

MasterTool® Programming

PONTO Series Programming

Rev. F 01/2005
Cód. Doc.: MP399602



altus

No part of this document may be copied or reproduced in any form without the prior written consent of ALTUS Information Systems S.A. who reserve the right to carry out alterations without advice.

According to legislation in force in Brazil, the Consumer Defence Code, we are giving the following information regarding personal safety and installation by the client.

The **industrial automation equipment**, built by ALTUS are strong and reliable due to the stringent quality control it is subjected to. However the electronic industrial control equipment (programmable controllers, numerical commands, etc.) can cause damage to the machines or processes through their controllers when there are defective components and programming or installation errors. This can even put human lives at risk.

The user should consider the possible consequences of the defects and should provide additional external installations for security so that, if necessary, the security of the system can be maintained especially during the initial installation and testing.

It is essential to completely read the manuals and/or about the technical characteristics of the product before it's installation or use.

ALTUS guarantee their equipment against genuine production faults for a period of twelve months starting from the shipping date. This guarantee is given in terms of factory maintenance, that is to say, the transportation costs of returning to factory will be borne by the client. The guarantee will be automatically suspended where there are modifications introduced to the equipment by personnel not authorized by ALTUS. ALTUS are exempt from any responsibility with regard to repairs or replacement parts owing to faults created by outside influences, through inappropriate use, as well as the result of accidents or force majeure.

ALTUS guarantees that their equipment works in accordance with the clear instructions contained in their manuals and/or the technical characteristics, not guaranteeing the success of any particular type of application of the equipment.

ALTUS does not acknowledge any other guarantee, direct or implied, principally when it is dealing with supply of third parties.

Requests for additional information about the supply and/or characteristics of the equipment and ALTUS services should be put in writing. The address for ALTUS can be found on the back cover. ALTUS is not responsible for supplying information about their equipment without formal registration.

COPYRIGHTS

MASTERTOOL and QUARK are the registered trademarks of ALTUS Information Systems S.A.

IBM is the registered trademark of the International Business Machines Corporation.

Summary

1. PREFACE.....	6
Description of this Manual	6
Documents of Ponto Series	6
Terminology	7
Conventions Used.....	8
Technical Support.....	9
Issues of this manual.....	10
2. INTRODUCTION	11
Programming Language.....	11
3. DIAGRAMS OF RELAYS LANGUAGE.....	12
Elements of Programming.....	12
Ponto Series Memory Organization	12
Logics	13
Operands.....	14
Identifying an Operand through Address	14
Identification of an Operand through Tag	14
Operands Used on MasterTool	15
Identification of Simple Operands	15
Identification of Constants Operands.....	16
Identification of table Operands	17
Operands %E – Input Relays.....	18
Operands %S – Output Relays	18
Operands %A – Auxiliary Relays.....	19
Operands %M - Memories	19
Operands %D - Decimals.....	20
Operands %F – Reals.....	21
Operands %I - Integer	21
Operands %KM, %KI, %KD e %KF - Constants	22
Operands %TM, %TI, %TD e %TF - Tables	23
Indirect Access	24
Declaration of Operands	25
Retentive Operands.....	26
Instructions	27
Restrictions Using Instructions on the PLCs.....	28
Graphic Representation of the Instructions	29
Description of Syntax Instruction	30
Restrictions in Positioning the Instructions.....	30
Programming Project	32
Structure of a Programming Project.....	32
Operating Status of the PLC.....	35
Execution of the Programming Project	37
Elaboration of the Programming Project.....	39
Depuration of Programming Projects	43
Program Execution Cycle Times	51
Protection Levels of the PLC	52

Interlocking of Commands in the PLC	53
4. INSTRUCTIONS.....	55
List of Instructions.....	55
Conventions Used.....	55
Instructions of the Relays Group	58
Contacts.....	59
Coils 60	
SLT – Jump Coil.....	61
PLS – Pulse Relay.....	63
RM, FRM – Master Relay, End of Master Relay	64
Instructions of Moving Group	65
MOV – Moving Simple Operands.....	66
MOP – Moving of parts (Subdivisions) of Operands	67
MOB – Moving of Blocks of Operands.....	69
MOT – Moving of Tables	71
CAB – Load Block	73
Arithmetic group Instructions.....	78
SOM - Sum.....	78
SUB - Subtraction.....	80
MUL - Multiplication.....	81
DIV - Division.....	82
AND – And binary between operands	83
OR – Or binary between operands.....	85
XOR – Or Exclusive between operands.....	87
CAR – Load Operands	89
Instructions of Comparison of Operands – Equals, More than and Less than.....	90
Instructions of counters group	93
CON – Simple Counter	94
COB – Bidirectional Counter	95
TEE – Timer to turn on	96
TED – Timer to turn off	97
Instructions of the Conversion Group	98
B/D - Conversion Binary-Decimal	99
D/B - Conversion Decimal-Binary	100
Instructions of the General Group	101
LDI – Connect/Disconnect indexed.....	102
TEI – Test of Indexed Status	104
SEQ - Sequencer.....	106
CHP – Call the Procedure Module.....	111
CHF – Call Function Module	112
ECH – Write of Operands on Another PLC for Ethernet.....	115
LTH – Reading of Operands from Another PLC for Ethernet	120
LAH – Free Updated Images Operands for Ethernet.....	122
Instructions of the Connections Group	123
LGH – Horizontal Connection	123
LGN – Denied Connection.....	123
LGV – Vertical Connection.....	123
5. FUNCTION MODULES	124
F-PID.033 – PID Control Function	125
Introduction	125
Programming.....	126
F-RAIZN.034 – Square Root Function.....	130

Introduction	130
Programming	130
F-ARQ2.035 to F-ARQ31.042 – Functions Data File.....	132
Introduction	132
Programming	132
F-MOBT.043 – Function for Moving Blocks from Table Operands	136
Introduction	136
Programming	136
F-RELG.048 – Function to Access the Real Time Clock	138
Introduction	138
Programming	138
F-PID16.056 – F Module for PID Control.....	140
Introduction	140
Programming	142
Operands	142
Inputs and Outputs	142
Functioning Characteristics	143
Unsaturation of the Integral Action	143
Manual Mode.....	143
Direct and Reverse Control	143
Sampling Interval.....	143
Execution Time.....	144
Table Position Parameters Description	144
Description of %A Operand Control	145
Application Notes	146
PID Controller Adjustments Suggestions.....	149
Determination of the Constants of the Controller Through the Period and Critical Gain.....	149
Determination of the Constants of the Controller Through the Constants of the Process.....	150
Gains X Scales.....	152
Example of Application	154
Uses of F-PID16.056	157
Comparison with F-PID.033	157
F-CTRL.059 – F Module for Advanced Control.....	159
Introduction	159
Programming	162
F-NORM.071 – Function to Normalization.....	164
Introduction	164
Programming	164
F-COMPF.072 – Function for Multiple Comparisons.....	166
Introduction	166
Programming	166
F-AES.087 – Inputs and Outputs Immediate Update Function	168
Introduction	168
Programming	168
F-ANDT.090, F-ORT.091 and F-XORT.092 – Function Logical Operations between Table Operands	170
Introduction	170
Programming	170
F-STCP.044 – CPU Status Function.....	172
Introduction	172
Programming	172
F-NEGT.093 – Function for the logic denial of Table Operands	177
Introduction	177
Programming	177
6. GLOSSARY	179

Glossary to Ponto Series	179
Network Glossary	179
General Glossary	180
Acronyms	183

1. Preface

Description of this Manual

This manual presents the programming language used on ALTUS Ponto Series programmable controllers, and orientations on implementing application programs. It was written assuming a familiarity with the use of standard IBM PC[®] microcomputers and Windows[™] operating environment.

The software programmer MasterTool Programming MT4000 or MT4100 referred to from now on as MasterTool[®] was developed for programming in the relay and blocks language of the programmable controller ALTUS Ponto Series.

This manual is divided into 4 chapters and glossary.

Chapter 1, **Introduction**, present the basic characteristics of ALTUS Ponto Series programming.

Chapter 2, **Diagrams f Relays Language**, show this language components.

Chapter 3, **Instructions**, describe the function and syntax of all instructions.

Chapter 4, **Function Modules**, describe the function and the programming parameters of input and output of the ALTUS function modules.

Documents of Ponto Series

To obtain additional information about Ponto Series can be used other documents (manuals and technical characteristics) besides this one. This documents can be found on www.altus.com.br.

Each product have a document named as Technical Characteristic (CT), in this document can be found the characteristics of the product. If the product have more informations, it can have a user's manual too (the code of the manual is cited on CT)

For example, the module PO2022 have all it characteristics informations, uses and buying, on it CT. On the other side, the PO5063 have, besides the, an user's manual.

Advices the following documents as additional information:

Technical characteristics of each product

- User's Manual of the Master Tool Programming
- User's Manual of the Ponto Series

Terminology

In this manual the words “software”, “hardware”, “mouse”, “tag” and “wire-info” are used freely, in general and frequently. For this reason, despite their being English words, they appear without inverted commas.

The following expressions are employed with frequency on the manual text. So, the need of being known to a better understanding.

- **PLC**: Programmable Logical Controller – understood as an equipment with an PLC, input and output modules and power supply.
- **CPU**: Central Processing Unit, is the main module of the CP, it process the data.

The word “module”, when referring to hardware, is used to determinate each component of an equipment.

The word “module”, when referring to software, is used to determinate each component of an application program.

Other expressions can be found on Appendix A, **Glossary**.

Conventions Used

The symbols used throughout this manual have the following significance:

- This mark indicates a list of items or topics

CAPITAL LETTER indicate names of keys, for example ENTER.

KEY 1 + KEY 2 is used for keys which have to be pressed simultaneously. For example, the simultaneous pressing of keys CTRL and END is indicated by CTRL + END.

KEY 1 , KEY 2 is used for keys which have to be pressed sequentially. For example, the message “Press ALT, F10” indicates that the ALT key should be pressed and freed and then the F10 key pressed and freed.

CAPITAL LETTERS indicate file names and folder names.

Italics indicate words and characters which are keyed in on the keyboard or viewed on screen. For example, if you are asked to key in A: *MasterTool* these characters are keyed in exactly as they appear in the manual.

BOLD-FACED TYPE is used for names of commands or options, or for emphasizing important parts of the text.

Warning messages have the following format and significance.

DANGER:

The label DANGER indicates a risk to life, serious harm to people or that substantial material damage may happen if the necessary precautions are not taken.

WARNING:

The label WARNING indicates that harm to people or minimal material damage can happen if the necessary precautions are not taken.

ATTENTION:

The label ATTENTION indicates a risk to life, of serious harm to people or that substantial material damage **can happen** if the necessary precautions are not taken.


Technical Support

Any questions about the product should be directed to ALTUS support service. The address and telephone number can be found on the back cover. Or on the internet:


- 1 www.altus.com.br
- 2 E-MAIL: altus@altus.com.br

In the event of the equipment already being installed, it is advisable to provide the following information before getting in contact:

- 3 Models of used equipments and configuration of the installed system
- 4 Serial number of the CPU, the equipment revision and the version of the executive software, on the label on the equipment

Information about the status of the PLC, available through the command **Communication, Status**, option information about MasterTool programmer or selecting the button 

- 5 Modules of the applicative program, obtained through the MASTERTOLL programmer

Version of MasterTool programmer, which can be obtained starting with command **Help, About MasterTool** or selecting the button 

Issues of this manual

The reference code, of the issue and the date of the current manual is indicated on the cover. A change in the issue can mean alterations to the functional specification or improvements to the Manual.

The following is an account of the corresponding alterations to each issue of this Manual.

Revisão: A	Data: 10/2004
Approval: Luiz Gerbase	
Author: Jean Schmith	

Observations:

- First issue of this manual

Revisão: F	Data: 01/2005
Approval: Luiz Gerbase	
Author: Jean Schmith	

Observations:

- Sincronization of the Portuguese and English versions of the manuals.

2. Introduction

Welcome to ALTUS language of Relays and Blocks, a language which allows constructing application programs for ALTUS PLCs with MasterTool Programming.

The applications program's objective is the execution of control tasks. This program, when loaded into the programmable controller (PLC), makes this pass to exercise the control functions of the machine or process which is being programmed.

Programming Language

Programmable controllers came to replace relay control panels. In this context, a programming language which approaches it more from the experience of technicians and engineers will be a more adequate solution for the development of PLC's applications programs.

In view of this, the available instructions for construction of the applications in MasterTool are programmed in a language of relays and blocks, very similar to language of electrical contacts and bobbins, used in the description of the relay control panels.

The main advantage of using this type of language is its quick learnship, since it is very much like conventional electrical outlines.

The accompaniment and verification of the functioning of on applications program is similar to the electrical outline, with the advantage of visualizing the status of the contacts and reels in the MasterTool window.

3. Diagrams of Relays Language

This chapter describes the ALTUS Relays and Blocks language. It detailing those elements of the language, the modular structure of an applications program and the function of each module.

After reading this chapter it will be possible to structure an applications program as well as carry out the configuration of the PLCs and router devices.

Elements of Programming

An applications program is made up of 4 basic elements:

- modules
- logics
- instructions
- operands

An applications program is composed of different **modules**, allowing a better structure for the routines according to its functions. The modules are programmed in the language of relays, following the global tendency for Normalization in this area.

A module of an application program is divided into **programming logics**. The format of an application program logic use don Ponto Series CLPs allows the maximum of eight elements in series and up to four elements in parallel.

The **instructions** are used to execute determined tasks for the environment of readings and for alterations to the value of the operands.

The **operands** identify different types of variables and constants used in the elaboration of an applications program, being able to have its value changed according to the program carried out. An example of variables are points of I/O and memory counters.

Each component element of the applications program is explained in detail in the following sections.

Ponto Series Memory Organization

The applications program is stored in the controller in an area of memory divided into **banks**. There can exist one or more RAM and EPROM memory banks, according to the model of the PLC and its memory configuration, each bank having 16, 32 or 64 Kbytes.

In this manual, in the MasterTool help and in MasterTool programmer, the name EPROM refers indistinctly to memory for permanent recording of the application program used in the PLC, that is to say of type EPROM cartridge or EPROM flash.

In the directory window of the PLC's modules (options **Communication, Modules**) it is possible to visualize the quantity of free memory in each bank, for each type existing in the controller. C.f. **Modules Option** in the section **Communication Command** in chapter 4.

The values of the numeric operands (% M, %D, %F, %TM, %TD and %TF) are stored in a separate area of the program, with different sizes according to the model of PLC. The amount of operands memory free can be checked in the editing window of module C in the operands panel. For further information about the **Editing Window** of module C, c.f. section Editing Windows, in chapter 3 of the User's Manual.

The binary operands (%E, %S and %A) have area permanently reserved for their values in the internal memory of the microprocessor.

The use of memory operands is shown in detail in the section **Declaration of Operands**, in the same chapter.

For further information about the capacities and memory organization of each controller, consult their respective Users Manuals (c.f. section **Related Manuals**, in the preface of this manual).

Logics

The word **logic** refers to a programming matrix made up of 32 cells (matrix elements arranged in four lines 0 to 3 to 8) columns (0 to 7). Instructions can be placed in each one of these cells, being possible to program up to 32 instructions in the same logic.

Each logic present to the program, simulates a short part of a real diagram of relays. Figure 2-1 shows the format of an applications program logic.

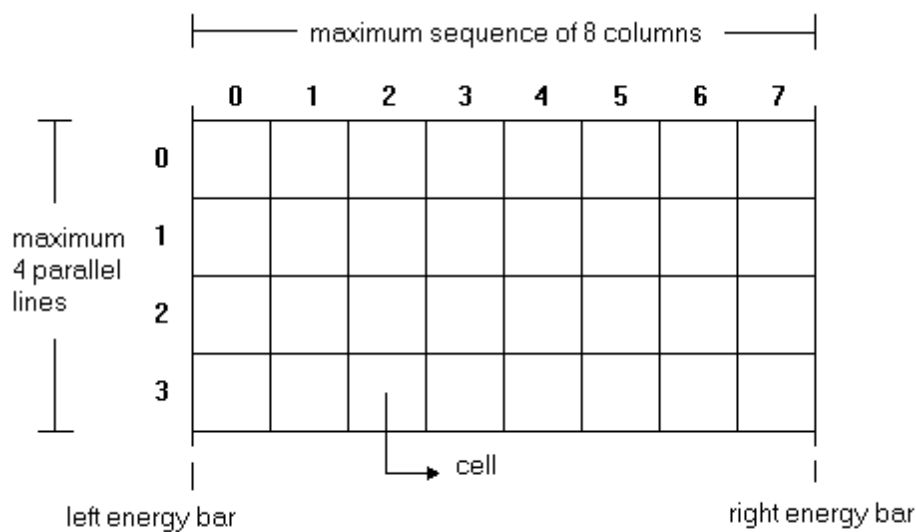


Figure 2-1 Logic Format

The two lateral lines of the logic represent energy bars between the instructions placed for execution.

Symbolic instructions usually found in diagrams are available for programming, such as contracts, coils, connections and instructions shown in boxes as timers, counters and arithmetics.

The logic should be programmed in a format which reel and inputs of instructions from boxes may be “powered” starting from the closure of a flow of “current” from the left to the right between the two bars, through the contacts or from the outputs of interconnected boxes. However, the flow of “electrical current” simulated in a logic flows only in the sense of from an energy bar on the left to the right, different from the real electrical outlines. The concept used simplifies very much the logic project of relays, once that is not necessary to be concerned with the escape paths of current.

The processing of the instructions of a logic carried out in columns, from column 0 to 7. One column is processed in the sequential order of its lines, from line 0 to line 3. Figure 2-2 shows the processing order of the logic cells. The number existing in each cell indicates its order in the processing.

	0	1	2	3	4	5	6	7
0	1	5	9	13	17	21	25	29
1	2	6	10	14	18	22	26	30
2	3	7	11	15	19	23	27	31
3	4	8	12	16	20	24	28	32

Figure 2-2 Processing order of the Logic Cells

Operands

Operands are elements used for MasterTool instructions in the elaboration of an applications program. The operands can define constant values, defined at the time of programming, or variables, identified through an address or tag, with values able to be changed during the execution of an applications program.

Identifying an Operand through Address

The identification and use of an operand through its address is characterized through character % as first character of the name. The rest of the name used should follow the rules for forming the addresses of operands.

The format of each operand can be seen in the section **Identification of Simple Operands** and in the subsequent sections, in this same chapter.

Identification of an Operand through Tag

The identification and use of an operand through its tag is characterized through use of a name, with up to 7 characters (alphanumeric), which can be attributed to any operand, except constants. This name passes to represent the operand in the processes of programming, monitoring, purifying and documentation of an applications program.

MasterTool does not allow the use of TAGs for operands of the type constant (%KM or %KD).

E.g.:

Attribute the tag **CONT1** to the operand **%M0000**. Always when the operand **%M000** needs to be used in the editing of the applications program, it can use its tag **CONT1**.

☺HINT:

The choice of name tag for the operand should reflect at the most the function which the contents of the operand executes in the applications program.

E.g.: **TANK 1**, stores the volume of tank 1.

The identification of an operand through its address can always be done, once the whole operand has an address. The identification of an operand through its tag, can only be achieved after attributing the tag to an operand.

The attributing of tags to operands can be achieved through the command **Operands** from the menu **Report** or directly at the time of programming. In the second case, to fill in the name of an instruction operand with a non-existent tag, indicates the non-existence of a tag definition, and asks which type of operand the tag should be created for.

For further information about creating and attributing tags to operands, c.f. sections about the command **Report, Operands**, on the chapter 4 and **Inserting Tags and Comments for Operands**, on the User's Manual of MasterTool.

The operands can also be visualized through their associated wire-info, However, an operand cannot be forced or monitored by keying in the wire-info instead of the tag or address.

Operands Used on MasterTool

The operand available in MasterTool are shown in table 2-1:

Type	Operand
%E	Input Relays
%S	Output Relays
%A	Auxiliary Relays
%M	Memorys
%I	Integers
%D	Decimals
%F	Reals
%KM	Memory Constants
%KD	Decimals Constants
%KF	Reals Constants
%TM	Memory Tables
%TI	Integer Tables
%TD	Decimals Tables
%TF	Reals Tables

Table 2-1 Operands Used in MasterTool

The operands are divided into 3 groups:

- simple operands
- constant operands
- table operands

Identification of Simple Operands

The simple operands are used with variables of storing the values in the applications programs. According to the instruction which they use, they can be referenced in full or in a subdivision (one part of the operand). The subdivisions of operands can be **word**, **octet**, **nibble** or **point**.

The general format of a simple operand can be seen in figure 2-3.

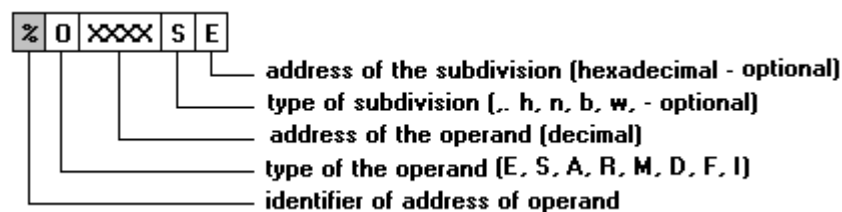


Figure 2-3 Format of simple operand

Operand type:

%E - input

%S - output

%A - auxiliary

%M – memory

%I - integer

%D - decimal

%F - real

Subdivision type:

. – point of box word (1 point)

h - point of high word (1 point)

n - nibble (4 point)

b - octet (8 point)

w - word (16 point)

Examples of Addresses:

%E0002.3 – point 3 of the input operand 2

%S0004.7 – point 7 of the output operand 4

%A0039n1 - nibble 1 of the auxiliary operand 39

%A0045 - auxiliary octet 45

%I0234 –integer operand 234

%M0205 - memory operand 205

%M0205b0 - octet 0 of the memory 205

%D0029 - decimal operand 29

%D0034w1 - word 1 of the decimal operand 34

%F0001 – real operand 1

Tags examples:

FORNO

LIMSUP

CHAVE1

Identification of Constants Operands

The constant operands are used to define the fixed values during the editing of an applications program.

The general format of a constant operand can be seen in figure 2-4.

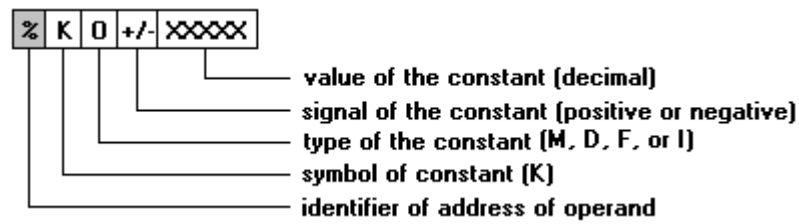


Figure 2-4 Format of a constant Operand

Constant type:

%M memory

%I integer

%D decimal

%F real

Examples:

%KM+05172 – memory positive constant

%KI-1 – integer negative constant

%KD-0974231 – negative decimal constant

%KF+0153.78 – positive real constant

Identification of table Operands

Tables of Operands are groups of simple operands set out in one dimensional arrays. Indices are used to determine the position of the table is required to be read or altered. Memory, integer, decimal and real operand tables are possible.

The general format of an operand table can be seen in figure 2-5.

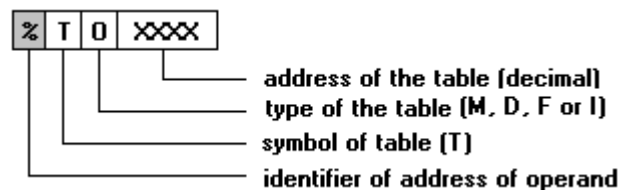


Figure 2-5 Format of a table Operand

Table type:

%TM memory

%TI integer

%TD decimal

%TF real

Examples:

%TM0026 – memory table 26

%TI0020 – integer table 20

%TD0015 – decimal table 15

%TF0069 – real table 69

Operands %E – Input Relays

Operands used to reference points of digital modules of input. Their quantity is determined through the number of I/O modules which are arranged behind the scenes of the system. C.f. item

Configuring the Bus in the section **Configuring the Module C** on the MasterTool User's Manual.

The operands %E are normally used in binary instructions (contacts, reels) and for movement. They use up one byte of memory (8 bits), storing the values of the points directly in each bit. The values of the operands are stored in the internal memory of the microprocessor, not using the space available in the applications program.

The formats of the operands %E can be seen in figure 2-6.

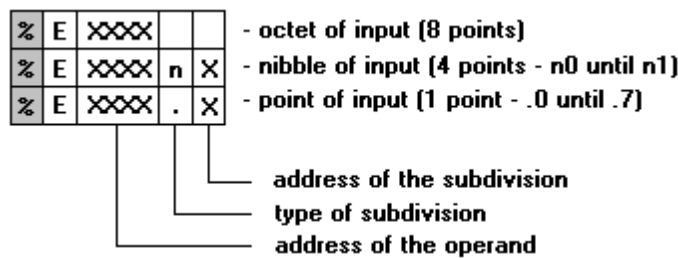


Figure 2-6 Format of Operands %E

Examples:

%E0018.6 - point 6 of the input octet 18

%E0021n0 - nibble 0 of the input octet 21

%E0025 – input octet 25

Operands %S – Output Relays

Operands are used to reference points of digital modules of output. Their quantity is determined through the number of I/O modules which are arranged behind the scenes in the system. C.f. item

Configuring the Bus in the section **Configuring the Module C** on the MasterTool User's Manual.

The operands % are used in binary instructions (contacts, reels) and for movement. They use up one byte of memory (8 bits), storing the values of the points directly in each bit. The values of the operands are stored in the internal memory of the microprocessor, not using the available space of the applications program.

The format of the operands can be seen in figure 2-7.

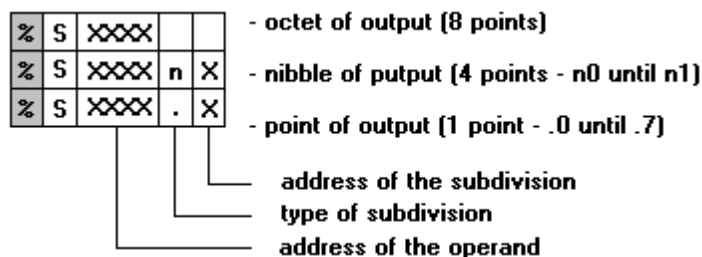


Figure 2-7 Format of Operands %S

Examples:

%S0011.2 - point 2 of the output octet 11

%S0010n1 - nibble 1 of the output octet 10

%S0015 – output octet 15

Operands %A – Auxiliary Relays

The auxiliary relays are operands used to store and manipulate the intermediate binary values in the processing of the applications program. Their quantity in the controllers is fixed (c.f. section **Declaration of the Operands** in this same chapter).

Operands %A are used in binary instructions (contacts, coils) and for movement. They use up one byte of memory (8 bits), storing values directly in each bit. The values of the operands are stored in the internal memory of the microprocessor, not using the space available to the applications program.

The formats of the Operands %A can be seen in figure 2-8.

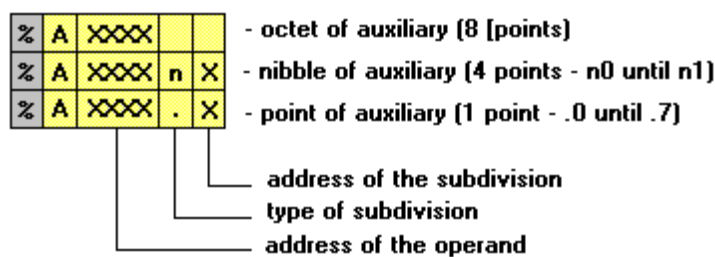


Figure 2-8 Formats of Operands %A

Examples:

%A0032.7 - point 7 of the auxiliary output 32

%A0087n1 - nibble 1 of the auxiliary output 87

%A0024 – auxiliary octet 24

Operands %M - Memories

The operands %M are used for numerical processing, storing values in simple precision, with signal.

The formats of the operands %M can be seen in figure 2-10.

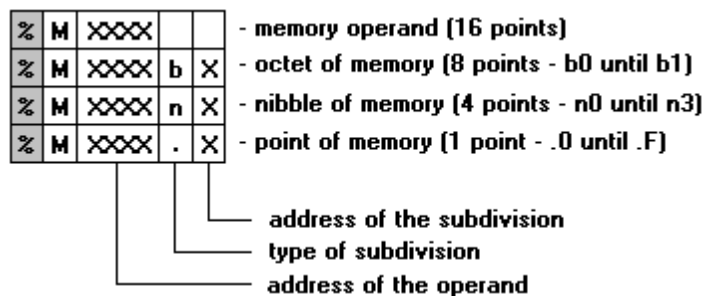


Figure 2-9 Formats of the Operands %M

The quantity of memory operands is configurable in the declaration of the module C, being the maximum limit depending on the PLC model in use (c.f. section **Declaration of Operands** in the same chapter).

The operands %M are used in instructions of movement, comparison, arithmetic, counting, timing and conversion. They can be used in contacts, for the same form as the operands %E, %S and %A.

These operands use up two bytes of memory (16 bits) storing the value in two complement from (2') according to figure 2-10.

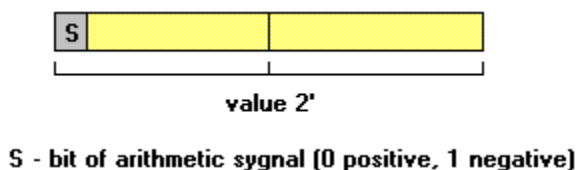


Figure 2-10 Format of the Memory Operand

Examples:

%M0032 - memory 32

%M0072n1 - nibble 1 of the memory 72

%M0084.F - point 15 of the memory 84

Operands %D - Decimals

The operands %D are used for numerical processing, storing values in BCD format with up to 7 digits and signal.

The formats of the operands %D can be seen in figure 2-11.

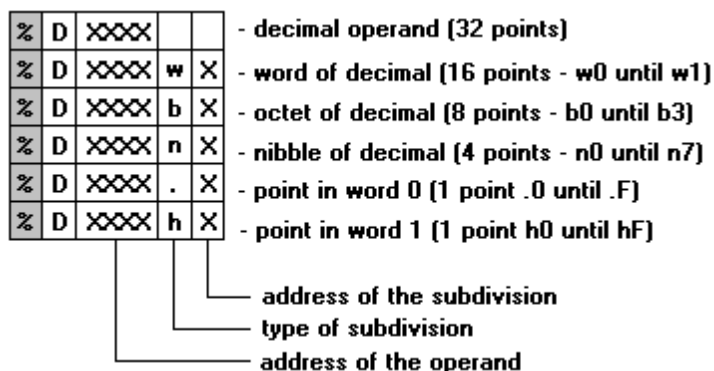


Figure 2-11 Formats of the Operands %D

The quantity of decimal operands is configurable in the declaration of module C, being the maximum limit depending on the PLC model being used (c.f. section **Declaration of Operands** in the same chapter).

The operands %D are used in instructions of movement, comparison, arithmetic and conversion. They can be used in contacts, in the same form as the operands %E, %S and %A. These operands use up four bytes of memory (32 bits), storing the value in the format BCD (each digit occupies 4 bits), with signal, according to figure 2-12.

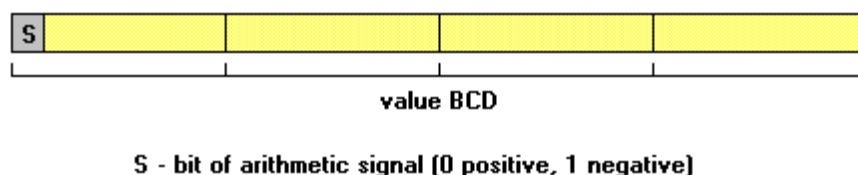


Figure 2-12 Format of the Operand Decimal

Examples:

%D0041 - decimal 41

%D0023b2 - octet 2 of the decimal 23

%D0059n6 - nibble 6 of the memory 59

%D0172hA - point 10 of the word 1 of the memory 172

Operands %F – Reals

The formats of the operands %F can be seen on the following figure:

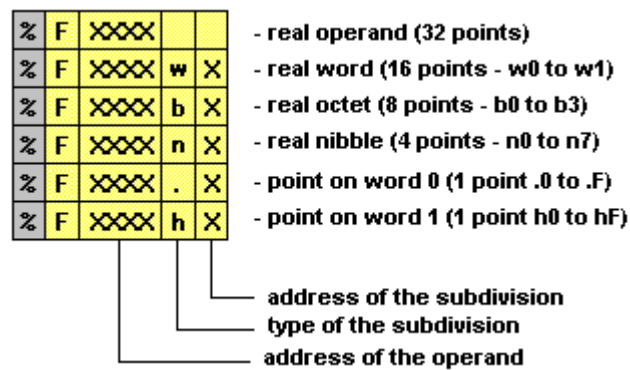


Figure 2-13 Formats of the Operands %F

The quantity of real operands is configurable in the declaration of module C, being the maximum limit depending on the PLC model being used (c.f. section **Declaration of Operands** in the same chapter).

The operands %F are used to the numeric processing, storing values in 32 bits with floating point, simple precision and signal, as IEEE 754. These operands use four bytes of memory (32 bits), storing the value as the following figure:



Figure 2-14 Formats of the Operand Real

The value of a real operand (%F) is obtained as the following expression:

$$\text{Value} = \text{Signal} \times 1, \text{Mantissa} \times 2^{(\text{Exp} - 127)}$$

So, the storing band values is from -3,4028234663852886E+38 to 3,4028234663852886E+38.

Values that the module is greater than zero and less than 1,1754943508222875E-38, are treated as zero by the PLCs. PLCs don't support denormalized numbers, infinity and NaNs (not a number).

Example:

%F0032 – real 32

Operands %I - Integer

The operands %I are used to the numerical processing, storing values in simple precision, with signal. The basically difference between this kind of operand and the memory operand %M, is that the integer operand %I is 32 bits.

The operands %I formats can be seen on the following picture.

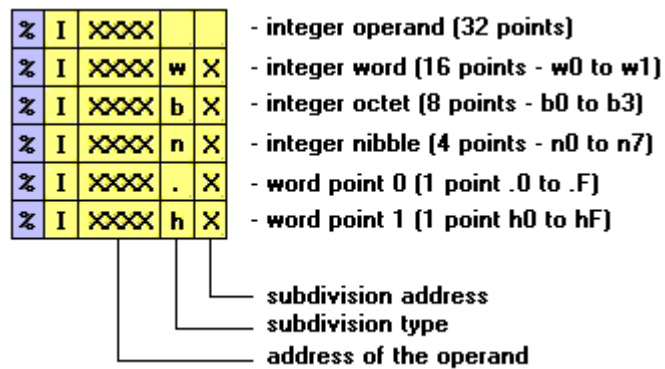


Figure 2-15 Formats of Operands %I

The quantity of integer operands can be configured on C module declaration, the max limit depends on the CPU model that is in use (see the section Operands Declaration on this chapter).

The %I operands are used on move, comparing, arithmetic's and conversion instructions. This operands use four memory bytes (32 bits), with signal, as the following figure:

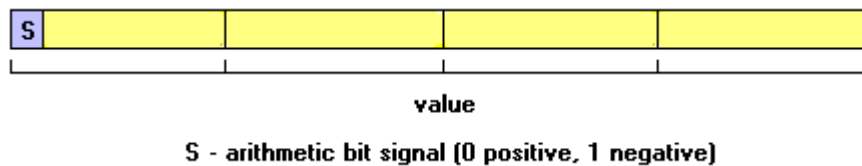


Figure 2-16 Format of the Operand Integer

Examples:

%I0041 - integer 41

%I0023b2 - octet 2 of the integer 23

%I0059n6 - nibble 6 of the integer 59

%I0172hA - ponto 10 of the word 1 of the integer 172

Operands %KM, %KI, %KD e %KF - Constants

Operands are used to define the fixed values in the elaboration of the applications program. These are two types of constant, %KM, %KD, %KF and %KI, each one following a different format from the representation of values, being identical to the operands %M, %D, %F and %I, respectively.

The format of the constant operands can be seen on the following figure.

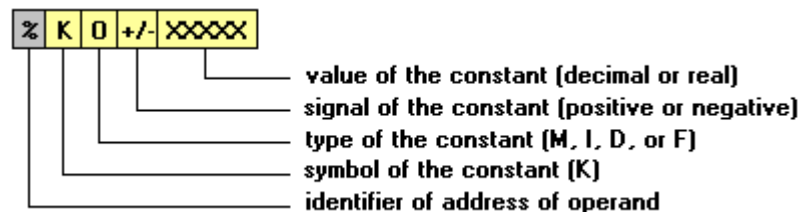


Figure 2-17 Format of the Operands Constants

These operands are used for instructions of movement, comparison, arithmetic, counting and timing.

Examples:

%KM+00241 – memory constant + 241

%KI+2000000000 – integer constant 2 bi or 2×10^9

%KD-0019372 – decimal constant - 19.372

%KF+0125.78 – real constant + 125.78

%KF+3.1415E23 – real constant 3.1415×10^{23}

The real constants can contain up to 8 significative digits.

Operands %TM, %TI, %TD e %TF - Tables

Tables of operands are grouped with simple operands, made up of one-dimensional arrays with the objective of storing numerical values. Each table has a number of configurable positions, where each position can count exactly the same values of an operand %M, %D, %F or %I if the table was of type %TM, %TD, %TF or %TI, respectively.

The format of the table operands can be seen on the following figure:

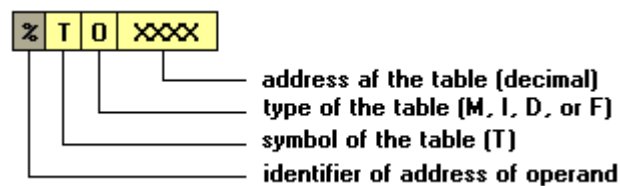


Figure 2-18 Format of the Operands Tables

The quantity of tables and the number of positions of each one is configurable in the declaration of module C. They can be defined in up to 255 tables in total and up to the maximum of 255 positions in each table, respecting the limit of the memory of the operands of the PLC.

The tables are used in instructions of movement.

Indirect Access

This form of access is used in conjunction with a memory operand %M to reference other operands in the system indirectly.

The sign *, placed in front of a type of operand, indicates that it is referenced through the address contained in the specific memory to the left of the sign.

The format of indirect access can be seen in figure 2-19.

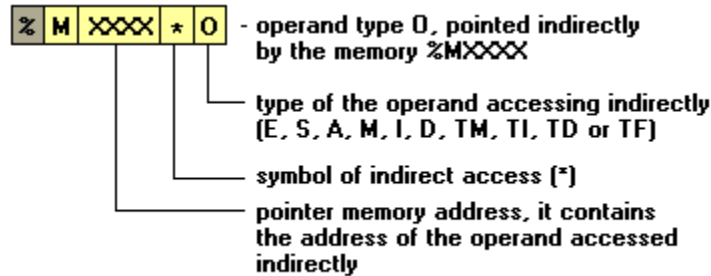


Figure 2-19 Format of an Indirect Access

In MasterTool, the indirect access to the tables is shown without the asterisk.

The indirect access is used in instructions of movement, comparison, counting and timing.

Examples:

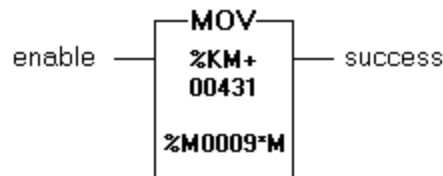
%M0043*E - input octet referenced indirectly through memory 43

%M1824*A - auxiliary octet referenced indirectly through memory 1824

%M0371TD - table of decimals referenced indirectly through memory 371

%M0009*M - memory operand referenced indirectly through memory 9

Example:



This instruction moves the value +431 to the memory operand whose address is the value correctly stored in %M0009. If %M0009 contains the value 32, then the value +431 may be stored in %M0032. If %M0009 contains the value 12 then the constant value will be stored in %M0012.

WARNING:

It is the responsibility of the applications program that the value contained in the reference memory (%M0009, in the example) represents valid addresses, not containing negative values or above of the existing addresses for that type of operand referenced indirectly. The instructions do not carry out invalid indirect access, normally having an output sign to indicate an error.

If in the program of the previous example there were 256 operands %M to be declared, the value of %M0009 should be between 0 and 255 so that the instruction will be executed correctly. If the value is not in this band, access will not be achieved.

Declaration of Operands

The operands %E, %S and %A occupy their own memory areas permanently reserved in the PLC's microprocessor. The number of these operands in the controllers, therefore is constant.

To represent fixed values, the constant operands (%KM, %KF, %KI and %KD) also do not occupy memory space, being stored in their own applications program in the programming stage. There are no limits to the number of constant operands used in the program.

The declaration of the operands is carried out through the editing window of module C of MasterTool, being stored in module C. The number of operands declared should be tailored to the maximum capacity of the available memory. C.f. items **Configuring Simple Operands**, **Configuring Table Operands** and **Configuring Retentive Operands** in the section **Configuring Module C** on the MasterTool User's Manual.

Should be declared the minimal quantity of memory operands (%M) to supply the diagnostic bytes used on the bus modules.

The reserve of the operands %M, %I, %F and %D is carried out in blocks of 256 bytes. In the case of memory operands, this quantity corresponds to 128 operands. In decimal operands, corresponds to 64 operands.

