10
ST     %Q2.3
    
```

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

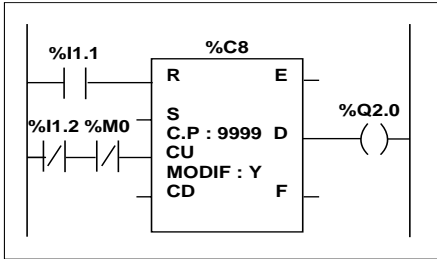
3.2-6 Principles for programming predefined function blocks

Control system function blocks can be programmed in two different ways :

- with instructions specific to each function block (eg : CU %Ci). This is the simplest and most direct way.
- with block structure instructions BLK,OUT_BLK and END_BLK.



Principle of direct programming



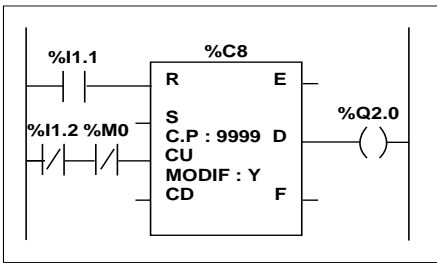
```
LD    %I1.1
R     %C8
LDN   %I1.2
ANDN  %M0
CU    %C8
LD    %C8.D
ST    %Q2.0
```

The instructions control the inputs of the blocks (eg : CU).
The outputs can be accessed in the form of bits (eg : %C8.D).

Principle of structured programming

This type of programming uses a sequence of instructions framed by the following instructions :

- **BLK** indicates the start of the block.
- **OUT_BLK** is used to directly wire the outputs of the block.
- **END_BLK** indicates the end of the block.



```
BLK   %C8
LD    %I1.1
R     %C8
LDN   %I1.2
ANDN  %M0
CU    %C8
OUT_BLK
LD    D
ST    %Q2.0
END_BLK
```

Structured programming requires the additional instructions BLK, OUT_BLK and END_BLK, and therefore needs more memory compared with direct programming. It should, however, be used if similarity with reversible programs for TSX 07 nano PLCs is required.

3.3 Rules for executing Instruction list programs

Instruction list programs are executed sequentially instruction by instruction.

The first instruction in a series of instructions must always be either an LD instruction or an unconditional instruction (eg : JMP).

All instructions (except LD and the unconditional instructions) use the Boolean result of the preceding instruction.

Example :

```
LD  %I1.1  Boolean result = state of bit %I1.1.
AND %M0    Boolean result = AND of the preceding Boolean result and the state of
           bit %M0.
OR   %M10  Boolean result = OR of the preceding Boolean result and the state of
           bit %M10.
ST   %Q2.0 %Q2.0 takes the state of the preceding Boolean result.
```

Parentheses can be used to modify the order in which Boolean results are taken into account :

Example :

```
LD  %I1.1  Boolean result = state of bit %I1.1.
AND %M0    Boolean result = AND of the preceding Boolean result and the state of bit
           %M0.
OR(  %M10  Boolean result = state of bit %M10.
AND  %I1.2 Boolean result = AND of the preceding Boolean result and the state of bit
           %I1.2.
)    Boolean result = OR of the preceding Boolean result and of the
     Boolean result of the instruction located before the instruction with
     parentheses.
ST   %Q2.0 %Q2.0 takes the state of the preceding Boolean result.
```

Sequencing of instructions can be modified by the jump (JMP) and call subroutine instructions.

Example :

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

```

Jump to label %L10 if %M0=1

4.1 Presentation of Structured Text language

4.1-1 Principle

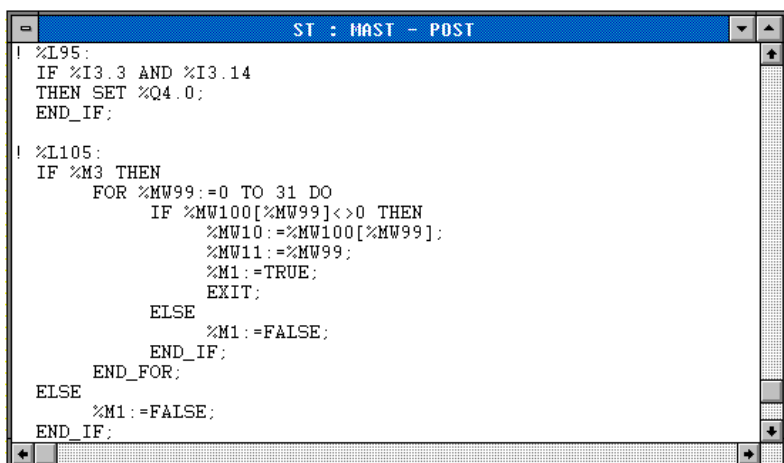
Structured Text language is used to create programs by writing programming lines made up of alphanumeric characters. This language can only be used by **PL7 Junior** software on TSX 57 PLCs.

The ST statement constitutes the basic unit of Structured Text language, and a series of statements is used to define a program.

The main instructions in Structured Text language are as follows :

- bit instructions,
- arithmetic and logic instructions on words and double words,
- arithmetic instructions on floating points,
- numerical comparisons on words, double words and floating points,
- numerical conversions,
- instructions on bit, word, double word and floating point tables,
- character string instructions,
- alphanumerical comparisons,
- time management instructions,
- program instructions,
- control instructions,
- standard function block instructions,
- explicit exchange instructions,
- application-specific instructions (communication, PID control, etc).

Example :



```
ST : MAST - PDST
| %L95:
IF %I3.3 AND %I3.14
THEN SET %Q4.0;
END_IF;

| %L105:
IF %M3 THEN
    FOR %MW99:=0 TO 31 DO
        IF %MW100[%MW99]<>0 THEN
            %MW10:=%MW100[%MW99];
            %MW11:=%MW99;
            %M1:=TRUE;
            EXIT;
        ELSE
            %M1:=FALSE;
        END_IF;
    END_FOR;
ELSE
    %M1:=FALSE;
END_IF;
```

4.1-2 Instructions

Bit instructions

Description	Function
:=	Bit assignment
OR	Boolean OR
AND	Boolean AND
XOR	Exclusive Boolean OR
NOT	Inversion
RE	Rising edge
FE	Falling edge
SET	Set to 1
RESET	Reset to 0

Numerical comparisons on words, double words and floating points

Description	Function
<	Strictly less than
>	Strictly greater than
<=	Less than or equal to
>=	Greater than or equal to
=	Equal to
<>	Different from

Bit tables

Description	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 a bit table to a bit table
AND_ARX	AND between two tables
OR_ARX	OR between two tables
XOR_ARX	Exclusive OR between two tables
NOT_ARX	Negation on a table
BIT_W	Copy a bit table to a word table
BIT_D	Copy a bit table to a double word table
W_BIT	Copy a word table to a bit table
D_BIT	Copy a double word table to a bit table

Integer arithmetic on words and double words

Description	Function
+	Addition
-	Subtraction
*	Multiplication
/	Integer division
REM	Remainder of the integer division
SQRT	Integer square root
ABS	Absolute value
INC	Incrementation
DEC	Decrementation

Arithmetic on floating points

Description	Function
+	Addition
-	Subtraction
*	Multiplication
/	Division
SQRT	Square root
ABS	Absolute value

Logic instructions on words and double words

Description	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	Logic rotate to left
ROR	Logic rotate to right

Numerical conversion instructions

Description	Function
BCD_TO_INT	BCD → Binary conversion
INT_TO_BCD	Binary → BCD conversion
GRAY_TO_INT	Gray → Binary conversion
INT_TO_REAL	Integer → Floating point conversion
DINT_TO_REAL	
REAL_TO_INT	Floating point → Integer conversion
REAL_TO_DINT	
DBCD_TO_DINT	Conversion of 32-bit BCD number into 32-bit integer
DINT_TO_DBCD	Conversion of 32-bit integer into 32-bit BCD number
DBCD_TO_INT	Conversion of 32-bit BCD number into 16-bit integer
INT_TO_DBCD	Conversion of 16-bit integer into 32-bit BCD number

Instructions on word and double word tables

Description	Function
Table := Table	Assignment between two tables
Table := Word	Initialize a table
+, -, *, /, REM	Arithmetic operations between tables
+, -, *, /, REM	Arithmetic operations between expressions and tables
SUM	Sum of the elements of a table
EQUAL	Comparison of two tables
NOT	Logic complement of a table
AND, OR, XOR	Logic operations between two tables
AND, OR, XOR	Logic operations between expressions and tables
FIND_EQW, FIND_EQD	Find first element equal to a value
FIND_GTW, FIND_GTD	Find first element greater than a value
FIND_LTW, FIND_LTD	Find first element less than a value
MAX_ARW, MAX_ARD	Find maximum value in a table
MIN_ARW, MIN_ARD	Find minimum value in a table
OCCUR_ARW, OCCUR_ARD	Number of occurrences of a value in a table
SORT_ARW, SORT_ARD	Sort a table in ascending or descending order
ROL_ARW, ROL_ARD	Rotate a table to the left
ROR_ARW, ROR_ARD	Rotate a table to the right

Instructions on floating point tables

Description	Function
Table := Table	Assignment between two tables
Table := Floating point	Initialize a table

Instruction on character string

Description	Function
STRING_TO_INT } STRING_TO_DINT } INT_TO_STRING } DINT_TO_STRING }	ASCII → Binary conversion Binary → ASCII conversion
STRING_TO_REAL	ASCII → Floating point conversion
REAL_TO_STRING	Floating point → ASCII conversion
<, >, <=, >=, =, <>	Alphanumeric comparison
FIND	Position of a substring
EQUAL_STR	Position of first different character
LEN	Length of a character string
MID	Extract a substring
INSERT	Insert a substring
DELETE	Delete a substring
CONCAT	Concatenate two strings
REPLACE	Replace a string
LEFT	Start of string
RIGHT	End of string

Time management instructions

Description	Function
RRTC	Read system date
WRTC	Update system date
PTC	Read date and stop code
ADD_TOD	Add a time period to a time of day
ADD_DT	Add a time period to a date and time
DELTA_TOD	Measure deviation between times of day
DELTA_D	Measure deviation between dates (without time)
DELTA_DT	Measure deviation between dates (with time)
SUB_TOD	Subtract a time period from a time of day
SUB_DT	Subtract a time period from a date and time
DAY_OF_WEEK	Read current day of week
TRANS_TIME	Convert duration to date
DATE_TO_STRING	Convert date to character string
TOD_TO_STRING	Convert time to character string
DT_TO_STRING	Convert complete date to character string
TIME_TO_STRING	Convert duration to character string

"Orphee" instructions

Description	Function
WSHL_RBIT, DSHL_RBIT	Shift word to left, with retrieval of shifted bits
WSHR_RBIT, DSHR_RBIT	Shift word to right with sign extension and retrieval of shifted bits
WSHRZ_C, DSHRZ_C	Shift word to right, replacing with 0, with retrieval of shifted bits
SCOUNT	Up/down counting with indication of under/overflow

Program instructions

Description	Function
HALT	Stop program execution
JUMP	Jump to a label
SRI	Call subroutine
RETURN	Return from subroutine
MASKEVT	Mask events in the PLC
UNMASKEVT	Unmask events in the PLC

All these functions and instructions are described in part B of this reference manual, as well as instructions relating to standard function blocks.

Instructions and functions relating to explicit exchanges and the various applications are described in the "Application-specific functions installation manual".

The control structures are described in section 4.2.-5 of this part.

4.2 Program structure

4.2-1 General

A Structured Text program is organized into statements. Each ST statement consists of the following elements :

- label,
- comments,
- instructions.

Each of these elements is optional, ie. it is possible to have an empty statement, a statement consisting only of comments or consisting only of a label. Each statement begins with an exclamation mark (which is generated automatically).

Example :

```
!      %L2 :   (* Here is a statement with a label, comments *)
          SET %M0; %MW4 := %MW2 + %MW9;
          (* and several instructions *)
          %MF12 := SQRT (%MF14);
```

4.2-2 Comment

A comment is enclosed at either end by the characters (* and *), it can be placed at any point in a statement and there is no restriction on the number of comments per statement. Its role is to facilitate the interpretation of the statement to which it is assigned, but it is not compulsory.

- Any characters can be used in a comment.
- The number of characters is restricted to 256 per comment.
- Nested comments are not permitted.
- A comment may be several lines in length.

Comments are stored in the PLC and can be accessed at any moment by the user. Because of this, **they consume program memory**.

4.2-3 Label

A label is used to reference a statement in a program entity (main program, subroutine, etc) but is not compulsory.

This label has the following syntax : %Li where i is between 0 and 999 and is located at the beginning of the statement. A label reference can only be assigned to a single statement within the same program entity (SR, Main, Program module).

On the other hand, a statement must be referenced in order to allow connection after a program jump.

The label references can be in any order; it is the order in which the statements are entered which is taken into account by the system during the scan.

4.2-4 Instructions

The program is made up of instructions. An ST statement can contain several instructions. Each instruction must end with the character '!':

4.2-5 Control structures

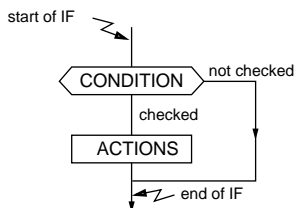
There are four control structures :

- the conditional action IF,
- the conditional iterative actions WHILE and REPEAT,
- the repetitive action FOR.

Each control structure is enclosed between key words, and it begins and ends in the same statement. It is possible to nest control structures one inside the other, regardless of their type. Control structures can be preceded or followed by any other language instruction.

The conditional action IF ... END_IF;

Simple form (the instruction performs an action if a condition is true).

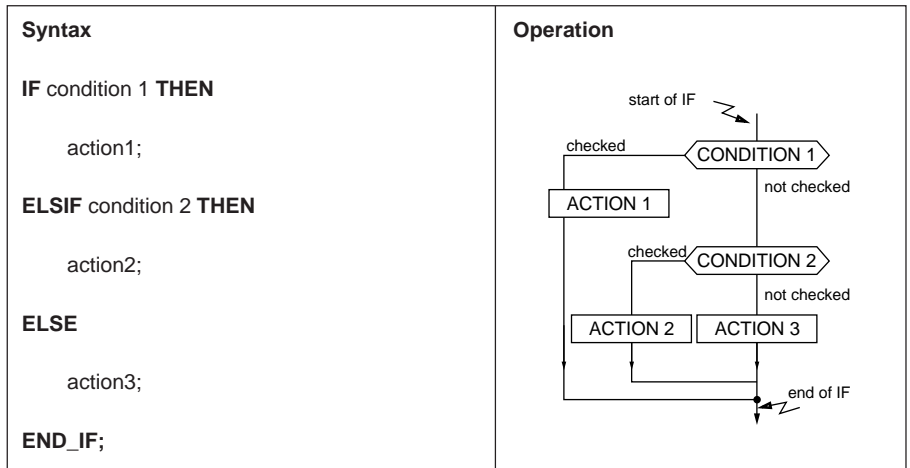
Syntax	Operation
<pre>IF condition THEN actions ; END_IF;</pre>	 <pre> graph TD Start([start of IF]) --> Cond{CONDITION} Cond -- checked --> Act[ACTIONS] Act --> End([end of IF]) Cond -- not checked --> End </pre>

Example :

```

ST : MAST - POST
! IF %M0 AND %M5 THEN
    RESET %M0;
    INC %MW87;
    %MW150 := %MW10+1;
    SET %M23;
END_IF;
```

General form



Example :

```

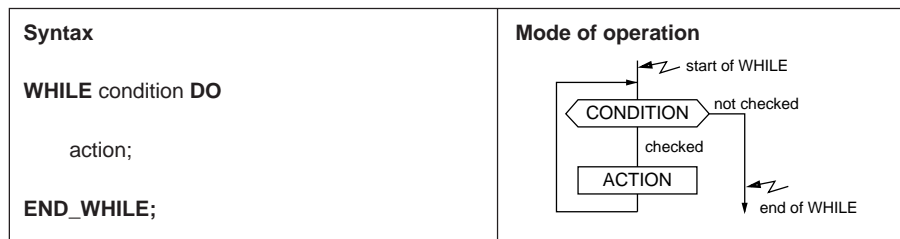
ST : MAST - POST
! IF %M0 AND %M5 THEN
    RESET %M0;
    INC %MW87;
ELSIF %M0 OR %M7 THEN
    %MW12 := %MW120 + 1;
    SET %M9;
ELSE
    %MW12 := 0;
    %MW87 := 0;
    RESET %M9;
END_IF;

```

- Conditions can be multiple.
- Each action represents a list of instructions.
- Several "IF" control structures can be nested.
- There is no restriction on the number of ELSIF instructions.
- There is a maximum of one ELSE part.

The conditional iterative action WHILE ... END_WHILE;

The instruction performs a repetitive action as long as a condition is checked.



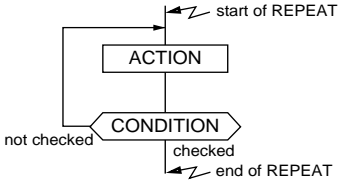
Example :

```
ST : MAST - POST  
| WHILE %MW3<100 DO  
    %MW100:=0;  
    %MW123:=%MW123+1;  
    RESET %M0;  
    SET %M34;  
END_WHILE;
```

- The condition can be multiple.
- The action represents a list of instructions.
- The condition is tested before executing the action. If, when the condition is first evaluated, its value is false, the action is not executed.
- Several WHILE control structures can be nested.

The conditional iterative action REPEAT ... END_REPEAT;

The instruction performs a repetitive action until a condition is checked.

Syntax REPEAT action; UNTIL condition END_REPEAT;	Mode of operation  <p>The flowchart illustrates the execution of a REPEAT loop. It starts at the 'start of REPEAT' point, which leads to a rectangular box labeled 'ACTION'. From the bottom of the 'ACTION' box, the flow goes down to a hexagonal box labeled 'CONDITION'. From the 'CONDITION' box, there are two paths: one labeled 'checked' that loops back to the top of the 'ACTION' box, and another labeled 'not checked' that exits the loop to the left. The loop ends at the 'end of REPEAT' point.</p>
--	--

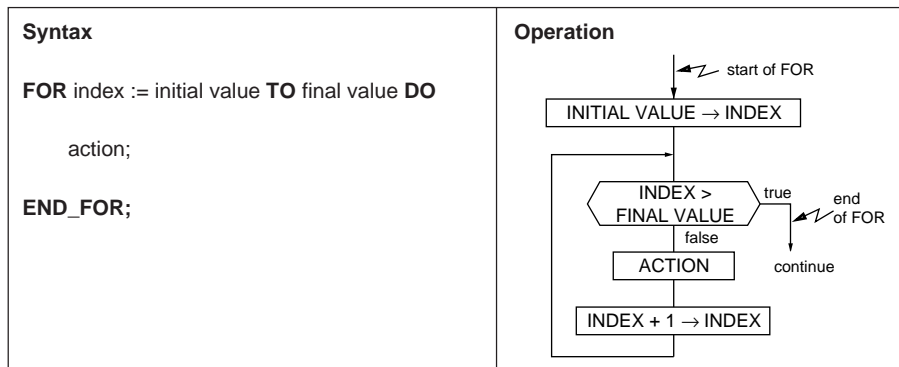
Example :

```
ST : MAST - POST
| REPEAT
|   %M5 :=FALSE;
|   %MW123 :=%MW123-1;
|   %MW100 :=0;
|   SET %M34;
| UNTIL %MW4>12 END_REPEAT;
```

- The condition can be multiple.
- The action represents a list of instructions.
- The condition is tested once the action has been executed. If, when the condition is first evaluated, its value is false, the action is executed once again.
- Several REPEAT control structures can be nested.

The repetitive action FOR ... END_FOR;

The instruction performs a processing operation a certain number of times, incrementing an index by 1 on each loop.



Example :

```

ST : MAST - POST
! FOR %MW99:=0 TO 31 DO
  %MW10:=%MW100[%MW99];
  %MW11:=%MW99;
  %M1:=TRUE;
  RESET %M2;
END_FOR;

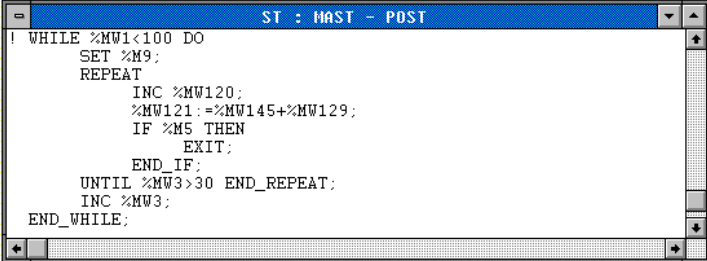
```

- When the index is strictly greater than the final value, execution is continued at the instruction following the END_FOR key word.
- The index is incremented automatically and is therefore not the responsibility of the user.
- The action represents a list of instructions.
- The initial value and the final value must be word-type numerical expressions.
- The index must be a word-type object which is accessible in read mode.
- Several FOR control structures can be nested.

Loop exit instruction EXIT

- The EXIT key word is used to stop execution of the loop and continue at the instruction following the key word at the end of the loop.
- It can be used only in the actions of one of the three WHILE, REPEAT or FOR loops.
- It is assigned to the closest enclosing loop, ie. it does not stop the execution of all the loops which surround it.

Example :



```
ST : MAST - POST
| WHILE %MW1<100 DO
|   SET %M9;
|   REPEAT
|     INC %MW120;
|     %MW121:=%MW145+%MW129;
|     IF %M5 THEN
|       EXIT;
|     END_IF;
|   UNTIL %MW3>30 END_REPEAT;
|   INC %MW3;
| END_WHILE;
```

In this example, the EXIT key word is used to stop the REPEAT loop but not the WHILE loop.

4.3 Rules for executing a Structured Text program

A Structured Text program is executed sequentially, instruction by instruction, while respecting the control structures.

In the case of arithmetic or Boolean expressions consisting of several operators, rules of priority have been defined between the various operators.

Operator priority rules

The table below gives the priority for evaluating a higher or lower priority expression.

Operator	Symbol	Priority
Parentheses	(expression)	Highest
Logic complement Inversion - on an operand + on an operand	NOT NOT - +	
Multiplication Division Modulo	* / REM	
Addition Subtraction	+ -	
Comparisons	<, >, <=, >=	
Comparison of equality Comparison of inequality	= <>	
Logic AND Boolean AND	AND AND	
Logic exclusive OR Boolean exclusive OR	XOR XOR	
Logic OR Boolean OR	OR OR	Lowest

Example :

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

In this example, the NOT is performed on %MW3, then the result is multiplied by 25. The sum of %MW10 and %MW12 is calculated, then a logic AND is performed between the result of the multiplication and the addition.

When there is conflict between two operators of the same priority, the first operator will take precedence (evaluation is performed from left to right).

Example :

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

In this example, %MW34 is first multiplied by 2, then the result is used to perform the modulo.

Use of parentheses

Parentheses are used to modify the order in which operators are evaluated, for example to give an addition higher priority than a multiplication.

Example :

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

In this example, the addition will be performed before the multiplication.

Parentheses can be nested; there is no limit to the levels of nesting.

Parentheses can also be used to avoid incorrect interpretation of the program.

Example :

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

By using operator priority rules, the following interpretation is obtained :

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

The user might well try to perform the following operation :

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

This example shows that parentheses can be used to clarify the program.

Implicit conversions

Implicit conversions relate to words and double words. The operators which are used in arithmetic expressions and comparisons and the assignment operator perform these implicit conversions (which are therefore not the responsibility of the user).

For an instruction of the form : <operand 1> <operator> <operand 2>, the possible conversions are as follows :

Operand 1 of type :	Operand 2 of type :	Conversion Operand 1	Conversion Operand 2	Operation of 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

For an assignment of the form <left operand> := <right operand> , the left operand imposes the type of operand which is expected in order to perform the operation, which means that the right operand must be converted if necessary, according to the table :

Left operand type	Right operand type	Right operand conversion
Word	Word	No
Word	Double word	Word
Double word	Word	Double word
Double word	Double word	No

Note :

Any operation between two immediately adjacent values is performed in double length.

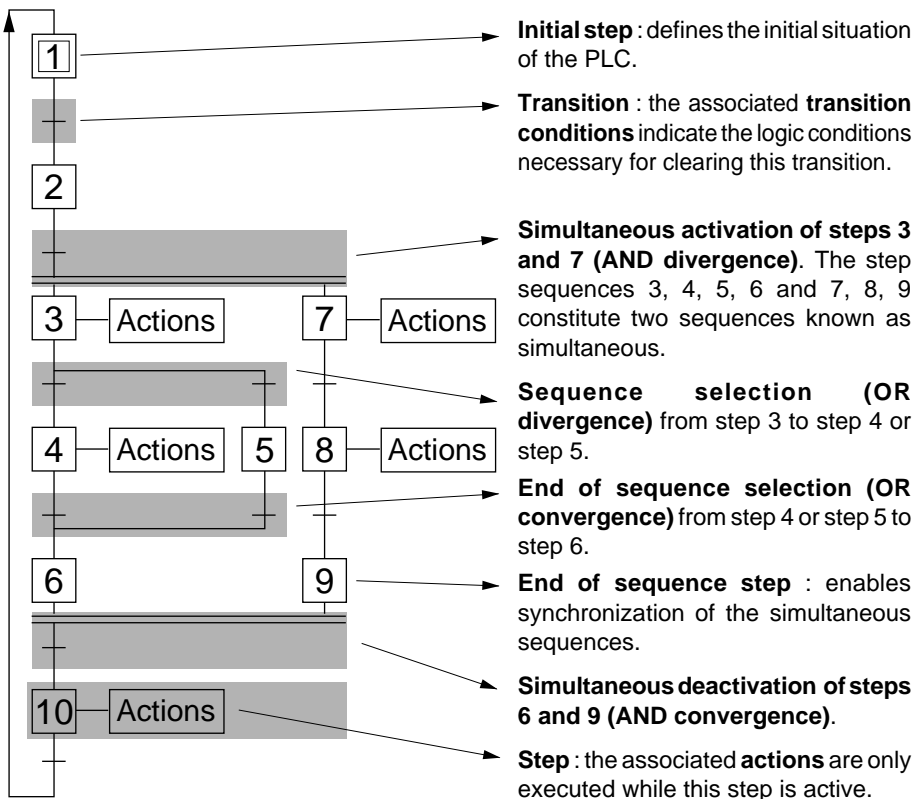
5.1 Presentation of Grafcet language

5.1-1 Reminder of principles of Grafcet

Grafcet language complies with "Sequential Function Chart" (SFC) language found in IEC 1131-3 standard.

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

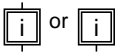




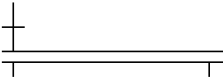

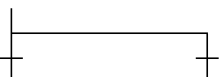

This graphic description of the sequential operation of the control system, and the various situations which occur, is performed using simple graphic symbols :






Transitions and directed links represent in symbolic form the possible progressions of **active steps**.

Actions associated with steps indicate in general terms "what is to be done" when they are active. In particular they describe orders which are to be sent to the operative part (process to be automated) or other automated systems. The set of active steps at any given time defines the situation of the Grafset chart.

5.1-2 Graphic symbols specific to Grafset language

Designation	Symbol	Functions
Initial steps	 or 	Indicate the initial steps active at the start of a cycle after an initialization or a cold restart.
Single steps	 or 	Indicate that the control system is in a stable state. The maximum number of steps can be configured : - from 1 to 96 for a TSX 37-10, - from 1 to 128 for a TSX 37-20 or TSX 57. The maximum number of simultaneously active steps can be configured.
Transitions		Used to change from one step to another. A transition condition associated with this transition is used to define the logic conditions required to clear this transition. The maximum number of transitions can be configured : - from 1 to 96 for a TSX 37-10, - from 1 to 128 for a TSX 37-20 or TSX 57. The maximum number of simultaneously validated transitions can be configured.
AND divergences		Transition from one step to several steps. Used to activate a maximum of 11 steps simultaneously.
AND convergences		Transition from several steps to one step. Used to deactivate a maximum of 11 steps simultaneously.
OR divergences		Transition from one step to several steps. Used to perform a sequence selection to a maximum of 11 steps.
OR convergences		Transition from several steps to one step. Used to end a sequence selection from a maximum of 11 steps.

Designation	Symbol	Functions
Source connector		'n' is the number of the step from which control has come (source step).
Destination connector		'n' is the number of the step to which control is going (destination step).
Directed links : <ul style="list-style-type: none"> • upwards • downwards • to the right or left 		These links are used for sequence selection, to jump over one or more steps, to repeat steps (sequence).

5.1-3 Objects specific to Grafcet

The user has available to him object bits associated with steps, system bits specific to Grafcet language, word objects indicating the activity time of steps and system words specific to Grafcet language.

Designation	Address	Description
Step bits	%Xi	State of step i of the main Grafcet chart (i from 0 to n) (n depends on the processor)
Grafcet system bits (1)	%S21	Initializes the Grafcet chart
	%S22	Resets all Grafcet charts to zero
	%S23	Freezes the Grafcet chart
	%S26	Set to 1 on : - table overflow (steps/transition), - execution of an incorrect chart (destination connector on a step which does not belong to the chart).
Step words	%Xi.T	Active time of step i of main Grafcet chart
Grafcet system words	%SW20	Word indicating, for the current cycle, the number of active steps, to be activated and deactivated.
	%SW21	Word indicating, for the current cycle, the number of validated transitions, to be validated or devalidated.

(1) Details of system bit usage can be found in section 5.2-3.

Step bits %Xi

- These are at 1 when the steps are active.
- These bits can be tested in all tasks, but can only be written in preprocessing of the master task (presetting of charts). These tests and actions are programmed in Ladder language, Instruction list language, or Structured text language.
- These bits cannot be indexed.

Step active time words %Xi.T

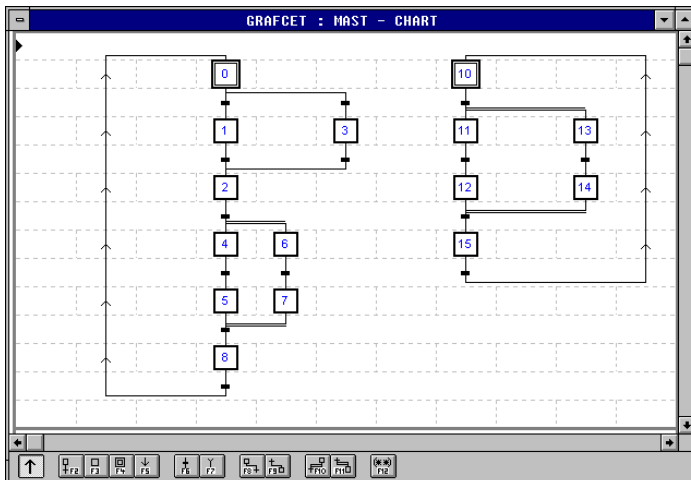
- These are incremented every 100 ms and have a value from 0 to 9999.
- Word incrementation : during the activity of the associated step.
- On deactivation of the step, the contents are frozen.
- On activation of the step, the contents are reset then incremented.
- The number of active time words cannot be configured, one word is reserved for each step.
- These words cannot be indexed.

5.1-4 Grafcet chart representation

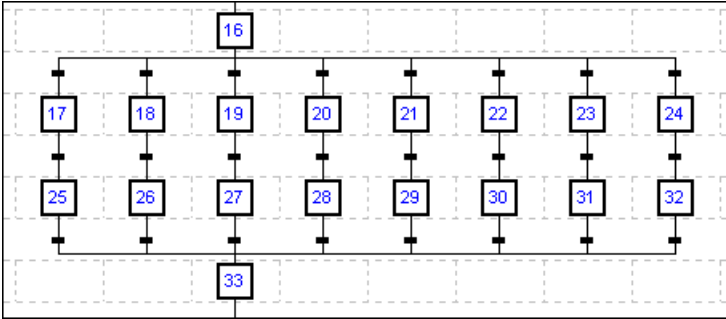
The main chart can be programmed on 8 pages (pages 0 to 7). Each Grafcet page has 14 lines and 11 columns defining 154 cells. One graphic element can be entered in each cell.

Write rules

- The first line is used to enter source connectors.
- The last line is used to enter destination connectors.
- The even lines (from 2 to 12) are step lines (for steps and destination connectors).
- The odd lines (from 3 to 13) are transition lines (for transitions and source connectors).
- Each step is numbered (from 0 to 127) in any order.
- Several charts can be represented on a single page.

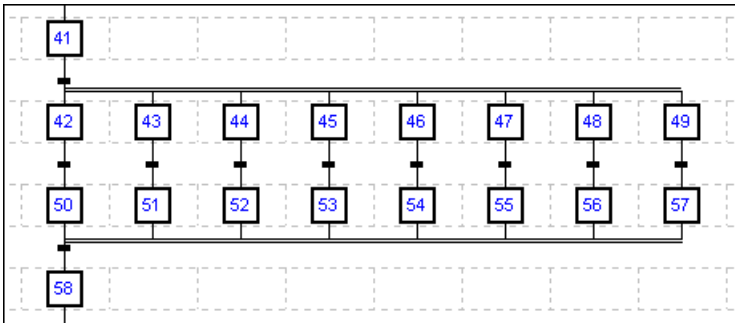


Sequence selection and end of sequence selection



- The number of transitions upstream of an end of sequence selection (OR convergence) or downstream of a sequence selection (OR divergence) must not exceed 11.
- A sequence selection can be directed to the left or right.
- A sequence selection must, in general, conclude with an end of sequence selection.
- To avoid clearing several transitions simultaneously, the associated transition conditions must be exclusive.

Simultaneous step activation and deactivation

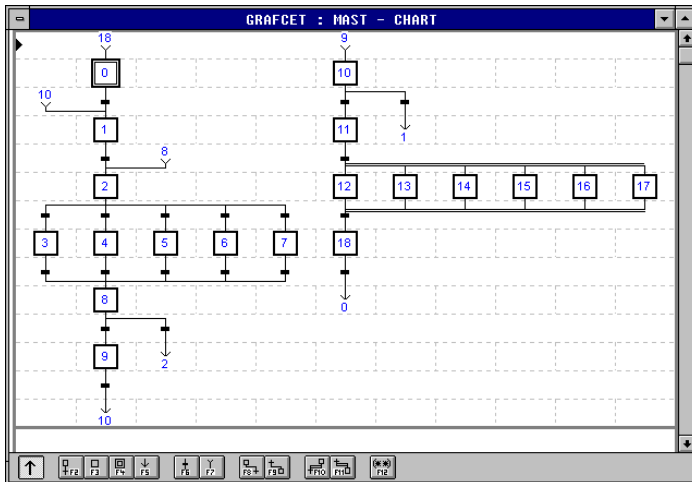


- The number of steps downstream of a simultaneous activation (AND divergence) or upstream of a simultaneous deactivation (AND convergence) must not exceed 11.
- A simultaneous activation of steps must, in general, conclude with a simultaneous deactivation of steps.
- Simultaneous activation is always represented from left to right.
- Simultaneous deactivation is always represented from right to left.

The use of connectors

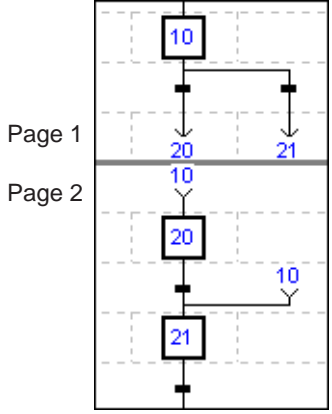
The purpose of connectors is to ensure the continuity of a Grafcet chart when the directed link, either on one page or between two consecutive pages, cannot be drawn. This continuity is provided by a destination connector which always has a corresponding source connector.

- A chart can be looped-back using connectors (for example, looping from step 18 to step 0).
- A sequence can be restarted using connectors (for example, step 10 to step 1 or step 8 to step 2).
- Connectors are used when a chart branch is longer than the page (for example, step 9 to step 10).



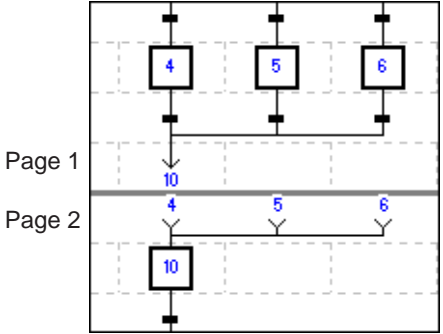
Connectors for sequence selection and end of sequence selection

- For a sequence selection, the transitions and the destination connectors must be entered in the same page.



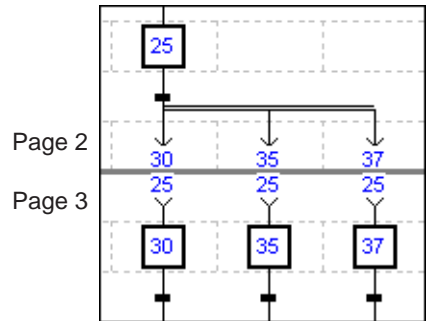
- For an end of sequence selection, the source connectors must be entered in the same page as the destination step.

- For an end of sequence selection followed by a destination connector, there must be the same number of source connectors as steps before the end of sequence selection.



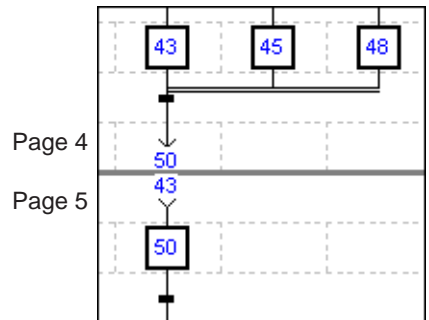
Connectors for simultaneous activation and deactivation of steps

- For simultaneous activation of steps, the destination connectors must be on the same page as the step and the divergence transition.



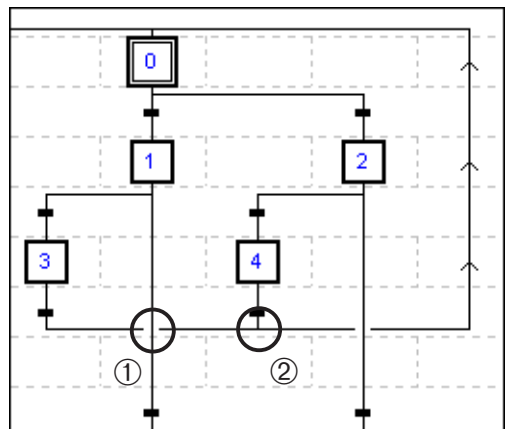
- For simultaneous deactivation, the steps and convergence transition must be on the same page as the destination connector.

When several steps converge on a single transition, the source connector has the number of the upstream step furthest left.



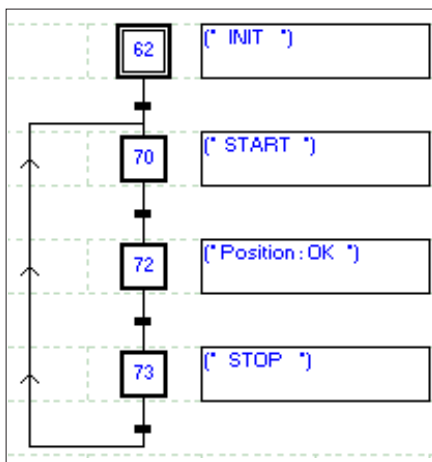
Directed links

- Directed links connect a step to a transition or a transition to a step. They can be vertical or horizontal.
- Directed links can :
 - cross ①, being different kinds,
 - meet ②, being the same kind.
- A link cannot be crossed by a simultaneous step activation or deactivation.



Comments

- In a Grafset page, it is possible to enter a comment in any cell. The text of the comment is enclosed by (* to the left and *) to the right. Its maximum size is 64 characters.
- A comment occupies two adjacent cells on a maximum of two lines. If the display zone is too small, the comment is shortened to fit the display, but when printing the document, the comment is shown in full.
- The comment entered in a Grafset page is stored in the graphic data loaded in the PLC.



5.1-5 Actions associated with steps

Each step has associated actions which can be programmed in Ladder, Instruction list, or Structured text language. These actions are only scanned if the step with which they are associated is active. PL7 software authorizes three types of action :

- **actions on activation** : actions executed once the step with which they are associated is activated.
- **actions on deactivation** : actions executed once the step with which they are associated is deactivated.
- **continuous actions** : actions executed continuously as long as the step with which they are associated is active.

These three types of action can be used for each step.

A single action can contain several programming elements (sequences, statements or rungs).

Referencing actions

These actions are referenced as follows :

MAST - CHART - PAGE n %Xi x

where : x = P1 Activation
 = N1 Continuous
 = P0 Deactivation
 n = Page number
 i = Step number

Example : MAST - CHART - PAGE 0 %X1 P1
 Action on activation of step 1 of page 0

Rules of use

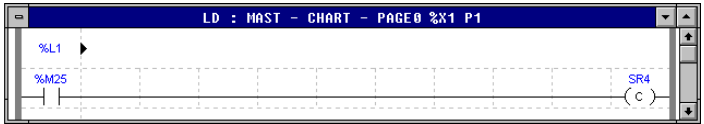
- All the actions are considered as stored actions, consequently :
 - an action which is governed by the duration of a step Xn must be reset on the deactivation of step Xn or the activation of step Xn+1,
 - an action affecting several steps is set to 1 on activation of step Xn and reset on deactivation of step Xn+m.
- All the actions can be controlled by logic conditions, i.e. be conditional.
- The actions which are governed by safety interlocks must be programmed in post-processing (processing performed with each scan, see section 5.2 "Organization of the master task").

Actions on activation or deactivation

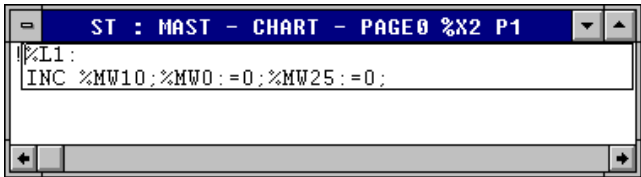
These actions are pulsed and are executed on a single scan. They are used to call a subroutine, increment a counter, etc.

Examples :

- Calling a subroutine :



- Incrementation of word %MW10, and resetting of %MW0 and %MW25 :



Continuous actions

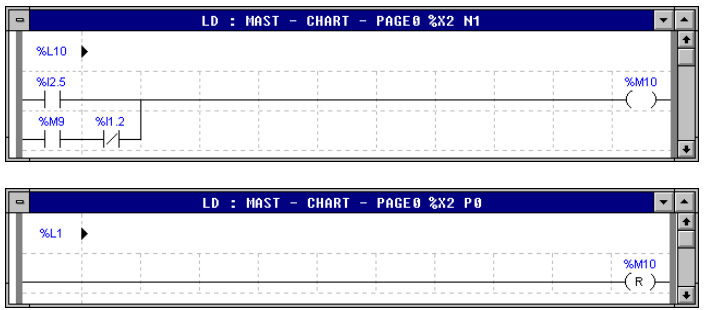
- **Conditional action**

Example :

Bit %M10 is governed by input %I2.5 or internal bit %M9 and to input %I1.2.

As long as step 2 is active and these conditions are present, %M10 is set to 1. The last state read on deactivation is stored in the memory because the associated actions are no longer scanned.

It is therefore necessary to reset bit %M10 to 0, in the action on deactivation of the step for example.

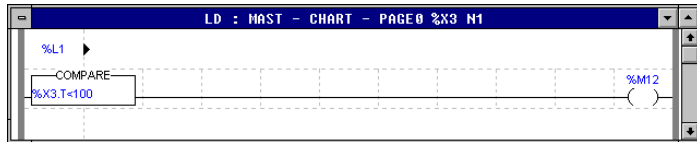


- **Timed conditional action**

This is a special case, in which the time is a logic condition. This link can be performed simply by testing the active time associated with the step.

Example :

Bit %M12 is controlled as long as the active time of step 3 is less than 10 seconds (time base : 100 ms).



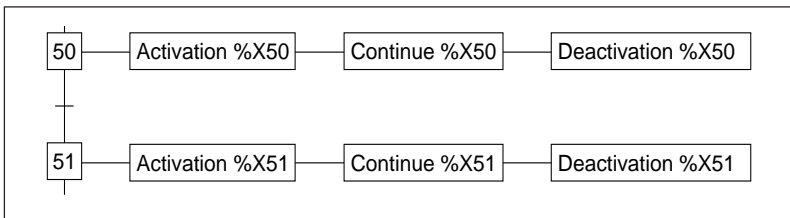
- These actions can also be unconditional.

Order of execution of the actions

In the example below, on one scan, the order of execution of the actions is as follows :

When step 51 is activated, the actions are executed in the following order :

1. actions on deactivation of step 50,
2. actions on activation of step 51,
3. continuous actions of step 51.



When step 51 is deactivated, the associated continuous actions are no longer scanned.

5.1-6 Conditions associated with transitions

- Each transition has an associated condition which can be programmed in Ladder, Instruction list or Structured text language.
- A transition condition is only scanned when the transition with which it is associated is validated.
- A transition condition corresponds to a rung, a list of instructions or a Structured text statement, comprising a series of tests on bits and/or words.
- **A transition condition which is not programmed is always a false transition condition.**

Referencing the transition conditions

The transition conditions are referenced as follows :

MAST - CHART - PAGE n %X(i) → %X(j)

where : n = Page number

i = Upstream step number

j = Downstream step number

Example : MAST - CHART - PAGE 0 %X(0) → %X(1)

Transition condition associated with step 0 and step 1 of page 0 of the chart.

During simultaneous step activation or deactivation, the address indicated is that in the column furthest to the left.

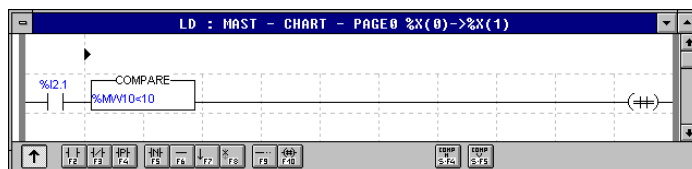
Rules for programming in Ladder language

The condition associated with the transition is programmed in the form of a rung comprising a test zone and an action zone.

The structure of the rung is the same as that of a rung programmed in a program module.

Only the following elements can be used :

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



Rules for programming in Instruction list language

The transition condition is programmed in the form of a list of instructions containing only test instructions.

The list of instructions for writing a transition condition differs from a standard list of instructions as follows :

- general structure :
 - no label (%L).
- list of instructions :
 - no action instructions (bit objects, words or function blocks),
 - no jumping, calling subroutines.

```
IL : MAST - CHART - PAGE0 %X(1)->%X(2)
|
|   LD   %I2.1
|   AND  [%MW10<10]
|
|
```

Rules for programming in Structured text language

The transition condition is programmed in the form of a Boolean expression or an arithmetic expression or a combination of the two.

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

- general structure :
 - no label (%L),
 - no action statement, conditional statement or iterative statement.
- list of instructions :
 - no action on bit object,
 - no jumping, calling subroutines,
 - no transfer, no action instruction on blocks.

```
ST : MAST - CHART - PAGE0 %X(2)->%X(3)
!%I2.1 AND %MW10<10
```

Transition condition using the active time of a step

In certain applications, actions are controlled with no monitoring of feedback data (end of travel, detector, etc). The duration of the step is conditioned by a time : PL7 language enables the active time associated with each step to be used.

Example :

If the user wishes to remain in step 3 for 15 seconds, the condition for transition between step 3 and step 4 will be (for example in structured text language) :



5.2 Organization of the master task

5.2-1 Description of the master task

A program written in Grafcet language has three consecutive processing sections : preprocessing, sequential processing and post-processing.

They are scanned in accordance with the basic scan cycle below :

Period (in periodic scanning) :

Time between two tasks scans, defined by configuration.

In cyclical scanning, inputs are read after updating outputs.

Reading inputs :

Reading the physical states of the PLC input modules (values frozen during processing).

Preprocessing :

Used to process :

- initializations on power failure or return,
- the presetting of the Grafcet chart,
- the input logic.

Sequential processing :

Used to process the sequential structure of the application and provides access to the processing of transition conditions and actions directly associated with steps.

Post-processing :

Used to process :

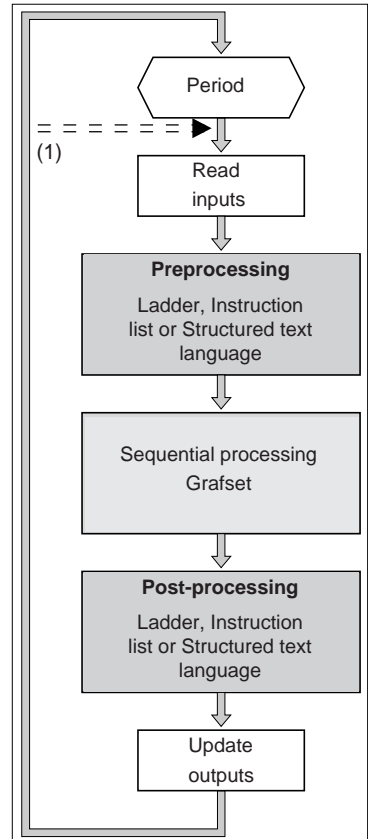
- the output logic,
- the monitoring and safety interlocks specific to outputs.

Updating of outputs :

Updating the physical state of the PLC output modules (values frozen during processing).

Effect of multi-task processing :

This structure remains the same, whether processing is multi-task or single task.



(1) in cyclical scanning

5.2-2 Preprocessing

Entered in Ladder language, Instruction list language or Structured text language, preprocessing is scanned in its entirety from top to bottom.

Executed before the sequential and post-processing sections, it is used to process all events which influence these :

- management of power returns and reinitializations,
- resetting or presetting of Grafcet charts.

It is, therefore, only in preprocessing that the bits associated with the steps will be used (setting to 0 or 1 of step bits %Xi by Set and Reset instructions).

Presetting the Grafcet chart

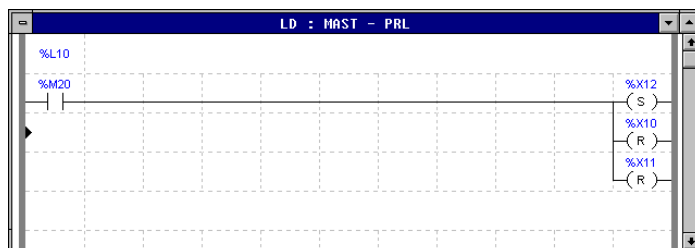
It may be necessary to preset a Grafcet chart when changing from normal operation to a specific mode of operation or on the occurrence of an incident (example : fault causing degraded operation).

This operation affects the normal operation of the application scan, and should therefore be used with caution. Presetting can be applied to all or part of sequential processing :

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

Note :

When resetting a step to zero, actions on deactivation of this step are not executed.



5.2-3 The use of system bits in preprocessing

As the system bits associated with the Grafcet chart are numbered in order of priority (%S21 to %S23), when several of them are simultaneously set to 1 in preprocessing, they are processed one by one in ascending order (only one is effective per scan cycle). These bits are effective at the start of sequential processing.

Initializing the Grafcet chart : %S21

Normally at 0, setting %S21 to 1 causes :

- the deactivation of the active steps,
- the activation of the initial steps.

Set to 1	Reset to 0
<ul style="list-style-type: none"> • By setting %S0 to 1 • By the user program • By the terminal (1) 	<ul style="list-style-type: none"> • By the system at the start of sequential processing • By the user program • By the terminal

• Use

When managed by the user program, %S21 must be set to 0 or 1 **in preprocessing**.

Resetting the Grafcet chart to zero : %S22

Normally at 0, setting %S22 to 1 causes the deactivation of the active steps of all sequential processing.

Set to 1	Reset to 0
<ul style="list-style-type: none"> • By the user program • By the terminal (1) 	<ul style="list-style-type: none"> • By the system at the end of post-processing

• Use

- this bit must be set to 1 **in preprocessing**,
- resetting %S22 to 0 is managed by the system; it **need not, therefore, be reset to 0** by the program or the terminal.

To restart sequential processing in a given situation, the application must contain an initialization or Grafcet chart preset procedure.

(1) In the CPU debug screen (Grafcet part) or in the animation table

Freezing the Grafcet chart : %S23

Normally at 0, setting %S23 to 1 maintains the state of the Grafcet charts. Irrespective of the value of the transition conditions downstream of the active steps, the Grafcet charts do not change. This frozen state is maintained as long as bit %S23 is at 1.

Set to 1	Set to 0
<ul style="list-style-type: none"> • By the user program • By the terminal (1) 	<ul style="list-style-type: none"> • By the user program • By the terminal (1)

(1) In the CPU debug screen (Grafcet part) or in the animation table.

- **Use**

- managed by the user program, this bit is set to 1 or 0 **in preprocessing**,
- bit %S23 associated with bits %S21 and %S22 is used to freeze sequential processing at initial state or state 0. Similarly, the Grafcet chart can be preset then frozen by %S23.

On starting a new application or on losing the system context, the system performs a cold start. Bit %S21 is set to 1 by the system before preprocessing is called and the Grafcet chart set on the initial steps. If the user wants the application to be processed in a particular way in the event of cold start, he can test %S0 which remains at 1 during the first scan of the master task (MAST).

