Ladder Programming Manual
MasterTool Extended Edition
MT8000

Rev. G 08/2010
Cód. Doc.: MP399103

altus

According to current legislation in Brazil, the Consumer Defense Code, we are giving the following information to clients who use our products, regarding personal safety and premises.

The industrial automation equipment, manufactured by Altus, is strong and reliable due to the stringent quality control it is subjected to. However, any electronic industrial control equipment (programmable controllers, numerical commands, etc.) can damage machines or processes controlled by them when there are defective components and/or when a programming or installation error occurs. 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 safety reasons. This concern is higher when in initial commissioning and testing.

The equipment manufactured by Altus does not directly expose the environment to hazards, since they do not issue any kind of pollutant during their use. However, concerning the disposal of equipment, it is important to point out that built-in electronics may contain materials which are harmful to nature when improperly discarded. Therefore, it is recommended that whenever discarding this type of product, it should be forwarded to recycling plants, which guarantee proper waste management.

It is essential to read and understand the product documentation, such as manuals and technical characteristics before its installation or use.

The examples and figures presented in this document are solely for illustrative purposes. Due to possible upgrades and improvements that the products may present, Altus assumes no responsibility for the use of these examples and figures in real applications. They should only be used to assist user trainings and improve experience with the products and their features.

Altus warrants its equipment as described in General Conditions of Supply, attached to the commercial proposals.

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

Altus does not acknowledge any other guarantee, directly or implied, mainly when end customers are dealing with third-party suppliers.

The requests for additional information about the supply, equipment features and/or any other Altus services must be made in writing form. Altus is not responsible for supplying information about its equipment without formal request.

COPYRIGHTS

Summary

1. Introduction

Software MasterTool Extended Edition

Software MasterTool Extended Edition MT8000, or simply MasterTool XE, is the tool for configuration and programming of ALTUS equipments (Grano Series, Ponto Series, PX Series and AL-2000 series), including PLCs and remotes. This tool also monitors processes, module configuration and generation of reports. MT8000 runs on Windows® 2000, Windows® XP and Windows® 7 operatinig systems (all 32bits)... English and Portuguese versions of the manuals and software are available.

MasterTool Extended Edition allows the development of application for all ALTUS PLCs series. The edition of the program uses the concept of symbolic programming (tags or nicknames), what permits registers of the project during the edition of the modules. The concept of the project, that establishes a relation among several files, eases the work, reducing the developing time, besides preventing the user from committing common configuration errors, through verification.

MasterTool ProPonto MT6000

Ponto's PLCs series were included in MasterTool version 3.00. And the software MasterTool ProPonto MT6000 is necessary for its programming. This software, named ProPonto in this document, is in MasterTool XE CDROM, in a sub-directory with the same name.

WARNING: ProPonto is necessary only to the configuration of PLCs series Ponto. Further information about ProPonto can be obtained in ProPonto's Manual, which can be found in the PDF format in the directory ProPonto/Manual do CD-ROM.

Lite, Professional and Advanced

MasterTool Extended Edition MT8000 software has three distribution versions, each of those with specific characteristics according to necessity. They are:

- **Lite**: programming software specific for small applications
- **Professional**: programming software that contains the necessary tools for all Altus PLCs' lines
- **Advanced**: programming software with tools for bigger applications

Each version has characteristics, ends and functions specific for each purpose. Details about the differences among them can be seen in MasterTool Extended Edition manual.

Documents Related to this Manual

To obtain additional information about MasterTool Extended Edition other documents can be consulted (manuals and technical characteristics). Those documents can be found on http://www.altus.com.br.

We suggest the following documents for additional information:

- MT8000 Technical Characteristics
- MasterTool Extended Edition's Manual
- MasterTool Extended Edition's ST Programming Manual
- MasterTool ProPonto - MT6000 Manual

Visual Inspection

Before proceed the installation, it is recommendable to make a careful visual inspection of the material, verifying if it does not have damages caused for the transport. Verities if all the CD-ROMs are in perfect state. In case of defects, informs the company or the nearer Altus deliver.

It is important to register the serial number of each equipment received, as well the revisions of software, in case it exists. This information will be necessary in case of need to contact the Altus technician support.

Technical Support

For contacting Altus Technical Support in São Leopoldo, RS, call +55-51-3589-9500. For other Technical Support centers, check our website (http://www.altus.com.br) or send e-mail to altus@altus.com.br.