The operands %TM, %TI, %TF and %TD are declared finding out the number of tables necessary for each type and the number of positions which each table contains. It is possible to define up to 255 tables in total and up to 255 positions for each table, respecting the limit of RAM memory of the operands.

Table 2-2 shows the memory space used up for each type of operands and where its values are stored.

Operand	Memory Occupied	Location
%E – input	1 byte	Microprocessor
%S – output	1 byte	Microprocessor
%A – auxiliary	1 byte	Microprocessor
%KM – constant M	-	-
%KI – constant I	-	-
%KD – constant D	-	-
%KF – constant F	-	-
%M – memory	2 bytes	RAM of operands
%I – integer	4 bytes	RAM of operands
%D – decimal	4 bytes	RAM of operands
%F – real	4 bytes	RAM of operands
%TM – table M	2 bytes per position	RAM of operands
%TI – table I	4 bytes per position	RAM of operands
%TD – table D	4 bytes per position	RAM of operands
%TF – table F	4 bytes per position	RAM of operands

Table 2-2 Occupied Memory and Location of Operands

Retentive Operands

Retentive Operands are operands which have their values preserved when the CPU is turned OFF (disconnected). The operands not retentive have their value zeroed at the moment the programmable controller is disconnected.

All the table operands are always retentive. It is possible to configure the number of operands %M (memory), %I (Integer), %F (real), %D (decimal), %S (output) and %A (auxiliary) retentive.

The retentive operands are configured starting from the last addresses up to the first, obeying the same rule as simple operands. That is to say, the reserve is carried out in blocks of 256 for numeric operands. The declaration of the operands %S and %A is carried out octet to octet.

For example, there are 512 operands %M declared (%M0000 to %M0511), and it is required that 128 of these operands are retentive, the operands %M0384 to %M0511 are considered retentive.

C.f. item **Configuring Retentive Operands** in the section **Configuring Module C** on the MasterTool User's Manual.

Instructions

The ALTUS PLCs use the language of relays and blocks to elaborate the applications program, whose main advantage, beyond and its graphic representation is to be similar to the conventional diagrams of relays.

The programming of this language, carried out through MasterTool, uses a group of powerful instructions in chapter 3 **Reference of Instructions**, in this manual.

MasterTool instructions can be divided into 7 groups:

- **RELAYS** containing the instructions:
 - RNA contact normally open
 - RNF contact normally closed
 - BOB simple reels
 - BBL reel connected
 - BBD reel disconnected
 - SLT reel jump
 - PLS pulse relay
 - RM master relay
 - FRM end of master relay
- **MOVEMENTS** containing the instructions:
 - MOV moving of simple operands
 - MOP moving of parts of operands
 - MOB moving of blocks of operands
 - MOT moving of tables of operands
 - CAB load block of constants
- **ARITHMETICS** containing the instructions:
 - SOM sum
 - SUB subtraction
 - MUL multiplication
 - DIV division
 - AND function “and” binary between operands
 - OR function “or” binary between operands
 - XOR function “or exclusive” binary between operands
 - CAR load operand
 - = equal
 - < less than
 - > more than
- **COUNTERS** containing the instructions:
 - CON simple counter
 - COB bidirectional counter
 - TEE timer to turn on



- TED timer to turn off
- **CONVERTORS** containing the instructions:
 - B/D conversion binary - decimal
 - D/B conversion decimal - binary
- **GENERAL** containing the instructions:
 - LDI connect or disconnect indexed points
 - TEI test the status of indexed points
 - SEQ sequencer
 - CHP call procedure module
 - CHF call function module
 - ECH write operands on another PLC for Ethernet
 - LTH read of operands from another PLC for Ethernet
 - LAH free image update of operands for Ethernet
- **CONNECTIONS** containing the instructions:
 - LGH horizontal connection
 - LGV vertical connection
 - LGN denied connection

Some special functions executed by the PLC are only obtained with the Modules Function, that are called by the instruction CHF. On the chapter 4 Referring the Modules Function presents a list of these modules, available to Ponto Series and come with the MasterTool Programming.

The execution time of each instruction to the Ponto Series PLCs should be consulted on the respective manual.

Restrictions Using Instructions on the PLCs

The language of relays and blocks is perfectly compatible between the PLCs programmed through MasterTool. Due to the characteristics of functioning, nevertheless, some instructions are not available in all the controllers. Table 2-3 shows the instructions and the controllers in which they cannot be used.

-  - indicates that the PLC have the instructions
-  - indicates that the PLC does not have the instructions

CPU's Instruction	AL-600	AL-3003 AL-3004	QK600, QK800, QK801	PL101, PL102, PL103, PL104 PL105	AL-2000, AL-2002, AL-2003, AL-2004, QK2000	PO3042 PO3142 PO3045 PO3145	PO3242 PO3342
MES							
AES							
CES							
A/D							
D/A							
ECR							
LTR							
LAI							
ECH							
LTH							
LAH							

Table 2-3 Non-existent Instructions in Certain PLCs

MasterTool does not permit an instruction which cannot be executed in the PLC for which it is configured to be inserted in the applications program.

WARNING:

On editing an applications program module, the type of CPU declared in the item **CPU Model** in the editing windows of module C should be from the CPU where the program was executed.

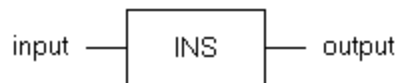
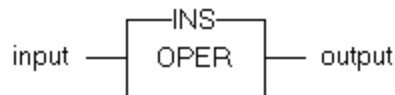
WARNING:

If is required to change the type of CPU for another, after the program to be edited, you should search and remove the instructions which cannot be used in the new type of CPU. This procedure should be carried out in all the program modules.

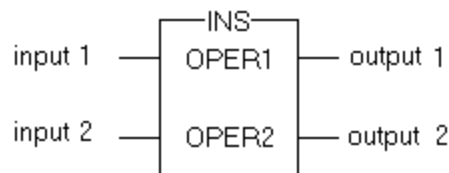
Graphic Representation of the Instructions

The following figures show the maximum configurations of input and outputs in each type, not being necessary all used in a certain instruction.

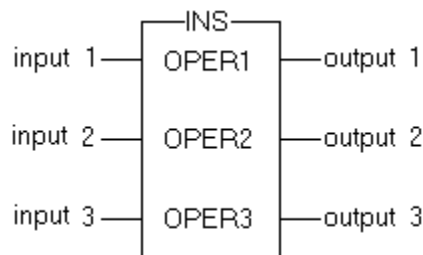
Instructions with one cell



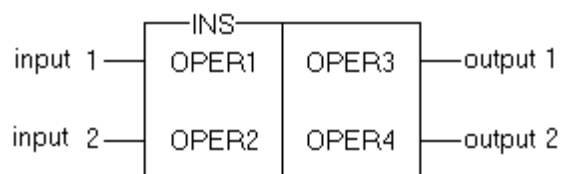
Instructions with two cells

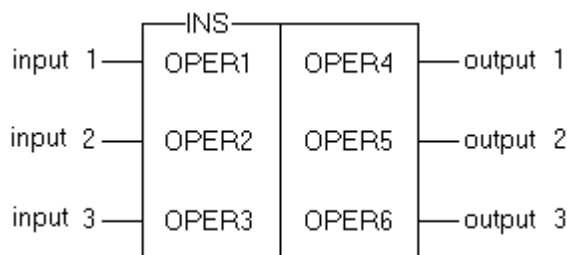


Instructions with three cells



Instructions with four cells



Instructions with six cells

Description of Syntax Instruction

The description of the possible operands to be programmed in the cells of each instruction is carried out in accordance with the format shown in figure 2-20.

OPER1	OPER2	OPER3
LIST OF OPERANDS POSSIBLES IN THE CELL	LIST OF OPERANDS POSSIBLES IN THE CELL	LIST OF OPERANDS POSSIBLES IN THE CELL

Figure 2-20 Formats of Syntax Instructions

Various different combinations of operands can be specified for the same instruction

Example:

OPER1	OPER2	OPER1	OPER2
%M	%M %M*M %KM	%D	%D %M*D %KD

This syntax declaration shows that, like the first operand, % M or % D can be used. If the first operand is % M, the second can only be % KM, % M or % M*M (accessed indirectly in memory). If the first is % D, the second can only be % KD, % D or % M*D (accessed indirectly in decimal).

Restrictions in Positioning the Instructions

There are rules to be respected as to the positioning of the instructions in the 8 logic columns. The instructions can be divided into three categories:

- Instructions which can be edited only in column 7:
 - BOB simple reel
 - BBL connected reel
 - BBD disconnected reel
 - SLT jump reel
 - RM master relay
 - FRM end of master relay
- Instructions which can be edited in columns 0 to 6:

- RNA normally open relay
- RNF normally closed relay
- PLS pulse relay
- LGH horizontal connection
- LGV vertical connection
- LGN denied connection
- DIV division
- MOB moving of blocks of operands
- > more than
- < less than
- = equal
- SEQ sequencer
- CHF call function module
- Instructions which can be edited in all the columns:
 - MOV moving of simple operands
 - MOP moving of parts of operands
 - MOT moving of table of operands
 - CAB load block of constants
 - SOM sum
 - SUB subtraction
 - MUL multiplication
 - AND function “and” binary between operands
 - OR function “or” binary between operands
 - XOR function “or exclusive” binary between operands
 - CON simple counter
 - COB birectional counter
 - TEE timer to turn on
 - TED timer to turn off
 - B/D conversion binary - decimal
 - D/B conversion decimal - binary
 - CAR load operand
 - LDI connect or disconnect indexed points
 - TEI status test of indexed points
 - CHP call procedure module
 - ECH write operands on another PLC for Ethernet
 - LTH read of operands from another PLC for Ethernet
 - LAH free image update of operands for Ethernet

Programming Project

Structure of a Programming Project

Functionally, a programming project, can be seen as a collection of modules used to carry out a specific task, also known as an applications program. This allows a hierarchical view of the project with the creation of sub-routines and functions.

The modules are called for execution through executive software (operating system of the PLC). or for other modules, through appropriate instructions. When stored on disk, the programming project corresponds to a group of files, where each file contains a module, named as shown in figure 2-21.

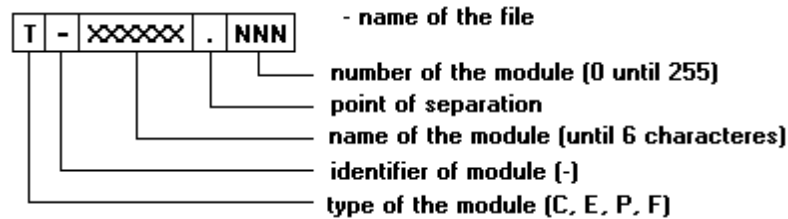


Figure 2-21 Format of Name of Modules in File

Example: **F-PID.033**

In some places in this manual and in the Help the program modules are referenced only through their type and number, when it is not relevant to use their name.

Example: **E018**

WARNING:

The file name corresponds to a program module which should not be changed through another application of Windows™. To change the name of a file, it should be read and saved with the name required through MasterTool. C.f. section **Saving a Module with Another Name** on the MasterTool User's Manual.

If the files name is modified through another Windows™ application, it can be given an name invalid for it, not being able any more to be read to MasterTool or loaded into the PLC.

There are 5 types of modules which can do part of a programming project:

- **Module C** (Configuration): there is a configuration module for the project, containing the configuration parameters of the PLC (C000).
- **Extended Module C** (Configuration): this configuration module exists when the user use on the project a specific characteristic of the PLC and needs an extended configuration module. For further information see the user's manual of the MasterTool Programming (C003 to C009).
- **Module E** (Execution): there can be up to 4 execution modules for the project. They are only called through the operating system of the PLC (E000, E001, E018 and E020).
- **Module P** (Procedure): there can be up to 112 procedure modules per project. They contain passages of the applications program being called through instructions placed in execution modules, procedure or function. After they are executed, the returns to the following instruction of the call. The modules P act as sub-routines not allowing parameter passing for the module called (P000 to P111).
- **Module F** (Function): there can be up to 112 function modules per project. They contain passages of the applications program written in generic form, allowing parameter passing to the module called, in this way they can be reappraised in various different applications programs.

They are similar to instructions, being able to be called for, modules of execution, procedure or function. (F000 to F111).

Module C - Configuration

Module C contains the configuration parameters of the PLC. Its creation is a pre-requisite for editing the rest of the MasterTool programming project modules. The definition of the parameters contained in module C is carried out through the editing window of module C. For further details regarding how to configure in module C, c.f. section **Configuring Module C** in chapter 5 of the MasterTool User's Manual.

There is only one module C per programming project, having as its own name the name of the project and the number 000.

Contents of a module C:

- **Declaration of the Bus of I/O modules:** specifies the configuration of the I/O modules to be used in the programmable controller, indicating the distribution of these modules and special modules in the PLC's bus. The declaration of the modules defines, in this way, the number of points and the I/O addresses to be used in applications program. The declaration takes place in the editing window of module C. For further information about how to configure the bus, c.f. the item **Configuring the Bus** in the section **Configuring Module C** in chapter 5 of the MasterTool User's Manual.
- **Declaration of Operands:** specifies the number of simple operands and tables of operands which are used in the programming project, within each available type. It also allows the definition of the retainability of the operands, that is to say, which operands can keep their contents even with a power cut.
 - **Declaration of Simple Operands:** allows the definition of the number of Memory operands (%M) and Decimal (%D). It takes place in the editing window of module C. For more information regarding how to declare simple operands, c.f. the item **Configuring Simple Operands** in the section **Configuring Module C** on the MasterTool User's Manual.
 - **Declaration of Table Operands:** allows the definition of the number of tables of Memory operands (%TM) and of Decimal operands (%TD) and of the number of positions in each one. One table shows a group of operands, being defined in the editing window of Module C. For further information about how to configure table operands, c.f. the item **Configuring Table Operands** in the section **Configuring Module C** on the MasterTool User's Manual.
 - **Declaration of Retentive Operands:** specifies the number of simple operands which are retentive, within the operands already declared. Retentive operands are those which continue with their contents unchanged through a power cut, those not being retentive are zeroed when the system restarts. The table operands are all retentive. The declaration is made in the editing window of Module C. For more information regarding how to configure retentive operands, c.f. the item **Configuring Retentive Operands** in the section **Configuring Retentive Operands** on the MasterTool User's Manual.
- **Declaration of the General Parameters of the CPU:** there are generic parameters necessary for the functioning of the programmable controller, such as the type of CPU in which the applications program will be loaded, the period of calling the activated modules for interruption and the maximum time of the scan cycle. These parameters are declared in the editing window of Module C. For more information about how to configure the general parameters, c.f. section **Configuring Module C** on the MasterTool User's Manual.
- **Declaration of the Parameters of the ALNET I Network:** specifies the parameters necessary for the functioning of communication in ALNET I. These parameters are configured in the editing window of Module C. For further information regarding how to configure parameters of ALNET I, c.f. item **Configuring Parameters of the ALNET I Network** in the section **Configuring Module C** on the MasterTool User's Manual.
- **Declaration of the Parameters of the Ethernet Network:** specifies the various parameters necessary for the functioning of communication in Ethernet, for the programmable controllers

which allow its use. These parameters are configured in the editing window of Module C. For further information regarding how to configure parameters of Ethernet, c.f. item **Configuring Parameters of the Ethernet Network** in the section **Configuring Module C** on the MasterTool User's Manual.

Extended Module C – Configuration

This modules have configurations of determined characteristics of the PLCs. This modules are totally controlled by the user, it should be created and erased as the need of the user. The quantity of this kind of module vary as the application.

For further information see the user's manual of MasterTool.

Modulo E - Execution

The modules E contain passages of the applications program, being called for execution through executive software. These are different Modules E, differing from each other through the way they are called for execution, according to their number.

Types of Modules:

- **E000 - Initialization Module:** is executed once, when the PLC is turned on or in the passage of programming mode for execution with MasterTool, before the cyclical execution of Module E001.
- **E001 - Sequential Module of Applications Program:** contains the main passage of the applications program, being executed cyclically.
- **E018 - Module Enabled for Time Interruption:** the passage of applications program placed in this module is called for execution at time intervals. It defines the calling period for the applications program in the general parameters of Module C, being able to choose between 50ms, 25ms, 10ms, 5ms, 3.125ms, 2.5ms, 1.25ms and 0.625ms. At the running time, the sequential execution of the applications program is interrupted and the module E018 is executed. After it is finished, the system returns to execution for the sequential processing point where the module E001 has been interrupted. The time continues to be counted during the call of Module E018, its execution having to be as short as possible so as not to an excessive increase in the time of Module E001 is cycle.

WARNING:

The execution time of Module E018 cannot be more or equal to the time period of the call. If this happens, the PLC goes into error mode displaying the message **Recessed in Module E018**, in the window **Information** (command **Communication, Status, Information**).

Module P - Procedure

The Modules P contain passages of applications programs called starting from Modules E, P or F through the instruction CHP (Procedure Call).

This type of module does not have parameter passing, being similar to the concept of the sub-routine.

The maximum number of modules of this type is 112 (P000 to P111).

The module P is useful to contain passages of applications programs which should be repeated several times in the main program, being like this programmed once only and called when necessary, being economical with the programs memory.

They can be used also for a better structure of the main program, dividing it into segments according to its function and declaring then in different Modules P. In this case, the execution module continues E001 only and calls the Modules P in the required sequence.

Examples:

- P-MECAN.000 - carries out the Mechanical breaking of the machine
- P-TEMPER.001 - achieves control of temperatures
- P-VIDEO.002 - achieves the man-machine interface
- P-IMPRES.003 - manages the printing of reports

Module F - Function

The Modules F contain passages of applications programs called from the start of Modules E, P or F, through the instruction CHF (Call Function).

In the call from Modules F it is possible to pass the values as parameters for the module called. These modules are usually written in generic form to be approved for different applications programs, in the language of relays or of machine, being similar to the instructions of the language of relays. The values of the parameters are sent and returned through the lists of existing operands in the call instruction and in Module F.

In the editing of an instruction CHF, 2 lists of operands should be defined that are used for:

- sending parameters for execution of the function module (Input)
- receiving the values returned through the function module (Output)

In editing the function module, 2 lists of operands should be defined, using the command **Editing, Edit, Parameters**, which are used for:

- receiving parameters of instruction CHF (Input)
- sending values of return for the instructions CHF (Output)

The passing of parameters is achieved through the copy of the values of the declared operands (passing of parameters for value). Figure 2-22 shows the flow of data between instruction CHF and the function module.

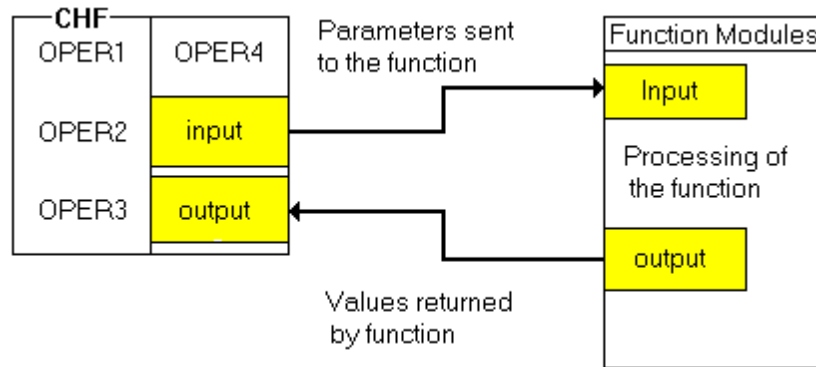


Figure 2-22 Parameter Passing for Module F

Further information regarding parameter passing can be found in the description of the instruction CHF in the same manual. The passing of all types of operands is permitted.

Examples:

- F-LINEAR.002 - executes the linearization of values read from a sensor
- F-PID.033 - carries out calculations for implementing the control PID loop

Operating Status of the PLC

There are five statuses or modes of operation of the PLC: initializing, execution, programming, cycling and error. The status in which the programmable controller finds itself is indicated in the LEDs of the front panel of the CPU, also being able to consult MasterTool, through the dialogue box **Status** (options **Communication, Status**, starting from the main menu). To obtain specific information about these operating modes, consult the User's Manual for the controller used.

- **Status Initialization** : the PLC initializes the different data structures for use by the executive program and achieves consistency in the programming project present in the memory. This

status occurs after the controller is turned on, passing after a few seconds to the execution status. If no applications program exists in memory, the PLC passes to error mode.

While the PLC is initialized, it can activate the command **Communication, Status, Programming**, or equivalent short cut in the tool bars, having done that the PLC passes directly to programming status, instead of executing the applications program. This procedure is useful for the reinitialization of PLCs with programs containing serious programming errors.

For example, a module with an infinite execution loop, programmed with an instruction for jumping to a previous logic, provokes the enabling of the CPU's guard dog circuit that is always connected, after Initialization status. Executing itself the previous procedure straight after being turned on, the PLC passes to the programming status after initializing, allowing the erasing or the substitution of the program.

- **Execution Status:** normally the programmable controller is found in this status, continually sweeping away the input points and updating the output points according to the logic programmed. This status shows that the PLC is executing an applications program correctly.
- **Programming Status:** The applications program is not executed, not having the reading of the input points, the outputs being deactivated and the PLC's memory compacted. The PLC remains non-operational, waiting for commands from MasterTool. This mode is normally used to load programming project modules for MasterTool through the serial channel. At the passing for execution or cycling status starting from the programming status, the operands are zero.
- **Cycling Status:** when in cycling mode, the programmable controller does not execute the module E001 cyclically, remaining to wait for the commands from MasterTool. Each command **execute cycle** activated in MasterTool (options **Communication, Status, Execute Cycle** starting from the main menu or equivalent shortcut) fires one single scan of the applications program (Module E001), the PLC remaining to wait for a new command after executing the scan. When the PLC passes to cycled mode, the counting of time in the timers stops, being the same increments of one unit of time for each two scans executed. The calls to the module of interruption of time E018 are not carried out in this mode. The Module E020, activated through the input of external interruption, continues being called in this mode.
- **Error Status:** shows there was some anomaly in the PLC during the processing of the programming project. The type of error occurring can be checked through the dialogue box (options **Communication, Status, Information** starting from the main menu), while the PLC is in this status. The output of the error status is only possible passing the programmable controller to programming mode.

In normal conditions, the programmable controller can be in the modes of execution, programming and cycling, these modes being Enabled through the MasterTool commands (options **Execution, Programming and Cycling** in the dialogue box **Status**, or their shortcut equivalents in the **Tool Bars**). In the event of some functional error in these modes, the PLC passes to error status. The recovery of error mode is only possible by passing the programmable controller to programming mode. Figure 2-23 shows the possibilities for changing status.

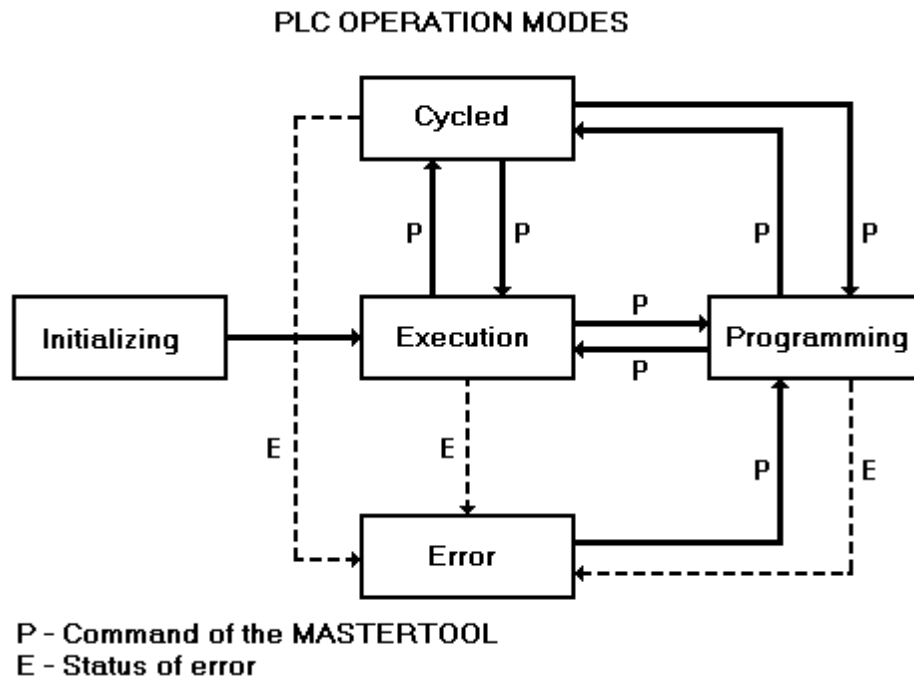


Figure 2-23 Operating Status of the PLC

In the modes of execution, programming and cycling it is possible to load and read project modules from the programming project through the serial channel of the programmable controller, as well as monitoring and forcing whatever operands are used. These operations are not possible if the PLC is in error mode.

The operands which are not retentive are zeroed in the passing of the programming mode for execution or programming for cycling, the rest of them remaining unchanged.

Execution of the Programming Project

When the PLC is powered or after the passing to execution mode, the Initialization of the system are carried out according to the contents of Module C, being straight after executing Module E000 once. The programmable controller then passes to cyclical processing of Module E001, updating the inputs and outputs and calling the Module E018, when it exists, for each period of interruption time programmed. Figure 2-24 shows the execution of the applications program in outline.

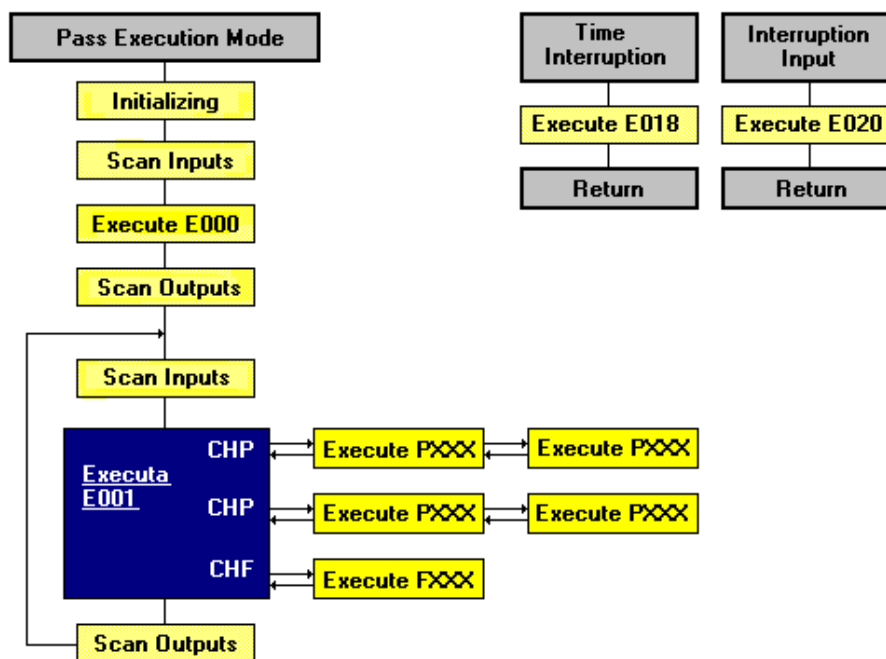


Figure 2-24 Execution of the Project Programming

Elaboration of the Programming Project

General Considerations

A programming project is made up at least one Module C (configuration) and one Module E001 (execution). The minimum condition for the execution of a programming project is the presence of these two modules in the programmable controller's CPU.

The first step in the editing of a MasterTool programming project is the creation or reading of the project. The configuration module of the project is created automatically when the new project is created, once this module has the declarations of the modules of input and output and the operands used in the whole project. Each module which contains passages of applications program (E, P or F) requires Module C to be present in MasterTool for it to be able to be edited.

After the creation or reading of a project, it can edit the project adding modules already in existence, creating new modules for the project or excluding modules already made part of the project.

MasterTool allows various modules to be loaded and remain simultaneously in its memory.

Considerations about Operands

The various modules which make up a programming project should preferably be programs using the same Module C. If a module already programmed needs to be used in another programming project, the operands used for the module should be obliged to be declared in Module C of the new project.

The available operands in the programmable controller are of common use to all the programming project modules present in the PLC (global operands). Consequently, there are two modules which can be inadvertently accessed by the same operand, with errors occurring in the functioning of both.

To elaborate a programming project, operands should be reserved in a sufficient number for the project, preferably separated in groups, each group used for only one module. The operands used in Modules F programmed in language of relays and blocks can also be accessed for any other program modules present in the PLC, the same applies to operands used in the parameter passing. To guarantee its generic character, each Module F should use a different group of operands from the rest used in the applications program.

Using of the Module P and F

Inside a programming project module the instructions can be placed to call other modules. The instructions CHP and CHF call, respectively the modules of procedure and function. They carry out the generating of calls to modules, verifying the existence or not of the modules in the directory of the programmable controller, based on their types and numbers.

Exist 32 levels of calling, so, can be executed up to 32 consecutive callings of modules without being finalized anyone. Should be considered that the module E018 (if it exists) and the modules called by it occupy call levels.

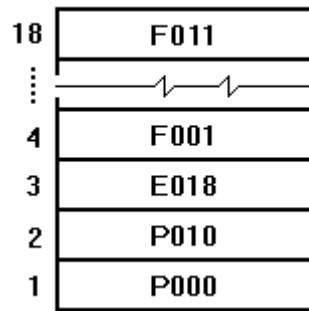


Figure 2-25 Maximum number of modules calling levels

When the maximum number of calls accumulated without return is surpassed, the system may not carry them out, continuing with the normal execution of the applications program. In cases where calls occur for non-existent modules or the above the number of total calls, warning messages are shown in the window **Information** (options, **Communication, Status, Information** starting from the main menu), since these situations can cause processing errors according to the programmed logic.

It is possible to call from a module to itself (programming for recourse) taking the necessary care, that is to say, should be predicted in the applications program passage with recourse one moment in which there are no more calls to the same module. Although it is possible, the use of such procedure is not advisable in programmable controllers, due to the long time for processing which a small passage of applications program can need to be executed and the facility of infinite loops of execution.

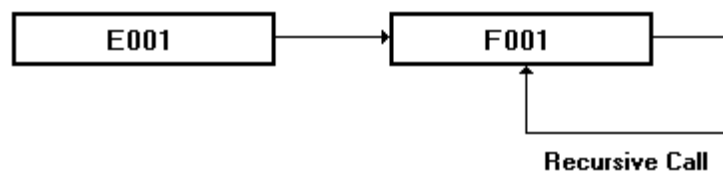


Figure 2-26 Recursive Call of Modules

Undue programming with dead locks should be avoided. If a programming project module calls another and this also carries out a call to the first, if the call instructions in the two modules can not be disabled, both remain called mutually until the passing of the programmable controller to error mode, for an excess of execution time of the applications program.

The same situation can occur with calls linked together between different modules, when a module called changes to call some initial module of the chain. For example, if module P005 calls P002, this calls P007 and this calls P005 again, the processing can remain in this loop if no calling instruction is disabled.

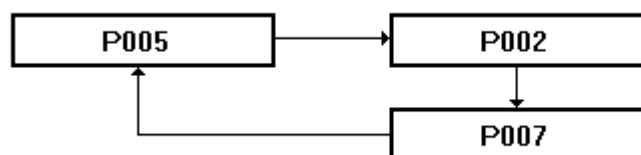


Figure 2-27 Module Call Loop

Use of Module E018

Module E018 should be used when quick processing is necessary for some points of input and output of the programmable controller, like for example, in sensing the end limit in Systems of rapid

positioning. The instruction for updating the points of I/O (AES) should be used in this case, carrying out a similar process in module E018 to a complete loop of main program execution. The inputs are read, the passage of the applications program required is executed and the outputs are updated.

In this way, this module makes itself useful when it requires a response from the operations of output after a fixed time of stimulating inputs, do not depend on the verification time of the main program, which can vary. This characteristic is also important in position control Systems.

Another application for Module E018 is the generation of time less than 100ms for the main program. Timers can be created with precision of 50ms, 10ms or less, if necessary, through the use of instructions counting in the module of time interruption.

This module is useful when precise time control is needed in the PLC's processing.

Care in Using the Module E018

Some special care is necessary in programming module E018. As it is called from synchronized mode to each fixed time period, interrupting the process of module E001, its execution time should be as short as possible so as not to add excessively to the overall cycle time of the applications program.

If the interval between the calls from module E018 is programmed for 25 ms, for example, and its execution time is 20 ms, they restore only 5 ms for the execution of the main program before which E018 will be called again. This situation considerably increases the cycle of module E001.

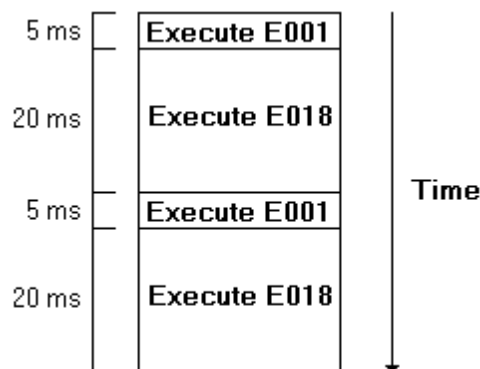


Figure 2-28 Care in Use of Module E018

If the execution of module E018 takes more than the time interval programmed for their calls, the PLC passes to error status, sending the message “Re-input in module E018” in the window.

Information (options **Communication, Status, Information** starting from the main menu). In this situation, the period of the call used should be increased or the execution time of module E018 should be reduced so that the programming project can be executed correctly.

The instructions behave the same when executed in module E018, except in relation to some other particular characteristics. The timers (TEE and TED) continue to count the time at each 100 ms, any which is in the period of enabling programmed for the module, exactly as in the execution cycle. The pulse relay (PLC) action its output during an execution of module E018, zeroing it in the next call. The instructions CHP and CHF can be used in the same way as in the main program the modules having to be enabled through them obeying the same rules of programming applying to module E018. The maximum number of levels of call from modules used in the module E018 should be added to the maximum level used in E001, the sum having to be less than the limit of the system (18 levels).

Using of Operands in Programming of Modules E018

Other care necessary is with the data sharing between the modules E018 and the rest present in the programmable controller. The interruptions can occur at any point of the main program of execution cycle (module E001 or modules P or F called through it), including during the processing of its

instructions. As the operands are all of common use to any programming project module, care should be taken not to inadvertently use, in modules E018 any operand which is used in another programming project module, since this use, according to the case, can cause incorrect functioning.

In order to share the data between the Modules E018 and other module any cyclical execution should use the instructions MOV (moving of simple operands) and MOB (moving of blocks of operands), to create an image of operands which contain the data to be shared. These instructions should be used in the modules pertaining to the normal execution cycle and not in modules E018.

For example if it is necessary that the module E018 uses the value contained in a memory used in the main program, it should pass the value this memory to another through the instruction MOV, the module E018 only having to use this last are. The MOV instruction should be in the main program, and not in the module E018.

The contrary flow of data also demands the creation of image operands. If module E018 manipulates a table and the main program needs to use the values in this table, these values should be copied to a second table for exclusive use of the main program, through the instruction MOB. The instruction MOB should be in the main program and not in Module E018.

A similar situation occurs for reel instructions. If some point of an operand is modified in the main program through a reel, it is not permitted to change any point pertaining to the whole octet of the same operand in Modules E018. This restriction does not exist when the octets used belong to the group %S0000 to %S0015.

However it is possible that the points of an operand are altered in the Modules E018 through a reel and are only tested for another module with contact instructions, for example. The opposite situation is permitted, that is to say the operand points changed in the main program through coils can be tested in Modules E018 through contacts.

Other care to be taken with respect to the updating of the inputs and outputs of Modules E018.

Preferably the inputs used in its processing should be only updated in these modules, using the instruction F-AES. As the application program of the cyclical execution can be interrupted in any place for these modules, if the input images of the main program are updated in these, these can take on different values at different points of the applications program during the same execution cycle. This fact can cause errors if an input operand is used in various areas of the main program, since normally it is supposed that its value remains unaltered in the same verification process.

Due to this fact, it is recommended to use exclusive input octets for the Modules E018, if it is necessary for its updating in it, not being the octets used in the main program.

If it is necessary to update the inputs used simultaneously in the interruptions and in the cyclical processing, the value of these can be copied to auxiliary operands in the rest of it. Also it cannot update input images in Modules E018 with the instruction F-AES, but only read directly the values of the I/O modules to memory operands through the instruction MES, and use these memories in contacts to carry out the processing in the interruption modules.

The updating of output octets in Modules E018 (through the instruction F-AES) is possible, since the points pertaining to these octets are action through coils only in these modules.

In Modules E018, the values with the instruction MES in output modules declared in the bus through MasterTool should not be written, since the verification of output also carries out the updating of the values in these modules.


When a Module E018 is being executed and the compaction is enabled, the modules can be transferred to another position in memory through the routine of compaction. During this transfer its call will be disabled, some interruptions being possible without which the Modules E018 will be processed. Attention should be paid to this effect of compaction regarding the execution of the module enabled for interruption. During the compaction of the rest of the modules, still, the Modules E018 continue being executed.

Depuration of Programming Projects

Various facilities are previewed in the programmable controller to help the depuration of the programming project, being described as follows.

Information about the status of the PLC

Various information about the status of the controller can be obtained with the enabling of the options **Communication, Status, Information** in MasterTool:

Shortcut: 

- **CPU Model** - indicates the type of controller with which MasterTool is communicating.
- **Version of Executive** - shows the number of the version of the executive program which the PLC contains.
- **Mode of Operation** - shows the actual operation of the PLC: execution, programming cycling or error.
- **Error/Warning Message** - if the PLC is in an error mode, a message is shown indicating the cause of the error. If the PLC is in another mode, a message indicates the existence of problems that do not cause the change to error mode (for example, the PLC's battery is flat). C.f. **Error Messages**, appendix A of the MasterTool User's Manual.
- **Outputs** - indicate if the outputs are enabled or disabled.
- **Forced Relays** - indicate if any forced point off input or output exists.
- **Change of Modules with PLC powered** - indicates the possibility of changing from modules with PLC powered.
- **Compacting RAM** - indicating if the PLC is compacting the RAM memory of the applications program.
- **Copying Module** - indicates if any module is being loaded into the PLC, transferring from RAM to EPROM flash or from EPROM flash to RAM, or if the PLC is erasing the flash memory.
- **Protection Level** - shows the current protection level of the PLC.
- **Cycle Times** - shows the instantaneous value, average, maximum and minimum of the cycle time of the applications program. C.f. section **Program Execution Cycle Times** in this same chapter.
- **Enabling Period of E018** - shows the period of module call enabled for time interruption E018, if it is present in the PLC.

The status windows of the PLC (options **Communication, Status, Information**), directory of modules (options **Communication, Modules**) and monitoring (options **Communications, Monitor Operands** or **Monitor Block of Operands** or **Monitor Tables**) supplies various information used to verify the correct functioning of the controller. This information can be obtained from a distance, if the PLC is connected to a network. When MasterTool is connected to any PLC, it regards the obtaining of this information as the first step to take.

Monitoring

Through MasterTool it is possible to monitor the values of on or more operands in the PLC in any mode of operation, except error mode.

The values of the operands contained in a logic of an applications program can be visualized directly in the PLC allowing the verification of its functioning.

For more information about how to carry out the monitoring, c.f. items **Monitoring Simple Operands**, **Monitoring Table Operands** and **Monitoring Programs** in the section **Communicating with the PLC or Router** on the MasterTool User's Manual.

The monitoring of operands in the PLC occurs at the end of the execution cycle of the applications program. Due to this, incoherent situations can be visualized in the monitoring of the logics, if the values of the operands are modified in the subsequent logics to be monitored.

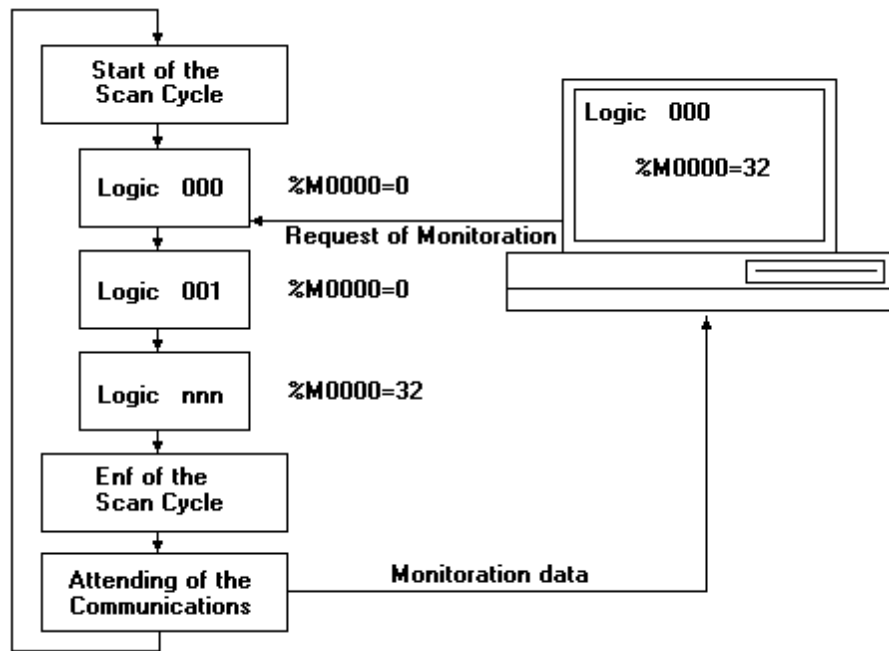


Figure 2-29 Incoherent Situation in Logic Monitoring

Forcing

The values of the operands can also be forced with MasterTool, that is to say, can modify the content of any programming project operand. The forcing of operands is permitted in any operating mode. The forcing of operands is permitted in any operating mode, except error mode. C.f. items **Forcing Simple Operands** and **Forcing Table Operands** in the section **Communicating with the PLC or Router** in chapter 5 of the MasterTool User's Manual.