After a power outage without changing application, the system performs a warm restart, restarting in the state preceding the power outage. If the user wants the application to be processed in a particular way in the event of a warm restart, he can test %S1 in preprocessing, and call the corresponding program.

5.2-4 Sequential processing

This processing section is used to program the sequential structure of the application. Sequential processing consists of :

- the main chart organized into 8 pages.

In the main chart, several unconnected Grafcet charts can be programmed and run simultaneously.

Principle of evolution

The evolution of the Grafcet chart is managed as follows :

Phase 1 :

1. Evaluation of the condition of validated transitions.
2. Request to deactivate associated upstream steps.
3. Request to activate relevant downstream steps.

Phase 2 :

Evolution of the state of the Grafcet chart as a function of the cleared transitions :

1. Deactivation of the steps upstream of the cleared transitions.
2. Activation of the steps downstream of the cleared transitions.
3. Devalidation of the cleared transitions.
4. Validation of the transitions downstream of the new activated steps.

The system updates two tables dedicated respectively to step activity and transition validity :

- **the step activity table** stores, for the current scan, the active steps, the steps to be activated and the steps to be deactivated,
- **the transition validity table** stores, for the current scan, the transitions located downstream of the steps concerned with the preceding table.

Phase 3 :

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

1. Actions on deactivation of the steps to be deactivated.
2. Actions on activation of the steps to be activated.
3. Continuous actions of the active steps.

Exceeding the possibilities

The number of elements in the step activity table and the transition validity table can be configured. Exceeding the capacity of either table causes :

- the PLC to stop (execution of the application stops),
- system bit %S26 to change to 1 (capacity of one of the two tables exceeded),
- the ERR indicator lamp on the PLC to flash.

The system provides the user with two system words :

- **%SW20** : word indicating, for the current scan, the number of steps active, to be activated or deactivated.
- **%SW21** : word indicating, for the current scan, the number of transitions validated, to be validated or devalidated.

In the event of a PLC blocking fault, system words **%SW125 to %SW127** are used to determine the nature of the fault.

- %SW125 = ERR7 (hex) Table overflow (steps/transitions).
- %SW125 = ERRE (hex) Execution of the incorrect Grafcet chart.
(problem of transition with unresolved destination connector).

%SW125	%SW126	%SW127	
ERR7	≠ 0	= 0	Step table overflow
ERR7	= 0	≠ 0	Transition table overflow
ERRE	Step n°	64	Incorrect execution of Grafcet

5.2-5 Post-processing

Entered in Ladder language, Instruction list language or Structured text language, post-processing is scanned from top to bottom. This processing is the last executed before activation of the outputs and is used to program the output logic.

Actions associated with the Grafcet chart

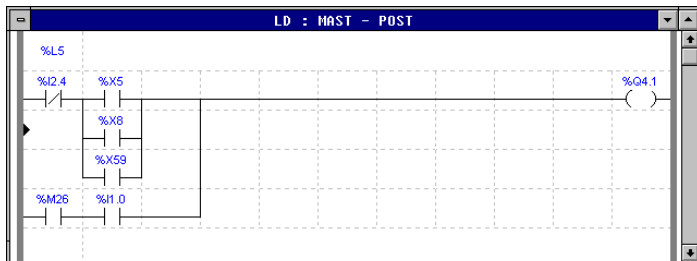
Post-processing is used to perform the actions generated in sequential processing by integrating the operating and stop modes as well the safety interlocks specific to the action into the equation of an output. It is also used to process an output activated several times in sequential processing.

As a rule, it is advisable to program actions which directly affect the process in post-processing.

Example :

- **%I2.4** : safety interlock for controlling output %Q4.1.
- **%M26** : internal bit resulting from the input logic controlling the operating and stop modes.
- **%I1.0** : push-button.

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



Actions independent of the Grafcet chart

Post-processing is also used to program the outputs which are independent of sequential processing.

Monitoring the execution of the Grafcet chart

In certain circumstances, it may be necessary to monitor the operation of the Grafcet chart by testing the active time of certain steps.

This time is tested by comparing with either a minimum value or a maximum value defined by the user. The use made of the fault indication is at the user's discretion (indication, special operating procedure, transmission of a message).

Example :

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

Section	Page
1 Description of basic instructions	1/1
1.1 Presentation of basic instructions	1/1
1.1-1 General	1/1
1.2 Boolean instructions	1/2
1.2-1 Presentation of Boolean instructions	1/2
1.2-2 Instruction format	1/3
1.2-3 Load instructions	1/4
1.2-4 Assignment instructions	1/5
1.2-5 Logic AND instructions	1/6
1.2-6 Logic OR instructions	1/7
1.2-7 Exclusive OR instructions	1/8
1.3 Predefined function blocks	1/9
1.3-1 Programming principles for predefined function blocks	1/9
1.3-2 Timer function block %Tmi	1/10
1.3-3 Up/down counter function block %Ci	1/14
1.4 Numerical processing on integers	1/17
1.4-1 General	1/17
1.4-2 Comparison instructions	1/19
1.4-3 Assignment instructions	1/20
1.4-4 Arithmetic instructions on integers	1/23
1.4-5 Logic instructions	1/25
1.4-6 Numerical expressions	1/27
1.5 Program instructions	1/28
1.5-1 Subroutine call	1/28
1.5-2 Subroutine return	1/29
1.5-3 Program jumps	1/30
1.5-4 Program end instructions	1/32
1.5-5 Stop program	1/33
1.5-6 Event masking/unmasking instructions	1/34
1.5-7 NOP Instruction	1/34

Section	Page
2 Description of advanced instructions	2/1
2.1 Presentation of advanced instructions	2/1
2.1-1 General	2/1
2.2 Advanced predefined function blocks	2/2
2.2-1 Monostable function block %MNi	2/2
2.2-2 Register function block %Ri	2/5
2.2-3 Drum controller function block %DRi	2/9
2.2-4 Timer function block %Ti (Series 7)	2/13
2.3 Vertical comparison blocks	2/17
2.4 Shift instructions	2/19
2.5 Floating point instructions	2/20
2.5-1 General	2/20
2.5-2 Floating point comparison instructions	2/21
2.5-3 Floating point assignment instructions	2/22
2.5-4 Floating point arithmetic instructions	2/22
2.6 Numeric conversion instructions	2/24
2.6-1 BCD <--> Binary conversion instructions	2/24
2.6-2 Integer <--> Floating point conversion instructions	2/26
2.6-3 Gray --> Integer conversion instructions	2/28
2.7 Word table instructions	2/29
2.7-1 General	2/29
2.7-2 Word table assignment	2/30
2.7-3 Arithmetic instructions on tables	2/32
2.7-4 Logic instructions on tables	2/33
2.7-5 Summing function on tables	2/34
2.7-6 Table comparison function	2/35
2.7-7 Find functions on tables	2/36
2.7-8 Find maximum and minimum values function on tables	2/38
2.7-9 Number of occurrences of a value in a table	2/39
2.7-10 Rotate shift function on tables	2/40
2.7-11 Sort function on tables	2/41

Section	Page	
2.8	Character string instructions	2/42
2.8-1	Format of a string or table of characters	2/42
2.8-2	Character string assignment	2/43
2.8-3	Alphanumeric comparisons	2/44
2.8-4	Numeric <---> ASCII conversion functions	2/45
2.8-5	Binary --->ASCII conversion	2/45
2.8-6	ASCII ---> Binary conversion	2/47
2.8-7	Floating point ---> ASCII conversion	2/48
2.8-8	ASCII --> Floating point conversion	2/49
2.8-9	Concatenation of two strings	2/50
2.8-10	Deletion of a character substring	2/51
2.8-11	Insertion of a character substring	2/52
2.8-12	Replacement of a character substring	2/54
2.8-13	Extraction of a character substring	2/56
2.8-14	Extraction of characters	2/58
2.8-15	Comparison of two character strings	2/60
2.8-16	Search for a character substring	2/61
2.8-17	Length of a character string	2/62
2.9	Time management instructions : Date, Time of day, Duration	2/63
2.9-1	Parameter format	2/63
2.9-2	Use of system bits and words - General	2/65
2.9-3	Read system date	2/66
2.9-4	Update system date	2/66
2.9-5	Read date and stop code	2/67
2.9-6	Read day of the week	2/68
2.9-7	Add / Remove a duration at a date	2/69
2.9-8	Add / Remove a duration at a time of day	2/70
2.9-9	Difference between two dates (no time)	2/72
2.9-10	Difference between two dates (with time)	2/73
2.9-11	Difference between two times	2/74
2.9-12	Convert a Date to a character string	2/75
2.9-13	Convert a complete Date to a character string	2/76
2.9-14	Convert a Duration to a character string	2/77
2.9-15	Convert a Time of day to a character string	2/78
2.9-16	Convert a Duration to HHHH:MM:SS	2/80

Section	Page
2.10 Bit table instructions	2/81
2.10-1 Copy one bit table to another bit table	2/81
2.10-2 Bit table logic instructions	2/82
2.10-3 Copy from a bit table to a word table	2/83
2.10-4 Copy from a word table to a bit table	2/85
2.11 "Orphee" functions : shift, counter	2/87
2.11-1 Shifts on words with retrieval of shifted bits	2/87
2.11-2 Up/down counting with indication of over/underflow	2/90
3 System bits and words	3/1
3.1 System bits	3/1
3.1-1 List of system bits	3/1
3.1-2 Detailed description of system bits	3/3
3.2 System words	3/8
3.2-1 List of system words	3/8
3.2-2 Detailed description of system words	3/9
4 Differences between PL7-2/3 and PL7-Micro/Junior	4/1
4.1 Differences between PL7-2/3 and PL7-Micro/Junior	4/1
5 List of reserved words	5/1
5.1 Reserved words	5/1

Section	Page
6 Conformity to the IEC standard 1131-3	6/1
6.1 Conformity to the IEC 1131-3 standard	6/1
6.1-1 Conformity tables	6/1
7 Quick reference guide	7/1
7.1 Quick reference guide	7/1
8 Performance	8/1
8.1 General	8/1
8.2 TSX 37 performance	8/3
8.2-1 Boolean instructions	8/3
8.2-2 Function blocks	8/4
8.2-3 Integer and floating point arithmetic	8/6
8.2-4 Program instructions	8/8
8.2-5 Command structure	8/8
8.2-6 Numeric conversions	8/9
8.2-7 Bit string	8/9
8.2-8 Word, double word and floating point tables	8/11
8.2-9 Time management	8/14
8.2-10 Character strings	8/15
8.2-11 Application-specific functions and Orphee function	8/16
8.2-12 Explicit I/O	8/17

Section	Page
<hr/> 8.3	<hr/> TSX 57 performance 8/18
8.3-1	Boolean instructions 8/18
8.3-2	Function blocks 8/19
8.3-3	Integer and floating point arithmetic 8/21
8.3-4	Program instructions 8/23
8.3-5	Command structure 8/23
8.3-6	Numeric conversions 8/24
8.3-7	Bit string 8/24
8.3-8	Word, double word and floating piont tables 8/26
8.3-9	Time management 8/29
8.3-10	Character strings 8/30
8.3-11	Application-specific functions and Orphee function 8/31
8.3-12	Explicit I/O 8/32
<hr/> 8.4	<hr/> Size of the application 8/34
8.4-1	Description of the memory zones 8/34
8.4-2	Memory size of PL7 objects 8/35
8.4-3	Module memory size 8/35
8.4-4	Memory size of advanced functions 8/39
<hr/> 8.5	<hr/> Appendix : method of calculating the number of instructions 8/44
<hr/> 9	<hr/> Index 9/1

1.1 Presentation of basic instructions

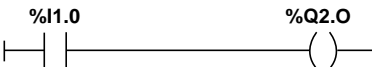
1.1-1 General

The instructions described in this section comply with the main basic instructions defined in IEC standard 1131.3.

These instructions always produce the same effect, irrespective of the language used. Only their presentation in the program changes.

Example of a Boolean equation :

In Instruction list language : LD %I1.0
ST %Q2.0

In Ladder language : 

In Structured text language : %Q2.0 := %I1.0 ;

These three Boolean equations are equivalent. Bit object %Q2.0 takes the value (assignment instruction) of bit object %I1.0 (load instruction).

Basic instructions include :

- Boolean instructions (processing on bits),
- Predefined control system timer and counter function blocks,
- Numerical instructions on integers (processing on words and double words),
- Program instructions.

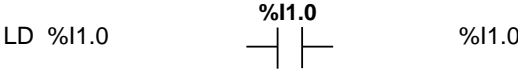
The other instructions are described in section 2, "Description of advanced instructions".

1.2 Boolean instructions

1.2-1 Presentation of Boolean instructions

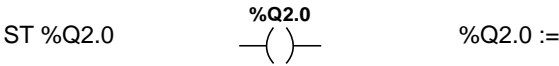
Boolean instructions act on all bit type data (I/O bits, internal bits etc).

- **Test elements**, example : N/O contact. Contact closed when the bit object which controls it is at state 1.



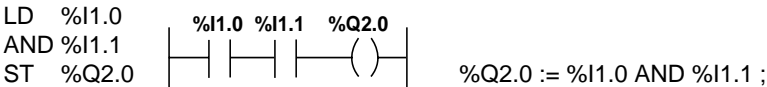
- **Action elements**, example : direct coil.

The associated bit object takes the logic value of the logic result of the test element.



- **Boolean equation :**

The Boolean result of the test elements is applied to the action element.

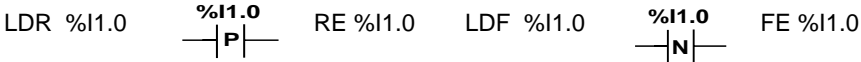


Rising and falling edges

Test instructions can be used to detect rising or falling edges on PLC I/O bits or internal bits.

Rising edge sensing contact :

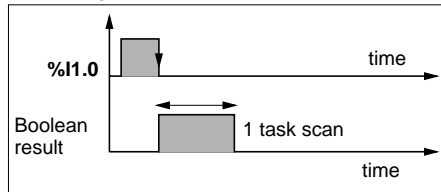
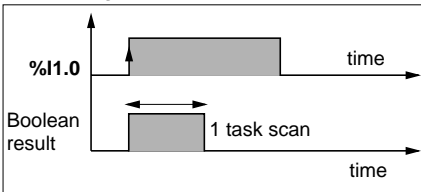
Falling edge sensing contact :



- **For all inputs (discrete, counter, etc) :** an edge is detected when the state of the bit has changed between scan n-1 and the current scan n. It remains detected during the current scan (see part A, section 1.3-2).

Rising edge : detects a change of the controlling input from 0 to 1.

Falling edge : detects a change of the controlling input from 1 to 0.



- **For outputs or internal bits :** detection of an edge is independent of the task scan. A rising or falling edge on internal bit %Mi is detected when its state has changed between two read operations. This rising or falling edge remains detected as long as the internal bit is not scanned in the action zone.
- Do not perform a SET or RESET on an object whose rising or falling edge is being tested (in Ladder language and Instruction list language).

1.2-2 Instruction format

Boolean instructions are described in the following way :

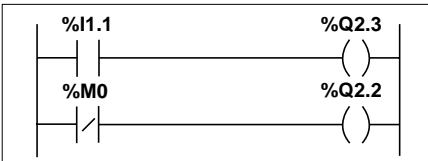
The instruction described appears in bold type. Each equation is illustrated using the different languages.

Load instructions

These instructions correspond to :

- N/O contacts : contact closed when the bit object which controls it is at state 1.
- ...

Ladder language



Instruction list language

```
LD    %I1.1
ST    %Q2.3
LDN   %M0
ST    %Q2.2
```

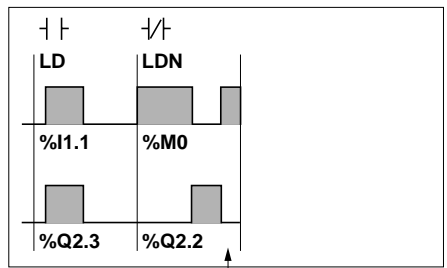
Structured text language

```
%Q2.3 := %I1.1 ;
%Q2.2 := NOT %M0 ;
```

Authorized operands

Code	Operand
┆┆ LD	%I,%Q,%M,%S,%BLK,%*:Xk, %Xi
┆/┆ LDN	%I,%Q,%M,%S,%BLK,%*:Xk, %Xi

Timing diagram

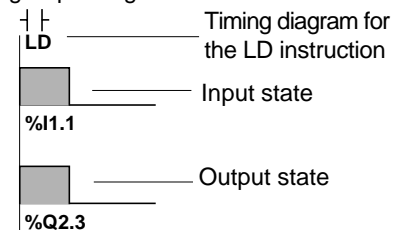


List of operands

- 0/1 immediate value 0 (false) or 1 (true)
- %I PLC input %Ix.i
- %Q PLC output %Qx.i
- %M internal bit %Mi
- %S system bit %Si
- %BLK function block bit, eg : %Tmi.Q
- %*:Xk word extract bit, eg : %MWi:Xk
- %Xi step bit

Timing diagram

The 4 timing diagrams have been grouped together.

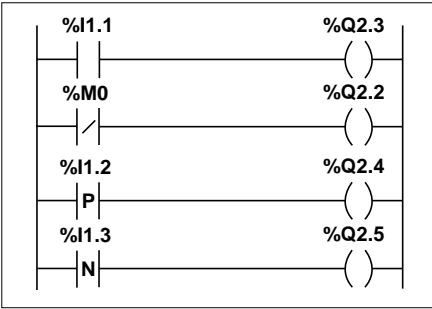


1.2-3 Load instructions

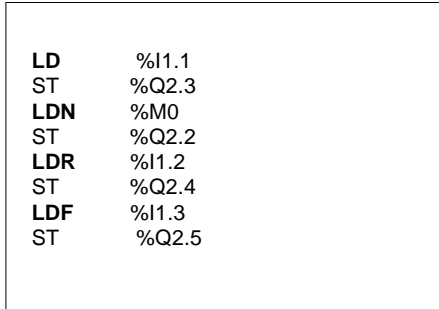
These instructions correspond to :

- N/O contacts : contact closed when the bit object which controls it is at 1.
- N/C contacts : contact closed when the bit object which controls it is at 0.
- Rising edge contacts : detects a change of the controlling bit from 0 to 1.
- Falling edge contacts : detects a change of the controlling bit from 1 to 0.

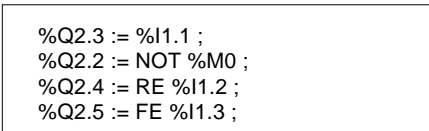
Ladder language



Instruction list language



Structured text language

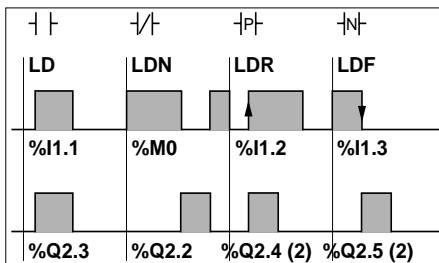


Authorized operands

Code	Operand
┆┆ LD	%I,%Q,%M,%S,%BLK,%•:Xk, %Xi (1)
┆/┆ LDN	%I,%Q,%M,%S,%BLK,%•:Xk, %Xi (1)
┆P┆ LDR	%I,%Q,%M
┆N┆ LDF	%I,%Q,%M

(1) True (1)/False (0) in Instruction list or Structured text language

Timing diagram



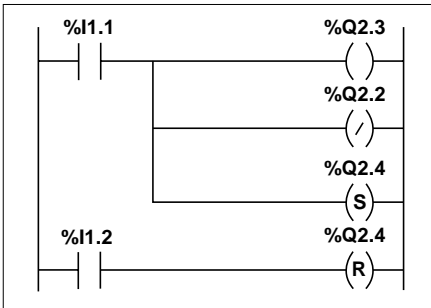
(2) Set to 1 during 1 cycle

1.2-4 Assignment instructions

These instructions correspond to :

- Direct coils : the associated bit object takes the value of the result of the equation.
- Negated coils : the associated bit object takes the inverse value of the result of the equation.
- Set (latch) coils : the associated bit object is set to 1 when the result of the equation is at 1.
- Reset (unlatch) coils : the associated bit object is set to 0 when the result of the equation is at 1.

Ladder language



Instruction list language

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

STN   %Q2.2

S     %Q2.4

LD    %I1.2
R     %Q2.4
```

Equivalent in Structured text language

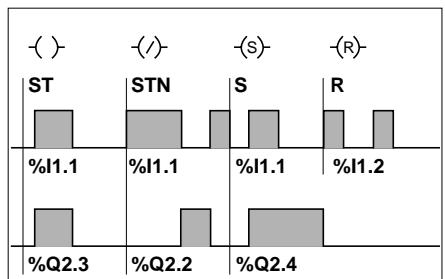
```
%Q2.3 := %I1.1 ;
%Q2.2 := NOT %I1.1 ;
IF %I1.1 THEN
    SET %Q2.4 ;
END_IF ;
IF %I1.2 THEN
    RESET %Q2.4 ;
END_IF ;
```

Authorized operands

Code	Operand
() ST	%I,%Q,%M,%S,%*Xk
(/) STN	%I,%Q,%M,%S,%*Xk
(S) S	%I,%Q,%M,%S,%*Xk, %Xi (1)
(R) R	%I,%Q,%M,%S,%*Xk, %Xi (1)

(1) Only in preprocessing.

Timing diagram

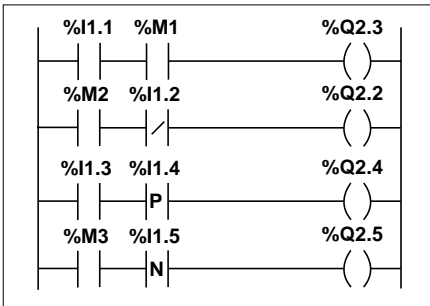


1.2-5 Logic AND instructions

These instructions perform :

- A logic AND between the operand and the Boolean result of the preceding instruction.
- A logic AND between the inverse of the operand and the Boolean result of the preceding instruction.
- A logic AND between the rising edge of the operand and the Boolean result of the preceding instruction.
- A logic AND between the falling edge of the operand and the Boolean result of the preceding instruction.

Ladder language



Instruction list language

```
LD    %I1.1
AND   %M1
ST    %Q2.3
LD    %M2
ANDN  %I1.2
ST    %Q2.2
LD    %I1.3
ANDR  %I1.4
ST    %Q2.4
LD    %M3
ANDF  %I1.5
ST    %Q2.5
```

Structured text language

```
%Q2.3 := %I1.1 AND %M1 ;
%Q2.2 := %M2 AND (NOT %I1.2) ;
%Q2.4 := %I1.3 AND (RE %I1.4) ;
%Q2.5 := %M3 AND (FE %I1.5) ;
```

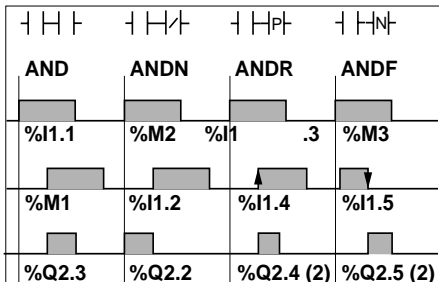
Note : The parentheses are optional but make the program easier to read.

Authorized operands

Code	Operand
AND	%I,%Q,%M,%S,%BLK,%*Xk, %Xi (1)
ANDN	%I,%Q,%M,%S,%BLK,%*Xk, %Xi (1)
ANDR	%I,%Q,%M
ANDF	%I,%Q,%M

(1) True (1)/False (0) in Instruction list or Structured text language

Timing diagram



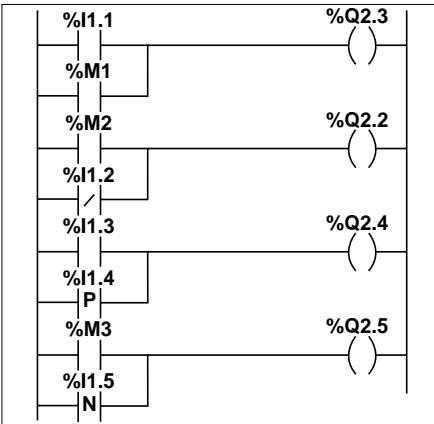
(2) Set to 1 during 1 cycle

1.2-6 Logic OR instructions

These instructions perform :

- A logic OR between the operand and the Boolean result of the preceding instruction.
- A logic OR between the inverse of the operand and the Boolean result of the preceding instruction.
- A logic OR between the rising edge of the operand and the Boolean result of the preceding instruction.
- A logic OR between the falling edge of the operand and the Boolean result of the preceding instruction.

Ladder language



Instruction list language

```

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

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

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

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

Structured text language

```

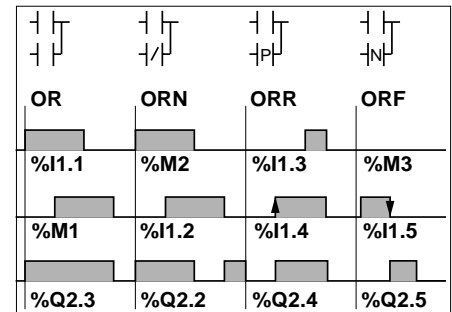
%Q2.3 := %I1.1 OR %M1 ;
%Q2.2 := %M2 OR (NOT %I1.2) ;
%Q2.4 := %I1.3 OR (RE %I1.4) ;
%Q2.5 := %M3 OR (FE %I1.5) ;
    
```

Note : The parentheses are optional but make the program easier to read.

Authorized operands

Code	Operand
OR	%I,%Q,%M,%S,%BLK,%*Xk, %Xi (1)
ORN	%I,%Q,%M,%S,%BLK,%*Xk, %Xi (1)
ORR	%I,%Q,%M
ORF	%I,%Q,%M

Timing diagram



(1) True (1)/False (0) in Instruction list or Structured text language

1.2-7 Exclusive OR instructions

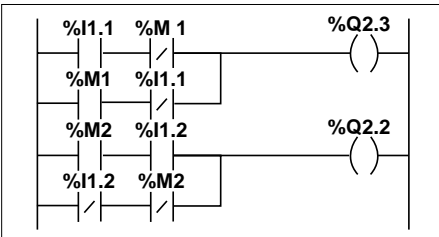
These instructions perform :

- An exclusive OR between the operand and the Boolean result of the preceding instruction.
- An exclusive OR between the inverse of the operand and the Boolean result of the preceding instruction.
- An exclusive OR between the rising edge of the operand and the Boolean result of the preceding instruction.
- An exclusive OR between the falling edge of the operand and the Boolean result of the preceding instruction.

Note :

There are no specific graphic elements for the exclusive OR in Ladder language. However, the exclusive OR can be programmed by using a combination of N/O and N/C contacts (see example below).

Ladder language equivalent



Structured text language

```

%Q2.3 := %I1.1 XOR %M1 ;
%Q2.2 := %M2 XOR (NOT %I1.2) ;
%Q2.4 := %I1.3 XOR (RE %I1.4) ;
%Q2.5 := %M3 XOR (FE %I1.5) ;
    
```

Instruction list language

```

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

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

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

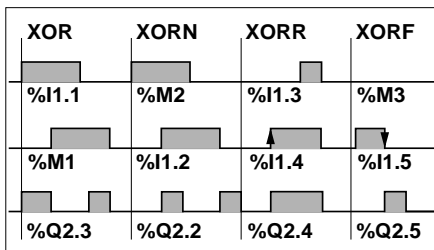
LD    %M3
XORF  %I1.5
ST    %Q2.5
    
```

Note : The parentheses are optional but make the program easier to read.

Authorized operands

Code	Operand
XOR	%I,%Q,%M,%S,%BLK,%*:Xk, %Xi
XORN	%I,%Q,%M,%S,%BLK,%*:Xk, %Xi
XORR	%I,%Q,%M
XORF	%I,%Q,%M

Timing diagram



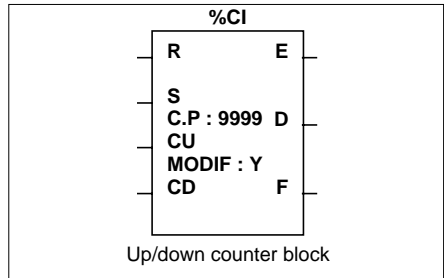
1.3 Predefined function blocks

1.3-1 Programming principles for predefined function blocks

The function blocks use bit objects and specific words.

Control system function blocks are pre-programmed in the PLC and therefore occupy a particular zone of the user memory.

In order to optimize memory occupation, the type and number of function blocks used must be defined at the outset, within the limits imposed by the system (via the Configuration and Data editors).



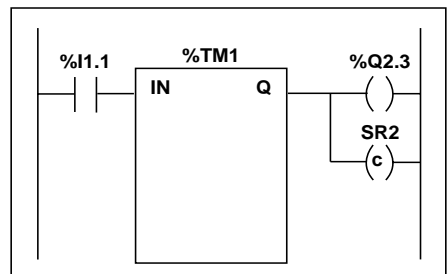
There are six types of control system function block :

Type of block	Max TSX 37	Max TSX 57	See section
Timer %Tmi	64 (1)	255 (1)	1.3-2
Up/down counter %Ci	32	255	1.3-3
Monostable %Mni	8	255	2.2-1
Register %Ri	4	255	2.2-2
Drum controller %DRi	8	255	2.2-3
Timer (Series 7) %Ti	64 (1)	255 (1)	2.2-4

(1) The total number of %Tmi + %Ti timers must be less than or equal to 64 on the TSX 37 and less than or equal to 255 on the TSX 57.

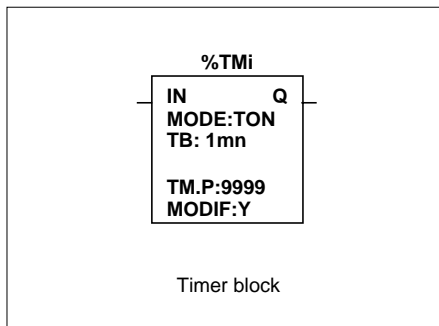
Each block contains :

- Inputs (eg : IN) which are used to control it.
- Outputs (eg : Q) which indicate its state. Each output has an associated output bit (eg : %TM1.Q) which can be tested by the user program. In addition, each output can control one or more coils (eg : %Q2.3 and SR2).
- Parameters which are used to adapt it to the application (preset, time base etc).



The parameters of function blocks (preset, current value, etc) are displayed within the block. In Instruction list language, predefined blocks are programmed using instructions (see part A, section 3.2-6).

1.3-2 Timer function block %TMI



Timers have three operating modes :

- **TON** : this mode is used to control on-delay actions. This delay is programmable and can be modified via the terminal.
- **TOF** : this mode is used to control off-delay actions. This delay is programmable and can be modified via the terminal.
- **TP** : this mode is used to create a pulse of an exact duration. This duration is programmable and can be modified via the terminal.

Characteristics

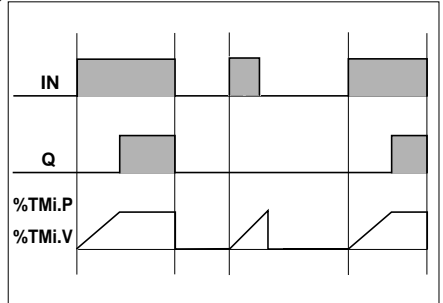
Timer number	%TMI	0 to 63 for a TSX 37, 0 to 254 for a TSX 57
Mode	TON TOF TP	<ul style="list-style-type: none"> • on-delay (default) • off-delay • monostable
Time base	TB	1min (default), 1s, 100ms, 10ms (max of 16 timers when 10ms). The smaller the time base, the greater the accuracy of the timer.
Current value	%TMI.V	Word which increments from 0 to %TMI.P when the timer is running. Can be read and tested but not written by the program (1).
Preset value	%TMI.P	$0 \leq \%TMI.P \leq 9999$. Word which can be read, tested and written by the program. It is set to 9999 by default. The time period or delay generated is equal to %TMI.P x TB.
Adjust via the terminal (MODIF)	Y/N	Y : the preset value can be modified %TMI.P in adjust mode. N : no access in adjust mode.
Setting input (instruction)	IN	The timer starts on a rising edge (TON or TP mode) or a falling edge (TOF mode).
Timer output	Q	Associated bit %TMI.Q is set to 1 depending on the function performed TON, TOF or TP.

(1) %TMI.V can be modified via the terminal.

Using as an on-delay timer : TON mode

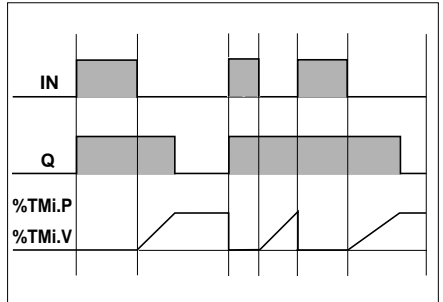
The timer is started on a rising edge at input IN: its current value %Tmi.V increases from 0 to %Tmi.P by one unit on each pulse of the time base TB. Output bit %Tmi.Q changes to 1 when the current value reaches %Tmi.P and then remains at 1 as long as input IN is at 1.

When input IN is at 0, the timer is stopped, even if its value is still changing : %Tmi.V takes the value 0.

**Using as an off-delay timer : TOF mode**

The current value %Tmi.V is set to 0 on a rising edge at input IN (even if the timer is still running). The timer is started on a falling edge at input IN.

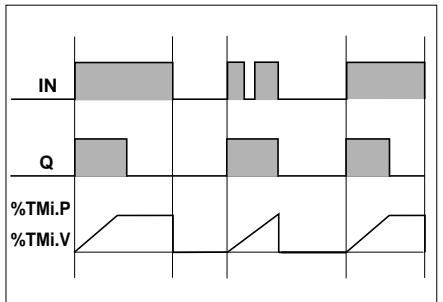
The current value increases to %Tmi.P by one unit on each pulse of the time base TB. Output bit %Tmi.Q changes to 1 when a rising edge is detected on input IN and the timer returns to 0 when the current value reaches %Tmi.P.

**Using as a monostable : TP mode**

The timer is started on a rising edge at input IN : (if the timer has not already started) its current value %Tmi.V increases from 0 to %Tmi.P by one unit on each pulse of the time base TB. Output bit %Tmi.Q changes to 1 when the timer is started and returns to 0 when the current value reaches %Tmi.P.

When input IN and output %Tmi.Q are at 0, Tmi.V takes the value 0.

This monostable cannot be reset.



Programming and configuration

Timer function blocks are programmed in the same way, irrespective of the mode of use selected. TON, TOF or TP mode can be chosen in the variables editor.

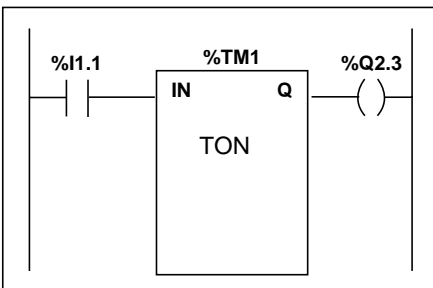
• Configuration

The following parameters must be entered in the variables editor :

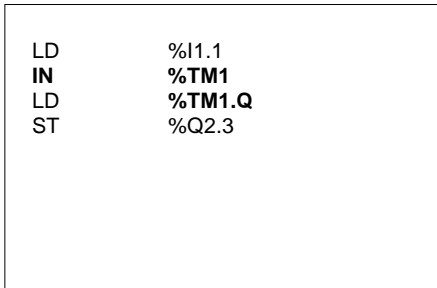
- Mode : TON, TOF or TP.
- TB : 1min, 1s, 100ms or 10ms.
- %TMi.P : 0 to 9999.
- MODIF : Y or N.

• Programming

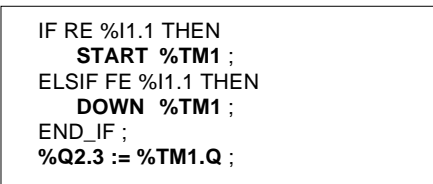
Ladder language



Instruction list language



Structured text language



The START %TMi instruction generates a rising edge on the timer block input IN (Mode TON and TP) or a falling edge on the timer block input IN (Mode TOF).

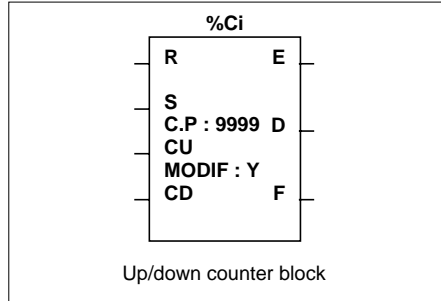
The DOWN %TMi instruction generates a falling edge on the timer block input IN (Mode TON and TP) or a rising edge on the timer block input IN (Mode TOF).

Special cases

- **Effect of a cold restart** : (%S0=1) forces the current value to 0, sets output %TMI.Q to 0 and the preset value is reset to the value defined during configuration.
- **Effect of a warm restart** : (%S1=1) has no effect on the current value of the timer nor on the preset value. The current value does not change during a power outage.
- **Effect of a PLC stop, de-activation of a task or execution of a break point** : does not freeze the current value.
- **Effect of a program jump** : the fact of not scanning the instructions where the timer block is programmed does not freeze the current value %TMI.V, which continues to increment to %TMI.P. Similarly, bit %TMI.Q associated with output Q of the timer block maintains its normal operation and can thus be tested by another instruction. However, the output wired directly to the block output is neither activated nor scanned by the PLC.
- **Testing bit %TMI.Q** : it is advisable to test bit %TMI.Q once only in the program.
- **Effect of modifying the preset %TMI.P** : modifying the preset value via an instruction or in adjust mode only takes effect when the timer is next activated : modifying the preset value in the variables editor is only taken into account after a cold restart (%S0=1).

1.3-3 Up/down counter function block %Ci

The up/down counter function block is used to upcount and downcount events. These two operations can be simultaneous.



Characteristics

Counter number	%Ci	0 to 31 for a TSX 37, 0 to 254 for a TSX 57
Current value	%Ci.V	Word incremented or decremented according to inputs CU and CD. Can be read and tested but not written by the program (1).
Preset value	%Ci.P	$0 \leq \%Ci.P \leq 9999$. Word can be read, tested and written. (default value 9999)
Adjust via terminal (MODIF)	Y/N	Y : the preset value can be modified in adjust mode. N : no access in adjust mode.
Reset input (instruction)	R	At state 1 : %Ci.V = 0.
Preset input (instruction)	S	At state 1: %Ci.V = %Ci.P.
Upcount input (instruction)	CU	Increments %Ci.V on a rising edge.
Downcount input (instruction)	CD	Decrements %Ci.V on a rising edge.
Underflow output	E (Empty)	The associated bit %Ci.E=1 when downcounter %Ci.V changes from 0 to 9999 (set to 1 when %Ci.V reaches 9999, and reset to 0 if the counter continues to downcount).(2)
Preset reached output	D (Done)	The associated bit %Ci.D=1 when %Ci.V=%Ci.P.
Overflow output	F (Full)	The associated bit %Ci.F =1 when %Ci.V changes from 9999 to 0 (set to 1 when %Ci.V reaches 0, and reset to 0 if the counter continues to upcount).

(1) %Ci.V can be modified via the terminal.

(2) When there is an upcount overflow or downcount underflow, bit %S18 changes to 1.

Operation

- **Upcount** : when a rising edge appears at the upcounting input CU, the current value is incremented by one unit. When this value is equal to the preset value %Ci.P, the "preset reached" output bit %Ci.D assigned to output D changes to state 1. Output bit %Ci.F (upcount overflow) changes to state 1 when %Ci.V changes from 9999 to 0, and is reset to 0 if the counter continues to upcount.
- **Downcount** : when a rising edge appears at the downcounting input CD, the current value %Ci.V is decremented by one unit. Output bit %Ci.E (downcount underflow) changes to state 1 when %Ci.V changes from 0 to 9999, and is reset to 0 if the counter continues to downcount.
- **Up/down count** : to use both the upcount and the downcount functions simultaneously, the two corresponding inputs CU and CD must be controlled. These two inputs are then scanned in succession. If they are both at 1 simultaneously, the current value remains unchanged.
- **Reset** : when input R is set to state 1, the current value %Ci.V is forced to 0, and outputs %Ci.E, %Ci.D and %Ci.F are at 0. The "reset" input has priority.
- **Preset** : if "preset" input S is at state 1 and the "reset" input R is at state 0, the current value %Ci.V takes the value %Ci.P, and output %Ci.D is set to 1.

Note

On resetting (input R or instruction R) :

- In Ladder language, the logs of inputs CU and CD are updated with the wired values.
- In Instruction list language and Structured text language, the logs of inputs CU and CD are not updated. Each one maintains the value it had before being called.

Special cases

- **Effect of a cold restart** : (%S0=1)
 - The current value %Ci.V is set to zero.
 - Output bits %Ci.E, %Ci.D and %Ci.F are set to zero.
 - The preset value is initialized with the value defined during configuration.
- **Effect of a warm restart (%S1=1), a PLC stop, de-activation of a task or execution of a break point** : this has no effect on the current value of the counter (%Ci.V).
- **Effect of modifying the preset %Ci.P** : modifying the preset value via an instruction or in adjust mode takes effect when the block is processed by the application (activation of one of the inputs).

Configuration and programming

Counting of a number of items = 5000. Each pulse on input %I1.2 (when internal bit %M0 is at 1) increments the upcounter %C8 up to its final preset value (bit %C8.D=1). The counter is reset by input %I1.1.

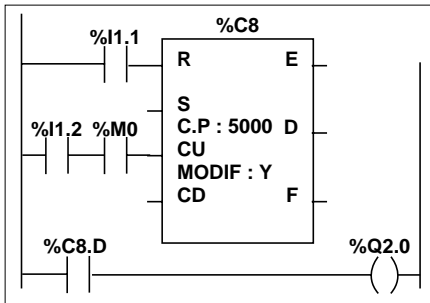
• Configuration

The following parameters must be entered via the variables editor :

- %Ci.P, set to 5000 in this example,
- MODIF : Y.

• Programming

Ladder language



Instruction list language

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

Structured text language

```
IF %I1.1 THEN
  RESET %C8 ;
END_IF ;
%M1 := %I1.2 AND %M0 ;
IF RE %M1 THEN
  UP %C8 ;
END_IF ;
%Q2.0 := %C8.D;
```

In Structured text language, 4 instructions are used to program the up/down counter function blocks :

- RESET %Ci : Resets the current value,
- PRESET %Ci : Loads the preset value into the current value,
- UP %Ci : Increments the current value,
- DOWN %Ci : Decrements the current value.

The CU and CD input logs are reset when the UP and DOWN instructions are used in Structured text language. The user must therefore manage the rising edges for these two instructions.

1.4 Numerical processing on integers

1.4-1 General

The numerical instructions described in this section apply to objects of the following type :

- bit tables,
- words,
- double words.

Instructions for other types of object are described in the section "Description of instructions and advanced functions".

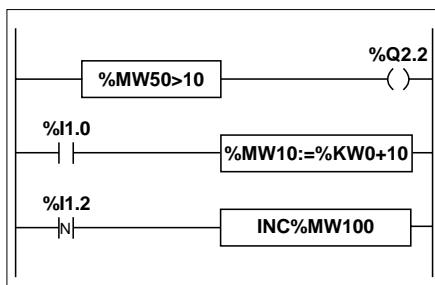
In Ladder language

Numerical instructions are entered in blocks :

- located in the test zone for comparison blocks.
- located in the action zone for operation blocks.

These blocks can contain :

- a simple expression, eg :
OP3:=OP1+OP2,
- a complex expression, eg :
OP5:=(OP1+OP2)*OP3-OP4.



In Instruction list language

Instructions are placed between square brackets.

They are executed if the Boolean result of the test instruction preceding the numerical instruction is at 1.

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

In Structured text language

Numerical instructions are entered directly. The conditional instruction IF enables numerical instructions to be conditioned via a Boolean expression.

```
%Q2.2 := %MW50 > 10 ;
IF %I1.0 THEN
    %MW10 := %KW0 + 10 ;
END_IF ;
IF FE %I1.2 THEN
    INC %MW100 ;
END_IF ;
```

List of operands

List of bit tables

Abbreviations	Full addressing	Type of word	Access
%M:L	%Mi:L	table of internal bits	R/W
%I:L	%Ixy.i:L	table of input bits	R/W
%Q:L	%Qxy.i:L	table of output bits	R/W
%Xi:L	%Xi:L	table of step bits	R

List of single format words

Abbreviations	Full addressing	Type of word	Access	Indexed form
Immed. val.	-	immediate values	R	-
%MW	%MWi	internal word	R/W	%MWi[%MWj]
%KW	%KWi	internal constant	R	%KWi[%MWj]
%SW	%SWi	system word	R/W (1)	-
%IW	%IWxy.i(.r)	input word	R	-
%QW	%QWxy.i(.r)	output word	R/W	-
%NW	%NW{j}k	common word	R/W	-
%BLK	eg : %Tmi.P	function block extract word	R/W (2)	-
%Xi.T	%Xi.T	step activity time	R	-

(1) Write depending on i. (2) Write depending on the type of word, for example : preset values (%Ci.P can be written, whereas the current values %Ci.V can only be read).

List of double words

Abbreviations	Full addressing	Type of double word	Access	Indexed form
Immed. val.	-	immediate values	R	-
%MD	%MDi	internal double word	R/W	%MDi[%MWj]
%KD	%KDi	internal double constant	R	%KDi[%MWj]
%SD	%SDi	system double word	R/W (1)	-
%ID	%IDxy.i(.r)	input double word	R	-
%QD	%QDxy.i(.r)	output double word	R/W	-

(1) Only double word %SD18.

Notes

There are other types of words and double words, such as %MWxy.i %KWxy.i and %MDxy.i %KDxy.i associated with applications. These double words behave like the words and double words %MWi %KWi and %MDi %KDi respectively.

Implicit conversion of words <--> double words

PL7 software allows mixing of operations using words and double words. Conversion to one or other of the formats is performed implicitly. An operation involving a double word or several immediate values is automatically performed internally in double format.

1.4-2 Comparison instructions

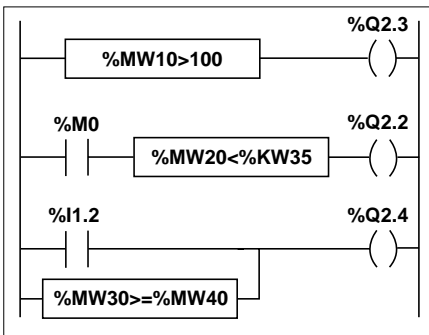
Comparison instructions are used to compare two operands.

- > : test if operand 1 is greater than operand 2.
- >= : test if operand 1 is greater than or equal to operand 2.
- < : test if operand 1 is less than operand 2.
- <= : test if operand 1 is less than or equal to operand 2.
- = : test if operand 1 is equal to operand 2.
- <> : test if operand 1 is different from operand 2.

The result is 1 when the comparison requested is true.

Structure

Ladder language



Comparison blocks are programmed in the test zone.

Instruction list language

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

The comparison is executed inside square brackets following instructions LD, AND and OR.

Structured text language

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

Note : The parentheses are optional but make the program easier to read.

Syntax

Operators : >, >=, <, <=, =, <>

Op1 Operator Op2

Operands

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

Notes

- In Ladder language, comparisons can also be executed using the vertical comparison block (see part B, section 2.3).
- In Instruction list language, comparison instructions can be used in parentheses.

1.4-3 Assignment instructions

These are used to load an Op2 operand into an Op1 operand.

Syntax : Op1:=Op2 <=> Op2->Op1

The following assignment operations can be performed :

- On bit tables
- On words or double words.

Several assignment instructions can be linked within the same block :
Op1:=Op2:=Op3:=Op4:=...

Assignment of bit tables (see bit table object, section 1.2-6, part A)

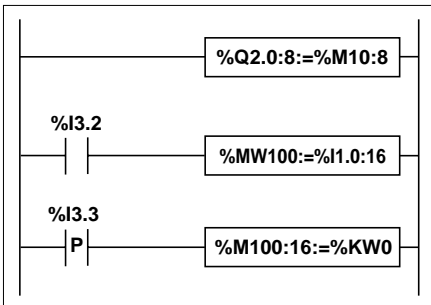
The following operations on bit tables can be performed :

- Bit table -> bit table example 1
- Bit table -> word or double word (indexed) example 2
- Word or double word (indexed) -> bit table example 3

Structure

Ladder language

Instruction list language



```
LD TRUE
[%Q2.0:8:= %M10:8] example 1

LD %I3.2
[%MW100:= %I1.0:16 ] example 2

LDR %I3.3
[%M100:16:=%KW0] example 3
```


Structured text language

```

%Q2.0:8 := %M10:8 ;      example 1
IF %I3.2 THEN
  %MW100 := %I1.0:16 ;  example 2
END_IF ;
IF RE %I3.3 THEN
  %M100:16 := %KW0 ;    example 3
END_IF ;

```

Syntax

Operator :=

Op1:=Op2

Operands

Type	Operand 1 (Op1)	Operand 2 (Op2)
Bit table	%M:L,%Q:L,%I:L	%M:L,%Q:L,%I:L, %Xi:L
Indexable words	%MW	%MW,%KW
Non-indexable words	%QW,%SW,%NW %BLK	Immed.val.,%IW,%QW,%SW %NW,%BLK, %Xi.T, Num. expr.
Indexable double words	%MD	%MD,%KD
Non-indexable double words	%QD,%SD,	Immed.val.,%ID,%QD,%SD Num. expr.

Rules for use

- The source and target bit tables are not necessarily the same length. If the source table is longer than the target table, only the least significant bits will be transferred. Otherwise the target table is completed with 0s.
- Example of assigning a bit table -> a word (or double word) : the bits in the table are transferred into the word (the low-order word for a double word) starting from the right (first bit of the table to bit 0 of the word). The bits of the word which are not transferred (length<16 or 32) are set to 0.
- Example of assigning a word -> bit table : the bits of the word are transferred starting from the right (bit 0 of the word to the first bit of the table).

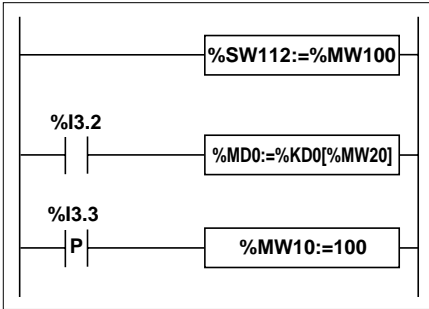
Assignment of words

The following assignment operations on words can be performed :

- word (indexed) -> word (indexed) or double word (indexed) example 1
- double word (indexed) -> double word (indexed) or word (indexed) example 2
- immediate value -> word (indexed) or double word (indexed) example 3

Structure

Ladder language



Instruction list language

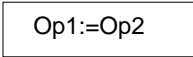
```
LD TRUE
[%SW112 := %MW100] example 1
LD %I3.2
[%MD0:= %KD0[%MW20] ] example 2
```

Structured text language

```
IF RE %I3.3 THEN
    %MW10 := 100 ; example 3
END_IF ;
```

Syntax

Operator :=



Operands

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

Note

Word <--> double word conversions are performed implicitly during double word --> word assignment. If the value of the double word cannot be contained in the word, bit %S18 is set to 1.

It is possible to execute multiple assignments.

Example : %MW0 := %MW2 := %MW4

Note that in the example %MD14 := %MW10 := %MD12, it is not necessarily true that %MD14 := %MD12, as the higher order word of the double word will be lost when assignment to %MW10 occurs, due to the conversion from a double word to a single word.

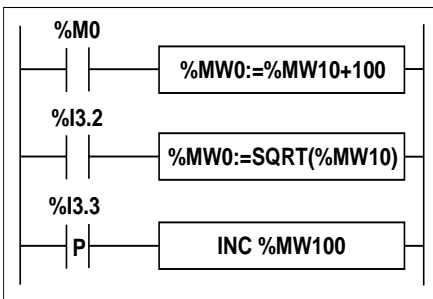
1.4-4 Arithmetic instructions on integers

These instructions are used to perform arithmetic operations between two operands or on one operand.

- | | |
|----------------------------------|---|
| + : add two operands | SQRT : square root of an operand |
| - : subtract two operands | INC : increment of an operand |
| * : multiply two operands | DEC : decrement of an operand |
| / : divide two operands | ABS : absolute value of an operand |
- REM** : remainder of division of the two operands

Structure

Ladder language



Instruction list language

```

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

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

LDR     %I3.3
[INC %MW100]
  
```

Structured text language

```

IF %M0 THEN
    %MW0 := %MW10 + 100 ;
END_IF ;
IF %I3.2 THEN
    %MW0 := SQRT (%MW10) ;
END_IF ;
IF RE %I3.3 THEN
    INC %MW100 ;
END_IF ;
  
```

Syntax

Operators

- **+, -, *, /, REM**

Op1:=Op2 Operator Op3

- **SQRT, ABS**

Op1:=Operator(Op2)

- **INC, DEC**

Operator Op1

Operands

Type	Operand 1 (Op1)	Operands 2 & 3 (Op2 and 3)
Indexable words	%MW	%MW,%KW
Non-indexable words	%QW,%SW,%NW, %BLK	Immed.val.,%IW,%QW,%SW %NW,%BLK,%Xi.T, Num. expr.
Indexable double words	%MD	%MD,%KD
Non-indexable double words	%QD,%SD,	Immed.val.,%ID,%QD,%SD Numerical expr.