In case of the equipment being already installed, provide the following information upon contacting us:

- MasterTool Extended Edition's software version
- Key version of MasterTool Extended Edition software
- Equipment's revision and the version of the executive software (on the side label on the equipment), when the support refers to dispositive communication
- The contents of the applicative program (modules)
- Windows operating system version, as well as the "service pack", of the computer where the software is being executed

Warning Messages used in this Manual

In this manual, warning messages will have those formats and meanings:

DANGER: indicates a risk to life, production, serious harm to people, or that substantial material or environmental damage may happen it the necessary precautions are not taken

WARNING:
Indicates configuration, application and installation details that *must* be followed to avoid situations that can cause system errors and related consequences.

ATTENTION:
Indicates important configuration, application or installation details to obtain the maximum performance of the system.

2. 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 would be a more adequate solution for the development of PLCs applications programs.

Because of that, the available instructions for construction of the applications in MasterTool XE 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 learning, since it is very much like conventional electrical outlines.

The accompaniment and verification of the functioning of an application program is similar to the electrical outline, with the advantage of visualizing the status of the contacts and bobbins in the MasterTool XE window.

This chapter describes the language of Altus' relays and blocks, detailing the language elements, module structures of an application program and the function of each module.

Upon the reading this chapter, it will be possible to structure an application program as well as do the configurations of PLCs.

Elements of Programming

An application program is made up of 4 basic elements:

- Modules
- Logics
- Instructions
- Operands

An applications program is composed by several 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 used on PLCs of the series AL-2000, PONTO, PX e GRANO allows up to eight elements in series and up to four ways in parallel.

The instructions are used to execute determined tasks through readings and/or alterations of the operands value.

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 is points of E/S and memory counters.

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

Logics

The word logic refers to a programming matrix made up of 32 cells (matrix elements) arranged in four lines (0 to 3) and 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 presented to the program simulates a short part of a real diagram of relays. The following figure shows the format of a logic of the applications program.

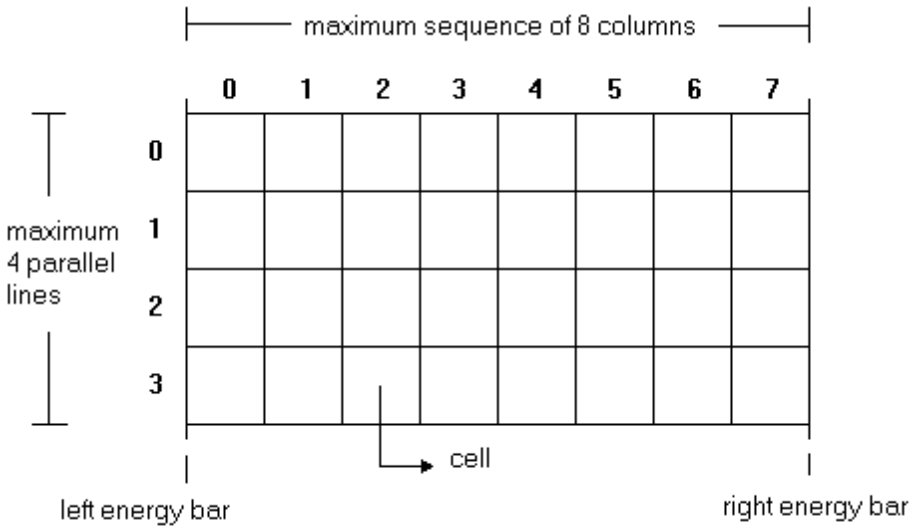


Figure 2-1. Logic format

The two lateral lines of the logic represent energy bars in which the instructions to be executed are placed.

Symbolic instructions, typically found in diagrams (contacts, coils, connections and instructions) are available for instructions in programming. They are represented in boxes, as timers, counters and arithmetics.

The logic should be programmed in a format which coils and inputs of boxes instructions 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 direction of the 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, as it is not necessary to be worried about the escape paths of current.

The processing of the instructions of a logic is carried out in columns, from column 0 to 7. A column is processed in the sequential order of its lines, from line 0 to line 3. The following figure shows the processing order of the logic cells. The existing number 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 XE 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 formatting the address 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 25 characters (alphanumeric), which can be attributed to any operand, except constants. This name represents the operand in the processes of programming, monitoring, purifying and documentation of an applications program.