The operands %A, %M, %D, %I, %F, %TM, %TD, %TI and %TF have their value altered only for one verification, straight after a command has been sent to the PLC. So that the forced value remains in the operands, it cannot have any instruction in the program which modifies it.

The forcing of the operands %E and %S is carried out in a permanent way in the controller. After the commands is sent to the PLC, the value is forced in all the verifications of the applications program, until the operand is freed. The LED FC in the CPU panel remains connected if there is some forced operand %E or %S.

The forced values in operands %E superimpose those obtained in the reading of the input modules, before the start of each execution cycle of the applications program. The program is executed with the value forced, as if the point of input corresponds with this value, being able to be visualized in the monitoring.

For example, if the operand %E0002.5 is forced with the value, the applications program will be executed with this value for this operand, not importing the status of the point in the module of corresponding input. The monitoring of %E0002.5 always the value 1.

The values forced in the operands %S are sent directly to the output modules, independent of the values obtained after the execution of the applications program. The monitoring shows the forced value, which corresponds to the value assumed through the corresponding point in the operand in the output module.

For example, if the operand %S0024.3 is forced with the value 0, the respective point in the output module remains disconnected, not importing the status of the coil which contains the monitoring of %S0024.3 always shows the value 0.

WARNING:

Incoherent situations can be visualized in monitoring logics with operands %S forced. This happens because the value monitored can be different from the value really obtained through the applications program.

WARNING:

All the forcing of operands %E and %S are removed when the turning off the PLC. The forcing of these operands should be used in temporary form, only to help the depuration of the programming project. The operands %E or %S should not be left forced in character permanently, since they are freed with the turning off and after the turning on of the controller.

Operands %E and %S stop being forced through the PLC through the command liberating from forcing. The liberation consists of canceling the forcing previously determined. The operands %E return to have their values updated according to the input modules, while the output modules receive the values obtained in the processing of the applications program.

WARNING:

Force operation doesn't actuate in %E or %S operands that has been updated by F-AES.087 instruction. This instruction read %E operands or write %S operands and it doesn't make operands forcing effects. For this reason, I recommend you don't make operands force with operands that has been updated by F-AES.087 actives program instructions.

For further information about how to free forced operands, c.f. item **Liberating Forced Operands** in the section **Communicating with the PLC or Router** on the MasterTool User's Manual.

Disabling the Outputs

For the "on Initialization " security when if the applications program is used directly in the machine, the enabling of output by the programmable controller can be disabled through the disable command. The application program continues to be executed in the PLC, with the verification of the inputs and calculation of the output values, however with all the output points kept deactivated. The operands %S can be monitored and given the values waiting for them.

For more information regarding the disabling of outputs, c.f. item **Enabling and Disabling the Outputs** in the section **Communicating with the PLC or Router** on the MasterTool User's Manual.

WARNING:

If the PLC is turned off, the disabling of the points of output is removed. That is to say, when the PLC is turned on again, the status of the memory operands will normally be transferred, to the end of each verification, for the points of output. The disabling should be used in temporary form, only to help the depuration of the programming project.

Modifications in the Program

The loading of the modules during the execution of the programming project (loading “on line”) makes possible successive modifications and messages from the module in the depuration for the programmable controller. In this mode it is not necessary to reinitialize the control application program not even a change of status from programmable controller to each alteration carried out in a module.

WARNING:

After any modification carried out in Module C of the programming project, it should be sent to the PLC.

WARNING:

If the declaration of the simple operands or tables may be modified, it advises itself to reinitialize the PLC, passing to programming mode, loading the Module C and returning to execution mode. Functioning errors can occur altering the configuration of the operands and sending the Module C, with the controller, into execution mode.

After a certain number of successive loads in execution mode, however, it can make necessary the compaction of the RAM memory for reasons explained in the section **Managing Programming Project Modules in the PLC**, in this chapter. This type of loading is only possible if there is enough free memory in the PLC storing the module to be sent.

At the end of the depuration of a program module, its transfer is suggested to an EPROM flash memory or its recording in the EPROM cartridge, freeing the space available in the RAM memory of the program.

Cycling Mode

The execution of the programming project in cycling mode makes use, in the verification of the functioning of rapid brakes in the applications program. The rest of the facilities of depuration continue acting in the same way as in the execution mode (monitoring, forcing, loading and other operations with modules).

In cycling mode, the operand values remain constants among the cycles, except the input points (%E) which continue being continually updated, showing their real values.

Managing Programming Project Modules

The modules which make up the applications program are independent among them selves, not needing the connection (“link”) through the auxiliary programs. The loading of modules in the programmable controller for the serial channel can be carried out in any order, allowing only the model altered to be loaded into the PLC, if the programming projects have to be maintained.

WARNING:

Only the module type and its number are relevant to the CPU in this system, the name being ignored. If two modules with equal type and number but with different names are to be loaded into the PLC, only the last to be loaded will be considered.

The programmable controller organizes an internal directory where the various information regarding modules contained in it are stored, able to be consulted for MasterTool through the directory command of modules (options **Communication, Modules** starting from the main menu). When this command is to be enabled, a dialogue box is opened, showing in its upper section, two panels called **RAM Modules** and **EPROM Modules** with the list of names and the memory occupied by each module in the PLC.

In the panel **Memory Occupied** details the total number of modules and the total memory space occupied by them (sum of all the individual occupations), beyond the total space occupied in RAM or EPROM.

The panel **Memory Free** shows the amounts of RAM memory and EPROM available for the loading new modules, in each memory bank existing in the programmable controller.

Only the modules present in the directory are considered valid for execution in the PLC.

A program module present in the directory can only be in one type of memory, RAM or EPROM, never in both at the same time. The modules loaded by the serial channel are always stored in RAM memory of the applications program.

Compaction

The memory of the programmable controller's memory is divided into one or more banks, depending on the CPU model used (c.f. table 2-1 in the section **Organization of Memory in PLCs** in this chapter.

As the modules which make up the programming project are sent to the PLC through the serial channel, they occupy the first memory bank, from its beginning to its end. When the space remaining in the first bank is not enough to load the next module, it will be loaded into the following bank, if one exists.

At each loading of a new module into the programmable controller, the executive software tests if there is enough space for it from the first to the last bank available. The loading of a new module is only possible if there is free memory available for its storage.

Inside the RAM memory bank, the loading of a module is always carried out starting from the first position after the last module present. If a module at the start of the bank is removed, the modules which are after it should be transferred to occupy its space in the memory, so that this space is available at the end of the bank for other modules to be loaded. This procedure names itself **compaction of RAM memory** of the applications program.

Example:

Supposing that the first memory bank of the programmable controller is initially with the following modules:

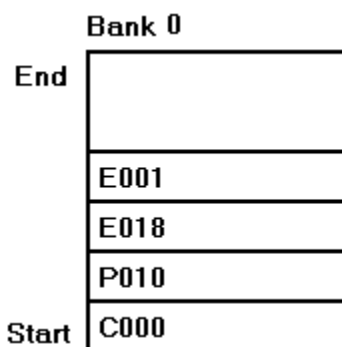


Figure 2-30 Compaction of RAM Memory

If Module P010 is removed from the PLC the bank 0 will pass to have the following organization:

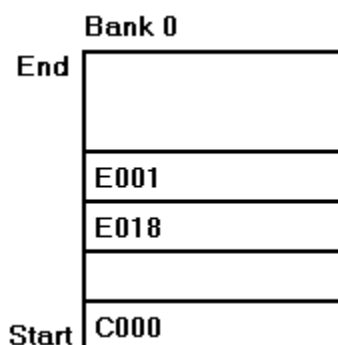


Figure 2-31 Compaction of RAM Memory-2

The space previously occupied by P010 is not taken advantage of by the programmable controller, since to carry out the compaction of the PLC's memory, bank 0 passes to the following configuration:

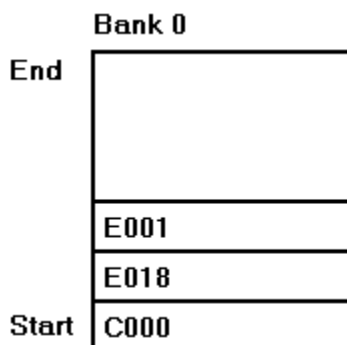


Figure 2-32 Compaction of RAM Memory-3

The Modules E018 and E001 are transferred to the space previously occupied by Module P010, making this space available to the end of the memory of the bank for loading the other module.

If the programmable controller is in programming mode or cycling, the RAM memory banks of the program are automatically kept compacted by the executive program. In execution mode, however, the compaction should be enabled manually, through MasterTool (options **Communication, Modules, Compact RAM** from the main menu). This procedure is common when different loadings of modules in execution mode are carried out (loads "on line"), typically when a module is being purified, needing successive alterations and transmissions for the PLC.

WARNING:

Depending on the location of the modules in memory, the procedure for compaction can much increase the time for some cycles of applications programs, when carried out in execution mode. It is important to be aware of the effects of this increase in processing time. Be advised that the compaction is not fired if the machine under control is in operation or with its main active enabling.

Due to this mechanism of managing the modules in the programmable controller, it is possible that the sum of the available memory in the PLC banks with the value occupied by modules is less than the total memory of the program, if it is in execution mode. This fact means that the program memory is not compacted. After the compaction, however, the sum of the values occupied with the free memory should be equal to the total memory.

On the MasterTool don't exist a Flash compaction, as the RAM compaction. The method to compact the Flash is to carry the modules to the RAM, clean the Flash and carry the modules to the flash.

Use of EPROM Flash Memory

The controllers contain EPROM Flash Memory and it is placed on the board of the PLC, and there is no need to remove them to store or erase programs. This operations are realized by the controller itself, through the MasterTool. This memory can be stored partially, although it don't permit the partial erasing. So, it is only possible to erase all the memory.

The memory configuration of each PLC model is shown in the section **Organization of Memory of the PLC's** in this chapter.

Transference of Modules from Ram to Flash

After they are loaded into the RAM memory of the program, through the serial of the PLC, the programming project's modules can be transferred to EPROM flash. This command is only unable in PLC's which have flash memory. For further information about the transferring modules from RAM to EPROM Flash c.f. item **Transferring Modules from RAM to EPROM Flash** in the section **Communicating with the PLC or Router** on the MasterTool User's Manual.

It is possible to transfer one single module or a group of modules, the same with the PLC executing the program. The transfer in execution mode is carried out partially in each verification, being able to wait several seconds until it is completed, mainly of these was a long time of cycle of execution. At the end of the transfer, the module in RAM is automatically erased and the information from the directory is modified.

Managing the module loading in EPROM flash is identical to the RAM memory, shown in the previous section Compaction. That is to say the RAM module is recorded in the first bank of flash which has enough space free for the counter, after the last module of the last module of the bank. The search for free space occurs in the sequential order of the banks (0, 1, 2 and 3).

Transference of Modules from EPROM to RAM:

The modules present in EPROM flash memory or in EPROM cartridge can be transferred to the RAM memory of the program. For further information about how to transfer modules from EPROM to RAM, c.f. item **Transferring Modules from EPROM to RAM** in the section **Communicating with the PLC or Router** on the MasterTool User's Manual.

It is possible to transfer one single module or a group of module, the same with the PLC executing the program. The transferring into execution mode is partially carried out in each verification, being able to wait several seconds until it is completed, mainly if the cycle execution time is long. At the end of the transfer, the information from the directory is modified.

The management of the loading of the module into EPROM flash is identical to that of RAM memory, shown in the previous section **Compaction**.

Erasing and Re-enabling Modules on EPROM

The erasing command can be used for modules stored in the EPROM memory of the PLC. As the erasing of EPROM's is only possible for all its contents, this command only retires the information from the modules directory, not carrying out a real erasing of the memory.

The same happens if a module recorded in EPROM is substituted for a new module of the same type and number loaded by the serial channel. The new module is stored in RAM, remaining the old one in EPROM, only the new one in RAM being shown in the directory.

The module removed through the erasing command or substituted with the load from a new module can be restored to the directory, since its contents are still recorded in EPROM memory. This recovery is possible with the modules re-enabling command.

The re-enabling renders the module non-existent in the directory and reappears in EPROM, or that one already existing in RAM may be substituted for a previous one in EPROM.

For further information regarding how to erase or re-enable modules, c.f. items **Erasing Modules in the PLC or Router** and **Re-enabling Modules in EPROM** in the section **Communicating with the PLC or Router** on the MasterTool User's Manual.

Erasing the EPROM Memory

With the total erasing from EPROM memory, all the modules are removed, all the available space being available for the recording of the new modules.

To erase the EPROM cartridge an appropriate eraser device should be used, after the removal of the cartridge from the PLC.

To erase the EPROM flash memory, use the options **Communications, Modules, Erase Flash** which are the PLC in programming mode. The erasing can wait several seconds, depending on the capacity of the flash used in the PLC. For further information regarding how to erase the flash memory, c.f. item **Erasing the EPROM Flash Memory** in the section **Communicating with the PLC or Router** on the MasterTool User's Manual.

Program Execution Cycle Times

The maximum time possible for the execution of a complete cycle of the applications program in the programmable controller is configurable for 100ms to 800ms. That is to say, the complete execution of a verification of Module E001 cannot be extended for more than the value configured, including the calls to the Modules P and F and the enabling of the time interruption Module E018. The executive software carries out a continuous verification in the cycle time, passing automatically to error status if this limit is overtaken.

It can be verified the execution times of the applications program through the PLC's information window (options **Communication, Status, Information** starting from the main menu) various execution cycle times being given, specified as follows:

- **Instantaneous cycle time:** shows the cycle time of the last verification executed by the PLC before sending the information of its status to MasterTool. This item is useful in cycling mode, when it shows the execution time of the last cycle fired in the programmable controller.
- **Average cycle time:** shows the average times of execution of the last 256 verifications carried out by the PLC. In execution mode this parameter gives a general idea of the processing time of the applications program, as opposed to the instantaneous cycle time, which can be shown an untypical value isolated from a verification. As this time is calculated only at each 256 scanning, at times its value needs a few seconds to be updated, mainly in the case of an abrupt increase in the execution time (including the new modules in the programmable controller for example).
- **Maximum cycle time:** shows the longest time between all the cycles carried out since the passing of the PLC into execution or cycling mode.
- **Minimum cycle time:** shows the shortest time between all the cycles carried out since the passing of the PLC to execution or cycling mode.

The cycle times are shown in milliseconds (ms), being the counts initialized in the passing from programming mode to execution or programming to cycling.

The service of the serial communication with MasterTool increases the application program's cycle time in the PLC, being able, in some cases, to overtake the maximum cycle time selected. If the time limit for execution is overtaken only due to the commands from the serial communication (monitoring, forcing and the rest), the PLC does not Enter error status. It is possible therefore, to indicate from the maximum cycle time greater than that selected without which the programmable controller will have to Enter error mode.

The procedure of compaction of program memory by the programmable controller always follows the previous rule. In some cases, the compaction routine needs to copy a much extended module into the memory between two cycles of the applications program, increasing in the extreme the execution time of one verification. In this situation the PLC does not Enter error status.

Status care should be taken when the execution cycle times move nearer to the maximum time selected. The simple fact that the applications program is to be executed correctly with the more common conditions of the input points does not guarantee that its verification time, in real conditions of the machine functioning, will remain inside the value limit.

WARNING:

Each programming project should be examined carefully in the search for situations which will cause the longer execution times.

These situations should be simulated and the times averaged, verifying if they are not excessive.

This procedure should be carried the same in the programming project with cycle times well below the limit, to ensure it functions well.

It is possible that in some isolated verifications the cycle time exceeds the maximum time selected without which the PLC passes to error mode, in case these sporadic verifications do not cause delays in the system timers.

WARNING:

If the PLC indicates a greater maximum cycle time than that selected without which it will have to have a memory compaction, even if it continues normally in execution mode, the program should be examined to reduce its cycle time in situations which cause greater times.

☺HINT:

Some typical procedures exist to reduce the execution time of the much extended applications programs. A good management of the modules call can reduce the total cycle time sensibly, the calls of a few modules of the applications program being carried out in each verification, not allowing then all to be fired in the same cycle. The use of jump instructions in the modules, reduces their execution time, since a jumped passage of applications program is disregarded by the executive software. The master relay and end of master relay instructions do not have this property, since the segment of applications program delimited by them continue to be executed the same as when the RM coil is disabled.

☺HINT:

The Initialization s of values in operands or tables in Module E000 should be carried out, devised specially for this intention the execution of module E000, for not to be cycled, can delay more than the maximum time, this time being disregarded in counting the time of the first verification of Module E001. As the mode is, executed, it becomes meaningless to the programming of the timers (TEE, TED) in module E000.

Protection Levels of the PLC

CPUs in the Ponto series have a mechanism to protect the programming project and the operands, allowing the blockage of the loading of program modules, forcing the values or same, readings of modules and monitoring for un-authorized operators.

These characteristics are of interest to critical processes, to avoid accidental modifications in the data or in the control program or in the need for secrecy.

The blocking of operations is carried out through the protection levels, which can be defined only for operators which know a pre-defined password. The controller can work on four different:

- **Level 0** - all the PLC's operands are permitted.
- **Level 1** - not possible to alter the programming project (to erase or load new program modules) or change the status of the PLC. Can force and monitor operands and read program modules.
- **Level 2** - not possible to alter the programming project (to erase or load new program modules). Not possible to force operands or change the status of the PLC. Possible to monitor operands and read program modules.
- **Level 3** - not possible to read or alter the programming project, to monitor or force variables not even to change the status of the PLC. Possible only to consult the status of the PLC and its directory.

The change of protection level is carried out with the options **Communication, Status, Protection** in MasterTool, having to key in the password to achieve correct access. The PLC's

protection level can be consulted with MasterTool through the options **Communication, Status, Information**.

The use of different protection levels from zero allows only authorized people, who know the password, modify the program or the PLC's data. Unauthorized operators, even are prevented from carrying out inadvertent alterations.

The access password can have from one to eight alphanumeric characters. It is defined or changed with the options **Communication, Status, Password**, the previous password and the new password having to be keyed in twice, for the change to be confirmed.

The PLC is supplied with a password. It is not necessary to key in any value in previous password field to define the first password.

WARNING:

The password should be written and kept in a secure place. If the password programmed in the PLC is lost, ALTUS should be contacted.

The PLC's protection acts not only to carry out operations with MasterTool, but also the commands received through ALNET I and ALNET II, with the same characteristics defined for each level.

For more information about how to alter the protection level and the password of the PLC, c.f. items **Altering the Protection Level** and **Altering the Password** in the section **Communicating with the PLC or Router** on the MasterTool User's Manual.

Interlocking of Commands in the PLC

In the Ponto series it is possible to use the ALNET I and ALNET II communication networks together. When interconnected in this way, it is possible to receive two commands simultaneously whose concurrent execution will not be desirable, due to their characteristics. For example, the PC can receive a command to transfer from EPROM to RAM through ALNET II while the same command is being loaded in ALNET I.

Similar situations occur with the commands for transferring program modules from EPROM Memory to RAM to flash or erasing from flash memory. The execution of these commands can be extended for several seconds, during these the PLC can receive other commands which conflict with operation in progress. For example, PLC can receive one command to erase the flash memory while a module may be being transferred to the same memory.

To resolve possible conflicts, there is a braking mechanism to execute some of the commands available in the PLC. These commands cannot be executed if the PLC is carrying out a specific operation. There are two internal signals, **loading module (CM)** and **compacting RAM (CR)** which are used for this intention. The tables 2-4 and 2-5 show the commands which use the braking and the enabling of the signals.

The status of the signals carrying module and compacting RAM can be verified in the information window of the PLC, options **Communication, Status, Information** on MasterTool. While any of the signals are enabled, the LED FC of the panel in the PLC remains alight.

Operation realized by the PLC	Command Blocked (ALNET I, ALNET II)	Signal ON
Loading Modules	Load Modules Transfer from EPROM to RAM Transfer from RAM to Flash Requesting Load of Modules Re-enabling of modules in EPROM Erasing of Flash EPROM Compaction	CM
Transfer from EPROM to RAM	Load Modules Transfer from EPROM to RAM Transfer from RAM to Flash Requesting Load of Modules	CM

	Re-enabling of modules in EPROM Erasing of Flash EPROM Compaction	
Transfer from RAM to Flash	Load Modules Transfer from EPROM to RAM Transfer from RAM to Flash Requesting Load of Modules Re-enabling of modules in EPROM Erasing of Flash EPROM Compaction	CM
Erasing of Flash EPROM	Load Modules Transfer from EPROM to RAM Transfer from RAM to Flash Requesting Load of Modules Re-enabling of modules in EPROM Erasing of Flash EPROM Compaction	CM
Legend: CM – Load Module		

Table 2-4 Braking of Commands in the PLC (loading module)

Operation realized on PLC	Blocked Command (ALNET I, ALNET II)	Signal ON
Compaction	Load Modules Transfer from EPROM to RAM Transfer from RAM to Flash Requesting Load of Modules Re-enabling of modules in EPROM Removal of Modules Compaction	CR
Legend: CR - Compacting RAM		

Table 2-5 Braking of Commands in the PLC (Compacting RAM)

For example, while a module is being loaded into the PLC through ALNET I or ALNET II, the commands for loading modules, transfer from EPROM to RAM, transfer from flash, requesting to load modules, re-enabling of modules in EPROM, erasing of EPROM Flash and compaction not be possible to execute, if they are received through another network. If they are received through another network. If they are received through PLC a reply indicating that their execution is impossible is transmitted to the applicant.

4. Instructions

This chapter gives a list of integral instructions of the ALTUS Language of Diagrams and Relays, describing the format, use, syntax and gives examples of each instruction.

List of Instructions

The ALTUS PLCs use the language of relays and blocks to elaborate the applications program, whose main advantage, apart from its graphic representation is being similar to the conventional diagrams of relays.

The programming of this language, carried out through MasterTool, uses a group of powerful instructions shown in the following sections.

MasterTool instructions can be divided into 7 groups:

- RELAYS
- MOVEMENTS
- ARITHMETIC
- COUNTERS
- CONVERSIONS
- GENERAL
- CONNECTIONS

Conventions Used

Different conversions are used for the presentation of groups and instructions making a better visualization and recognition of the items described, aiming at a simpler method of learning and a source of direct consulting of the required topics.

Presentation of the Groups

The descriptions of each group follows this routine.

1. The group is described with a little containing the name of the group.
2. Straight after the little, a brief descriptions of the common characteristics of the group is given.
3. Finishing the presentation of the group, a table is displayed containing the name a the instruction in the first column, the description of the name of the instructions in the second column and in the sequence of keys to carry out the insertion of the instruction directly through the keyboard in the third column.

Example:

Instructions of the Relays Group

The instructions of the **Relays** group are used for the logic processing of the diagrams of relays. Through these instructions it is possible to manipulate the values of the digital points of input (%I) and output (%O) as well as points of auxiliary (%A), memory (%M) and decimal (%D) operands.

They are also used for divert the flow and control of the processing of the applications program.

Name	Description of Name	Editing Sequence	Tool Bars
RNA	contact normally open	ALT, R, A	
RNF	contact normally closed	ALT, R, F	
BOB	Simple coil	ALT, R, B	
SLT	Jump coil	ALT, R, S	
BBL	Connected coil	ALT, R, L	
BBQ	Disconnected coil	ALT, R, D	
PLS	Pulse relay	ALT, R, P	
FRM	End of master relay	ALT, R, M	
RM	Master relay	ALT, R, R	

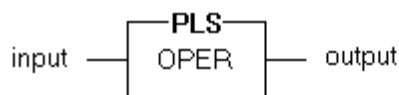
Presentation of the Instructions

The description of each instruction is made in the following way.

1. The instruction is described with a little containing the name of the instruction and the description of the name. A figure presented as an instruction is visualized in the diagram of relays containing its operands, input and output. Above each figure a brief description of the significance of each operand is displayed.
2. The item **Description** contains information describing the functioning of the instruction according to the enabled inputs and the types of operand used. Also described in this item are the outputs which are enabled after the execution of the instruction.
3. The item **Syntax** describes the combinations of operands which can be used in the instruction. This item is only present in instructions which have operands.
4. The item **Example** gives an example of the use of an instruction describing its behavior. This item is only present in instructions which require major detailing of their functioning.
5. There are also other items which describe a specific characteristic of the instruction if it is necessary.

Example:

PLS – Pulse Relay



Description:

The instruction pulse relay generates a pulse from a scan on its output, that is to say, remains powered during a scan of the applications program when the status of its input may pass from turned off to powered.

The auxiliary relay declared serves as data storage, avoiding limits as to the number of pulse instructions present in the applications program.

WARNING:

The value of the auxiliary relay should not be modified in any other point of the applications program.

Syntax:

OPER1
%AXXXX.X

Instructions of the Relays Group

The instructions of the **Relays** group are used for the logic processing of the diagrams of relays. Through these instructions it is possible to manipulate the values of the digital points of input (%E) and output (%S) as well as points of auxiliary (%A), memory (%M) and decimal (%D) operands.

They are also used to divert the flow and control of the processing of the applications program.










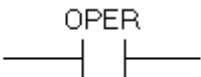
Name	Description of Name	Editing Sequence	Tool Bars
RNA	Contact normally open	ALT, R, A	
RNF	Contact normally closed	ALT, R, F	
BOB	Simple Coil	ALT, R, B	
SLT	Jump Coil	ALT, R, S	
BBL	Connected Coil	ALT, R, L	
BBD	Disconnected Coil	ALT, R, D	
PLS	Pulse Relay	ALT, R, P	
FRM	End of master relay	ALT, R, M	
RM	Master relay	ALT, R, R	

Table 3-1 Instructions of Relays Group

Contacts

- **RNA** contact normally open



- **RNF** contact normally closed



Description:

These instructions reflect, logically, the real behavior of an electrical contact of a relay in the applications program.

The contact normally open, closes according to the status of its associated operand. If the operand point is in the logic status **1** or **0**, the contact normally open is closed or opened respectively.

The contact normally has behavior opposite to normally open. If the point of the associated operand is in the logic status **1** or **0**, the contact normally closed is opened or closed respectively.

When a contact is closed, the instruction transmits the logic status of its input to its output. If it is open, the input value is not placed on the output.

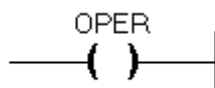
Syntax:

OPER1
%EXXXX.X
%SXXXX.X
%AXXXX.X
%MXXXX.X
%DXXXX.X
%DXXXXhX

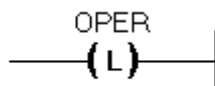
Table 3-2 Syntax of the Instructions RNA and RNF

Coils

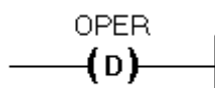
- **BOB** Simple Coil



- **BBL** Connected Coil



- **BBD** Disconnected Coil



Description:

The coil instructions modify the logic status of the operand in the image memory of the programmable controller, according to the status of the enabling line of the instructions.

The simple coils connect or disconnect the operand point according to the enabling line, while the of type connected and of type disconnected only connect or disconnect. Operands when the line is powered (“set/reset”).

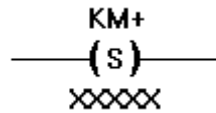
These instructions can only be positioned in column 7 of the logic.

Syntax:

OPER1
%SXXXX.X
%AXXXX.X
%MXXXX.X
%DXXXX.X
%DXXXXhX

Table 3-3 Syntax of Instructions BOB, BBL and BBD

SLT – Jump Coil

*Description:*

The instruction jump coil serves as a controller of execution sequence of an applications program, being used to divert its processing to a determined logic.

Its operand is a constant which determines the number of logics to be jumped starting with the powering of the coil the determining of the logic destination is carried out by the sum of the constant which accompanies the instruction with the number of the logic where it is found.

When the enabling line of the jump coil is turned off, the jump does not take place, and the following instruction which in the coil is declared and executed.

Example:

If the following instruction is in logic 2, the execution of the applications program is diverted to logic 7 if the enabling line is powered, that is to say, if the value of the point %A0009.3 is **1**. If the value of this point is **0**, the execution continues normally in logic 003.

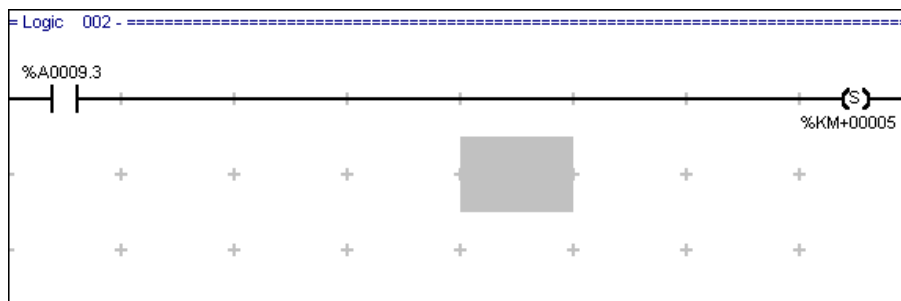


Figure 3-1 Example of SLT Instruction

This instruction can only be placed in 7 column of the logic.

In this instruction it is possible to use a constant %KM with zero value or with negative value. If programmed with zero value, the logic destination is the same as that which contains the jump coil, when it is powered. That is to say, the processing is diverted to the start of the coil's own logic. If the value programmed is negative, the processing is diverted to a logic before the logic which contains the jump coil.

WARNING:

The use of a zero constant or negative corresponds to an unconventional use of the instruction. If it is required to use it there, the necessary precautions should be taken to avoid the input in a loop or the excessive increase of the cycle time of the applications program. It is recommended nevertheless, to use the jump coil only with positive constants greater than zero.

The control of the execution of these situations should be carried out through a braking which disconnects the jump from the previous logic, after a certain number of loops have been executed in the return passage.

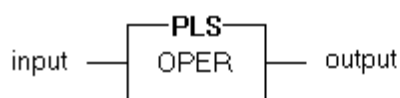
If the logic destination overtakes the last logic the applications program, the PLC jumps to the end of the program and continue its normal cycle.

If the logic destination of a return jump is less than the first logic of the applications program, the execution is restarted starting from logic 0.

Syntax:

OPER1
%KM+XXXXX
%KM-XXXXX

Table 3-4 Syntax Instruction SLT

PLS – Pulse Relay*Description:*

The instruction pulse relay generates a pulse for a scan in its output, that is to say, it remains powered during a scan of the applications program when the status of its input may pass from turned off to powered.

The auxiliary relay declared serves as a memorizer, avoiding limits as to the number of pulse instructions present in the applications program.

WARNING:

The value of the auxiliary relay should not be modified in any other point of the applications program.

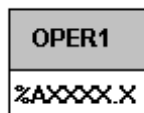


Syntax:

Table 3-5 Syntax of PLS Instruction

RM, FRM – Master Relay, End of Master Relay

- RM Master Relay

- FRM End of Master Relay


Description:

The master relay instructions end of master relay instructions are used to delimit passages of the applications programs, the logic bar of supply in these powered or not, according to the status of the enabling line.

These instructions do not need operands since it is possible to position them only in column 7 of the logic.

When the input of instruction RM is turned off, the logic bar of the supply is turned off since the following logic until the logic which contains the FRM instruction.

As these instructions always act on the logic following the one counted, it is advisable that their position should always be as the instructions in the logic in which they are present. This being so, the passage of applications program delimited visually through instructions in the diagram corresponds exactly to that controlled by the instructions, therefore avoiding bad interpretation of its functioning.

WARNING:

The instructions CON, COB, TEE and TED contain outputs powered in the same way without their outputs being enabled. These outputs remain powered the same within the passage over the turned off command of a master relay, being able to result in unwanted enabling.

Instructions of Moving Group

These instructions are used to Manipulate and transfer numerical values between constants, simple operands or tables of operands.







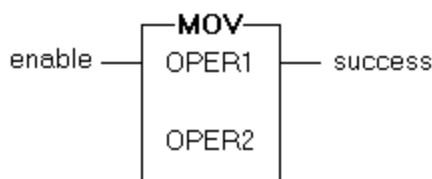
Name	Description of Name	Editing	Tool Bars
MOV	Moving simple operands	ALT, M, V	
MOP	Moving of parts of operands	ALT, M, P	
MOB	Moving of blocks of operands	ALT, M, B	
MOT	Moving of tables of operands	ALT, M, T	
CES	Conversion of inputs or outputs	ALT, M, S	
CAB	Load block	ALT, M, C	

Table 3-6 Instructions of Group Movements

MOV – Moving Simple Operands



OPER1 – origin operand

OPER2 – destination operand

Description:

This instruction moves the contents of simple operands, without carrying out conversions between different types of operands, when the enabled input is enabled.

The operand which occupies the first instruction cell (OPER 1) is the origin operand, whose value is moved to the destination operand, specified in the second cell (OPER 2).

If the format of the destiny operand is less than the origin, the more significant octets are zeroed. If the moving is carried out, the output **success** is enabled.

If the indirect indices exceed the limits of the operands declared in the configuration module, the moving is not carried out and the output **success** is not lit up.

The moving of subdivisions of operands is not permitted. For this reason, the instruction MOP should be used.

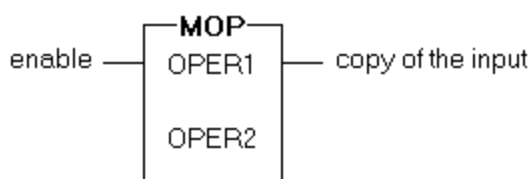
When the destination operand is an integer (%M) and at least one of the other operands of the instruction is real (%F) the result stored is stopped, only the integer part of the result is stored on M operand.

Syntax:

OPER1	OPER2	OPER1	OPER2
%E	%E		
%S	%S		
%A	%A		
%M	%M	%M	
%I	%I	%F	%M
%D	%D	%I	%F
%M*E	%M*E	%M*M	%I
%M*S	%M*S	%M*F	%M*M
%M*A	%M*A	%M*I	%M*F
%M*M	%M*M	%KM	%M*I
%M*I	%M*I	%KF	
%M*D	%M*I	%KI	
%KM	%M*D		
%KD			

Table 3-7 Syntax of the Instruction MOV

MOP – Moving of parts (Subdivisions) of Operands



OPER1 – origin operand

OPER2 – destination operand

Description:

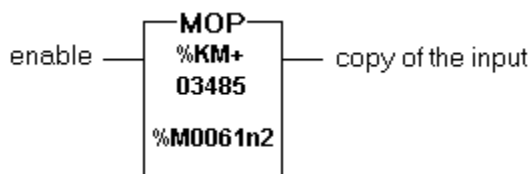
This instruction moves the contents of parts of simple operands (words, octets, nibbles, points) when the enabled input is powered. The conversion between types of operands is not carried out, only the moving of values.

The operand which occupies the first cell of the instruction (OPER 1) is the origin operand, whose value is moved to the destiny operand specified in the second cell (OPER 2). The type of subdivision used in the first operand should be the same as the second.

WARNING:

If the moving is carried out from a constant to an operand, the subdivision is always considered a less significant equal constant to that declared in the destination operand. Due to this characteristic, the real value to be moved should always be declared in the origin constant to make the program clearer.

Example:



The destination operand is declared with nibble division. Therefore, the less significant nibble of the origin constant (with value equal to 1101 in binary, 13 in decimal) to be moved to nibble 2 of memory M0061.

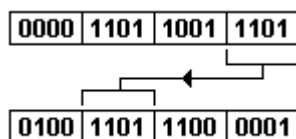


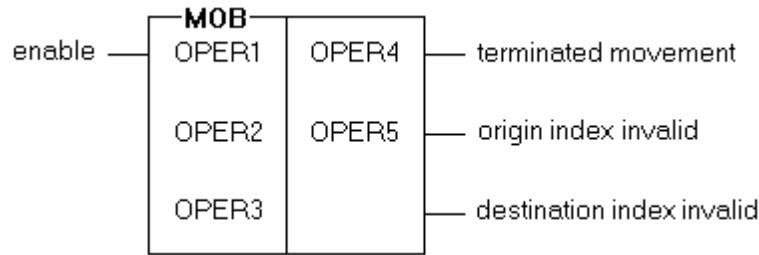
Figure 3-2 Example of Instruction MOP

The remaining bits which make up the constant are ignored, that is to say, the result of the moving will be identical using a constant %KM00013. The example shown uses a the functioning higher value than that of the moving to better illustrate of the MOP. For better interpretation of the program the value %KM00013 should be used.

Syntax:

OPER1	OPER2	OPER1	OPER2
%EXXX.X %SXX.X %AXX.X %MXX.X %DXX.X %DXXhX %FXX.X %FXXhX %IXXX.X %IXXXhX %KMXXXXX %KDXXXXX	%EXXX.X %SXX.X %AXX.X %MXX.X %DXX.X %DXXhX %FXX.X %FXXhX %IXXX.X %IXXXhX %IXXXhX	%MXXbX %DXXbX %FXXbX %IXXXbX %EXXX %SXX %AXX %KMXXXXX %KDXXXXX	%MXXbX %DXXbX %FXXbX %IXXXbX
OPER1	OPER2	OPER1	OPER2
%EXXXnX %SXXnX %AXXnX %MXXnX %DXXnX %FXXnX %IXXXnX %KMXXXXX %KDXXXXX	%EXXXnX %SXXnX %AXXnX %MXXnX %DXXnX %FXXnX %IXXXnX %IXXXnX	%MXXbX %DXXbX %FXXbX %IXXXbX	%EXXX %SXX %AXX
OPER1	OPER2	OPER1	OPER2
%DXXwX %FXXwX %IXXXwX %MXX %KMXXXXX %KDXXXXX	%DXXwX %FXXwX %IXXXwX	%DXXwX %FXXwX %IXXXwX	%MXX

Table 3-8 Syntaxes of the Instruction MOP

MOB – Moving of Blocks of Operands

OPER1 – first operand of origin block

OPER2 – number of transfers to be carried out

OPER3 – control operand

OPER4 – first operand of designation block

OPER5 – number of transfers for scan

Description:

This instruction carries out the copy the value of a block of origin operands to the destination block.

It specifies the first operand of the origin block in OPER 1 and the first operand of the destination block in OPER 4. The total number of transfers to be carried out is declared in parameter OPER 2, to the number of transfers for the scan (OPER 5) should always be specified and a memory accumulated to count the number of transfers (OPER 3).

If the origin or destination block is a table, the transfer should begin in its first position.

If the operand format is less than the origin, the more significant octets of the origin value are ignored. If opposite is the case, the more significant octets of the destination are zeroed.

The number of transfers for scan is limited in 255 operands. In general, if possible, a high number of transfers in the some scan should be avoided, to reduce the execution time of the program.

In each MOB instruction a memory is used as control operand (OPER 3), which should be zeroed before the first execution.

WARNING:

The control operand should not have its contents altered in any part of the applications program, under penalty of preventing the correct execution of the instruction. Each occurrence of this instruction in the program should have an operand of exclusive control, different from to rest. This operand cannot be retentive.

When connected, the outputs of the second and third cells show, respectively, that at least one of the component operands of the origin or destination block has a greater address than the maximum number declared for the operand or table used, no moving being carried out. If the value of the second operand is negative the output **origin index invalid** is enabled.

The output of the first cell is enabled in the scan in which the moving is completed.

WARNING:

The input **enable** should remain active until the moving is concluded. As this instruction is executed in multiple execution cycles, it should not be jumped while the moving is still in progress.

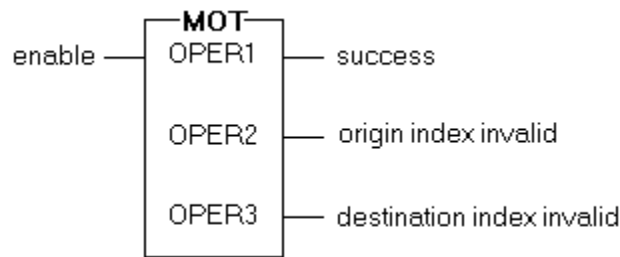
Syntax:

OPER1	OPER2	OPER3	OPER4	OPER5
%E %S %A %M %I %D %TM %TD	%KM	%M	%E %S %A %M %I %D %TM %TD	%KM

OPER1	OPER2	OPER3	OPER4	OPER5
%F %TF	%KM	%M	%F %TF	%KM

Table 3-9 Syntax of the Instruction MOB

MOT – Moving of Tables



OPER1 – origin table or origin operand

OPER2 – table index

OPER3 – destination operand or destination table

Description:

This instruction allows the two operations: to transfer the value from one position of the table to a simple operand or from one simple operand to a position in the table.

The operand which occupies the first instruction cell (OPER 1) is the origin operand, whose value is moved to the destination operand specified in the third cell (OPER 3). OPER 2 contains the position of the table declared in OPER 1 or OPER 3.

Reading the contents of the table:

Allows reading of the contents of a table position and loads into a memory operand or decimal operand.