Note :

The operations INC and DEC cannot be used in numerical expressions.

Rules for use

- **Addition : Overflow during operation**

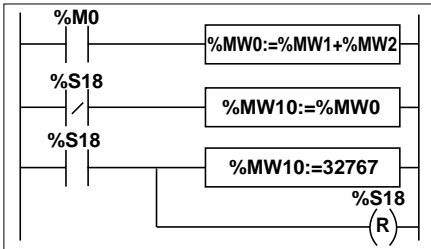
If the result exceeds the limits :

- 32768 or +32767 for a single length operand,
 - 2 147 483 648 or +2 147 483 647 for a double length operand,
- bit %S18 (overflow) is set to 1. The result is therefore not significant. The user program manages bit %S18.

Example :

Ladder language

Instruction list language



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

Structured text language

```
IF %M0 THEN
  %MW0 := %MW1 + %MW2 ;
END_IF ;
IF %S18 THEN
  %MW10 := 32767 ; RESET %S18 ;
ELSE
  %MW10 := %MW0 ;
END_IF ;
```

Where %MW1 = 23241 and %MW2 = 21853, the real result (45094) cannot be expressed in a 16-bit word. Bit %S18 is set to 1 and the result obtained (-20442) is incorrect. In this example, when the result is greater than 32767, its value is set to

- **Multiplication :**

Overflow during operation.

If the result exceeds the capacity of the result word, bit %S18 (overflow) is set to 1 and the result is not significant.

- **Division/Remainder :**

Division by 0.

If the divider is 0, division is impossible and system bit %S18 is set to 1. The result is then incorrect.

Overflow during operation.

- **Square root extraction :**

Square root extraction is only performed on positive values. Thus, the result is always positive. If the square root operand is negative, system bit %S18 is set to 1 and the result is incorrect.

Note :

- When the result of an operation is not an integer (in the case of division or square root extraction), the result is rounded down to the nearest integer.
- The sign of the remainder (REM) is that of the numerator.
- The user program is responsible for managing system bit %S18. It is set to 1 by the PLC and must be reset by the program so that it can be re-used (see previous page for example).

1.4-5 Logic instructions

The associated instructions are used to perform a logic operation between two operands or on one operand.

AND : AND (bit-wise) between two operands

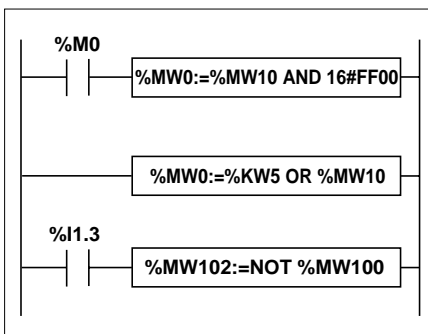
OR : Logic OR (bit-wise) between two operands

XOR : Exclusive OR (bit-wise) between two operands

NOT : Logic complement (bit-wise) of an operand

Structure

Ladder language



Instruction list language

```
LD %M0
[%MW0 := %MW10 AND 16#FF00]

LD TRUE
[%MW0 := %KW5 OR %MW10]

LD %I1.3
[%MW102:= NOT %MW100]
```

 Structured text language

```

IF %M0 THEN
  %MW0 := %MW10 AND 16#FF00 ;
END_IF ;
%MW0 := %KW5 OR %MW10 ;
IF %I1.3 THEN
  %MW102 := NOT %MW100 ;
END_IF ;
  
```

Syntax

Operators

- **AND,OR,XOR,**

Op1:=Op2 Operator Op3

- **NOT,**

Op1:=NOT Op2

Operands

Type	Operand 1 (Op1)	Operands 2 & 3 (Op2 and 3)
Indexable words	%MW	%MW,%KW
Non-indexable words	%QW,%SW,%NW, %BLK	Immed. val.,%IW,%QW,%SW %NW,%BLK,%Xi.T,Num. expr.
Indexable double words	%MD	%MD,%KD
Non-indexable double words	%QD,%SD	Immed. val.,%ID,%QD,%SD Num. expr.

1.4-6 Numerical expressions

Numerical expressions are composed of several numerical operands and the arithmetic and logic operators described above.

Example : `%MW25 * 3 - SQRT(%MW10) + %KW8* (%MW15 + %MW18) AND 16#FF`

The number of operators and operands in an arithmetic expression is not limited.

Numerical expressions on integer objects

Operands in the same numerical expression can be both single or double length.

Example : `%MW6 * %MW15 + SQRT(%DW6) / (%MW149[%MW8]) + %KD29) AND 16#FF`

An operand or an operation to a single operand can be preceded by the sign + or - (by default, the sign +).

Example : `SQRT (%MW5) * - %MW9`

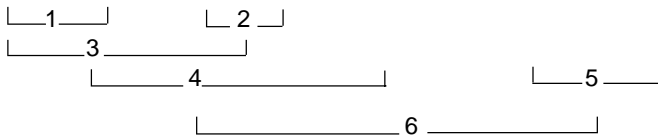
All word objects can be used in arithmetic expressions. Certain words can be indexed.

Execution priority of instructions

In numerical expressions, priority of different instructions is observed. They are executed in the order described below :

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

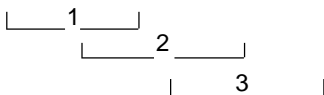
In the example below, the instructions are executed following the order of numbering :
`SQRT (%MW3) + %MW5 * 7 AND %MW8 OR %MW5 XOR %MW10`



Parentheses

Parentheses are used to modify the order in which priorities are executed. Their use is recommended for structuring numerical expressions.

`((%MW5 AND %MW6) + %MW7) * %MW8`



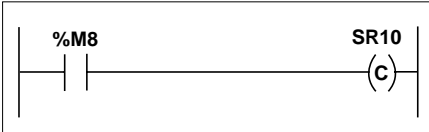
1.5 Program instructions

1.5-1 Subroutine call

Subroutine call instructions are used to call a subroutine module located in the same task.

Structure

Ladder language



Instruction list language

```
LD %M8
SR10
```

Structured text language

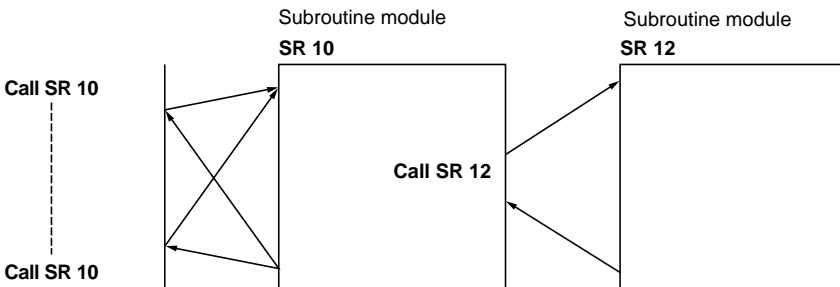
```
IF %M8 THEN
  SR10 ;
END_IF ;
```

SR_i represents the subroutine module called : i (number from 0 to 253).

Rules

- A subroutine can only be called if the subroutine module has already been created.
- A subroutine return is performed on the action immediately following the subroutine call instruction.
- A subroutine can call another subroutine. The number of cascaded calls is limited to 8.
- Subroutines are assigned to a task. They can only be called from within the same task.

Principle

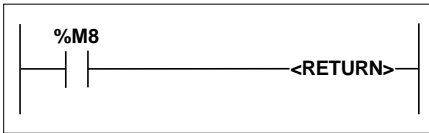


1.5-2 Subroutine return

Subroutine return instructions are reserved for subroutine modules and are used to return to the calling module, if the Boolean result of the preceding test instruction is 1.

Structure

Ladder language



Instruction list language

```
LD    %M8
RETC
```

Structured text language

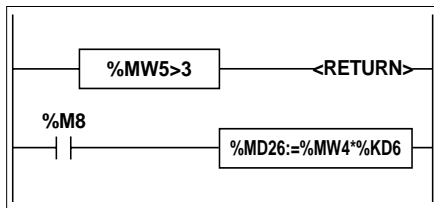
```
IF %M8 THEN
  RETURN ;
END_IF ;
```

Rule for use

Subroutine return instructions are implicit at the end of each subroutine, but can be used to return to the calling module before the end of the subroutine.

Example :

Ladder language



Instruction list language

```
LD    [%MW5>3]
RETC
LD    %M8
[%MD26:=%MW4*%KD6]
```

Structured text language

```
IF (%M5 > 3) THEN
  RETURN ;
END_IF ;
IF %M8 THEN
  %MD26 := %MW4 * %KD6 ;
END_IF ;
```

Instruction list language includes the following additional instructions :

RETCN : subroutine return if the Boolean result of the preceding test instruction is 0.
RET : unconditional subroutine return.

1.5-3 Program jumps

Jump instructions allow connection to a line of programming identified by a label %Li :

JMP : unconditional program jump,

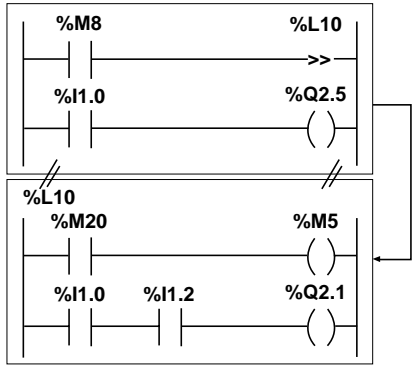
JMPC : program jump if the Boolean result of the preceding test instruction is 1,

JMPCN : program jump if the Boolean result of the preceding test instruction is 0,

%Li represents the label of the line to which the connection is made (i numbered from 1 to 999 with up to 256 labels).

Structure

Ladder language



Instruction list language

```

LD    %M8
JMPC %L10
LD    %I1.0
ST    %Q2.5
.....
%L10 :
LD    %M20
ST    %M5
LD    %I1.0
AND   %I1.2
ST    %Q2.1
    
```

Jump to label %L10 if %M8 is at 1.

Structured text language

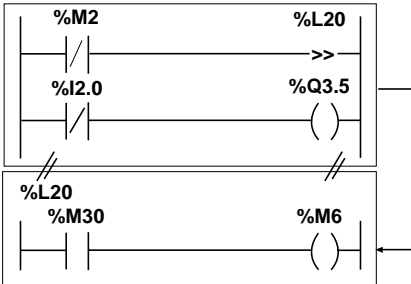
```

IF %M8 THEN
    JUMP %L10 ;
END_IF ;
%Q2.5 := %I1.0 ;

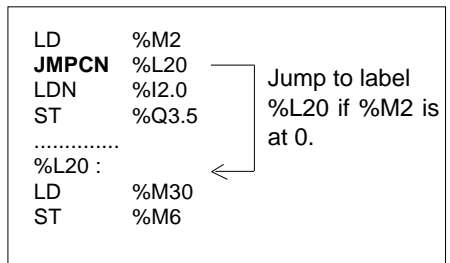
%L10 :
    %M5 := %M20 ;
    %Q2.1 := %I1.0 AND %I1.2 ;
    
```

Jump to label %L10, if %M8 is at 1.

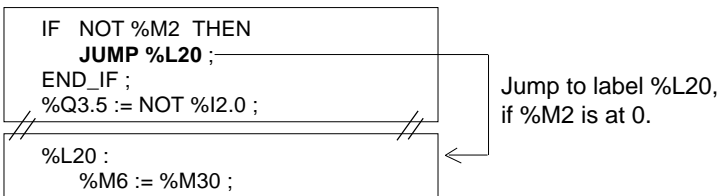
Ladder language



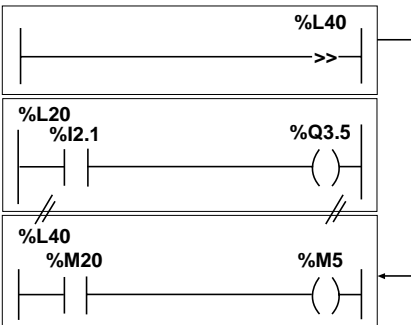
Instruction list language



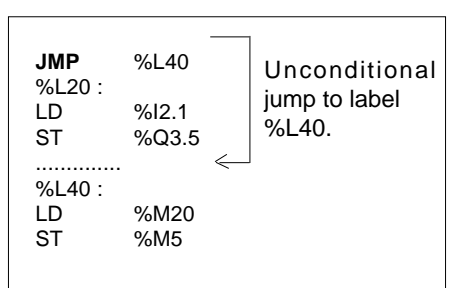
Structured text language



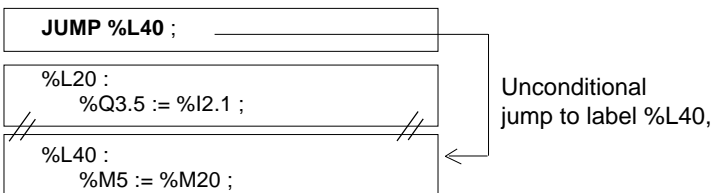
Ladder language



Instruction list language



Structured text language



Rules

- A program jump is performed within the same programming entity (main module of a master task (MAIN), subroutine %SRI, etc).
- A program jump is performed to a line of programming which is downstream or upstream.

When the jump is upstream, attention must be paid to the program scan time : the program scan time is then extended and can mean the task period including the upstream jump is exceeded.

1.5-4 Program end instructions

The end of the execution of a program scan is defined using the instructions END, ENDC and ENDCN :

END : unconditional end of program

ENDC : end of program if the Boolean result of the preceding test instruction is 1.

ENDCN : end of program if the Boolean result of the preceding test instruction is 0.

By default (normal mode), when the end of program is activated, the outputs are updated and the next scan is started.

If scanning is periodic, the outputs are updated when the end of period is reached and the next scan is started.

Note :

These instructions can only be used in Instruction list language in the master task.

Example :

Instruction list language

```
LD    %M1
ST    %Q2.1
LD    %M2
ST    %Q2.2
.....
END
```

```
LD    %M1
ST    %Q2.1
LD    %M2
ST    %Q2.2
.....
LD    %I1.2
ENDC  → If %I1.2 =1, end of
LD    %M2  program scanning.
ST    %Q2.2 If %I1.2 =0, continues
.....  program scanning until
END      new END instruction.
```

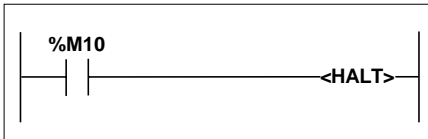
1.5-5 Stop program

Execution of an application program can be stopped using the instruction HALT (stops all tasks). This freezes the variable objects in this program.

A program stopped in this way must be initialized to restart it (using the PL7 command INIT). Any instructions following the instruction HALT will therefore not be executed.

Structure

Ladder language



Instruction list language

```
LD %M10  
HALT
```

Structured text language

```
IF %M10 THEN  
    HALT ;  
END_IF ;
```

1.5-6 Event masking/unmasking instructions

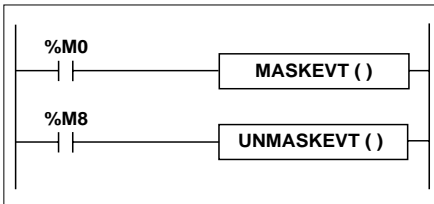
Event masking/unmasking instructions are used to mask or unmask all the events which activate event-triggered tasks.

MASKEVT : masks all events. The events are stored by the PLC. However, the associated event-triggered tasks remain inactive as long as the masking operation is enabled (until the next UNMASKEVT instruction).

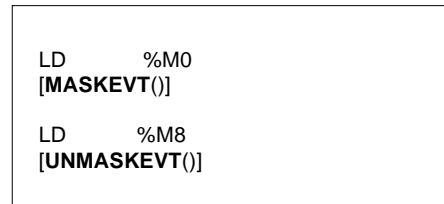
UNMASKEVT : unmasks all events. The events which were stored during the mask period are processed. The event processing mechanism is operational until the next MASKEVT instruction.

Structure

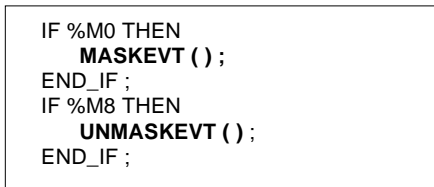
Ladder language



Instruction list language



Structured text language



1.5-7 NOP Instruction

The **NOP** instruction does not perform any operation. It is used for "reserving" lines in a program which allow the user to insert instructions later without modifying the line numbers.

2.1 Presentation of advanced instructions

2.1-1 General

The instructions described in this section are suitable for advanced programming requirements.

They have the same effect, irrespective of the language used. Only the syntax differs.

They are :

- Either basic instructions of the software.
- Or Functions considered as extensions of the software.

Extended Function type instructions are used to enhance the basic software using special programming instructions.

- Operations on character strings, word tables, etc.
- Application-specific functions : Communication, PID control, Man-Machine Interface, etc.

They include the following families :

- Character strings.
- Word tables.
- Management of Dates, times and time periods.
- Conversions.
- Bit tables.
- "Orphee" functions.

- Communication.
- PID control.
- Man-Machine Interface.
- Motion control.

==>

see the relevant application-specific function

Notes on programming

Function type instructions require additional application memory occupation (only when they are actually used in the program).

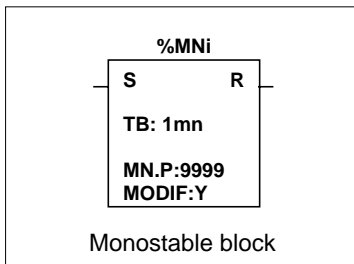
This memory occupation should be taken into account by the programmer for each function, irrespective of the number used, and the maximum memory size of the PLC used must be observed.

2.2 Advanced predefined function blocks

2.2-1 Monostable function block %MNI

The monostable function block is used to create a pulse of an exact duration.

This duration is programmable and can be modified via the terminal.



Characteristics

Number	%MNI	0 to 7 for a TSX 37, 0 to 254 for a TSX 57
Time base	TB	1min, 1s, 100ms, 10ms (1min by default).
Current value	%MNI.V	Word which decreases from %MNI.P to 0 when the timer is running. Can be read and tested but not written.
Preset value	%MNI.P	0 < %MNI.P ≤ 9999. Word which can be read, tested and written. The duration of the pulse (PRESET) is equal to : %MNI.P x TB.
Edit via terminal MODIF	Y/N	Y : possibility of changing the preset value in adjustment mode. N : no access in adjustment mode.
Start input (or instruction)	S(Start)	On a rising edge %MNI.V = %MNI.P then %MNI.V decreases to 0.
Monostable output	R(Running)	The associated bit %MNI.R is at 1 if %MNI.V > 0 (monostable running) %MNI.R = 0 if %MNI.V = 0.

Operation

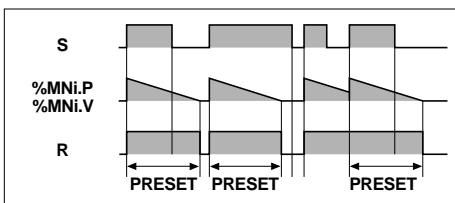
When input S of the monostable is at 1 (rising edge), the current value %MNI.V takes the preset value %MNI.P and decreases to 0 by one unit on each pulse of the time base TB. Output bit %MNI.R (Running) assigned to output R changes to 1 when the current value %MNI.V is other than 0.

When the current value %MNI.V = 0, output bit %MNI.R returns to 0.

Start input S :

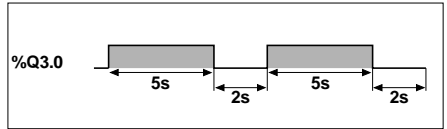
Current value %MNI.V :

Running output R :



Programming and configuration

- **Example of use** : flashing at variable cyclical periods : the preset value of each monostable defines the duration of each pulse.



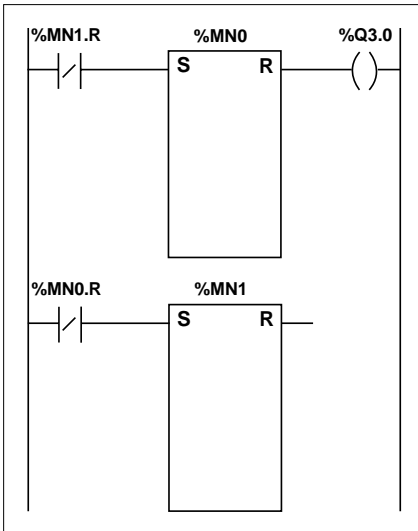
- **Configuration**

The following parameters must be entered in the variables editor :

- TB : 1min, 1s, 100ms, 10ms or 1ms (100ms in this example).
- %MNi.P : 0 to 9999 (%MN0.P=50 and %MN1.P=20 in this example).
- MODIF : Y or N.

- **Programming**

Ladder language



Instruction list language

```
LDN    %MN1.R
ANDN   %Q3.0
S      %MN0
LD     %MN0.R
ST     %Q3.0
LDN    %MN0.R
S      %MN1
```

Structured text language

```
%M0:=NOT %MN1.R ;
IF RE %M0 THEN
  START %MN0 ;
END_IF ;
%Q3.0 := %MN0.R ;
%M1 := NOT %MN0.R ;
IF RE %M1 THEN
  START %MN1 ;
END_IF ;
```

In the example above, output %Q3.0 is set to 1 for 5s (%MN0.P) and reset to 0 for 2s (%MN1.P).

In Structured text language, the instruction **START %Mni** is used to start the monostable function block. This instruction forces a rising edge on input S of the block and thus reinitializes the function block. The conditioning instruction must therefore be an edge-type.

Note

The monostable function can also be performed by the function block %TMi in TP mode (see part B, section 1.3-2).

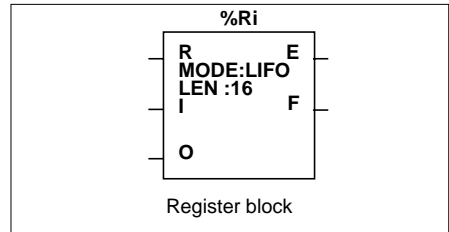
Special cases

- **Effect of a cold restart** : (%S0 = 1) the preset value %MNi.P is loaded into the current value %MNi.V. Since the preset value which may have been modified by the terminal is lost, output %MNi.R is reset to 0.
- **Effect of a warm restart** : (%S1) has no effect on the current value of the monostable (%MNi.V).
- **Effect of a PLC stop, de-activation of the task and break point** : a PLC stop, de-activation of the current task or execution of a break point does not freeze the current value.
- **Effect of a program jump** : the fact of not scanning the rung where the monostable block is programmed does not freeze current value %MNi.V which continues to decrease to 0.
Similarly, bit %MNi.R assigned to the monostable block output continues to operate normally and can thus be tested in another rung.
However, the coils directly "connected" to the block output (eg %Q3.0) will not be activated since they are not scanned by the PLC.
- **Testing bit %MNi.R** : the state of this bit can change during a scan.

2.2-2 Register function block %Ri

A register is a memory block which is used to store up to 255 words of 16 bits in two different ways :

- Queue (first in, first out) known as FIFO stack (First In, First Out).
- Stack (last in, first out) known as LIFO stack (Last In, First Out).



Characteristics

Register number	%Ri	0 to 3 for a TSX 37, 0 to 254 for a TSX 57
Mode	FIFO LIFO	Queue. Stack (default selection).
Length	LEN	Number of 16-bit words ($1 \leq \text{LEN} \leq 255$) in the register memory block.
Input word	%Ri.I	Register input word. Can be read, tested and written.
Output word	%Ri.O	Register output word. Can be read, tested and written.
Storage input (or instruction)	I (In)	On a rising edge, stores the contents of word %Ri.I in the register.
Retrieval input (or instruction)	O (Out)	On a rising edge, loads a data word into word %Ri.O.
Reset input (or instruction)	R (Reset)	At state 1 initializes the register.
Empty output	E (Empty)	The associated bit %Ri.E indicates that the register is empty. Can be tested.
Full output	F (Full)	The associated bit %Ri.F indicates that the register is full. Can be tested.

Note :

When the two inputs I and O are activated simultaneously, storage is performed before retrieval.

FIFO (First In, First Out)

The first data item entered is the first to be retrieved.

When a storage request is received (rising edge at input I or activation of instruction I), the contents of input word %Ri.I (which have already been loaded) are stored at the top of the stack (fig a).

When the stack is full (output F=1), no further storage is possible and system bit %S18 changes to 1.

When a retrieval request is received (rising edge at input O or activation of instruction O) the data word lowest in the stack is loaded into output word %Ri.O and the contents of the register are moved down one place in the stack (fig.b).

When the register is empty (output E=1), no further retrieval is possible. Output word %Ri.O does not change and retains its value. The stack can be reset at any time (state 1 at input R or activation of instruction R).

LIFO (Last In, First Out)

The last data item entered is the first to be retrieved.

When a storage request is received (rising edge at the input or activation of instruction I), the contents of input word %Ri.I (which have already been loaded) are stored at the top of the stack (fig c).

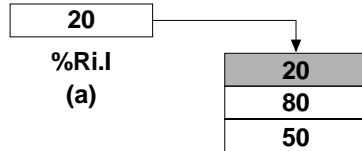
When the stack is full (output F=1), no further storage is possible and system bit %S18 changes to 1.

When a retrieval request is received (rising edge at input O or activation of instruction O) the highest data word (last word to be entered) is loaded into output word %Ri.O (fig.d).

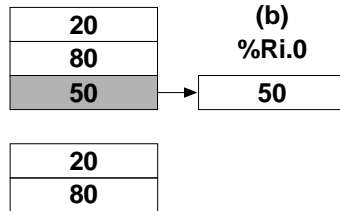
When the register is empty (output E=1), no further retrieval is possible. Output word %Ri.O does not change and retains its last value. The stack can be reset at any time (state 1 at input R or activation of instruction R). The element indicated by the pointer is then the highest in the stack.

Example :

Storage of the contents of %Ri.I at the top of the stack.

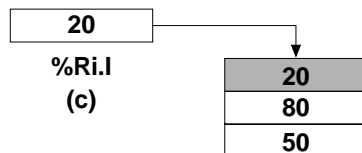


Retrieval of the first data item which is then loaded into %Ri.O.

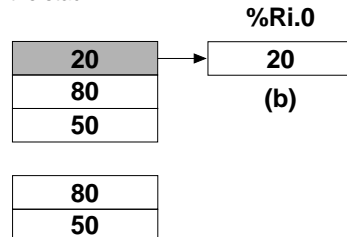


Example :

Storage of the contents of %Ri.I at the top of the stack.



Retrieval of the data word highest in the stack.



Programming and configuration

• Configuration

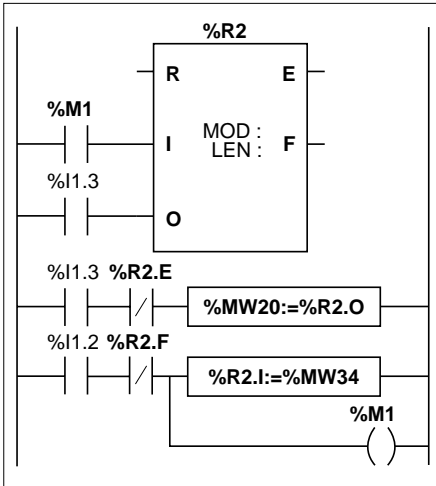
The following parameters must be entered in the configuration editor :

- Number : 1 to 4 for a TSX 37, 1 to 255 for a TSX 57,
- Length : 1 to 255.

The operating mode (FIFO or LIFO) must be entered in the variables editor.

• Programming

Ladder language



Instruction list language

```
LD    %M1
I     %R2
LD    %I1.3
O     %R2
LD    %I1.3
ANDN  %R2.E
[%MW20:=%R2.O]
LD    %I1.2
ANDN  %R2.F
[%R2.I:=%MW34]
ST    %M1
```

Structured text language

```
IF RE %M1 THEN
  PUT %R2 ;
END_IF ;
IF RE %I1.3 THEN
  GET %R2 ;
END_IF ;
IF (%I1.3 AND NOT %R2.E) THEN
  %MW20 := %R2.O ;
END_IF ;
%M1 := %I1.2 AND NOT %R2.F ;
IF %M1 THEN
  %R2.I := %MW34 ;
END_IF ;
```

The programming example shows word %MW34 being loaded into %R2.I at the storage request %I1.2, if register R2 is not full (%R2.F=0). The storage request in the register is made by %M1. The retrieval request is made by input %I1.3 and %R2.O is loaded into %MW20 if the register is not empty (%R2.E=0).

In Structured text language, 3 instructions are used to program the register function blocks :

- RESET %Ri : Initializes the register,
- PUT %Ri : Stores the contents of word %Ri.I in the register,
- GET %Ri : Loads a data word into word %Ri.O.

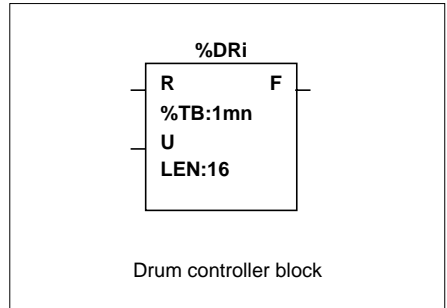
The PUT and GET instructions create a rising edge on inputs I and O respectively of the function block. The conditioning instruction must therefore be an edge-type.

Special cases

- **Effect of a cold restart** : (%S0=1) initializes the contents of the register. The Output bit %Ri.E assigned to output E is set to 1.
- **Effect of a warm restart** : (%S=1) has no effect on the contents of the register, nor on the state of its output bits.
- When resetting to 0 (input R or instruction R)
 - In Ladder language, the memorized values of inputs I and O are updated with the actual values.
 - In Instruction list language, the memorized values of inputs I and O are not updated. Each one retains the value it had before being called.
 - In Structured text language, the memorized values of inputs I and O are updated with 0.

2.2-3 Drum controller function block %DRi

The drum controller operates on a similar principle to an electromechanical drum controller, which changes step according to external events. On each step, the high point of a cam gives an order which is executed by the control system. In the case of a drum controller, these high points are symbolized by state 1 for each step and are assigned to output bits %Qi.j or internal bits %Mi, known as control bits.



Characteristics

Number	%DRi	0 to 7 for a TSX 37, 0 to 254 for a TSX 57
Step number	LEN	1 to 16 (default 16).
Time base	TB	1min, 1s, 100ms, 10ms (default 1min).
Time envelope or time period of current step	%DRi.V	0 ≤ %DRi.V ≤ 9999. Word is reset on each step change. Can be read and tested but not written. The duration is equal to %DRi.V x TB.
Current step number	%DRi.S	0 ≤ %DRi.S ≤ 15. Word which can be read and tested. Can only be written with an immediate value.
Return to step 0 input	R (RESET)	At state 1 initializes the drum controller to step 0.
Advanced input	U (UP)	On a rising edge, causes the drum controller to advance by one step and updates the control bits.
Output	F (FULL)	Indicates that the current step equals the last step defined. The associated bit %DRi.F can be tested (%DRi.F=1 if %DRi.S=configured step number - 1).
State of a step	%DRi.Wj	16-bit word defining the states of step j of drum controller i. Can be read and tested but not written.
Control bits		Outputs or internal bits associated with the step (16 control bits).

Note : Bit %S18 changes to 1, if a non-configured step is written.

Operation

The drum controller comprises :

- A matrix of constant data (the cams) organized :
 - in columns : in steps from 0 to N-1 (N is the step number configured). Each column shows the states of the step in the form of 16 data bits numbered 0 to F.
- A list of control bits (1 per line) corresponding to either outputs %Qxy.i, or to internal bits %Mi. During the current step, the control bits take on the binary states defined for this step.

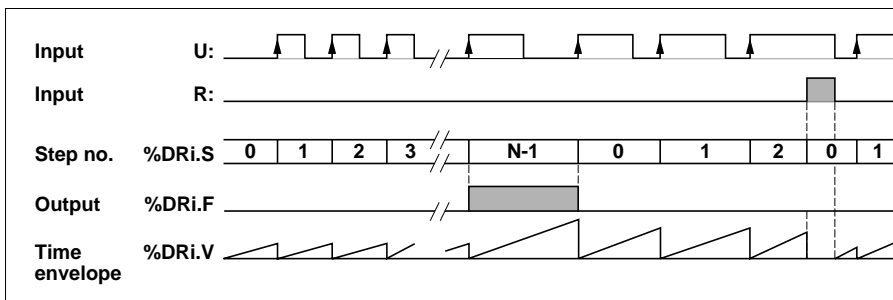
The table below summarizes the main characteristics of the drum controller (controller configured with 16 steps).

	Step																Address
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
	0	0	0	1	0	1	0	1	0	1	1	0	1	1	0	0	%Q2.0
	1	0	1	0	0	1	1	1	0	0	0	1	0	0	0	1	%Q2.1
	2	0	0	1	1	0	1	0	1	0	0	0	0	0	0	0	%M23
	3	0	0	0	1	0	0	0	0	0	0	1	0	0	1	1	%M32
	4	1	0	0	0	1	1	1	0	0	0	1	1	0	1	0	%Q2.9
	5	1	0	1	0	0	0	1	1	0	0	1	0	0	0	0	%Q2.10
	6	1	0	0	0	0	1	0	1	0	0	0	0	0	1	1	%Q2.11
	7	0	1	0	1	0	0	0	0	0	1	0	0	0	0	1	%Q2.12
	8	1	0	0	0	1	0	0	0	0	1	0	1	0	1	0	%M8
	9	0	1	0	0	0	1	1	0	0	1	0	0	0	0	1	%M9
	A	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	%M1
	B	1	0	0	1	1	0	0	0	1	0	0	0	0	1	1	%Q2.6
	C	1	0	1	0	1	0	1	0	1	1	1	0	0	0	0	%Q2.7
	D	0	1	0	0	1	0	1	1	1	0	0	1	0	0	1	%Q2.8
	E	1	1	0	0	0	0	0	0	0	0	1	0	0	1	0	%M100
	F	0	0	1	0	0	0	0	1	0	1	0	1	1	1	0	%M13

In the above example, for step 1, control bits %Q2.1;%Q3.5; %Q2.8;%Q3.6;%M5 and %M6 are set to state 1. The other control bits are set to 0.

The current step number is incremented on each rising edge at input U (or activation of instruction U). This number can be modified by the program.

Operating diagram



Programming and configuration

In this example, the first 5 outputs %Q2.0 to %Q2.4 are activated in succession each time input %I1.1 is set to 1.

Input I1.0 resets the outputs to step 0.

Configuration

The following information is defined in the variables editor :

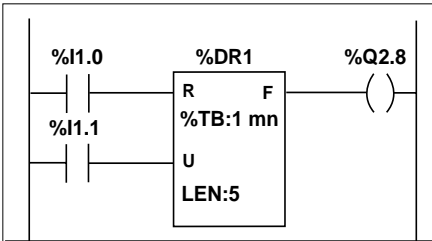
- Step number : 5 (LEN:5).
- The state of the outputs (control bits) for each step of the drum controller.

	Step	Assignment of control bits
	0 1 2 3 4	
	0 : 1 0 0 0 0	%Q2.0
	1 : 0 1 0 0 0	%Q2.1
Bit	2 : 0 0 1 0 0	%Q2.2
	3 : 0 0 0 1 0	%Q2.3
	4 : 0 0 0 0 1	%Q2.4

- Time base (TB:1 min).

Programming

Ladder language



Instruction list language

```
LD    %I1.0
R     %DR1
LD    %I1.1
U     %DR1
LD    %DR1.F
ST    %Q2.8
```

Structured text language

```
IF %I1.0 THEN
  RESET %DR1 ;
END_IF ;
IF RE %I1.1 THEN
  UP %DR1 ;
END_IF ;
%Q2.8 := %DR1.F ;
```

In Structured text language, 2 instructions are used to program the drum controller function blocks :

- RESET %DRi : Initializes the controller to step 0,
- UP %DRi : Advances the controller by one step and updates the control bits. This instruction creates a rising edge on input U of the function block : this conditioning instruction must therefore be an edge-type.

Note

When resetting to 0 (input R, instruction R or RESET instruction)

- In Ladder language, the memorized value of input U is updated with the actual values.
- In Instruction list language, the memorized value of input U is not updated. It retains the value it had before being called.
- In Structured text language, the memorized value of input U is updated with 0.

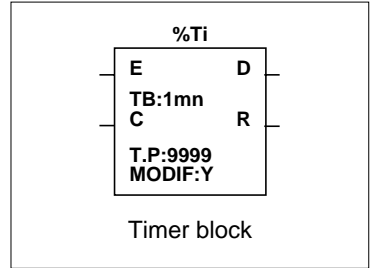
Special cases

- **Effect of a cold restart** : (%S0=1) resets the drum controller to step 0 (with updating of the control bits).
- **Effect of a warm restart** : (%S1=1) updates the control bits, according to the current step.
- **Effect of a program jump, de-activation of the task and break point** : the fact of not scanning the drum controller means that the control bits are not reset to 0.
- **Updating the control bits** : only occurs when there is a change of step or in the case of a cold or warm restart.

2.2-4 Timer function block %Ti (Series 7)

This timer function block, which is compatible with Series 7 PL7-2/3 blocks is used to provide control for time-delayed actions.

The value of this delay is programmable and can be modified via the terminal.



Characteristics

Number	%Ti	0 to 63 for a TSX 37, 0 to 254 for a TSX 57
Time base	TB	1min, 1s, 100ms, 10ms (default 1min).
Current value	%Ti.V	Word which decreases from %Ti.P to 0 when the timer is running. Can be read and tested but not written.
Preset value	%Ti.P	$0 < \%Ti.P \leq 9999$. Word which can be read, tested and written. It is set to value 9999 by default. The duration is equal to %Ti.P*TB.
Adjust via terminal MODIF	Y/N	Y : the preset value can be modified in adjust mode. N : no access in adjustmode.
Setting input	E(Enable)	At state 0, resets the timer %Ti.V = %Ti.P.
Control input	C(Control)	At state 0, freezes the current value %Ti.V.
Timer output done	D(Done)	Associated bit %Ti.D = 1, if timer done %Ti.V = 0
Timer running output	R(Running)	Associated bit %Ti.R = 1 if timer %Ti.P > %Ti.V > 0 and if input C is at 1.

Note :

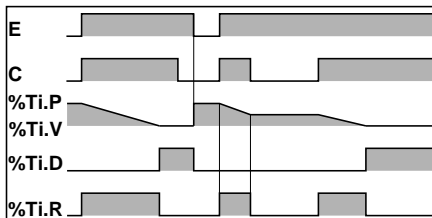
%Ti function blocks cannot be programmed in Instruction list language. The objects of %Ti blocks (%Ti.V, %Ti.P, %Ti.D and %Ti.R) can, however, be accessed.

The total number of %Tmi + %Ti should be less than 64 on the TSX 37 and less than 255 on the TSX 57.

Operation

The timer changes when its two inputs (E and C) are at 1. It behaves like a downcounter.

- Current value %Ti.V decreases from the preset %Ti.P to 0, by one unit on each pulse of the time base TB.
- Output bit %Ti.R (Timer running) assigned to output R is then at state 1 and output bit %Ti.D (Timer done) assigned to output D is at state 0.
- When current value %Ti.V = 0, %Ti.D changes to 1 and %Ti.R returns to 0.

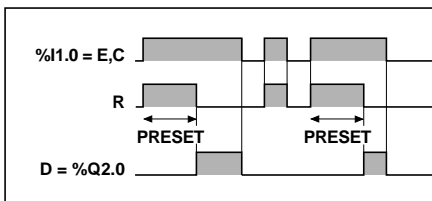
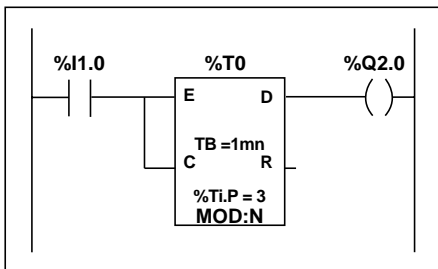


E	0	0	1	1
C	0	1	0	1
%Ti.P	%Ti.V = %Ti.P	%Ti.V = %Ti.P	%Ti.V frozen	%Ti.V decr. from %Ti.P -> 0
%Ti.V				
%Ti.D	0	0	0	1 if Timer done
%Ti.R	0	0	0	1 if Timer running

Standard operations

The Timer function block can be programmed to perform the following functions :

- **On-time delay**
Ladder language



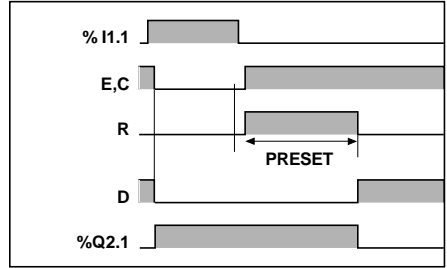
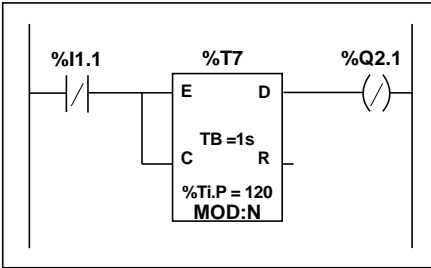
Structured text language

```

IF %I1.0 THEN
  START %T0 ;
ELSE
  PRESET %T0 ;
END_IF ;
%Q2.0 := %T0.D ;
    
```

- **Off-time delay**

Ladder language

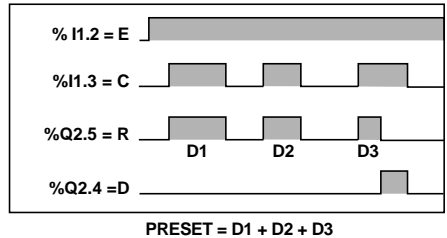
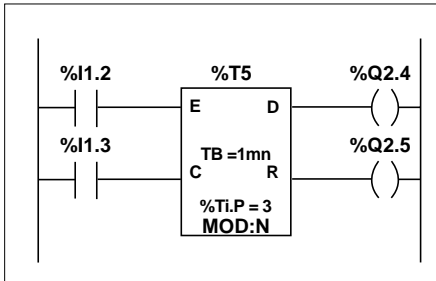


Structured text language

```
IF %I1.1 THEN
  PRESET %T7 ;
ELSE
  START %T7 ;
END_IF ;
%Q2.1 := NOT %T7.D ;
```

- **Cumulated on-time delay**

Ladder language

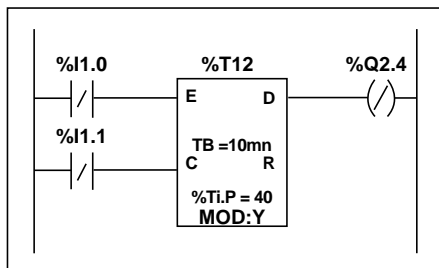


Structured text language

```
IF %I1.2 THEN
  IF %I1.3 THEN
    START %T5 ;
  ELSE
    STOP %T5 ;
  END_IF ;
ELSE
  PRESET %T5 ;
END_IF ;
%Q2.4 := %T5.D ;
%Q2.5 := %T5.R ;
```

• Cumulated off-time delay

Ladder language

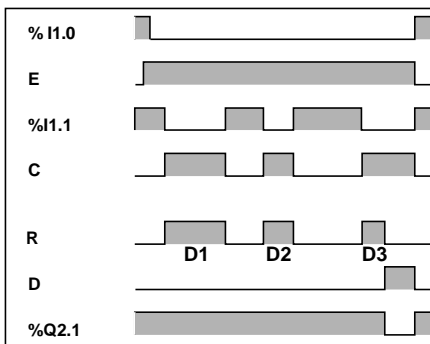


Structured text language

```

IF %I1.0 THEN
    PRESET %T12 ;
ELSE
    IF %I1.1 THEN
        STOP %T12 ;
    ELSE
        START %T12 ;
    END_IF ;
END_IF ;
%Q2.4 := NOT %T12.D ;

```



PRESET = D1 + D2 + D3

In Structured text language, 3 instructions are used to program the timer function blocks %Ti :

- PRESET %Ti : Resets the timer,
- START %Ti : Starts the timer running,
- STOP %Ti : Freezes the current value of the timer.

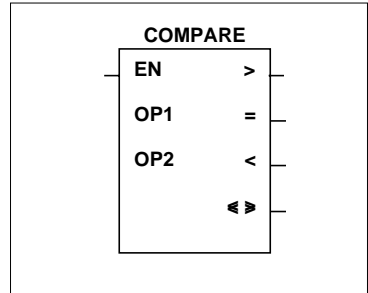
Special cases

- **Effect of a cold restart** : (%S0 = 1) the preset value (defined by the variables editor) is loaded into the current value and output %Ti.D is set to 0, since the preset value which may have been modified by the terminal is lost.
- **Effect of a warm restart** : (%S1) has no effect on the current value of the timer.
- **Effect of a PLC stop** : a PLC stop, de-activation of the current task or execution of a break point does not freeze the current value.
- **Effect of a program jump** : the fact of not scanning the rung where the timer block is programmed does not freeze current value %Ti.V which continues to decrease to 0. Similarly, bits %Ti.D and %Ti.R assigned to timer block outputs D and R continue to operate normally and can thus be tested in another rung. However, the coils directly "connected" to the block outputs will not be activated since they are not scanned by the PLC.
- **Testing bits %Ti.D and %Ti.R** : the state of these bits can change during a scan.

2.3 Vertical comparison blocks

Vertical comparison blocks are used to compare two operands (OP).

These two operands are either 16-bit words (possibly indexed) or immediate values.



The number of vertical comparison blocks is neither limited nor numbered.

Characteristics

Command input	EN	At state 1, compares the two operands.
Greater than output	>	Is at state 1 if the contents of OP1 are greater than those of OP2.
Equal to output	=	Is at state 1 if the contents of OP1 are equal to those of OP2.
Less than output	<	Is at state 1 if the contents of OP1 are less than those of OP2.
Different from output	<>	Is at state 1 if the contents of OP1 are different from those of OP2.
Operand no.1	OP1	This operand is a single length word object (it can be indexed).
Operand no.2	OP2	This operand is a single length word object (it can be indexed).

Operation

When the command input is set to 1, the two operands are compared and the four outputs are activated according to the result of the comparison. Setting the command input to 0 resets the activated outputs.

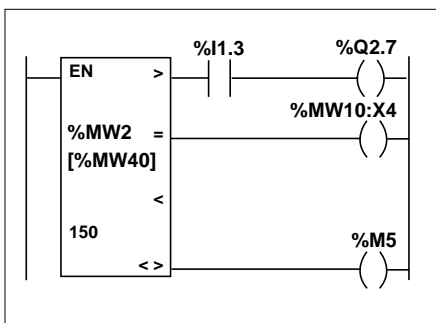
• Example of use

The program below shows the comparison of word %MW2 indexed by word %MW40 with the immediate value 150.

If the contents of %MW2[%MW40] are greater than 150 and %I1.3 = 1, coil %Q2.7 is activated.

If the contents are equal to 150, coil %MW10:X4 is activated. Coil %M5 is only controlled if the contents are different from 150 (< or >).

Ladder language



This function block does not exist in Instruction list language or Structured text language. Use comparison operations >, <, =, <>

Special cases

- **Effect of a cold restart** : (%S0) operand OP1 and possibly OP2 (if OP2 is an internal word) are reset and the outputs are activated according to the result of their comparison with the new values.
- **Effect of a warm restart** : (%S1) has no effect on the comparison block.

2.4 Shift instructions

Shift instructions consist of moving bits of a word or double word operand a certain number of positions to the right or to the left.

- **Logic shift :**

- **SHL**(op2,i) logic shift of i positions to the left.
- **SHR**(op2,i) logic shift of i positions to the right.

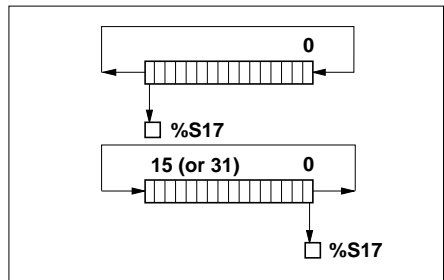
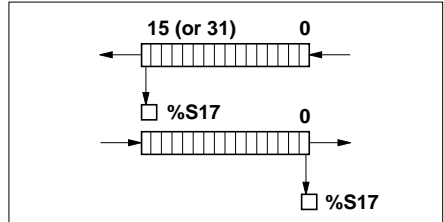
- **Rotate shift :**

- **ROL**(op2,i) rotate shift of i positions to the left.
- **ROR**(op2,i) rotate shift of i positions to the right.

If the operand to be shifted is a single length operand, the variable i will be between 1 and 16.

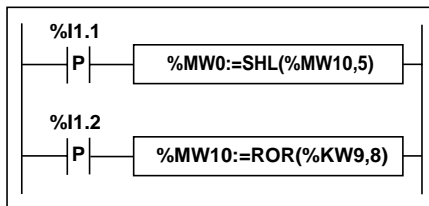
If the operand to be shifted is a double length operand, the variable i will be between 1 and 32.

The state of the last output bit is stored in bit %S17.



Structure

Ladder language



Instruction list language

```
LDR    %I1.1
[ %MW0 := SHL(%MW10,5) ]
```

Structured text language

```
IF RE %I1.2 THEN
    %MW10 := ROR (%KW9,8) ;
END_IF ;
```

Syntax

Operators **SHL,SHR,ROL,ROR**

Op1:=Operator(Op2,i)

Operands

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

2.5 Floating point instructions

2.5-1 General

PL7 Micro software is used to perform operations on floating point objects.

The floating point object format used is that of standard IEEE STD 734-1985 (equivalent to IEC 559). Words are 32 bits long, which corresponds to single length floating point numbers.

Examples of floating point values : 1285.28 1.28528E3

Floating point values are between $-3.402824E+38$ and $-1.175494E-38$, and $1.175494E-38$ and $3.402824E+38$.

Representation is accurate to 2^{-24} . When viewing floating point numbers, no more than six digits can be displayed after the comma.

Notes

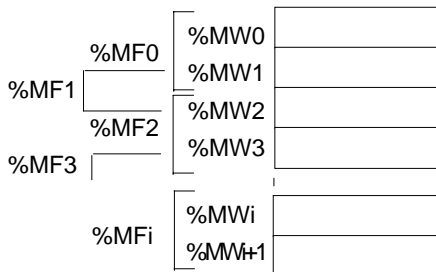
- The value "1285" is interpreted as an integer value. To be considered as a floating point value it must be written : "1285.0",
- The conversion instructions Integer <--> Floating Point are used to change from one format to the other.

Addressing floating point objects

Abbreviations	Full addressing	Type of floating point	Access	Indexed form
Immed. val.	-	immediate values	R	-
%MF	%MFi	internal floating point val.	R/W	%MFi[%MWj]
%KF	%KFi	floating point constant	R	%KFi[%MWj]

Possibility of overlap between objects :

The single and double length and floating point words are stored within the data area in a single memory zone. Thus, floating point word %MFi corresponds to single length words %MWi and %MWi+1 (word %MWi containing the least significant bits and word %MWi+1 containing the most significant bits of word %MFi).



Examples :

%MF0 corresponds to %MW0 and %MW1

%KF543 corresponds to %KW543 and %KW544.

2.5-2 Floating point comparison instructions

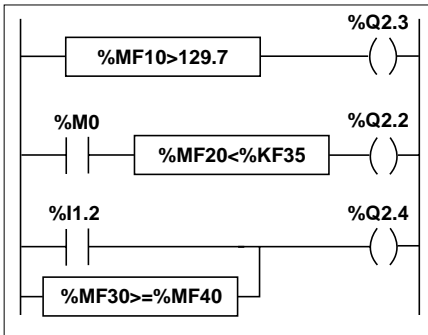
Comparison instructions are used to compare two operands.

- > : test if operand 1 is greater than operand 2.
- >= : test if operand 1 is greater than or equal to operand 2.
- < : test if operand 1 is less than operand 2.
- <= : test if operand 1 is less than or equal to operand 2.
- = : test if operand 1 is equal to operand 2.
- <> : test if operand 1 is different from operand 2.

The result is 1 when the comparison requested is true.

Structure

Ladder language



Instruction list language

```
LD      [%MF10 > 129.7]
ST      %Q2.3
LD      %M0
AND     [%MF20 < %KF35]
ST      %Q2.2
LD      %I1.2
OR      [%MF30 >= %MF40]
ST      %Q2.4
```

The comparison is executed inside square brackets following instructions LD, AND and OR.

The comparison blocks are programmed in the test zone.

Structured text language

```
%Q2.3 := %MF10 > 129.7 ;
%Q2.2 := (%MF20 < %KF35) AND %M0 ;
%Q2.4 := (%MF30 >= %MF40) OR %I1.2 ;
```

Syntax

Operators >, >=, <, <=, =, <>

Op1 Operator Op2

Operands

Type	Operands 1 and 2 (Op1 and Op2)
Indexable floating point objects	%MF, %KF
Non-indexable floating point objects	Floating point immediate value. Floating point numeric expression.

Note

In Instruction language, comparison instructions can be used in parentheses.

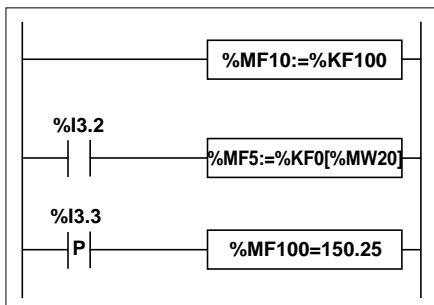
2.5-3 Floating point assignment instructions

The following floating point assignment operations can be performed :

- floating point (indexed) -> floating point (indexed) example 1
- floating point immediate value -> floating point (indexed) example 2

Structure

Ladder language



Instruction list language

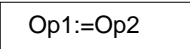
LD TRUE		
[%MF10 := %KF100]		example 1
LD %I3.2		
[%MF5:=%KF0[%MW20]]		example 1
LDR %I3.3		
[%MF100:=150.25]		example 2

Structured text language

%MF10 := %KF100 ;	example 1
IF %I3.2 THEN	
%MF5 := %KF0 [%MW20] ;	example 1
END_IF ;	
IF RE %I3.3 THEN	
%MF100 := 150.25 ;	example 2
END_IF ;	

Syntax

Operator :=



Operands

Type	Operand 1 (Op1)	Operand 2 (Op2)
Indexable floating points		%MF,%KF
Non-indexable floating points		Floating point immediate value. Floating point numeric expr.

It is possible to execute multiple assignments.

Example : %MF0 := %MF2 := %MF4

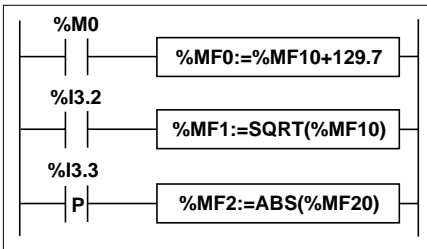
2.5-4 Floating point arithmetic instructions

These instructions are used to perform arithmetic operations between two operands or on one operand.

- | | | | |
|---|-------------------------|-------------|--------------------------------|
| + | : add two operands | SQRT | : square root of an operand |
| - | : subtract two operands | ABS | : absolute value of an operand |
| * | : multiply two operands | / | : divide two operands |

Structure

Ladder language



Instruction list language

```
LD      %M0
[%MF0 := %MF10 + 129.7]

LD      %I3.2
[%MF1 := SQRT(%MF10)]

LDR     %I3.3
[%MF2 := ABS(%MF20)]
```

Structured text language

```
IF %M0 THEN
  %MF0 := %MF10 + 129.7 ;
END_IF ;
IF %I3.2 THEN
  %MF1 := SQRT (%MF10) ;
END_IF ;
IF RE %I3.3 THEN
  %MF2 := ABS (%MF20) ;
END_IF ;
```

Syntax

Operators

- +, -, *, /

Op1:=Op2 Operator Op3

- **SQRT, ABS**

Op1:=Operator(Op2)

Operands

Type	Operand 1 (Op1)	Operands 2 and 3 (Op2 and 3)
Indexable words	%MF	%MF,%KF
Non-indexable words		Floating point immediate value. Floating point numeric expression.

Rules for use

- Operations on floating points and on integers cannot be directly mixed. Conversion operations convert to one or other of these formats (see section 2.6, part B).
- System bit %S18 is controlled in the same way as for integer operations. It is set to 1 if :
 - Capacity is exceeded during an operation.
 - Division by 0 occurs.
 - The square root of a negative value is extracted (see section 1.4-4, part B).

2.6 Numeric conversion instructions

2.6-1 BCD <--> Binary conversion instructions

There are six types of conversion instruction :

- **BCD_TO_INT** : 16-bit BCD number --> 16-bit integer conversion.
- **INT_TO_BCD** : 16-bit integer --> 16-bit BCD number conversion.
- **DBCD_TO_DINT** : 32-bit BCD number --> 32-bit integer conversion.
- **DINT_TO_DBCD** : 32-bit integer --> 32-bit BCD number conversion.
- **DBCD_TO_INT** : 32-bit BCD number --> 16-bit integer conversion.
- **INT_TO_DBCD** : 16-bit integer --> 32-bit BCD number conversion.

Review of the BCD code :

The BCD (Binary Coded Decimal) code represents a decimal digit (0 to 9) by coding 4 bits. A 16-bit word object can thus contain a number expressed in 4 digits (0 ≤ N ≤ 9999).

Decimal	0	1	2	3	4	5	6	7	8	9
BCD	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001

Example :

- Word %MW5 expresses the BCD value "2450" which corresponds to the binary value : 0010 0100 0101 0000,
- Word %MW12 expresses the decimal value "2450" which corresponds to the binary value : 0000 1001 1001 0010.

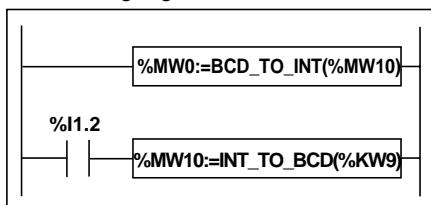
Word %MW5 is converted to word %MW12 by using instruction BCD_TO_INT.

Word %MW12 is converted to word %MW5 by using instruction INT_TO_BCD.

Structure

Conversion operations are performed as follows :

Ladder language



Instruction list language

```
LD      TRUE
[%MW0 := BCD_TO_INT(%MW10)]

LD      %I1.2
[%MW10 := INT_TO_BCD(%KW9)]
```

Structured text language

```
%MW0 := BCD_TO_INT (%MW10) ;
IF %I1.2 THEN
    %MW10 := INT_TO_BCD (%KW9) ;
END_IF ;
```

Syntax

Operators (conversion of a 16-bit number)

- **BCD_TO_INT** Op1:=Operator(Op2)
- **INT_TO_BCD**
- **INT_TO_DBCD**

Operands

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

Syntax

Operators (conversion of a 32-bit number)

- **DBCD_TO_DINT** Op1:=Operator(Op2)
- **DINT_TO_DBCD**
- **DBCD_TO_INT**

Operands

Type	Operand 1 (Op1)	Operand 2 (Op2)
Indexable words	%MW	
Non-indexable words	%QW,%SW,%NW, %BLK	
Indexable double words	%MD	%MD, %KD
Non-indexable double words	%QD, %SD	Immed. val.,%ID,%QD,%SD Numeric expr.

Application examples

The BCD_TO_INT instruction is used to process a setpoint value at PLC inputs via BCD encoded thumbwheels.

The INT_TO_BCD instruction is used to display numeric values (for example, the result of a calculation or the current value of a function block) on BCD coded displays.

Rules for use

- BCD-->Binary conversion

The BCD-->Binary conversion instructions ensure that the conversion operator is applied to a BCD coded value. If the value is not a BCD coded value, system bit %S18 is set to 1 and the result gives the value of the first faulty 4-bit byte.

Eg : BCD_TO_INT(%MW2) where %MW2=4660 gives a result of 1234.

However, %MW2=242 (16#00F2) sets %S18 to 1 and the result is 15.

For the DBCD_TO_INT instruction, if the BCD number is greater than 32767, system bit %S18 is set to 1 and value -1 is loaded in the result.

- Binary--> BCD conversion

The INT_TO_BCD instruction ensures that the conversion operator is applied to a value between 0 and 9999 (or 0 and 9999 9999). If this is not the case, system bit %S18 is set to 1 and the result gives the value of the input parameter.

Eg : INT_TO_BCD(%MW2) where %MW2=2478 gives a result of 9336.

However, %MW2=10004 sets %S18 to 1 and the result is 10004.

For the INT_TO_DBCD instruction, if the input parameter is negative, system bit %S18 is set to 1 and the result gives the value of the input parameter.

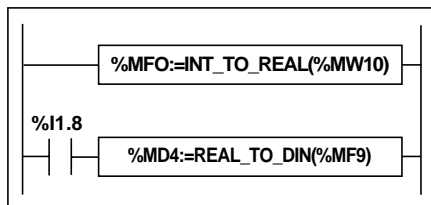
2.6-2 Integer <--> Floating point conversion instructions

There are four conversion instructions :

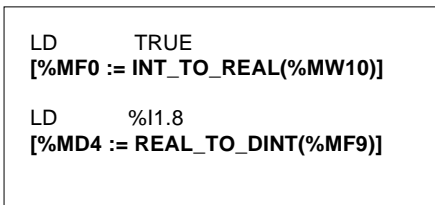
- **INT_TO_REAL** : integer word --> floating point conversion.
- **DINT_TO_REAL** : integer double word --> floating point word conversion.
- **REAL_TO_INT** : floating point word --> integer word conversion (the result is the nearest algebraic value).
- **REAL_TO_DINT** : floating point word --> integer double word conversion (the result is the nearest algebraic value).

Structure

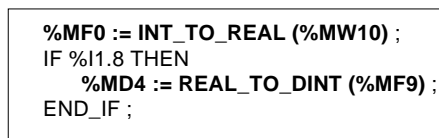
Ladder language



Instruction list language



Structured text language



Syntax

Operator

Op1:=INT_TO_REAL(Op2)

Operands

Type	Operand 1 (Op1)	Operand 2 (Op2)
Indexable words		%MW,%KW
Non-indexable words		Immed. val.,%IW,%QW,%SW %NW,%BLK,%Xi.T,Num. expr.
Indexable floating point words	%MF	

Example : integer word --> floating point word conversion : 147 --> 1.47e+02

Operator

Op1:=DINT_TO_REAL(Op2)

Operands

Type	Operand 1 (Op1)	Operand 2 (Op2)
Indexable words		%MD,%KD
Non-indexable words		Val.imm.,%ID,%QD,%SD Numerical expr.
Indexable floating point words	%MF	

Example : integer double word --> floating point word conversion : 68905 000 --> 6.8905e+07

Operator

Op1:=REAL_TO_INT(Op2)
Op1:=REAL_TO_DINT(Op2)

Operands

Type	Operand 1 (Op1)	Operand 2 (Op2)
Indexable words	%MW	
Non-indexable words	%QW,%NW,%BLK	
Indexable double words	%MD	
Non-indexable double words	%QD	
Indexable floating point words		%MF,%KF
Non-indexable floating point words		Floating point immed. value

Example : floating point word --> integer word conversion 5978.6 --> 5978
floating point word --> integer double word conversion -1235978.6-->-1235979**Note** : If, during conversion of a real word to an integer (or a real word to a whole double word), the floating point value exceeds the limits of the word (or double word), bit %S18 is set to 1.

2.6-3 Gray --> Integer conversion instructions

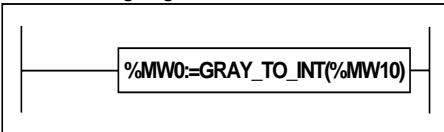
The instruction **GRAY_TO_INT** converts a word in Gray code to an integer (pure binary code).

Review of Gray code : Gray or "reflected binary" code is used to code a changing numeric value into a series of binary configurations which are distinguished from each other by the change of state of a single bit. This code is used, for example, to avoid the following random conditions : in pure binary code, the value 0111 changing to 1000 may generate random values between 0 and 1000 since the bits do not change value at exactly the same time.

Decimal	0	1	2	3	4	5	6	7	8	9
Binary	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001
Gray	0000	0001	0011	0010	0110	0111	0101	0100	1100	1101

Structure

Ladder language



Instruction list language

```

LD      TRUE
[%MW0 := GRAY_TO_INT(%MW10)]
  
```

Structured text language

```
%MW0 := GRAY_TO_INT (%MW10) ;
```

Syntax

Operator

```
Op1:=GRAY_TO_INT(Op2)
```

Operands

Type	Operand 1 (Op1)	Operand 2 (Op2)
Indexable words	%MW	%MW,%KW
Non-indexable words	%QW,%SW,%NW,%BLK	Immed.val.,%IW,%QW,%SW %NW,%BLK,%Xi.T,Num. expr.

2.7 Word table instructions

2.7-1 General

PL7 software is used to perform operations on tables of :

- words,
- double words,
- floating point words.

Word tables are sequences of adjacent words of the same type and of a defined length, L.

%KW10	16 bits
%KW14	

Example of word tables : %KW10:5

Type	Format	Maximum address	Size	Write access
Internal words	Single length	%MWi:L	$i+L \leq N_{max} (1)$	Yes
	Double length	%MDi:L	$i+L \leq N_{max}-1 (1)$	Yes
	Floating point	%MFi:L	$i+L \leq N_{max}-1 (1)$	Yes
Constant words	Single length	%KWi:L	$i+L \leq N_{max} (1)$	No
	Double length	%KDi:L	$i+L \leq N_{max}-1 (1)$	No
	Floating point	%KFi:L	$i+L \leq N_{max}-1 (1)$	No
System words	Single length	%SW50:4 (2)	-	Yes

(1) N_{max} = maximum number of words defined during software configuration.

(2) Only words %SW50 to %SW53 can be addressed in table form.

General rules for operations on tables

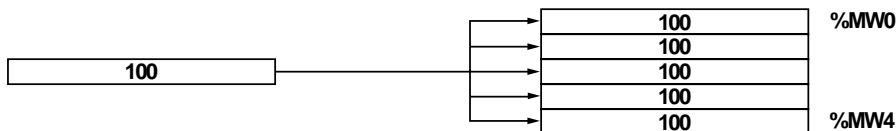
- Table operations are only performed on tables containing objects of the same type.
- Table operations can only be performed on a maximum of two tables.
- If the tables in an operation are of different sizes, the result table will correspond to the smaller of the two tables.
- The user should avoid performing operations on tables which overlap (for example : %MW100[20]:= %MW90[20]+%KW100[20]).
- Operations on two tables are performed on each element of the same position of the two tables and the result is transferred to the element in the same position of the result table.
- If during an operation between two elements, system bit %S18 is set to 1, the result for this operation is then incorrect, but operations on the following elements will be performed correctly.
- If one of the operands is a numeric expression it must be placed in parentheses.
- The position of the first word in the table corresponds to position 0.

2.7-2 Word table assignment

The following can be assigned to word tables :

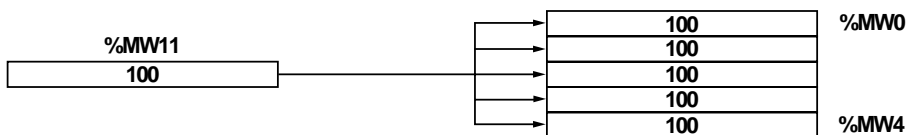
- immediate value -> word table (indexed) example 1
- double format immediate value -> double word table (indexed)
- floating point immediate value -> floating point table (indexed)

Example 1 : %MW0 :5:= 100



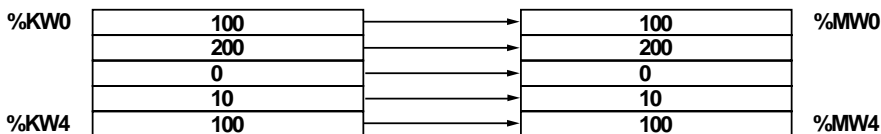
- word (indexed) -> word table (indexed) example 2
- double word (indexed) -> double word table (indexed)
- floating point (indexed) -> floating point table (indexed)

Example 2 : %MW0 :5:= %MW11



- word table (indexed) -> word table (indexed) example 3
- double word table (indexed) -> double word table (indexed)
- floating point table (indexed) -> floating point table (indexed)

Example 3 : %MW0 :5:= %KW0:5

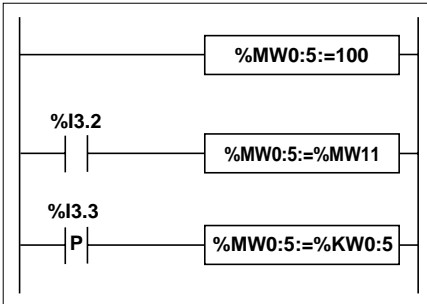


Note :

Multiple assignments are authorized in examples 1 and 2 (%MW0:4 := %MW10:6 := %MW100) but not in example 3.

Structure

Ladder language



Instruction list language

```
LD TRUE
[ %MW0 :5:= 100]      example 1

LD %I3.2
[ %MW0:5 := %MW11]   example 2
```

Structured text language

```
IF RE %I3.3 THEN
    %MW0:5 := %KW0:5 ; example 3
END_IF ;
```

Syntax

Operator :=

Op1 := Op2

Word tables

Type	Operand 1 (Op1)	Operand 2 (Op2)
Tables of indexable words	%MW:L	%MW:L,%KW:L
Indexable words		%MW,%KW
Non-indexable words		Immed.val.,%IW,%QW,%SW %NW,%BLK,%Xi.T,Num. expr.

Double word tables

Type	Operand 1 (Op1)	Operand 2 (Op2)
Tables of indexable words	%MD:L	%MD:L,%KD:L
Indexable double words		%MD,%KD
Non-indexable double words		Immed.val.,%ID,%QD, Numeric expr.

Floating point word tables

Type	Operand 1 (Op1)	Operand 2 (Op2)
Tables of floating point words	%MF:L	%MF:L,%KF:L
Indexable floating point words		%MF,%KF
Non-indexable floating point words		Floating point immed.val. Floating point numeric expr.

Note :

Multiple assignments on tables are prohibited.

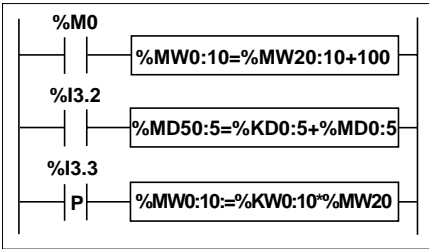
2.7-3 Arithmetic instructions on tables

These instructions are used to perform an arithmetic operation between two word table (or word and word table) type operands.

- + : add
- : subtract
- REM : remainder of division
- * : multiply
- / : divide

Structure

Ladder language



Instruction list language

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

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

Structured text language

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

Syntax

Operators

- +, -, *, /, REM



Operands

Word tables

Type	Operand 1 (Op1)	Operands 2 & 3 (Op2 and 3)
Tables of indexable words	%MW:L	%MW:L,%KW:L
Indexable words		%MW,%KW
Non-indexable words		Immed.val.,%IW,%QW,%SW %NW,%BLK,%Xi.T,Num. expr.

Double word tables

Type	Operand 1 (Op1)	Operands 2 & 3 (Op2 and 3)
Tables of indexable words	%MD:L	%MD:L,%KD:L
Indexable double words		%MD,%KD
Non-indexable double words		Immed.val.,%ID,%QD, Numeric expr.

2.7-4 Logic instructions on tables

The associated instructions are used to perform a logic operation between two word table (or word and word table) type operands.

AND : AND (bit-wise).

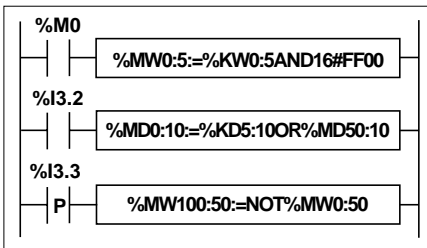
OR : Logic OR (bit-wise).

XOR : Exclusive OR (bit-wise).

NOT : Logic complement (bit-wise) of a table (only one operand).

Structure

Ladder language



Instruction list language

```
LD      %M0
[%MW0:5:=%KW0: 5 AND 16#FF00]
```

Structured text language

```
IF %I3.2 THEN
  %MD0:10 := %KD5:10 OR %MD50:10 ;
END_IF ;
IF RE%I3.3 THEN
  %MW100:50 := NOT %MW0:50 ;
END_IF ;
```

Syntax

Operators

AND, OR, XOR

```
Op1:=Op2 Operator Op3
```

NOT

```
Op1:=NOT Op2
```

Operands

Word tables

Type	Operand 1 (Op1)	Operands 2 & 3 (Op2 and 3)
Tables of indexable words	%MW:L	%MW:L,%KW:L
Indexable words		%MW,%KW
Non-indexable words		Immed.val.,%IW,%QW,%SW %NW,%BLK,%XiT,Num. expr.

Double word tables

Type	Operand 1 (Op1)	Operands 2 & 3 (Op2 and 3)
Tables of indexable words	%MD:L	%MD:L,%KD:L
Indexable double words		%MD,%KD,%SD
Non-indexable double words		Immed.val.,%ID,%QD, Numeric expr.

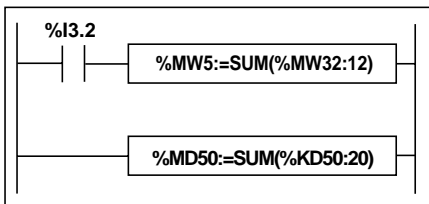
2.7-5 Summing function on tables

The **SUM** function adds together all the elements in a word table :

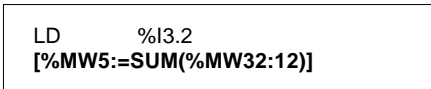
- If the table comprises single format words, the result is given in the form of a single format word.
- If the table comprises double words, the result is given in the form of a double word.

Structure

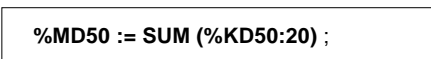
Ladder language



Instruction list language



Structured text language



Syntax

Function

```
Res:=SUM(Tab)
```

Parameters

Word tables

Type	Result (Res)	Table (Tab)
Tables of indexable words		%MW:L,%KW:L
Indexable words	%MW	
Non-indexable words	%QW,%SW,%NW	

Double word tables

Type	Result (Res)	Table (Tab)
Tables of indexable words		%MD:L,%KD:L
Indexable double words	%MD	
Non-indexable double words	%QD,%SD	

Note :

Bit %S18 is set to 1 when the result exceeds the limits of the word or double word format depending on the table operand.

Example %MW5:=SUM(%MW30:4)

%MW30= 10

%MW31= 20

%MW32= 30

%MW33= 40

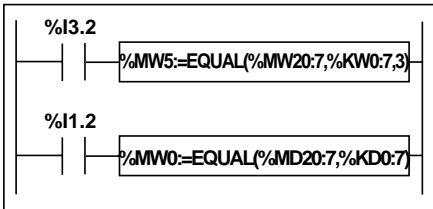
%MW5=10+20+30+40=100

2.7-6 Table comparison function

The **EQUAL** function compares two tables element by element. If a difference is detected, the position of the first elements which are not the same is given in the form of a word. Otherwise, the value given is equal to -1. The third parameter provides the position from which the comparison begins (example : 0 to start at the beginning). This third parameter is optional. If it is omitted, the comparison is performed on all of the table.

Structure

Ladder language



Instruction list language

```
LD      %I3.2
[%MW5:=EQUAL(%MW20:7,%KW0:7,3)]
```

Structured text language

```
IF %I1.2 THEN
  %MW0 := EQUAL (%MD20:7,%KD0:7) ;
END_IF ;
```

Syntax

Function

```
Res:=EQUAL(Tab1,Tab2,position)
```

Parameters

Word tables

Type	Result (Res)	Table (Tab)	Position
Tables of indexable words		%MW:L,%KW:L	
Indexable words	%MW		%MW,%KW
Non-indexable words	%QW,%SW, %NW		Immed. val.%QW, %IW,%SW,%NW %Xi.T,Num. expr.

Double word tables

Type	Result (Res)	Table (Tab)	Position
Tables of indexable words		%MD:L,%KD:L	
Indexable double words	%MD		%MD,%KD
Non-indexable double words	%QD,%SD		Immed. val.%QD, %ID,%SD Numeric expr.

Note :

- The tables must be the same length.
- If the position parameter is greater than the size of the tables, the result is then equal to this position.

Example %MW5:=EQUAL(%MW30:4,%KW0:4,1)

0 %MW30= 10 %KW0= 20

1 %MW31= 20 %KW1= 20

2 %MW32= 30 %KW2= 30

3 %MW33= 40 %KW3= 60 ==> %MW33 ≠ %KW3==> %MW5= 3

2.7-7 Find functions on tables

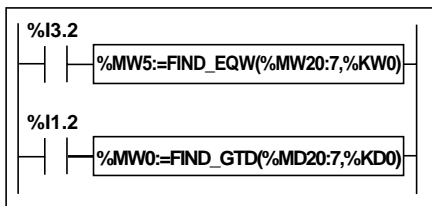
There are six find functions :

- **FIND_EQW** : find the position in a word table of the first element equal to a given value.
- **FIND_GTW** : find the position in a word table of the first element greater than a given value.
- **FIND_LTW** : find the position in a word table of the first element less than a given value.
- **FIND_EQD** : find the position in a double word table of the first element equal to a given value.
- **FIND_GTD** : find the position in a double word table of the first element greater than a given value.
- **FIND_LTD** : find the position in a double word table of the first element less than a given value.

The result of these instructions is equal to the position of the first element found or to - 1 if the search is unsuccessful.

Structure

Ladder language



Instruction list language

```
LD      %I3.2
[%MW5:=FIND_EQW(%MW20:7,%KW0)]
```

Structured text language

```
IF %I1.2 THEN
  %MW0:=FIND_GTD(%MD20:7,%KD0) ;
END_IF ;
```

Syntax

Function

FIND_EQW,FIND_GTW,FIND_LTW

```
Res:=Function(Tab,Val)
```

Parameters

Word tables

Type	Result (Res)	Table (Tab)	Value (Val)
Tables of indexable words		%MW:L,%KW:L	
Indexable words	%MW		%MW,%KW
Non-indexable words	%QW,%SW %NW		Immed. val.%QW, %IW,%SW,%NW %Xi.T,Num. expr.

Example %MW5:=FIND_EQW(%MW30:4,%KW0)

Position

- 0 %MW30= 10
- 1 %MW31= 20
- 2 %MW32= 30 ==> %KW0= 30 ==> %MW5= 2
- 3 %MW33= 40

Function

FIND_EQD,FIND_GTD,FIND_LTD

Res:=Function(Tab,Val)

Parameters

Double word tables

Type	Result (Res)	Table (Tab)	Value (Val)
Tables of indexable words		%MD:L,%KD:L	
Indexable (double) words	%MW		%MD,%KD
Non-indexable (double) words	%QW,%SW		Immed. val.%QD, %ID,%SD Numeric expr.

Example %MW5:=FIND_GTD(%MD30:4,%KD0)

Position

0 %MD30= 100000

1 %MD32= 200000 ==> %KD0= 250000 ==> %MW5= 2

2 %MD34= 300000

3 %MD36= 400000

2.7-8 Find maximum and minimum values function on tables

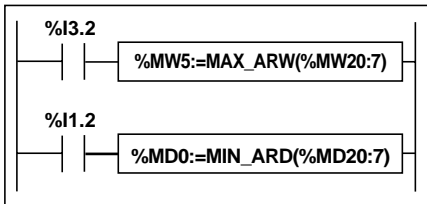
There are four find functions :

- **MAX_ARW** : find the maximum value in a word table.
- **MIN_ARW** : find the minimum value in a word table.
- **MAX_ARD** : find the maximum value in a double word table.
- **MIN_ARD** : find the minimum value in a double word table.

The result of these instructions is equal to the maximum (or minimum) value found in the table.

Structure

Ladder language



Instruction list language

```
LD      %I3.2
[%MW5:=MAX_ARW(%MW20:7)]
```

Structured text language

```
IF %I1.2 THEN
    %MD0 := MIN_ARD (%MD20:7) ;
END_IF ;
```

Syntax

Function

MAX_ARW,MIN_ARW

Parameters

Word tables

Type	Result (Res)	Table (Tab)
Tables of indexable words		%MW:L,%KW:L
Indexable words	%MW	
Non-indexable words	%QW,%SW,%NW	

```
Res:=Function(Tab)
```

Function

MAX_ARD,MIN_ARD

Parameters

Double word tables

Type	Result (Res)	Table (Tab)
Tables of indexable words		%MD:L,%KD:L
Indexable double words	%MD	
Non-indexable double words	%QD,%SD	

```
Res:=Function(Tab)
```

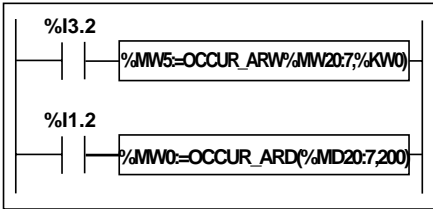
2.7-9 Number of occurrences of a value in a table

There are two find functions :

- **OCCUR_ARW** : searches a word table for the number of elements equal to a given value.
- **OCCUR_ARD** : searches a double word table for the number of elements equal to a given value.

Structure

Ladder language



Instruction list language

```
LD      %I3.2
[ %MW5:=OCCUR_ARW(%MW20:7,%KW0)]
```

Structured text language

```
IF %I1.2 THEN
  %MW0:=OCCUR_ARD(%MD20:7,200) ;
END_IF ;
```

Syntax

Function

```
Res:=OCCUR_ARW(Tab,Val)
```

Parameters

Word tables

Type	Result (Res)	Table (Tab)	Value (Val)
Tables of indexable words		%MW:L,%KW:L	
Indexable words	%MW		%MW,%KW
Non-indexable words	%QW,%SW, %NW		Immed. val.%QW, %IW,%SW,%NW %Xi.T,Num. expr.

Function

```
Res:=OCCUR_ARD(Tab,Val)
```

Parameters

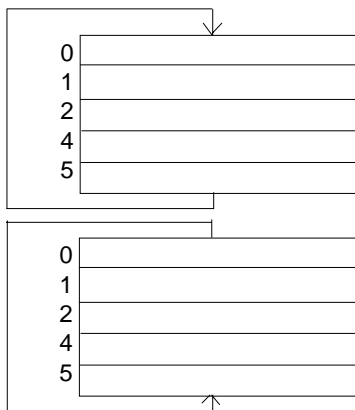
Double word tables

Type	Result (Res)	Table (Tab)	Value (Val)
Tables of indexable words		%MD:L,%KD:L	
Indexable (double) words	%MW		%MD,%KD
Non-indexable (double) words	%QW,%SW		Immed. val.%QD, %ID,%SD Numeric expr.

2.7-10 Rotate shift function on tables

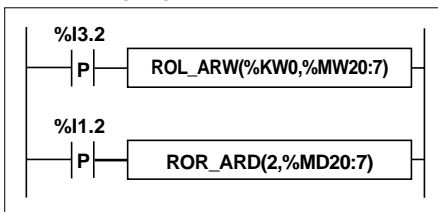
There are four shift functions :

- **ROL_ARW** : performs a rotate shift of elements in word tables by n positions from top to bottom.
- **ROL_ARD** : performs a rotate shift of elements in double word tables by n positions from top to bottom.
- **ROR_ARW** : performs a rotate shift of elements in word tables by n positions from bottom to top.
- **ROR_ARD** : performs a rotate shift of elements in double word tables by n positions from bottom to top.



Structure

Ladder language



Instruction list language

```
LDR    %I3.2
[ROL_ARW(%KW0,%MW20:7)]
```

Structured text language

```
IF RE%I1.2 THEN
    ROR_ARD (2,%MD20:7) ;
END_IF ;
```

Syntax

Functions **ROL_ARW,ROR_ARW**

```
Function(n,Tab)
```

Parameters

Word tables

Type	Number of positions (n)	Table (Tab)
Tables of indexable words		%MW:L
Indexable words	%MW,%KW	
Non-indexable words	Immed. val.%QW,%IW,%SW %NW,%Xi.T,Numeric expr.	

Functions **ROL_ARD,ROR_ARD**

```
Function(n,Tab)
```

Parameters

Double word tables

Type	Number of positions (n)	Table (Tab)
Tables of indexable words		%MD:L
Indexable words	%MW,%KW	
Non-indexable words	Immed. val.%QW,%IW,%SW %NW,%Xi.T,Numeric expr.	

Note : If the value of n is negative or zero, no shift is performed.

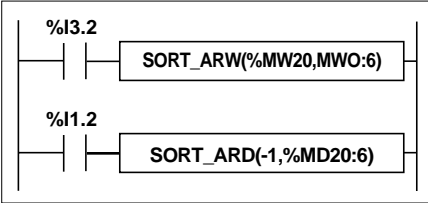
2.7-11 Sort function on tables

There are two sort functions :

- **SORT_ARW** : sorts the elements of the word table into ascending or descending order and stores the result in the same table.
- **SORT_ARD** : sorts the elements of the double word table into ascending or descending order and stores the result in the same table.

Structure

Ladder language



Instruction list language

```
LD      %I3.2
[SORT_ARW(%MW20,%MW0:6)]
```

Structured text language

```
IF %I1.2 THEN
  SORT_ARD (-1,%MD20:6) ;
END_IF ;
```

Syntax

Function

SORT_ARW (dir.,Tab)

- The "direction" parameter determines the order of the sort : if the direction is ≥ 0 , the sort is performed in ascending order. If < 0 , the sort is performed in descending order.
- The result (sorted table) is given in the Tab parameter (table to be sorted).

Parameters

Word tables

Type	Direction of the sort	Table (Tab)
Tables of indexable words		%MW:L
Indexable words	%MW,%KW	
Non-indexable words	Immed. val.%QW,%IW,%SW %NW,Numeric expr.	

Function

SORT_ARD(dir.,Tab)

- The "direction" parameter determines the order of the sort : if the direction is ≥ 0 , the sort is performed in ascending order : if < 0 , the sort is performed in descending order.
- The result (sorted table) is given in the Tab parameter (table to be sorted).

Parameters

Double word tables

Type	Direction of the sort	Table (Tab)
Tables of indexable words		%MD:L
Indexable words	%MW,%KW	
Non-indexable words	Immed. val.%QW,%IW,%SW %NW,Numeric expr.	

2.8 Character string instructions

2.8-1 Format of a string or table of characters

A character table is composed of a series of bytes in which a character string can be stored. The size of the table is used to specify the maximum length of the character string (up to 255 characters).

Example : **%MB4:6 represents a table of 6 bytes containing a string of up to 6 characters.**

The first byte at the beginning of a table must be even (it is not possible to enter a byte table which starts with an odd byte, eg :%MB5:6).

Byte tables use the same memory zone as words %MW and %MD. There is therefore a risk of overlap : see section 1.2-4, part A.

The term character string represents all the characters between the start of the table and the first string termination character encountered.

The NUL character (hexa code 00) is known as the String termination character. It is symbolized by \emptyset throughout this section.

Examples :

- The following table (of 12 elements) contains the character string 'ABCDE' (5 characters long).

'A'	'B'	'C'	'D'	'E'	\emptyset	'J'	'K'	'L'	'M'	'N'	'O'
-----	-----	-----	-----	-----	-------------	-----	-----	-----	-----	-----	-----

- The following table (of 10 elements) contains the character string 'ABCDEJKLMN' (10 characters long).

'A'	'B'	'C'	'D'	'E'	'J'	'K'	'L'	'M'	'N'
-----	-----	-----	-----	-----	-----	-----	-----	-----	-----

The length of a character string is therefore determined either by the number of characters before the string termination character \emptyset , or by the size of the table if no string termination character is detected.

Notes :

System bit %S15 is set to 1 in the following cases :

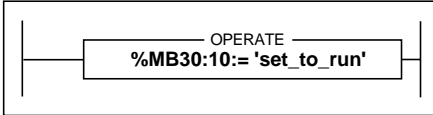
- If, when writing a string in a table, the string is longer than the size of the table. (Impossible to write the string termination character \emptyset).
- If the user attempts to access a character which is not in that string.
- Incorrect parameters :
Length to be deleted zero (DELETE function), length to be extracted zero (MID function), length to be replaced zero (REPLACE function), search for a substring which is longer than the string (FIND function).

2.8-2 Character string assignment

This function is used to transfer a character string to a byte table of length L.

Structure

Ladder language



Instruction list language

```
LD      TRUE
[ %MB30:10 := 'set_to_run']
```

Structured text language

```
%MB30:10:='set_to_run';
```

Example **Transfer of the character string 'set_to_run' to a byte table 10 characters long**

%MB	30	31	32	33	34	35	36	37	38	39
	's'	'e'	't'	'_'	't'	'o'	'_'	'r'	'u'	'n'

Syntax

Operator

```
Op1:=Op2
```

Operands

Type	Operand 1 (Op1)	Operand 2 (Op2)
Byte tables	%MB:L	%MB:L,%KB:L Immediate value

2.8-3 Alphanumeric comparisons

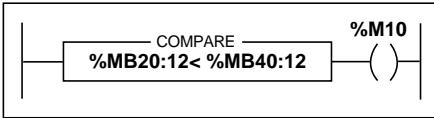
These operators are used to compare two character strings contained in the byte tables provided as parameters. Comparison is performed character by character.

The result is a bit which equals 1 if both strings satisfy the conditions determined by the operator, character by character. Otherwise, the bit equals 0.

The order of characters is determined by the ASCII code table (ISO 646). For example, string 'Z' is larger than string 'AZ' which is larger than the string 'ABC'.

Structure

Ladder language



Instruction list language

```
LD  [%MB20:12 < %MB40:12]
ST  %M10
```

Comparison blocks are programmed in the test zone.

The comparison is executed inside square brackets after the instructions LD, AND and OR.

Structured text language

```
%M10 := %MB20:12 < %MB40:12 ;
```

Example : `%MB20:12 < %MB40:12` ==> YES The result equals 1

where

%MB	20	21	22	23	24	25	26	27	28	29	30	31
	'a'	'b'	'c'	'd'	'e'	'f'	'g'	'i'	∅	'k'	'w'	'z'
%MB	40	41	42	43	44	45	46	47	48	49	50	51
	'a'	'b'	'c'	'd'	'e'	'f'	'h'	'i'	∅	'k'	'w'	'z'

The elements after the termination character are not taken into account.

Syntax

Operator

<, >, <=, >=, =, < >

Op1 Operator Op2

Operands

Type	Operand 1 (Op1) and Operand 2 (Op2)
Byte tables	%MB:L,%KB:L, immediate value

2.8-4 Numeric <--> ASCII conversion functions

These functions are used to convert a numeric (or floating point) value to a character string in ASCII code or vice versa.

The result of the conversion must be transferred to a PL7 object via an assignment operation : byte table, single or double length word, floating point.

The conversions possible are :

INT_TO_STRING	Binary -->ASCII conversion
DINT_TO_STRING	Binary -->ASCII conversion
STRING_TO_INT	ASCII-->Binary conversion
STRING_TO_DINT	ASCII-->Binary conversion
REAL_TO_STRING	Floating point -->ASCII conversion
STRING_TO_REAL	ASCII-->Floating point conversion

Review of floating point format : ==> See section 2.5, part B

Review of the ASCII code :

All 256 alphanumeric and control characters can be coded on 8 bits.

This code, known as ASCII (American Standard Code for Information Interchange), is compatible with the notion of bytes. Any tables of n bytes can therefore be formed by n ASCII codes defining n characters.

2.8-5 Binary -->ASCII conversion

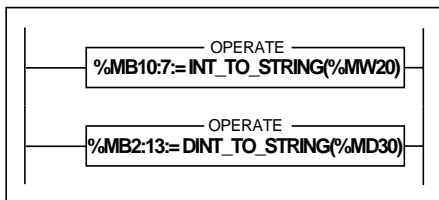
These functions are used to convert a numeric value (single or double length word) to a character string in ASCII code.

Each digit, as well as the sign of the value provided as a parameter, is coded in ASCII in an element of the result byte table.

- **INT_TO_STRING** function : The contents of a single length word can be between -32768 and +32767, that is, 5 digits plus the sign. The result will therefore be a table of 6 characters plus the string termination character. The sign '+' or '-' is stored in the first character, the units in the sixth character, the tens in the fifth, and so on.
- **DINT_TO_STRING** function : The contents of a double length word can be between -2147483648 and +2147483647, that is, 10 digits plus the sign. The result will therefore be a table of 12 characters plus the string termination character. The sign '+' or '-' is stored in the first character, the unit in the twelfth character, the tens in the eleventh, and so on. The second character is always '0'.

Structure

Ladder language



Instruction list language

```
LD TRUE
[ %MB10:7 := INT_TO_STRING (%MW20)]
```

Structured text language

```
%MB2:13:=DINT_TO_STRING (%MD30) ;
```

Example : Binary ---> ASCII conversion

%MB10:7 := INT_TO_STRING (%MW20) where %MW20 = - 3782 in decimal
 ==> The result is stored in the following 7-byte table %MB10:7

%MB	10	11	12	13	14	15	16
	'.'	'0'	'3'	'7'	'8'	'2'	Ø

Example : **%MB2:13 := DINT_TO_STRING (%MD30)** where %MD30 = - 234701084

%MB	2	3	4	5	6	7	8	9	10	11	12	13	14
	'.'	'0'	'0'	'2'	'3'	'4'	'7'	'1'	'1'	'0'	'8'	'4'	Ø

Syntax

Operator

```
result := INT_TO_STRING (value)
```

Operands

Type	Result	Value
6-byte tables + string termination character	%MB:7	
Indexable words		%MW, %KW
Non-indexable words		%IW,%QW,%SW,%NW Immed. val., %Xi.T, Num. expr.

Operator

```
result := DINT_TO_STRING (value)
```

Operands

Type	Result	Value
12-byte tables + string termination character	%MB:13	
Indexable double words		%MD, %KD
Non-indexable double words		%ID,%QD,%SD, Immed. val., Numeric expr.

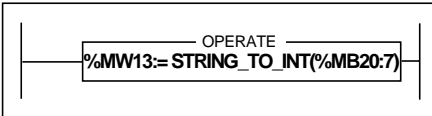
2.8-6 ASCII ---> Binary conversion

This function is used to convert a character string representing a numeric value into binary code (result transferred to a single or double length word).

Each element of the table provided as a parameter represents the ASCII code of a character. Authorized characters are digits and the characters '+' and '-'.

- **STRING_TO_INT** function : converts a string of 6 characters representing a numeric value between -32768 and +32767. The first character must represent the sign and the following characters the value. The second character represents the tens of thousands, ..., and the sixth character the units. The value must be right-justified in the string.
- **STRING_TO_DINT** function : converts a string of 12 characters representing a numeric value between -2147483648 and +2147483647. The first character must represent the sign and the following characters the value. The second is the character '0', the third the thousands of millions, ..., the twelfth the units. The value must be right-justified in the string.

Structure



```
LD      TRUE
[%MW13 := STRING_TO_INT (%MB20:7)]
```

Example : `%MW13 := STRING_TO_INT (%MB20:7)` where

%MB	20	21	22	23	24	25	26	
	'.'	'0'	'2'	'3'	'4'	'7'	Ø	==> result %MW13 = -2347 in decimal

Syntax

Operator

```
result := STRING_TO_INT (string)
```

Operands

Type	Result	String
Indexable words	%MW	
Non-indexable words	%QW,%SW,%NW.	
6-byte tables + string termination character		%MB:7,%KB:7, Immed. val.

Bit %S18 is set to 1 if the value described by the string is not between -32768 and +32767 or if one of the 6 characters is incorrect.

Operator

```
result := STRING_TO_DINT (string)
```

Operands

Type	Result	String
Indexable double words		%MD
Non-indexable double words	%QD,%SD	
12-byte tables + string termination character		%MB:13,%KB:13, Immed. val.

Bit %S18 is set to 1 if the value described by the string is not between -2147483648 and +2147483647 or if one of the 12 characters is incorrect.

2.8-7 Floating point ---> ASCII conversion

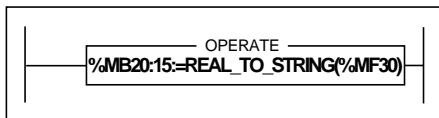
This function is used to convert a real numeric value contained in a floating point type word to a character string coded in ASCII. The result is transferred to a table of 14 bytes + the string termination character.

Each digit in the value and the characters '+', '-', '.', 'e' and 'E' are coded in ASCII in an element in the result table.

The sign of the value is located in the first character, the decimal point (.) in the third, the exponent 'e' in the eleventh and the sign of the exponent in the twelfth.

Structure

Ladder language



Instruction list language

```
LD      TRUE
[ %MB20:15 := REAL_TO_STRING (%MF30)]
```

Structured text language

```
%MB20:15 := REAL_TO_STRING (%MF30) ;
```

Example : `%MB20:15 := REAL_TO_STRING (%MF30)` where `%MF30 = - 3.234718 e26`
 ==> result

%MB	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34
	'.'	'3'	'.'	'2'	'3'	'4'	'7'	'1'	'8'	'0'	'e'	'+'	'2'	'6'	Ø

Syntax

Operator

```
result := REAL_TO_STRING (value)
```

Operands

Type	Result	Value
14-byte tables + string termination character	%MB:15	
Indexable words		%MF, %KF
Non-indexable words		Immediate val., Num. expr.

Bit %S18 is set to 1 if the floating point value provided as a parameter is not between - 3.402824e+38 and -1.175494e-38 or +1.175494e-38 and +3.402824e+38. In this case, the value of the result is incorrect.

2.8-8 ASCII --> Floating point conversion

This function is used to convert a character string representing a real numeric value to floating point (result transferred to a floating point type word).

Each element in the table provided as a parameter represents the ASCII code of one character. Authorized characters are digits and the characters '+', '-', '.', 'e' and 'E'. The string termination character is not used to determine the end of the string. This means that the 14 characters of the table must all be correct.

The sign of the value must be located in the first character, the decimal point (.) in the third, the 'e' in the eleventh and the sign of the exponent in the twelfth.

For example, the value 3.12 must be in the form '+3.120000e+00'.

Structure

Ladder language



Instruction list language

```
LD      TRUE
[ %MF18 := STRING_TO_REAL (%MB20:14)]
```

Structured text language

```
%MF18 := STRING_TO_REAL (%MB20:14) ;
```

Example : `%MF18 := STRING_TO_REAL (%MB20:14)`

where

%MB	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34
	'.'	'3'	'.'	'2'	'3'	'4'	'7'	'1'	'8'	'0'	'e'	'+'	'2'	'6'	Ø

==> result %MF18 = - 3.234718e26

Syntax

Operator

```
result := STRING_TO_REAL (string)
```

Operands

Type	Result	String
Indexable words	%MF	
14-byte tables		%MB:14, %KB:14 Immediate value

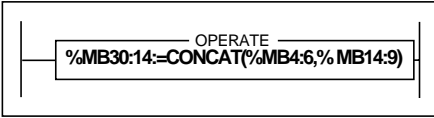
Bit %S18 is set to 1 if the value described by the string is not between -3.402824e+38 and -1.175494e-38 or +1.175494e-38 and +3.402824e+38 or if one of the 14 characters is incorrect.

2.8-9 Concatenation of two strings

This function allows concatenation of two character strings as defined by parameters. The result is a byte table containing a character string.

Structure

Ladder language



Instruction list language

```
LD TRUE
[ %MB30:14 := CONCAT (%MB4:6, %MB14:9)]
```

Structured text language

```
%MB30:14 := CONCAT (%MB4:6, %MB14:9) ;
```

Example : **%MB30:14 := CONCAT (%MB4:6, %MB14:9)**

%MB	4	5	6	7	8	9
	'i'	'n'	'c'	'o'	'n'	Ø

%MB	14	15	16	17	18	19	20	21	22
	't'	'e'	's'	't'	'a'	'b'	'l'	'e'	Ø

%MB	30	31	32	33	34	35	36	37	38	39	40	41	42	43
	'i'	'n'	'c'	'o'	'n'	't'	'e'	's'	't'	'a'	'b'	'l'	'e'	Ø

Syntax

Operator

```
result :=CONCAT (string1, string2)
```

Operands

Type	Result	String 1 and 2
Byte tables	%MB:L	%MB:L,%KB:L, Immed. val.

- If the result table is too short, the result is truncated and system bit %S15 is set to 1. **%MB30:10 := CONCAT (%MB4:6, %MB14:9)**

%MB	30	31	32	33	34	35	36	37	38	39
	'i'	'n'	'c'	'o'	'n'	't'	'e'	's'	't'	'a'

==> %S15 at 1

- If the result table is too long, termination characters 'Ø' are added to the string. **%MB30:15 := CONCAT (%MB4:6, %MB14:9)**

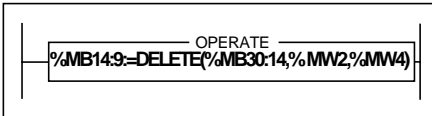
%MB	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44
	'i'	'n'	'c'	'o'	'n'	't'	'e'	's'	't'	'a'	'b'	'l'	'e'	Ø	Ø

2.8-10 Deletion of a character substring

This function is used to delete a number of characters (zone length L), from a given position (position of the first character to be deleted) in the string defined as a parameter. The result is a byte table containing a character string.

Structure

Ladder language



Instruction list language

```
LD      TRUE
[%MB14: 9 := DELETE (%MB30:14, %MW2, %MW4)]
```

Structured text language

```
%MB14:9 := DELETE (%MB30:14, %MW2, %MW4) ;
```

Example : **%MB14: 9 := DELETE (%MB30:14, %MW2, %MW4)**

with %MW2 = 5 (5 characters to be deleted) %MW4 = 3 (position =3)

%MB	30	31	32	33	34	35	36	37	38	39	40	41	42	43
	'i'	'n'	'c'	'o'	'n'	't'	'e'	's'	't'	'a'	'b'	'l'	'e'	Ø
%MB	14	15	16	17	18	19	20	21	22					
	'i'	'n'	's'	't'	'a'	'b'	'l'	'e'	Ø					

Syntax

Operator

```
result :=DELETE (string, length, pos)
```

Operands

Type	Result	String	Length pos (position)
Byte tables	%MB:L	%MB:L,%KB:L Immediate val.	
Indexable words			%MW, %KW
Non-indexable words			%IW,%QW,%SW,%NW Immediate value, %Xi.T Numeric expr.

Notes :

Some parameters may overlap depending on the indices of PL7 objects :

- Table containing the source string.
- Table containing the result string.
- Word containing the length to be deleted.
- Word containing the position of the first character to be deleted.

A negative length or position is interpreted as being 0. The position parameter starts at the value 1 which corresponds to the first position in the character string.

If the result table is too long, termination characters \emptyset are added to the string.

System bit %S15 is set to 1 in the following cases :

- The length to be deleted is zero, the output table is a copy of the source table.
- The position is greater than the length of the string, or the position of the first termination character found is less than or equal to the position of the first character to be deleted. The result is therefore an empty string.
- The position is equal to 0. The result table therefore contains an empty string.
- The result table is too short. It has therefore been truncated.

2.8-11 Insertion of a character substring

Insertion of the character substring defined by the second parameter (string2) in the character string defined by the first parameter (string1).

The insertion is made in the first string, after the character in the location given by the position parameter (Pos).

The result of the insertion is a new character string transferred to a byte table.

Structure

Ladder language



Instruction list language

```
LD      TRUE
[%MB2:14 := INSERT (%MB20:9, %MB30:6, %MW40)]
```

Structured text language

```
%MB2:14 := INSERT (%MB20:9, %MB30:6, %MW40) ;
```

Example : **%MB2:14 := INSERT (%MB20:9, %MB30:6, %MW40)**

where %MW40 := position 2

%MB	20	21	22	23	24	25	26	27	28
	'i'	'n'	's'	't'	'a'	'b'	'l'	'e'	\emptyset

%MB	30	31	32	33	34	35
	'c'	'o'	'n'	't'	'e'	\emptyset

%MB	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	'i'	'n'	'c'	'o'	'n'	't'	'e'	's'	't'	'a'	'b'	'l'	'e'	\emptyset

Syntax

Operator

result :=INSERT (string1, string2, pos)

Operands

Type	Result	String 1 and 2	Pos (position)
Byte tables	%MB:L Immed. value	%MB:L,%KB:L	
Indexable words			%MW, %KW
Non-indexable words			%IW,%QW,%SW,%NW Immediate value, %Xi.T Numeric expr.

Notes :

The position parameter starts at the value 1 which corresponds to the first position in the character string.

It is impossible to insert at the beginning of a string. In order to do this, use the CONCAT function.

If the table is too long, termination type characters must be added.

System bit %S15 is set to 1 in the following cases :

The value of the position parameter is negative or equal to 0. In this case, it is interpreted as being 0 and the result table contains an empty string (composed of termination characters).

The position provided as a parameter is greater than or equal to the length of the source string. The result table then contains an empty string (composed of termination characters).

If the result table is too short, truncation occurs.

2.8-12 Replacement of a character substring

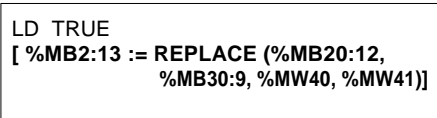
This function is used to replace a section of a character string defined in the source table (string1) by a character substring defined in the replacement table (string2). The replacement to be made is defined by the position (pos.) and length parameters. **This length corresponds to the length of the string which is removed and not to the length of the substring which replaces it.**

Structure

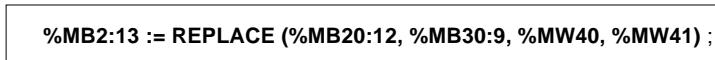
Ladder language



Instruction list language



Structured text language



Example : `%MB2:13 := REPLACE (%MB20:12, %MB30:9, %MW40, %MW41)`
 where %MW40 = 3 (length=3) and %MW41 = 9 (position=9)

%MB	20	21	22	23	24	25	26	27	28	29	30	31
String 1	'm'	'i'	's'	'e'	'_'	'e'	'n'	'_'	'r'	'u'	'n'	∅

%MB	30	31	32	33	34	35	36	37	38
String 2	's'	't'	'o'	'p'	∅	'r'	'u'	'n'	∅

%MB	2	3	4	5	6	7	8	9	10	11	12	13	14
	'm'	'i'	's'	'e'	'_'	'e'	'n'	'_'	's'	't'	'o'	'p'	∅

Syntax

Operator

```
result := REPLACE (string1, string2, length, pos.)
```

Operands

Type	Result	String 1 and 2	Length pos (position)
Byte tables	%MB:L	%MB:L,%KB:L Immed. value	
Indexable words			%MW, %KW
Non-indexable words			%IW,%QW,%SW,%NW Immediate value,%Xi.T Numeric expr.