The instruction is programmed in the following way:

- **OPER1** - specifies the address of the table to be read
- **OPER2** - specifies the position (%KM) to be read or the memory (%M) which contains this position
- **OPER3** - specifies where the contents of the table position should be transferred to

If the first operand to reference a table indirectly is not specified or if the value of the second operand is negative or greater than the last position defined for the table, the transfer is not carried out or the output **origin index invalid** is enabled. If the third operand to indirectly reference an operand is not declared, the transfer is not carried out and the output **destination index invalid** is enabled.

Writing values into table:

It allows a constant value or the contents of a memory operand or decimal operand to be written into a table position.

The instruction is programmed in the following way:

- **OPER1** - specifies the origin operand
- **OPER2** - specifies the position (%KM) to be written in the table or the memory (%M) which contains this position
- **OPER3** - specifies the address of the table where the contents are transferred to

If the first operand indirectly references an undeclared the transfer of the contents is not carried out and the output **origin index invalid** is enabled. If the value of the second operand is negative or greater than the last position defined for the table, or if the third operand indirectly to reference a table is not specified, the transfer of the contents is not carried out and the output **destination index invalid** is enabled.

This instruction simplifies the programming of a series of algorithms involving decodifications, sequencings, generating of curves, storing and comparison of values, among others.

Syntax:

Reading:

OPER1	OPER2	OPER3
%TM %M*TM	%KM %M	%M %M*M

OPER1	OPER2	OPER3
%TD %M*TD	%KM %M	%D %M*D

OPER1	OPER2	OPER3
%TF %M*TF	%KM %M	%F %M*F

OPER1	OPER2	OPER3
%TI %M*TI	%KM %M	%I %M*I

Writing:

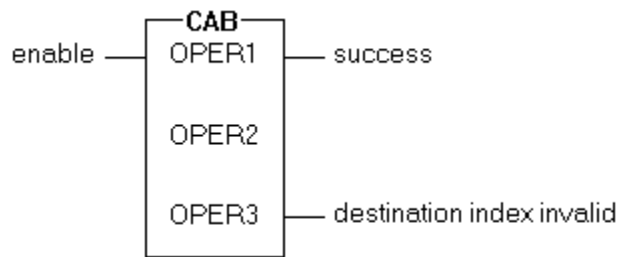
OPER1	OPER2	OPER3
%KM %M %M*M	%KM %M	%TM %M*TM

OPER1	OPER2	OPER3
%KD %D %M*D	%KM %M	%TD %M*TD

OPER1	OPER2	OPER3
%KF %F %M*F	%KM %M	%TF %M*TF

OPER1	OPER2	OPER3
%KI %I %M*I	%KM %M	%TI %M*TI

Table 3-10 Syntax of the Instructions MOT

CAB – Load Block

OPER1 – initial operand or table to be loaded

OPER2 – number of operands or positions of table

OPER3 – table of constants to be loaded

Description:

This instruction allows the loading of up to 255 constant values in a block of operands or tables.

The initial operand or table to be carried is specified in the first parameter (OPER1), the number of operands or positions of the table to be loaded in the second operand (OPER2) and the value of the constants in the third (OPER3).

The value of the second operand should be positive, less or equal to %KM+128.

The third operand (OPER3) is made up of a table of constant values to be loaded. These values are declared by selecting the button **Block**, an editing window being open in MasterTool. The constants are of type %KM if the type of the first operand is %E, %S, %A, %M, %TM or they are of type %KD if the first operand is %D or %TD. If the first operand is an octet (%E, %S or %A), only the values of the less significant octets of each constant declared are moved.

Also it is possible to carry out the declaration of the values of the table in ASCII. This mode allows. In this mode it is possible to insert the addresses or tags of operands which should represent its value at the moment when the instruction is executed. The address or tag of operand should be keyed in between keys ({ }).

E.g.: If %M0000 has the value 35 and that it has loaded the following text in ASCII “Value of {%M0000}”. The text is as follows:

Value of %M0000:00035.

When the button **Block** is selected the dialogue box **CAB - Values** is shown:

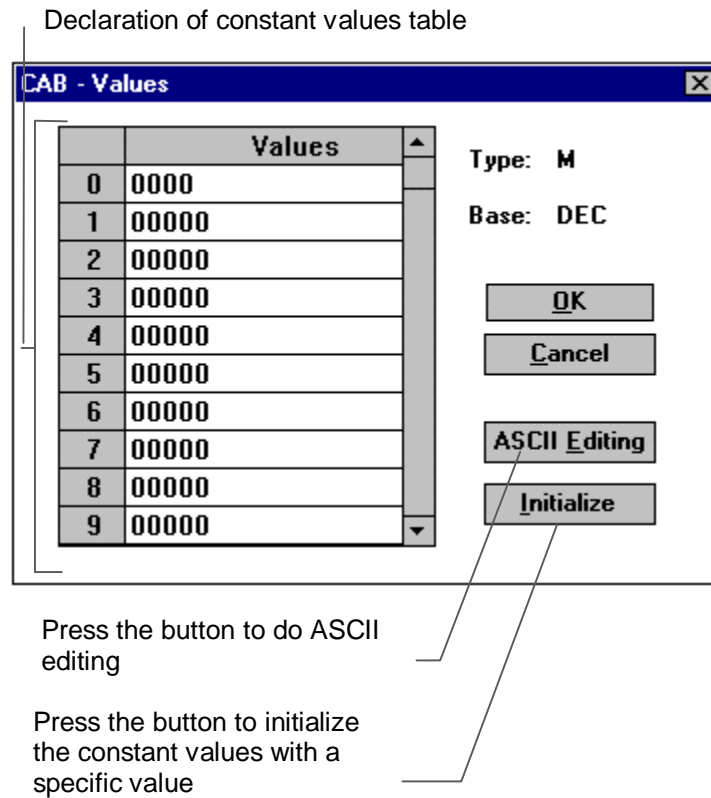


Figure 3-3 Dialogue Box CAB - Values

To carry out the editing of the constants

1. Position the cursor on the index to be edited. If it is necessary to roll the pages, the keys PAGE DOWN and PAGE UP or the vertical roll bar can be used.
2. Key in the constant value.

To carry out the editing in ASCII

1. Select the button **Editing ASCII**. The dialogue box **CAB - Editing in ASCII** is shown.
2. Key in the text which it is required to be loaded in the constants of the CAB and select the Ok button.

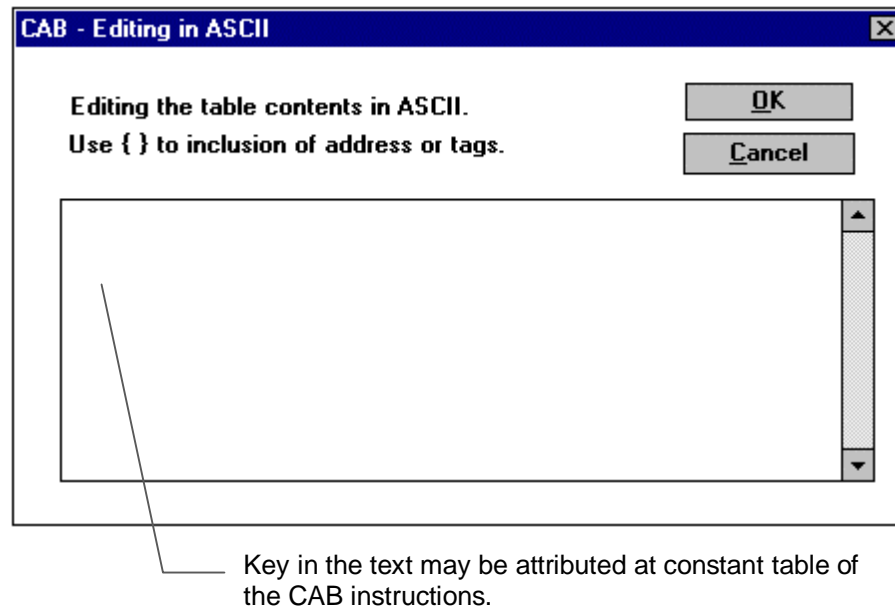


Figure 3-4 Dialogue Box CAB - Editing in ASCII

To initialize the constants with a specific value

1. Select the button **Initialize**. The window **CAB-Initialize table** is displayed.
2. In the item **Value**, key in the value to be initialized in the constants.
3. In the item **Initial Position**, key in the number of the first position to receive the value of Initialization.
4. In the item **Final position**, key in the number of the last position to receive the Initialization value.
5. Select the button **Ok**.

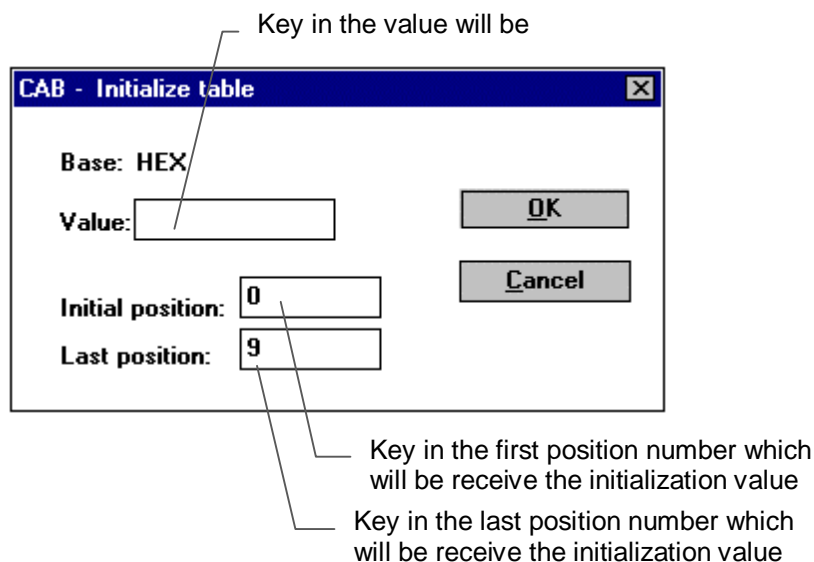


Figure 3-5 CAB – Initialize table

The output **destination index invalid** is enabled when some operand can not be accessed or a table position does not exist. The output **success** is always enabled when the instruction is executed correctly. If the output **destination index invalid** is enabled, no loading of constants occurs.

The loading of the constant values is entirely carried out in one scan of the applications program, be able to cause an excessive time cycle when it is extended. In most parts of applications programs, the instruction CAB can only be executed in the Initialization (loading of tables whose contents are only read) or at some special times, not needing to be called in all the scans. In these cases, it is recommended that it is programmed in the applications program module of Initialization or that it is enabled only at the necessary loading times.

Syntax:

OPER1	OPER2	OPER3
%E %S %A %M %TM %M*E %M*S %M*A %M*M %M*TM	%KM	MEMORY TABLE VALUES

OPER1	OPER2	OPER3
%D %TD %M*D %M*TD	%KM	DECIMAL TABLE VALUES

OPER1	OPER2	OPER3
%F %TF %M*F %M*TF	%KM	REAL TABLE VALUES

OPER1	OPER2	OPER3
%I %TI %M*I %M*TI	%KM	INTEGER TABLE VALUES

Table 3-14 Syntax of the Instruction CAB

Arithmetic group Instructions

The arithmetic instructions modify the values of numerical operands, allowing arithmetic and logic calculations to be carried out between them. They also allow comparison between values of operands.












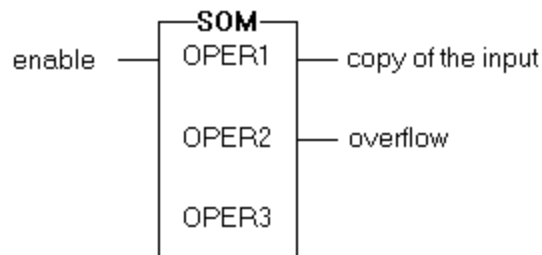
Name	Description of Name	Editing Sequence	Tool Bar
SOM	Sum	ALT, A, S	
SUB	Subtraction	ALT, A, B	
MUL	Multiplication	ALT, A, M	
DIV	Division	ALT, A, D	
AND	Function "and" binary between operands	ALT, A, A	
OR	Function "or" binary between operands	ALT, A, O	
XOR	Function "or exclusive" binary between operands	ALT, A, X	
CAR	Load operands	ALT, A, C	
IGUAL	Equal	ALT, A, I	
MENOR	Less than	ALT, A, N	
MAIOR	More than	ALT, A, R	

Table 3-15 Arithmetic Instructions of the Group

SOM - Sum



OPER1 – first plot

OPER2 – second plot

OPER3 - total

Description:

This instruction carries out the arithmetic sum of operands. When the input **enabled** is powered, the values of the specified operands in the first two cells are added and the result stored in the operand of the third cell.

If the result of the operation is more or less than is allowed to be stored, the output **overflow** is powered and the maximum or minimum storable value is attributed the total variable as the result.

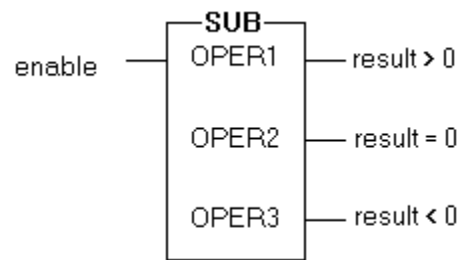
If the input **enable** is not powered, all the outputs are turned off and the value of OPER3 is not altered.

Syntax:

OPER1	OPER2	OPER3
%KD %D	%KD %D	%D

OPER1	OPER2	OPER3
%KF %F %KM %M %KI %I	%KF %F %KM %M %KI %I	%F %M %I

Table 3-16 Syntaxes of the Instruction SOM

SUB - Subtraction

OPER1 – first plot

OPER2 – second plot

OPER3 - result

Description:

This instruction carries out the subtraction arithmetic between operands. When **enable** is powered, the value of the operand of the second cell is subtracted from the first cell. The result is stored in the memory specified in the third cell.

The lines of output **result > 0**, **result = 0** and the **result < 0** can be used for comparisons and are enabled according to the result of the subtraction.

If the input **enable** is not powered, all the outputs are turned off and OPER3 remains unaltered.

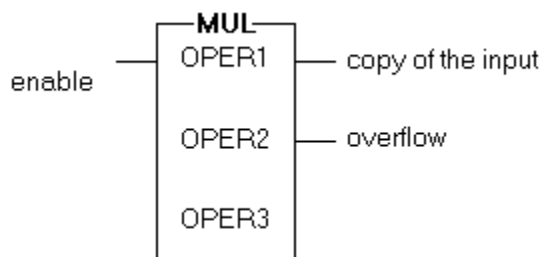
If the result of the operation exceeds the greatest or smallest storable value in the operand, the respective value limit is considered as the result.

Syntax:

OPER1	OPER2	OPER3
%KD %D	%KD %D	%D

OPER1	OPER2	OPER3
%KF %F %KM %M %KI %I	%KF %F %KM %M %KI %I	%F %M %I

Table 3-17 Syntaxes of the Instruction SUB

MUL - Multiplication

OPER1 - multiplied

OPER2 - multiplier

OPER3 - product

Description:

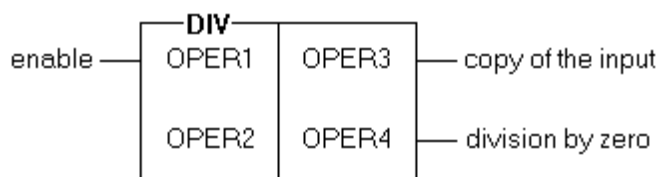
This instruction carries out the multiplication arithmetic of operands. When the input **enable** is powered, the multiplication of the contents of the specified operand takes place in the first cell by those specified in the second.

The result is stored in the specified memory of the third cell. If this is more than the maximum value storable in a memory, the final result is this value and the output **overflow** is powered. If the output **enable** is turned off, no output is lit and OPER3 remains unchanged.

Syntax:

OPER1	OPER2	OPER3
%KF	%KF	
%F	%F	%F
%KM	%KM	%M
%M	%M	%I
%KI	%KI	
%I	%I	

Table 3-18 Syntax of the Instruction MUL

DIV - Division

OPER1 - divided

OPER2 - divider

OPER3 - quotient

OPER4 - remainder

Description:

This instruction carries out the division arithmetic of operands. When the input **enable** is powered, the division of the value of the operand in the first cell by the second takes place, the result being stored in the specified memory in the third cell and the remainder of the operation placed in the fourth operand. The operands of the first and second cells can be of the type memory or constant.

If the value of the second operand is zero, the output division by zero is enabled and the maximum or minimum storable value is placed in the operand, according to the sign of OPER1. In this case, zero will be stored in OPER4 (remainder). The outputs of the instruction are only powered if the input **enable** is enabled. If it is not enabled, OPER3 and OPER4 remain unchanged.

Always that the OPER1 (divided), OPER2 (divider) or OPER3 (quotient) is an operand of real type the fourth parameter (rest) will be unconsidered.

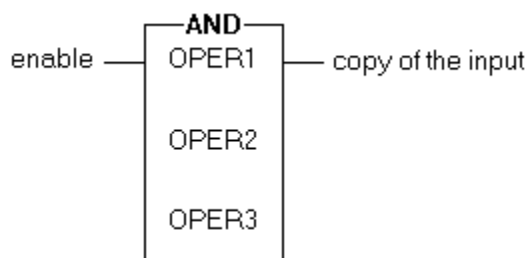
Syntax:

OPER1	OPER2	OPER3	OPER4
%KM %M %KI %I	%KM %M %KI %I	%M %I	%M %I

OPER1	OPER2	OPER3	OPER4
%KF %F %KM %M %KI %I	%KF %F %KM %M %KI %I	%F %M %I	%M (NU)

Table 3-19 Syntax of the Instruction DIV

NU= Not used, any memory can be used.

AND – And binary between operands

OPER1 - first operand

OPER2 - second operand

OPER3 - result

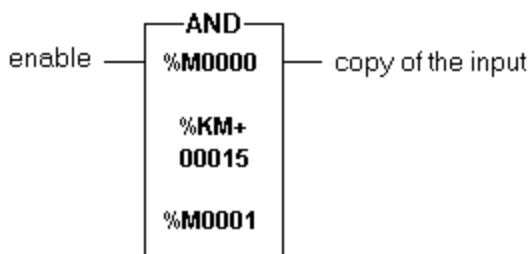
Description:

This instruction carries out the operation “and” binary between the first two operands, storing the result in the third.

The operation is carried out point between the operands. The table to follow shows the possible combinations of the “and” point to point operation.

point OPER1	point OPER2	point OPER3 (result)
0	0	0
0	1	0
1	0	0
1	1	1

Table 3-20 Point to Point Operations

Example:

In this example it is required to keep the less significant value of the nibble of %M0000, zeroing the rest of the operand. If %M0000 contains 215 (11010111 binary), the result of the “and” binary with 15 (00001111 binary) is 7 (00000111 binary).

Decimal	Binary
215	00000000 11010111 (contents of %M0000)
AND 15	AND 00000000 00001111 (value of %KM+00015)
7	00000000 00000111 (result in %M0001)

Therefore, the decimal value 7 is stored in %M0001.

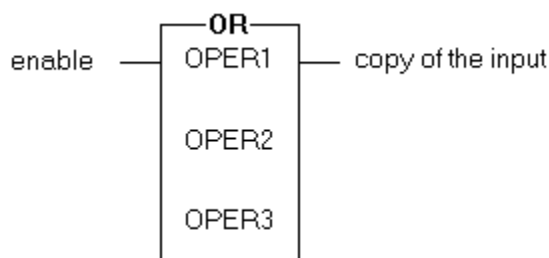
Syntax:

OPER1	OPER2	OPER3
%KM %M	%KM %M	%M

OPER1	OPER2	OPER3
%KD %D	%KD %D	%D

OPER1	OPER2	OPER3
%I %KI	%I %KI	%I

Table 3-21 S Syntaxes of the Instruction AND

OR – Or binary between operands

OPER1 – first operand

OPER2 - second operand

OPER3 - result

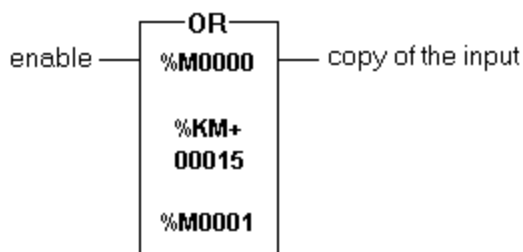
Description:

This instruction carries out the operation “or” binary between the values of the first two operands, storing the result in the third.

The operation is carried out point to point between the operands. The table to follow shows the possible combinations of the operation “or” point to point.

point OPER1	point OPER2	point OPER3 (result)
0	0	0
0	1	1
1	0	1
1	1	1

Table 3-22 Operations Point to Point (OR)

Example:

In this example it is required to force the less significant nibble of %M0000 to 1, saving the value in the other nibbles. If %M000 contains 28277 (0110111001110101 binary) the result is 28287 (0110111001111111 binary).

Decimal	Binary
28277	01101110 01110101 (content of %M0000)
OR 15	OR 00000000 00001111 (value of %KM+00015)
28287	01101110 01111111 (result of %M0001)

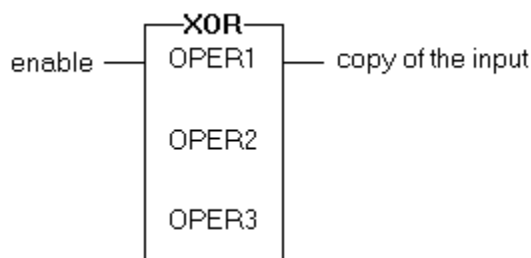
Syntax:

OPER1	OPER2	OPER3
%KM %M	%KM %M	%M

OPER1	OPER2	OPER3
%KD %D	%KD %D	%D

OPER1	OPER2	OPER3
%I %KI	%I %KI	%I

Table 3-23 Syntaxes of the Instruction OR

XOR – Or Exclusive between operands

OPER1 – first operand

OPER2 – second operand

OPER3 - result

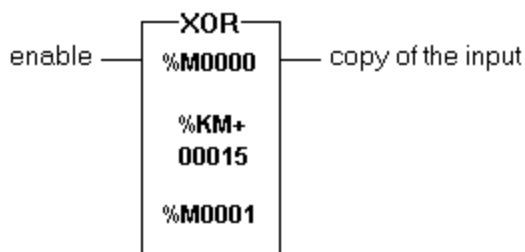
Description:

This instruction carries out the operation “or exclusive” binary between the two first operands, storing the result in the third.

The operation is carried out point to point between the operands. The table to follow shows the possible combinations of the operation “or exclusive” point to point.

Point OPER1	point OPER2	point OPER3 (result)
0	0	0
0	1	1
1	0	1
1	1	0

Table 3-24 Operations Point to Point (XOR)

Example:

In this example it is required to invert the points contained in the less significant nibble of %M0000, saving the rest of the operand. If %M0000 contains 1612 (0000011001001100 binary), the result is 16603 (0000011001000011 binary).

Decimal	Binary
1612	00000110 01001100 (content of %M0000)
XOR 15	XOR 00000000 00001111 (value of %KM+00015)
1603	00000110 01000011 (result of %M0001)

Therefore, the decimal value 1603 is stored in M001.

Syntax:

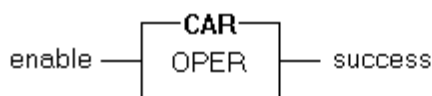
OPER1	OPER2	OPER3
%KM %M	%KM %M	%M

OPER1	OPER2	OPER3
%KD %D	%KD %D	%D

OPER1	OPER2	OPER3
%I %KI	%I %KI	%I

Table 3-25 Syntaxes of the Instruction XOR

CAR – Load Operands



OPER - operand to be loaded

Description:

The instruction loaded in the operand carries the loading of the value of the operand specified in the special internal register in the PLC, for the subsequent use of the instructions of comparison (more than, less than, equals). The operand remains loaded until the next instruction for loading, being able to be used for different logics, including subsequent scan cycles.

The output **success** is enabled if the loading is carried out. If some indirect access of the operand is not possible (invalid index), the output **success** is not enabled.

See considerations and examples shown in the following section, **Instructions of Comparison of Operands**.

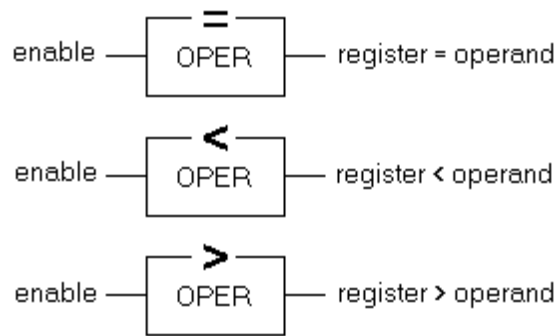
The comparing of decimal operands and real operands is not allowed.

Syntax:

OPER1
%E
%S
%A
%M
%D
%F
%I
%KM
%KD
%KF
%KI
%M*E
%M*S
%M*A
%M*M
%M*D
%M*F
%M*I

Table 3-26 Syntax of the Instruction CAR

Instructions of Comparison of Operands – Equals, More than and Less than



OPER – operand to be compared

Description:

The instructions more than, less than and equals carry out comparisons of the operand specified with the value loaded previously in the internal register with the instruction CAR (Load Operand), supplying the result of the comparison in its outputs. If any indirect access is invalid, the output is disabled.

For example, the instruction more power to its output if the value of the operand present in the last active CAR instruction is greater than the value of its operand. The equals instructions and less than work in an identical way, changing only the type of the comparison carried out.

If the operands to be compared are of the same type, they are compared according to their storage format (taking their signs into consideration). If they are not of the same type, they are compared point to point (as binary values without sign).

The comparing of decimal operands and real operands is not allowed.

WARNING:

It is suggested that operands of equal types are always compared to avoid wrong interpretation in the results when the operands have negative values. C.f. following example.

Example:

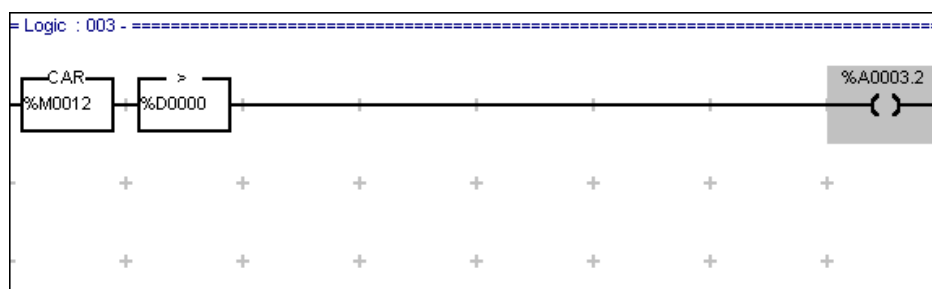


Figure 3-6 Example of Instructions of Comparison

As the types of operands are different (%M and %D), the comparison is carried out point to point, without taking the arithmetic signs into consideration. Due to this fact, if %M0012 has value -45 and %D0010 has the value +21, the operand %A0003.2 will be powered, as if the value of %M0012 is greater than %D0010, which actually is not.

%M0012	= -45					1111	1111	1101	0011
%D0000	= +21	0000	0000	0000	0000	0000	0000	0010	0001

To consider the signs in the comparison of the example, the value of the memory operand should be converted to a decimal, using this last in the instruction CAR, as shown in the logic to follow:

The value 111 111 1101 0011 (%M0012) is greater than 100001 (%D0010) in the comparison point to point. Showing it as a negative value.

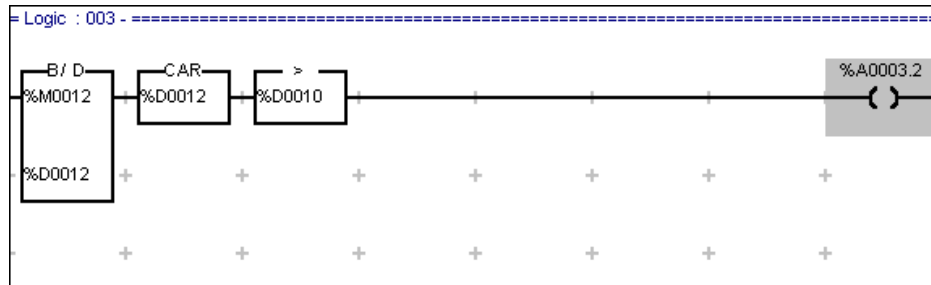


Figure 3-7 Example of the Instructions of Comparison

WARNING:

Due to the processing order of the instructions in the logic, care should be taken in positioning the instructions of comparison to avoid errors in interpretation in its functioning. C.f. section **Logics** in this same chapter and the example to follow.

Example:

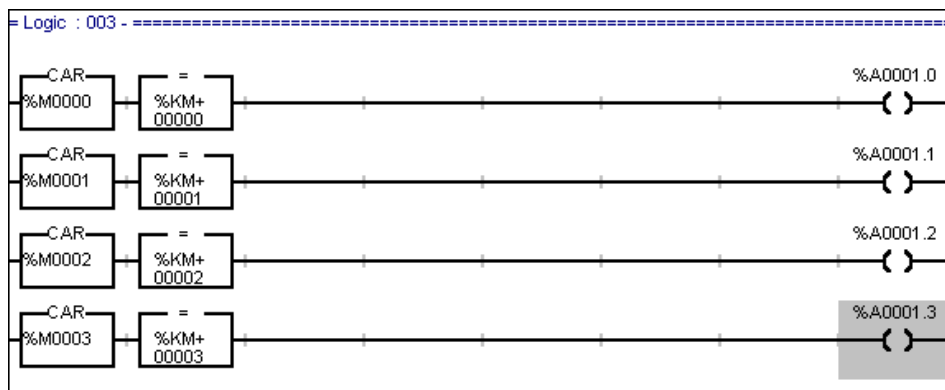


Figure 3-8 Incorrect Use of the Instruction CAR

In the logic shown, it is required to compare the value of the operands %M0000, %M0001, %M0002 and %M0003 with the constants %KM00000, %KM00001, %KM00002 and %KM00003, respectively. However, the functioning. As the processing of the logic takes place in columns, at the end of the execution of column 0 the value of %M0003 will be loaded to the comparisons in column 1. In reality, only the value of the operand %M0003 will be compared with the constants present in column 1.

For the required functioning, the logic should be programmed in the following way:

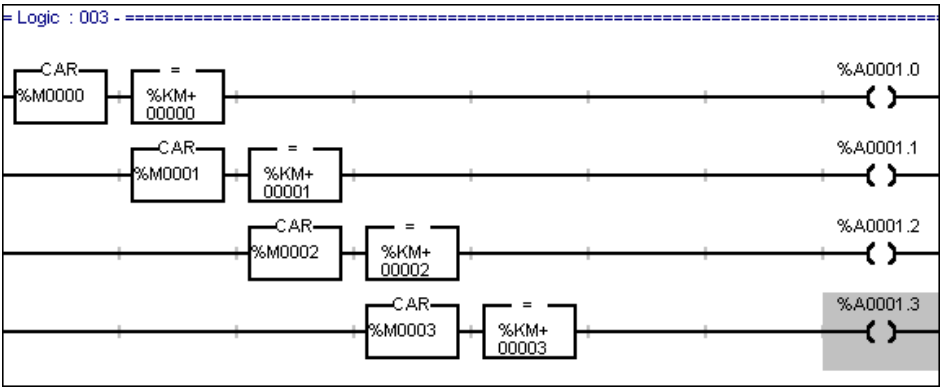


Figure 3-9 Correct Use of the Instructions CAR

WARNING:
To avoid wrong interpretations in the functioning of the comparison, it is suggested to use only one instruction CAR for the column of the logic.

Syntax:

OPER1
%E
%S
%A
%M
%D
%F
%I
%KM
%KD
%KF
%KI
%M*E
%M*S
%M*A
%M*M
%M*D
%M*F
%M*I

Table 3-27 Syntax of the Instructions More than, Equals and Less than

Instructions of counters group

The counter instructions are used to carry out counts of events or the time of the applications program.





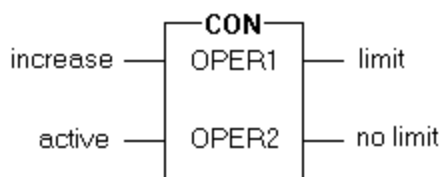
Name	Description of Name	Editing Sequence	Tool Bar
CON	Simple counter	ALT, C, N	
COB	Bidirectional counter	ALT, C, B	
TEE	Timer to turn on	ALT, C, T	
TED	Timer to turn off	ALT, C, D	

Table 3-28 Instructions of Group Counters

CON – Simple Counter



OPER1 - counter

OPER2 – count limit

Description:

This instruction carries out simple counts, with the increase of one unit in each enabling.

The instruction simple counter has two operands. The first always of type %M, specifies the memory which writes up the events. The second establishes the value limit of the counting to power of the upper cell and can be of type %KM or operand %M referenced indirectly.

If the input **active** is turned off, the memory in OPER1 is zeroed, the output **no limit** powered and the output **limit** turned off.

When the input **active** is powered, each transition of connection in the input **increase** raises the value of the operand counter (OPER1) by one unit.

If the value of the first operand is equal to the second operand, the output **limit** is powered. The counter variable is not increased with new transitions in the input **increment**, staying with the value limit. If it is less, the output **limit** is turned off. The logic status of the output **no limit** is exactly the opposite of the output **limit**, being the deactivated instruction.

In case of invalid indirect access to the second operand of the instruction, the output no limit is powered.

WARNING:

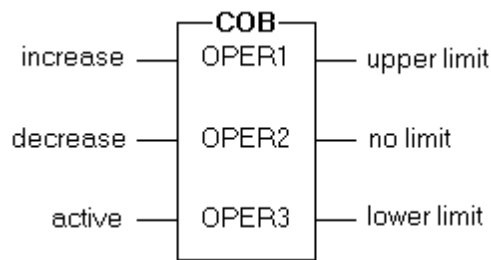
With the input **active** deactivated, the output **no limit** always remains powered, also when the instruction is in a command passage through the instruction RM (master relay). Due to this care should be taken not to carry out unrequired enabling in the logic.

Syntax:

OPER1	OPER2
%M	%KM %M %M*M

Table 3-29 Syntax of Instruction CON

COB – Bidirectional Counter



OPER1 - counter

OPER2 – count step

OPER3 – count limit

Description:

This instruction carries out counts with the value for increase or decrease defined for an operand. The bidirectional counter instruction allows counts in both directions, that is, increases or decreases the contents of type memory.

The first operand contains the accumulated memory of the value counted while the second specifies the value of the increase or decrease required. The third operand contains the value limit of the count.

The count always takes place when the input **active** is powered and the inputs **increase** or **decrease** have a transition from disconnected to connected. If both the inputs have the transition in the same scan cycle of the program, there is no increase nor decrease in the value of the memory declared in OPER1.

If the value of the increase is negative, the input **increase** causes decreases and the input **decrease** causes increases in the value of the count.

If the value of the first operand makes more than or equal to the third operand, the output **upper limit** is powered, not being increased.

If the value of the first operand is equal to or less than zero, the output **lower limit** is enabled, zero being stored in the first operand.

If the value of the first operand is between zero and the limit, the output **no limit** is enabled. If the input **active** is not powered, the output **lower limit** is powered and the first operand is zeroed.

In case of invalid indirect access to any one of the operands of the instruction, the outputs lower limit is powered.

WARNING:

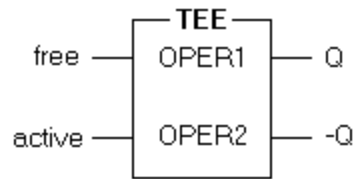
With the input **active** deactivated, the output **lower limit** always remains powered, the same when the instruction is in a passage commanded by the instruction RM (master relay). Due to this care should be taken not to carry out unrequired enabling in the logic.

Syntax:

OPER1	OPER2	OPER3
%M	%M	%M
%M*M	%M*M	%M*M
	%KM	%KM

Table 3-30 Syntax of the Instruction COB

TEE – Timer to turn on



OPER1 – time accumulator

OPER2 – time limit (tenths of seconds)

Description:

This instruction carries out time counts with the powering of its two enabling inputs.

The instruction TEE has two operands. The first (OPER1) specifies the accumulated memory of the time count. The second operand (OPER2) shows the maximum time to be accumulated. The time count is carried out in tenths of seconds, that is to say, each unit increased in OPER1 corresponds to 0.1 seconds.

While the inputs **free** and **active** are powered simultaneously, the operand OPER1 is increased by each tenth of a second. When OPER1 is more than or equal to OPER2, the output **Q** is powered and **-Q** turned off, OPER1 keeping the same value as OPER.

In the disabling of the input **free**, there is an interruption in the count time, OPER1 keeping the same value. Disabling the input **active**, the value in OPER1 is zeroed.

If OPER2 is negative or the indirect access is invalid, OPER1 is zeroed and the output **-Q** is powered.

The logic status of output **Q** is exactly the opposite of the output **-Q** being the deactivated instruction.

WARNING:

With the input **active** deactivated, the output **-Q** always remains powered, the same when the instruction is in a passage commanded by the instruction RM (master relay). Due to this care should be taken not to carry out unrequired enabling in the logic.

Diagram of Times:

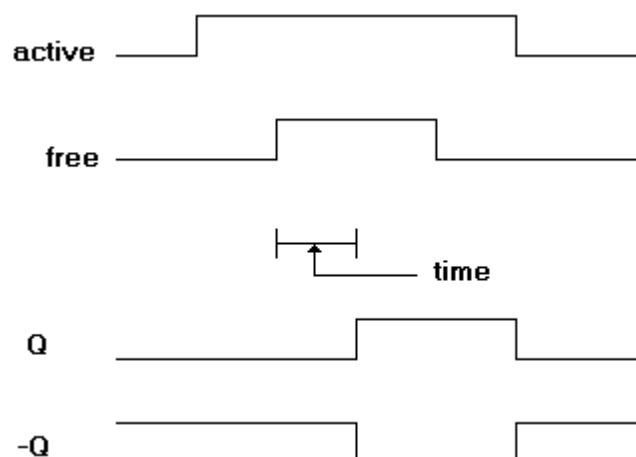


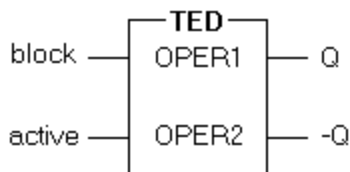
Figure 3-10 Diagram of Times of the Instruction TEE

Syntax:

OPER1	OPER2
%M	%M %M*M %KM

Table 3-31 Syntax of the Instruction TEE

TED – Timer to turn off



OPER1 – time accumulator

OPER2 – time limit (tenths of seconds)

Description:

This instruction carries out the time counts with the turning off its enabling input.

The instruction TED has two operands. The first (OPER1) specifies the accumulated memory of the time count. The second operand (OPER2) shows the maximum time to be accumulated. The time count is carried out in tenths of seconds, that is to say, each unit increased in OPER1 corresponds to 0.1 seconds.

While the input **active** is powered and the input **block** turned off, the operand OPER1 is increased by each tenth of a second. When OPER1 is greater than or equal to OPER2, the output **Q** is turned off and **-Q** powered, OPER1 keeping the same value as OPER2.

The output **Q** always powered when the input **active** is powered and OPER1 is less than OPER2.

Enabling the input **block**, there is an interruption in the time count, while disabling the input **active**, the time of the accumulator is zeroed and the output **Q** is disabled.

If OPER2 is negative or the indirect access is invalid, OPER1 is zeroed and the output **Q** is powered.

The logic status of output **-Q** is exactly the opposite of the output **Q**, being the deactivated instruction.

WARNING:

With the input **active** deactivated, the output **- Q** always remains powered, the same when the instruction is in a passage commanded by instruction RM (master relay). Due to this care should be taken not to carry out unrequired enabling in the logic.

Diagram of Times:

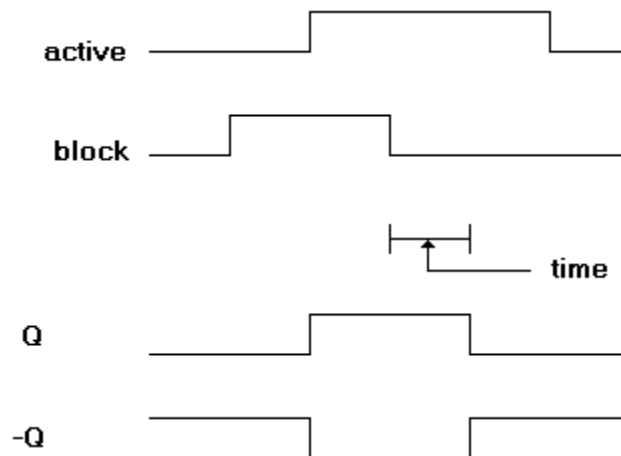


Figure 3-11 Diagram of Times of Instruction TED

Syntax:

OPER1	OPER2
%M	%M %M*M %KM

Table 3-32 Syntax of the Instruction TED

Instructions of the Conversion Group

This group has instructions which allow the conversion between the formats of storing the values used in the operands of the applications program and accesses to analog modules in the input and output bus.