Notes :

The position parameter starts at the value 1 which corresponds to the first position in the character string.

If the output table is too long, termination type characters are added to the string.

System bit %S15 is set to 1 in the following cases :

- If the value of the position parameter is negative or equal to 0. In this case, it is interpreted as being 0 and the result table contains an empty string (composed of termination characters).
- If the position provided as a parameter is greater than or equal to the length of the source string, the result table then contains an empty string (composed of termination characters).
- If the result table is too short, truncation occurs.
- If the position of the first string termination character is less than or equal to the position of the first character to be replaced, the output table is a copy of the source table up to the string termination character and completed by termination characters.

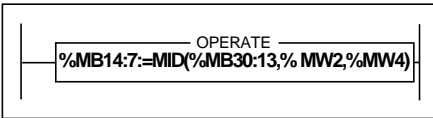
2.8-13 Extraction of a character substring

This function is used to extract a number of characters from a source string provided as a parameter (string).

The position of the first character to be extracted is determined by the position parameter (pos), and the number of characters to be extracted is given by the length parameter. The extracted string is stored in a byte table (result).

Structure

Ladder language



Instruction list language

```
LD TRUE
[%MB14: 7 := MID (%MB30:13, %MW2, %MW4)]
```

Structured text language

```
%MB14:7 := MID (%MB30:13, %MW2, %MW4) ;
```

Example : **%MB14: 7 := MID (%MB30:13, %MW2, %MW4)**

where %MW2 = 4 (length), %MW4 = 9 (position)

%MB	30	31	32	33	34	35	36	37	38	39	40	41	42
	'm'	'i'	's'	'e'	'_'	'e'	'n'	'_'	's'	't'	'o'	'p'	Ø

==> result

%MB	14	15	16	17	18	19	20
	's'	't'	'o'	'p'	Ø	Ø	Ø

Syntax

Operator

```
result :=MID (string, length, pos)
```

Operands

Type	Result	String	Length pos (position)
Byte tables	%MB:L Immed. value	%MB:L,%KB:L	
Indexable words			%MW,%KW
Non-indexable words			%IW,%QW,%SW,%NW Immediate value,%Xi.T Numeric expr.

Notes :

The position parameter starts at value 1 which corresponds to the first position in the character string.

If the output table is too long, termination type characters are added to the result string.

If the length provided as a parameter is greater than the size of the source string, the result table then contains the source string.

If the last element of the table or the string termination character is reached before the number of characters defined by the length parameter has been extracted, extraction stops at this point.

System bit %S15 is set to 1 in the following cases :

- If the value of the length parameter to be extracted is negative or equal to 0. In this case, it is interpreted as being 0 and the result table contains an empty string (composed of termination characters).
- If the value of the position parameter for the beginning of the extraction is zero or greater than or equal to the length of the table, or greater than or equal to the position of the first termination character. In this case, the result table contains an empty string (composed of termination characters).
- If the result table is too short, truncation occurs.

2.8-14 Extraction of characters

Extraction of a number of characters the furthest to the left (LEFT) or furthest to the right (RIGHT) in a source string provided as a parameter (string).

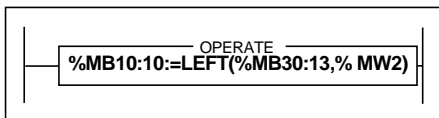
The number of characters to be extracted is defined by the length parameter.

The extracted string is stored in a byte table (result).

Structure

Ladder language

Instruction list language



```
LD TRUE
[%MB10: 10 := LEFT (%MB30:13, %MW2)]
```

Structured text language

```
%MB10:10 := LEFT (%MB30:13, MW2) ;
```

Example : **%MB10: 10 := LEFT (%MB30:13, %MW2)**

where %MW2 = 8 (length)

%MB	30	31	32	33	34	35	36	37	38	39	40	41	42
	'm'	'i'	's'	'e'	'_'	'e'	'n'	'_'	's'	't'	'o'	'p'	Ø

==> result

%MB	10	11	12	13	14	15	16	17	18	19
	'm'	'i'	's'	'e'	'_'	'e'	'n'	'_'	Ø	Ø

Syntax

Operator

```
result :=LEFT (string, length)
```

```
result :=RIGHT (string, length)
```

Operands

Type	Result	String	Length
Byte tables	%MB:L	%MB:L,%KB:L Immed. value	
Indexable words			%MW,%KW
Non-indexable words			%IW,%QW,%SW,%NW Immediate value,%Xi.T Numeric expr.

Notes :

If the output table is too long, termination type characters are added to the result string.

If the length provided as a parameter is greater than the size of the source string, the result table then contains the source string.

System bit %S15 is set to 1 in the following cases :

- If the value of the length parameter to be extracted is negative or 0. In this case, the result table contains an empty string (composed of termination characters).
- If the result table is too short, truncation occurs.

2.8-15 Comparison of two character strings

This function is used to compare two character strings. The result is a word containing the position of the first different character.

If the two character strings are exactly the same, the result is -1.

Structure

Ladder language

Instruction list language



```
LD TRUE
[%MW2 := EQUAL_STR (%MB18:14, %MB50:14)]
```

Structured text language

```
%MW2 := EQUAL_STR (%MB18:14, %MB50:14) ;
```

Example : **%MW2 := EQUAL_STR (%MB18:14, %MB50:14)** where

%MB	18	19	20	21	22	23	24	25	26	27	28	29	30	31
	'a'	'b'	'c'	'd'	'e'	'f'	'g'	'h'	'i'	'p'	'w'	'x'	'y'	'z'

%MB	50	51	52	53	54	55	56	57	58	59	60	61	62	63
	'a'	'b'	'c'	'd'	'?'	'f'	'g'	'h'	∅	'v'	'w'	'x'	'y'	'z'

==> MW2 := 5

Syntax

Operator

```
result :=EQUAL_STR (string1, string2)
```

Operands

Type	Result	String 1 and 2
Indexable words	%MW	
Non-indexable words	%QW,%SW,%NW.	
Byte tables		%MB:L,%KB:L Immediate value

Note :

A negative length or position is interpreted as being equal to 0.

Upper case letters are different from lower case letters.

2.8-16 Search for a character substring

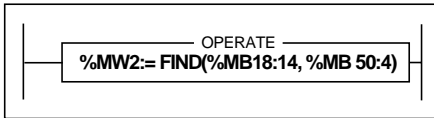
This function is used to search for the character substring defined by the second parameter in the character string defined by the first parameter.

The result is a word containing the position in the first string of the beginning of the substring searched for.

If the search is not successful, the result is -1.

Structure

Ladder language



Instruction list language

```
LD TRUE
[ %MW2 := FIND (%MB18:14, %MB50:4)]
```

Structured text language

```
%MW2 := FIND (%MB18:14, %MB50:4) ;
```

Example : `%MW2 := FIND (%MB18:14, %MB50:4)` where

%MB	18	19	20	21	22	23	24	25	26	27	28	29	30	31
	'a'	'b'	'c'	'd'	'e'	'f'	'g'	'h'	'i'	Ø	'w'	'x'	'y'	'z'

%MB	50	51	52	53
	'f'	'g'	'h'	Ø

==> `MW2 := 6` Indicates that the beginning of the string searched for is located from the sixth character onwards.

Syntax

Operator

```
result :=FIND (string1, string2)
```

Operands

Type	Result	String 1 and 2
Indexable words	%MW	
Non-indexable words	%QW,%SW,%NW.	
Byte tables		%MB:L,%KB:L Immediate value

Note :

A negative length or position is interpreted as being equal to 0.

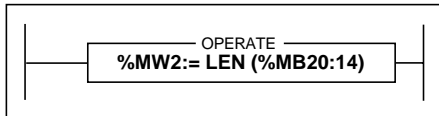
2.8-17 Length of a character string

This function gives the length of the character string provided as a parameter, that is, the number of characters located before the termination character.

Structure

Ladder language

Instruction list language



```
LD TRUE
[%MW2 := LEN (%MB20:14)]
```

Structured text language

```
%MW2 := LEN (%MB20:14) ;
```

Example : `%MW2 := LEN (%MB20:14)` where

%MB	20	21	22	23	24	25	26	27	28	29	30	31
	'a'	'b'	'c'	'd'	'e'	'f'	'g'	Ø	'n'	'o'	'p'	'r'

==> %MW2 = 7

Syntax

Operator

```
result := LEN (string)
```

Operands

Type	Result	String
Indexable words	%MW	
Non-indexable words	%QW,%SW,%NW.	
Byte tables		%MB:L, %KB:L, Immediate value

Note :

If no termination character is found, the function gives the size of the table (see section 2.8-1).

2.9 Time management instructions : Date, Time of day, Duration

2.9-1 Parameter format

The Date, Time of day and Duration parameters used by these instructions correspond to the standard formats defined by IEC standard 1131-3.

• Duration format (TIME type)

This format is used to code durations expressed in tenths of a second and corresponds to the TIME format of the standard.

These values are displayed in the following way : **sssssssss.d**

which gives for example : 3674.3

for 1 hour, 1 minute, 14 seconds and 3 tenths of a second.

The value is coded on 32 bits (a double word) of which the limits are set at [0, 4294967295] tenths of a second, which represents approximately 13 years and 7 months.

• Data format (DATE type)

This format is used to code the year, the month and the day. It corresponds to the DATE format of the standard.

The value is displayed in the following way : **yyyy-mm-dd**

which gives for example : 1984-06-25

The value is coded in BCD on 32 bits (a double word) with 3 fields :

31		16		8		0	Year : 4 digits			
<table style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 33%; border: 1px solid black; text-align: center; padding: 5px;">Year</td> <td style="width: 33%; border: 1px solid black; text-align: center; padding: 5px;">Month</td> <td style="width: 33%; border: 1px solid black; text-align: center; padding: 5px;">Day</td> </tr> </table>							Year	Month	Day	Month : 2 digits
Year	Month	Day								
							Day : 2 digits			

Example :

expressed in hexadecimal format

19h	84h	06h	25h	= 1984-06-25
-----	-----	-----	-----	--------------

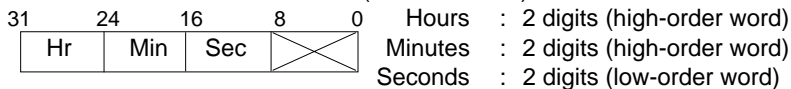
Only values within the time period [1990-01-01, 2099-12-31] are permitted.

• Time of day format (TOD type)

This format is used to code the hour, the minutes and the seconds. It corresponds to the TIME_OF_DAY format of the standard.

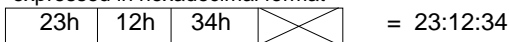
The value is displayed in the following way : **hh:mm:ss**
which gives for example : 23:12:34

The value is coded in BCD on 32 bits (a double word) with 3 fields :



Example :

expressed in hexadecimal format



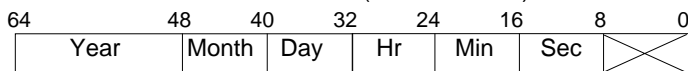
Only values within the time period [00:00:00, 23:59:59] are permitted.

• Date and time format (DT type)

This format is used to code the year, the month, the day, the hour, the minutes and the seconds. It corresponds to the DATE_AND_TIME format of the standard.

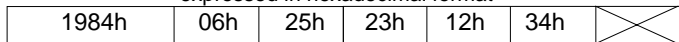
The value is displayed in the following way : **yyyy-mm-dd-hh:mm:ss**
which gives for example : 1984-06-25-23:12:34

The value is coded in BCD on 64 bits (a 4-word table) :



Example :

expressed in hexadecimal format



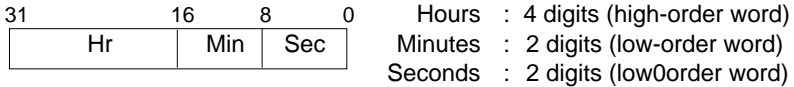
Only values within the time period [1990-01-01-00:00:00, 2099-12-31-23:59:59] are permitted.

• Hour, Minute, Second Format (HMS type)

This format, **used exclusively by the function TRANS_TIME**, is used to code the hour, the minutes and the seconds.

The value is displayed in the following way : **hh:mm:ss**
which gives for example : 23:12:34

The value is coded in BCD on 32 bits (a double word) with 3 fields :



Example :

expressed in hexadecimal format



2.9-2 Use of system bits and words - General

System bit **%S17** is set in the following cases :

- Result of an operation outside the permitted time period values.
- An input parameter cannot be interpreted and is not consistent with the required format (DAT, DT or TOD).
- Operation on a Time of day (TOD) format leading to a change in the day.
- Access clash to the real-time clock.

System bit **%S15** is set to 1 if a string written in a table is longer than the size of that table.

System words :

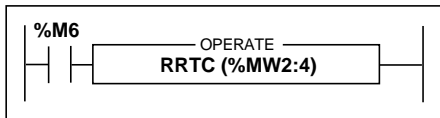
- **%SD18** : absolute time counter is also used to perform time period calculations (incremented every 1/10 of a second by the system).
- **%SW49** to **%SW53** can also be used to display dates (see section 3.2-2, part B).

2.9-3 Read system date

Reads the system date (Real-Time Clock) and transfers to the object given as a parameter in the Date and time (DT) format.

Structure

Ladder language



Instruction list language

```
LD %M6
[RRTC (%MW2:4)]
```

Structured text language

```
IF %M6 THEN
  RRTC (%MW2:4) ;
END_IF ;
```

Example : **RRTC (%MW2:4)**

The result is transferred to the table of internal words which is 4 words long : %MW2 to %MW5.

Syntax

Operator

```
RRTC(date)
```

Operand

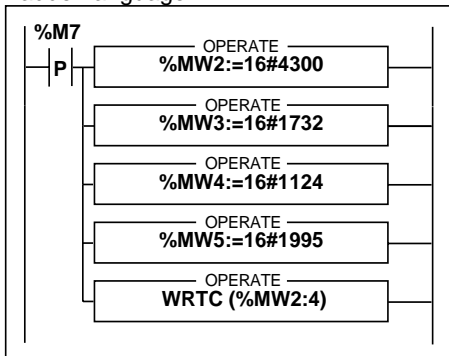
Type	Date
4-word tables in date and time format	%MW:4

2.9-4 Update system date

Updates the system date (Real-Time Clock) and transfers to the object given as a parameter in Date and time (DT) format.

Structure

Ladder language



Instruction list language

```
LDR %M7
[%MW2:= 16#4300]
[%MW3:= 16#1732]
[%MW4:= 16#1124]
[%MW5:= 16#1995]
[WRTC (%MW2:4)]
```


Structured text language

```

IF RE %M7 THEN
  %MW2 := 16#4300 ;
  %MW3 := 16#1732 ;
  %MW4 := 16#1124 ;
  %MW5 := 16#1995 ;
  WRTC (%MW2:4) ;
END_IF ;

```

Example : The new date is loaded into an internal word table, %MW2:4, which is 4 words long and then sent to the system using the WRTC functions.

Syntax

Operator

WRTC(date)

Operand

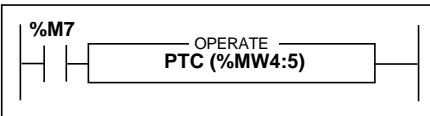
Type	Date
4-word tables	%MW:4, %KW:4 in date and time format

2.9-5 Read date and stop code

Reads the date of the last PLC stop and the code specifying the cause of the stop (in the fifth word, equivalent to %SW58. See section 3.2-2, part B).

Structure

Ladder language



Instruction list language

```

LD %M7
[PTC (%MW4:5)]

```

Structured text language

```

IF %M7 THEN
  PTC (%MW4:5) ;
END_IF ;

```

Example : **PTC (%MW4:5)**

The result is transferred to the table of internal words which is 5 words long : %MW4 to %MW8.

Syntax

Operator

PTC (date)

Operand

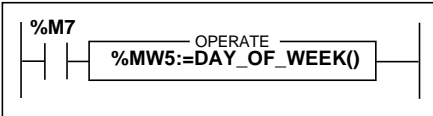
Type	Date
5-word tables in date and time format	%MW:5

2.9-6 Read day of the week

This function gives the current day of the week in the form of a digit from 1 to 7 which is transferred to a word (1 = Monday, 2 = Tuesday, 3 = Wednesday, 4 = Thursday, 5 = Friday, 6 = Saturday, 7 = Sunday).

Structure

Ladder language



Instruction list language

```
LD    %M7
[%MW5 := DAY_OF_WEEK()]
```

Structured text language

```
IF %M7 THEN
  %MW5 := DAY_OF_WEEK ();
END_IF ;
```

Example : %MW5 := DAY_OF_WEEK() %MW5 := 4 corresponds to Thursday

Syntax

Operator

```
result :=DAY_OF_WEEK()
```

Operand

Type	Result
Indexable words	%MW
Non-indexable words	%QW, %SW, %NW

Note

If the function was unable to update the result following an access error to the real-time clock, the result given is 0 and system bit %S17 is set to 1.

2.9-7 Add / Remove a duration at a date

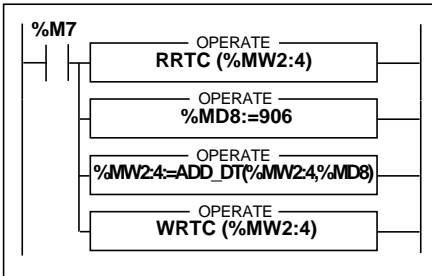
Adds or removes a duration (in tenths of a second) (In2) at a source date (In1). The result is a new date, transferred to a table of 4 words.

ADD_DT () = Add a duration

SUB_DT () = Remove a duration

Structure

Ladder language



Instruction list language

```
LD      %M7
[RRTC (%MW2:4)]
[%MD8 := 906]
[%MW2:4:= ADD_DT(%MW2:4, %MD8)]
[WRTC (%MW2:4)]
```

Structured text language

```
IF %M7 THEN
  RRTC (%MW2:4) ;
  %MD8 := 906 ;
  %MW2:4 := ADD_DT (%MW2:4, %MD8) ;
  WRTC (%MW2:4) ;
END_IF ;
```

Example : **%MW2:4 := ADD_DT(%MW2:4, %MD8)**

%MW2:4 := Source date

%MD8 := 906 (906 tenths of a second rounded to 1 min. 31 s)

%MW2:4 := New date

Syntax

Operators

result :=**ADD_DT** (In1, In2)

result :=**SUB_DT** (In1, In2)

Operands

Type	Result	In1(initial date)	In2 (time period)
Table of four words in the date and time format	%MB:4	%MW4:4, %KW:4	
Indexable double words			%MD,%KD
Non-indexable double words			%ID,%QD Immediate value Numeric expr.

Notes :

- The "duration" parameter (expressed in 1/10 of a second) will be rounded up or down so that the date and time can be increased or decreased (precision to within one second).
 - ssssssss.0 to ssssssss.4 rounded to ssssssss.0
 - ssssssss.5 to ssssssss.9 rounded to ssssssss.0 + 1.0
- Provision must be made in the application to handle leap years.
- If the result of the operation is outside the permitted time period values, system bit %S17 is set to 1 and the value of the result equals the minimum limit (for SUB_DT) or remains blocked at the maximum (for ADD_DT).
- If the "source date" input parameter cannot be interpreted and is not consistent with the DT (DATE_AND_TIME) format, system bit %S17 is set to 1 and the value of the result is 0001-01-01-000:00:00.

2.9-8 Add / Remove a duration at a time of day

Adds or removes a time period at a time of day. The result is a new time of day which is transferred to a double word.

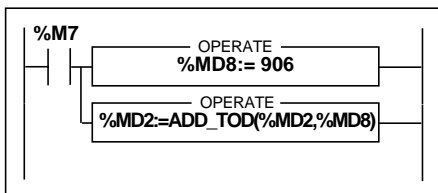
ADD_TOD () = Add a duration

SUB_TOD () = Remove a duration

Structure

Ladder language

Instruction list language



```
LD      %M7
[%MD8 := 906]
[%MD2 := ADD_TOD (%MD2, %MD8)]
```

Structured text language

```

IF %M7 THEN
  %MD8 := 906 ;
  %MD2 := ADD_TOD (%MD2, %MD8) ;
END_IF ;

```

Example : **%MD2 := ADD_TOD (%MD2, %MD8)**
 %MD2 := Initial time (eg. 12:30:00)
 %MD8 := 906 (906 tenths of a second rounded to 1 min. 31 s)
 %MD2 := New time (eg. 12:31:31)

Syntax

Operators

```
result :=ADD_TOD (In1, In2)
```

```
result :=SUB_TOD (In1, In2)
```

Operands

Type	Result	In1(initial time) and In2 (time period)
Indexable double words	%MD	%MD,%KD
Non-indexable double words	%QD	%ID,%QD Immediate value, Numeric expr.

result and **In1** are in the TOD format and **In2** is in the time period format.

Notes :

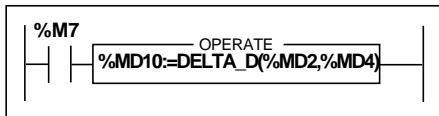
- The "time period" parameter (expressed in 1/10 of a second) will be rounded up or down so that the date and time can be increased or decreased (precision to within one second).
 - ssssssss.0 to ssssssss.4 rounded to ssssssss.0
 - ssssssss.5 to ssssssss.9 rounded to ssssssss.0 + 1.0
- The day changes if the result of the operation is outside the permitted time period values. In this case, system bit %S17 is set to 1 and the value of the result can be interpreted with a modulo 24:00:00.
- If the "time of day" input parameter cannot be interpreted in the TOD format, system bit %S17 is set to 1 and the result is 00:00:00.

2.9-9 Difference between two dates (no time)

This function is used to calculate the difference in days between two dates. The result, given as an absolute value, is transferred to a double word.

Structure

Ladder language



Instruction list language

```
LD      %M7
[%MD10 := DELTA_D (%MD2, %MD4)]
```

Structured text language

```
IF %M7 THEN
  %MD10 := DELTA_D (%MD2, %MD4) ;
END_IF ;
```

Example : **%MD10 := DELTA_D (%MD2, %MD4)**
 %MD2 := Date number1 (eg. 1994-05-01)
 %MD4 := Date number2 (eg. 1994-04-05)
 ==> %MD10 = 22464000 (==> difference = 26 days)

Syntax

Operator

```
result :=DELTA_D(Date1,Date2)
```

Operands

Type	Result	Date 1 and 2
Indexable double words	%MD	%MD,%KD
Non-indexable double words	%QD	%ID,%QD Immediate value, Numeric expr.

result is in the TIME format and **Date 1 and 2** are in the DATE format.

The TIME format is defined to be accurate to within one tenth of a second. The DATE format is defined to be accurate to within one day. The time difference calculated will therefore be a multiple of 864000 (= 1day = 24 h x 60 min x 60 s x 10 tenths of a second).

Warning

- Overflow occurs if the result exceeds the maximum value permitted for a duration (TIME). In this case, the result is 0 and system bit %S18 is set to 1.
- If one of the input parameters cannot be interpreted and is not consistent with the DATE format, system bit %S17 is set to 1 and the result is 0.

2.9-10 Difference between two dates (with time)

This function is used to calculate the time difference between two dates. The result, given as an absolute value, is transferred to a double word.

Structure

Ladder language



Instruction list language

```
LD TRUE
[%MD10 := DELTA_DT (%MW2:4, %MW6:4)]
```

Structured text language

```
%MD10 := DELTA_DT (%MW2:4, %MW6:4) ;
```

Example : **%MD10 := DELTA_DT (%MW2:4, %MW6:4)**
 %MW2:4 := Date number1 (eg. 1994-05-01-12:00:00)
 %MW6:4 := Date number2 (eg. 1994-05-01-12:01:30)
 ==> %MD10 = 900 (==> difference = 1 minute and 30 seconds)

Syntax

Operator

```
result :=DELTA_DT(Date1,Date2)
```

Operands

Type	Result	Date 1 and 2
Indexable double words	%MD,	
Non-indexable double words	%QD	
4-word tables in DT format		%MW:4, %KW:4

result is in the TIME format and **Date 1 and 2** are in the DT format.

The TIME format is defined to be accurate to within one tenth of a second. The DT format is defined to be accurate to within one second. The time difference calculated will therefore be a multiple of 10.

Warning

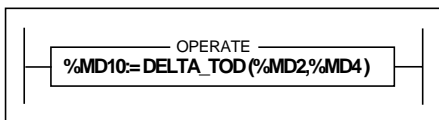
- Overflow occurs if the result exceeds the maximum value permitted for a duration (TIME). In this case, the result is 0 and system bit %S18 is set to 1.
- If one of the input parameters cannot be interpreted and is not consistent with the DT format, system bit %S17 is set to 1 and the result is 0.

2.9-11 Difference between two times

This function is used to calculate the time difference between two times of day. The result is transferred to a double word as an absolute value, giving a duration.

Structure

Ladder language



Instruction list language

```
LD TRUE
[%MD10 := DELTA_TOD (%MD2, %MD4)]
```

Structured text language

```
%MD10 := DELTA_TOD (%MD2, %MD4) ;
```

Example : **%MD10 := DELTA_TOD (%MD2, %MD4)**
 %MD2 := Time1 (eg. 02:30:00)
 %MD4 := Time2 (eg. 02:40:00)
 ==> %MD10 = 6600 (==> difference = 11 minutes)

Syntax

Operator

```
result :=DELTA_TOD(Time1,Time2)
```

Operands

Type	Result	Time 1 and 2
Indexable double words	%MD	%MD,%KD
Non-indexable double words	%QD	%ID,%QD Immediate value, Numeric expr.

result is in the TIME format and **Time 1 and 2** are in the TOD format.

The TIME format is defined to be accurate to within one tenth of a second. The TOD format is defined to be accurate to within one second. The time difference calculated will therefore be a multiple of 10.

Attention

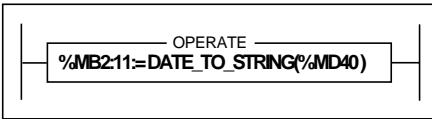
If one of the input parameters cannot be interpreted and is not consistent with the TOD format, system bit %S17 is set to 1 and the result is 0.

2.9-12 Convert a Date to a character string

This instruction converts a date to a character string (no time) in the format : YYYY-MM-DD (10 characters). This string ends with the termination character Ø. Each character Y,M,D represents a number.

Structure

Ladder language



Instruction list language

```
LD      TRUE
[%MB2:11 := DATE_TO_STRING (%MD40)]
```

Structured text language

```
%MB2:11 := DATE_TO_STRING (%MD40) ;
```

Example : **%MB2:11 := DATE_TO_STRING (%MD40)**
 %MD40 := DATE (eg. 1998-12-27)

```
==>  %MB      2  3  4  5  6  7  8  9  10  11  12
      '1' '9' '9' '8' '.' '1' '2' '.' '2' '7'  Ø
```

Syntax

Operator

```
result :=DATE_TO_STRING(Date)
```

Operands

Type	Result	Date
11-byte tables	%MB:11	
Indexable double words		%MD,%KD
Non-indexable double words		%ID,%QD Immediate value, Numeric expr.

Notes : If the input parameter (date) cannot be interpreted and is not consistent with the DATE format, system bit %S17 is set to 1 and the function returns the string : '****_**_**'. If the output string is too short, truncation occurs and system bit %S15 is set to 1.

%MB2:8 := DATE_TO_STRING (%MD40)

```
==>  %MB      2  3  4  5  6  7  8  9
      '1' '9' '9' '8' '.' '1' '2' '.'   ==> %S15 = 1
```

If the output string is too long, termination type characters Ø are added to the string.

%MB2:12 := DATE_TO_STRING (%MD40)

```
==>  %MB      2  3  4  5  6  7  8  9  10  11  12  13
      '1' '9' '9' '8' '.' '1' '2' '.' '2' '7'  Ø  Ø
```

2.9-13 Convert a complete Date to a character string

This instruction converts a complete date (with time) to a character string in the format : YYYY-MM-DD-HH:MM:SS (19 characters). This string ends with the termination character Ø. Each character Y,M,D,H,M,S represents a number.

Structure

Ladder language

Instruction list language



```
LD TRUE
[%MB2:20 := DT_TO_STRING (%MW50:4)]
```

Structured text language

```
%MB2:20 := DT_TO_STRING (%MW50:4) ;
```

Example : **%MB2:20 := DT_TO_STRING (%MW50:4)**
 %MW50:4 := Date and time (type DT) (eg. 1998-12-27-23:14:37)

%MB	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
	'1'	'9'	'9'	'8'	'.'	'1'	'2'	'.'	'2'	'7'	'.'	'2'	'3'	'.'	'1'	'4'	'.'	'3'	'7'	Ø

Syntax

Operator

```
result :=DT_TO_STRING(Date)
```

Operands

Type	Result	Date
20-byte tables	%MB:20	
4-word tables in DT format		%MW:4, %KW:4

Notes : If the input parameter (date) cannot be interpreted and is not consistent with the DT format (DATE_AND_TIME), system bit %S17 is set to 1 and the function returns the string '****_**_**_** : ** : **'. If the output string is too short, truncation occurs and system bit %S15 is set to 1.

%MB2:8 := DT_TO_STRING (%MW50:4)

==> %MB

2	3	4	5	6	7	8	9
'1'	'9'	'9'	'8'	'.'	'1'	'2'	'.'

==> %S15 = 1

- If the output string is too long, termination type characters Ø are added to the string.

%MB2:21 := DT_TO_STRING (%MW50:4)

==>

%MB	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
	'1'	'9'	'9'	'8'	'.'	'1'	'2'	'.'	'2'	'7'	'.'	'2'	'3'	'.'	'1'	'4'	'.'	'3'	'7'	Ø	Ø

2.9-14 Convert a Duration to a character string

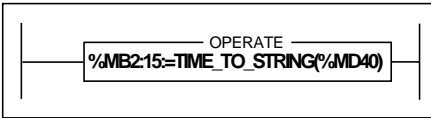
This instruction converts a time period (in the TIME format) to a character string.

The format of the result can be broken down into hours, minutes, seconds and tenths of a second over 15 characters : HHHHHH:MM:SS.D. This string ends with the termination character Ø. Each character H,M,S,D represents a number.

The maximum time period corresponds to 119304 hours, 38 minutes, 49 seconds and 5 tenths of a second.

Structure

Ladder language



Instruction list language

```
LD TRUE
[%MB2:15 := TIME_TO_STRING (%MD40)]
```

Structured text language

```
%MB2:15 := TIME_TO_STRING (%MD40) ;
```

Example : `%MB2:15 := TIME_TO_STRING (%MD40)`

where `%MD40 := 27556330.3` (TIME format)

<i>%MB</i>	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
	'0'	'0'	'7'	'6'	'5'	'4'	':'	'3'	'2'	':'	'1'	'0'	':'	'3'	Ø

Syntax

Operator

```
result :=TIME_TO_STRING(Duration)
```

Operands

Type	Result	Duration
15-byte tables	%MB:15	
Indexable double words		%MD,%KD
Non-indexable double words		%ID,%QD Immediate value, Numeric expr.

Time period : is in the TIME format.

Note :

If the output string is too short, truncation occurs and system bit %S15 is set to 1.

`%MB2:8 := TIME_TO_STRING (%MD40)`

```
==> %MB
```

2	3	4	5	6	7	8	9
'0'	'0'	'7'	'6'	'5'	'4'	':'	'3'

```
==> %S15 = 1
```

If the output string is too long, termination type characters \emptyset are added to the string.

%MB2:16 := TIME_TO_STRING (%MD40)

==>

%MB	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
	'0'	'0'	'7'	'6'	'5'	'4'	':'	'3'	'2'	':'	'1'	'0'	':'	'3'	\emptyset	\emptyset

2.9-15 Convert a Time of day to a character string

This instruction converts a time of day (in the format TOD - TIME_OF_DAY) to a character string in the format HH:MM:SS on 8 characters plus a termination character \emptyset . Each character H,M,S represents a number.

Structure

Ladder language

Instruction list language



```
LD TRUE
[%MB2:9 := TOD_TO_STRING (%MD40)]
```

Structured text language

```
%MB2:9 := TOD_TO_STRING (%MD40) ;
```

Example : **%MB2:9 := TOD_TO_STRING (%MD40)**

where %MD40 := 23:12:27 (TOD format)

==>

%MB	2	3	4	5	6	7	8	9	10
	'2'	'3'	':'	'1'	'2'	':'	'2'	'7'	\emptyset

Syntax

Operator

```
result :=TOD_TO_STRING(time)
```

Operands

Type	Result	Time of day
9-byte tables	%MB:9	
Indexable double words		%MD,%KD
Non-indexable double words		%ID,%QD Immediate value, Numeric expr.

time : is in the TOD format.

Note :

If the output string is too short, truncation occurs and system bit %S15 is set to 1.

%MB2:8 := TOD_TO_STRING (%MD40) (where %MD40 := 23:12:27)

==> %MB

2	3	4	5	6	7	8	9
'2'	'3'	':'	'1'	'2'	':'	'2'	'7'

==> %S15 = 1

If the output string is too long, termination type characters \emptyset are added to the string.

%MB2:10 := TOD_TO_STRING (%MD40) (where %MD40 := 23:12:27)

==> %MB

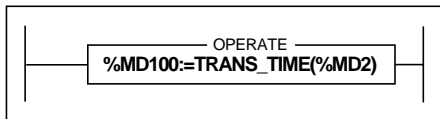
2	3	4	5	6	7	8	9	10	11
'2'	'3'	':'	'1'	'2'	':'	'2'	'7'	\emptyset	\emptyset

2.9-16 Convert a Duration to HHHH:MM:SS

This instruction converts a duration (in the TIME format) to a number of hours-minutes-seconds, HHHH:MM:SS. The limit values are [0000:00:00, 9999:59:59].

Structure

Ladder language



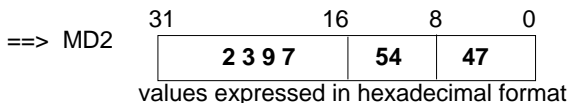
Instruction list language

```
LD      TRUE
[%MD100 := TRANS_TIME (%MD2)]
```

Structured text language

```
%MD100 := TRANS_TIME (%MD2);
```

Example : **%MD100 := TRANS_TIME (%MD2)**
 where %MD2 := 86324873 tenths of a second



Syntax

Operator

```
result :=TRANS_TIME(Duration)
```

Operands

Type	Result	Time period
Indexable double words	%MD	%MD,%KD
Non-indexable double	%QD	%ID,%QD
words		Immediate value, Numeric expr.

result : is in the HMS format.

duration : is in the TIME format.

Notes :

The "duration" parameter (expressed in 1/10 of a second) will be rounded up or down to allow conversion (accuracy to within one second).

- ssssssss.0 to ssssssss.4 rounded to ssssssss.0
- ssssssss.5 to ssssssss.9 rounded to ssssssss.0 + 1.0

The maximum converted time period can reach 10000 hours. This means that if the value of the duration (TIME) provided as a parameter is greater than or equal to 360000000, it cannot be converted. System bit %S15 is set to 1 and the result is 0000:00:00.

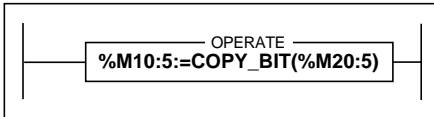
2.10 Bit table instructions

2.10-1 Copy one bit table to another bit table

This function copies one bit table into another bit table bit-wise.

Structure

Ladder language



Instruction list language

```
LD      TRUE
[%M10:5 := COPY_BIT (%M20:5)]
```

Structured text language

```
%M10:5 := COPY_BIT (%M20:5) ;
```

Syntax

Operator

```
result :=COPY_BIT (Tab)
```

Operands

Type	Result	Table (tab)
Bit table	%M:L, %Q:L, %I:L	%M:L, %Q:L, %I:L, %Xi:L

Notes :

- The tables can be of different sizes. In this case, the result table contains the result of the function executed on a length which is equivalent to the smallest table size, and the rest of the result table is not modified.
- Beware of overlapping between the input table and the result table.

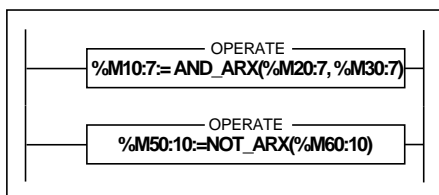
2.10-2 Bit table logic instructions

Associated functions are used to execute a bit-wise logic operation between two bit tables and load the result into another bit table.

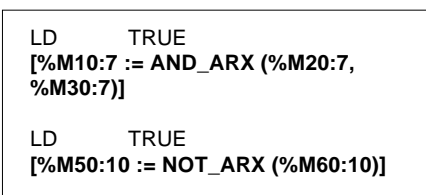
- **AND_ARX** : logic AND (bit-wise).
- **OR_ARX** : logic OR (bit-wise).
- **XOR_ARX** : exclusive OR (bit-wise).
- **NOT_ARX** : logic complement (bit-wise) of a table.

Structure

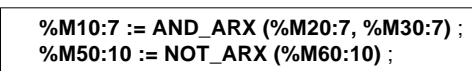
Ladder language



Instruction list language

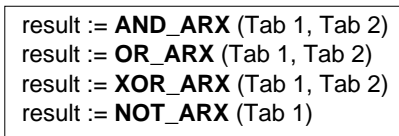


Structured text language



Syntax

Operator



Operands

Type	Result	Table 1 and 2 (tab)
Bit table	%M:L, %Q:L, %I:L	%M:L, %Q:L, %I:L, %Xi:L

Notes :

- The tables can be of different sizes. In this case, the result table contains the result of the function executed on a length which is equivalent to the smallest table size, and the rest of the result table is not modified.
- Beware of overlapping between the input table and the result table.

2.10-3 Copy from a bit table to a word table

This function copies bits from a bit table or part of a bit table to a word table (or double word table).

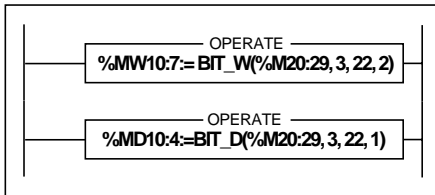
Copying from the bit table is from a certain row (brow) for a number of bits (nbit).

Copying to the word table (or double word table) is from the row (wrow or draw) beginning with the least significant bit of each word.

- **BIT_W** : Copies from a bit table to a word table.
- **BIT_D** : Copies from a bit table to a double word table.

Structure

Ladder language



Instruction list language

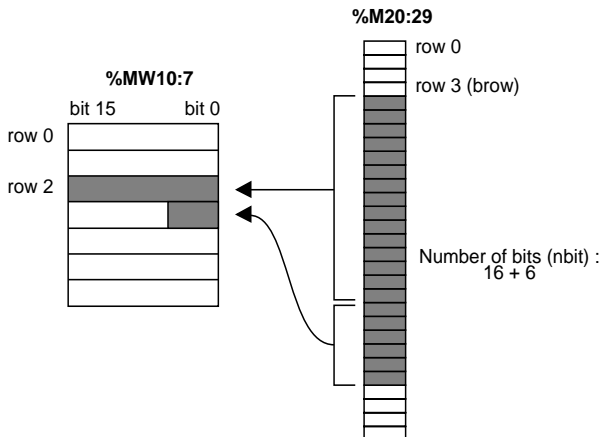
```
LD      TRUE
[%MW10:7 := BIT_W (%M20:29, 3, 22, 2)]

LD      TRUE
[%MD10:4 := BIT_D (%M20:29, 3, 22, 1)]
```

Structured text language

```
%MW10:7 := BIT_W (%M20:29, 3, 22, 2) ;
%MD10:4 := BIT_D (%M20:29, 3, 22, 1) ;
```

Example : `%MW10:7 := BIT_W (%M20:29, 3, 22, 2) ;`



Syntax

Operator

```
result := BIT_W (Tab, brow, nbit, wrow)
result := BIT_D (Tab, brow, nbit, drow)
```

Operands

Type	Result	Table (tab)	brow - nbit wrow or drow
Word tables	%MW:L		
Double word tables	%MD:L		
Bit tables		%M:L, %Q:L, %I:L, %Xi.L	
Indexable words			%MW, %KW
Non-indexable words			%IW, %QW, %SW, %NW, %Xi.T Immediate value Numeric expr.

Notes :

- If the number of bits to be processed is greater than the number of bits remaining in the table from the row (brow), the function copies up to the last element in the table.
- If the number of bits to be copied is greater than the number of bits constituting the words remaining in the result table, the function stops copying at the last element in the word table (or double word table).
- A negative value in the brow, nbit, wrow or drow parameters is interpreted as zero.

2.10-4 Copy from a word table to a bit table

This function copies bits constituting all or part of a word table (or double word table) to a bit table.

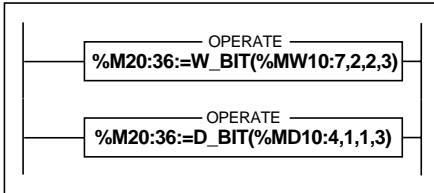
Copying from the word table (or double word table) is from a certain row word (wrow or drow) for a number of words (nwd).

Copying to the bit table is from the row (brow) beginning with the least significant bit of each word.

- **W_BIT** : Copies from a word table to a bit table.
- **D_BIT** : Copies from a double word table to a bit table.

Structure

Ladder language



Instruction list language

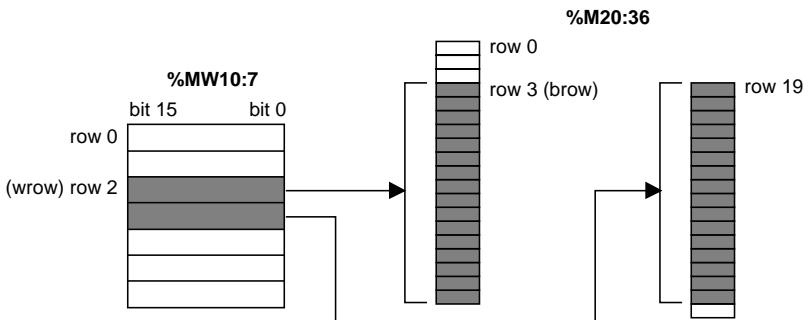
```
LD TRUE
[%M20:36 := W_BIT (%MW10:7, 2, 2, 3)]

LD TRUE
[%M20:36 := D_BIT (%MD10:4, 1, 1, 3)]
```

Structured text language

```
%M20:36 := W_BIT (%MW10:7, 2, 2, 3) ;
%M20:36 := D_BIT (%MD10:4, 1, 1, 3) ;
```

Example : `%M20:36 := W_BIT (%MW10:7, 2, 2, 3) ;`



Syntax

Operator

result := W_BIT (Tab, wrow, nwd, brow) result := D_BIT (Tab, drow, nwd, brow)
--

Operands

Type	Result	Table (tab)	wrow or drow nwd - brow
Bit tables	%M:L,%Q:L,%I:L		
Word tables		%MW:L,%KW:L	
Double word tables		%MD:L,%KD:L	
Indexable words			%MW, %KW
Non-indexable words			%IW, %QW, %SW, %NW, %Xi.T Immediate value Numeric expr.

Notes :

- If the number of bits to be processed is greater than the number of bits remaining in the table from the row (wrow), the function copies up to the last element in the table.
- If the number of bits to be copied is greater than the number of bits remaining in the result table, the function stops copying at the last element in the table.
- A negative value in the brow, nbit, wrow or drow parameters is interpreted as zero.

2.11 "Orphee" functions : shift, counter

2.11-1 Shifts on words with retrieval of shifted bits

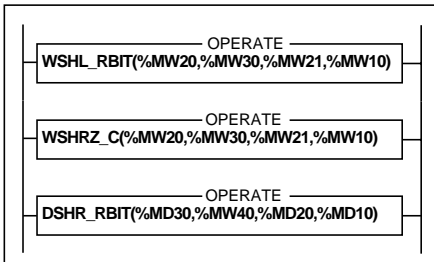
These functions execute arithmetical shifts to the left or right for a number of shifts (nbit) on a word or a double word (a).

After a shift operation, the value is loaded into (result) and the shifted bits are loaded into (rest).

- **WSHL_RBIT** : Shift to left on a word with retrieval of shifted bits.
- **DSHL_RBIT** : Shift to left on a double word with retrieval of shifted bits.
- **WSHRZ_C** : Shift to right on a word with filling of spaces by 0 and retrieval of shifted bits.
- **DSHRZ_C** : Shift to right on a double word with filling of spaces by 0 and retrieval of shifted bits.
- **WSHR_RBIT** : Shift to right on a word with extension of sign and retrieval of shifted bits.
- **DSHR_RBIT** : Shift to right on a double word with extension of sign and retrieval of shifted bits.

Structure

Ladder language



Instruction list language

```
LD      TRUE
[WSHL_RBIT(%MW20,%MW30,%MW21,%MW10)]

LD      TRUE
[WSHRZ_C(%MW20,%MW30,%MW21,%MW10)]

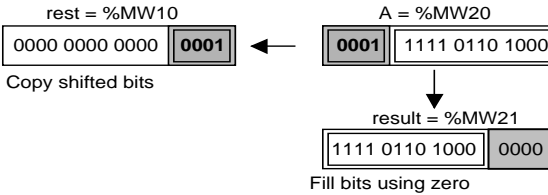
LD      TRUE
[DSHR_RBIT(%MD30,%MW40,%MD20,%MD10)]
```

Structured text language

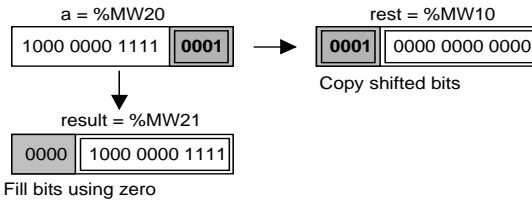
```
WSHL_RBIT (%MW20,%MW30,%MW21,%MW10) ;
WSHRZ_C (%MW20,%MW30,%MW21,%MW10) ;
DSHR_RBIT (%MD30,%MW40,%MD20,%MD10) ;
```

Example :

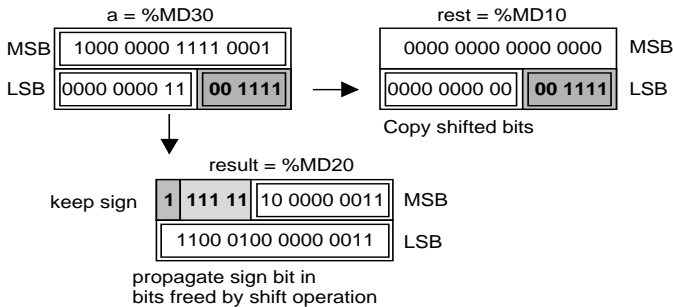
WSHL_RBIT(%MW20,%MW30,%MW21,%MW10) where %MW30 = 4



WSHRZ_C(%MW20,%MW30,%MW21,%MW10) where %MW30 = 4



DSHR_RBIT(%MD30,%MW40,%MD20,%MD10) where %MW40 = 6



Syntax

Operator

WSHL_RBIT (a, nbit, result, rest)
WSHRZ_C (a, nbit, result, rest)
WSHR_RBIT (a, nbit, result, rest)

Operands

Type	a	nbit	result rest
Indexable words	%MW,%KW	%MW,%KW	%MW
Non-indexable words	%IW, %QW, %SW, %NW Immediate value Numeric expr.	%IW, %QW, %SW, %NW, %Xi.T Immediate value Numeric expr.	%QW, %SW,%NW

Syntax

Operator

DSHL_RBIT (a, nbit, result, rest)
DSHRZ_C (a, nbit, result, rest)
DSHR_RBIT (a, nbit, result, rest)

Operands

Type	a	nbit	result rest
Indexable double words	%MD,%KD		%MD
Non-indexable double words	%ID,%QD,%SD Immediate value Numeric expr.		%QD,%SD
Indexable words		%MW, %KW	
Non-indexable words		%IW, %QW, %SW, %NW, %Xi.T Immediate value Numeric expr.	

Notes :

- If the parameter (nbit) is not between 1 and 16 for shifts on words, or between 1 and 32 for shifts on double words, the outputs (result) and (rest) are not significant and system bit %S18 is set to 1.

2.11-2 Up/down counting with indication of over/underflow

This function executes up/down counting with an indication of an over/underflow. The function is only executed when the enable input (en) is at 1.

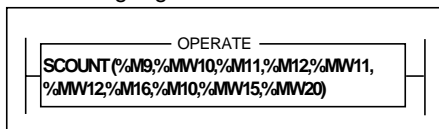
Two independent inputs (cu and cd) are used to upcount and downcount events. Output (Qmin) is set to 1 when the minimum threshold (min) is reached and output (Qmax) is set to 1 when the maximum threshold (max) is reached.

The initial counter value is fixed by parameter (pv) and the current counter value is given by parameter (cv).

A 16-bit word (mwd) is used to store the state of the cu and cd inputs (bit 0 to store cu and bit 1 to store cd).

Structure

Ladder language



Instruction list language

```
LD      TRUE
[SCOUNT(%M9,%MW10,%M11,%M12,%MW11,
%M12,%M16,%M10,%MW15,%MW20)]
```

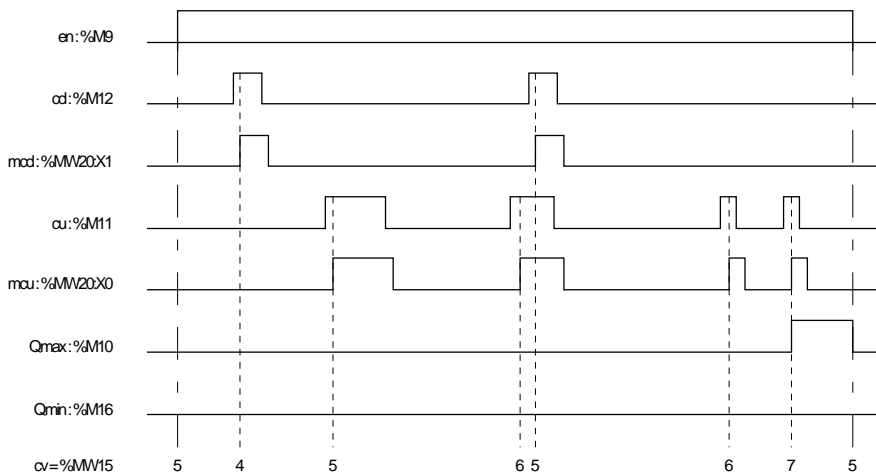
Structured text language

```
SCOUNT(%M9,%MW10,%M11,%M12,%MW11,%MW12,%M16,%M10,%MW15,%MW20) ;
```

Example :

SCOUNT (%M9,%MW10,%M11,%M12,%MW11,%MW12,%M16,%M10,%MW15,%MW20)

where %MW10 (pv) = 5, %MW11 (min) = 0, %MW12 (max) = 7



Syntax

Operator

SCOUNT (en, pv, cu, cd, min, max, Qmin, Qmax, cv, mwd)

Operands

Type	en, cu, cd	Qmin, Qmax	pv, min, max	cv, mwd
Bits	%I,%Q,%M,%S, %BLK,%.:Xk	%I,%Q,%M		
Indexable words			%MW,%KW	%MW
Non-indexable words			%IW, %QW, %SW, %NW, %Xi.T, Immed. val. Numeric expr.	%QW,%SW %NW

Notes :

- If (en) = 0 then the function is no longer enabled and on each call, there is :
 - Qmin = Qmax = 0
 - mcu = mcd = 0
 - cv = pv
- If max > min then :
 - cv ≥ max ---> Qmax = 1 and Qmin = 0
 - min < cv < max ---> Qmax = Qmin = 0
 - cv ≤ min ---> Qmax = 0 and Qmin = 1
- If max < min then :
 - max ≤ cv ≤ min ---> Qmax = 1 and Qmin = 0
 - cv < max ---> Qmax = 0 and Qmin = 1
 - cv > min ---> Qmax = 1 and Qmin = 0
- If max = min then :
 - cv < min and max ---> Qmax = 0 and Qmin = 1
 - cv ≥ min and max ---> Qmax = 1 and Qmin = 0
- Modifying parameter (pv) with (en) at 1 has no effect on operation.
- A negative value for parameters (pv) and (min) is interpreted as a zero value.
- A value less than 1 for parameter (max) is interpreted as 1.

3.1 System bits

3.1-1 List of system bits

Bit	Function	Init. state	Control (1)
%S0	1 = cold start (power return with loss of data)	0	S or U->S
%S1	1 = warm restart (power return no loss of data)	0	S or U->S
%S4,%S5, %S6,%S7	Time base 10ms, 100ms, 1s, 1mn	-	S
%S8	Wiring test (can be used on a non-configured TSX 37 PLC)	1	U
%S9	1 = force PLC outputs into fallback position	0	U
%S10	0 = I/O fault	1	S
%S11	1 = watchdog overflow	0	S
%S13	1 = first scan after setting to RUN	-	S
%S15	1 = character string fault	0	S->U
%S16	0 = task I/O fault	1	S->U
%S17	1 = overflow	0	S->U
%S18	1 = overflow or arithmetic error	0	S->U
%S19	1 = task period overflow	0	S->U
%S20	1 = index overflow	0	S->U
%S21	1 = Grafcet initialization	0	S or U->S
%S22	1 = Grafcet resetting	0	U->S
%S23	1 = Grafcet preposition and freeze	0	U->S
%S26	1 = table overflow (steps/transitions)	0	S
%S30	1 = activation of the master task	1	U
%S31	1 = activation of the fast task	1	U
%S38	1 = enable events	1	U
%S39	1 = saturation of event processing	0	S->U
%S40 to %S47	1 = I/O fault of a TSX 57 rack	1	S
%S49	1 = reset tripped solid state outputs	0	U
%S50	1 = set real-time clock	0	U
%S51	1 = loss of real-time clock time	0	S

(1) See next page.

Bit	Function	Init. state	Control (1)
%S59	1 = enable adjustment of current date	0	U
%S66	1 = battery indicator always off	0	U
%S67	0 = memory cartridge battery operating	-	S
%S68	0 = backup battery (processor) operating	-	S
%S69	1 = enable WORD memory display mode on displays	0	U
%S70	1 = update data on As-i bus or TSX Nano link	0	S->U
%S73 (2)	1 = switch to protected mode on AS-i bus	0	U->S
%S74 (2)	1 = save configuration on AS-i bus	0	U->S
%S80	1 = reset message counters	0	S->U
%S90	1 = update common words	0	S->U
%S96 (2)	0 = application program backup invalid 1 = application program backup valid	0	S->U
%S97 (2)	0 = %MW backup invalid 1 = %MW backup valid	-	S
%S98 (2)	1 = replace TSX SAZ 10 module pushbutton with discrete input	0	U
%S99 (2)	1 = replace centralized display block pushbutton with discrete input	0	U
%S100	Terminal port protocol	-	S

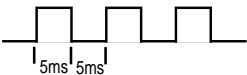
(1) S = controlled by the system, U = controlled by the user, U->S = set to 1 by the user, reset to 0 by the system, S->U = set to 1 by the system, reset to 0 by the user.

(2) only on TSX 37.

3.1-2 Detailed description of system bits

TSX 37 and TSX 57 PLCs have %Si system bits which indicate the status of the PLC or enable the user to intervene in its operation.

These bits can be tested in the user program in order to detect any operating event which requires special processing. Some of them must be reset to their initial or normal state by the program. However, the system bits which have been reset to their initial or normal state by the system must not be reset by the program or the terminal.

System bits	Function	Description
%S0	Cold start	<p>Normally at 0. It is set to 1 by :</p> <ul style="list-style-type: none"> • A power return with loss of data (battery fault). • The user program. • The terminal. • Changing a cartridge. • Pressing the RESET button. <p>This bit is set to 1 during the first complete scan. It is reset to 0 before the next scan. Operation : see part A, section 1.4.</p>
%S1	Warm restart	<p>Normally at 0. It is set to 1 by :</p> <ul style="list-style-type: none"> • A power return with saving of data. • The user program. • The terminal. <p>It is reset to 0 by the system at the end of the first complete scan and before the outputs are updated. Operation : see part A, section 1.4.</p>
%S4	Time base 10ms	<p>Changes in the state of these bits are controlled by an internal clock. They are not synchronized with the PLC scan.</p> <p>Example : %S4</p> 
%S5	100ms	
%S6	1s	
%S7	1min	
%S8	Wiring	<p>Normally at 1, this bit is used to test the wiring when the TSX 37 test PLC is in the "not configured" state.</p> <ul style="list-style-type: none"> • At state 1 the outputs are forced to 0. • At state 0 the outputs can be modified by an adjustment terminal.
%S9	Placing outputs in fallback position	<p>Normally at 0. It can be set to 1 by the program or by the terminal :</p> <ul style="list-style-type: none"> • At state 1 PLC outputs are forced into fallback position. • At state 0 outputs are updated normally.
%S10	I/O fault	<p>Normally at 1. It is set to 0 when an I/O fault is detected on the base or extension PLC (configuration fault, exchange fault, hardware fault). Bit %S10 is reset to 1 when the fault disappears.</p>

System bits	Function	Description
%S11	Watchdog overflow	Normally at 0. It is set to 1 by the system when the execution of a task exceeds the maximum execution time (watchdog) declared during configuration. Watchdog overflow causes the PLC to change to STOP and the application stops in error mode (ERR indicator lamp flashing).
%S13	First scan	Normally at 0. It is set to 1 by the system during the first scan after the PLC has been set to RUN.
%S15	Character string fault	Normally at 0. It is set to 1 when the destination zone of a character string transfer is not sufficiently large to receive that character string. This bit must be reset to 0 by the user.
%S16	Task I/O fault	Normally at 1. It is set to 0 by the system if a fault occurs in an I/O module configured in the task. This bit must be reset to 1 by the user. This task controls its own %S16 bit.
%S17	Output bit on shift or arithmetic carry	Normally at 0. It is set to 1 by the system : <ul style="list-style-type: none"> • During a shift operation. It contains the state of the last bit. • If overflow occurs in a non-signed arithmetic operation (dates). This bit must be reset to 0 by the user.
%S18	Arithmetic overflow or error	Normally at 0. It is set to 1 in the case of overflow during a 16-bit operation, where : <ul style="list-style-type: none"> • Result is greater than + 32767 or less than - 32768 for single length operations. • Result is greater than + 2 147 483 647 or less than - 2 147 483 648 for double length operations. • Result is greater than +3.402824E+38 or less than -3.402824E+38 for floating point operations (software version > 1.0). • Capacity overflow in DCB. • Division by 0. • The square root of a negative number. • Forcing to a non-existent step on a drum controller. • Stacking a full register and unstacking an empty register. It must be tested by the user program after each operation where there is a risk of overflow, then reset to 0 by the user if an overflow occurs.
%S19	Task period overrun (periodic scan)	Normally at 0. It is set to 1 by the system in the event of a scan period overrun (task scan time greater than the period defined by the user during configuration or programmed in word %SW associated with the task). This bit is reset to 0 by the user. Each task controls its own %S19 bit.

System bits	Function	Description
%S20	Index overflow	Normally at 0. It is set to 1 when the address of the indexed object becomes less than 0 or exceeds the number of objects declared during configuration. It must be tested by the user program after each operation where there is a risk of overflow, then reset to 0 by the user if an overflow occurs.
%S21	Grafcet initialization	This bit is controlled by the user to initialize the Grafcet (preferably set to 1 during preprocessing). It is reset to 0 by the system after Grafcet initialization (at the end of preprocessing, during assessment of the new Grafcet state). Grafcet initialization involves deactivating all active steps and activating initial steps. On a cold start, this bit is set to 1 by the system during preprocessing.
%S22	Resetting Grafcet	Normally at 0, this bit can only be set to 1 by the program during preprocessing. At state 1, it causes all Grafcet steps to deactivate. It is reset to 0 by the system after acknowledgment of the end of the preprocessing.
%S23	Grafcet freezing	Normally at 0, setting %S23 to 1 causes the Grafcet state to be maintained. Whatever the value of the transition conditions upstream of the active steps, the Grafcet chart does not change. Freeze is maintained as long as bit %S23 is set to 1. This bit is controlled by the user program. It is set to 1 or 0 only during preprocessing.
%S26	Table overflow (steps/transitions)	Normally at 0, it is set to 1 by the system when activation capacities (steps or transitions) are exceeded or when trying to execute an incorrectly defined chart (such as a destination connector to a step which is not part of the chart). An overflow causes the PLC to STOP. This bit is reset to 0 when the terminal is initialized.
%S30	Master task activation/deactivation	Normally at 1. If it is set to 0 by the user, the master task is deactivated.
%S31	Fast task activation	Normally at 1. If it is set to 0 by the user, the fast task is deactivated.
%S38	Enable/disable events	Normally at 1. If it is set to 0 by the user, events are disabled.
%S39	Saturation of event processing	This bit is set to 1 by the system to show that one or more events cannot be processed due to saturation of the stacks. This bit is reset to 0 by the user.

System bits	Function	Description
%S40 to %S47	I/O fault (racks) (1)	Bits %S40 to %S47 are assigned to racks 0 to 7 respectively. Normally at 1, each of these bits is set to 0 should an I/O fault occur on the corresponding rack. The bit is reset to 1 when the fault has cleared.
%S49	Reactivate outputs	Normally at state 0. This bit can be set to 1 by the user to request a reactivation every 10s from the occurrence of a solid state output fault triggered by an over-current or a short-circuit.
%S50	Updating the date and time using words %SW50 to 53	Normally at 0. This bit can be set to 1 or to 0 by the program or by the terminal. <ul style="list-style-type: none"> • At 0 it accesses the date and time by reading system words %SW50 to 53. • At 1 it updates the date and time by writing system words %SW50 to 53.
%S51	Loss of real-time clock time	This bit, which is managed by the system, signals at 1 either that the real-time clock is missing or that the system words relating to the real-time clock are not significant. In this case, the clock should be set. Setting the time automatically changes the bit to 0.
%S59	Updating the date and time using word %SW59	Normally at 0. This bit can be set to 1 or to 0 by the program or by the terminal. <ul style="list-style-type: none"> • At 0 the system does not control system word %SW59. • At 1 the system controls the rising and falling edges on word %SW59 to adjust the current date and time (in increments).
%S66	Control of battery indicator	Normally at 0. This bit can be set to 1 or to 0 by the program or by the terminal. It is used to switch the battery indicator on or off, if the backup battery is faulty or missing : <ul style="list-style-type: none"> • At 0 the battery indicator lights up if the battery is faulty or missing • At 1 the battery indicator is always off On a cold start, %S66 is reset to 0 by the system.
%S67	State of cartridge battery	This bit is used to check operation of the backup battery for the RAM memory cartridge. <ul style="list-style-type: none"> • At 0 the battery is present and operating. • At 1 the battery is absent or not operating.
%S68	State of processor battery	This bit is used to check operation of the backup battery for program and data in the RAM memory. <ul style="list-style-type: none"> • At 0 the battery is present and operating. • At 1 the battery is absent or not operating.
%S69	Display of user data on PLC displays	Normally at 0. This bit can be set to 1 or to 0 by the program or by the terminal. <ul style="list-style-type: none"> • At 0 the state of the I/O are displayed on the PLC indicator lamps (WRD indicator lamp off). • At 1 user data is displayed (WRD indicator lamp on). (see words %SW67,68 and 69).

(1) only on TSX 57 PLCs.

System bits	Function	Description
%S70	Update data on AS-i bus or TSX Nano link	This bit is set to 1 by the system at the end of each TSX Nano or AS-i bus scan. On power-up, it indicates that all the data has been refreshed at least once and that it is therefore significant. This bit is reset to 0 by the user.
%S73	Switch to protected mode on AS-i bus	Normally at 0. This bit is set to 1 by the user to change to protected mode on AS-i bus. Bit %S74 must first be at 1. This bit is only used in a wiring test, and has no application in the PLC.
%S74	Save configuration present on AS-i bus	Normally at 0. This bit is set to 1 by the user to save the configuration present on the AS-i bus. This bit is only used in a wiring test, and has no application in the PLC.
%S80	Reset message counters	Normally at 0. This bit can be set to 1 by the user in order to reset message counters %SW80 to %SW86.
%S90	Refresh common words	Normally at 0. This bit is set to 1 when common words are received from another station on the network. This bit can be set to 0 by the program or by the terminal to check the exchange cycle of common words.
%S96	Validity of application program backup	0 -> application program backup invalid, 1 -> application program backup valid. This bit can be read at any time (by the program or in adjust mode) and in particular after a cold or warm restart. It is relevant to a Backup application created using PL7 in the internal Flash EPROM.
%S97	Validity of %MW backup	0 -> %MW backup invalid, 1 -> %MW backup valid. This bit can be read at any time (by the program or in adjust mode) and in particular after a cold start or warm restart.
%S98	Locate pushbutton on TSX SAZ 10 module remotely	Normally at 0. This bit is managed by the user : 0 -> pushbutton on TSX SAZ 10 module active, 1 -> pushbutton on TSX SAZ 10 module replaced by a discrete input (see %SW98).
%S99	Locate pushbutton on display block remotely	Normally at 0. This bit is managed by the user : 0 -> pushbutton on centralized display block active, 1 -> pushbutton on centralized display block replaced by a discrete input (see %SW99).
%S100	Terminal port protocol	Set to 0 or 1 by the system depending on the state of the INL/DPT shunt on the terminal port. • If the shunt is absent (%S100=0), UNI-TELWAY master protocol is used. • If the shunt is present (%S100=1), the protocol used is that indicated by the application configuration.

3.2 System words

3.2-1 List of system words

Word	Function	Control
%SW0	Value of master task period (periodic task)	U
%SW1	Value of fast task period	U
%SW8	Control input acquisition of each task	U
%SW9	Control output updating of each task	U
%SW10	First scan after cold restart	S
%SW11	Watchdog time	S
%SW12	UNI-TELWAY terminal port address	S
%SW13	Main address of the station	S
%SW17	Fault status on floating point operation	S and U
%SD18	Absolute time counter	S and U
%SW20	Number of steps active, to activate or to deactivate	S
%SW21	Number of transitions validated, to validate and to devalidate	S
%SW30	Execution time of master task scan	S
%SW31	Maximum time of master task scan	S
%SW32	Minimum time of master task scan	S
%SW33	Execution time of last fast task scan	S
%SW34	Maximum time of fast task scan	S
%SW35	Minimum time of fast task scan	S
%SW48	Number of events processed	S and U
%SW49 (1)	Real-time clock function : words containing the current	S and U
%SW50 (1)	date and time values (in BCD)	
%SW51 (1)	%SW49 =day of the week (type of day)	
%SW52 (1)	%SW50 = seconds %SW51 = hours and minutes	
%SW53 (1)	%SW52 = month and day %SW53= century and year	
%SW54 (1)	Real-time clock function : words containing the date and time of the	S
%SW55 (1)	last power failure or PLC stop (in BCD)	
%SW56 (1)	%SW54 = seconds and fault code %SW55 = hour and minute	
%SW57 (1)	%SW56 = month and day %SW57= century and year	
%SW58	Identification code of last stop and day of the week (type of day)	
%SW59	Incremental adjustment of the current date and time	U
%SW67	Control "Display" mode,	S and U
%SW68	%SW67: read pushbuttons	
%SW69	%SW68: current and maximum indices of "displayed objects"	
%SW69	%SW69: number of the first object in the zone displayed	
%SW80	No of messages transmitted by the system to the terminal port	S and U
%SW81	No of messages received by the system from the terminal port	
%SW82	No of messages transmitted by the system to the PCMCIA card	
%SW83	No of messages received by the system from the PCMCIA card	
%SW84	No of telegrams transmitted by the system	
%SW85	No of telegrams received by the system	
%SW86	No of messages refused by the system	
%SW96 (2)	Control / diagnostics of the save / retrieve application program and %MW function	S and U
%SW97 (2)	Number of %MW to be saved	U

(1) only on TSX 37-21/22 and TSX 57 PLCs.

(2) only on TSX 37.

Word	Function	Control
%SW98 (2)	Module/channel geographical address of the discrete input replacing the pushbutton on the TSX SAZ 10 module	U
%SW99 (2)	Module/channel geographical address of the discrete input replacing the pushbutton on the centralized display block	U
%SW108	No. of bits forced	S
%SW109	Count number of analog channels forced to 0	S
%SW124	Type of last CPU fault found	S
%SW125	Type of blocking fault	S
%SW126	Address of blocking fault instruction	S
%SW127		

S = controlled by the system, U = controlled by the user,
(2) only on TSX 37.

3.2-2 Detailed description of system words

System words	Function	Description
%SW0	Master task scan period	Modifies the master task scan period defined during configuration via the user program or the terminal. The period is expressed in ms (1..255ms). %SW0=0 during cyclic operation. On a cold restart : takes the value defined by configuration.
%SW1	Fast task scan period	Modifies the fast task scan period defined during configuration via the user program or the terminal. The period is expressed in ms (1..255ms). On a cold restart : takes the value defined by configuration.
%SW8	Control input acquisition of tasks	Inhibits the acquisition phase of the inputs for each task. %SW8:X0 1= inhibition in the master task %SW8:X1 1= inhibition in the fast task
%SW9	Control output updating of tasks	Inhibits the updating phase of the outputs for each task. %SW9:X0 1= inhibition in the master task %SW9:X1 1= inhibition in the fast task
%SW10	First scan after a cold restart	If the bit of the current task is at 0, this means that it is performing its first scan after a cold restart. %SW10:X0 : is assigned to the master task, MAST. %SW10:X1 : is assigned to the fast task, FAST.
%SW11	Watchdog time	Reads the watchdog time defined during configuration. It is expressed in ms (10...500ms).
%SW12	UNI-TELWAY terminal port address	UNI-TELWAY terminal port address (in slave mode) defined in configuration and loaded in this word during a cold restart.

System words	Function	Description
%SW13	Main address of the station	Indicates for the main network : <ul style="list-style-type: none"> • The station number (low-order byte) from 0 to 127. • The network number (high-order byte) from 0 to 63. (position of dip switch on the PCMCIA card)
%SW17	Fault status on floating point operation	On detecting a fault in a floating point arithmetic operation, bit %S18 is set to 1 and the %SW17 fault status is updated in line with the following code : <p>%SW17:X0 = Invalid operation/the result is not a number %SW17:X1 = Non-standard operand / the result is correct %SW17:X2 = Division by 0 / the result is $\pm \infty$ %SW17:X3 = Overflow / the result is $\pm \infty$ %SW17:X4 = Underflow / the result is ± 0</p> This word is reset to 0 by the system during a cold restart and by the program for reuse.
%SD18	Absolute time counter	This double word is used to calculate time periods. It is incremented every 1/10th of a second by the system (even if the PLC is in STOP). It can be read and written by the user program or by the terminal.
%SW20	Grafcet activity level	This word contains the number of steps active, to activate and deactivate for the current scan. It is updated by the system every time the Grafcet chart changes.
%SW21	Grafcet transitions validity table	This word contains the number of chart transitions validated, to validate and to devalidate for the current scan. It is updated by the system every time the Grafcet chart changes.
%SW30	Master task scan time (1)	Indicates the scan time of the last master task scan (in ms).
%SW31	Master task maximum scan time (1)	Indicates the longest scan time of the master task since the last cold restart (in ms).
%SW32	Master task minimum scan time (1)	Indicates the shortest scan time of the master task since the last cold restart (in ms).
%SW33	Fast task scan time (1)	Indicates the scan time of the last fast task scan (in ms).

(1) This time corresponds to the time elapsed between the beginning (acquisition of inputs) and end (update of outputs) of a scan cycle. This time includes processing of event-triggered and fast tasks as well as processing of terminal requests.

System words	Function	Description
%SW34	Fast task maximum scan time (1)	Indicates the longest scan time of the fast task since the last cold restart (in ms).
%SW35	Fast task minimum scan time (1)	Indicates the shortest scan time of the fast task since the last cold restart (in ms).
%SW48	Number of events	Indicates the number of events processed since the last cold restart (in ms). This word can be written by the program or by the terminal.
%SW49 %SW50 %SW51 %SW52 %SW53	Real-time clock function (2)	System words containing the current date and time in BCD : %SW49 : day of the week (1 for Monday to 7 for Sunday). %SW50 : Seconds (SS00) %SW51 : Hours and Minutes (HHMM) %SW52 : Month and Day (MMDD) %SW53 : Year (YYYY) These words are controlled by the system when bit %S50 is at 0. These words can be written by the user program or by the terminal when bit %S50 is set to 1.
%SW54 %SW55 %SW56 %SW57 %SW58	Real-time clock function (2)	System words containing the date and time of the last power failure or PLC stop (in BCD) : %SW54 : Seconds (00SS), %SW55 : Hours and Minutes (HHMM), %SW56 : Month and Day (MMDD), %SW57 : Year (YYYY). %SW58 : high order byte containing the day of the week (1 for Monday to 7 for Sunday)
%SW58	Code of last stop	The low order byte contains the code of the last stop. 1= Change from RUN to STOP by terminal 2= Stop on software fault (PLC scan overshoot) 4= Power outage 5= Stop on hardware fault 6= Stop on HALT instruction

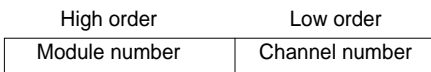
(1) This time corresponds to the time elapsed between the beginning (acquisition of inputs) and end (update of outputs) of a scan cycle. This time includes processing of event-triggered and fast tasks as well as processing of terminal requests.

(2) Only on TSX 37-21/22 and TSX 57 PLCs.

System bits	Function	Description		
%SW59	Adjust current date	Contains two sets of 8 bits to adjust the current date. The operation is always performed on a rising edge of the bit. This word is enabled by bit %S59.		
		Increment		
		Decrement		
		Parameter		
		bit 0	bit 8	Day of the week
		bit 1	bit 9	Seconds
		bit 2	bit 10	Minutes
		bit 3	bit 11	Hours
bit 4	bit 12	Days		
bit 5	bit 13	Months		
bit 6	bit 14	Years		
bit 7	bit 15	Centuries		
%SW66 (1)	Control 7-segment display	Contains the Hexadecimal/BCD value that the user wishes to see displayed on the optional 7-segment display. The display appears when bit %S66 is at 1.		
%SW67 %SW68 %SW69	Management of "WORD" mode	When %S69=1, these words enable the display block (PLC front panel) to be used in WORD mode : <ul style="list-style-type: none"> • %SW67 : control and status of WORD mode. • %SW68 : maximum index and current index. • %SW69 : number of the first object in the zone displayed. For more information on these system words see part F, section 1.5 (installation manual).		
%SW80	Management of messages and telegrams	No of messages sent by the system to the terminal port.		
%SW81		No of messages received by the system from the terminal port.		
%SW82		No of messages sent by the system to the PCMCIA card.		
%SW83		No of messages received by the system from the PCMCIA card.		
%SW84		No of telegrams sent by the system.		
%SW85		No of telegrams received by the system.		
%SW86	No of messages refused by the system.			

(1) Not used in the current version.

System words	Function	Description
%SW96	Control/diagnostics of the save/retrieve function	Control and/or diagnostics of the application program and %MW save/retrieve function : bit 0 : request to transfer to the save zone. This bit is active on a rising edge. It is reset to 0 by the system once the rising edge is taken into account. bit 1 : when this bit is at 1, this signifies that the save function is complete. This bit is reset to 0 by the system once the rising edge is taken into account. bit 2 : report of the save : 0 -> save with no errors, 1 -> error during save. bits 3 to 5 : reserved. bit 6 : validity of the application program backup (identical to %S96). bit 7 : validity of the %MW backup (identical to %S97). bits 8 to 15 : this byte is only significant if the report bit is at 1 (bit 2 = 1, error during save). 1 -> number of %MW to be saved greater than the number of %MW configured, 2 -> number of %MW to be saved greater than 1000 or less than 0, 3 -> number of %MW to be retrieved greater than the number of %MW configured, 4 -> size of the application in the internal RAM greater than 15 Kwords (remember that the %MW are always saved when the application program is saved in the internal Flash EPROM), 5 -> operation prohibited in RUN, 6 -> Backup cartridge present in the PLC, 7 -> writing fault in Flash EPROM..
%SW97	Number of %MW to be saved	Is used to set the parameters for the number of %MW to be saved. When this word is between 1 and 1000, the first 1000 %MW are transferred to the internal Flash EPROM. When this word is 0, only the application program contained in the internal RAM is transferred to the internal Flash EPROM. Any saved %MW are then erased. On a cold restart, this word is initialized to -1 if the internal Flash EPROM does not contain any %MW backup. Otherwise, it is initialized to the value of the number of saved words.
%SW98	Discrete input address	When bit %S98 = 1, this word indicates the geographical address (module / channel) of the discrete input which replaces the pushbutton of the TSX SAZ 10 module :



System words	Function	Description				
%SW99	Discrete input address	When bit %S99 = 1, this word indicates the geographical address (module / channel) of the discrete input which replaces the pushbutton on the centralized display block : <table border="1" style="margin-left: auto; margin-right: auto;"> <tr> <td style="text-align: center;">High order</td> <td style="text-align: center;">Low order</td> </tr> <tr> <td style="text-align: center;">Module number</td> <td style="text-align: center;">Channel number</td> </tr> </table>	High order	Low order	Module number	Channel number
High order	Low order					
Module number	Channel number					
%SW108	No. of forced bits	Indicates the number of forced bits in the application. Normally at 0, it is updated by the bit forcing and unforcing system in the application memory.				
%SW109	Forced analog channel counter	Indicates the number of forced analog channels.				
%SW124	Type of CPU fault	The system writes in this word the last type of CPU fault found (these codes are not changed on a cold restart) : 16#30 : system code fault 16#60 to 64 : battery overload 16#90 : system interrupt fault : IT not anticipated 16#53 : time-out fault during I/O exchanges				
%SW125	Type of blocking fault	The system writes in this word the last type of blocking fault found : 16#DB0 : watchdog overflow 16#2258 : execution of the HALT instruction 16#DEF8 : execution of a JMP instruction to an undefined label 16#2XXX : execution of a CALL instruction to an undefined subroutine 16#0XXX : execution of an unknown function 16#DEFE : Grafcet with source or destination connector undefined 16#DEFF : floating point not implemented 16#DEF0 : division by 0, (1-->%S18) 16#DEF1 : error when transferring a character string (1-->%S15) 16#DEF2 : capacity overflow, (1-->%S18) 16#DEF3 : index overflow (1-->%S20)				
%SW126 %SW127	Address of blocking fault instruction	Instruction address which generated the blocking application fault. %SW126 contains the offset of this address %SW127 contains the base of this address				

4 Differences between PL7-2/3 and PL7 Micro/Junior

4.1 Differences between PL7-2/3 and PL7-Micro/Junior

Immediate values

Objects	PL7-2/3	PL7 Micro/Junior
Base 10 integer	1234	1234
Base 2 integer	L'10011110'	2#10011110
Base 16 integer	H'ABCD'	16#ABCD
Floating point	-1.32e12 (PL7-3)	-1.32e12
Character string	M'aAbBcB'	'aAbBcC'

Labels

Label	PL7-2/3	PL7 Micro/Junior
Label	Li i = 0 to 999	%Li i = 0 to 999

Bits

Objects	PL7-2/3	PL7 Micro/Junior
Input bit in rack	Ixy,i	%I<rack_mod>.<channel>
Indexed input bit in rack	Ixy,i (Wj) (PL7-3)	
Remote input bit	RIx,y,i (PL7-3)	%I<path>\<mod>.<channel>
Indexed remote input bit	RIx,y,i (Wj) (PL7-3)	
Output bit in rack	Oxy,i	%Q<rack_mod>.<channel>
Indexed output bit in rack	Oxy,i (Wj) (PL7-3)	
Remote output bit	ROx,y,i (PL7-3)	%Q<path>\<mod>.<channel>
Indexed remote output bit	ROx,y,i (Wj) (PL7-3)	
I/O fault bit in rack		
• module fault bit	Ixy,S / Oxy,S	%I<rack_mod>.MOD.ERR
• channel fault bit		%I<rack_mod>.<channel>.ERR
Remote I/O fault bit	(PL7-3)	
• module fault bit		%I<path>\<mod>.MOD.ERR
• channel fault bit	RDx,y,i / ERRORx,y,i	%I<path>\<mod>.<channel>.ERR
• output channel trip bit	TRIPx,y,i	
• output channel reset bit	RSTx,y,i	
Internal bit	Bi	%Mi
Indexed internal bit	Bi(Wj) (PL7-3)	%Mi[%MWj]
System bit	SYi	%Si
Step bit	Xi	%Xi
Macro-step bit	XMj (PL7-3)	
Step bit i of macro-step j	Xj,i (PL7-3)	
Input step bit of macro-step j	Xj,I (PL7-3)	
Output step bit of macro-step j	Xj,O (PL7-3)	
Bit j of internal word i	Wi,j	%MWi:Xj
Bit j of indexed internal word i	Wi(Wk),j(PL7-3)	%MWi[%MWk]:Xj

Bit j of constant word i	CWi,j	%KW _i :X _j
Bit j of indexed constant word i	CWi(Wk),j (PL7-3)	%KW _i [%MWk]:X _j
Bit j of register i	I/Ow _{xy} ,i,j	
Bit k of common word j of station i	COM _{i,j,k} COMX _{i,j,k} (X = B, C, D)	%NW _{i,j} :X _k %NXW _{i,j} :X _k
Bit j of system word i	SW _{i,j}	%SW _i :X _j

Words

Objects	PL7-2/3	PL7 Micro/Junior
Single length internal word	Wi	%MW _i
Indexed single length internal word	Wi(Wj) (PL7-3)	%MW _i [%MWj]
Double length internal word	DWi (PL7-3)	%MD _i
Indexed double length internal word	DWi(Wj) (PL7-3)	%MD _i [%MWj]
Real internal word		%MF _i
Indexed real internal word		%MF _i [%MWj]
Single length constant word	CWi	%KW _i
Indexed single length constant word	CWi(Wj)	%KW _i [%MWj]
Double length constant word	CDWi (PL7-3)	%KD _i
Indexed double length constant word	CDWi(Wj) (PL7-3)	%KD _i [%MWj]
Real constant word		%KF _i
Indexed real constant word		%KF _i [%MWj]
Single length input register word	IW _{xy} ,i	%IW<rack_mod>.<channel>
Double length input register word		%ID<rack_mod>.<channel>
Single length output register word	OW _{xy} ,i	%QW<rack_mod>.<channel>
Double length output register word		%QD<rack_mod>.<channel>
Remote input register word	RIW _{x,y,i} (PL7-3)	%IW<path>\<mod>.<channel>
Remote output register word	ROW _{x,y,i} (PL7-3)	%QW<path>\<mod>.<channel>
System word	SW _i	%SW _i
Common word j of station i	COM _{i,j} COMX _{i,j} (where X = B, C, D)	%NW _{{i}j} %NW _{[r.]j} r = network no.
Status word of a remote discrete module	STATUSA _{x,y,i} (PL7-3) STATUSB _{x,y,i} (PL7-3)	
Status word of a remote discrete module channel	STS _{x,y,i} (PL7-3) %IW<path>\<mod>.<channel>.ERR	
Active time of Grafcet steps	X _i ,V	%X _i .T
Active time of step i of macro-step j	X _{j,i} ,V (PL7-3)	

Active time of input step of macro-step j	Xj,I,V (PL7-3)
Active time of output step of macro-step j	Xj,O,V (PL7-3)

Function blocks

Objects	PL7-2/3	PL7 Micro/Junior
Timer	Ti	%Ti
• preset value (word)	Ti,P	%Ti.P
• current value (word)	Ti,V	%Ti.V
• timer running (bit)	Ti,R	%Ti.R
• timer done (bit)	Ti,D	%Ti.D
Monostable	Mi	%MNi
• preset value (word)	Mi,P	%MNi.P
• current value (word)	Mi,V	%MNi.V
• monostable running (bit)	Mi,R	%MNi.R
Up/down counter	Ci	%Ci
• preset value (word)	Ci,P	%Ci.P
• current value (word)	Ci,V	%Ci.V
• upcounting overrun (bit)	Ci,E	%Ci.E
• preset done (bit)	Ci,D	%Ci.D
• downcounting overrun (bit)	Ci,F	%Ci.F
Register	Ri	%Ri
• input word (word)	Ri,I	%Ri.I
• output word (word)	Ri,O	%Ri.O
• register full (bit)	Ri,F	%Ri.F
• register empty (bit)	Ri,E	%Ri.E
Text	TXTi	no text block
Drum controller	Di (PL7-2)	%DRi
• number of active step (word)	Di,S	%DRi.S
• active time of current step (word)	Di,V	%DRi.V
• 16 command bits (word)	Di,Wj	%DRi.Wj
• last step in progress (bit)	Di,F	%DRi.F
Fast Counter / Timer	FC (PL7-2)	-
• preset value (word)	FC,P	-
• current value (word)	FC,V	-
• external reset (bit)	FC,E	-
• preset done (bit)	FC,D	-
• counting in progress (bit)	FC,F	-
Real-time clock	H (PL7-2)	-
• "WEEK" or "YEAR" type		
day selection MTWTFSS (word)	VD	-
• start of active time period (word)	BGN	-
• end of active time period (word)	END	-
• current value < setpoint (bit)	<	-
• current value = setpoint (bit)	=	-
• current value > setpoint (bit)	>	-

Bit and word tables

Objects	PL7-2/3	PL7 Micro/Junior
Bit strings		
• Internal bit string	Bi[L]	%Mi:L
• Input bit string	Ixy,i[L] (PL7-3)	%Ixy.i:L
• Output bit string	Oxy,i[L] (PL7-3)	%Qxy.i:L
• Grafacet step bit string	Xi[L] (PL7-3)	%Xi:L
• macro-step bit string	XMi[L] (PL7-3)	
Character strings		%MBi:L (1) (where i is even)
Word tables		
• internal word table	Wi[L]	%MWi:L
• indexed internal word table	Wi(Wj)[L]	%MWi[%MWj]:L
• internal double word table	DWi[L] (PL7-3)	%MDi:L
• indexed internal double word table	DWi(Wj)[L] (PL7-3)	%MDi[%MWj]:L
• constant word table	CWi[L]	%KWi:L
• indexed constant word table	CWi(Wj)[L]	%KWi[%MWj]:L
• constant double word table	CDWi[L] (PL7-3)	%KDi:L
• indexed constant double word table	CDWi(Wj)[L] (PL7-3)	%KDi[%MWj]:L
• real table		%MFi:L
• indexed real table		%MFi[%MWj]:L
• constant real table		%KFi:L
• indexed constant real table		%KFi[%MWj]:L
• remote input element table	Rlx,y,i[L] (PL7-3)	
• remote output element table	ROx,y,i[L] (PL7-3)	
• remote input indexed element table	Rlx,y,i(Wj)[L] (PL7-3)	
• remote output indexed element table	ROx,y,i(Wj)[L] (PL7-3)	

Optional function blocks

Objects	PL7-3	PL7 Micro/Junior
Optional function block	<OFB>i	
OFB element	<OFB>i, <element>	
Indexed OFB element	<OFB>i,<element>(Wj)	
OFB element table	<OFB>i,<element>[L]	
Indexed OFB element table	<OFB>i,<element>(Wj)[L]	

Instructions

Objects	PL7-2	PL7-3	PL7 Micro/Junior
Instructions on bits			
• Reverse logic		NOT	NOT
• AND	AND	•	AND
• OR	OR	+	OR
• Exclusive OR	XOR		XOR
• Rising edge		RE	RE
• Falling edge		FE	FE
• Set to 1		SET	SET
• Reset to 0		RESET	RESET
Instructions on words and double words			
• Addition	+	+	+
• Subtraction	-	-	-
• Multiplication	*	*	*
• Division	/	/	/
• Comparisons	>, >=, <, <=, =, <>		>, >=, <, <=, =, <>
• Division remainder	MOD	REM	REM
• Square root		SQRT	SQRT
• Absolute value			ABS
• Logic AND	AND	AND	AND
• Logic OR	OR	OR	OR
• Exclusive logic OR	XOR	XOR	XOR
• Logic complement	CPL	CPL	NOT
• Incrementation		INC	INC
• Decrementation		DEC	DEC
• Logic shift to left		SHL	SHL
• Logic shift to right		SHR	SHR
• Circular shift to left	SLC	SLC	ROL
• Circular shift to right	SRC	SRC	ROR
Floating point instructions (1)			
• Addition		ADDF	+
• Subtraction		SUBF	-
• Multiplication		MULF	*
• Division		DIVF	/
• Square root		SQRTF	SQRT
• Absolute value			ABS
• Equality test		EQUF	=
• Strict superiority test		SUPF	>
• Strict inferiority test		INFF	<
• Other tests			>=, <=, <>

Instructions (continued)

Objects	PL7-2	PL7-3	PL7 Micro/Junior
Instructions on byte strings			
• Circular shift		SLCWORD	
Conversion instructions			
• BCD to binary conversion	BCD	DTB	BCD_TO_INT
• Binary to BCD conversion	BIN	BTD	INT_TO_BCD
• ASCII to binary conversion	ATB	ATB	STRING_TO_INT or STRING_TO_DINT
• Binary to ASCII conversion	BTA	BTA	INT_TO_STRING or DINT_TO_STRING
• Gray to binary conversion		GTB	GRAY_TO_INT
• Floating point to integer conversion		FTB	REAL_TO_INT or REAL_TO_DINT
• Integer to floating point conversion		FTF	INT_TO_REAL or DINT_TO_REAL
• BCD to floating point conversion		DTF	BCD_TO_REAL
• Floating point to BCD conversion		FTD	REAL_TO_BCD
• ASCII to floating point conversion		ATF	STRING_TO_REAL
• Floating point to ASCII conversion		FTA	REAL_TO_STRING
Instructions on tables			
• Arithmetic operations		+, -, *, /, REM	+, -, *, /, REM
• Logic operations		AND, OR, XOR	AND, OR, XOR, NOT
• Addition of words in a table		+	SUM
• Search for 1st different word		EQUAL	EQUAL
• Search for 1st equal word		SEARCH	FIND_EQU
Instructions on program			
• Jump		JUMP Li	JUMP %Li
• Call for sub-routine			CALL SRi SRi
• Return from sub-routine		RET	RETURN
• Stop application		HALT	HALT
• Conditional phrase		IF/THEN/ELSE	IF/THEN/ELSE/END_IF
• Iterative phrase		WHILE/DO	WHILE/DO/END_WHILE
Instructions on interruptions			
• Test		READINT	
• Masking		MASKINT	MASKEVT
• Unmasking		DMASKINT	UNMASKEVT
• Acknowledgment		ACKINT	
• Generation of an IT to module		SETIT	
Explicit I/O instructions			
• Read discrete inputs		READBIT	
• Write discrete outputs		WRITEBIT	
• Read registers		READREG	
• Write registers		WRITEREG	
• Read words		READEXT	
• Write words		WRITEEXT	

Instructions (continued)

Objects	PL7-3	PL7 Micro/Junior
Instructions on functions blocks		
• Preset	PRESET Ti / Ci	PRESET %Ti / %Ci
• Start	START Ti / Mi	START %Ti / %MNi
• Activate task	START CTRLi	
• Reset	RESET Ci / Ri / TXTi	RESET %Ci / %Ri
• Deactivate task	RESET CTRLi	
• Upcounting	UP Ci	UP %Ci
• Downcounting	DOWN Ci	DOWN %Ci
• Store in a register	PUT Ri	PUT %Ri
• Retrieve from a register	GET Ri	GET %Ri
• Receive a message	INPUT TXTi	
• Transmit a message	OUTPUT TXTi	
• Transmit/Receive message	EXCHG TXTi	
• Execute an OFB	EXEC <OFBi>	
• Read telegrams	READTLG	

Delimiters

Objects	PL7-2/3	PL7 Micro/Junior
Assignment	->	:=
Left parenthesis for indexing	([
Right parenthesis for indexing)]
Length of table	[length]	:length

5.1 Reserved words

The following reserved words should not be used as symbols.

TO * = Letter

SRI	AUX	CU
AUXi	BCD_TO_INT	D
EVTi	BIT_D	DATE
XMi	BIT_W	DATE_AND_TIME
i = entier	BLK	DAT_FMT
	BLOCK	DAY_OF_WEEK
	BODY	DA_TYPE
	BOOL	DEACTIVATE_PULSE
ABS	BOTTOM	DEC
ACCEPT	BTI	DELETE
ACOS	BTR	DELTA_D
ACTION	BY	DELTA_DT
ACTIVATE_PULSE	BYTE	DELTA_TOD
ACTIVE_TIME	C	DINT
ADD	CAL	DINT_TO_REAL
ADDRESS	CALC	DINT_TO_STRING
ADD_DT	CALCN	DISPLAY_ALRM
ADD_TOD	CALL	DISPLAY_GRP
ADR	CALL_COIL	DISPLAY_MSG
AND	CANCEL	DIV
ANDF	CASE	DMOVE
ANDN	CD	DO
ANDR	CHART	DOWN
AND_ARX	CH_M	DRUM
ANY	CLK	DS
ANY_BIT	CLOSE	DSHL_RBIT
ANY_DATE	CLOSED_CONTACT	DSHRZ_C
ANY_INT	COIL	DSHR_RBIT
ANY_NUM	COMMAND	DSORT_ARC
ANY_REAL	COMMENTS	DSORT_ARW
ARRAY	COMP4	DT
AR_D	COMPCH	DTS
AR_F	CONCAT	DWORD
AR_W	CONF	D_BIT
AR_X	CONFIGURATION	E
ASIN	CONSTANT	EBOOL
ASK	CONTROL_LEDS	ELSE
ASK_MSG	COPY_BIT	ELSIF
ASK_VALUE	COS	EMPTY
ASSIGN_KEYS	CTD	EMPTY_LINE
AT	CTU	END
ATAN	CTUD	ENDC

ENDCN	FIND_LTD	JMPC
END_ACTION	FIND_LTW	JMPCN
END_BLK	FOR	JUMP
END_BLOCK	FROM	JUMP_COIL
END_CASE	FUNC	L
END_COMMENTS	FUNCTION	LAD
END_CONFIGURATION	FUNCTION_BLOCK	LANGAGE
END_FOR	F_B	LANGUAGE
END_FUNCTION	F_EDGE	LD
END_FUNCTION_BLOCK	F_TRIG	LDF
END_IF	GE	LDN
END_MACRO_STEP	GET	LDR
END_PAGE	GET_MSG	LE
END_PHRASE	GET_VALUE	LEFT
END_PROG	GLOBAL_COMMENT	LEN
END_PROGRAM	GR7	LIFO
END_REPEAT	GRAY_TO_INT	LIMIT
END_RESOURCE	GT	LINT
END_RUNG	GTI	LIST
END_STEP	H	LIT
END_STRUCT	HALT	LN
END_TRANSITION	HALT_COIL	LOCATION
END_TYPE	HASH_COIL	LOG
END_VAR	H_COMPARE	REAL
END_WHILE	H_LINK	LT
EQ	I	LWORD
EQUAL	IF	M
ERR	IL	MACRO_STEP
EVT	IN	MAIN
EXCHG	INC	MASKEVT
EXCH_DATA	INCJUMP	MAST
EXIT	INDEX_CH	MAX
EXP	INFO	MAX_ARD
EXPT	INITIAL_STEP	MAX_ARW
F	INIT_BUTTONS	MAX_PAGES
FALSE	INPUT	MAX_STEP
FAST	INPUT_CHAR	MCR
FBD	INSERT	MCR_COIL
FE	INT	MCS
FIFO	INTERVAL	MCS_COIL
FIND	INT_TO_BCD	MID
FIND_EQ	INT_TO_REAL	MIN
FIND_EQD	INT_TO_STRING	MIN_ARD
FIND_EQW	ITB	MIN_ARW
FIND_GTD	ITS	MOD
FIND_GTW	JMP	MONO

MOVE	ON	QUERY
MPP	OPEN	R
MPS	OPEN_CONTACT	R1
MRD	OPERATE	RCV_TLG
MS	OR	RE
MUL	ORF	READ
MUX	ORN	READ_EVT_UTW
M_CH	ORR	READ_ONLY
M_MACRO_STEP	OR_ARX	READ_PARAM
N	OTHERS	READ_STS
N1	OUT	READ_VAR
NAME	OUTIN_CHAR	READ_WRITE
NB_ACTIVE_STEPS	OUTPUT	REAL
NB_ACTIVE_TIME	OUT_BLK	REAL_TO_DINT
NB_BLOCKS	P	REAL_TO_INT
NB_COMMON_WORDS	P0	REAL_TO_STRING
NB_CONSTANT_WORDS	P1	REG
NB_CPT	PAGE	REM
NB_DRUM	PAGE_COMMENT	REPEAT
NB_INTERNAL_BITS	PANEL_CMD	REPLACE
NB_INTERNAL_WORDS	PERIOD	RESET
NB_MACRO_STEPS	PHRASE	RESET_COIL
NB_MONO	PHRASE_COMMENT	RESOURCE
NB_PAGES	PID	RESTORE_PARAM
NB_REG	PID_MMI	RET
NB_TIMER	PLC	RETAIN
NB_TM	POST	RETC
NB_TRANSITIONS	PRESET	RETCN
NE	PRINT	RETURN
NIL	PRINT_CHAR	RET_COIL
NO	PRI00	RIGHT
NON_STORED	PRI01	ROL
NOP	PRIORITY	ROL_ARD
NOT	PRL	ROL_ARW
NOT_ARX	PROG	ROR
NOT_COIL	PROGRAM	ROR_ARD
NOT_READABLE	PROG_LANGAGE	ROR_ARW
NO_GR7	PROG_LANGUAGE	RRTC
NO_PERIOD	PT	RS
N_CONTACT	PTC	RTB
O	PUT	RTC
OCCUR	PV	RTS
OCCUR_ARD	PWM	RUNG
OCCUR_ARW	P_CONTACT	R_EDGE
OF	Q	R_TRIG

S	STOP	UP
S1	STR	USINT
SAVE_PARAM	STRING	USORT_ARD
SCHEDULE	STRING_TO_DINT	USORT_ARW
SD	STRING_TO_INT	UTIN_CHAR
SEARCH	STRING_TO_REAL	VAR
SEL	STRUCT	VAR_ACCESS
SEMA	SUB	VAR_EXTERNAL
SEND	SUB_DT	VAR_GLOBAL
SENDER	SUB_TOD	VAR_INPUT
SEND_ALARM	SUM	VAR_IN_OUT
SEND_MBX_ALARM	SU_TYPE	VAR_OUTPUT
SEND_MBX_MSG	S_T_AND_LINK	VERSION
SEND_MSG	S_T_OR_LINK	V_COMPARE
SEND_REQ	T	V_LINK
SEND_TLG	TAN	W
SERVO	TASK	WHILE
SET	TASKS	WITH
SET_COIL	THEN	WORD
SFC	TIME	WRITE
SHIFT	TIMER	WRITE_CMD
SHL	TIME_OF_DAY	WRITE_PARAM
SHOW_ALARM	TMAX	WRITE_VAR
SHOW_MSG	TMOVE	WRTC
SHOW_PAGE	TO	WSHL_RBIT
SHR	TOD	WSHRZ_C
SHRZ	TOF	WSHR_RBIT
SIN	TOFF	W_BIT
SINGLE	TON	XM
SINT	TOP	XM_MONO
SL	TP	XM_MULTI
SLCWORD	TRANSITION	XOR
SMOVE	TRANS_TIME	XORF
SOFT_CONFIGURATION	TRUE	XORN
SORT	TRUNC	XORR
SORT_ARD	TYPE	XOR_ARX
SORT_ARW	TYPES	YES
SQRT	T_S_AND_LINK	
SR	T_S_OR_LINK	
ST	U	
STANDARD	UDINT	
START	UINT	
STD	ULINT	
STEP	UNMASKEVT	
STI	UNTIL	
STN		

6.1 Conformity to the IEC 1131-3 standard

IEC standard 1131-3 "PLCs - Part 3: Programming languages" defines the syntax and semantics of the software elements used to program PLCs.

This standard describes 2 textual languages, IL (Instruction List) and ST (Structured Text), 2 graphic languages, LD (Ladder Diagram) and FBD (Function Block Diagram) and a graphic chart, SFC (Sequential Function Chart), used to structure the internal organization of a programmed sequence.

PL7 Junior Windows programming software is used to program a PLC conforming to the IEC standard: PL7 Junior implements a subset of the language elements defined in the standard and defines extensions which are authorized within this standard.

IEC standard 1131-3 does not define the rules of interaction of software supplied by a manufacturer claiming to conform to the standard, leaving a greater flexibility of presentation and programming element entry for the comfort of the user.

The elements of the standard implemented in PL7 Junior, the specific implementation information and the cases of detected errors are summarized in the conformity tables below.

Note: Grafcet language is very similar to the 1131-3 SFC language, however not all of its features satisfy conformity requirements: 41, 43 and 45 in table 48. These features are shown in gray in the conformity tables.

6.1.1 Conformity tables

This system conforms to the recommendations of IEC 1131-3, as far as the following language characteristics are concerned:

Common elements

Table n°	Feature n°	Description of features
1	1	Required character set see paragraph 2.1.1 of 1131-3
1	2	Lower case characters

B

1	3a	Number sign (#)
1	4a	Dollar sign (\$)
1	5a	Vertical bar ()
1	6a	Subscript delimiters: Left and right brackets "[]"
2	1	Upper case and numbers
2	2	Upper and lower case, numbers, embedded underlines
3	1	Comments
4	1	Integer literals (Note 1)
4	2	Real literals (Note 1)
4	3	Real literals with exponents
4	4	Base 2 literals (Note 1)
4	6	Base 16 literals (Note 1)
4	7	Boolean Zero and One
4	8	Boolean TRUE and FALSE
5	1	Character string literal features
6	2	\$\$ Dollar sign
6	3	\$' Single quote
6	4	\$L or \$I Line feed
6	5	\$N or \$n New line
6	6	\$P or \$p Form feed (page)
6	7	\$R or \$r Carriage return

6	8	\$T or \$t Tab
7	1a	Duration literals with short prefix t# (Note 2)
10	1	BOOL -1 bit-
10	10	REAL -32 bits-
10	12	TIME -32 bits- (Note 3)
10	13	DATE -32 bits- (Note 3)
10	14	TIME_OF_DAY -32 bits- (Note 3)
10	15	DATE_AND_TIME -64 bits- (Note 3)
10	16	STRING
10	17	BYTE -8 bits-
10	18	WORD -16 bits-
10	19	DWORD -32 bits-
15	1	I prefix for Input location
	2	Q prefix for Output location
	3	M prefix for Memory location
15	4	X prefix, single bit size
	5	None prefix, single bit size
	6	B prefix, byte size (8 bits)
	7	W prefix, word size (16 bits)
	8	D prefix, double word size (32 bits)

16	VAR_EXTERNAL VAR_GLOBAL CONSTANT AT	Keywords (Note 4)
21	1	The PL7 overloaded functions are as follows: ABS, EQUAL, ROL, ROR, SHL, SHR, SQRT, SUM
21	2	In general, the PL7 functions belong to this category (typed functions).
22	1	Type conversion functions: DINT_TO_STRING, INT_TO_STRING, STRING_TO_DINT, STRING_TO_INT, DATE_TO_STRING, DT_TO_STRING, TIME_TO_STRING, TOD_TO_STRING, REAL_TO_STRING, STRING_TO_REAL, REAL_TO_INT, REAL_TO_DINT, INT_TO_REAL, DINT_TO_REAL (Note 5)
22	3	Conversion function BCD_TO_INT (Note 6)
22	4	Conversion function INT_TO_BCD (Note 6)
23	1	ABS function: absolute value
23	2	SQRT function: square root
25	1	SHL function: shift left
25	2	SHR function: shift right
25	3	ROR function: rotate right
25	4	ROL function: rotate left
29	1	LEN function: string length
29	2	LEFT function: leftmost n characters

29	3	RIGHT function: rightmost n characters
29	4	MID function: n characters from a given position
29	5	CONCAT function: extensible concatenation (Note 7)
29	6	INSERT function: insert one string into another
29	7	DELETE function: delete characters
29	8	REPLACE function: replace characters
29	9	FIND function: find one string inside another
32	Input read Input write Output read Output write	(Note 8)
33	1	RETAIN qualifier on internal variables of the predefined function blocks (Note 9)
33	2	RETAIN qualifier on outputs of the predefined function blocks (Note 9)
37	1	Pulse timer: TP (Note 10)
37	2a	On-delay timer: TON (Note 10)
37	3a	Off delay timer: TOF (Note 10)
38	timing diagrams	TP, TON, TOF
39	19	Use of directly represented variables (address)
40	1	Step, graphical form Note: A step number replaces a step identifier

B

40	2	Step, textual form used in the source form of Grafcet only
42	2l	Declarations of actions in LD language Note: PL7 Junior does not implement action blocks
46	1	Single sequence, alternation step/transition
46	2c	"Or" divergence: user ensures that the transition conditions are mutually exclusive
46	3	"Or" convergence
46	4	"And" divergence "And" convergence
46	5c	Sequence jump in an "or" divergence
46	6c	Sequence loop: return to a previous step
46	7	Directional arrows Note: Directional arrows point up and down
48	40	Graphic representation
	42	Note: Grafcet is very similar to the 1131-3 SFC language, but cannot however claim to conform (features 41, 43, 45 are missing)
	46	
	57	
50	5b	Preemptive scheduling with the multi-tasks model

Note 1: The underline characters () inserted between the digits of a numerical literal are not accepted

Note 2: These literals are only visible in the application source, to show the time of configured tasks.

Note 3: These types of data are not yet implemented in a manner visible to the user. This table defines, however, the memory occupation of their internal representation.

Note 4: These key words are only used in the sources generated by PL7 and by the PL7-2 application conversion tool.

Note 5: Effects of conversions to limits:

DINT_TO_STRING: If the string accepting the result is less than 13 characters, truncation occurs and %S15 is set.

INT_TO_STRING: If the string accepting the result is less than 7 characters truncation occurs and %S15 is set.

STRING_TO_DINT and STRING_TO_INT: If the string cannot be converted to an integer, the result is indeterminate and %S18 is set.