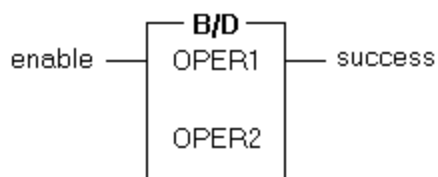
Name	Description of Name	Editing Sequence	Tool Bar
BIN/DEC	Conversion binary-decimal	ALT, V, B	
DEC/BIN	Conversion decimal-binary	ALT, V, D	

Table 3-33 Group Converter Instructions

B/D - Conversion Binary-Decimal

OPER1 - origin

OPER2 - destination

Description:

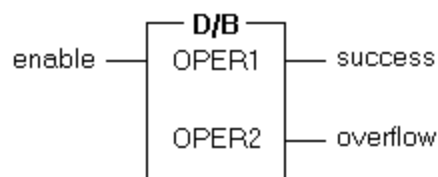
This instruction converts values stored in binary format, contained in memory operands (%M), to decimal format (BCD), storing them in decimal operands (%D).

The binary value contained in the first operand (OPER1) is converted to decimal value and stored in the second operand (OPER2). The output **success** is enabled and the conversion is carried out correctly. If any invalid indirect access happens to the operand, the output **success** is not powered.

Syntax:

OPER1	OPER2
%M	%D
%M*M	%M*D

Table 3-34 Syntax of the Instruction B/D

D/B - Conversion Decimal-Binary

OPER1 - origin

OPER2 - destination

Description:

This instruction converts values stored in decimal format, contained in decimal operands (%D), to binary format, storing them in memory operand (%M).

The decimal value contained in the first operand (OPER1) is converted to binary value and stored in the second operand (OPER2). The output **success** is enabled if the conversion is correctly carried out. If any invalid indirect access to the operand happens, the output **success** is not powered.

If the value converted results in a value greater than the maximum storable in operands %M, the output **success** is not powered, the limit value being stored in the destination operand. In this case, the output **overflow** is powered.

Syntax:

OPER1	OPER2
%D %M*D	%M %M*M

Table 3-35 Syntax of the Instruction D/B

Instructions of the General Group

The general group instructions allow the testing and enabling of points indirectly, implementations of status machines, calls for procedures and functions.









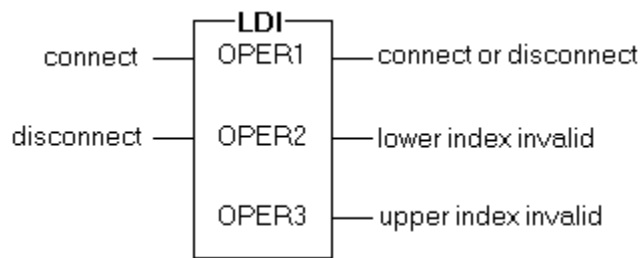
Name	Description of Name	Editing Sequence	Tool Bar
LDI	Connect or disconnect indexed points	ALT, G, L	
TEI	Status test of indexed points	ALT, G, T	
SEQ	Sequencer	ALT, G, S	
CHP	Call the procedure module	ALT, G, P	
CHF	Call the function module	ALT, G, F	
ECH	Write operands on another PLC for Ethernet	ALT, G, E	
LTH	Read operands from another PLC for Ethernet	ALT, G, T	
LAH	Free images update for Ethernet	ALT, G, A	

Table 3-40 Instructions of the general group

LDI – Connect/Disconnect indexed

OPER1 - address of point to be connected or disconnected

OPER2 - address lower limit

OPER3 - address upper limit

Description:

This instruction is used to connect or disconnect indexed points for a memory, delimited by operands of upper and power limit.

The first operand specifies the memory whose contents reference the auxiliary operand, input or output to be connected or disconnected. It should be declared as the operand of indirect access to the operand %E or %A (%MXXXX*E or %MXXXX*A). The same when the instruction is used to connect or disconnect points of output (%O), the representation in this operand will be as indirect access to the input (%MXXXX*E).

The second operand the address of the first valid output or auxiliary relay in the instruction. It should be specified with subdivision of point (%RXXXX.X, %SXXXX.X or %AXXXX.X).

The third operand specifies the address of the last output relay or valid help in the instruction. It should be specified with subdivision of point (%EXXXX.X, %SXXXX.X or %AXXXX.X).

If the inputs **connect** or **disconnect** will be enabled, the point specified by the value contained in the memory operand (OPER1) is connected or disconnected if there is a limit for OPER2 and OPER3 in the addresses areas. For example, if these operands correspond to %S0003.3 and %S0004.5, respectively, this instruction only acts for the elements of %S0003.3 to %S0003.7 and from %S0004.5.

If the relay or help pointed at the memory index is outside the defined limits for the defined limits for the parameters of the second and third cells, the output **upper index invalid** or **lower index invalid** is connected. The output of the first cell is enabled if any one of the inputs **connect** or **disconnect** is powered and the access is correctly carried out.

If the inputs remain disabled, all the outputs of the instruction remain turned off.

If both the inputs are powered simultaneously, no operation is carried out, and all the turned off.

In OPER1 a value which specifies the required point should be loaded to connect or disconnect, according to the following formula:

$$\text{VALUE OPER1} = (\text{OCTET} * 8) + \text{POINT}$$

Example:

For example, if S0010.5 is the point requires to be connected indirectly, then:

$$\text{OCTET} = 10$$

$$\text{POINT} = 5$$

$$\text{VALUE OPER1} = (10 * 8) + 5 = 85$$

The value to be loaded in OPER1 is 85.

WARNING:

This instruction allows the points of the operands %E to be connected or disconnected indirectly superimposing the value of the scan of the input modules after their execution.

Syntax:

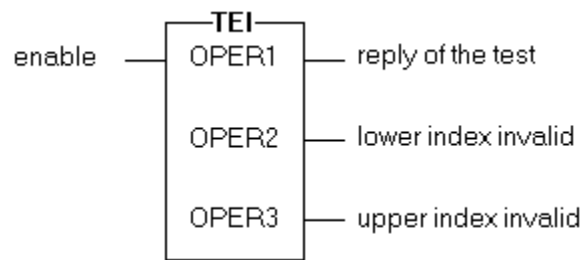
OPER1	OPER2	OPER3
%M*E	%E	%E

OPER1	OPER2	OPER3
%M*S	%S	%S

OPER1	OPER2	OPER3
%M*A	%A	%A

Table 3-41 Syntaxes of the Instruction LDI

TEI – Test of Indexed Status



OPER1 - address of point to be tested

OPER2 - address lower limit

OPER3 - address upper limit

Description:

This instruction is used to test the status of the points indexed for a memory, delimited for operands of lower and upper limit.

The first operand specifies the memory whose contents reference the auxiliary operand or output relay to be tested. The operand %E or %A (%MXXXX*E or %MXXXX*A) should be declared as the operand of indirect access. The same when the instruction is used to test output points (%S), the representation of this operand will be as indirect access to the input (%MXXXX*E).

The second operand specifies the address of the valid output or auxiliary relay in the instruction. It should be specified with the subdivision of point (%EXXXX.X, %SXXXX.X or %AXXXX.X).

The third operand specifies the address of the last a valid output or auxiliary relay in the instruction. It should be specified with the subdivision of point (%EXXXX.X, %SXXXX.X or %AXXXX.X).

If the input **enable** is powered, the status of the relay or auxiliary specified for the value contained in the memory index (OPER1) is examined. According to whether they are 1 or 0, the output **answer** is connected or not.

The point indexed by memory is tested if it is in the area of addresses limited for OPER2 and OPER3. For example if these operands corresponds to %S0003.3 and %S0004.5, respectively, this instruction only acts for the elements of %S0003.3 to %S0003.7 and from %S0004.0 to %S0004.5.

If the relay or auxiliary pointed at the memory index is outside the limits defined by the parameters of the second and third cells, the output **upper index invalid** or **lower index invalid** is connected the output of the first cell disconnected. This verification is only carried out at the moment when the input **enable** is powered.

The calculation of the value to be stored in the first operand, to reference the required point, is the same specified in the instruction **LDI**.

Syntax:

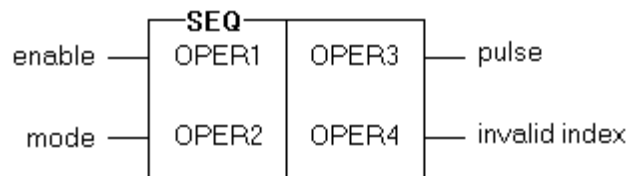
OPER1	OPER2	OPER3
% M * E	% E	% E

OPER1	OPER2	OPER3
% M * S	% S	% S

OPER1	OPER2	OPER3
% M * A	% A	% A

Table 3-42 Syntaxes of the Instructions TEI

SEQ - Sequencer



OPER1 - table of conditions or first table of statuses

OPER2 - index of the table(s) (current status)

OPER3 - operand base of the first series of conditions

OPER4 - operand base of the second series of conditions

Description:

This instruction allows the programming of complex sequencer with specific conditions of evolution for each status. Its form of programming is similar to “state machine”.

The instruction can be executed in two modes: the 1000 mode and the 3000 mode. When the input **mode** is turned off, the instruction is executed in 3000 mode. In the 3000 mode more complex sequences can be programmed.

Mode 1000:

In this mode a fixed sequence of evolution of the statuses occurs. The evolution always happens from the current status to the following one, and from the last to the first.

The first operand specifies a table where each position contains the address of an auxiliary operand point which is tested as a condition of evolution for the next status.

The second operand specifies a memory which stores the current status and serves from index to a specified table in the first operand.

The third operand is irrelevant, however an operand of type memory or auxiliary should be specified in this cell, since MasterTool achieves the consistency according to the 3000 mode.

The fourth operand is irrelevant, however it should be specified in an operand of type memory or auxiliary in this cell, since MasterTool achieves consistency in accordance with the 3000 mode.

When the input **enable** is turned off, the outputs **pulse** and **invalid index** are turned off, independent of any other condition. When the input **enable** is powered, the **pulse** output is normally powered, and the output **invalid index** is normally turned off.

Beyond this, when the input **enable** is powered, the table position (OPER1) indexed by the current status (OPER2) is accessed and the auxiliary operand point referenced in this table position is examined. If this point is powered, the contents of OPER2 is increased (or zeroed, if it is pointed at the last table position OPER1) and a turning off **pulse** occurs in the output pulse with the duration of a program cycle. If the point examined is turned off nothing happens and the memory value in OPER2 remains unchanged.

The output **invalid index** is activated if the memory OPER2 (current status) contains a value which indexes a non-existent position in the table specified in OPER1. This can happen by modifying the memory OPER2 at one point of the applications program outside the instruction SEQ (in the Initialization of OPER2, for example). Care should be taken to define and initialize the table specified in OPER1 with the legal values.

The values in decimal format which specify the points of auxiliary operands which have to be tested as conditions of evolution should be loaded into the table specified in OPER1. The calculation of these values is specified through the equation:

$VALUE = (\text{address of the operand} * 8) + \text{address of the subdivision}$

Example:

If %A0030.2 is the point which it is required to use as a condition of evolution starting from the status 4, then:

Address of operand = 30

Address of subdivision = 2

$VALUE = (30 * 8) + 2 = 242$

The value to be loaded in position 4 of the table OPER1 should be 242 so that the point %A0030.2 causes the evolution for the next status, that is the status 5 (or the status 0, if the table has 5 positions).

Mode 3000:

In this mode it is possible to define the evolution sequence and choose one of two paths starting from the current status. Therefore, 2 degrees of freedom are offered in relation to the 1000 mode, allowing more complex status machines to be used.

The first operand specifies the first of the two subsequent tables that are used for each instruction. The two tables have to be the same size. Each position of the first table contains the next status if the condition associated to operand 3 is powered. Each position of the second table contains the next status if the condition associated to the operand 4 is powered.

The second operand specifies a memory which shows what the current status is and serves as an index for the tables specified in the first operand.

The third operand specifies an operand which serves from base to determine the condition of evolution starting from the status OPER2 to the status indexed for OPER2 in the first table.

The fourth operand specifies an operand which serves from base to determine the condition of evolution starting from the status OPER2 for the status indexed for OPER2 in the second table.

When the input **enable** is turned off, the outputs **pulse** and **invalid index** are turned off, independent of any other condition. When the input **enable** is powered, the **pulse** output is normally powered, and the output **invalid index** is normally turned off.

After this, when the input **enable** is powered, the instruction searches the value of the memory OPER2 (current status) and tests the respective condition of evolution with base in OPER3. If this condition is powered, the operand OPER2 is loaded with a new status, indexed through operand OPER2 in the first table specified for OPER1. If the condition of evolution associated with OPER2 and with the base in OPER3 is turned off, it tests the evolution condition associated to OPER2 and with base in OPER4. If this last condition is powered, the operand OPER2 is loaded with a new status, indexed through its own operand OPER2 in the second table specified for OPER1. If at least one of the 2 conditions above are powered, a status transition occurs, and a turning off pulse with the duration of an applications program cycle takes place in the **pulse** output of the instruction. If neither of the 2 conditions are powered, nothing happens and the value of memory OPER2 (current status) remains unchanged, as well as the **pulse** output continuing powered.

The output **invalid index** is activated if the memory OPER2 contains a value which indexes a non-existent position in the tables specified in OPER1. This can happen by modifying the memory OPER2 in one point of the applications program outside of the instruction SEQ (in the Initialization of OPER2, for example) or in the appropriate SEQ instruction, if any of the positions of the tables specified in OPER1 contain invalid values for being the next status. Care should be taken to define the 2 tables specified for OPER1 with the same size, and they should be initialized with legal values (example: if the tables have 10 positions, only values between 0 and 9 should be loaded in positions of this table, since only these can have legal status).

The conditions of evolution associated to the current status (OPER2) are determined with base in OPER3 (next status is loaded starting from the first table) or with base in OPER4 (next status is loaded starting from the second table). Knowing that the operands OPER3 and OPER4 are of memory type (16 bits) or of auxiliary type (8 bits), suppose the following is the case:

ESTADO = contents of operand OPER2 (current status)

END3 = address of OPER3

END4 = address of OPER4

END1 = address of point to be tested, with base in OPER3

SUB1 = subdivision of point to be tested, with base in OPER3

END2 = address of point to be tested, with base in OPER4

SUB2 = subdivision of point to be tested, with base in OPER4

The points tested as evolution condition associated to each table are:

$M\langle \text{END1} \rangle.\langle \text{SUB1} \rangle$ or $A\langle \text{SUB1} \rangle$ (first table) and $M\langle \text{END2} \rangle.\langle \text{SUB2} \rangle$ or $A\langle \text{END2} \rangle.\langle \text{SUB2} \rangle$ (second table)

where:

$\text{END1} = \text{END3} + \text{STATUS}/16$ (if operand %M)

$\text{END1} = \text{END3} + \text{STATUS}/8$ (if operand %A)

$\text{SUB1} = \text{REST}(\text{STATUS}/16)$ (if operand %M)

$\text{SUB1} = \text{REST}(\text{STATUS}/8)$ (if operand %A)

$\text{END2} = \text{END4} + \text{STATUS}/16$ (if operand %M)

$\text{END2} = \text{END4} + \text{STATUS}/8$ (if operand %A)

$\text{SUB2} = \text{REST}(\text{STATUS}/16)$ (if operand %M)

$\text{SUB2} = \text{REST}(\text{STATUS}/8)$ (if operand %A)

Example:

They may be:

OPER1 = %TM000

OPER2 = %M0010

OPER3 = %M0100

OPER4 = %A0020

Where:

%TM000	Position	Value
	000	00001
	001	00002
	002	00004
	003	00001
	004	00000

%TM001	Position	Value
	000	00001
	001	00003
	002	00001
	003	00004
	004	00000

%M0010 = 00001

%M0100	XXXXX
%M0101	XXXXX
%M0102	XXXXX
%M...	...

%A0020	XXXXX
%A0021	XXXXX
%A0022	XXXXX
%A...	...

Then the evolution starting from status 1 are:

For the first table:

- $100 + 1/16 = 100$
- $\text{rest}(1/16) = 1$
- point to be tested = %M0100.1

For the second table:

- $20 + 1/8 = 20$
- $\text{rest}(1/8) = 1$
- point to be tested = %A0020.1

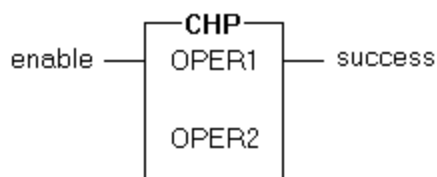
Based on the conditions of %M0100.1 and %A0020.1 we have, starting from one of the tables, the new status of the operand %M0010:

%M0100.1	%A0020.1	%M0010	Observation
0	0	00001	No status changing
0	1	00003	Status changing according to %TM001
1	0	00002	Status changing according to %TM000
1	1	00002	Status changing according to %TM000 (OPER3 have priority over OPER4)

Syntax:

OPER1	OPER2	OPER3	OPER4
% TM % M*TM	% M % M*M	% M % A	% M % A

Table 3-43 Syntax of the Instruction SEQ

CHP – Call the Procedure Module

OPER1 – name of module to call

OPER2 – number of module to call

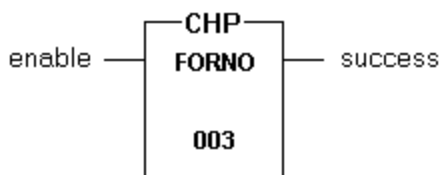
Description:

This instruction carries out the diversion of the processing of the current module to the Procedure module specified in their operands, if it is present in the PLC. At the end of the execution of the module called, the processing returns to the instruction following the CHP. There is no passing of parameters to the module called.

The first operand (OPER1) is documentation and specifies the name of the module to be called. The second operand (OPER2) specifies the number of this module, the fact that the module called is of type procedure being implicit.

If the module called does not exist, the output success is turned off and the execution continues normally after the instruction. The name of the module is not considered for the PLC for the call but only its number. If there is a module P with the same number as the module called, however with different name, this same module is executed like this.

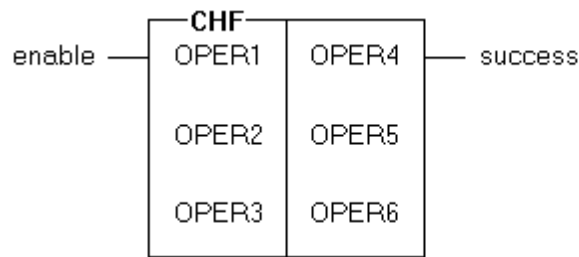
C.f. section **Use of Modules P and F** in chapter 2 of this manual.

Example:*Syntax:*

OPER1	OPER2
NOME	NÚMERO

Table 3-44 Syntax of CHP Instruction

CHF – Call Function Module



OPER1 - name of module to call

OPER2 - number of parameters to send

OPER3 - number of parameters to return

OPER4 - number of module to call

OPER5 - list of parameters to send

OPER6 - list of parameters to return

Description:

The instruction the function Module carries out diversion of the processing of the current module to the module specified, if this is present in the PLC. At the end of the execution of the module called, the processing returns to the instruction following the CHF.

The name and number of the module should be declared as operands OPER1 and OPER4 respectively, the fact that the module called is of function type being implicit. If the module called does not exist in the controller, the execution continues normally after the call instruction, with the output succeeded disconnected from it. The name of the module is not taken into consideration by the PLC, being in the applications program only as a documentation reference, only its type and number being taken into consideration for the call. If there is a module F with the same number called but a different name, this module is executed.

The passing of values of operands (parameters) to the module called and vice-versa after its execution. In the fifth cell of the instruction (OPER5) a list of operand to be sent to the module called is specified. Before the execution of the module, the values of these operands are copied to the operands specified in the list of parameters of the input of the module F, declared in the MasterTool option **Parameters** when it was programmed.

After calling for the execution of module F, the values of the operands declared in the list of parameters of output (option MasterTool **Parameters** in its programming) are copied to the operands declared in the list of operands to return from the instruction CHF (OPER6). Having finalized the copy of the return, the processing continues in the instruction following the call.

WARNING:

MasterTool does not achieve any consistency in relation to the operands programmed as parameters, as much in the CHF instruction as in module F.

The list of operands to be sent to module F should count the same number of operands with the same type of them declared as input parameters of the module, so that the copy of their values is correctly made. The copy of the operands is carried out in the same order in which they are arranged in the list. If one of the list has fewer operands in relation to another, the values of the surplus operands are not copied. If the operands have different types, the copy of the values is carried out with the same rules used in the instruction MOV (simple moving of operands). This principal is also valid for the list return parameters of Module F.

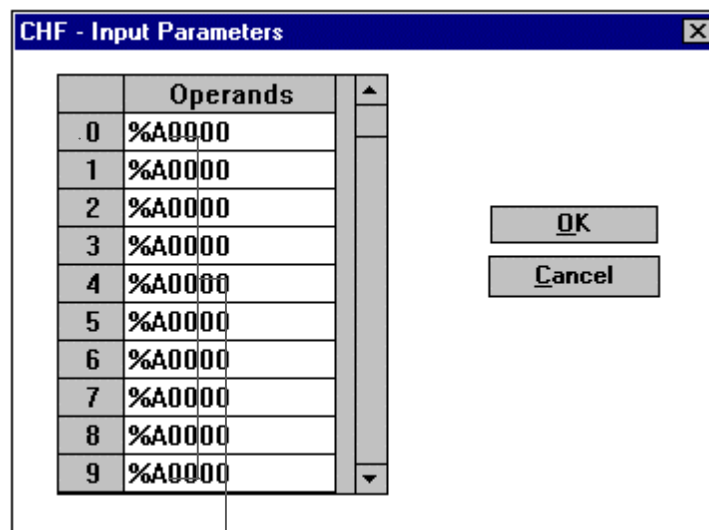
The passing of parameters is carried out with the copy of values of declared operands (parameter passing for value), although these operands still remain in overall use, usable for any module present in the PLC. The F module can be programmed in generic form, to be re-used in different applications programs as new instructions. It is advisable that they use their own operands, not used for any other module present in the applications program, avoiding inadvertent alteration in operands used in other modules.

The passing of simple operands and constants for module F is possible. The passing of tables as parameters is not permitted, due to the long time that is needed to copy the contents of module F. Meanwhile, the address of a table can be passed to Module F contained in an operand memory and indirect access to the table is carried out in this module.

It's not possible to pass operands with subdivisions for module F, for example %M004.2, %A0021n1, etc. Only simple operands should be used.

To carry out the editing of parameters

1. Declare the number of parameters to send and return in OPER2 and OPER3, limited to 10 for each one (%KM + 00000 to %KM + 00010).
2. Select the button **Input**. The window **CHF - Input Parameters**.
3. Place the cursor on the index to be editing and key in the address or tag of the required operand for that position.
4. Repeat step until all the operand used as input parameters have been edited.
5. Select button **Ok**.
6. To edit the output parameters of the CHF, repeat step 2 selecting the button **Output**, and after repeat steps 3, 4 and 5.



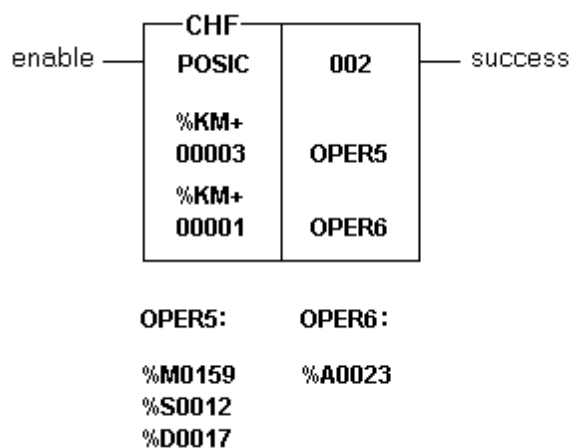
Key in the used operands as input parameters

Figure 3-12 Dialogue Box CHF - Input Parameters

C.f. section **Use of Modules P and F** in chapter 2 of this manual.

If the value of OPER2 or OPER3 is more than 10, MasterTool considers such a value as equal to 10 (%KM + 00010).

Example:



Syntax:

OPER1	OPER2	OPER3	OPER4	OPER5	OPER6
NAME	%KM	%KM	NUMBER	%KM %KD %KF %TM %TD %TF %M %D %F %E %S %A	%KM %KD %KF %TM %TD %TF %M %D %F %E %S %A

Table 3-45 Syntax of Instruction CHF

ECH – Write of Operands on Another PLC for Ethernet



OPER1 – IP address of the remote controller

OPER2 – not used

OPER3 – instruction control operand

OPER4 – edition operand window

Description:

This instruction carries the writing of values of operands of the controller where it is being executed in operands presented in other PLCs, through the Ethernet communication. For its use, therefore, it is essential that the controller who executes is connected to other PLCs through the Ethernet.

Through the ECH can be transferred individual values of operands or sets of operands, being possible the programming of up to 6 different communications in one same instruction.

To program the instruction, it must be declared in the first cell OPER1 the IP address of the programmable controller destination that will receive the written values.

On the third cell OPER3 must be declared a decimal operand (%D) to be used by the proper instruction in the control of its processing.

WARNING:

The %D operand programmed on OPER3 cannot have its value modified in none another point of the applicatory program for the correct functioning of the ECH. Consequently , each new instruction ECH or inserted LTH in the applicatory program must use an %D operand different from the others. This operand cannot be retainer.

To carry through the edition of the ECH parameters

1. Select the PLC button. The Parameters dialog is presented.

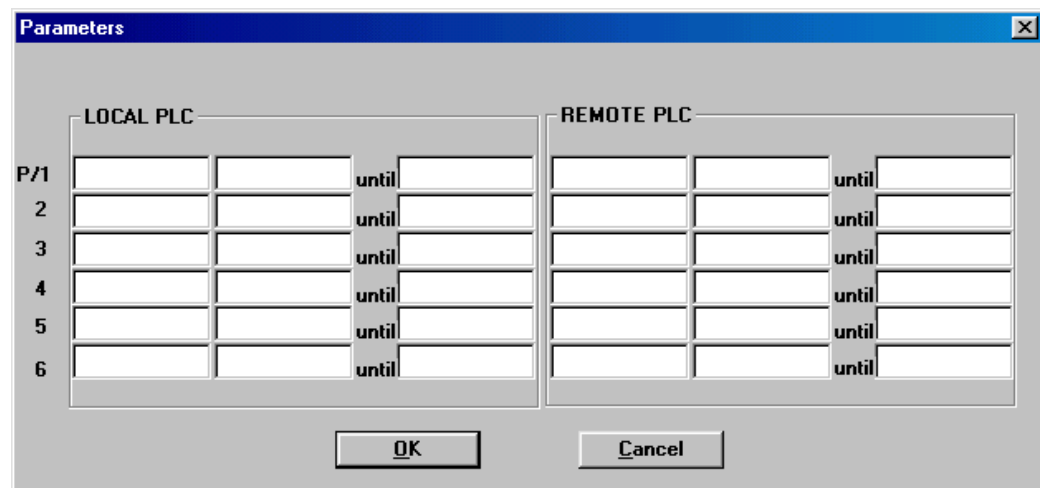


Figure 3-13 Dialog box parameters

This box of dialogue is divided in two parts: REMOTE CPU and LOCAL CPU, each one contend three columns. In the three columns that compose the local CPU can be defined the operand or the group of operands whose values will be sent to the programmable controller destination. In the columns pertaining to the remote CPU, will be declared the operands that will receive the values in the destination controller, being able to be different of the local CPU types. The dialogue box have

six lines, allowing that up to six different communications can be defined in same instruction ECR to the same destination controller.

The operands specified for the local CPU are consisted by the MasterTool in accordance with the constant declarations in module C presented, by belonging to the applicatory program that is being edited. The operands declared for the remote CPU do not suffer consistency in relation to the type and addresses, by belonging to a applicatory program of another programmable controller. However, the number of busy bytes for the operands block declared in the local CP must be equal to the number of bytes busied by the operands of the remote CPU in each communication, for the correctly carried through writing. The maximum number of possible bytes busy by a block of operands in each communication is limited in 220.

Following are related the types of possible operands to be programmed for the local and remote CPU, with the correct disposal in the edition columns and its respective meanings.

LOCAL PLC or REMOTE PLC	Meaning
%XXXXX	Individual Operand %XXXXX
%SXXXX	Individual Operand %SXXXX
%AXXXX	Individual Operand %AXXXX
%MXXXX	Individual Operand %MXXXX
%DXXXX	Individual Operand %DXXXX
%FXXXX	Individual Operand %FXXXX
%XXXX .. %YYYY	Operands Group %XXXX a %YYYY
%SXXXX .. %YYYY	Operands Group %SXXXX a %YYYY
%AXXXX .. %YYYY	Operands Group %AXXXX a %YYYY
%MXXXX .. %YYYY	Operands Group %MXXXX a %YYYY
%DXXXX .. %YYYY	Operands Group %DXXXX a %YYYY
%FXXXX .. %YYYY	Operands Group %FXXXX a %YYYY
%TMXXXX YYY	Table %TMXXXX position YYY
%TDXXXX YYY	Table %TDXXXX position YYY
%TFXXXX YYY	Table %TFXXXX position YYY
%TMXXXX III .. FFF	Table %TMXXXX position III a FFF
%TDXXXX III .. FFF	Table %TDXXXX position III a FFF
%TFXXXX III .. FFF	Table %TFXXXX position III a FFF

Table 3-47 Operands to Local PLC and Remote PLC on ECH

The MasterTool allows the free edition of the operands inside of a same line, making possible the exchange of columns with the aid of the horizontal arrows keyboard keys movement. The consistency are carried through in attempt of line exchange (vertical arrows) or confirmation of the edited content in the window with the ENTER keyboard key. The ESC Keyboard key can given up the carried through alterations, remaining the instruction with the previous content before the opening of the edition window.

The following table shows the busy number of octets for each type of possible operand of being programmed in the definitions of writings.

Operand	Number of bytes
%E	1
%S	1
%A	1
%M	2
%F	4
%D	4
%TM	2 per position
%TD	4 per position
%TF	4 per position

Table 3-48 Number of bytes of the operands on ECH

The calculation of the busy number of bytes in declarations of the local and remote CPU is carried through by multiplying the number of declared operands by the number of octets of the corresponding type. On the following table, some examples are shown.

LOCAL PLC or REMOTE PLC	Calculus	Bytes
%E0004	1 operand x 1 byte	1
%S0020	1 operand x 1 byte	1
%A0018	1 operand x 1 byte	1
%M0197	1 operand x 2 bytes	2
%D0037	1 operand x 4 bytes	4
%E0005 .. %E0008	4 operands x 1 byte	4
%S0024 .. %S0031	8 operands x 1 byte	8
%A0089 .. %A0090	2 operands x 1 byte	2
%M0002 .. %M0040	39 operands x 2 bytes	78
%D0009 .. %D0018	10 operands x 4 bytes	40
%TM0031 101	1 position x 2 bytes	2
%TD0002 043	1 position x 4 bytes	4
%TM0000 000 .. 002	3 positions x 2 bytes	6
%TD0007 021 .. 025	5 positions x 4 bytes	20

Table 3-49 Examples of length in Bytes

Enabling the enable input, is turned on the communication of the first present writing in the ECH, being powered the busy output. At the moment that this communication is completed, the instruction turns on the next writing, independently of the state of the enabling input, repeating this procedure for the other existing communications in this instruction. On the end of the last writing, the busy output of ECH is unpowered, with the application of a pulse with duration of one sweeping in the error output in case that it has not been possible to carry through some communication.

On six first nibbles of D operand programmed in OPER3 the states of the six communications of the instruction are placed. The last two nibbles are used for the control of its processing.

%D operand programmed in OPER3 on ECR and LTR instructions

7	6	5	4	3	2	1	0
		1	2	3	4	5	6

Control of the instruction Communication Status

Communication status 1 - Nibble 5
 Communication status 2 - Nibble 4
 Communication status 3 - Nibble 3
 Communication status 4 - Nibble 2
 Communication status 5 - Nibble 1
 Communication status 6 - Nibble 0

Figure 3-14 Operand of Control of the Instruction ECH and LTH

The state of the communication stored in each nibble is codified in the following form:

- 0 - communication with success
- 1 - not defined operand
- 2 - local controller address equal to the remote (communication with the CPU)
- 3 - invalid operand box
- 4 - invalid operand type
- 5 - package transmission timeout
- 6 - no space on the transmission waiting stack
- 7 - lacking of transmission buffer
- 8 - solicitation timeout
- 9 - hardware error
- 10 - protected remote CPU

In summary, to execute an ECH instruction all the existing communications are carried through, even if the enabling input is unpowered. When all the writings are completed, the next found instruction ECH or LTH in the applicatory program with the enable input powered becomes active, starting to process its communications.

WARNING:

The application program cannot carry through jumps on the active instruction ECH or do not execute the module that contains it, to assure its correct processing.

In a applicatory program being executed in the CPU, only one instruction of access to the Ethernet net (ECH or LTH) is considered active, even if some other instructions with input enable exist.

The busy output determines which is the active instruction, being able to be used to synchronize the communications with the applicatory program. To prevent overloads in the traffic of information in the net, it is advised turn on the ECH instructions periodically, preventing to permanently keep its enabled in the applicatory program, if possible. A recommended procedure is disconnect the enable input after that the busy output is powered, preventing a new enabling of the instruction after its ending.

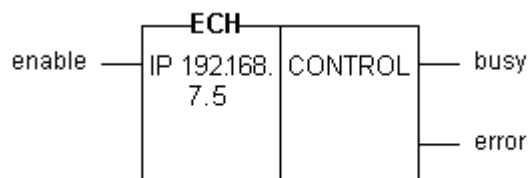
If the instruction is programmed specifying that the IP address is equal to the address of the proper controller who executes it (written of proper values), the error output is powered. Case none operand has been defined on OPER4, the error and busy output keep unpowered.

Syntax of the Instruction:

OPER1	OPER2	OPER3	OPER4
IP Address		%D	COMMUNICATIONS

Table 3-50 Syntax of the Instruction ECH

Example:

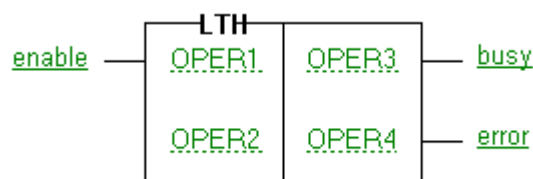


Content of the edition window on OPER4 of ECH

COM	Local PLC				Remote PLC			
1	%M0004				%A0014	..	%A0015	
2	%S0038 .. %S0041				%D0027			
3	%TD0007	028	..	030	%M0009	..	%M0014	
4	%M0006				%M0018			
5	%A0013 .. %A0020				%D0003	..	%D0004	
6	%TM0019	000	..	004	%TM0032	018	..	022

This instruction carries through writes in the programmable controller with IP address 192.168.7.5. Six communications defined, transferring data of diverse types between the CPUs. Communication 0 sends the content of a memory operand in the local CPU for two auxiliary operands in the remote CPU, being transferred 2 octets. Communications 1, 2, 3, 4 and 5 transfer, respectively, 4, 12, 2, 8 and 10 octets between the controllers.

LTH – Reading of Operands from Another PLC for Ethernet



OPER1 - IP address of the remote controller

OPER2 – not used

OPER3 – operand of control of the instruction

OPER4 – edition operand window

Description:

This instruction carries through the reading of values of operands presented in other programmable controllers for operands of the programmable controller where it is being executed, through the Ethernet communication. For its use, therefore, it is essential that the CPU that executes it is connected to other CPUs by the Ethernet.

Through the LTH values of individual operands or sets of operands can be read, being possible the programming of up to 6 different communications of reading in the same instruction.

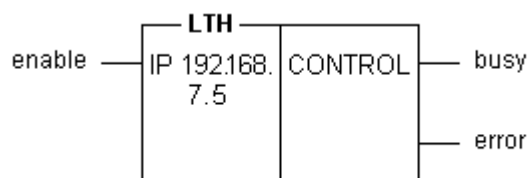
The programming of instruction LTH is identical to the ECH, observing the same restrictions. In the LTH, the transference of the values occurs from the declared operands in the remote CPU to the local CPU, being this the only difference between both.

Syntax of the Instruction:

OPER1	OPER2	OPER3	OPER4
IP Address		%D	COMUNICATIONS

Table 3-51 Syntax of the Instruction LTH

Example:



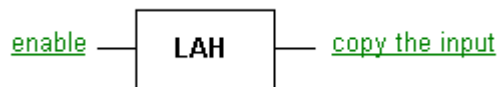
Content of the edition window on OPER4 in a LTH:

COM	Local PLC				Remote PLC			
1	%M0004				%A0014	..	%A0015	
2	%S0038 .. %S0041				%D0027			
3	%TD0007	028	..	030	%M0009	..	%M0014	
4	%M0006				%M0018			
5	%A0013 .. %A0020				%D0003	..	%D0004	
6	%TM0019	000	..	004	%TM0032	018	..	022

This instruction carries through readings in the programmable controller with the IP address 192.168.7.5. Six communications for the same one are defined, transferring data of diverse types between the CPUs. Communication 0 reads the content of two auxiliary operands in the remote CPU

for one memory operand in the local CPU, being transferred 2 octets. Communications 1, 2, 3, 4 and 5 transfer, respectively, 4, 12, 2, 8 and 10 octets between the programmable controllers.

LAH – Free Updated Images Operands for Ethernet



Description:

The instruction LAH carries through the processing of the hanging communications of the Ethernet net for the local CPU.

When returning for the processing of executive software, on each sweepings end, the CPU processes the solicitations of reading and other services that have been requested to it by other CPUs in the net, during the execution of the applicatory program.

The programmable controller have an memory area reserved for the storage of up to 32 communications received during the execution loop of the applicatory program, while executive software does not process it. If the applicatory program have relatively high time of execution and the programmable controller receives many solicitations from services of the net, can occur the situation that the CPU can not take care of it, arriving at the limit of 32 hanging communications waiting to the processing. In this case that, the CPU returns a reply to the one who requests indicating the impossibility to take care of its communication.

The LAH instruction executes the hanging processing of receptions and transmissions in the CPU, diminishing the possibility of occurrence of the previously described situation and reducing the attendance time to the solicitations. Its recommended it use in applicatory programs with high time of cycle, having to be inserted in intermediate points of the modules, dividing in stretches with approximately 20 ms of execution time.

WARNING:

The values of the operands of the applicatory program can be modified after the execution of a LAH, therefore another equipment turned on plugged to the net can request to write in the same ones. It must be considered the influence of this fact if inserting this instruction in the applicatory program.

Instructions of the Connections Group

The Instructions of the connection group allows the constructions of series and parallel ways and the inversion of the signal.




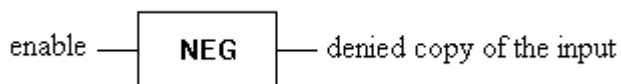
Name	Description of Name	Sequence Edition	Tool Bar
LGH	Horizontal connection	ALT, L, H	
LGN	Denied Connection	ALT, L, N	
LGV	Vertical connection	ALT, L, V	

Table 3-52 Instructions of the Connections group

LGH – Horizontal Connection



LGN – Denied Connection



LGV – Vertical Connection



Description:

The connections are auxiliary elements on the construction of the relays diagram, to connect other instructions.

The denied connection inverts the logic status of its input in its output.

5. Function Modules

This chapter contains the description of the Function modules (F) which accompany MasterTool, available for the programmable controllers in the series AL - 600, AL - 2000, QUARK and PICCOLO.

The function modules implement different routines for specific use or for access to special I/O modules for the applications program, being similar to the instructions, however loaded as program modules. Its execution is activated for other modules through the instruction CHF.

The modules which accompany MasterTool are programmed in Machine language, not being able to be read to the programmer and visualized as the modules in diagram of relays. They should be loaded directly from disk to the PLC (options **Communication, Read/Send Module**).

Each model of PLC has a group of function modules. The following list presents the function modules to the Ponto Series PLCs.

- F-PID.033
- F-RAIZN.034
- F-ARQ2.035 to F-ARQ31.042
- F-MOBT.043
- F-RELG.048
- F-PID16.056
- F-CTRL.059
- F-NORM.071
- F-COMPF.072
- F-AES.087
- F-ANDT.090
- F-ORT.091
- F-XORT.092
- F-NEGT.093

Durante a instalação do MasterTool são copiados diversos módulos com o mesmo nome, sendo armazenados em subdiretórios diferentes, conforme o tipo de UCP ao qual se destinam. Mesmo possuindo o mesmo nome, estes módulos diferem no seu conteúdo.

ATENÇÃO:

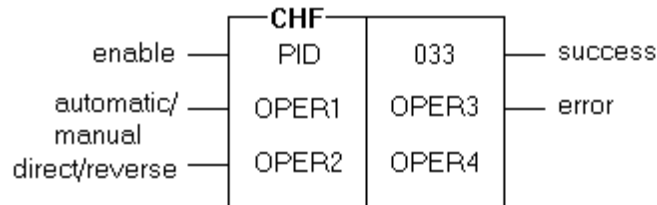
Os arquivos contidos no subdiretório de um CP não devem ser copiados para o de outro CP, sob o risco de perda de módulos. Deve-se carregar no controlador somente os módulos contidos no subdiretório correspondente à UCP utilizada.

During the installation of MasterTool different modules are copied with the same name, being stored in different subdirectories, according to the type of CPU to which they are destined. The CPU having the same name, these modules differ in their contents.

The execution time of each function module is different on each PLC, so, this information can be obtained on the manual of the respective PLC.

There is much more function modules available, that are more specific to some applications or products. The complete list of all function modules available, and how to obtain it can be found on the document F Modules List.