DATE_TO_STRING: If the string accepting the result is less than 11 characters truncation occurs and %S15 is set.

DT_TO_STRING: If the string accepting the result is less than 20 characters truncation occurs and %S15 is set.

TIME_TO_STRING: If the string accepting the result is less than 15 characters truncation occurs and %S15 is set.

TOD_TO_STRING: If the string accepting the result is less than 9 characters truncation occurs and %S15 is set.

REAL_TO_STRING: If the string accepting the result is less than 15 characters truncation occurs and %S15 is set.

STRING_TO_REAL: If the string cannot be converted to a real value, the value of the result is "1.#NAN" (16#FFC0_0000) and %S18 is set.

REAL_TO_INT: If the real cannot be converted within the limits [-32768, +32767], the value of the result is -32768 and %S18 and %SW17:X0 are set.

REAL_TO_DINT: If the real cannot be converted within the limits[-2147483648, +2147483647], the value of the result is -2147483648 and %S18 and %SW17:X0 are set.

INT_TO_REAL: Conversion is always possible.

DINT_TO_REAL: Conversion is always possible.

Note 6: As the type INT is not formally implemented, even though it is still used, these functions enable the coding format of a WORD to be changed.

Note 7: Limit the CONCAT function to the concatenation of 2 strings.

Note 8: This paragraph applies to the predefined PL7 function blocks.

Note 9: The RETAIN qualifier is implicit.

Note 10: The timers TP, TON, TOF respect the timing diagrams of table 38, but have a different I/O interface from that of 1131-3.

IL language elements

Table n°	Feature n°	Description of features
51	Instruction fields	Label, operator, operand, comment
52	1	LD
52	2	ST
52	3	S and R
52	4	AND
	6	OR
	7	XOR
52	18	JMP
52	20	RET
52	21)
54	11	IN (Note 11)
54	12	IN (Note 11)
54	13	IN (Note 11)

Note 11: The operator PT is not implemented.

ST language elements (Note 12)

Table n°	Feature n°	Description of features
55	1	Place in parentheses
55	2	Function evaluation
55	5	NOT

55	6	* Multiply
	7	/ Divide
55	9	+ Add
	10	- Subtract
55	11	<, >, <=, >= Comparison
55	12	= Equality
55	13	<> Inequality
55	15	Boolean AND
55	16	XOR Boolean Exclusive or
55	17	Boolean OR
56	1	:= Assignment
56	3	RETURN structure
56	4	IF structure "if... then... elsif... then... else... end_if"
56	6	FOR structure "for... to... do... end_for" (Note 13)
56	7	WHILE structure "while... do... end_while"
56	8	REPEAT structure "repeat ... until... end_repeat"
56	9	EXIT structure

Note 12: This language is used entirely in ST modules. An ST subset is also used in the OPERATE and COMPARE blocks of IL and LD languages.

Note 13: Implementation of the FOR loop with an implicit step of 1 (by 1).

Common graphic elements

Table n°	Feature n°	Description of features
57	2	Graphic horizontal lines
57	4	Graphic vertical lines
57	6	Graphic horizontal/vertical line connection
57	8	Graphic crossing of lines without connection
57	10	Graphic connected and non-connected corners
57	12	Blocks with graphic connected lines
58	2	Unconditional jump: LD language
58	4	Conditional jump: LD language
58	5	Conditional return: LD language
58	8	Unconditional return: LD language

LD language elements

Table n°	Feature n°	Description of features
59	1	Left power rail
59	2	Right power rail
60	1	Horizontal link
60	2	Vertical link
61	1	Open contact
61	3	Closed contact
61	5	Positive transition-sensing contact
61	7	Negative transition-sensing contact

62	1	Coil
62	2	Negated coil
62	3	SET (latch) coil
62	4	RESET (unlatch) coil

Implementation-dependent parameters

Parameter	PL7 limitations and behavior
Procedure for processing errors	Numerous errors are indicated on execution with system bits and words.
National characters used	ÀÁÂÃÄÅÆÇÈÉÊËÌÍÎÏÐÑÒÓÔÕÖ ØÙÚÛÜÝÞàáâãäåæçèéêëìíîïð óôõö÷ùúûüýþÿ #, \$,
Maximum length of identifiers	32
Maximum length of comment	222
Range of values of duration	(Note 14)
Range of values for variables of type TIME	(Note 14)
Precision of representation of seconds in types TIME_OF_DAY and DATE_AND_TIME	(Note 15)
Maximum number of array subscripts	1 (Note 16)
Maximum number of array	Depending on the indexed area (Note 16)
Default maximum length of STRING variables	Not applicable
Maximum permitted length of STRING variables	255
Maximum number of hierarchical levels	3
Logical or physical mapping	Logical mapping
Maximum range of subscript values	Depending on the indexed area (Note 16)

Initialization of system inputs	Initialized to zero by the system.
Effects of type conversions on accuracy	See table 22, feature 1
Precision on time elapsed associated with a step	100ms
Maximum number of steps per chart	96 on PLC 3710 v1.5 128 on PLCs 3720 v1.5, 5710 and 5720
Maximum number of transitions per chart and per step	1024 transitions per chart 11 transitions per step
Action control mechanism	P0, P1 and N1 qualifiers
Maximum number of action blocks per step	3 actions are possible: on activation (P1), continuous (N1) and on deactivation (P0)
Graphic indication of step status	Active step in reverse video
Transition clearing time (deactivation of upstream steps and activation of downstream steps)	Clearing time is variable and is never zero
Depth of divergent and convergent constructions	Limited by the entry grid
List of PLCs which can be programmed by PL7 Junior	TSX 3710, 3720, 5710, 5720
Maximum number of tasks	2 periodic tasks 8 event-triggered tasks
Task interval resolution	From 1 ms to 255 ms
Preemptive or non-preemptive scheduling	Preemptive scheduling

Maximum length of an expression	Variable
Partial evaluation of Boolean expressions	No
Maximum length of control structures in ST	Variable
Value of control variable after execution of a FOR loop	The value of the control variable equals the value of the limit + 1 (since the step is 1)
Graphic/semi-graphic representation	Graphic representation
Restrictions on network topology	An LD network can occupy a maximum of 11 columns and 7 lines

Note 14: These types of data are not yet implemented in a manner visible to the user. This table, however, defines their ranges of values in IEC 1131-3 format.

TIME: from T#0 to T#429496729.5s

TIME_OF_DAY: from TOD#0:0:0 to TOD#23:59:59

DATE_AND_TIME: from DT#1990-01-01:0:0:0 to DT#2099-12-31:23:59:59

DATE: from D#1990-01-01 to D#2099-12-31

Note 15: Rounding is performed as follows: x.0 s to x.4 s, are rounded to x s and x.5 s to x.9 s are rounded to x+1 s.

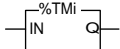
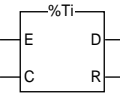
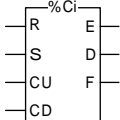
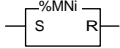
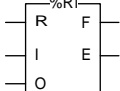

Note 16: It is possible to index all types of directly represented variables positively and negatively within the limit of their respective maximum number defined in configuration.

Error conditions	
Error conditions	PL7 limitations and behavior
Type conversion errors	Indicated during the with a system bit: see table Common elements: table 22, feature 1
Numerical result exceeds range for data type	Indicated during the execution with system bit %S18
Invalid character position specified	Indicated during the execution with system bit %S18
Result exceeds maximum string length	Indicated during the execution with system bit %S15
Overflow errors during evaluation of a transition	Detected during programming
Execution deadlines not met	Indicated on execution with system bit %S19
Other task scheduling conflicts	Detected during configuration
Division by zero	Detected during programming if possible, otherwise indicated on execution with system bit %S18
Type of data invalid for an operation	
Failure of a FOR or WHILE iteration to finish	The PLC goes to watchdog overflow fault and the CPU involved is indicated

7.1 Quick reference guide

Boolean instructions	LD	IL
Accumulator or rung initialization		LD TRUE
Test (read) direct, negated, rising edge, falling edge		LD LDN LDR LDF
AND logic		AND ANDN ANDR ANDF AND(AND(N AND(R AND(F
OR logic direct, negated, rising edge, falling edge		OR ORN ORR ORF OR(OR(N OR(R OR(F
Inversion		N
Exclusive OR logic (direct, negated, rising edge, falling edge)		XOR XORN XORR XORF
Write (direct, negated)		ST STN
Set to 1 Set to 0		S R
Operation block (for contents see following pages)		[action]
Horizontal comparison block (for contents see following pages)		LD [comparison] AND [comparison] AND([comparison] OR [comparison] OR([comparison] XOR [comparison]
Vertical comparison block		
PuSh memory ReaD memory PoP memory		MPS MRD MPP

Boolean instructions	ST
Assignment	: =
Boolean OR	OR
Boolean AND	AND
Boolean Exclusive OR	XOR
Negation	NOT
Rising, falling edge	RE, FE
Set to 1, set to 0	SET, RESET

Function blocks	LD	IL
IEC timer		IN BLK . . END_BLK structure
PL7-3 timer		
Up/down counter		R S CU CD BLK . . END_BLK structure
Monostable		S BLK . . END_BLK structure
Register		R I O BLK . . END_BLK structure
Drum		R U BLK . . END_BLK structure

Function blocks	ST
IEC timer	START %Tmi DOWN %Tmi
PL7-3 timer	PRESET %Ti START %Ti STOP %Ti
Up/down counter	RESET %Ci PRESET %Ci UP %Ci, DOWN %Ci
Monostable	START %Mni
Register	RESET %Ri PUT %Ri GET %Ri
Drum	RESET %DRi UP %DRi

Control structures	ST
Conditional action	IF...THEN... ELSIF...THEN... ELSE...END_IF;
Conditional iterative action	WHILE...DO...END_WHILE;
Conditional iterative action	REPEAT...UNTIL...END_REPEAT;
Repetitive action	FOR...DO...END_FOR;
Loop output instruction	EXIT

Arithmetic operations on integers (single and double length)	LD/IL/ST
Transfer or initialization	: =
Comparisons	= <> <= < > >=
Addition, subtraction, multiplication, division, remainder	+ - * / REM
AND, OR, exclusive OR, complement	AND OR XOR NOT
Absolute value	ABS
Square root	SQRT
Increment	INC
Decrement	DEC
Shift to the left	SHL
Shift to the right	SHR
Rotate shift to the left	ROL
Rotate shift to the right	ROR

Arithmetic operations on floating points	LD/IL/ST
Transfer or initialization	: =
Comparisons	= <> <= < > >=
Addition, subtraction, multiplication, division	+ - * /
Absolute value	ABS
Square root	SQRT

Numeric conversions	LD/IL/ST
Convert BCD to single length integer	BCD_TO_INT
Convert GRAY to single length integer	GRAY_TO_INT
Convert single length integer to BCD	INT_TO_BCD
Convert single length integer to floating point	INT_TO_REAL
Convert double length integer to floating point	DINT_TO_REAL
Convert floating point to single length integer	REAL_TO_INT
Convert floating point to double length integer	REAL_TO_DINT
Convert 32-bit BCD to 32-bit integer	DBCD_TO_DINT
Convert 32-bit integer to 32-bit BCD	DINT_TO_DBCD
Convert 32-bit BCD to 16-bit integer	DBCD_TO_INT
Convert 16-bit integer to 32-bit BCD	INT_TO_DBCD

Bit tables	LD/IL/ST
Transfer or initialization	: =
Copy a bit table into a bit table	COPY_BIT
AND between two tables	AND_ARX
OR between two tables	OR_ARX
Exclusive OR between two tables	XOR_ARX
Negation on a table	NOT_ARX
Copy a bit table into a word table	BIT_W
Copy a bit table into a double word table	BIT_D
Copy a word table into a bit table	W_BIT
Copy a double word table into a bit table	D_BIT

Instructions on tables	LD/IL/ST
Transfer and initialization	: =
Arithmetic operations between tables	+ - * / REM
Logic operations between tables	AND OR XOR
Arithmetic operations between a table and an integer	+ - * / REM
Logic operations between a table and an integer	AND OR XOR
Complement of the elements of a table	NOT
Sum of all the elements of a table	SUM
Comparison of two tables	EQUAL
Find 1st element of a table equal to a value	FIND_EQW, FIND_EQD
Find 1st element of a table greater than a value	FIND_GTW, FIND_GTD
Find 1st element of a table less than a value	FIND_LTW, FIND_LTD
Find highest value in a table	MAX_ARW, MAX_ARD
Find lowest value in a table	MIN_ARW, MIN_ARD
Number of occurrences of a value in a table	OCCUR_ARW, OCCUR_ARD
Rotate shift a table to the left	ROL_ARW, ROL_ARD
Rotate shift a table to the right	ROR_ARW, ROR_ARD
Sort a table (ascending or descending)	SORT_ARW, SORT_ARD

Instructions on floating point tables	LD/IL/ST
Transfer and initialization	: =

"Orphee" instructions	LD/IL/ST
Shift to the left on word with recovery of shifted bits	WSHL_RBIT, DSHL_RBIT
Shift to the right on word with sign extension and recovery of shifted bits	WSHR_RBIT, DSHR_RBIT
Shift to the right on word with filling with 0 and recovery of shifted bits	WSHRZ_C, DSHRZ_C
Up/down counting with indication of overshoot	SCOUNT

Explicit exchanges	LD/IL/ST
Read %M parameters of a logic channel	READ_PARAM
Read status %M of a logic channel	READ_STS
Restore %M parameters of a logic channel	RESTORE_PARAM
Save %M parameters of a logic channel	SAVE_PARAM
Write control %M of a logic channel	WRITE_CMD
Write %M parameters of a logic channel	WRITE_PARAM

Time management instructions	LD/IL/ST
Comparisons	= <> <= < > >=
Transfer	: =
Read date and code of last PLC stop	PTC
Read system date	RRTC
Update system date	WRTC
Add a time to a full date	ADD_DT
Add a time to a time of day	ADD_TOD
Convert date to string	DATE_TO_STRING
Day of the week	DAY_OF_WEEK
Difference between two dates	DELTA_D
Difference between two full dates	DELTA_DT
Difference between two times of day	DELTA_TOD
Convert full date to string	DT_TO_STRING
Subtract a time from a full date	SUB_DT
Subtract a time from time of day	SUB_TOD
Convert time to string	TIME_TO_STRING
Convert time of day to string	TOD_TO_STRING
Change time to hours-min-sec format	TRANS_TIME

Instructions on character strings	LD/IL/ST
Comparisons	= <> <= < > >=
Transfer	: =
Convert double integer to string	DINT_TO_STRING
Convert single integer to string	INT_TO_STRING
Convert string to double integer	STRING_TO_DINT
Convert string to single integer	STRING_TO_INT
Convert string to floating point	STRING_TO_REAL
Convert floating point to string	REAL_TO_STRING
Concatenate two strings	CONCAT
Delete substring	DELETE
Find first different character	EQUAL_STR
Find substring	FIND
Insert substring	INSERT
Extract left part of a string	LEFT
Length of a string	LEN
Extract substring	MID
Replace substring	REPLACE
Extract right part of a string	RIGHT

Multitasks and events	LD/IL/ST
Task activation / deactivation	%Si position %SWi position
Adjust task cycle time	
Global masking of events	MASKEVT
Global unmasking of events	UNMASKEVT

Communication	LD/IL/ST
Request to stop a function in progress	CANCEL
Send and/or receive data	DATA_EXCH
Request to read character string	INPUT_CHAR
Send and/or request to receive a character string	OUT_IN_CHAR
Send character string	PRINT_CHAR
Receive telegram	RCV_TLG
Read basic language objects	READ_VAR
Send/receive UNI-TE requests	SEND_REQ
Send telegram	SEND_TLG
Write basic language objects	WRITE_VAR

Integrated MMI	LD/IL/ST
Blocking entry of a variable on the CCX17	ASK_MSG
Dynamic assignment of keys on screen border	ASSIGN_KEYS
Command to control LEDs on front panel of CCX17	CONTROL_LEDS
Multiple entry of a variable on the CCX17	GET_MSG
Send command to the CCX17	PANEL_CMD
Display alarm message in PLC memory	SEND_ALARM
Display message in PLC memory	SEND_MSG

Basic PID control	LD/IL/ST
Mixed PID controller	PID
Management of CCX17 dedicated MMI for controlling PID loops.	PID_MMI
Pulse width modulation of a numeric value	PWM
PID output stage for controlling discrete valve	SERVO

8.1 General

This section calculates for TSX 37/57 PLCs :

- the execution time of the application program,
- the memory size of the application program.

Execution time of the application program

The execution time of the program is calculated using the tables on the following pages, by adding up the time taken for each program instruction.

Note : the time obtained is a maximum time. In fact, an operate block or a subroutine will only be processed if the execution condition (logic equation conditioning the execution of the block or subroutine) is true. It may be, therefore, that the actual time is much less than the maximum time calculated.

Calculating the complete cycle time involves parameters which are specific to the PLC (overhead time, duration of I/O exchange, etc.). Please refer to the installation manual of the PLC (performance section) for the complete calculation procedure.

Application memory size

The size of the application is the sum of the following elements :

Element	Calculation method
• Program	Add up each of the program instructions, (see tables in sections 8.2 and 8.3). and multiply by the coefficient which corresponds to the language used (see next page)
• Advanced functions	See section 8.4.4
• Configured PL7 objects	See section 8.4.2
• Configured I/O module	See section 8.4.3

In the tables on the following pages, information on sizes refers to the volume of instruction codes. In order to find out the total size of an instruction or a program, it is necessary to use a multiplication coefficient which takes account of information that is typical to a language (for example : graphic information in the case of Ladder language).

- Ladder language : Total volume = 1.7 x Code volume
- Structured Text language : Total volume = 1.6 x Code volume
- Instruction List language :
 - for TSX37 PLCs : Total volume = 1.4 x Code volume
 - for TSX57 PLCs : Total volume = 1.6 x Code volume
- Grafcet language :

The volume associated to the chart itself is as follows :

Chart volume (in words) = 214 + 17 * no. of chart steps + 2 * total no. of configured steps + 4 * no. of programmed actions

NB : program comments occupy 1 byte per character.

Note

The figures shown in the following tables are average estimations obtained from a typical application. It is not possible to provide exact data, since PL7 optimizes memory use according to the contents and structure of the application.

Section 8.4.1 describes the various memory zones occupied by the application.

8.2 TSX 37 performance

8.2.1 Boolean instructions

LD	IL	ST	Objects	Execution time (µs)			Size (words) 37xx
				3710	3720 ram	3720 cart	
				0.25	0.13	0.19	1
	LD,		%M1 (1)	0.25	0.13	0.19	1
	LDN		%M1[%MW2]	13.10	12.85	12.85	7
			%MWO:X0 (2)	6.06	5.75	5.75	4
			%IWi.j:Xk (3)	77.04	69.25	69.25	8
			%MWO[%MW10]:X0	16.29	15.55	15.55	8
			%KWO[%MW10]:X0	87.27	79.05	79.05	12
	LDR,		%M1	0.50	0.25	0.38	2
	LDF		%M1[%MW2]	13.01	12.75	12.75	7
	AND,		idem	LD,	LDN		
	ANDN , AND (, AND (N , idem OR						
	ANDR, ANDF, AND (R, AND (F, idem OR		idem	LDR	LDF		
	XOR, XORN		%M1	1.25	0.63	0.94	5
			%M1[%MW2]	26.94	26.08	26.26	13
			%MWO:X0	12.86	11.88	12.06	10
			%IWi.j:Xk	83.84	75.38	75.56	14
			%MWO[%MW10]:X0	33.33	31.48	31.66	14
			%KWO[%MW10]:X0	104.3	94.98	95.16	18
	XORR, XORF		%M1	2.25	1.13	1.69	9
			%M1[%MW2]	27.28	26.13	26.44	19
	ST, STN,		%M1	0.50	0.25	0.38	2
	S, R		%M1[%MW2]	13.10	12.85	12.85	7
			%MWO:X0	5.88	5.60	5.60	4
			%NW{i}:Xk (3)	76.86	69.10	69.10	8
			%MWO[%MW10]:X0	16.41	15.65	15.65	8
multiple coils in Ladder, "cost" of the 2 nd and subsequent coils				0.25	0.13	0.19	1
operation block	[action]		block executed	0.74	0.75	0.75	1
			not executed	5.55	5.40	5.40	1

(1) This concerns all the forceable bit objects : %I, %Q, %X, %M, %S

(2) Other objects of the same type : output bits of function block %Tmi.Q ..., system word extract bits %SWi:Xj

(3) Other objects of the same type: common word extract bits %NW{i}:Xk, I/O word extract bits %IWi.j.Xk, %QWi.j.Xk, extract bits of %KW, fault bits %li.j.ERR

LD	IL	ST	Objects	Execution time (μ s)			Size (words)
				3710	3720 ram	3720 cart	37xx
horizontal comparison block	LD [comparison]		time in addition to the comparison	0.00	0.00	0.00	0
vertical comparison block			between 2 %MWi	12.38	11.85	11.85	4
convergence))		0.25	0.13	0.19	1
divergence not followed by a convergence			ladder, 1 divergence	0.25	0.13	0.19	1
	MPS, MPP, MRD		list MPS+MPP	0.75	0.38	0.56	3
			list MRD	0.25	0.13	0.19	1

8.2.2 Function blocks

LD	IL	ST	objects/conditions	Execution time (μ s)		Size (words)
				3710	3720	37xx

IE timer

rising edge on IN	IN %TM1 (rising edge)	START %TM1	start timer	43.39	41.11	3
falling edge on IN	IN %TM1 (falling edge)	DOWN %TM1	stop timer	17.47	17.01	
IN =1	IN %TM1 (=1)		timer on	18.74	17.99	
IN =0	IN %TM1 (=0)		timer off	17.40	16.67	

PL7-3 timer

		START %T1	enable			3
		STOP %T1	freeze	12.63	12.15	
E=0		RESET %T1	reset	12.94	12.15	
			timer on	17.55	17.00	
			timer off			

Up/down counter

reset, R=1	R %C8 (=1)	RESET %C8	reset	18.69	17.92	3
preset, S=1	S %C9 (=1)	PRESET %C9	preset	20.42	19.73	
rising edge on CU	CU %C8 (rising edge)	UP %C8	up	19.92	19.10	
rising edge on CD	CD %C9 (rising edge)	DOWN %C9	down	19.92	19.10	
inactive inputs	R/S/CU/CD inactive bit		no action	13.27	12.81	

Function blocks (continued)

LD	IL	ST	objects/conditions	Execution time (µs)		Size (words)
				3710	3720	37xx

Monostable

rising edge on S	S %MN0, rising edge	START %MN0	start	35.08	33.16	3
S=1	S %MN0, S=1/0		active monostable	11.64	11.17	

Register

edge on I	I %R2 (edge)	PUT %R2	store	21.90	21.27	3
edge on O	O %R2 (edge)	GET %R2	retrieve	21.90	21.27	
R=1	R %R1 (=1)	RESET %R2	reset	16.90	16.02	
inactive inputs	I/O/R, inactive bit		no action	12.61	12.19	

Drum

edge on U	U %DR0	UP %DR1	up, fixed	181.37	169.13	3
			by control bit	19.30	19.30	
R=1	R %DR1	RESET %DR2	reset, fixed	174.15	162.03	
			by control bit	19.30	19.30	
inactive inputs	R/U, inactive bit		no action, fixed	175.92	164.00	
			by control bit	19.30	19.30	

8.2.3 Integer and floating point arithmetic

Corrections according to object type

The times and volumes on the following pages are given for %MW0, %MD0 or %MF0

Execution time (μ s)		Size (words)
3710	3720	37xx

- Value remove for immediate values

16#1234/%MW0		1.20	1.10	0
16#12345678 / %MD0 or %MF0		1.21	0.75	1

- Value to add for indexed words/double words/floating points

%MW2[%MW0] or %MD2[%MW0] or %MF2[%MW0]	object following the :=	10.52	10.05	4
	1st operation : the 1st operand not being indexed, or assignment	11.20	10.60	5
	2nd operand if the 1 st operand is also indexed	13.37	12.60	5

- Value to add for objects of the following type :

%KWi, %KWif[%MW0], %KDi, %KFi, common words, I/O words

70.98	63.50	2
-------	-------	---

Correction according to the context of the operation

- Value to add if the operation is in at least the 2nd position in the phrase, example *%MW2 in := %MW0 * %MW1 * %MW2, concerns the following operations

%MW0		0.69	0.55	0
%MD0 and %MF0		0.99	0.75	0

- Value to add for an operation with the result of an operation in parentheses or of a higher priority, example : %MW0 + %MW2 + (...)

%MW0		2.86	2.55	1
%MD0 and %MF0		3.60	3.15	1

ST	Objects	Conditions	Execution time (µs)		Size (words) 37xx
			3710	3720	
object after the :=	%MW0		4.81	4.50	2
	%MD0,%MF0		6.45	5.70	2
:=	%MW0		4.46	4.30	2
	%MD0 and %MF0		5.15	4.85	2
=, <>, <=, <, >, >=	%MW0		8.94	8.50	4
	%MD0		10.71	10.26	4
	%MF0		29.06	28.39	4
AND, OR, XOR	%MW0		7.29	6.90	3
	%MD0		9.21	8.55	3
+, -	%MW0		7.29	6.90	3
	%MD0		9.21	8.55	3
	%MF0		62.83	61.20	3
*	%MW0		9.75	9.10	3
	%MD0		39.63	36.50	3
	%MF0		58.26	56.90	3
/, REM	%MW0		10.69	10.08	3
	%MD0		205.21	201.38	3
/	%MF0		62.47	60.25	3
ABS, -object	%MW0		7.20	6.95	3
	%MD0		9.97	9.53	3
	%MF0		13.01	12.50	3
NOT	%MW0		6.69	6.45	3
	%MD0		7.80	7.40	3
SQRT	%MW0		17.02	16.70	3
	%MD0		85.73	85.25	3
	%MF0		165.04	158.40	3
INC, DEC	%MW0		4.86	4.40	2
	%MD0		5.20	4.75	2
SHL, SHR, ROL, ROR	%MW0	for 1 bit	17.74	17.05	5
	%MD0	for 1 bit	20.58	19.15	5
		per additional bit	0.063		

8.2.4 Program instructions

ST	Objects	Conditions	Execution time (μs)		Size (words)
			3710	3720	37xx
Jump %Li			41.93	38.20	3
Maskevt			12.21	10.80	1
Unmaskevt			40.27	37.10	1
SRi			48.68	42.88	3
Return			42.18	38.33	3

8.2.5 Command structure

ST		Execution time (μs)		Volume (words)
		3710	3720	37xx
<cond>	condition evaluation			
forceable bit	see Boolean instruction LD %M1			
comparison	see comparisons =,<,> ...			
if <cond > then <action> end_if;	the times and volumes indicated below should be added to those of the action contained in the structure			
true condition		3.60	3.30	2
false condition (jump)		5.55	5.40	
If <cond> then <action1> else <action2> end_if;				
true condition		9.15	8.70	4
false condition		5.55	5.40	
while <cond> do.<action> end_while				
go to loop with loop-back		9.15	8.70	2
exit loop		5.55	5.40	
repeat <action> until <cond> end_repeat				
go to loop with loop-back		5.55	5.40	2
last pass		3.60	3.30	
for <word1:=word2> to <word3> do <action> end_for				
entry to the for command, executed once only		8.58	8.25	15
go to loop with loop-back		29.38	27.35	
exit loop		20.42	19.40	

8.2.6 Numeric conversions

ST	Execution time (µs)			Volume (words)
	3710	3720 ram	3720 cart	
BCD_TO_INT	25.03	24.55	24.55	3
INT_TO_BCD	21.66	21.15	21.15	3
GRAY_TO_INT	36.98	36.55	36.55	3
INT_TO_REAL	40.90	40.75	40.75	3
DINT_TO_REAL	33.32	32.55	32.55	3
REAL_TO_INT	58.75	58.55	58.55	3
REAL_TO_DINT	44.59	44.05	44.05	3
DBCD_TO_DINT	1 324.85	1 065.15	1 134.70	5
DBCD_TO_INT	1 265.54	925.70	986.15	5
DINT_TO_DBCD	1 124.85	825.15	879.10	5
INT_TO_DBCD	564.85	445.15	474.40	5

8.2.7 Bit string

ST	conditions	Execution time (µs)			Volume (words)
		3710	3720 ram	3720 cart	
					37xx

Initializing a bit table

%M30:8 := 0	8 bits	19.38	18.88	18.88	6
%M30:16 := 1	16 bits	20.38	19.88	19.88	6
%M30:24 := 2	24 bits	24.25	23.35	23.35	6
%M30:32 := 2	32 bits	25.25	24.35	24.35	6

ST	conditions	Execution time (µs)			Volume (words) 37xx
		3710	3720 ram	3720 cart	

Copying a bit table to a bit table

	conditions	3710	3720 ram	3720 cart	Volume (words) 37xx
%M30:8 := %M20:8	8 bits	25.54	24.79	24.79	6
%M30:16 := %M20:16	16 bits	26.16	25.41	25.41	6
%M30:24 := %M20:24	24 bits	33.41	32.26	32.26	6
%M30:32 := %M20:32	32 bits	35.91	34.76	34.76	6
%M30:16 := COPY_BIT(%M20:16)	16 bits	281.63	230.00	244.95	9
	32 bits	440.82	360.00	383.40	9
	128 bits	1 261.22	1 030.00	1 096.95	9

Logic instructions on bit tables

	conditions	3710	3720 ram	3720 cart	Volume (words) 37xx
AND_ARX, OR_ARX, XOR_ARX					
%M0:16 := AND_ARX(%M30:16,%M50:16)	16 bits	397.42	320.00	340.80	12
%M0:32 := AND_ARX(%M30:32,%M50:32)	32	620.97	500.00	532.50	12
%M0:128 := AND_ARX(%M30:128,%M50:128)	128	1 887.74	1 520.00	1 618.80	12
NOT_ARX					
%M0:16 := NOT_ARX(%M30:16)	16 bits	281.63	230.00	244.95	9
	32	440.82	360.00	383.40	9
	128	1 261.22	1 030.00	1 096.95	9

Copying a bit table to a word table

	conditions	3710	3720 ram	3720 cart	Volume (words) 37xx
%MW1 := %M30:8	8 bits	14.84	14.36	14.36	5
%MW1 := %M30:16	16 bits	16.34	15.86	15.86	5
%MD2 := %M30:24	24 bits	14.54	14.23	14.23	5
%MD2 := %M30:32	32 bits	16.04	15.73	15.73	5
%MW1:4 := BIT_W(%M40:80,0,17,2)	17 bits	501.43	390.00	415.35	16
%MD1:4 := BIT_D(%M30:80,0,33,0)	33 bits	379.53	530.00	564.45	16

Copying a word table to a bit table

	conditions	3710	3720 ram	3720 cart	Volume (words) 37xx
%M30:8 := %MW1	8 bits	19.28	18.68	18.68	5
%M30:16 := %MW2	16 bits	20.28	19.68	19.68	5
%M30:24 := %MD1	24 bits	21.20	20.37	20.37	5
%M30:32 := %MD3	32 bits	22.20	21.37	21.37	5
%M30:32 := W_BIT(%MW200:2,0,2,0)	32 bits	488.68	370.00	394.05	16
%M30:32 := D_BIT(%MD0:1,0,2,0)	32 bits	567.33	460.00	489.90	16

8.2.8 Word, double word and floating point tables

ST	conditions	Execution time (µs)			volume (words)
		3710	3720 ram	3720 cart	
					37xx

Initializing a word table with a word

%MW0:10 := %MW100	10 words	47.46	42.15	42.15	7
	per word	0.34	0.20	0.20	
%MD0:10 := %MD100	10 double words	81.27	74.45	74.45	7
	per double word	2.87	2.65	2.65	

Copying a word table to a word table

%MW0:10 := %MW20:10;	10 words	95.80	85.35	85.35	9
	per word	0.77	0.50	0.50	
%MD0:10 := %MD20:10;	10 double words	111.13	97.65	97.65	9
	per double word	1.54	1.00	1.00	

Arithmetical and logic instructions between two word tables

+, -					
%MW0:10 := %MW10:10 + %MW20:10;	10 words	168.04	151.95	151.95	14
	per word	7.13	6.35	6.35	
%MD0:10 := %MD10:10 + %MD20:10;	10 double words	239.17	214.40	214.40	14
	per double word	13.84	12.25	12.25	
*					
%MW0:10 := %MW10:10 * %MW20:10;	10 words	189.32	175.40	175.40	14
	per word	9.27	8.70	8.70	
%MD0:10 := %MD10:10 * %MD20:10;	10 double words	710.35	603.80	603.80	14
	per double word	61.64	51.20	51.20	
/, REM					
%MW0:10 := %MW10:10 / %MW20:10;	10 words	224.76	181.40	181.40	14
	per word	13.14	9.30	9.30	
%MD0:10 := %MD10:10 / %MD20:10;	10 double words	2 192.38	2 157.35	2 157.35	14
	per double word	209.16	206.55	206.55	
AND, OR, XOR					
%MW0:10 := %MW10:10 AND %MW20:10;	10 words	163.69	147.40	147.40	14
	per word	6.66	5.85	5.85	
%MD0:10 := %MD10:10 AND %MD20:10;	10 double words	240.14	215.90	215.90	14
	per double word	13.94	12.40	12.40	

ST	conditions	Execution time (μs)			Volume (words)
		3710	3720 ram	3720 cart	37xx

Arithmetical and logic instructions between 1 word table and 1 word

+					
%MW0:10 :=%MW10:10 + %MW20;	10 words	119.12	108.55	108.55	12
or %MW0:10 := %MW20 + %MW10:10	per word	2.87	2.65	2.65	
%MD0:10 :=%MD10:10 + %MD20;	10 double words	159.68	147.45	147.45	12
	per double word	6.57	6.25	6.25	

*					
%MW0:10 :=%MW20*%MW10:10;	10 words	166.86	132.45	132.45	12
	per word	7.94	5.05	5.05	
%MD0:10 :=%MD20*%MD10:10;	10 double words	587.01	522.95	522.95	12
	per double word	49.18	43.80	43.80	
/, REM					
%MW0:10 :=%MW10:10 / %MW30;	10 words	196.69	155.85	155.85	12
	per word	10.86	7.30	7.30	
%MD0:10 :=%MD10:10 / %MD30;	10 double words	2 230.17	2 173.95	2 173.95	12
	per double word	213.66	208.90	208.90	
AND, OR, XOR					
%MW0:10 :=%MW10:10 AND %MW20;	10 words	117.20	106.45	106.45	12
	per word	2.64	2.40	2.40	
%MD0:10 :=%MD20*%MD10:10;	10 double words	587.01	522.95	522.95	12
	per double word	6.47	6.15	6.15	
NOT					
%MW0:10 :=NOT(%MW10:10);	10 words	110.28	100.25	100.25	9
	per word	2.96	2.75	2.75	
%MD0:10 :=NOT(%MD10:10);	10 double words	126.39	114.00	114.00	9
	per double word	4.50	4.05	4.05	

Table summing function

%MW20:=SUM(%MW0:10);	10 words	74.30	69.00	69.00	10
	per word	2.44	2.35	2.35	
%MD20:=SUM(%MD0:10);	10 double words	83.58	76.90	76.90	10
	per double word	3.17	2.95	2.95	

ST	conditions	Execution time (μs)			Volume (words) 37xx
		3710	3720 ram	3720 cart	

Comparison of tables function

%MW20:=EQUAL(%MW0:10;%MW10:10);	10 words	103.78	93.50	93.50	11
	per word	1.13	0.90	0.90	
%MD20:=EQUAL(%MD0:10;%MD10:10);	10 double words	116.17	103.40	103.40	11
	per double word	2.23	1.75	1.75	

Search

%MW20 := FIND_EQW(%MW0:10,%KW0)	10 words, max. instance	340.00	250.00	266.25	15
%MD20 := FIND_EQD(%MD0:10, %KD0)	10 double words, max. instance	350.00	260.00	276.90	16

Searching for max. and min. values

%MW20 := MAX_ARW(%MW0:10)	10 words	350.00	260.00	276.90	9
%MD20 := MAX_ARD(%MD0:10)	10 double words	410.00	300.00	319.50	9

Number of occurrences

%MW20 := OCCUR_ARW(%MW0:10, %KW0)	10 words	350.00	250.00	266.25	15
%MD20 := OCCUR_ARD(%MD0:10, %KD0)	10 double words	370.00	270.00	287.55	16

Rotate shift

ROL_ARW(word or value,%MWj:10)	10 words	550.00	400.00	426.00	9
ROL_ARD(%MDi,%MDj:10)	10 double words	590.00	430.00	457.95	9

Sort

SORT_ARW(%MWi,%MWj:10)	10 words, max. instance	970.00	700.00	745.50	9
SORT_ARD(%MDi,%MDj:10)	5 double words, max. instance	610.00	450.00	479.25	9

8.2.9 Time management

ST	Execution time (μ s)			Volume (words)
	3710	3720 ram	3720 cart	37xx

Date, time and duration OF

%MW2:4 := ADD_DT(%MW2:4,%MD8)	4 400.00	3 300.00	3 514.50	13
%MD2 := ADD_TOD(%MD2,%MD8)	2 100.00	1 550.00	1 650.75	9
%MB2:11 := DATE_TO_STRING(%MD40)	1 370.00	900.00	958.50	9
%MW5 := DAY_OF_WEEK()	220.00	280.00	298.20	5
%MD10 := DELTA_D(%MD2,%MD4)	1 520.00	1 130.00	1 203.45	9
%MD10 := DELTA_DT(%MD2:4,%MW6:4)	3 170.00	2 300.00	2 449.50	13
%MD10 := DELTA_TOD(%MD2,%MD4)	2 330.00	1 700.00	1 810.50	9
%MB2:20 := DT_TO_STRING(%MW50:4)	2 050.00	1 450.00	1 544.25	11
%MW2:4 := SUB_DT(%MW2:4,%MD8)	4 750.00	3 500.00	3 727.50	13
%MD2 := SUB_TOD(%MD2,%MD8)	2 330.00	1 700.00	1 810.50	9
%MB2:15 := TIME_TO_STRING(%MD40)	1 560.00	1 200.00	1 278.00	9
%MB2:9 := TOD_TO_STRING(%MD40)	1 270.00	800.00	852.00	9
%MD100 := TRANS_TIME(%MD2)	500.00	500.00	532.50	7

Access real-time clock

RRTC(%MW0:4)	93.60	84.80	84.80	5
WRTC(%MW0:4)	248.61	230.85	230.85	5
PTC(%MW0:5)	97.98	88.60	88.60	5

8.2.10 Character strings

ST	Conditions	Execution time (μs)			Volume (words)
		3710	3720 ram	3720 cart	37xx

Character string assignment, feedback

%MB0:8:=%MB10:8	8 characters	105.16	93.80	93.80	9
	per character	1.65	1.30	1.30	
%MB0:8:='abcdefg'	8 characters	120.72	110.20	110.20	11
	per character	4.15	3.85	3.85	0,5

Word <-> character string conversions

%MW1:=STRING_TO_INT(%MB0:7)		97.69	91.95	91.95	7
%MB0:7:=INT_TO_STRING(%MW0)		104.36	96.70	96.70	7

Double word <-> character string conversions

%MD1:=STRING_TO_DINT(%MB0:13)		1 070.53	965.62	965.62	7
%MB0:13:=DINT_TO_STRING(%MD0)		322.29	295.35	295.35	7

Floating point <-> character string conversions

%MF1:=STRING_TO_REAL(%MB0:15)		1 783.70	1 634.53	1 634.53	7
%MB0:15:=REAL_TO_STRING(%MF0)		741.75	681.20	681.20	7

String manipulation OF

%MB10:20 := CONCAT(%MB30:10,%MB50:10)		1 170.00	770.00	820.05	15
%MB10:20 := DELETE(%MB10:22,2,3);		950.00	600.00	639.00	15
%MW0 := EQUAL_STR(%MB10:20,%MB30:20);	the 5th character is different	860.00	520.00	553.80	13
%MW0 := FIND(%MB10:20,%MB30:10);		1 610.00	1 000.00	1 065.00	13
%MB10:20 := INSERT(%MB30:10,%MB50:10,4);		1 270.00	800.00	852.00	17
%MB10:20 := LEFT(%MB30:30,20);		920.00	570.00	607.05	13
%MW0 := LEN(%MB10:20);		770.00	340.00	362.10	9
%MB10:20 := MID(%MB30:30,20,10);		1 080.00	700.00	745.50	15
%MB10:20 REPLACE(%MB30:20,%MB50:10,10,10);	:=	1 450.00	870.00	926.55	19
%MB10:20 := RIGHT(%MB30:30,20);		1 480.00	950.00	1 011.75	13

8.2.11 Application-specific functions and Orphee function

ST	Conditions	Execution time (µs)			Volume (words)
		3710	3720 ram	3720 cart	
					37xx

Communication

SEND_REQ(%KW0:6,15,%MW0:1,%MW10:10,%MW30:4)		2182	1818	1936	21
SEND_TLG(%KW0:6,1,%MW0:5,%MW30:2)		1636	1364	1452	15

Man-machine interface

SEND_MSG(ADR#1.0,%MW0:2,%MW10:2)		2 240	2 000	2208	19
SEND_ALARM(ADR#1.0,%MW0:2,%MW10:2)		2 240	2 000	2208	19
GET_MSG(ADR#1.0,%MW0:2,%MW10:2)		2 240	2 000	2 208	19
GET_VALUE(ADR#1.0,%MW0,%MW10:2)		1 120	1 000	1 104	17
ASK_MSG(ADR#1.0,%MW0:2,%MW10:2,%MW20:2)		2 240	2 000	2 208	23
ASK_VALUE(ADR#1.0,%MW0,%MW10:2,%MW20:2)		2 240	2 000	2 208	21
DISPLAY_ALRM(ADR#1.0,%MW0,%MW10:2)		1 120	1 000	1 104	17
DISPLAY_GRP(ADR#1.0,%MW0,%MW10:2)		1 120	1 000	1 104	17
DISPLAY_MSG(ADR#1.0,%MW0,%MW10:2)		1 120	1 000	1 104	17
CONTROL_LEDS(ADR#1.0,%MW0:2,%MW10:2)		2 240	2 000	2 208	19
ASSIGN_KEYS(ADR#1.0,%MW0:2,%MW10:2)		2 240	2 000	2 208	19
PANEL_CMD(ADR#1.0,%MW0:2,%MW10:2)		2 240	2 000	2 208	19

Process control

PID('PIDS1','Unit',%IW3.5,%MW12,%M16,%MW284:43)	deval_mmi=0	1320	1100	1172	24
	deval_mmi=1	1080	900	958,5	
PWM(%MW11,%Q2.1,%MW385:5)		600	500	532,5	11
SERVO(%MW12,%IW3.6,%Q2.2,%Q2.3,%MW284:43,%MW390:10)		960	800	852	19
PID_MMI(ADR#0.0.4,%M1,%M2:5,%MW410:62)	EN=1	1140	950	1012	20

ST	Conditions	Execution time (µs)			Volume (words) 37xx
		3710	3720 ram	3720 cart	

Orphee function

DSHL_RBIT(%MD102,16,%MD204,%MD206)	read 10 words	440	320	341	13
DSHR_RBIT(%MD102,16,%MD204,%MD206)	write 10 words	660	480	511	13
DSHRZ_C(%MD102,16,%MD204,%MD206)	mirror req. 10 words	410	310	330	13
WSHL_RBIT(%MW102,8,%MW204,%MW206)	exchange 10 words	300	220	234	13
WSHR_RBIT(%MW102,8,%MW204,%MW206)	20 bytes	390	280	298	13
WSHRZ_C(%MW102,8,%MW204,%MW206)	20 bytes	300	220	234	13
SCOUNT(%M100,%MW100,%M101,%M102,%MW101,%MW102,%M200,%M201,%MW200,%MW201)	20 bytes	510	410	437	25

8.2.12 Explicit I/O

Read_Sts %Chi.MOD					
Any application, except for the processor communication channel		30	30	32	2
Read_Sts %Chi					
Analog input		180	180	216	6
Analog output		90	70	74	
CTZ counter module		110	95	104	
Write_Param %Chi					
Analog input		790	570	790	6
CTZ counter module		1127	1080	1083	
Read_Param %Chi					
Analog input		260	290	316	6
CTZ counter module		338	295	300	
Save_Param %Chi					
Analog input		1234	1220	1240	6
CTZ counter module		1370	1220	1240	
Restore_Param %Chi					
Analog input		550	510	535	6
CTZ counter module		1160	1080	1097	
Write_Cmd %Chi					
Discrete output		50	47	52	6

8.3 TSX 57 performance

8.3.1 Boolean instructions

LD	IL	ST	Objects	Execution time (µs)			Size (words) 57xx
				5710	5720 ram	5720 cart	
				0.29	0.12	0.21	1
 	LD,		%M1 (1)	0.58	0.25	0.37	1
	LDN		%M1[%MW2]	2.33	1.00	1.58	6
			%MW0:X0 (2)	1.46	0.62	1.00	4
			%IWj:Xk (3)	2.33	1.00	1.58	6
			%MW0[%MW10]:X0	3.50	1.50	2.37	9
			%KW0[%MW10]:X0	3.50	1.50	2.37	9
 	LDR,		%M1	0.87	0.37	0.58	2
	LDF		%M1[%MW2]	2.62	1.12	1.79	7
 	AND,			idem	LD,	LDN	
	ANDN , AND (, AND (N , idem OR						
 	ANDR, ANDF, AND (R, AND (F, idem OR			idem	LDR,	LDF	
 	XOR, XORN		%M1	2.04	0.87	1.37	5
			%M1[%MW2]	5.54	2.37	3.79	13
			%MW0:X0	4.08	1.75	2.83	10
			%IWj:Xk	4.96	2.12	3.42	14
			%MW0[%MW10]:X0	7.87	3.37	5.37	14
			%KW0[%MW10]:X0	7.87	3.37	5.37	18
 	XORR, XORF		%M1	2.25	1.13	1.69	9
			%M1[%MW2]	27.28	26.13	26.44	19
 	ST, STN,		%M1	1.17	0.50	0.75	2
	S, R		%M1[%MW2]	2.62	1.12	1.75	7
			%MW0:X0	1.75	0.75	1.17	4
			%NW{ij}:Xk (3)	2.62	1.12	1.75	8
			%MW0[%MW10]:X0	3.79	1.25	2.54	8
	multiple coils in Ladder, "cost" of 2nd and subsequent coils				0.87	0.37	0.54
operation block	[action]		block executed	0.58	0.25	0.42	1
			not executed	0.71	0.37	0.54	1

- (1) This refers to all forceable bit objects : %I, %Q, %X, %M, %S
- (2) Other objects of the same type : output bits of function block %Tmi.Q ..., system word extract bits %SWi:Xj, extract bits of %KW, fault bits %li.j.ERR
- (3) Other objects of the same type: common word extract bits %NW{ij}:Xk, I/O word extract bits %IWj.Xk, %QWj.Xk

LD	IL	ST	Objects	Execution time (µs)			Size (words)
				5710	5720 ram	5720 cart	57xx
horizontal comparison block	LD [comparison]		time in addition to the comparison	0.00	0.00	0.00	0
vertical comparison block			between 2 %MWi	2.04	0.87	1.37	5
convergence))		0.29	0.12	0.21	1
divergence not followed by a convergence			ladder, 1 divergence	0.29	0.12	0.21	1
	MPS, MPP, MRD		list MPS+MPP	0.87	0.37	0.62	3
			list MRD	0.29	0.12	0.21	1

8.3.2 Function blocks

LD	IL	ST	objects/conditions	Execution time (µs)		Size (words)
				5710	5720	57xx

IE timer

rising edge on IN	IN %TM1 (rising edge)	START %TM1	start timer	67.87	46.92	3
falling edge on IN	IN %TM1 (falling edge)	DOWN %TM1	stop timer	27.84	19.50	
IN =1	IN %TM1 (=1)		timer on	32.15	22.66	
IN =0	IN %TM1 (=0)		timer off	28.60	20.18	

PL7-3 timer

		START %T1	enable			3
		STOP %T1	freeze	23.08	16.16	
E=0		RESET %T1	reset	23.46	16.26	
			timer on	30.40	21.51	
			timer off			

Up/down counter

reset, R=1	R %C8 (=1)	RESET %C8	reset	30.38	21.26	3
preset, S=1	S %C9 (=1)	PRESET %C9	preset	33.99	23.55	
rising edge on CU	CU %C8 (rising edge)	UP %C8	up	33.81	23.49	
rising edge on CD	CD %C9 (rising edge)	DOWN %C9	down	33.81	23.49	
inactive steps	R/S/CU/CD inactive bit		no action	22.37	16.22	

Function blocks (continued)

LD	IL	ST	objects/conditions	Execution time (µs)		Size (words) 57xx
				5710	5720	

Monostable

rising edge on S	S %MN0, rising edge	START %MN0	start	57.42	40.65	3
S=1	S %MN0, S=1/0		active monostable	20.38	14.11	

Register

edge on I	I %R2 (edge)	PUT %R2	store	35.69	24.92	3
edge on O	O %R2 (edge)	GET %R2	retrieve	35.69	24.92	
R=1	R %R1 (=1)	RESET %R2	reset	26.83	18.80	
inactive inputs	I/O/R, inactive bit		no action	20.71	15.50	

Drum

edge on U	U %DR0	UP %DR1	up, fixed	268.69	185.41	3
			per control bit	25.00	25.00	
R=1	R %DR1	RESET %DR2	reset, fixed	257.01	176.06	
			per control bit	25.00	25.00	
inactive inputs	R/U, inactive bit		no action, fixed	259.07	179.06	
			per control bit	25.00	25.00	

8.3.3 Integer and floating point arithmetic

Corrections according to object type

The times and volumes on the following pages are given for %MW0, %MD0 or %MF0

	Execution time (µs)			Volume (words)
	5710	5720 ram	5720 cart	57xx

- Value to remove for immediate values

16#1234 / %MW0	0.29	0.12	0.17	0
16#12345678 / %MD0 or %MF0	0.29	0.12	0.12	1

- Value to add for indexed words/double words/floating points

%MW2[%MW0] %MD2[%MW0] %MF2[%MW0]	or or or	object following the :=	2.04	0.87	1.37	5
		1st operation, assignment	2.04	0.87	1.37	5
		2nd operation (or 1st indexed operand)	2.04	0.87	1.37	5

- Value to add for objects of the following type :

Common words, I/O words

0.87	0.37	0.58	2
------	------	------	---

Correction according to the context of the operation

- Value to add if the operation is in at least the 2nd position in the phrase, example *%MW2 in := %MW0 * %MW1 * %MW2, concerns the following operations: *,/,REM on words and double words, or all operations on floating points

%MW0	0.58	0.25	0.37	1
%MD0 and %MF0	0.87	0.37	0.54	1

- Value to add for an operation with the result of an operation in parentheses, or of a higher priority, example : %MW0 + %MW2 + (...)