F-PID.033 – PID Control Function



Introduction

The function **F-PID.033** implements the proportional control algorithm, integral and derivative. Starting from a measured variable (MV) and from the required set point (SP) the function calculates the Controlled variable (CV) for the system controlled. This value is calculated periodically, taking into consideration the proportional, integral and derivative factors programmed. The function's blocks diagram is shown in figure 4-1.

The most important characteristics introduced by the control loop implemented are:

- unsaturation of the integral action (anti-reset windup)
- accompaniment of the output in manual mode and balanced manual/automatic (output tracking and bumpless transfer)
- direct or reverse action
- adjustable maximum and minimum output limits
- derivative action calculated for different samples
- capacity to carry out discrete integral
- shift with signal
- execution time of 1.6 ms in the worst case
- resolution of output of 1: 1000

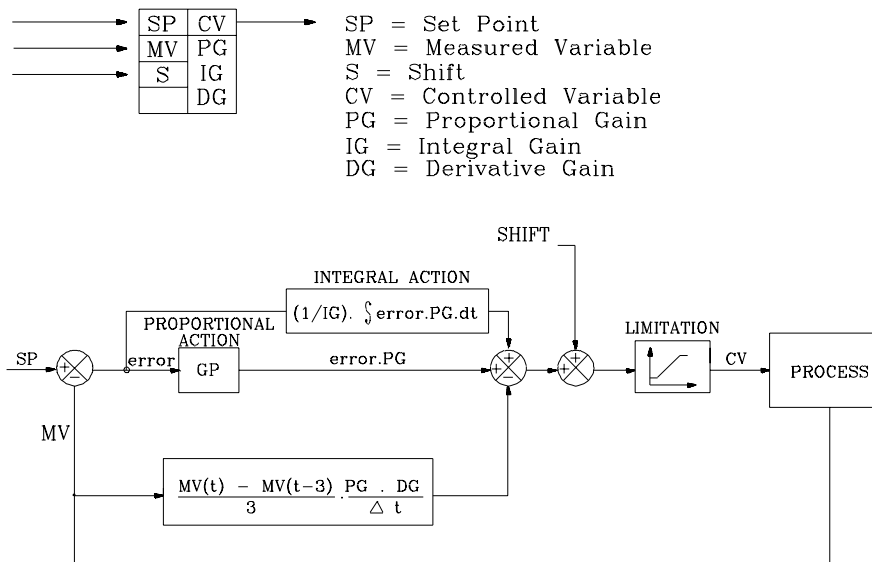


Figure 4-1 Diagram in Blocks of the Function PID

The use of the function PID in the application program allows a series of facilities which are integrated into the system, without the use of external controllers. For example:

- automatic/manual function
- inhibition of integral or derivative factor
- cascade loops
- generation of curves of set points
- modification of the control parameters by program
- modification of the control policy in function of the process's status

Programming

Operands

The cells of the CHF instruction used for the function call are programmed in the following way:

- **OPER1** - Specifies the number of parameters passed to the function in OPER3. It is compulsory for this operand to be a memory constant with value 5 (%KM+00005).
- **OPER2** - Specifies the number of parameters passed to the function in OPER4. It is compulsory for this operand to be a memory constant with value 0 (%KM+00000).
- **OPER3** - Contains the parameters passed to the function declared through a window visualized in MasterTool when the CHF instruction is edited. The number of editable parameters is specified in OPER1, being set at 5 for this module:

%TMXXXX - Table which contains the parameters used by the control algorithm. Should count 16 positions.

%MXXXX - Memory which contains the measure value of the process.

%MXXXX - Contains the set point, which is the value required for the measured variable. Its value can be modified according to the control politic required.

%MXXXX - Memory which contains the controlled variable in the process.

%AXXXX - Auxiliary octet which contains control points of the PID function.

- **OPER4** – Not used.

Inputs and Outputs

Description of the inputs:

- **enable** - when this input powered the function is called, the programmed parameters in the CHF instruction being analyzed. If the number of parameters or their type is different from the needs of the function all the outputs of the instruction are turned off. If they are correct, the control calculation PID is carried out.
- **automatic (0)/manual (1)** - when powered, the action operand does not receive the value calculated by the function (manual mode).
- **direct (0)/reverse (1)** - specifies form of action of the control.

Description of the outputs:

- **success** - is powered when then function has been correctly executed.
- **error** - is powered if an error occurs in the specification of the operands or there is an attempt to access operands not declared.

Additional Parameters

Apart from operands programmed in the CHF call instruction other parameters should be loaded into the table declared in OPER3. This table should contain 16 positions, being used to define the parameters used for the control algorithm and to store intermediate results. The table 4-1 presents the parameters which should be loaded in each table position as well as minimum and maximum values.

P o	Stored Parameter	Form.	Allowed Variation	Table Value
00	Proportional Gain x 10	GP x 10	GP: 1,0 to 100,0	10 to 1000
01	Integral factor- fraction part	dt / GI	GI: 1 to 1000 s/rep	0,0001 to 10,000
02	Integral factor- integer part		dt: 0,1 to 10 s	
03	Derivative factor – fraction part	GD / 3dt	GD: 1 to 1000 s	0,0333 to 3333,3333
04	Derivative factor – integer part		dt: 0,1 to 10 s	
05	Dislocated	DE	0 to 1000	0 to 1000
06	Minimum value of output		0 to 1000	0 to 1000
07	Maximum value of output		0 to 1000	0 to 1000
08	Not used			
09	Measured Value N – 1			0 to 1000
10	Measured Value N – 2			0 to 1000
11	Measured Value N – 3			0 to 1000
12	Error			0 to 1000
13	Proportional action x 10			0 to 65535
14	Integral action – fractional part x 10			0 to 65535
15	Integral action – integer part x 10			0 to 65535

Table 4-1 Additional Parameters of the PID

To make possible a greater execution speed, some parameters should be loaded in the table already pre-calculated. Being values relatively fixed, in this way avoiding recalculation for each function call.

The parameters which should be pre-calculated are:

- Proportional gain x 10 (position 0) - Is calculated by multiplying the proportional gain required for 10.
- Integral multiplicative factor - Is calculated by dividing the sample interval (dt) by the whole gain required. The unit of dt is seconds, its minimum value being 0.1 seconds and maximum 10.0 seconds and should be equal to the interval in which the routine is executed. The G1 is

seconds/repetition, able to vary from 1 to 1000 seconds/repetition G1 equals to 1 second/repetition signifies the maximum integral effect.

- Multiplicative derivative factor (positions 3 and 4) - Is calculated by dividing the derivative gain (DG) by the sample interval (dt) and by value 3. The unit of DG is seconds, being possible to vary from 1 to 1000 seconds. DG equal to 1000 seconds signifies maximum derivative effect. It is recommended that the greater the value of the DG, the greater should be the sample interval. The same for the DG values = 1 second, the sample interval should be more than 0.2 seconds. If such care is not taken, the derivative term only produces “noise” and the control action will be very abrupt.
- Dislocating (position 5) - Allows the introduction of a shift (“bias”) in the controlled value, avoiding negative errors causing saturation in the minimum value of the output. Generally this value is set to 50% (500) or equal to the set point, if the proportional gain is small.
- The minimum and maximum output values (positions 6 and 7) - They are optional values which limit the excursion of the controlled value, being able to be modified dynamically in the function of the operational conditions. If the maximum value is more than or equal to 1000 and the minimum value equals 0, no limitation is carried out.

The measure value, the controlled value, the dislocating, the maximum and minimum values have as variation the band from 0 to 1000, which corresponds to a variation of 0 to 100% in the variables of the process.

The remaining positions of the table are used exclusively by the function PID, not being able to be modified by the applications program. Position 12 (error) can be consulted by the program. Positions 14 and 15 accumulate the whole factor, being able to be zeroed, if necessary. It is recommended that these positions are zeroed at the beginning of the processing to avoid random value becoming stored.

Apart from the table of parameters, same control points are used by the function, contained in the auxiliary octet specified (%AXXXX).

- %AXXXX.4 - Signal for whole action - Is used by the function PID. When turned off, the integral term is positive, if the opposite it is negative. It can be read by the program, if required.
- %AXXXX.5 - Signal for dislocating. Indicates to the function what the signal for dislocating is, having to be enabled by the program. The point turned off indicates positive shift. When powered, the shift is negative.
- %AXXXX.6 - Inhibits derivative action - When powered the function does not execute the derivative action.
- %AXXXX.7 - Inhibits whole action. When powered, the whole action is not calculated, remaining attributed as the last value calculated before the inhibition, unless the value limit are exceeded.

Characteristics of Functioning

The unsaturation of the whole action (anti-reset windup) is done in a mode to avoid the integral term continuing to accumulate error when trouble in the process causes the saturation of the output of the controller in some limits. At the moment when the output value reaches any of the limits (maximum or minimum), the integral term is set at its current value, blocking its undefined increase, without influencing the output.

This ensures that it will have an answer from the controller so the trouble which saturated the output disappears. The function can be executed in manual mode, by powering the second input of the CHF instruction. In this mode, the routine does not modify the action output value any more, but accompanies it. That is, in function of the value of the fixed output and of the measure value of the process, the proportional and derivative term are calculated and the integral term is forced to an adequate value, in a way that, when a transition occurs from manual to automatic, the routine reassumes control with the initial value of the output equal to the last value of the output in manual mode. This act of communication from manual/automatic is called balanced (bumpless transfer).

The form of control can be direct or reverse. This selection is carried out by turning off or powering the third input of the CHF instruction. If the process is such that the measure value grows when the

value of the output of the action grows, the direct action should be selected. If the measure value decreases with the increase of the output of the action, then the reverse action should be used.

The interval between samples of a PID loop can vary from 0.1 to 10.0 seconds. It is the responsibility of the user to program a trigger of the function, that is to say, a passage of applications program that only enables the PID routine in the time intervals required. Note that the value of the sample interval used for the calculation of the multiplicative factors integral and derivative should coincide with the time interval of the calls of the trigger. As each routine execution can last up to 3 ms, it is advisable that each different control loop is fired in different scans of the program.

Example of Application

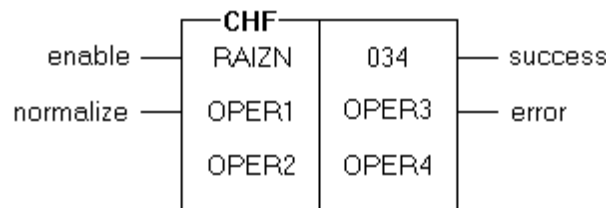
As an example of use, the following adjustment values are required for a control loop:

PA	=	62
GP	=	5 (GP = 100 / proportional band in %)
GI	=	100 seconds/repetition
GD	=	5 seconds
dt	=	1 second
DES	=	50%
MAX	=	80%
MIN	=	0%

The values which should be loaded in the table of parameters are:

Position	Value	
0	50	GP X 10 (50)
1	100	dt / GI (0,0100)
2	0	
3	6666	GD / 3dt (1,6666)
4	1	
5	500	DES
6	0	MIN
7	800	MAX
8	620	PA

F-RAIZN.034 – Square Root Function



Introduction

The function **F-RAIZN.034** extracts the square root of a value supplied and normalizes the result to a previously defined scale, if required.

The calculation carried out corresponds to the following expression:

Op Destination = Square Root (Op Source) *Normalization Constant/256

The Normalization executed together with the processing of the square root ensures very precise results, since internal variables with greater storage capacity than memory operands are used.

This function is typically used in the linearization of the readings from translators which supply values in quadratic scale, that is to say, with the output proportional to the root of the signal measure.

Programming

Operands

The cells of the CHF instruction used to call the function are programmed in the following way:

- **OPER1** - Specifies the number of parameters passed to the function in OPER3. It is compulsory for this operand to be a memory constant with value 3 (%KM+00003).
- **OPER2** - Should be an operand of type memory constant with value 0 (%KM+00000). It determines the number of parameters possible to be programmed in the editing window of OPER4. As this function does not need any parameter in OPER4, the value of OPER2 is 0.
- **OPER3** - Contains the parameters which are passed to the function, declared through a window visualized in MasterTool when the CHF instruction is edited. The number of editable parameters is specified in OPER1, being set at 3 for this module:
 - **%MXXXX** - Operand with the value to be extracted to the square root (source). This value should be positive so that the calculation can be carried out.
 - **%MXXXX or %KM+XXXX** - Memory or constant operand for the Normalization of button of scale of the square root extracted. The value programmed is divided by 256 and multiplied by the root of the operand supplied, giving the value of the destination operand, when the instructions second input is powered.
 - **%MXXXX or %FXXXX** - Operand which receives the result of the normalized square root (destination). Should be necessarily of the same type that the source operand.
- **OPER4** – Not used.

Inputs and Outputs

Description of the inputs:

- **enable** - when this input is powered the function is called, the parameters programmed in the CHF instruction being analyzed. If they are incorrect, all the outputs of the instruction are turned off. If they are correct, the calculations are carried out, the outputs success or error being enabled.

- **normalize** - when powered, carries out the adjustment of the button of scale to value of the square root obtained. If turned off, the value of the memory operand destination simply receives the square root of the source operand.

Description of the outputs:

- **success** - indicates that the calculation of the root and its Normalization has been carried out correctly. When turned off, indicates that the input enabled is not enabled, the module is not loaded into the PLC, the operands were not correctly defined or negative values are stored in them.
- **error** - this output is always powered when one of the following errors occurs:
 - negative values exist in the supply operand or in the Normalization constant
 - error in the specification of the operands or attempt to access the operands not declared.
 - operand source with a different destination operand

WARNING:

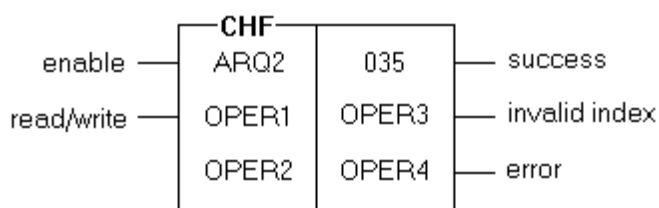
In the version 1.00 of **F-RAIZN.034** the output error is not enabled in the attempt to access the operands not declared.

Example of Application

To normalize the value of the destination operand in a form that has the same scale as the operand supplied, the value to be declared in the Normalization operand should be equal to the square root of the operand supplied multiplied by 256.

For example, there may be the case of a transducer which supplies values from 0 to 1024, proportional to the root of an outflow, and it may be required that these values are linearized to the same scale of values (0 to 1024). The Normalization constant programmed is 8192 (square root (1024) *256).

F-ARQ2.035 to F-ARQ31.042 – Functions Data File



Introduction

The function data file allow the use of the applications program memory to store large quantities of information, using concepts of registers and fields. In this way it obtains great flexibility in the utilization of the PLC's memory banks, apart from a substantial increase in the data storage capacity.

There are different function which implement data files, being identical in the programming mode and functioning, differing only in the storage capacity. The modules available are:

- F-ARQ2.035 - File function with 2 Kbytes of data
- F-ARQ4.036 - File function with 4 Kbytes of data
- F-ARQ8.037 - File function with 8 Kbytes of data
- F-ARQ12.038 - File function with 12 Kbytes of data
- F-ARQ15.039 - File function with 15 Kbytes of data
- F-ARQ16.040 - File function with 16 Kbytes of data
- F-ARQ24.041 - File function with 24 Kbytes of data
- F-ARQ31.042 - File function with 31 Kbytes of data

Each file can have up to 255 registers, numbered from 0 to 254, being that each register can have up to 255 fields, also numbered from 0 to 254. Note, however, that the total quantity of memory occupied cannot exceed the modules capacity.

Each field occupies the same number of bytes of the operand where the files readings or writings are carried out.

Programming

Operands

The cells of the CHF instruction used to call the function are programmed in the following way:

- **OPER1** - Specifies the number of parameters passed to the function in OPER3. It is compulsory for this operand to be a memory constant with value 5 (%KM+00005).
- **OPER2** - Specifies the number of parameters passed to the function in OPER4. It is compulsory for this operand to be a memory constant with value 0 (%KM+0000).
- **OPER3** - Contains the parameters passed to the function, declared through a window visualized in MasterTool when the CHF instruction is edited. The number of editable parameters is specified in OPER1, being set at 5 for this module:

%MXXXX, %DXXXX, %AXXXX, %EXXXX, %SXXXX, %TMXXXX, %TDXXXX, %KM+XXXXX or %KD+XXXXXXX - Operand from where the data is read in the writing operations in the file or to where the data is copied into readings of the file (parameter 1).

%MXXXX - Number of register from/to which the data will be copied (parameter 2). Should contain between 0 and the total number of registers less 1.

%MXXXX - Number of field from/to which the data will be copied (parameter 3). Should contain between 0 and the total number of fields less 1.

%KM+XXXXX - Total number of registers (1 to 255) required for the file (parameter 4).

%KM+XXXXX - Total number of fields (1 to 255) required for the file (parameter 5).

- **OPER4** – Not used.

Inputs and Outputs

Description of the inputs:

- **enable** - when this input is powered the function is called, the parameters programmed in the CHF instruction being analyzed. If the number of parameters or their type are different from the needs of the module, the error output is powered. If they are correct a attempt to access the file is carried out.
- **read/write** - when powered, the value of the first parameter is copied to the register and the field specified in the second and third parameters. If it is turned off, the value is read from the field and copied to the first parameter.

Description of the outputs:

- **success** - indicates that the access to the data file was correctly carried out.
- **invalid index** - this output is connected:
 - the field to be read or written was not specified
 - the declaration of the registers and fields exceeds the modules memory capacity
 - there is an attempt to read when the first parameter is a constant
 - there is an attempt to write the module being stored in EPROM memory
- **error** - is powered if an error occurs in the specification of the parameters or attempt to access the operands not declared.

Description of Functioning

For correct declaration of the number of fields and registers of the file, the following calculation should be carried out:

Occupation of the file = Num. registers X Num. fields X Num. bytes per field (parameter 4)
(parameter 5)

The number of bytes per field occupied for each type of operand can be obtained from table 4-2.

Parameter 1	Number of bytes per field
%MXXXX	2
%DXXXX	4
%AXXXX	1
%EXXXX	1
%SXXXX	1
%TMXXXX	2
%TDXXXX	4
%KM+XXXXX	2
%KD+XXXXXXXX	4

Table 4-2 Occupation of the Field of the Files

The value obtained in the previous calculation should be less than or equal to the total capacity of the function used, according to table 4-3.

Function	Capacity (bytes)
F-ARQ2.035	2048
F-ARQ4.036	4096
F-ARQ8.037	8192
F-ARQ12.038	12288
F-ARQ15.039	15360
F-ARQ16.040	16384
F-ARQ24.041	24576
F-ARQ31.042	31744

Table 4-3 Capacity of the Functions Data Files

WARNING:

Different CHF instruction for access to the same file can be declared in the same program. In all these instructions the operands with the values to be written or that receive (parameter 1 in OPER 3) the same number of bytes per field.

Therefore, it is possible to indistinctly read or write operands %E, %S and %A of one file or %KM, %M and %TM of another. Never the less they should not be accessed with operands %M or %D in the same file.

If the first parameter is a table (%TM or %TD), all the fields of the register indicated in the second parameter are copied, that is to say, the transfer of data is carried out between the register and the table, being that the value of the third parameter (number of field is ignored).

If the table has fewer positions than the number of fields in the register, only the fields which correspond to the existing positions are transferred. If the table has more positions than the number of fields in the register, only the existing fields are transferred.

The operation of writing the data copies it to the appropriate area of memory occupied by the function module.

WARNING:

If the module **F-ARQ** is stored in EPROM Flash, it is not possible to write data in the file, only to read data. To carry out the writing of the data into the files, the F modules corresponding to the them should be in the RAM memory of the applications program.

WARNING:

During the reading of a PLC's module data file with MasterTool or during its transferring from RAM to Flash, no writing of data should be carried out.

This is because the writing of data modifies the module read, being considered invalid by the programmer or by the PLC due to the altering of its checksum.

The functions data files are modules of the applications programs being able to be loaded or read by the PLC and stored on disks. For example, there may be the case of a PLC controlling a injector machine, storing different configuration parameters in an **F-ARQ8.037** module. After the parameters are stored, this module F can be read and stored on disk, to load in other equal injected machines.

Example of Application

As an example if it a file with 120 registers and with 8 fields for register to store operands %D, the occupation of memory will be:

Occupation of the file = 120 registers X 8 fields/register X 4 bytes/field

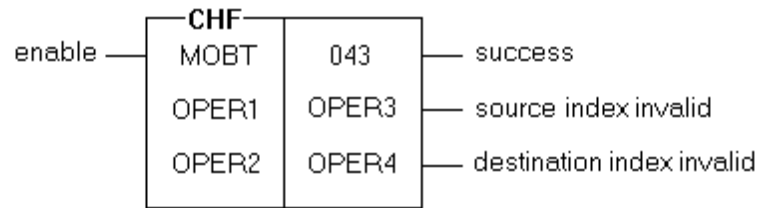
Occupation of the file = 3840

The configuration requires 3840 bytes to be occupied, the module **F-ARQ4.036** having to be used, then it allows the storing of 4096 bytes.

The parameters programmed in OPER3 of the CHF instruction for the access to the file are:

- %D0020 - operand to where it will be read or with the value to be written in the file
- %M0100 - contains the number of the register to be read or written, having to have between 0 and 119 (120 register in total).
- %M0101 - contains the number of the field to be read or written, having to have between 0 and 7 (8 register in total).
- %KM+00120 - total number of registers.
- %KM+00008 - total number of fields.

F-MOBT.043 – Function for Moving Blocks from Table Operands



Introduction

The function **F-MOBT.043** carries out the copy of blocks of numeric operands (%M or %D) or positions of tables (%TM or %TD) up to 255 values of simple operands can be copied to tables and vice versa, also transferring the positions from one table to another. It is possible to specify the initial position of the block to be copied into the table supplied and into the destination table.

Programming

Operands

The cells of the CHF instruction used to call the function are programmed in the following way:

- **OPER1** - Specifies the number of parameters passed to the function in OPER3. It is compulsory for this operand to be a memory constant with value 5 (%KM+00005).
- **OPER2** - Should be an operand of type memory constant with value 0 (%KM+00000). It determines the number of parameters possible to be programmed in the editing window of OPER4. As this function does not need any parameter in OPER4, the value of OPER2 is 0.
- **OPER3** - Contains the parameters passed to the function, declared through a window visualized in MasterTool when the CHF instruction is edited. The number of editable parameter is specified in OPER1, being set at 5 for this module:
 - **%MXXXX, %DXXXX, %TMXXXX or %TDXXXX** - Initial operand from where the values are copied (source operand).
 - **%KMXXXX** - Initial position to be transferred from the source operand is a simple operand (%M or %D).
 - **%MXXXX, %DXXXX, %TMXXXX or %TDXXXX** - Initial operand where the values are copied to (destination operand).
 - **%KMXXXX** - Initial position where the values in the destination table are copied to. This parameter is disregarded if the destination operand (%M or %D).
 - **%KMXXXX** - Number of simple operands or table positions to be transferred starting from the operand or from the initial position in the parameters previously declared. It should be less than or equal to 255.
- **OPER4** – Not used.

Inputs and Outputs

Description of the Inputs:

- **enable** - when this input is powered the function is called, the parameters programmed in the CHF instruction being analyzed. If these are incorrect, the outputs of the invalid index are enabled.

Description of the outputs:

- **success** - indicates that the moving was correctly carried out
- **source index invalid** - indicates that there was an error in the specification of the supply operand:
 - the operand is not declared in module C
 - the type of parameter 2 is not %KM
 - the initial position programmed does not exist, if the source operand is table
 - there are not enough operands or table positions to carry out the movement
- **destination index invalid** - indicates that there was an error in the specification of the destination operand:
 - the operand is not declared in module C
 - the type of parameter 4 is not %KM
 - the initial position programmed does not exist, if the destination operand is table
 - there are not enough operand or table positions to carry out the movement

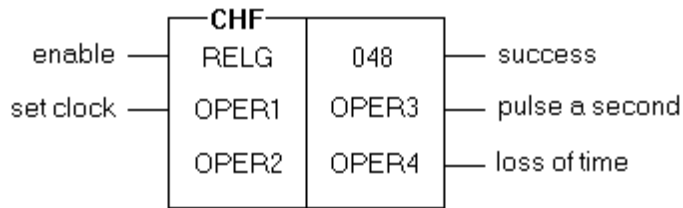
If the two outputs of the invalid index are enabled simultaneously, some of the following errors occur:

- the number of parameters programmed in OPER1 is different from five.
- the type of parameter 5 is not %KM
- the total number of position to be transferred is greater than 255

WARNING:

This function allows the moving of a large number of operands in one scan. It should be used with care so that the maximum time of the program cycle is not exceeded.

F-RELG.048 – Function to Access the Real Time Clock



Introduction

The function **F-RELG.048** carries out the access of the real time clock contained in the CPU. The clock has complete hour and calendar, allowing the development of applications programs which depend on precise time bases. The time information is kept the same when there is power failure in the system, since the CPU is powered by batteries.

This function has similar characteristics to function **F-SINC.049**, since both execute accesses to the same clock, differing only in the methods of setting. They can be used simultaneously in the same program, if necessary.

Programming

Operands

The cells of the CHF instruction used to call the function are programmed in the following way:

- **OPER1** - Specifies the number of parameters passed to the function in OPER3. This operand must be a memory constant with value 2 (%KM+00002).
- **OPER2** - Should be an operand of type memory constant with value 0 (%KM+00000). Determines the number of parameters possible to be programmed in the editing window of OPER 4. As this function does not need any parameter in OPER4, the value of OPER2 is 0.
- **OPER3** - Contains the parameter passed to the function, declared through a window visualized in MasterTool when the CHF instruction is edited. The number of editable parameters is specified in OPER1, being set at 2 for this module:
 - **%MXXXX or %TMXXXX** - Specification of the operands to where the clock values are read. If this parameter is specified as memory, the values are read to the memory declared and the following six. If it is specified as table, the values are placed starting from position 0 to 6. If the operands are not declared, the reading of the time values is not carried out and the instruction outputs are disconnected. It is possible to use tables with more than 7 positions, being that values are read from operands in the following sequence:

Operand	Position Table	Content	Format
%MXXXX	0	Seconds	000XX
%MXXXX + 1	1	Minutes	000XX
%MXXXX + 2	2	Hours	000XX
%MXXXX + 3	3	Days of Month	000XX
%MXXXX + 4	4	Month	000XX
%MXXXX + 5	5	Year	000XX
%MXXXX + 6	6	Days of the Week	000XX

Table 4-4 Values Read from the Clock (F-RELG.048)

The contents of these operands can be read at any time, but are updated with the real hour of the clock only when the instruction is executed. The 24 hour format is used in the time count. The days of the week are counted with values from 1 to 7.

Value	Days of the Week
1	Sunday
2	Monday
3	Tuesday
4	Wednesday
5	Thursday
6	Friday
7	Saturday

Table 4-5 Values of the Days of the Week (F-REL.G.048)

- **%MXXXX or %TMXXXX** - Specification of the operands from where the clock values are set, with the enabling of some of the inputs to set the function. If this parameter is specified as memory, the values are copied from the memory declared and the following 6. If it is specified as table, the values are copied from position 0 to 6. If the operands are not declared, the setting is not carried out and the outputs of the instructions are disconnected. The values to be copied to clock should be placed in the operands in the same sequence as the operands of reading (seconds, minutes, hours, day of the month, year and day of the week).
- **OPER4** – Not used.

Inputs and Outputs

Description of the inputs:

- **enable** - when this input is powered the function is called, the parameters programmed in the CHF instruction are analyzed. If they are incorrect, all the output of the instruction are turned off. If they are correct, the time values of the clock are transferred to the memory operands or to a table declared as first parameter in OPER3, the output success is powered and the output pulse a second is connected by a scan at each second.
- **set clock** - when powered, the values of the operands declared as second parameter in OPER3 are set in the clock, if the values are correct. While the input is enabled the time is not counted, the output pulse a second remaining turned off.

Example:

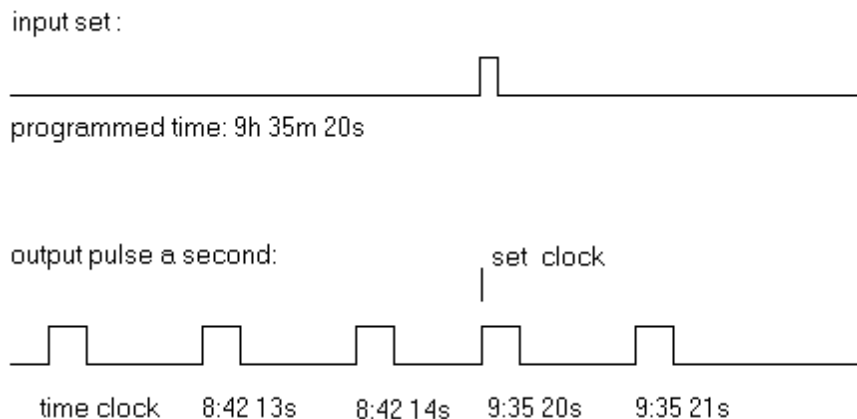
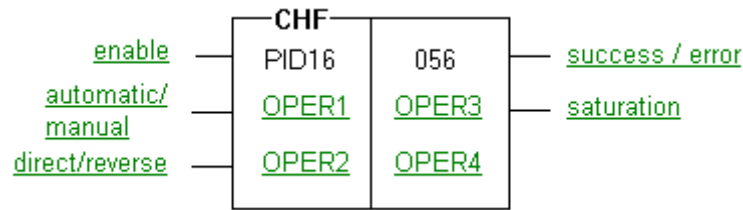


Figure 4-2 Example of Diagram of Setting Input Times

Description of the outputs:

- **success** - is powered when the function has been correctly executed.
- **pulse a second** - indicates if there was a change in the clock. The pulse lasts one scan and can be used to synchronize events of the application program.
- **loss of time** - this output is connected if the clock was left without battery power during a failure of the main supply. It is deactivated with the setting of the clock.

F-PID16.056 – F Module for PID Control



Introduction

The F-PID16.056 function, available to AL-2003, AL-2004, PO3145, PO3142 and PO3242 CPUs, implements the proportional, integral and derivate control algorithm. From a measured value (VM) and the point of desired adjustment (PA) the function calculates the performance value (VA) for the controlled process. This value is periodically calculated, considering the proportional, integral and derivative programmed factors. It is a PID control type ISA algorithm where the proportional gain is the gain of the controller, applied in the error of integral and derivative parcels of the controller.

The F module can be represented by the diagram below:

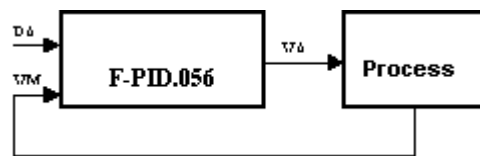


Figure 4-10 Diagram of blocks of F-PID16.056

Details of the PID controller diagram is presented below:

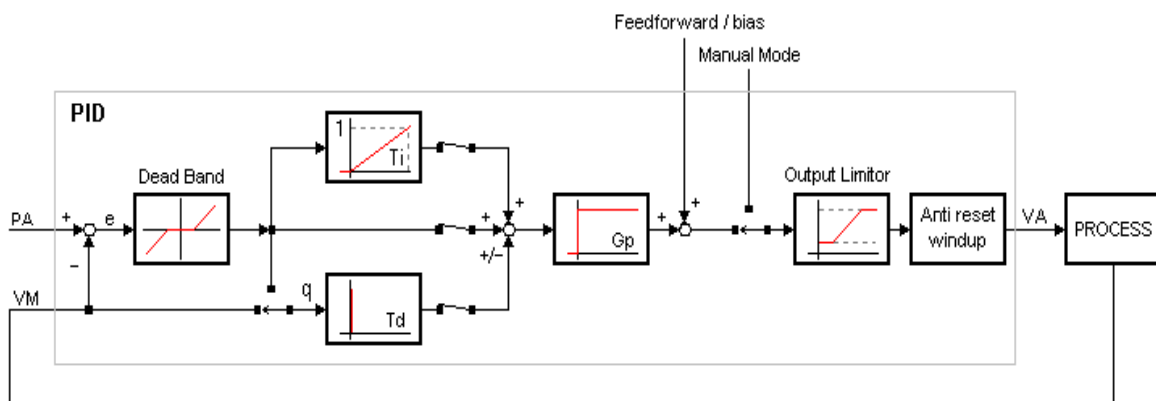


Figure 4-11 Diagram of blocks detailed of F-PID16.056

The basic control algorithm used on PID controller is described on the following equation:

$$VA = Gp \cdot (e + \frac{1}{Ti} \cdot \int e \cdot dt \pm Td \cdot \frac{\partial q}{\partial t}) + Bias$$

Where:

VA is the performance value;

Gp is the proportional gain of the controller;

e is the system(PA-VM) error;

Ti is the integral time constant (s/rep);

Td is the derivative constant time in (s);

dt is the sampling period;

q represents the system error (+) or the measure variable (-), as the selection;

PA is the adjustment time;

VM is the variable measured on the process that is being controlled;

Bias is a displacement inserted through an additional point after the calculation of the algorithm.

Bias is a displacement inserted through an additional point after the calculation of the algorithm.

Operation in 4 quadrants (positive and negative values in inputs and outputs);

It can be used in cascade mode, implementing complex algorithms of control;

16 bits operation;

Use of parameters (Gp, Ti, Td) directly on ISA format;

Derivative term selection acting on error function (positive) or on the measured variable (negative);

Individual inhibition of the derivative, integral or proportional terms;

Derivative action calculated on three samplings (filter);

Direct or reverse action;

Adjustable output limits;

Unsaturation of the integral action (anti reset windup);

Limitation of the growth tax;

Feedforward / bias;

Manual or automatic mode;

Output tracking for soft transition (bumpless) from the manual mode to the automatic mode;

Configurable dead zone applied on error.

Programming

Operands

The cells of CHF instruction used for the call of the function are programmed in the following way:

OPER1 - - Specifies the number of parameters that are passed to the function on OPER3. This operand must be obligatorily a memory constant with value 6 (%KM+00006).

OPER2 - - Specifies the number of parameters that are passed to the function on OPER4. This operand must be obligatorily a memory constant with value 0 (%KM+00000).

OPER3 - - It contains the parameters that are passed to the function, declared when the CHF instruction is edited. The number of editable parameters is specified on OPER1, and it is equal to 6 for this module:

- **%TMXXXX** - Table that contains the parameters used on control algorithm. Must contain at least 30 positions.
- **%MXXXX** - Memory that contains the measured value of the process (VM), normally gotten through an analogical input.
- **%MXXXX** - It contains the adjustment point (PA), that is the desired value to the measured variable (set point).
- **%MXXXX** - Memory that contains the performance value (VA) generated for the control algorithm. In manual mode the control algorithm does not act on this variable, that can be manipulated by the user.
- **%MXXXX** - Memory to *feedforward/bias*. The value of this operand will be added to the output value of the PID controller, before the limitation (limits upper and lower declared on parameters table).
- **%AXXXX** - Auxiliary Octet that contains control points of the PID function.

OPER4 – Not used.

Inputs and Outputs

Description of the Inputs:

enable - when this input is powered the function is called, being analyzed the parameters programmed on CHF instruction. If the number of parameters or its type are different of the function needs, the output success/error will be unpowered. If it is correct, the PID control calculus is realized.

automatic(0)/manual(1) - when powered, the performance variable is not modified by the function, being able to be modified manually (manual mode).

reverse (0)/direct (1) - it specifies the way as the controller will act on the process.

WARNING:

The direct and reverse concept on F-PID.033 module are the opposite of F-PID16.056.

Description of the Outputs:

success(1)/error(0) - it is powered when the function is correctly executed. Always that occurs errors in the specification of the operands, invalid attempt in accessing the operands not declared or invalid parameters, this output is not powered, indicating error.

saturation - when powered indicates that the output of the controller reached the saturation, or on max limit, or on min limit.

Functioning Characteristics

Unsaturation of the Integral Action

The unsaturation of the integral action (anti-reset windup) is made in order to prevent that the integral term continues to accumulating errors when a riot in the process cause the saturation of the output of the controller in one of the limits. In the moment that the value of the output reaches one of the limits (maximum or minimum), the integral term is fixed in its current value, hindering its indefinite growth. This assures that it will have a reply of the controller so soon disappears the riot that took it to saturate the output.

Manual Mode

The function can be executed in manual mode, powering the second input of CHF instruction. On this mode, the routine does not modify the value of the performance output, but follows it (output tracking). Or either, in function of the output value on the manual mode and of the measured value of the process, the terms proportional and derivative are calculated and the integral term is forced for an adequate value that, when there is the manual transition for automatic, the routine can reassume the control with the initial value of the output equal to the last value of the output in the manual mode. This behavior is called balanced manual/automatic commutation (bumpless transfer). It is important to observe that this resource only success when the integral action is enabled. Although in manual mode, the performance output cannot assume bigger values than the limits declared in the table of parameters. When this occur the value of the performance output will be forced to the closer limit.

Direct and Reverse Control

The control can be direct or reverse. This selection is realized unpowering or powering the third input of the CHF instruction.

On direct control, in case that an increase in the measured value occurs (VM), the controller must increase the performance output (VA) in order to control the process.

On reverse control, in case that an increase in the measured value occurs (VM), the controller must diminish the performance output (VA) in order to control the process.

Considering two examples that use the same valve, controlled for an analogical output 4-20 mA (VA varies from 0 to 4095). Assumes that with VA = 0 (4 mA) the valve is total closed, and with VA = 4095 (20 mA) the valve total is opened.

In the first example, it is desired to control the level of a tank (VM = level of the tank), through a valve of exhaustion of the tank. Therefore, the more the valve is open, faster the level of the tank diminishes. In this in case, if the tank level increases (VM), the controller must increase VA to open the valve. Therefore, it is a direct control.

On second example, it is desired to control the outflow through the valve. Therefore, the more the valve open, greater will be the outflow. In this case, if the outflow increases (VM) through the valve, the controller must diminish VA to close the valve. Therefore, it is a reverse control.

Sampling Interval

The interval between samplings of a PID loop can vary from 0,01 to 10 seconds. It is responsibility of the user to program a "trigger" to the function, or either, a stretch of applicatory program that only enables F-PID16.056 routine in the desired intervals of time. It is advised to use a E018 module, therefore this module is executed inside of an fixed time interval time that can be used to generate one or more bases of time for the execution of one or more PID loops. Notice that the value of the sampling interval (dt) declared in the table of parameters of the controller must coincide with the time interval of the "trigger" calls.

Execution Time

The worse case of execution of a control loop with the F-PID16.056 reaches the time of 360 us. This time is valid for the AI-2003, AI-2004, PO3145, PO3142 and PO3242 CPUs.

Table Position Parameters Description

The table with the parameters of the controller is also used to store variables of internal use, the total sum of the integral action and the measured variable or errors of previous cycles for the derivative action calculation. Each position of this table is described below:

Table Position	Parameters	Description
00	GP (x10)	Proportional gain (without unit). The possible values are inside of the interval from 0,1 to 3000. The proportional gain must be multiplied by 10 to be declared in this field, assuming the interval from 1 to 30000. Values out of this band will make that F module enters on error mode and does not execute the control algorithm.
01	Ti (x10)	Constant of integral time (s/repetition). The possible values are inside of the interval from 0,1 to 3200 s/rep. The constant of integral time must be multiplied by 10 to be declared in this field, assuming the interval from 1 to 32000. It is observed here that lesser the value of Ti, greater will be the integral action. Values out of this band will make that F module enters in error mode and does not execute the control algorithm.
02	Td (x100)	Derivative constant of time (s). The possible values are inside of the interval from 0 to 320 s in units of 0,01s. The constant of time derivative must be multiplied by 100 to be declared in this field, assuming the interval from 0 to 32000. Values out of this band will make that F module enters in error mode and do not execute the control algorithm. If attributed value zero to the derivative time constant, the action will be calculated with value zero, not influencing in the performance output. One sends regards to disable the derivative action when it is not used.
03	dt (x100)	Sampling interval of the process that is being controlled (s). The possible values are inside of the interval from 0,01 to 10 s. The sampling interval must be multiplied by a factor of 100 to be declared in this field, assuming the interval from 1 to 1000. Values out of this band will make that F module enters in error mode and does not execute the control algorithm. It is responsibility of the user to enable the F module in this time interval.
04	Maximun output value	Maximum value of allowed output. It can assume values from -30000 to +30000. Must necessarily be bigger than the minimum output value. Values out of this band will make that the F module enters in error mode and does not execute the control algorithm.
05	Minimun output value	Minimum value of allowed output. It can assume values from -30000 to +30000, Must necessarily be minor than the maximum output value. Values out of this band will make that F module enters in error mode and does not execute the control algorithm.
06	Dead Zone	Dead Zone. It can assume values from 0 to +30000. Always if the absolute value of the error is minor than the value defined in this field, the controller will be executed considering the error as zero. To disable this resource it is enough to declare the value zero for the dead zone. Values out of this band will make that F module enters in error mode and does not execute the control algorithm.
07	Maximun allowed variation.	The value declared in this field indicates the absolute value of the maximum variation that the output of the controller can have to each sampling interval (dt). It can assume values from 1 to 30000. Value 1 represents a very small variation while that value 30000 represents a great variation to each cycle of sampling. Declaring zero in this field the maximum allowed variation is not verified, allowing any variation. Values out of this band will make with that F module enters in error mode and does not execute the control algorithm.
08	Accumulated AI x100 (Hi)	The value presented in this field is the accumulated integral action. Aiming to get more numerical resolution, the accumulated integral action is stored multiplied for a factor of 100. Three operands of 16 bits are used to keep the high part, low part and factionary part of the integral action. These fields must be started with zero to prevent that some random value is stored.
09	Accumulated AI x100 (Lo)	
10	Accumulated AI x100 (frac)	
11	Value of previous performance (VA)	This field is restricted to F module use and it must not have its content modified. The value of this field is the variable of performance of the previous cycle, used to limit the maximum variation.
12-13	Reserved	Reserved Operands.

14	VM(t-3) or error(t-3)	This field is restricted to F module use and it must not have its content modified. Description of the three last errors or measured variables, used for calculation of the derivative term. The derivative action can act in function of the error as in the measured variable, however never the selection must be changed from error to measured variable or vice versa during the control process.
15	VM(t-2) or error(t-2)	
16	VM(t-1) or error(t-1)	
17-27	Internal use	These positions of the table are used exclusively by PID function, can not be modified by the applicatory program.
28-29	Reserved	Reserved Operands

Table 6: Description of the positions of the parameters table of the F-PID16.056

Always if occur some alteration in the parameters GP, Ti, Td or dt, the F-PID16.056 module needs a cycle execution to adapt the controller to the new parameters, not executing the control in this cycle and keeping the variable of performance (VA) unchanged.

Description of %A Operand Control

The auxiliary operand of F module control is used in agreement with the table below.

Bit	Description
0	inhibits(1) / enable(0) integral action
1	inhibits(1) / enable(0) derivative action
2	inhibits(1) / enable(0) proportional action
3	Derivative action in error function (1) or in process variable (0)
4	Reserved
5	Internal use
6	Internal use
7	Internal use

Table 7: Description of the auxiliary operand control

Through the auxiliary control operand of the F-PID16.056 function is possible to incapacitate the proportional action, integral and/or derivative and also to select the derivative action acting in function of the error or the measured variable in the process. When some action of control (either it proportional, integral or derivative) will not be used, this must be incapacitated enabling the corresponding bit.

Incapacitating the action of proportional control it can not generate action to the error, but the gain of the system continues being applied on the integral and derivative actions. For a pure integrator, for example, only the integral action must be enabled, adjust the desired time constant Ti and attribute to constant GP (proportional gain) a unitary gain.

The derivative action acting in function of the measured variable is recommended for the majority of the applications, therefore it prevents great variations on VA output when the PA adjustment point is modified. For special applications exists the possibility of the derivative action selection in function of the error of the system.

The bits of internal use are of F-PID16.056 function exclusive use and they must not have its content modified.

Application Notes

Time Sampling Selection

The efficiency of the digital controller is directly related with the used sampling interval. The more that this interval diminishes, the result of the digital controller come close to the result of an analogical controller. It is advised to use a time of sampling of order of one tenth of the time constant of the system, or either

$$T_A = \frac{T}{10}$$

where T_A is the used sampling time and T is the time constant of the system.

Example: The time constant of a first-class system can be gotten from its reply graph of the performance variable (VA) to a step in the PA point of adjustment with open control loop (PID disabled or in manual mode), as the figure below:

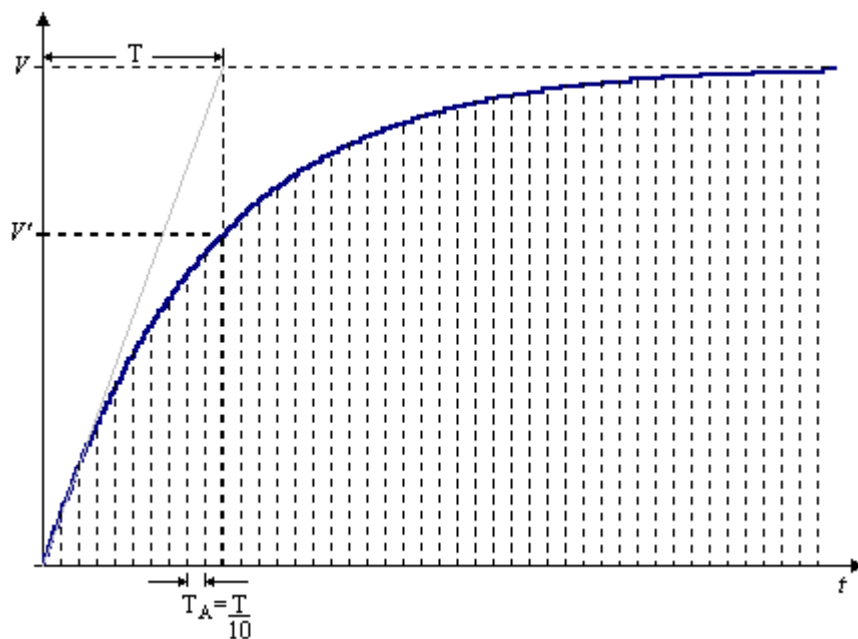


Figure 33: Time constant of the system and sampling interval

This figure demonstrates the attainment of the time constant of the system in two distinct ways. Most usual is to take as time constant of the system the time necessary to the system to reach 63.212% of the final value. Another way is to trace the first derivate of the reply to step curve, the time constant is that straight line that crosses the final value of the system reply.

Defined the time constant, is enough to define the sampling interval of order of one tenth of this value.

It is important to remember that the update of the inputs and outputs occurs in the same order of a CPU time cycle. Always that the time of cycle of the CPU is greater than the sampling time advises to use the AES instruction for the AI-2003 and AI-2004 CPUs or F-AES for the Ponto series.

Feedforward/Bias

Through operating the memory used for feedforward/bias it is possible to inject some variable of the system on the output of the controller and/or apply a displacement on the same one.

The objective of feedforward is to measure the main riots of the process and to calculate the change necessary in the performance variable to compensate them before these cause alterations in the controlled variable. The manipulation of the riots of the process can be made through the blocks of

advanced control (F-CTRL.059) that possess advance-delay blocks, derivation with secular retardation and first-class secular retardation.

It can be cited as example, a system where the variable to be controlled is the temperature of a hot mixture. In one determined phase of the process it is necessary to spill cold water in this mixture. Without feedforward, it would be necessary to wait the cold water to change the state of the mixture for then the controller to generate the corrective action. Using feedforward, a value associated with the temperature of the cold water would be injected in the exit of the controller, making with that this takes “the corrective action” before the cold water start to modify the state of the hot mixture, speeding the reply of the controller.

The bias it is used always that it is desired to apply some displacement on the output of the controller.

Control in Cascade

Probably the control in cascade is one of the most used techniques of advanced control in the practical one. He is composed at least two control loops. The figure below shows a controller in cascade with two loops.

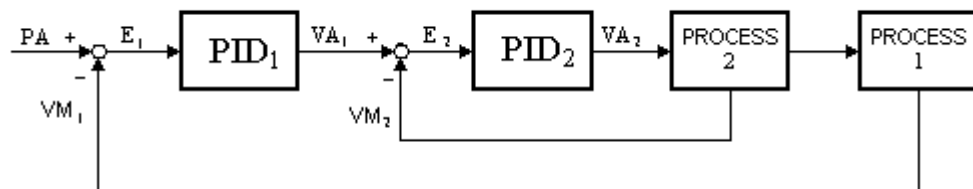


Figure 34: Controller in cascade with two loops

The external loop is called the master controlling and the internal loop the slave controller. The master controlling has its adjustment point fixed and its output supplies the adjustment point of the slave controller (VA_1). The performance variable of the slave controller (VA_2) will act on process 2 that, in turn, will act on process 1, closing the controlling master loop.

This type of controller is applied, for example, in the temperature control for the vapor injection. Beyond the variation of the temperature, that must be controlled, the system is displayed to variations of pressure in the vapor line. One becomes then desirable an outflow slave controller acting in function of the pressure variations and a controlling master to manipulate the reference of the slave, controlling then the temperature of the process. This example can be represented in agreement with the graphically figure below.

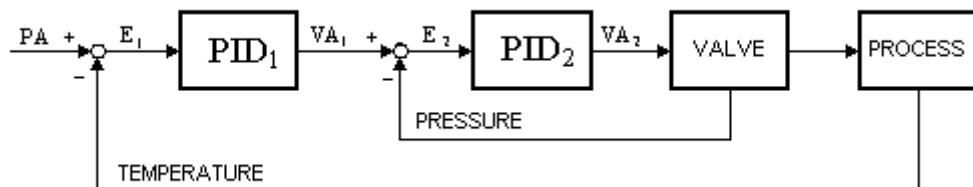


Figure 35: Application of a controller in cascade

In case that a temperature controller that only act directly on the vapor valve, it would not have to compensate eventual variations of pressure in the vapor line.

Three main advantages in the use of controllers in cascade:

- Any riot that affects the slave controller is detected and compensated for this controller before affecting the controlled variable for the master controlling;
- Increase of the controllability of the system. In the case of the temperature control for the vapor injection, the system reply is improved because of the outflow controller increasing the controllability of the main loop;
- Non linearities of an internal loop are manipulated inside of this loop and not perceived by the external loop. In the previous example, the pressure variations are compensated by the slave controller and the master controlling “can see” just a linear relation between the valve and the temperature.

Important Notes

To use controllers in cascade the following cares must be taken:

- As the adjustment point of the slave controllers is manipulated in agreement with the masters controlling output, will be able to occur brusque variations in the error of the slave controller. If the slave controllers will be with the derivative acting in function of the error will appear derivative actions with great values. Therefore it advised to use the slave controllers with the derivative action in function of the measured variable;
- The slave controller must be sufficient fast to eliminate the riots from loops before these affect the loops of the master controlling;

PID Controller Adjustments Suggestions

Two methods for the determination of the PID controller constants of controller are presented. The first method consists on the determination of the constants in function of the period of oscillation and the critical gain, while it determines the constants of the controller in function of the time constant (T), of the dead time (T_m) and of the static gain of the system (K). For bigger details is advised reading literature references to it.

WARNING:

Altus are not responsible for eventual damages caused by configuration errors of the constants of the controller or parameters configuration. One sends regards that a qualified person executes this task.

Determination of the Constants of the Controller Through the Period and Critical Gain

This method generates a cushioned reply whose tax of damping is equal to 1/4. Or either, after syntonizing a loop through this method, a reply as shown in the figure below is expected:

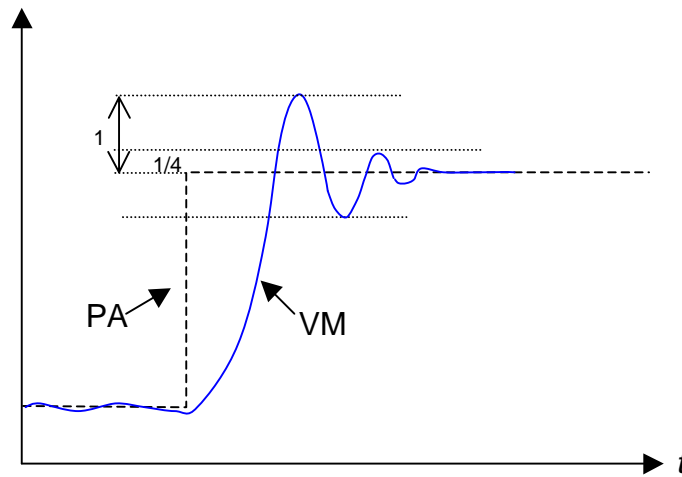


Figure 36: Reply with 1/4 of damping tax

The critical gain is defined as the gain of a proportional controller who generates an oscillation of constant amplitude in the system in closed loop and the critical period is the period of this oscillation. The critical gain is a measure of controllability of the system, or either, bigger the critical gain easier will be the control of the system. The critical period of oscillation is a measure of the speed of the system reply in closed loop, or either, how much bigger the period of oscillation slower will be the system. In elapsing of this chapter the critical gain will be called as G_{Pc} and the critical period as T_c.

It is important to remember that slightly lesser gains that G_{Pc} generate oscillations whose period decreases with the time, while that bigger gains that G_{Pc} generate oscillations whose amplitude grows with the time. In the case of bigger gains that G_{Pc} is necessary to have well-taken care of not to become the system critically unstable.

The process to determine G_{Pc} and T_c consists of closing the loop with the controller in automatic mode disabling the integral action and the derivative. The steps are the following ones:

- Remove the integral and derivative action through operand %A of control;
- Increase the proportional gain with small increments. After each increment inserting a small riot in the system through a small step in the adjustment point (PA). To verify the behavior of the system (VM), the amplitude of oscillation must increase while the gain increases. The critical gain (G_{Pc}) will be that one that generate oscillations with constant amplitude (or almost constant) as the figure below;

- To measure the period of these oscillations (T_c).

To determine the constants of the controller it is enough to apply the values of GP_c and T_c on the equations of the table below.

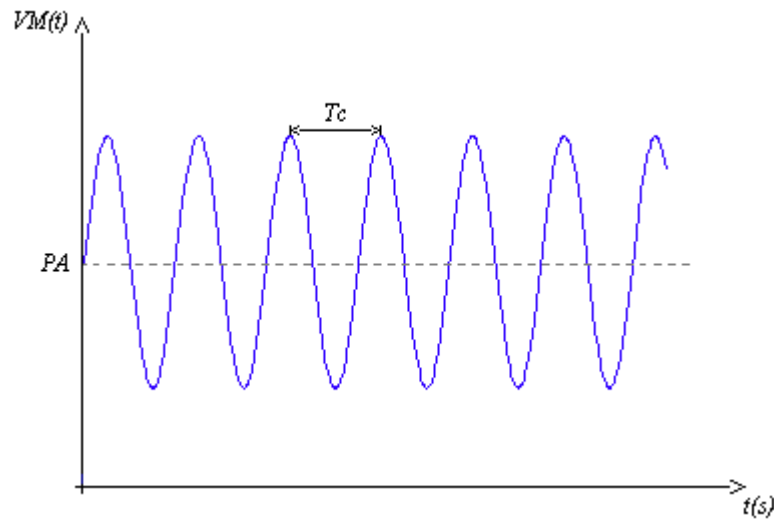


Figure 37: Graphic representing of a system out of control when subjected to GP_c

Type of the Controller	Constants
Proportional (P)	$GP = 0,5 \cdot GP_c$
Proportional and Integral (PI)	$GP = 0,45 \cdot GP_c$ $Ti = \frac{T_c}{1,2}$
Proportional, Integral and Derivative (PID)	$GP = 0,75 \cdot GP_c$ $Ti = \frac{T_c}{1,6}$ $Td = \frac{T_c}{10}$

Table 8: Equations to determine the constants of the controller

Determination of the Constants of the Controller Through the Constants of the Process

This method applies the linear processes, first-class (similar to a circuit RC) and with dead time. In the practical, many industrial processes is adapted to this model.

The method requires, initially, to determine the following characteristics of the process in open loop:

- K : Static gain of the process. Defined as the division between a variation of VM and a variation of VA , or either, $K = \Delta VM / \Delta VA$;
- T_m : Dead time, defined as the time between the beginning of a variation on output VA (t_0) and the beginning of the reaction of the system;
- T : time constant of the system, defined as the time that the measured variable leads to reach 63.212% of its final value;

Moreover, the method requires two additional parameters, that are not characteristic of the process in itself, and must be informed by the user:

T_r : time of reply desired after the tuning of the loop. Through this parameter the user can inform a requirement of performance of the controlled loop.

dt : time of sampling in seconds, or either, the period of call of the F-PID16.056 and update of input VM and output VA. The constant dt symbolizes an additional dead time, that must be added the T_m . In the practical, adds $dt/2$ to the value of T_m , therefore this is the inserted average dead time.

The time of T_r reply can be compared with a “time constant” of the closed loop, as illustrates the figure below.

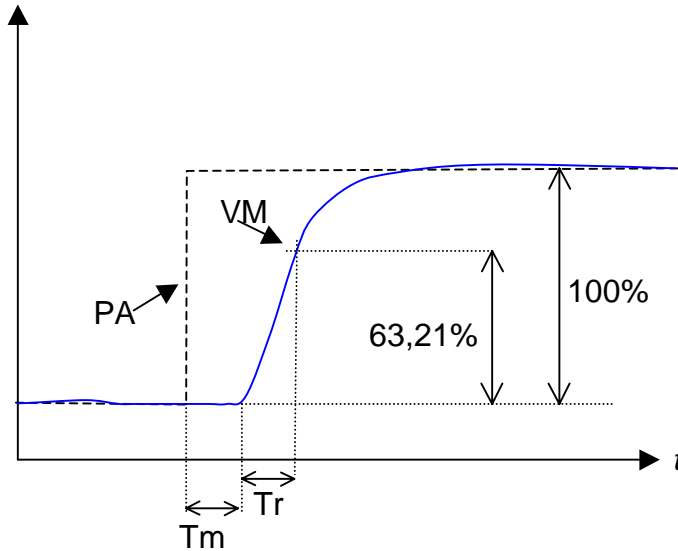


Figure 38: Specification of the Reply Time T_r

The T_r parameter shows the desired time of reply. It is about the measured time between the beginning of the reply of the system (after the dead time T_m), and the moment where VM reach 63.21% of its total excursion. Through T_r the user can specify a “performance requirement” for the controlled loop. Be careful in not to specify minor times of reply that one tenth of the constant of time of the system, therefore the system can be unstable. How much minor the value of T_r , greater the need of gain.

To follow, it is described how to determine, through a test of opened loop, the other parameters (K , T_m and T), that characterize the process. A simple way to determine these constants of the process is to place the F-PID16.056 module in manual mode, generate a small step in VA and plot the reply of VM in time. For slow processes this can manually be made, but for fast processes the use of an oscilloscope or any another device that monitors the variation of VM is advised. The step in VA must be sufficient to cause a perceivable variation in VM.

The figures below represent, respectively, a step on VA output, applied in the instant t_0 and the reply of a first-class linear system with dead time.

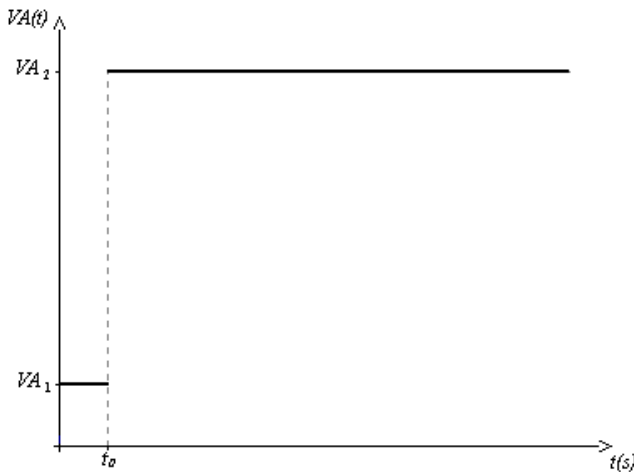


Figure 39: Step on VA

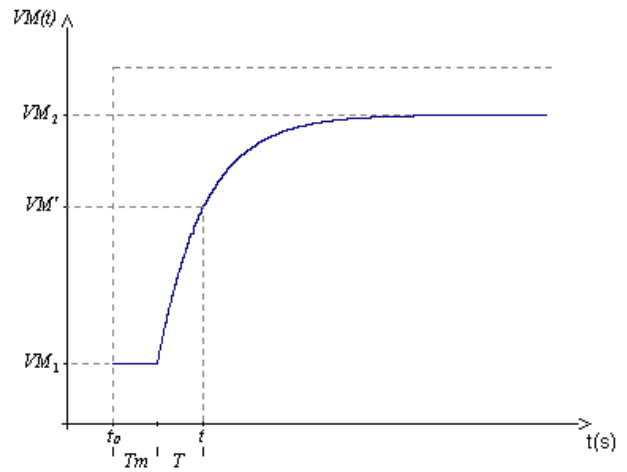


Figure 40: Reply to the step

Through these figures it can be gotten all the necessary constants for the determination of the parameters of the controller. The static gain of the process is gotten through the division between the variation of the measured variable and the variation of the performance variable, or either:

$$K = \frac{VM_2 - VM_1}{VA_2 - VA_1}$$

The dead time, T_m , is the time between the moment of application of the step in VA (t_0) and the beginning of the reply of the system.

The constant of time of the system, T , is the time between the beginning of the reaction of system and 63.212% of the final value of VM (VM), or either:

$$0,63212 = \frac{VM' - VM_1}{VM_2 - VM_1}$$

From the constants of the system, K , T_m and T , can be gotten the parameters of the controller using the equations of the table below:

Type of the Controller	Constants
Proportional, Integral and Derivative (PID)	$GP = \frac{T}{K * (Tr + Tm + dt/2)}$ $Ti = T$ $Td = Tm/2 + dt/4$

Table 9: Equations to determine the parameters of the controller

Gains X Scales

It is important to remember that the proportional gain will only execute its action in correct way when the Input and the output of the system are using the same scales. For example, a proportional controller with unitary gain and input (VM) using the band from 0 to 1000 will only be really unitary if the output band (VA) also is from 0 to 1000.

In many cases the scales of inputs and outputs are different. A system can be cited as example where the card of analogical entrance is of 4-20 mA, where 4 mA corresponds to value 0, and 20 mA corresponds to value 30000. And the card of analogical output is from 0V to 10 V, where 0 V corresponds to value 0, and 10V corresponds to value 1000. In cases like this, the adjustment of scales can be made through the proportional gain instead of a normalization of the values of input or output.

A strategy that can be adopted is, initially, to determine the gain in percentile terms (independent of scales), without worrying about the type of modules of used analogical input and output. Later, after this gain determined, the correction of scales must be executed, before introducing the proportional gain in the F-PID16.056 module.

The strategy consists in determining the proportional gain of the system using the percentile band (0% the 100%) of the variable measured (VM) as the performance value (VA), without taking in consideration the absolute values, as much of VM as of VA.

This will lead to the determination of a proportional gain called GP%. This GP% gain cannot directly be used in the F-PID16.056. Before it is necessary to make a correction of scales, that considers the absolute values of these variable.

Warning: In the previous section, *Suggestions for Adjustments of PID Controller*, are suggested methods of adjustment which the correction of scales are implicit to the method, not having to be considered. In the following chapter, *Example of Application*, the correction of scales also is unnecessary, therefore was used one of the methods boarded in the section *Suggestions for Adjustments of PID Controller*.

The correction of scales is illustrated on the example below.

Considering a conditional air system where the module of analogical input is reading an electrical resistance PTC (positive thermal coefficient) and the module of analogical output generates a tension from 0 to 10V to act on the responsible valve for the circulation of the water that cools insufflating air.

The entrance module works with a band from 0 to 30000, however the useful band is from 6634 to 8706 with the following meaning:

- $EA_0 = 6634 = 0\% = 884,6 \text{ ohms}$ (corresponds to the minimal temperature that can be measured)
- $EA_1 = 8706 = 100\% = 1160,9 \text{ ohms}$ (corresponds to the maximal temperature that can be measured)

The output module uses the same band from 0 to 30000 without restrictions and with the following meaning:

- $SA_0 = 0 = 0\% = 0V$ (corresponds to the minimal water outflow through the valve)
- $SA_1 = 30000 = 100\% = 10V$ (corresponds to the maximal water outflow through the valve)

Assuming that the GP% gain has been previously determined, the GP gain can be calculated by the following equation:

$$GP = GP\% * R$$

where:

$$R = \frac{SA_1 - SA_0}{EA_1 - EA_0}$$

For the previous example:

$$R = \frac{30000 - 0}{8706 - 6634} = 14,478$$

This reason R is a constant that, when multiplied by the proportional gain of the controller, it compensates the differences between the inputs and outputs bands without the necessity of a direct normalization.

Example of Application

On this item it is demonstrated a practical example of the F-PID16.056 module use, enclosing diverse phases of the project, of the process and its system of control.

Description of the Process

The process example has as objective the warm water supply, with controlled temperature, to the consumer. The heating will be made through a gas burner, being controlled on the variation of the gas outflow through a valve.

The figure below illustrates this process.

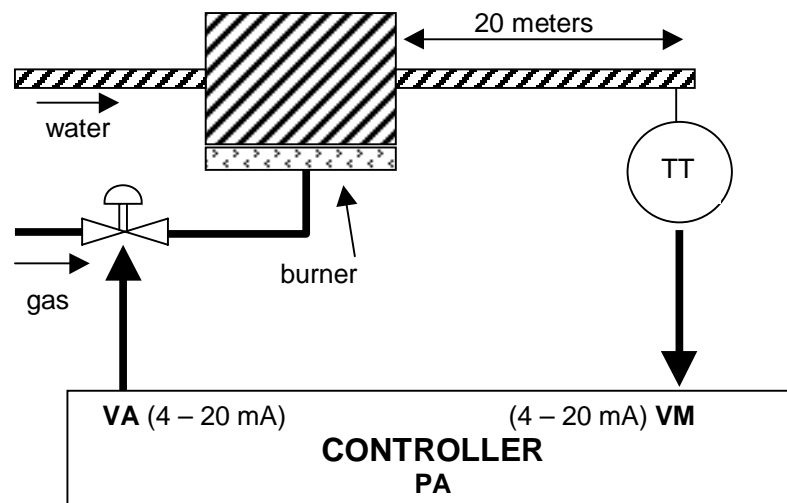


Figure 41: Process of burning water

On the figure, it is observed that the temperature transmitter (TT) is close to the consumer, who is located 20 meters from the point of the water heating. Processes as this are good examples to illustrate how can be introduced “dead times”. This because the warm water in the heating point takes some time to cover the distance until the point of measurement next to the consumer. Dead times had been argued previously.

Some hypotheses had been assumed on the model of this process:

It is assumed that the water that arrives at the heating point on the burner has fixed temperature, of 30 °C.

It is assumed that the water outflow is constant.

Some characteristics of this process and the used elements:

The warm water must have its programmable temperature between 50 °C and 80 °C.

The temperature TT transmissor has output from 4 to 20 mA, and if floodgate of linear form, such that the 4 mA correspond to 30 °C and 20 mA correspond to 130 °C.

It is assumed that, to increase in 10 °C the water temperature, is necessary to inject 1 m³/h of gas. This behavior is linear. The gas valve closes with 4 mA, injecting 0 m³/h of gas. On the other hand, with 20 mA, it injects 8 m³/h of gas.

Description of the Analogical Modules

As it can be seen in figure 4-13, is needed an analogical output from 4 to 20 mA, and an analogical entrance from 4 to 20 mA, as interfaces between the controller and the process.

Internally to the controller, these bands from 4 to 20 mA correspond the numerical bands in M operand (VM and VA). These bands of numerical values can vary in function of the input modules and selected analogical output. In this example, the following is assumed:

analogical input VM (0 a 30000):

VM = 0 ---> 4 mA ---> 30 °C

VM = 30000 ---> 20 mA ---> 130 °C

analogical output VA (0 a 10000):

VA = 0 ---> 4 mA = 0 ---> 0 m³/h

VA = 10000 ---> 20 mA ---> 8 m³/h

Adjustment Point

The PA operand must be used to program the desired temperature, between 50 °C and 80 °C.

As this operand must be compared with VM, it must have the same numerical band of VM, or either:

PA = 0 ---> 30 °C

PA = 30000 ---> 130 °C

Or to restrict the band between 50 °C and 80 °C:

PA = 6000 ---> 50 °C

PA = 15000 ---> 80 °C

General Diagram and Boundary-Values

Figure 42 shows a general diagram of the system (controlling + process), where inside of the controlling is revealed the F-PID16.056 module. To observe that PA, VM and VA are M operands.

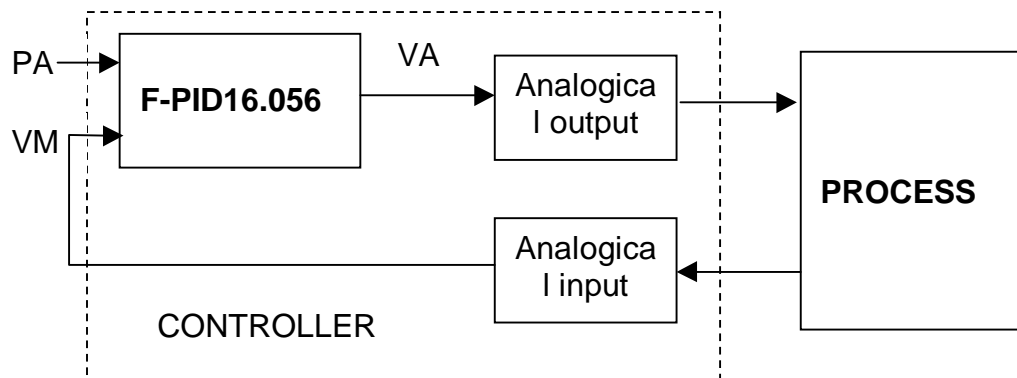


Figure 42: General Diagram

PA:

minimum = 6000 (50 °C)

maximum = 15000 (80 °C)

VM:

minimum = 0 (30 °C)

maximum = 30000 (130 °C)

VA:

minimum = 0 (0 m³/h)

maximum = 7500 ---> ($6 \text{ m}^3/\text{h}$)

It is observed that in the VA case, even so the valve has capacity to inject $8 \text{ m}^3/\text{h}$, is desired to limit this outflow in $6 \text{ m}^3/\text{h}$.

Parameters of the Process

Figure 43 shows the result of a test of an opened loop of the process. To execute this test, was directly used the VA and VM variables, with its internal units.

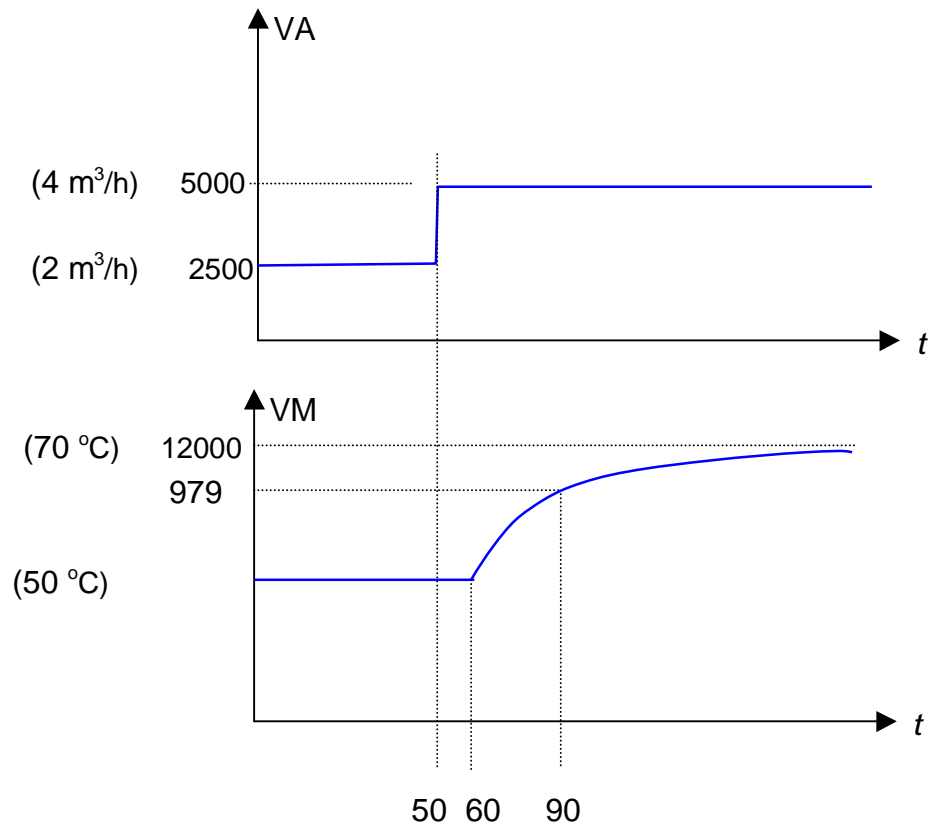


Figure 43: Open Loop Test

On this figure can be determined the 3 basic parameters, as explained previously.

$T_m = 10$ seconds (dead time, since the step was applied in $t = 50$ s and the reply initiated in $t = 60$ s).

$T = 30$ seconds (time constant, since the reply t initiated in $t = 60$ s, and reached 63.21% of the excursion in $t = 90$ s):

$$9792 = 6000 + (12000 - 6000) * 0,6321.$$

$K = 2.4$ (static gain of the process)

$$2.4 = 12000 - 6000$$

$$5000 - 2500$$

Tunning of the Controller

Since the test of opened loop was carried through, will be used the second tunning method previously described on **Notes of Application**.

To use this method, beyond the determined parameters of the process in the previous section (T_m , T and K), also it is necessary that the user informs others 2 parameters:

Tr, or desired time of reply. In this example, it will be decided in 10 seconds (the time constant in open loop divided by 3).

dt, or time of cycle of the F-PID16.056. As commented previously, this time must be 10 times minor than the time constant in loop opened, or still minor. Therefore, the value must be minor than 3 seconds. Was selected dt = 1 second.

Now, it is possible to apply the equations of the method:

$$GP = T / (K * (Tr + Tm + Dt/2)) = 30 / (2.4 * (10 + 10 + 1/2)) = 0,609$$

$$Ti = T = 30 \text{ s/rep}$$

$$Td = Tm/2 + Dt/4 = 10/2 + 1/2 = 5.25 \text{ s}$$

Uses of F-PID16.056

To each one second, must be executed the F-PID16.056, setting in motion its input ENABLE during only one sweeping.

The AUTOMATIC/MANUAL input can be controlled during the operation of the process. Normally the process will be in automatic.

For this process, the REVERSE/MANUAL input will have to be in state 0 (reverse). The process demands control reverse therefore, in the case of an increase of VM, the controller must diminish VA in order to control the process. In other terms, if the temperature increases, the valve must be closed.

Operand TMXXXX:

$$\text{position 0} = GP \times 10 = 6$$

$$\text{position 1} = Ti \times 10 = 300$$

$$\text{position 2} = Td \times 100 = 525$$

$$\text{position 3} = dt \times 100 = 100$$

$$\text{position 4} = \text{maximum output value} = 7500$$

$$\text{position 5} = \text{minimum output value} = 0$$

$$\text{position 6} = \text{dead zone} = 0 \text{ (disable)}$$

$$\text{position 7} = \text{maximum variation allowed} = 0 \text{ (disable)}$$

positions 8 to 29, begins with zeros only on CPU powering

Operand AXXXX of control: all the start bits must be zero.

Comparison with F-PID.033

The F-PID16.056 module was developed aiming to improve the interface with the user, to optimize the execution time and to become it compatible with variable of 16 bits or little resolution.

The main changes:

- Inputs and outputs with a range from -30000 to 30000;
- Parameters inputs without initial calculation (direct input of Gp, Ti, Td and dt);
- Sampling interval (dt) from 10ms to 10s, while that on F-PID.033 the minimum limit is 100ms.

Together with these alterations, a set of new characteristics was added to the previous F-PID. The table below brings a comparison of the characteristics between the F-PID16.056 module and the F-PID.033.

CHARACTERISTIC	F-PID.033	New PID
Parameters programmed directly in ISA format (G_p , T_i , T_d)		X
Calculated derivative action in function of the error or the measured variable.		X
Calculated derivative action on 3 samplings	X	X
Direct or reverse action	X	X
Input and output interval from -100% to +100%		X
Dead band		X
Unsaturation of the integral action ("anti-reset windup")	X	X
<i>Feedforward / bias</i> input (displacement)	X	X
Inhibition of the derivative term	X	X
Inhibition of the integral term	X	X
Inhibition of the proportional term		X
Limitation of the growth tax		X
Adjustable output limits	X	X
Manual / automatic mode	X	X
Accompaniment of the output on the manual mode and balanced manual/automatic commutation ("Output tracking" and "bumpless transfer").	X	X

Table 10: Comparison between the F-PID.033 and the F-PID16.056

F-CTRL.059 – F Module for Advanced Control

Introduction

The module function F-CTRL.059 uses the control algorithms lead/lag, first-order retardation and derivation with first-order retardation. Each operation mode (algorithm) is selected through an index on F-CTRL.059 module.

From an input value, the module calculates an output value in function of the selected algorithm. All the modules use two constants, a time T constant and a second constant K whose function vary as the selected algorithm. The algorithms are executed in discrete mode, and the time of shot of the function must be declared with the parameters.

These functions are used in advanced control algorithms to the optimization of the control loop. Generally used with PID function.

1st Order retardation

When the selected algorithm is the first-order retardation, the F module applies on the input (V_i) operand value a first-order retardation. The output value (V_o) of this function is proportional to the input, however, been late as an exponential function.

This algorithm needs two constants. A T time constant that, in analogy with an RC circuit, represent it load time constant (63,212% at final value) and a constant of proportional gain K .

On frequency domain (s), the first-class retardation follows the transference function below:

$$V_o(s) = \frac{K}{1 + T \cdot s} \times V_i(s)$$

Where $V_i(s)$ and $V_o(s)$ are Laplace transformations of the input and output signals.

The reply to the step is represented on 4-16 figure, where can be observed the T time constant associated with V' value, that represents 63,212% of start and the end value difference.

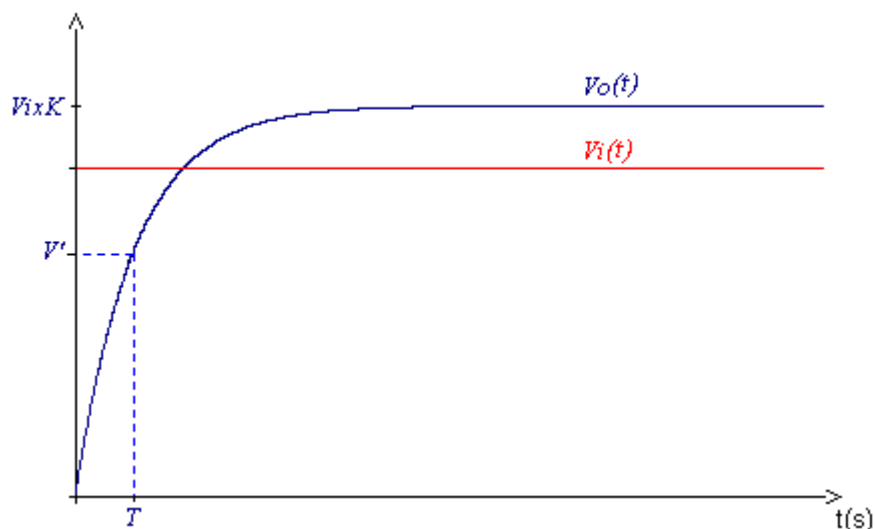


Figure 4-16 first – order retardation

Derivation with Retardation of 1st Order

When this algorithm is selected, the F module applies on the input operand value it derivation with the first-order retardation. The Output value is the input V_i derivation with the retardation as an exponential function.

This algorithm needs two constants. A T time constant that can be extended, on the same analogy of the first-order retardation, as the time constant of discharge of a RC circuit. The second constant is the derivation constant K that, divided by the T constant, will determine a third constant that can be treated as a gain.

On frequency domain (s), the derivation of first-order retardation follows the transference function below:

$$V_o(s) = \frac{K \cdot s}{1 + T \cdot s} \times V_i(s)$$

Where $V_i(s)$ and $V_o(s)$ are Laplace transformations of the input and output signals.

The reply to the step is represented by 4-17 figure. On start instant ($t = t_0$) can be observed that the function (V_o) output is the input step, with A amplitude, multiplied by the K/T division. On $t = t_0 + T$ instant, the system output value is equal to V' , or either, 36,788% of $A \times K/T$. When the V_i input is constant, the output of this function returns zero with a first-order retardation.

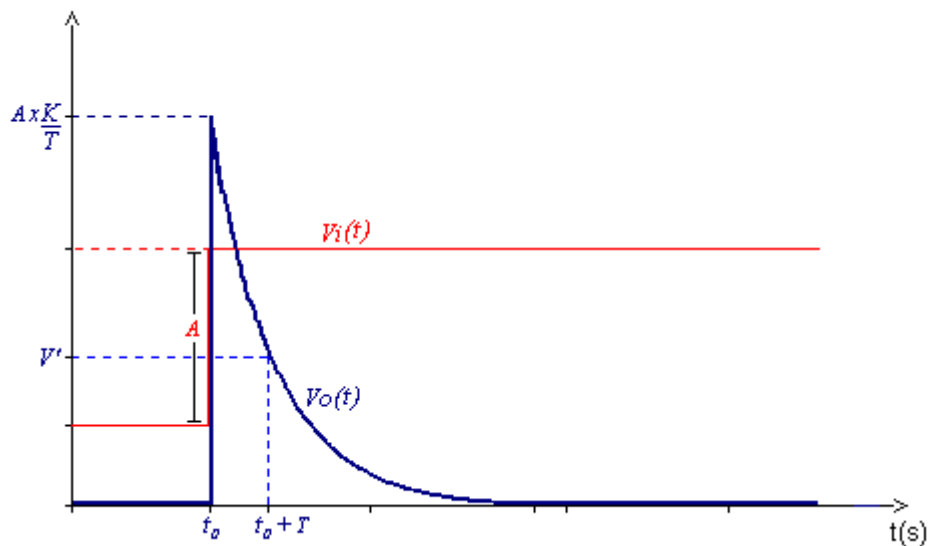


Figure 4-17 Derivation with first order retardation

It is important to remember that the step can not be seen by the F-CTRL.059 module as a instant variation, but as a variation between two sampling. On contrary case it derivation would have an infinite value.