%MW0	0.29	0.12	0.17	1
%MD0 and %MF0	0.58	0.25	0.33	1

ST	objects	conditions	Execution time (µs)			Volume (words)
			5710	5720 ram	5720 cart	57xx
object after the :=	%MW0		0.87	0.37	0.58	2
		%MW0+(..., or before *, /, REM	1.17	0.50	0.75	2
	%MD0		1.17	0.50	0.75	2
		%MD0(..., or before *, /, REM	1.75	0.75	1.08	2
:=	%MW0		0.87	0.37	0.58	2
	%MD0 et %MF0		1.17	0.50	0.75	2
=, <, <=, >, >=	%MW0		1.17	0.50	0.79	4
	%MD0		1.46	0.62	1.04	4
	%MF0		48.36	33.88	34.13	4
AND, OR, XOR	%MW0		0.87	0.37	0.58	3
	%MD0		1.17	0.50	0.75	3
+, -	%MW0		0.87	0.37	0.58	3
	%MD0		1.17	0.50	0.75	3
	%MF0		99.42	71.51	71.76	3
*	%MW0		10.83	9.14	9.35	3
	%MD0		55.31	42.71	42.96	3
	%MF0		87.60	63.61	63.86	3
/, REM	%MW0		11.93	9.99	10.20	3
	%MD0		333.15	226.54	226.79	3
/	%MF0		95.83	68.51	68.76	3
ABS, -object	%MW0		0.87	0.37	0.58	2
	%MD0		1.17	0.54	0.75	2
	%MF0		18.82	13.01	13.26	2
NOT	%MW0		0.87	0.37	0.58	2
	%MD0		1.17	0.54	0.75	2
SQRT	%MW0		22.20	16.24	16.45	3
	%MD0		111.29	89.66	89.91	3
	%MF0		233.62	173.01	173.26	3
INC, DEC	%MW0		1.17	0.50	0.75	2
	%MD0		1.75	0.75	1.08	2
SHL, SHR, ROL, ROR	%MW0	for 1 bit	2.92	1.25	1.96	10
	%MD0	for 1 bit	3.21	1.37	2.12	10
		per additional bit	0.042			

8.3.4 Program instructions

ST	Objects	Conditions	Execution time (μs)			Volume (words)
			5710	5720 ram	5720 cart	57xx
Jump %Li			2.58	1.25	1.71	3
Maskevt			33.98	23.96	23.96	1
Unmaskevt			34.54	24.41	24.41	1
SRi			3.92	1.75	2.42	2
Return			1.00	0.50	0.71	2

8.3.5 Command structure

ST		Execution time (μs)			Volume (words)
		5710	5720 ram	5720 cart	57xx
<cond>	condition evaluation				
forceable bit	see Boolean instruction LD %M1				
comparison	see comparisons =,<,> ...				
if <cond > then <action> end_if;	the times and volumes indicated below should be added to those of the action contained in the structure				
true condition		0.58	0.25	0.42	2
false condition (jump)		0.71	0.37	0.54	
If <cond> then <action1> else <action2> end_if;					
true condition		1.29	0.62	0.96	4
false condition		0.71	0.37	0.54	
while <cond> do.<action> end_while					
go to loop with loop-back		1.29	0.62	0.96	2
exit loop		0.71	0.37	0.54	
repeat <action> until <cond> end_repeat					
go to loop with loop-back		0.71	0.37	0.54	2
last pass		0.58	0.25	0.42	
for <word1:=word2> to <word3> do <action> end_for					
entry to the for command, execute once only		1.75	0.75	1.17	15
go to loop with loop-back		5.08	2.12	3.29	
exit loop		2.46	1.12	1.71	

8.3.6 Numeric conversions

ST	Execution time (μ s)			Volume (words) 57xx
	5710	5720 ram	5720 cart	
BCD_TO_INT	30.33	24.44	24.65	3
INT_TO_BCD	24.48	20.19	20.40	3
GRAY_TO_INT	59.73	40.79	41.00	3
INT_TO_REAL	60.35	40.64	40.85	3
DINT_TO_REAL	49.14	34.76	35.01	3
REAL_TO_INT	91.59	60.86	61.11	3
REAL_TO_DINT	68.84	48.31	48.56	3
DBCD_TO_DINT	1 625.73	1 069.50	1 192.30	5
DBCD_TO_INT	1 379.53	909.63	1 014.07	5
DINT_TO_DBCD	1 251.05	819.50	913.55	5
INT_TO_DBCD	620.01	439.50	489.85	5

8.3.7 Bit string

ST	conditions	Execution time (μ s)			Volume (words) 57xx
		5710	5720 ram	5720 cart	

Initializing a bit table

%M30:8 := 0	8 bits	6.71	2.87	4.12	7
%M30:16 := 1	16 bits	11.37	4.87	6.79	7
%M30:24 := 2	24 bits	24.47	15.31	16.81	12
%M30:32 := 2	32 bits	29.13	17.31	19.48	12

Copying a bit table to a bit table

%M30:8 := %M20:8	8 bits	13.71	5.87	8.17	8
%M30:16 := %M20:16	16 bits	16.62	7.12	9.83	8
%M30:24 := %M20:24	24 bits	45.46	24.31	28.85	13
%M30:32 := %M20:32	32 bits	57.13	29.31	35.52	13
%M30:16 := COPY_BIT(%M20:16)	16 bits	322.00	230.00	256.45	17
	32 bits	490.00	350.00	390.25	17
	128 bits	1 526.00	1 090.00	1 215.35	17

ST	conditions	Execution time (µs)			Volume (words) 57xx
		5710	5720 ram	5720 cart	

Logic instructions on bit tables

Instruction	Conditions	5710	5720 ram	5720 cart	Volume (words)
AND_ARX, OR_ARX, XOR_ARX					
%M0:16 := AND_ARX(%M30:16,%M50:16)	16 bits	434.00	310.00	345.65	24
%M0:32 := AND_ARX(%M30:32,%M50:32)	32	686.00	490.00	546.35	24
%M0:128 AND_ARX(%M30:128,%M50:128)	128	2 198.00	1 570.00	1 750.55	24
NOT_ARX					
%M0:16 := NOT_ARX(%M30:16)	16 bits	322.00	230.00	256.45	17
	32	490.00	350.00	390.25	17
	128	1 526.00	1 090.00	1 215.35	17

Copying a bit table to a word table

Instruction	Conditions	5710	5720 ram	5720 cart	Volume (words)
%MW1 := %M30:8	8 bits	8.75	3.75	5.25	6
%MW1 := %M30:16	16 bits	15.75	6.75	9.25	6
%MD2 := %M30:24	24 bits	23.04	10.54	13.83	6
%MD2 := %M30:32	32 bits	30.04	13.54	17.83	6
%MW1:2 := BIT_W(%M40:17,0,17,0)	17 bits	518.00	370.00	412.55	23
%MD1:2 := BIT_D(%M30:33,0,33,0)	33 bits	728.00	520.00	579.80	23

Copying a word or a word table to a bit table

Instruction	Conditions	5710	5720 ram	5720 cart	Volume (words)
%M30:8 := %MW1	8 bits	6.71	2.87	4.08	6
%M30:16 := %MW2	16 bits	11.37	4.87	6.75	6
%M30:24 := %MD1	24 bits	24.76	16.11	17.36	11
%M30:32 := %MD3	32 bits	29.42	18.11	20.02	11
%M30:32 := W_BIT(%MW0:2,0,2,0)	32 bits	518.00	370.00	412.55	23
%M30:32 := D_BIT(%MD0:1,0,2,0)	32 bits	616.00	440.00	490.60	23

8.3.8 Word, double word and floating point tables

ST	conditions	Execution time (μ s)			Volume (words) 57xx
		5710	5720 ram	5720 cart	

Initializing a word table with a word

%MW0:10 := %MW100	10 words	74.48	50.16	51.12	10
	per word	0.47	0.25	0.25	
%MD0:10 := %MD100	10 words	115.45	78.29	79.29	10
	per word	4.41	2.95	2.95	

Copying a word table to a word table

%MW0:10 := %MW20:10;	10 words	139.71	93.76	95.26	15
	per word	0.95	0.50	0.50	
%MD0:10 := %MD20:10;	10 words	151.18	102.46	103.96	15
	per word	2.02	1.30	1.30	

Arithmetical and logic instructions between 2 word tables

+, -					
%MW0:10 := %MW10:10 + %MW20:10;	10 words	236.81	162.54	164.79	23
	per word	10.05	7.15	7.15	
%MD0:10 := %MD10:10 + %MD20:10;	10 words	325.79	229.79	232.04	23
	per word	18.41	13.55	13.55	
*					
%MW0:10 := %MW10:10 * %MW20:10;	10 words	246.52	184.99	187.24	23
	per word	11.03	9.40	9.40	
%MD0:10 := %MD10:10 * %MD20:10;	10 words	881.77	658.44	660.69	23
	per word	74.04	56.40	56.40	
/, REM					
%MW0:10 := %MW10:10 / %MW20:10;	10 words	249.44	192.64	194.89	23
	per word	11.35	10.15	10.15	
%MD0:10 := %MD10:10 / %MD20:10;	10 words	3 669.10	2 501.99	2 504.24	23
	per word	352.83	240.75	240.75	
AND, OR, XOR					
%MW0:10 := %MW10:10 AND %MW20:10;	10 words	235.38	160.14	162.39	23
	per word	9.94	6.90	6.90	
%MD0:10 := %MD10:10 AND %MD20:10;	10 words	322.35	231.09	233.34	23
	per word	18.05	13.65	13.65	

ST	conditions	Execution time (µs)			Volume (words) 57xx
		5710	5720 ram	5720 cart	

Arithmetical and logic instructions between 1 word table and 1 word

+, -					
%MW0:10 :=%MW10:10 + %MW20;	10 words	170.69	115.14	116.85	18
or %MW0:10 := %MW20 + %MW10:10	per word	4.37	2.85	2.85	
%MD0:10 :=%MD10:10 + %MD20;	10 double words	230.52	156.91	158.66	18
	per double word	9.92	6.75	6.75	
*					
%MW0:10 :=%MW20*%MW10:10;	10 words	180.13	137.04	138.75	18
	per word	5.31	5.05	5.05	
%MD0:10 :=%MD20*%MD10:10;	10 double words	747.33	568.16	569.91	18
	per double word	61.59	47.85	47.85	
/, REM					
%MW0:10 :=%MW10:10 / %MW30;	10 words	212.87	166.39	168.10	18
	per word	8.48	7.90	7.90	
%MD0:10 :=%MD10:10 / %MD30;	10 double words	3 536.66	2 418.56	2 420.31	18
	per double word	340.51	232.90	232.90	
AND, OR, XOR					
%MW0:10 :=%MW10:10 AND %MW20;	10 words	170.89	115.59	117.30	18
	per word	4.39	2.90	2.90	
%MD0:10 :=%MD20*%MD10:10;	10 double words	747.33	568.16	569.91	18
	per double word	9.81	6.50	6.50	
NOT					
%MW0:10 :=NOT(%MW10:10);	10 words	161.71	109.06	110.56	15
	per word	4.37	2.85	2.85	
%MD0:10 :=NOT(%MD10:10);	10 double words	184.14	125.51	127.01	15
	per double word	6.61	4.50	4.50	

Table summing function

%MW20:=SUM(%MW0:10);	10 words	111.87	76.61	77.57	16
	per word	3.40	2.40	2.40	
%MD20:=SUM(%MD0:10);	10 double words	130.74	87.74	88.74	16
	per double word	4.96	3.30	3.30	

ST	conditions	Execution time (µs)			Volume (words) 57xx
		5710	5720 ram	5720 cart	

Comparison of tables function

%MW20:=EQUAL(%MW0:10;%MW10:10);	10 words	142.90	99.14	100.85	17
	per word	1.14	0.95	0.95	
%MD20:=EQUAL(%MD0:10;%MD10:10);	10 double words	155.56	110.06	111.81	17
	per double word	2.33	2.00	2.00	

Search

%MW20 := FIND_EQW(%MW0:10,%KW0)	10 words	374.68	240.00	267.60	14
%MD20 := FIND_EQD(%MD0:10, %KD0)	10 double words	394.40	260.00	289.90	15

Searching for max. and min. values

%MW20 := MAX_ARW(%MW0:10)	10 words	404.26	260.00	289.90	12
%MD20 := MAX_ARD(%MD0:10)	10 double words	483.14	310.00	345.65	12

Number of occurrences

%MW20 := OCCUR_ARW(%MW0:10, %KW0)	10 words	394.40	260.00	289.90	14
%MD20 := OCCUR_ARD(%MD0:10, %KD0)	10 double words	423.98	280.00	312.20	15

Rotate shift

ROL_ARW(word or value,%MWj:10)	10 words	621.18	400.00	446.00	12
ROL_ARD(%MDi,%MDj:10)	10 double words	670.48	430.00	479.45	12

Sort

SORT_ARW(%MWi,%MWj:10)	10 words	1 133.90	720.00	802.80	12
SORT_ARD(%MDi,%MDj:10)	10 double words	700.06	440.00	490.60	12

8.3.9 Time management

ST	Execution time (μ s)			Volume (words) 57xx
	5710	5720 ram	5720 cart	

Date, time and duration OF

%MW2:4 := ADD_DT(%MW2:4,%MD8)	5 176.50	3 500.00	3 902.50	19
%MD2 := ADD_TOD(%MD2,%MD8)	2 435.42	1 640.00	1 828.60	9
%MB2:11 := DATE_TO_STRING(%MD40)	1 429.70	970.00	1 081.55	12
%MW5 := DAY_OF_WEEK()	552.16	390.00	434.85	5
%MD10 := DELTA_D(%MD2, %MD4)	1 725.50	1 170.00	1 304.55	9
%MD10 := DELTA_DT(%MD2:4,%MW6:4)	3 549.60	2 410.00	2 687.15	19
%MD10 := DELTA_TOD(%MD2,%MD4)	2 632.62	1 780.00	1 984.70	9
%MB2:20 := DT_TO_STRING(%MW50:4)	2 297.38	1 550.00	1 728.25	17
%MW2:4 := SUB_DT(%MW2:4,%MD8)	5 492.02	3 750.00	4 181.25	19
%MD2 := SUB_TOD(%MD2,%MD8)	2 622.76	1 780.00	1 984.70	9
%MB2:15 := TIME_TO_STRING(%MD40)	1 922.70	1 270.00	1 416.05	12
%MB2:9 := TOD_TO_STRING(%MD40)	1 281.80	830.00	925.45	12
%MD100 := TRANS_TIME(%MD2)	788.80	530.00	590.95	7

Access real-time clock

RRTC(%MW0:4)	164.81	111.44	112.19	8
WRTC(%MW0:4)	152.94	103.79	104.54	8
PTC(%MW0:5)	165.36	111.79	112.54	8

8.3.10 Character strings

ST	conditions	Execution time (µs)			Volume (words) 57xx
		5710	5720 ram	5720 cart	

Character string assignment, feedback

%MB0:8:=%MB10:8	8 characters	147.75	103.56	103.56	15
	per character	1.64	1.25	1.25	
%MB0:8:='abcdefg'	8 characters	193.86	136.88	136.88	14
	per character	5.94	4.35	4.35	0,5

Word <-> character string conversions

%MW1:=STRING_TO_INT(%MB0:7)		145.69	104.31	104.52	10
%MB0:7:=INT_TO_STRING(%MW0)		149.67	109.21	109.42	10

Double word <-> character string conversions

%MD1:=STRING_TO_DINT(%MB0:13)		1 408.43	1 061.01	1 061.01	10
%MB0:13:=DINT_TO_STRING(%MD0)		411.64	317.69	317.94	10

Floating point <-> character string conversions

%MF1:=STRING_TO_REAL(%MB0:15)		2 606.63	1 815.08	1 815.33	10
%MB0:15:=REAL_TO_STRING(%MF0)		1 084.46	752.94	753.27	10

String manipulation OF

%MB10:20 := CONCAT(%MB30:10,%MB50:10)		1 106.00	790.00	880.85	24
%MB10:20 := DELETE(%MB10:22,2,3);		896.00	640.00	713.60	21
%MW0 := EQUAL_STR(%MB10:20,%MB30:20);		756.00	540.00	602.10	19
%MW0 := FIND(%MB10:20,%MB30:10);		1 456.00	1 040.00	1 159.60	19
%MB10:20 := INSERT(%MB30:10,%MB50:10,4);		1 162.00	830.00	925.45	26
%MB10:20 := LEFT(%MB30:30,20);		826.00	590.00	657.85	19
%MW0 := LEN(%MB10:20);		490.00	350.00	390.25	12
%MB10:20 := MID(%MB30:30,20,10);		994.00	710.00	791.65	21
%MB10:20 := REPLACE(%MB30:20,%MB50:10,10,10);		1 246.00	890.00	992.35	28
%MB10:20 := RIGHT(%MB30:30,20);		1 358.00	970.00	1 081.55	19

8.3.11 Application-specific functions and Orphee function

ST	Conditions	Execution time (µs)			Volume (words)
		5710	5720 ram	5720 cart	57xx

Communication

SEND_REQ(%KW0:6,15,%MW0:1,%MW10:10,%MW30:4)		2 800	2 000	2 230	33
SEND_TLG(%KW0:6,1,%MW0:5,%MW30:2)		2 100	1 500	1 673	24

Man-machine interface

SEND_MSG(ADR#1.0,%MW0:2,%MW10:2)		2 800	2 000	2 230	25
SEND_ALARM(ADR#1.0,%MW0:2,%MW10:2)		2 800	2 000	2 230	25
GET_MSG(ADR#1.0,%MW0:2,%MW10:2)		2 800	2 000	2 230	25
GET_VALUE(ADR#1.0,%MW0,%MW10:2)		1 400	1 000	1 115	20
ASK_MSG(ADR#1.0,%MW0:2,%MW10:2,%MW20:2)		2 800	2 000	2 230	32
ASK_VALUE(ADR#1.0,%MW0,%MW10:2,%MW20:2)		2 800	2 000	2 230	27
DISPLAY_ALARM(ADR#1.0,%MW0,%MW10:2)		1 400	1 000	1 115	20
DISPLAY_GRP(ADR#1.0,%MW0,%MW10:2)		1 400	1 000	1 115	20
DISPLAY_MSG(ADR#1.0,%MW0,%MW10:2)		1 400	1 000	1 115	20
CONTROL_LEDS(ADR#1.0,%MW0:2,%MW10:2)		2 800	2 000	2 230	25
ASSIGN_KEYS(ADR#1.0,%MW0:2,%MW10:2)		2 800	2 000	2 230	25
PANEL_CMD(ADR#1.0,%MW0:2,%MW10:2)		2 800	2 000	2 230	25

Process control

PID('PIDS1','Unit',%IW3.5,%MW12,%M16,%MW284:43)	deval_mmi=0	1700	1100	1227	32
	deval_mmi=1	1500	900	1004	
PWM(%MW11,%Q2.1,%MW385:5)		700	500	557,5	17
SERVO(%MW12,%IW3.6,%Q2.2,%Q2.3,%MW284:43,%MW390:10)		1000	800	892	31
PID_MMI(ADR#0.0.4,%M1,%M2:5,%MW410:62)	EN=1	1400	1000	1115	30

ST	Conditions	Execution time (µs)			Volume (words) 57xx
		5710	5720 ram	5720 cart	

Orphee function

ST	Conditions	5710	5720 ram	5720 cart	Volume (words) 57xx
DSHL_RBIT(%MD102,16,%MD204,%MD206)	read 10 words	493	320	357	17
DSHR_RBIT(%MD102,16,%MD204,%MD206)	write 10 words	749	510	569	17
DSHRZ_C(%MD102,16,%MD204,%MD206)	mirror req. 10 words	493	310	346	17
WSHL_RBIT(%MW102,8,%MW204,%MW206)	exchange 10 words	365	220	245	17
WSHR_RBIT(%MW102,8,%MW204,%MW206)	20 bytes	424	290	323	17
WSHRZ_C(%MW102,8,%MW204,%MW206)	20 bytes	365	220	245	17
SCOUNT(%M100,%MW100,%M101,%M102,%MW101,%MW102,%M200,%M201,%MW200,%MW201)	20 bytes	670	420	468	38

8.3.12 Explicit I/O

ST	Conditions	Execution time (µs)			Volume (words) 57xx
		5710	5720 ram	5720 cart	

ST	Conditions	5710	5720 ram	5720 cart	Volume (words) 57xx
Read_Sts %CHI.MOD					
Whatever the application, except for the processor communication channel		997	712	748	2
Read_Sts %CHI					
Discrete input		462	330	347	6
Discrete output		630	450	473	
Analog input		510	380	390	
Analog output		500	370	380	
CTY		510	380	390	
CFY		500	370	380	
CAY		518	370	389	
Write_Param %CHI					
Analog input		860	620	630	6
Analog output		810	580	600	
CTY		532	380	399	
CFY		0	0	0	
CAY		860	620	630	

ST	Conditions	Execution time (µs)			Volume (words)
		5710	5720 ram	5720 cart	57xx
Read_Param %CHi					
Analog input		180	120	130	6
Analog output		180	120	130	
CTY		1 134	810	851	
CFY		1 064	760	798	
CAY		784	560	588	
Save_Param %CHi					
Analog input		1 300	880	890	6
Analog output		1 300	890	900	
CTY		180	120	130	
CFY		180	120	130	
CAY		532	380	399	
Restore_Param %CHi					
Analog input		800	570	590	6
Analog output		800	570	590	
CTY		630	450	473	
CFY		0	0	0	
CAY		1 300	880	890	
Write_Cmd %CHi					
Discrete output		1 722	1 230	1 292	6
Analog input					
. input forcing		200	140	150	
. input forcing		1 390	1 020	1 040	
Analog output : output forcing					
Discrete output		210	150	150	
Smove %CHi					
CFY		700	500	525	19
CAY		820	580	610	

8.4 Size of the application

8.4.1 Description of the memory zones

The application is divided into several memory zones :

- bit memory zone :
 - this zone is specific to TSX 37 PLCs and is limited to 1280 bits,
 - this zone is part of the data memory zone for TSX 57 PLCs,
- data memory zone (words)
- application memory zone, comprising :
 - configuration
 - program,
 - constants

The bit and data memory zones are always stored in the internal RAM, the application memory zone can be stored in the internal RAM or on a memory card. The memory structure is described in section 1.3. part A.

8.4.2 Memory size of PL7 objects

	Bit memory (in words)	Data (in words)	Application (in words)
Grafcet steps (%Xi, %Xi.T)	0.5	1	
%Mi	0.5		
Numerics (%MWi)		1	
Constants (%KWi)			1.25
%NWi		1	
%Ti		4	2
%TMi		5	2
%MNi		4	2
%Ci		3	1
%Ri (length lg)		6+lg	2
%DRi		6	49

Grafcet interpreter data = 355 + 2 x No active steps configured + (No. of valid transitions configured) / 2

8.4.3 Module memory size

Note

This information is given for a particular processor version. It may be subject to slight variations as the product develops.

For each module type, the following tables provide the size occupied in each of the zones as well as a fixed size which should be added to the power consumption table the first time an application-specific function is used.

Module memory power consumption table, on TSX 37

Processors	Bit memory (words)	Data (words)	Application Zone (words)
TSX 37-10	70	1560	920
TSX 37-21	70	1570	930
TSX 37-22	70	2110	1280
Use of FAST task (TSX 37)		260	
First use of event (TSX 37)		520	

Discrete family	Bit memory (words)	Data (words)	Application Zone (words)
8 discrete inputs	4	12	40
16 discrete inputs	8	12	50
4 discrete outputs	2	12	40
8 discrete outputs	4	12	40
16 discrete inputs / 12 discrete outputs	16	20	100
32 discrete inputs / 32 discrete outputs	32	20	142

4 ANA input family	Bit memory (words)	Data (words)	Application Zone (words)
AEZ414	0	156	56
Add. amount 1st module in 4 ANA input family			120

8 ANA input family	Bit memory (words)	Data (words)	Application Zone (words)
AEZ801/AEZ802	0	212	72
Add. amount 1st module in 8 ANA input family			120

ANA output family	Bit memory (words)	Data (words)	Application Zone (words)
ASZ200	0	52	40
ASZ401	0	100	59
Add. amount 1st module in ANA output family			120

Counter family	Bit memory (words)	Data (words)	Application Zone (words)
CTY1A	16	108	64
CTY2A	32	212	106
Add. amount 1st Upcounter channel			144
Add. amount 1st Downcounter channel			144
Add. amount 1st Up/Down counter channel			144

Communication family	Bit memory (words)	Data (words)	Application Zone (words)
STZ010	0	36	168
SCP111/ SCP112/ SCP114 on UC UTW)	0	40	763
FPP 20 on UC (Channel 0 UTW)	0	40	755

Module memory power consumption table, on TSX 57

Processors	Bit memory (words)	Data (words)	Application Zone (words)
TSX 57-10	70	3852	1164
TSX 57-20	70	4125	1227
Use of FAST task (TSX 57)		520	
Add. amount 1st module		600	

Single discrete input family	Bit memory (words)	Data (words)	Application Zone (words)
8 discrete inputs	4	100	100
16 discrete inputs	8	130	110
32 discrete inputs	16	230	120
64 discrete inputs	32	430	190
Add. amount 1st single discrete input family module			610

Single discrete output family	Bit memory (words)	Data (words)	Application Zone (words)
8 discrete outputs	4	110	100
16 discrete outputs	8	160	110
32 discrete outputs	16	280	120
64 discrete outputs	32	550	190
Add. amount 1st single discrete output family module			570

Event-triggered discrete input family	Bit memory (words)	Data (words)	Application Zone (words)
16 discrete inputs (DEY 16FK)	8	220	130
Add. amount 1st EVT discrete input family module			680

Analog input family	Bit memory (words)	Data (words)	Application Zone (words)
AEY414	4	430	160
AEY800	8	840	240
AEY1600	16	1670	430
Add. amount 1st analog input family module			2990

Analog output family	Bit memory (words)	Data (words)	Application Zone (words)
ASY410	4	430	160
Add. amount 1st analog output family module			1700

Counter family	Bit memory (words)	Data (words)	Application Zone (words)
CTY2A module	32	410	170
CTY4A module	64	800	250
Add. amount 1st configured counter channel			1740

Servo-Motor family	Bit memory (words)	Data (words)	Application Zone (words)
CAY21	78	1050	280
CAY41	156	2090	480
Add. amount 1st configured servo-motor channel			2150

Communication module family	Bit memory (words)	Data (words)	Application Zone (words)
SCY21600 (Channel 0 UTW)	1	230	80
SCP111/ SCP112 / SCP114 (UTW) on SCY21600 (Channel 1 UTW)	1	450	40
Add. amount 1st configure channel UTW			1280

UC communication sub-module family	Bit memory (words)	Data (words)	Application Zone (words)
SCP111/ SCP112/ SCP114 (UTW) on UC (Channel 0 UTW)	1	60	580
FPP 20 on UC (Channel 0 UTW)	1	60	580
FPP 10 on UC (Channel 0 UTW)	1	40	870

Weighing family	Bit memory (words)	Data (words)	Application Zone (words)
AWY001	1	170	120
Add. cost 1st configured weighing channel			3920

8.4.4 Memory size of advanced functions

The following tables show for each advanced function (OF), the size of the code embedded in the application (application zone) when an advanced function is called.

The functions in the same family share the code (common code). This common code is embedded in the PLC on the **first call** of a function for that family. The code specific to a function is embedded on the **first call** for that function.

Example :

- First call of a function of the Numeric conversions family, ie DBCD_TO_DINT
Code embedded in the application zone :
 - Common code = 154 words
 - OF code DBCD_TO_INT = 149 words
- Call of another function of the Numeric conversions family, ie DINT_TO_DBCD
Code engaged in the application zone :
 - OF code DINT_TO_DBCD = 203 words
- Call of a function of the Numeric conversions family which has already been called (DBCD_TO_DINT or DINT_TO_DBCD) : no code embedded

Numeric conversions	OF	code size (in words)
Conversion of a 32-bit BCD number to 32-bit integer	DBCD_TO_DINT	145
Conversion of a 32-bit BCD number to 16-bit integer	DBCD_TO_INT	149
Conversion of a 32-bit integer to a 32-bit BCD number	DINT_TO_DBCD	203
Conversion of a 16-bit integer to a 32-bit BCD number	INT_TO_DBCD	75
	common code	154

Bit strings	OF	code size (in words)
Logic AND between two tables	AND_ARX	209
Copy of a bit table to a double word table	BIT_D	248
Copy of a bit table to a word table	BIT_W	205
Copy of a bit table to a bit table	COPY_BIT	146
Copy of a double word table to a bit table	D_BIT	196
One's complement in a table	NOT_ARX	157
Logic OR between two tables	OR_ARX	209
Copy a word table to a bit table	W_BIT	195
Exclusive OR between two tables	XOR_ARX	209
	common code	427

Word table instructions	OF	code size (in words)
Search in a table for 1st element equal to a value	FIND_EQW	75
Search in a table for 1st element greater than a value	FIND_GTW	75
Search in a table for 1st element less than a value	FIND_LTW	75
Search for the greatest value in a table	MAX_ARW	78
Search for the smallest value in a table	MIN_ARW	78
Number of occurrences of a value in a table	OCCUR_ARW	74
Left rotate a table	ROL_ARW	145
Right rotate a table	ROR_ARW	150
Sort a table (ascending or descending)	SORT_ARW	144
	common code	162

Double word table instructions	OF	code size (in words)
Search in a table for 1st element equal to a value	FIND_EQD	79
Search in a table for 1st element greater than a value	FIND_GTD	80
Search in a table for 1st element less than a value	FIND_LTD	80
Search for the greatest value in a table	MAX_ARD	95
Search for the smallest value in a table	MIN_ARD	95
Number of occurrences of a value in a table	OCCUR_ARD	78
Left rotate a table	ROL_ARD	163
Right rotate a table	ROR_ARD	170
Sort a table (ascending or descending)	SORT_ARD	178
	common code	162

Date, time and time periods	OF	code size (words)
Add a duration to a complete date	ADD_DT	519
Add a duration to a time of day	ADD_TOD	188
Convert a date to a string	DATE_TO_STRING	150
Day of the week	DAY_OF_WEEK	99
Difference between two dates	DELTA_D	374
Difference between two complete dates	DELTA_DT	547
Difference between two times of day	DELTA_TOD	110
Convert a complete date to a string	DT_TO_STRING	266
Remove a time period from a complete date	SUB_DT	548
Remove a time period from a time of day	SUB_TOD	186
Convert a time period to a string	TIME_TO_STRING	413
Convert a time of day to a string	TOD_TO_STRING	156
Format a time period as hours-minutes-seconds	TRANS_TIME	211
	common code	1703

Character string instructions	OF	code size (words)
Concatenation of two strings	CONCAT	224
Delete a substring	DELETE	279
Search for first different character	EQUAL_STR	212
Search for a substring	FIND	225
Insert a substring	INSERT	287
Extract from the left part of the string	LEFT	38
Length of a string	LEN	70
Extract a substring	MID	44
Replace a substring	REPLACE	365
Extract from the right part of the string	RIGHT	55
	common code	418

Orphee functions	OF	code size (words)
Shift to left on 32 with retrieval of shifted bits	DSHL_RBIT	152
Shift to right on 32 with sign extension, retrieval of shifted bits	DSHR_RBIT	152
Shift to right on 32 with filling of spaces by 0, retr. of shifted bits	DSHRZ_C	133
Shift to left on 16 with retrieval of shifted bits	WSHL_RBIT	91
Shift to right on 16 with sign extension, retrieval of shifted bits	WSHR_RBIT	103
Shift to right on 16 with filling of spaces by 0, retr. of shifted bits	WSHRZ_C	90
	common code	173
Up/down counting with indication of an under/overflow	SCOUNT	617

Application-specific functions

Process control functions	OF	code size (in words)
Mixed PID controller	PID	1800
Pulse width modulation of a numeric value	PWM	600
PID output stage for control of discrete valve	SERVO	1200
Management of CCX17 dedicated MMI for controlling PID loops	PID_MMI	4400
	common code	400

Man-Machine Interface Functions	OF	code size (in words)
Blocking entry of a variable on CCX17	ASK_MSG,	46.5
Blocking entry of a variable on message contained in CCX17	ASK_VALUE,	46.5
Dynamic assignment of keys	ASSIGN_KEYS,	46.5
LED control command	CONTROL_LEDS,	46.5
Display an alarm contained in CCX17	DISPLAY_ALARM,	46.5
Display a group of messages contained in CCX17	DISPLAY_GRP,	46.5
Display a message contained in CCX17	DISPLAY_MSG,	46.5
Multiple entry of a variable on CCX17	GET_MSG,	46.5
Multiple entry of a variable on message contained in CCX17	GET_VALUE,	46.5
Send a command to CCX17	PANEL_CMD,	46.5
Display an alarm message contained in the PLC memory	SEND_ALARM,	46.5
Display a message contained in the PLC memory	SEND_MSG	46.5
	common code	573

Communication Functions	OF	code size (in words)
Read base language objects	READ_VAR	617
Write base language objects	WRITE_VAR	500
Send/receive UNI-TE requests	SEND_REQ	438
Send and/or receive data	DATA_EXCH	375
Send a character string	PRINT_CHAR	476
Request read of a character string	INPUT_CHAR	625
Send and/or receive a character string	OUT_IN_CHAR	531
Send a telegram	SEND_TLG	219
Receive a telegram	RCV_TLG	172
Request to stop a communication function in progress	CANCEL	133
	common code	506
Shift 1 byte to the right of a byte table	ROR1_ARB	235

Axis Control Functions (1)	OF	code size (in words)
Automatic motion command	SMOVE	0

Explicit Exchange Functions (1)	OF	code size (in words)
Read status parameters	READ_STS	0
Read adjust parameters	READ_PARAM	0
Update adjust parameters	WRITE_PARAM	0
Save adjust parameters	SAVE_PARAM	0
Restore adjust parameters	RESTORE_PARA	0
Update command parameters	M WRITE_CMD	0

(1) Specific OF, the code is included in the volume of the I/O module.

8.5 Appendix : method of calculating the number of instructions

This method is used to calculate the number of basic Boolean or numerical instructions (assembler code).

Note : this is the method which was used to calculate the performance information given in section 1.3 part A.

Calculating the number of Boolean instructions

The number of each of the following elements is taken into account :

- Boolean operations : load (LD), AND, OR, XOR, ST, etc
- closing parentheses (or ladder convergences : vertical convergence links)
- comparison (AND[...], OR[...], etc) and operate ([...]) blocks

Do not count the NOT, RE and FE operators as Boolean instructions.

Example :

```
LD   %M0
AND( %M1
OR   %M2
)
ST   %M3
= 5 Boolean instructions
```

Calculating the number of numerical instructions

The number of each of the following elements is taken into account :

- assignments (:=)
- loading of first value after :=
- arithmetic instructions (+, -, *, /, <, =, etc), operations on words or word tables, double words, floating point values
- logic instructions on words
- (OF, EQUAL, etc) functions regardless of the number of parameters
- function blocks (or function block instruction)

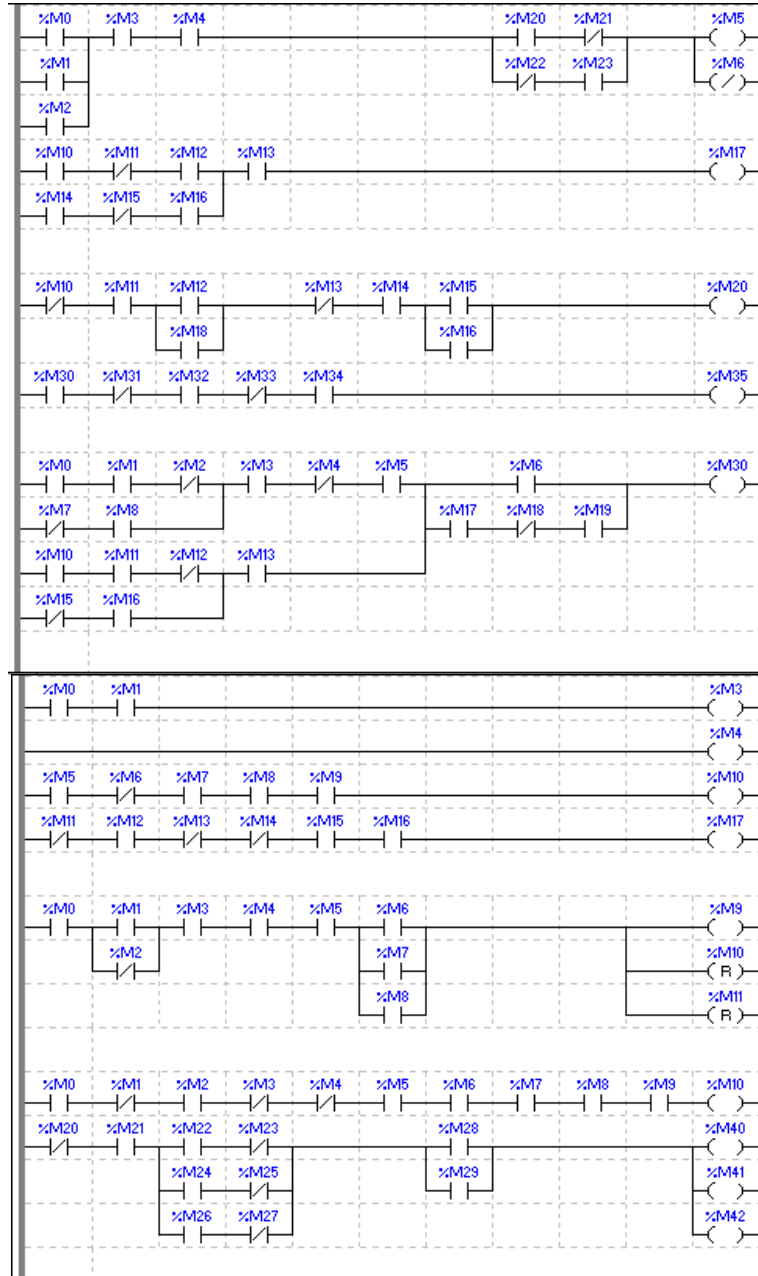
Example : %MW0:=(%MW1+%MW2)*%MW3;

instructions counted :

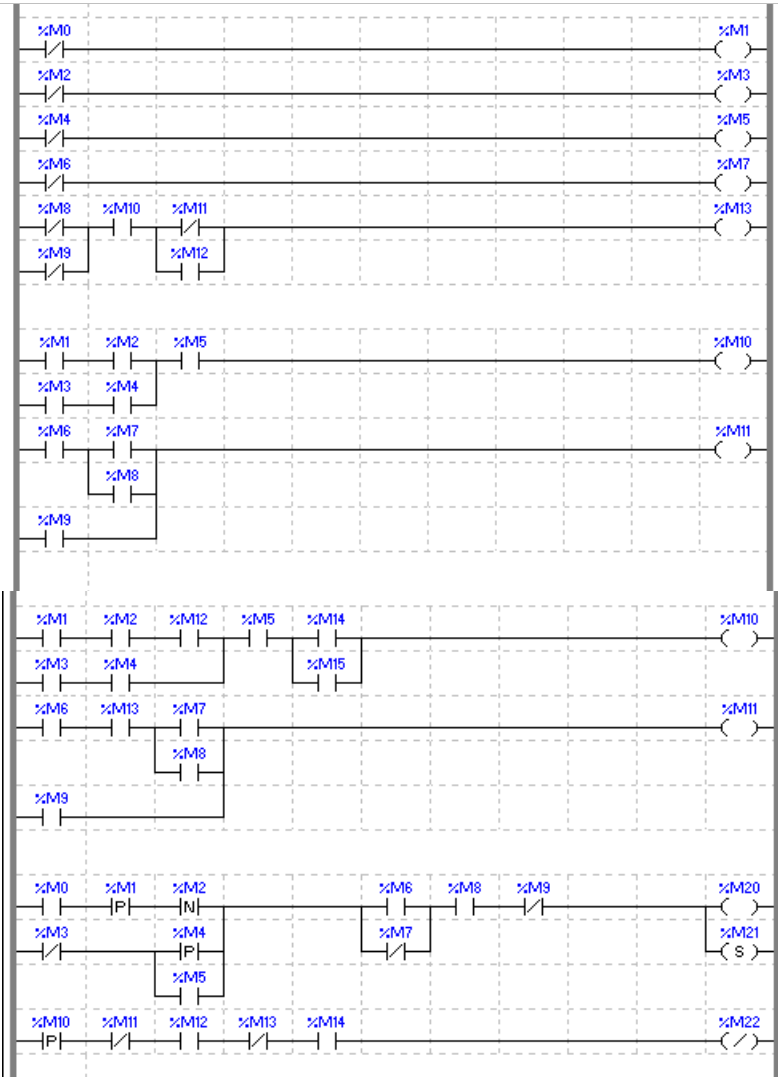
```
:=
  %MW1 (corresponds to the load instruction in the accumulator)
+
*
```

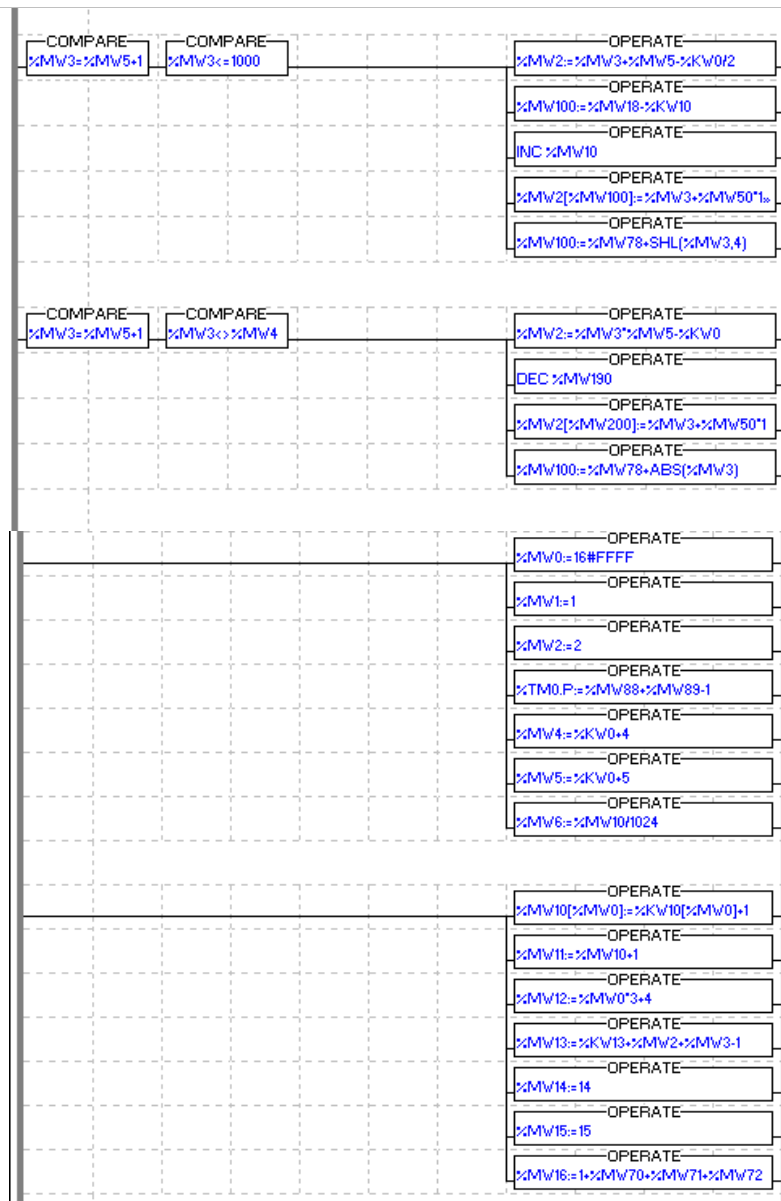
ie. 4 instructions.

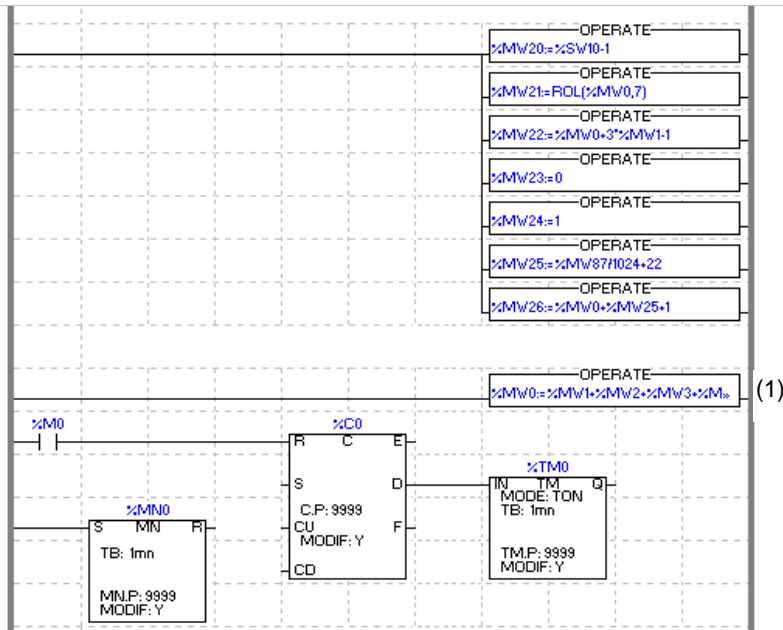
Example of application which is 65% de boolean and 35% numerical :



B







(1) : $\%Mw0 = \%Mw1 * \%Mw2 * \%Mw3 + \%Mw4 + \%Mw5 + \%Mw6 + \%Mw7 + \%Mw8 + \%Mw9 + \%Mw10 + 1$

Result

	Number of instructions	%	
Boolean without edge	187	54,05%	64,16%
Boolean with edge	4	1,16%	
Operation Block	31	8,96%	35,84%
Function Block	3	0,87%	
Single arithmetic (+,-,=,AND,...)	111	32,08%	
Indexed arithmetic	4	1,16%	
*,/	6	1,73%	
Immediates values	24		
Total number	346		100,00%

9.1 Index

Symbols

%Ci	B 1/14	B	
%DRi	B 2/9		
%Li	B 1/30	BCD <--> Binary conversion	B 2/24
%MNi	B 2/2	Binary --->ASCII conversion	B 2/45
%Ri	B 2/5	Bit memory	A 1/20, A 1/22
%Ti	B 2/13	Bit table	B 2/81
%TMi	B 1/10	Bit tables	A 1/16
*	B 1/23	BIT_D	B 2/83
+	B 1/23	BIT_W	B 2/83
-	A 1/10, B 1/23	Bits	A 1/6
/	B 1/23	BLK	A 3/8
:=	B 1/20	Boolean instructions	B 1/2
<	B 1/19	Byte	A 1/11
<=	B 1/19	C	
<>	B 1/19	CALL	A 2/3
=	B 1/19	Character string	A 1/16, B 2/42
>	B 1/19	Cold restart	A 1/31
>=	B 1/19	Comments	A 2/5, A 3/4, A 4/6, A 5/10
A		Common words	A 1/13
ABS	B 1/23	Comparison	B 1/19
Absolute value	B 1/23	Complement	B 1/25
Action zone	A 2/1	CONCAT	B 2/50
Actions	A 5/11	Concatenation of two strings	B 2/50
Add	B 1/23	Conditions	A 5/14
ADD_DT	B 2/69	Constant	A 1/11
ADD_TOD	B 2/70	Control structures	A 4/8
Addressing	A 1/7	Conversion	B 2/24
Addressing TSX 37 objects	A 1/7	COPY_BIT	B 2/81
Addressing TSX 57 objects	A 1/9	Cyclic execution	A 1/33
Alphanumeric comparisons	B 2/44		
AND	B 1/6, B 1/25		
AND convergences	A 5/2		
AND divergences	A 5/2		
AND_ARX	B 2/82		
ANDF	B 1/6		
ANDN	B 1/6		
ANDR	B 1/6		
Arithmetic on integers	B 1/23		
ASCII ---> Binary conversion	B 2/47		
ASCII --> Floating point conversion	B 2/49		
Assignment	B 1/20		

D		FIND_EQD	B 2/36
D_BIT	B 2/85	FIND_EQW	B 2/36
Date	B 2/63	FIND_GTD	B 2/36
DATE_TO_STRING	B 2/75	FIND_GTW	B 2/36
DAY_OF_WEEK	B 2/68	FIND_LTD	B 2/36
DEC	B 1/23	FIND_LTW	B 2/36
Decrement	B 1/23	Floating point	A 1/12, B 2/20
DELETE	B 2/51	Floating point ---> ASCII conversion	B 2/48
DELTA_D	B 2/72	FOR ... END_FOR	A 4/12
DELTA_DT	B 2/73	Forcing	A 1/23
DELTA_TOD	B 2/74	Freezing the Grafcet chart	A 5/20
Destination connector	A 5/3	Function block objects	A 1/15
DINT_TO_REAL	B 2/27	Function blocks	B 1/9
DINT_TO_STRING	B 2/45		
Direct coils	B 1/5	G	
Directed links	A 5/3, A 5/9	GET	B 2/8
Divide	B 1/23	Grafcet language	A 5/1
Double length	A 1/12	Grafcet objects	A 1/18, A 5/4
DOWN	B 1/12	Graphic elements, Grafcet	A 5/2
Downcounter	B 1/14	Graphic elements, Ladder language	A 2/2
Drum controller	B 2/9	Gray --> Integer conversion	B 2/28
DSHL_RBIT	B 2/87	GRAY_TO_INT	B 2/28
DSHR_RBIT	B 2/87		
DSHRZ_C	B 2/87	H	
DT_TO_STRING	B 2/76	HALT	B 1/33
E		I	
Edges	A 1/22, B 1/2	IF ... END_IF	A 4/8
END	B 1/32	Immediate values	A 1/13
END_BLK	A 3/8	INC	B 1/23
ENDC	B 1/32	Increment	B 1/23
ENDCN	B 1/32	Index overrun	A 1/18
EQUAL	B 2/35	Indexation	A 1/17
EQUAL_STR	B 2/60	Indexed objects	A 1/17
Event masking/unmasking	B 1/34	INSERT	B 2/52
Event-triggered tasks	A 1/40	Instruction list language	A 3/1
Exclusive OR	B 1/8	INT_TO_REAL	B 2/27
EXIT	A 4/13	INT_TO_STRING	B 2/45
		Integer <--> Floating point conversion	B 2/26
F		Internal words	A 1/11
Falling edge contacts	B 1/4	Inverse coils	B 1/5
FAST	A 1/38		
Fast task	A 1/38		
FIFO stack	B 2/5		
FIND	B 2/61		
Find functions on tables	B 2/36		

J

JMP B 1/30
 JMPC B 1/30
 JMPCN B 1/30
 Jump B 1/30

L

Label A 2/5, A 3/4, A 4/7
 Ladder language A 2/1
 LD B 1/4
 LDF B 1/4
 LDN B 1/4
 LDR B 1/4
 LEFT B 2/58
 LEN B 2/62
 LIFO stack B 2/5
 Logic AND B 1/6
 Logic OR B 1/7, B 1/25
 Logic shift B 2/19

M

MASKEVT B 1/34
 Master task A 1/38
 MAX_ARD B 2/38
 MAX_ARW B 2/38
 Memory card A 1/20
 MID B 2/56
 MIN_ARD B 2/38
 MIN_ARW B 2/38
 Monostable B 1/11, B 2/2
 MPP A 3/7
 MPS A 3/7
 MRD A 3/7
 Multiply B 1/23
 Multitask A 1/37

N

N/C contacts B 1/4
 N/O contacts B 1/4
 NOT B 1/25
 NOT_ARX B 2/82
 Numerical expression B 1/27

O

Objects which can be symbolized A 1/19
 OCCUR_ARD B 2/39
 OCCUR_ARW B 2/39
 Operating modes A 1/29, A 5/1
 OR B 1/7, B 1/25
 OR convergences A 5/2
 OR divergences A 5/2
 OR_ARX B 2/82
 ORF B 1/7
 ORN B 1/7
 ORR B 1/7
 OUT_BLK A 3/8
 Overflow B 1/24

P

Parentheses A 3/5
 Periodic execution A 1/34
 Post-processing A 5/23
 Power outage A 1/29
 Pre-processing A 5/18
 Program instructions B 1/28
 PTC B 2/67
 PUT B 2/8

R

R B 1/5
 Read day of the week B 2/68
 Read system date B 2/66
 REAL_TO_DINT B 2/27
 REAL_TO_INT B 2/27
 Register B 2/5
 REM B 1/23
 REPEAT ... END_REPEAT A 4/11
 REPLACE B 2/54
 RESET B 2/8, B 2/11
 Reset coils B 1/5
 RET B 1/29
 RETCN B 1/29
 RIGHT B 2/58
 Rising edge contacts B 1/4
 ROL_ARD B 2/40
 ROL_ARW B 2/40
 ROR_ARD B 2/40

ROR_ARW	B 2/40	Time of day	B 2/63
Rotate shift	B 2/19	Time period	B 2/63
RRTC	B 2/66	TIME_TO_STRING	B 2/77
Rung	A 2/4	Timer	B 1/10, B 2/13
Rungs, execution	A 2/11	TOD_TO_STRING	B 2/78
S		TRANS_TIME	B 2/80
S	B 1/5	Transitions	A 5/2
SCOUNT	B 2/90	TSX 37-10 memory	A 1/25
Sequence	A 3/4	TSX 37-21/22 memory	A 1/26
Sequence selection	A 5/6	TSX 57-10 memory	A 1/27
Sequential processing	A 5/21	TSX 57-20 memory	A 1/28
Set coils	B 1/5	U	
Simultaneous step activation	A 5/6	UNMASKEVT	B 1/34
Single task	A 1/32	UP	B 2/11
Sort function on tables	B 2/41	Upcounter	B 1/14
SORT_ARW	B 2/41	Update system date	B 2/66
SORT_ARD	B 2/41	User memory	A 1/20
Source connector	A 5/3	V	
SQRT	B 1/23	Vertical comparison	B 2/17
Square root	B 1/23	W	
SRI	A 1/32	W_BIT	B 2/85
ST	B 1/5	Warm restart	A 1/30
START	B 1/12, B 2/3	Watchdog	A 1/36
Steps	A 5/2	WHILE ... END_WHILE	A 4/10
STN	B 1/5	Word extract bits	A 1/13
Stop code	B 2/67	Word memory	A 1/20, A 1/24
STRING_TO_DINT	B 2/47	Word tables	A 1/16, B 2/29
STRING_TO_INT	B 2/47	WRTC	B 2/67
STRING_TO_REAL	B 2/49	WSHL_RBIT	B 2/87
Structured data	B 4/4	WSHR_RBIT	B 2/87
Structured Text language	A 4/1	WSHRZ_C	B 2/87
SUB_DT	B 2/69	X	
SUB_TOD	B 2/70	XOR	B 1/8, B 1/25
Subroutine	A 1/32, B 1/28	XOR_ARX	B 2/82
Subroutine return	B 1/29	XORF	B 1/8
Subtract	B 1/23	XORN	B 1/8
SUM	B 2/34	XORR	B 1/8
Symbol	A 1/4, A 1/19		
System bits	B 3/1		
System words	B 3/7		
T			
Table comparison	B 2/35		
Test zone	A 2/1		
Time	B 2/63		