Lead/Lag

When this algorithm is selected, the F module applies on the input operand value the lead or the lag as the relation between the declared constants.

The algorithm needs two constants. One T time constant that, as the same way that the algorithms before, can be extended as the time constant of a RC circuit. And a K constant that, with the constant T , will define the algorithm behavior as lead or lag.

Always if the T time constant is bigger than the constant K , the algorithm will behave as lag. When K is bigger then T it behavior will be as lead. The constants K and T are known too as lead and lag constants, respectively.

On frequency domain (s), the Lead/Lag follows the transference function below:

$$V_o(s) = \frac{1 + K \cdot s}{1 + T \cdot s} \times V_i(s)$$

Where $V_i(s)$ and $V_o(s)$ are the Laplace transformations of the inputs and outputs signals.

The reply to step of the lead is represented by the 4-18 figure. On $t = t_0$ instant can be observed that the function output $V_o(t_0)$ is equal to V'' that can be described as

$$V'' = V_i(t) + A \times \frac{K}{T}, \text{ to } t < t_0,$$

or either, the input value before the step plus the amplitude of the step applied to input (A) multiplied to the division of K/T . On $t = t_0 + T$ instant, the system output is equal to V' , or either, 36,788% of the difference between the max value (V'') and the $V_i(t)$ value for $t > t_0$ plus a displacement equal to $V_i(t_0)$.

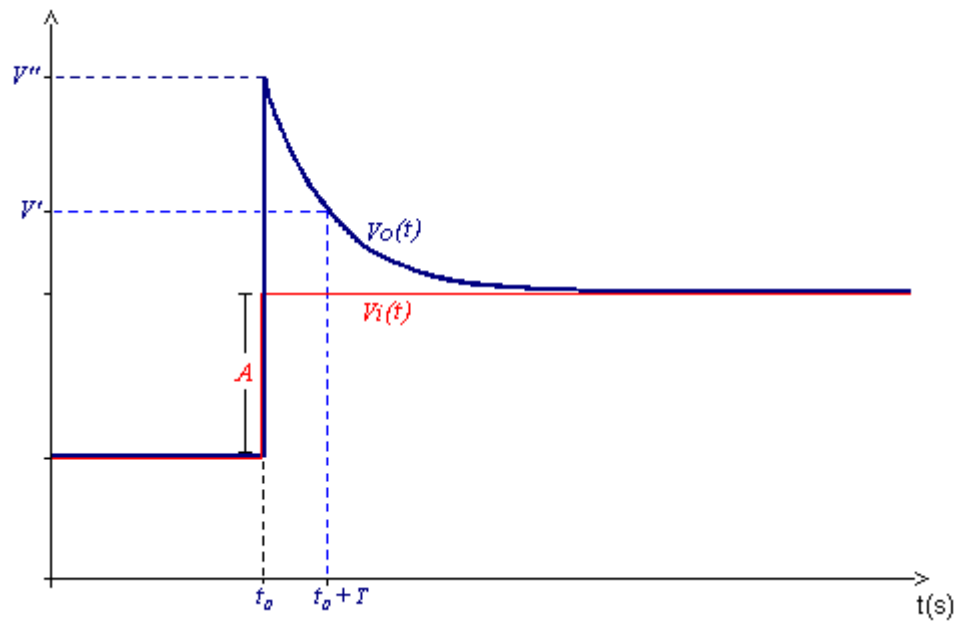


Figure 4-18 Lead

The reply to step of the lag is represented by 4-19 figure. On $t = t_0$ figure it can be observed that the $V_o(t_0)$ function output is equal to V'' that, on the same way of lead, can be written as:

$$V'' = V_i(t) + A \times \frac{K}{T}, \text{ to } t < t_0,$$

differing of the lead graph because K is minor than T . On $t = t_0 + T$ instant, the output of the system is equal to V' , or either, 63,212% of the difference between the $V_i(t)$ value, to $t > t_0$, and the V'' value, plus a displacement equal to $V_i(t_0)$.

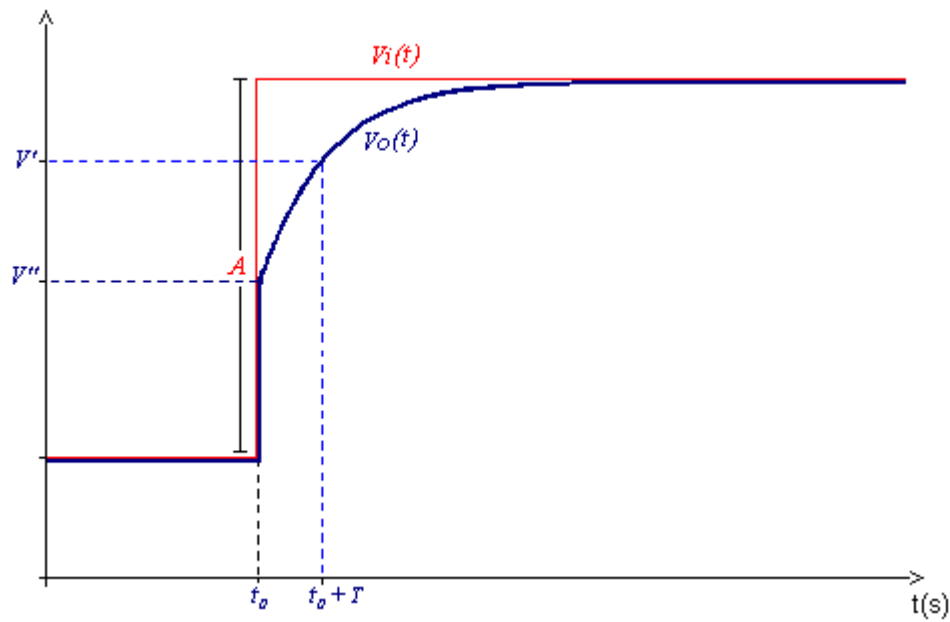


Figure 4-19 Lag

Programming

Operands

The cells of CHF instruction used to the function call are programmed as follows:

- **OPER1** - It specifies the number of parameters that are passed to the function in OPER3. This operand must be obligatorily a memory constant with value 8 (**%KM+00008**).
- **OPER2** - It specifies the number of parameters that are passed to the function in OPER4. This operand must be obligatorily a memory constant with value 0 (**%KM+00000**).
- **OPER3** - It contains the parameters that are passed to the function, declared when the CHF instruction is edited. The number of editable parameters is specified on OPER1, equal to 8 on this module:
 - **%KMXXXX** – Memory constant that points the used algorithm, it can assume the following values:
 - %KM0000** – first-order retardation;
 - %KM0001** – derivation with first-order retardation;
 - %KM0002** – lead/lag.
 - **%KMXXXX** – Constant with the sampling interval value. It assumes values from 0,01 to 10s, and it must be multiplied by 100 to be declared on this field.
 - **%MXXXX** – Memory with the time T constant value. It assumes values from 0,01 to 320s, and it must be multiplied by 100 to be declared on this field.
 - **%MXXXX** – Memory with the K constant value. It assumes values from 0,01 to 320, and it must be multiplied by 100 to be declared on this field.
 - **%MXXXX** – Memory with input value from -32768 to +32767.
 - **%MXXXX** – Memory with the output value from -30000 to +30000.
 - **%MXXXX** – Internal use. Not declared.

%MXXXX – Internal Use. Not Declared.

- **OPER4** – Not used.

Inputs and Outputs

Description of the inputs:

- **enable** - when this input is powered the function is called, analyzing the parameters programmed on CHF instruction. If the number of parameters or its type are different from the function needs, the success/error outputs will be unpowered. If it is correct, the algorithm selected is realized.

Description of the outputs:

- **success(1) /error (0)** - it is powered when the function is correctly executed. It is not powered if an operand specification error occurs, or trying to access not declared operands or invalid parameters.

Characteristics of Functioning

For each sampling interval the function input is applied on the algorithm and update the function output value.

It can be observed that the algorithm is applied on a discrete form, the sampling time (dt) must be 10 times minor than the time constant T to a satisfactory result. The interval between the sampling of the F-CTRL.059 loops module can vary from 0,01 to 10 seconds. It is the user responsibility programming the “start” of the function, or either, an application program that enable the F module only on desired time interval. It is advised to use a E018 module, this module is executed in a fixed time interval that can be used to generate one or more time bases to one or more F-CTRL.059 loops execution. Note that the sampling interval declared on parameters must coincide with the time interval of the “starter” call.

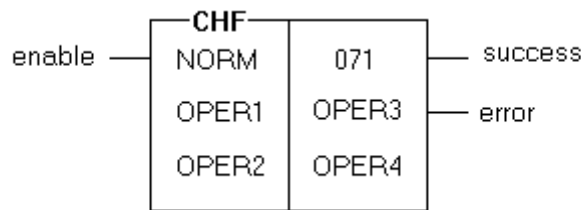
It is important to remember that the inputs and outputs update occurs on the same time order of the CPU cycle. Always if the CPU cycle time is bigger than the sampling time it is advised the use of AES instruction to the AL-2003 and AL-2004 CPUs or F-AES to the Ponto series.

Execution Times

- First-order retardation: 298 µs
- Derivation with first-order retardation: 338 µs
- Lead/Lag: 338 µs

These times are valid to AL-2003, AL-2004, PO3145, PO3142 and PO3242 CPUs.

F-NORM.071 – Function to Normalization



Introduction

The function **F-NORM.071** normalizes whole operands, implementing the function $M[\text{output}] = (M[\text{input}] - A) \cdot C / (B - A)$, where A, B and C are constants.

$$M(\text{saída}) = \frac{(M(\text{entrada}) - A) \cdot (C)}{B - A}$$

Programming

- OPER1 - Specifies the number of parameters passed to the function in OPER3. This operand must be a memory constant with value 6 (%KM+00006).
- OPER2 - Should be an operand of type memory constant with the value 0 (%KM+00000). It determines the number of parameters possible to be programmed in the editing window of OPER4. As this function does not need any parameter in OPER4, the value of OPER2 is 0.
- OPER3 - Contains the parameters passed to the function, declared through a window visualized in MasterTool when the CHF instruction is edited. The number of editable parameter is specified in OPER1, being 6 for this call:
 - %KM+XXXX - Number of operands to process (1 to 127)
 - %MXXXX - Initial input operand
 - %MXXXX - Initial output operand
 - %KM+XXXX - Offset to subtract from the input operand (A)
 - %KM+XXXX - Value reference of the input (B)
 - %KM+XXXX - Value normalized by the output corresponding to B (C)

From version 1.10 of F-NORM.071, available to PLCs AL-2003, AL-2004 and Ponto Series, can be used a **D** parameter, that defines the start input band. When the 6 parameters on CHF from F-NORM.071 version 1.10, will be admitted that the 7th parameter **D** is equal to zero and executed the same normalization algorithm.

The normalization algorithm will be typed as:

$$M(\text{saída}) = \frac{(M(\text{entrada}) - A) \cdot (C - D)}{B - A} + D$$

After the 7th value declared (%KM+00007) to the OPER1, the parameters passed through OPER3 are as following:

- %KMXXXXX number of operands;
- %MXXXXX first input operand;
- %MXXXXX first output operand;
- %KMXXXXX start of the input band(A);

- **%KMXXXXXX** end of the output band(B);
- **%KMXXXXXX** end of the output band(C);
- **%KMXXXXXX** start of the input band (D) (only on version 1.10).

Operation

A **F-NORM.071** implements the following calculation:

$$M[\text{output}] = (M[\text{input}] - A) * C / (B - A)$$

being:

- M [input] - range of whole operands of input
- M [output] - range of whole operands of output
- A - offset for the input
- B - reference value of the input to normalize
- C - normalize d value of the output corresponding to B

The output is the Normalization of the input and the way that for input data with the value **A** the corresponding output is **0**, and for an input value **B** the corresponding output will be **C**. If in this range, the output value will be proportional to the input, according to the formula given.

The function works with a band of up to 127 operand (1 to 127).

Inputs and Outputs

Descriptions of inputs of the function:

- **enable** - when this input is powered the function is called, the parameters programmed in the CHF instruction being analyzed. If they are incorrect, the output instruction error is powered, and the rest become turned off. If the parameters are correct, only the success output is powered.

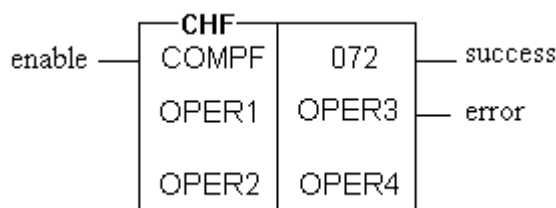
Descriptions of the output of the functions:

- **success** - indicates that the call parameters are correct and that the function has been correctly executed. See observation.

- **error** - is connected if an error occurs in the call parameters. See observation.

Observation: when both outputs (success and error) stay powered is because M input operands range is the same of M output operands range (in parameters).

F-COMPF.072 – Function for Multiple Comparisons



Introduction

The function **F-COMPF.072** divides an operand into specified ranges, presenting output in binary form, where the bit connected indicates the operand pertaining to the respective band.

Programming

The cells of the CHF instruction used for the call are programmed in the following way:

- **OPER1** - Specifies the number of parameters passed to the function in OPER3. This operand must be a memory constant with value 4 (%KM+00004).
- **OPER2** - Should be an operand of type memory constant with the value 0 (%KM+00000). It determines the number of parameters possible to be programmed in the editing window of OPER4. As this function does not need any parameter in OPER4, the value of OPER2 is 0.
- **OPER3** - Contains the parameters passed to the function declared through a window visualized in MasterTool when the CHF instruction is edited. The number of editable parameters is specified in OPER1, being 4 for this call:
 - **%KM+XXXX** - Number of operands %MXXXX to examine
 - **%MXXXX** - Initial input operand for comparison
 - **%MXXXX** - Initial output operand for the indicator bits
 - **%TMXXXX** - Table which specifies up to 16 ranges of values to qualify the input (operands/c.f. format to follow)

Table Position	Contents
0	Reserved
1	Reserved
2	Start range 0
3	End range 0
4-31	<continue the range definitions>
32	Start range 15
33	End Range 15

Table 4-11 Band definition

The table should have a minimum size of 4 position (1 range). To optimize the function's execution time, it is recommended that the table is defined with the exact size to count the definitions of the necessary ranges.

- **OPER4** – Not used.

Operation

The beginning and end of each comparison range are specified as whole numbers.

The operand is considered in the range if this condition is true:

(start of range) = %MXXXX < (end of band)

Each range in the table %TMXXXX corresponds to a bit in the operand %MXXXX being that the 0 bit of the output operand corresponds to the range 0 and so on successively. The bits correspond to the ranges not defined are always 0. The ranges can be superimposed.

The number of operands to process is given by the first parameter (%KM+XXXX), being able to be defined from 1 to 127.

Inputs and Outputs

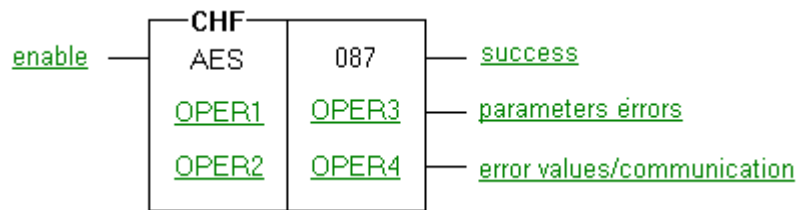
Description of the function inputs:

- **enable** - when this input is powered the function is called, with the parameters programmed being analyzed in the CHF instruction. If they are incorrect, the error output of the instruction is powered and the rest are turned off. If the parameters are correct only the output success is powered

Description of the function output:

- **success** - indicates that the call parameters are correct and that the function has been correctly executed.
- **error** - is connected if an error occurs in the call parameters.

F-AES.087 – Inputs and Outputs Immediate Update Function



Introduction

This instruction executes an immediate update on the image memory and on specified module physical positions. Its action is equal to the I/O points sweeping made by the executive program on each sweeping end, however with a limited number of positions.

If the CPU is a PO3242 or a PO3342 this function allows to update the PROFIBUS network devices too.

Programming

The cells of the instruction CHF used for the call are programmed as follows:

- **OPER1** - Specifies the number of parameters that are passed to the function on OPER3. This operand must be obligatorily a memory constant with value 2 (%KM+00002).
- **OPER2** - It must be an operand of type constant memory with value 0 (%KM+00000).
- **OPER3** - It contains the parameters that are passed to the function, declared when the CHF instruction are edited. The number of edited parameters is specified on OPER1, it is equal to 2 to this F module:
 - **MXXXX, KMXXXX** Specification of the start physical position on bus to be updated. The existing module on the position here declared will have the operand value updated correspondent to the outputs, on the case of the output module, or the inputs will be read to the correspondent operand, on an input module. The value of the physical position is from 0 to 39.
 PROFIBUS Network: the position must be from a PO4053 module. Case the network is redundant, the position of just one of the modules is sufficient to the network update.
 - **MXXXX, KMXXXX - MXXXX, KMXXXX** - Specification of how many positions on the bus will be updated, including the start position. For Example, if it is specified the value 2, will be updated the position declared as start position, and the next position. The value of this parameter must be from 1 to 10.
- **OPER4** – Not used.

Inputs and Outputs

Description of the function inputs:

- **enable** - when this input is powered, the function is called, analyzing the programmed parameters on CHF instruction. If it all correct, the function is executed and the positions are updated. If there is some incorrect parameter, the outputs are enabled pointing the error and it is not updated.

Description of the function output:

- **success**- This output is enabled when the function is correctly executed and the bus positions updated.

- **parameters error:** this output is enabled when some operand not declared is passed as a parameter. When this happens, no position is updated.
- **values error:** this output is enabled when some of the parameters contain an invalid value. This happens if the value is out of the range allowed to this function, or if there is no module declared at C module to the position that must be updated.

If none of three outputs are powered, means that the function could not be executed because an internal stop of the CPU. This can happen, for example, if a function is in a E018 module, and if the E018 is executed on the same instant that the C module is charging on CPU.

Execution Time

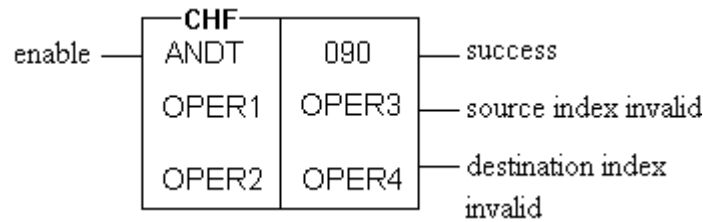
If it is used to update the PROFIBUS network the update time of the network should be considered (see uses manual of PO4053 MU209903).

Redundancy

If the related module position is from a PO4053 of redundancy couple, call just once to F-AES to use the two PO4053 redundancy.

The use of the function F-AES.087 with the E/S forcing at the same time will generate variations on the physical E/S points. This situation should be avoided.

F-ANDT.090, F-ORT.091 and F-XORT.092 – Function Logical Operations between Table Operands



Introduction

The function **F-ANDT.090**, **F-ORT.091** and **F-XORT.092** allow the carrying out of logical operations AND/and), OR (or) or XOR (or exclusive), respectively, between simple operands (M or D) and or tables (TM or TD). Up to 255 logic operations in one single function call. It is necessary that the three operands (supply, supply 2 and destination) are of the same type (memory or decimal).

Programming

The cells of the CHF instruction used to call the programs in the following way:

- **OPER1** - Specifies the number of parameters passed to the function in OPER3. This operand must be a memory constant with value 3 (**KM+00003**).
- **OPER2** - Should be an operand of type memory constant with value 0 (**KM+00000**). Determine the number of parameters possible to be programmed in the editing window of OPER4. As this function does not need any parameter in OPER 4, the value of OPER is 0.
- **OPER3** - Contains the parameters passed to the function, declared through a window visualized in AL-3830 when the CHF instruction is edited. The number of editable parameters is specified in OPER1, being set at 3 for this module:
 - MXXXX, DXXXX, TMXXXX or TDXXXX – simple or table operand where the value will be used to carry out the logic (operand source 1).
 - MXXXX, DXXXX, TMXXXX or TDXXXX - simple or table operand where the value will be used to carry out the logic (operand source 2).
 - MXXXX, DXXXX, TMXXXX or TDXXXX - simple or table operand where the result of the logic will be stored (destination operand).
 - KMXXXX – number of simple operands or positions on the table where the logic operation will be done.
- **OPER4** – Not used.

Inputs and Outputs

Description of the function input:

- **enable** - when this input is powered the function is called, the parameters programmed in the CHF instruction being analyzed in the CHF instruction. If they are incorrect, the outputs of the invalid index are enabled.

Description of the function output:

- **success** - indicates that the moving was carried out correctly.

- **source index invalid** - indicates that there was an error in the specification of the supply operand:
 - the operand is not declared in the module C
 - there are not enough positions to carry out the logic
- **destination index invalid** - indicates that there was an error in the specification of the destination operand:
 - the operand is not declared in module C
 - there are not enough positions to carry out the logic

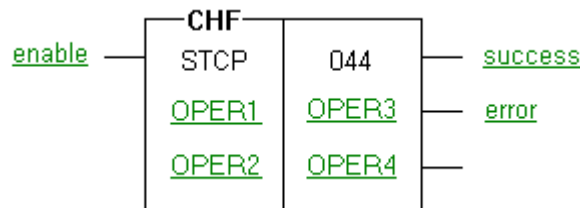
If the two outputs of the invalid index are enabled simultaneously, some of the following errors occurs:

- the number of parameter programmed in OPER is different from three
- the type of one the parameters in OPER3 is not valid
- the parameters in OPER3 is different one with other (memory and decimals)
- the total number of positions to be transferred is more than 255

WARNING:

This function allows the denial of a large number of operands in a single scan. It should be used with care so that the maximum time of the program cycle is not exceeded.

F-STCP.044 – CPU Status Function



Introduction

The function F-STCP.044 returns the CPU status in a M box operand or in a TM. This status correspond to the same parameters that are answered on command 37 from ALNET1 protocol.

Programming

The instruction cells CHF used to call the function are programmed as following:

- **OPER1** - It specifies the number of parameters that are passed to the function on OPER3. This operand must be a constant memory with a value equal to 1(**KM+00001**).
- **OPER2** - Must be an constant memory operand with a value equal to 0 (**KM+0000**). This operand defines the number of possible parameters to be programmed on OPER4 edit window. As this function do not need any parameter on OPER4, the value of OPER2 is 0.
- **OPER3** - It contains the parameters that are passed to the function, declared on a window in MasterTool, AL-3830 or AL-3832 when the instruction CHF is edited. The number of edited parameters is specified on OPER1, fixed in 1 to this module:

TMXXXX or MXXXX: Table or a box of M operands where will be written the status that was read from PLC. The table must contain 50 positions. In the case of M operands the function write on the declared operand and on the next 49, and these must be declared on C module of the project.

- **OPER4** – Not used.

Inputs and Outputs

Description of the inputs:

- **enable** - when this input is powered the function is called, being analyzed the parameters programmed on CHF instruction. If the number of parameters or it type are different of the function needs or if exist a minimal number of declared operands after pointed on the function, it will have a powering on error output. If these are correct, it scan the status parameters and copy to the operands.

Description of the outputs:

- **success** - it is powered when the function was correctly installed.
- **error** - it is powered if there is an error on specification or on trying to access not declared operands.

Description of the Operand Status

Below, the description of the operands and the index that it can be found in the function table. In the case of M operand, the declared operand represent the 00 index.

Operand	Identification	Description
00	Identification of the PLC Model	00H - AL-3003 01H - AL-3004 20H - AL-2000 21H - AL-2002 22H - QK2000

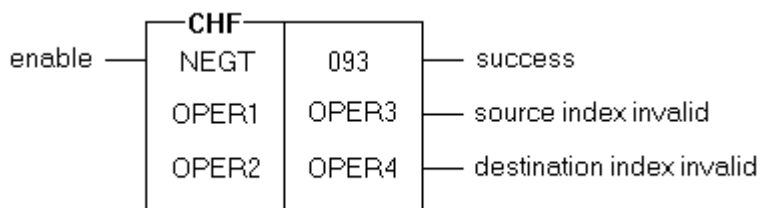
		23H - AL-2003 24H - AL-2004 40H - AL-600 41H - QK600 50H - QK800 51H - QK801 A0H - PL101 A1H - PL102 A2H - PL103 A3H - PL104 A4H - PL105 A5H - PL106 A6H - PL107 B0H - GR310 B1H - GR316 B2H - GR330 B3H - GR350 B4H - GR351 B6H - GR370 B7H - GR371 C0H - PO3042 C1H - PO3142 C2H - PO3242 C3H - PO3342 C6H - PO3045 C7H - PO3145 C8H - PO3245 C9H - PO3345
01	Character 0 of auxiliary identification of the PLC	String of auxiliary identification of the PLC, it can take up to 8 characters, on ASCII format
02	Character 1 of auxiliary identification of the PLC	
03	Character 2 of auxiliary identification of the PLC	
04	Character 3 of auxiliary identification of the PLC	
05	Character 4 of auxiliary identification of the PLC	
06	Character 5 of auxiliary identification of the PLC	
07	Character 6 of auxiliary identification of the PLC	
08	Character 7 of auxiliary identification of the PLC	
09	Executive version (high part)	Format V.R.C, where V is the number of the version, R is the number of the issue and C is the number of the last correction. On the high part the V is stored and on the low part R and C are stored on the nibbles 1 and 0, respectively.
10	Executive version (low part)	

11	Operation Mode 1 of the PLC	<div> <div>F E D C B A 9 8</div> <div>exe prg cic tst cop for cpt sai</div> </div> <p> exe: PLC on execution mode prg: PLC on programming mode cic: PLC on cycled mode tst: PLC on mode test cop: copying module from EPROM to RAM for: exist relays forcing cpt: compacting RAM sai: digital outputs disabled </p> <div> <div>7 6 5 4 3 2 1 0</div> <div>trc apg prt</div> </div> <p> apg: erasing flash EPROM prt: protection level of the PLC (0 – no protection - to 3 – total protection) trc: modules changing of E/S with the PLC powered </p>
12	Message Code 1	
13	Message Code 2	
14	Message Code 3	
15	Message Code 4	
16	Immediate time cycle	In ms
17	Medium time cycle	In ms
18	Max time cycle	In ms
19	Min time cycle	In ms
20	Period E018	00H - 50ms 01H - 25ms 02H - 10ms 03H - 5ms 04H - 3,125ms 05H - 2,5ms 06H - 1,25ms 07H - 0,625ms FFH - Sem E018
21	Reserved Operand	
22	Maximum time of program scanning	00 – 100ms 01 - 200ms 02 - 300ms 03 - 400ms 04 - 500ms 05 - 600ms 06 - 700ms 07 - 800ms

23	RAM status of the applicative program	<div>General information of the RAM status of the applicative program and the pointer to the RAM applicative program bank existence:</div> <div><div>F E D C B A 9 8</div><div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div>cpt</div></div></div> <div>cpt: RAM compacted (0) or not (1)</div> <div><div>7 6 5 4 3 2 1 0</div><div><div>B7</div><div>B6</div><div>B5</div><div>B4</div><div>B3</div><div>B2</div><div>B1</div><div>B0</div></div></div> <div>bits 7-0: exist banks</div>
24	Free bytes on RAM applicative program bank 0	
25	Free bytes on RAM applicative program bank 1	
26	Free bytes on RAM applicative program bank 2	
27	Free bytes on RAM applicative program bank 3	
28	Free bytes on RAM applicative program bank 4	
29	Free bytes on RAM applicative program bank 5	
30	Free bytes on RAM applicative program bank 6	
31	Free bytes on RAM applicative program bank 7	
32	Status of the EPROM of the applicative program	<div>Pointer of the EPROM banks of the applicative programs existence:</div> <div><div>F E D C B A 9 8</div><div><div>B15</div><div>B14</div><div>B13</div><div>B12</div><div>B11</div><div>B10</div><div>B9</div><div>B8</div></div></div> <div><div>7 6 5 4 3 2 1 0</div><div><div>B7</div><div>B6</div><div>B5</div><div>B4</div><div>B3</div><div>B2</div><div>B1</div><div>B0</div></div></div> <div>bits F-0: exist banks</div>
33	Free bytes on EPROM applicative program bank 0	
34	Free bytes on EPROM applicative program bank 1	
35	Free bytes on EPROM applicative program bank 2	
36	Free bytes on EPROM applicative program bank 3	
37	Free bytes on EPROM applicative program bank 4	
38	Free bytes on EPROM applicative program bank 5	
39	Free bytes on EPROM applicative program bank 6	
40	Free bytes on EPROM applicative program bank 7	
41	Free bytes on EPROM applicative program bank 8	

42	Free bytes on EPROM applicative program bank 9	
43	Free bytes on EPROM applicative program bank 10	
44	Free bytes on EPROM applicative program bank 11	
45	Free bytes on EPROM applicative program bank 12	
46	Free bytes on EPROM applicative program bank 13	
47	Free bytes on EPROM applicative program bank 14	
48	Free bytes on EPROM applicative program bank 15	
49	Reserved Operand	

F-NEGT.093 – Function for the logic denial of Table Operands



Introduction

The function **F-NEGT.093** carries out the logic denial of simple (M or D) or table operands (TM or TD). Up to 255 positions can be denied in one single function call. The result of the alteration can be stored in this operand, substituting the original value, or in another operand, since it may be of the same type as the first (memory or decimal).

Programming

The cells of the instruction CHF used to call the function are programmed in the following way:

- **OPER1** - Specifies the number of parameters passed to the function in OPER3. This operand must be a memory constant with value 4 (**KM+00004**).
- **# - OPER2** - Should be an operand of type memory constant with the value 0 (**KM+00000**). It determines the number of parameters possible to be programmed in the editing window of OPER4. As this function does not need any parameter in OPER4, the value of OPER2 is 0.
- **# - OPER3** - Contains the parameters passed to the function, declared through a window visualized in AL-3830 when the CHF instruction is edited. The number of editable parameters is specified in OPER1, being set at 3 for these modules:
 - MXXXX, DXXXX, TMXXXX or TDXXXX – Simple operand or table operand where the values will be denied (source operand).
 - MXXXX, DXXXX, TMXXXX or TDXXXX – simple or table operand where the denied values will be stored (destination operand).
 - KMXXXX – Number of simple operands or positions on the table to be denied. Should be less or equal to 255.
- **OPER4** – Not used.

Inputs and Outputs

Description of the function inputs:

- **enable** - when this input is powered the function is called, the parameters programmed being analyzed in the CHF instruction. If they are incorrect, the outputs of the invalid index are enabled.

Description of the function outputs:

- **success** - indicates that the moving was correctly carried out.
- **source index invalid** - indicates that there was an error in the specification of the source operand:

- operand is not declared in module C
 - there are not enough positions to carry out the logic
- **destination index invalid** - indicates that there was an error in the specification of the destination operand:
 - the operand is not declared in module C
 - there are not enough positions to carry out the logic

If the two outputs of the invalid index are enabled simultaneously, some of the following errors occur:

- the number of parameter programmed in OPER1 is different from three
- the type of one or more parameters in OPER3 is not valid
- the destination type operand is different from the supply operand
- the total number of positions to be transferred is more than 255

WARNING:

These functions allow the execution of logic operations of a large number of operands in a single scan. It should be used with care so that the maximum cycle time of the program is not exceeded.

6. Glossary

Glossary to Ponto Series

Address of the Field Network Head: it is the address of a node in the field network. It is adjusted in the field network head module base.

Base: component where the IO modules are inserted, CPUs, power supplies and remaining Ponto Series modules.

Bus: set of IO modules connected to a CPU or Field Network Head.

Bus Expander: module that connects one segment to the next

Bus Segment: part of a bus. A local or remote bus that may be divided into four segments.

Bus termination: component that must be connected to the last module in a bus.

Commercial Code: it is the product code, formed by the letters PO and followed by four digits.

CPU: central processing unit. It is responsible for the application program execution.

Expansion cable: cable that connects bus expander.

Field cabling: cables connecting the sensors, actuators and other process devices to the Ponto Series IO modules.

Field network cable: cable that connects the nodes in a field network, such as the Field Network Interface and the Field Network Head.

Field Network Interface: master module for the field networks, located in the local bus and performs the communication with the field network heads.

Field Network Head: slave module of a field network. It is responsible for the exchange of data between the modules and the field network master.

Local Bus: set of IO modules connected to a CPU.

Mechanic Switch Code: two digits defined by mechanical switches, programmable in the base and with objective of avoiding the connection of incompatible modules.

Rail: metallic element with normalized shape accordingly to the DIN50032 norm. It is also called TS35 rail.

Remote Bus: set of IO modules connected to a Field Network Head.

Network Glossary

Backoff: time that a node in a CSMA/CD network takes before transmitting data after a collision has occurred.

Baud rate: rate that the information bits are transmitted through a serial interface or communication network (measured in Bits/second)

Bridge: equipment to connect two communication networks with the same protocol.

Broadcast: simultaneous communication to all the nodes in a communication network.

CSMA/CD. Type of access to the physical media based on data collisions. It is used for Ethernet networks.

Communication network: set of equipment (nodes) interconnected by communication channels.

Deterministic communication network: communication network where the transmission and reception of information among the nodes is guaranteed to occur within a maximum established time period.

EIA RS-485: industrial standard (physical level) for data communication.

Frame: information until transmitted in the network.

Gateway: equipment to connect two communication networks with different protocols. The AL 2400/S-C or QK2400 gateways allows interconnection of ALNET I and ALNET II networks.

Media access: method used by all nodes in a network to synchronize data transmission and resolve possible conflicts in simultaneous transmissions.

Master: equipment connected to a communication network originating all the command requests to other network equipment.

Master-slave communication network: communication network where the data transfer are initiated only by one node (the network master). The remaining network nodes (slaves) only reply when requested.

Multicast: simultaneous communication with a group of nodes connected to a network.

Multi-master communication network: communication network where the data transfer are initiated by any node connected to the data bus.

Node: any station in a network with the capacity to communicate using a established network.

Peer to peer: type of communication where two partners exchange data without relying on the master.

Protocol: rules of procedures and formats that, under control signals, allow the establishment of data transmission and error recovery among equipment.

Serial Channel/Canal: equipment interface that transfer data in the serial mode.

Slave: equipment connected to a communication network that only transmits upon the master requests.

Sub network: segment of a communication network that connects a group of equipment (nodes) with the goal of isolating the local data traffic or utilizing different protocols or physical media.

Time-out: maximum preset time to a communication to take place. When exceeded then an error is generated.

Token: it is a mark that indicates who is the bus master in a moment.

General Glossary

Active CPU: in a redundant system is the CPU that is controlling the system – reading the inputs, executing the application program and activating the outputs.

Adjustment bridge: Switch for selection of addresses and configuration. It is composed by pins on the circuit board and one small removable connector used for a selection.

Algorithm: finite and well defined sequence of instructions with the goal to solve problems.

Altus Relay and Block Language: it is a set of rules, conventions and syntaxes utilized when building a application program to run in a PLC.

Application Program: it is the program uploaded into the PLC and has the instructions that define how the machinery of process will work.

Arrestor: lightning protection device using inert gases.

Bus: electrical signal set logically grouped with the goal of transferring information and control among several system elements.

Assembly language: microprocessor programming language, it is also known as machine language.

Backup CPU: in a redundant system, it is the CPU supervising the active CPU. Thus it is not controlling the system, but ready to take control when the main CPU fails.

Bit: information basic unit, it may be at 1 or 0 status.

Byte: information unit composed by eight bits.

Configuration Module (C Module): unique module in a remote application program that carries several needed parameters for its operation, such as the operands quantity and disposition of IO modules in the buses.

CPU: central processing unit. It controls the data flux, interprets and executes the program instructions as well as monitors the system devices.

Default: pre defined value for a variable. It is used when there is no definition.

Diagnostic: procedures to detect and isolate failures. Also it relates to the data set used for such tasks, and also serves for analysis and correction of problems.

Download: load of program of module configuration.

E2PROM: non volatile memory that may be erased by electricity.

Encoder: position measurement transducer.

EPROM (Erasable Programmable Read Only Memory): memory for read only, erasable and programmable. The memory doesn't lose its contents upon shutting its power off.

Execution Modules (E Modules): modules that have the application program. It may be one of the three types: E000, E001 and E018. The E000 module is executed just once upon system powering or when setting programming into execution mode. The E001 module has the main program that is executed cyclically, while the E018 module is activated by the time interruption.

Executive Program: it is the operating system of a PLC. It controls the PLC basic functions and executes the application programs.

Flash EPROM: non volatile memory that may be erased by electricity.

Function Module (F Module): PLC module called from the main module (M module) or from another module or procedure. It passes parameters and return values, and serves as a sub-routine.

Hardkey: connector normally attached to the parallel port of a microcomputer with the goal to protect illegal execution of a software.

Hardware: physical equipment used to process data where normally programs (software) are executed.

Hot swap: procedure of replacing modules in a system without shutting it down. It is normal procedure for IO modules.

IEC Pub. 144 (1963): norm for protection of accidental access to equipment, and sealing for water, dust and other foreign objects to the equipment.

IEC 1131: generic norm for operation and utilization of programmable controllers.

IEC-536-1976: norm for electrical shock protection

IEC-801-4: norm for tests of immunity against interference by pulses train

IEEE C37.90.1 (SWC- Surge Withstand Capability): norm for oscillatory wave noises protection.

Interface: device that adapts electrically or logically the transferring of signals between two equipment.

Interruption: priority event that temporarily halts the execution of a program. The interruptions are divided into two generic types: hardware and software. The former is caused by a signal coming from a periferic, while the later is caused within a program.

IO (input/output): input or output devices in a system. In the PLCs they are typically the digital or analog modules that monitor or actuate the devices controlled by the system.

IO Module: module belonging to the IO subsystem.

IO Subsystem: set of digital or analog IO modules of a PLC.

Kbytes: unit that assesses memory size. It represents 1024 bytes.

LED (Light Emitting Diode): type of semiconductor diode that emits light when energized. It's used for visual indication.

Logic: graphic matrix where are inserted the relay diagram language instructions that are part of an application program. A set of sequentially organized logics makes up a program module.

Menu: set of available options for a program, they may be selected by the user in order to activate or execute a specific task.

Module (hardware): basic element of a system and has very specific functionality. It's normally connected to the system by connectors and may be easily replaced.

Module (software): part of a program capable of performing a specific task. It may be executed independently or in conjunction of other modules through the passing of information and parameters.

Module address: address used by the CPU in order to access a specific IO module.

Nibble: information unit composed by four bits.

Non-operant CPU: CPU that is not in the active status (controlling the system) neither on the backup status (supervising the active CPU), thus not ready to control the system.

Octet: set of eight bits numbered from 0 to 7.

Operands: elements over which the instructions work. They may represent constants, variables or set of variables.

PC: Programmable Controller

Procedure Module (P Module): PLC module called from the main module (M module) or from another module or procedure and it does not pass parameters.

Programmable Controller: equipment that controls a industrial system based on a application program written in relay and blocks language. It is composed by a CPU, power supply and a structure of IOs.

Programming language: it is a set of rules, conventions and syntaxes utilized when building a program.

RAM (Random Access Memory): memory where all the addresses may be accessed directly and in a random order at the same speed. It is volatile, in other words, its content may be erased when the energy is shut down, unless there is a battery to keep its contents.

Redundant CPU: it is the other CPU in a redundant system. For instance, the redundant CPU of CPU2 is CPU1 and vice versa.

Redundant system: system that has backup or double elements to execute specific tasks. Such system may suffer failures without stopping the execution of its tasks.

Ripple: undulation present in continuous voltages.

Scanning cycle: a complex execution of the PLC application program.

Slot: device to plug in integrated circuits or other components, thus facilitating their substitution and maintenance.

Software: computer programs, procedures and rules related to the operation of a data processing system.

System Setup: procedure when the control system is finally tested. It consists of a through test when all the programs from remote stations and CPUs are put to work together.

Supervision Station: equipment connected to a PLC network with the goal of monitoring and controlling the process variables.

Tag: name associated to a operand or to a logic that identifies its content.

Toggle: element with two stable states that are switchable at each activation.

Upload: program reading or module configuration.

Varistor: protection device against voltage spikes.

Word: information unit composed by sixteen bits.

Watchdog timer: electronic circuit that checks the equipment operation integrity.

Acronyms

BAT - battery

BT – battery test

CPU – central processing unit

DP: Decentralized Periphery

EEPROM - Electric Erasable Programmable Read Only Memory

EMI: Electromagnetic Interference.

EPROM: Erasable Programmable Read Only Memory

ER - error

ESD: Electrostatic Discharge.

EX - execution

E2PROM: Electric Erasable Programmable Read Only Memory

IO – inputs and outputs

FC: Forcing

Flash EPROM: Flash Erase Programmable Read Only Memory

FMS: Fieldbus Message System

INTERF: Interface

ISOL: Isolation

LED –light emitting diode

Max: maximum

Min: minimum

Obs: notes

PAs – adjustment jumps

PA: Process Automation

PG - programming

PID – proportional, integrated and derivate control

RAM - random access memory

ref: reference

RX – serial receiving

SELEC: selectable

TC – Technical Characteristics

TX – serial transmitting

UTIL: utilization

WD - watchdog timer