ATTENTION: The MasterTool XE does not allow the use of Tags for operands of the constant type (%KM , %KD, %KF or %KI).

E.g.:

Attribution of the tag **CONT1** to the operand **%M0000**.

Whenever the operand **%M0000** needs to be used in the editing of the applications program, its tag **CONT1** can be used.

ATTENTION: The choice of the tag name for the operand should reflect at the most the function which the contents of the operand execute in the applications program. E.g.: **TANK 1**, stores the volume of tank 1.

ATTENTION: The identification of an operand through its address can always be done, as 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.

For further information about creating and attributing tags to operands, check MasterTool XE Manual.

ATTENTION: The operands can also be visualized through associated wire-info. However, an operand cannot be forced or monitored by typing the wire-info through its tag or address.

Operands Used on MasterTool XE

The operands available in MasterTool XE are shown in table bellow:

Type	Operand
%E	Input Relays
%S	Output Relays
%R	Bus Address
%A	Auxiliary Relays
%M	Memories
%D	Decimals
%F	Reals
%I	Integers
%KM	Memory Constants
%KD	Decimal Constants
%KF	Reals Constants
%KI	Integer Constants
%TM	Memory Tables
%TD	Decimal Tables
%TF	Real Tables
%TI	Integer Tables

Table 2-1. Operands used on MasterTool XE

The operands are divided into 3 groups:

- Simple operands
- Constant operands
- Table operands

Identification of Simple Operands

The simple operands are used as variables of storing the values in the applications programs. According to the instruction which they use, they can be referenced completely 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 bellow:

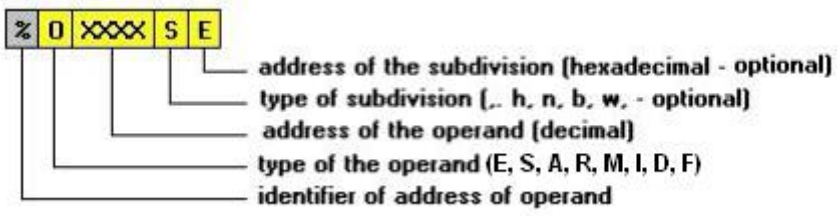


Figure 2-3. Simple operand format

Operand type:

- **%E** - input
- **%S** - output
- **%A** - auxiliary
- **%M** - memory
- **%I** - integer
- **%D** - decimal
- **%F** - real

Subdivision type:

- **.** - point of low 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

Identification of Constant Operands

The constant operands are used to define the fixed values during the editing of an applications program. Are possible constant operands memory, integer, decimal and real.

The general format of a constant operand can be seen in the following figure:

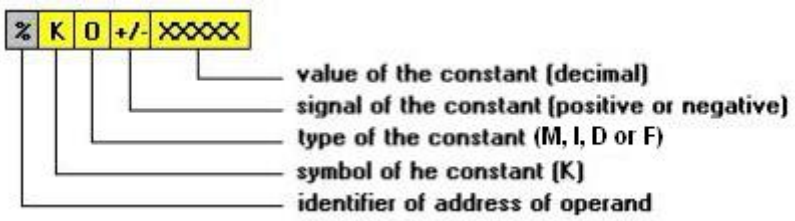


Figure 2-4. Constant operand format

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 dimensional arrays. Indices are used to determine the position of the table to be read or altered. Memory, integer, real or decimal operands tables are possible.

The general format of an operand table can be seen in the figure bellow:

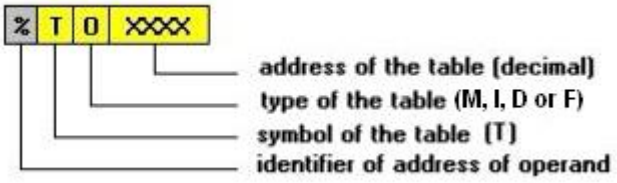


Figure 2-5. Table operands format

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

%E Operands - Input Relays

%E operands are used as reference points of digital modules of input. Their quantity is determined through the number of E/S modules which are arranged behind the scenes of the system.

The %E operands are normally used in movement and binary instructions (contacts, coils). They use 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 the figure bellow:

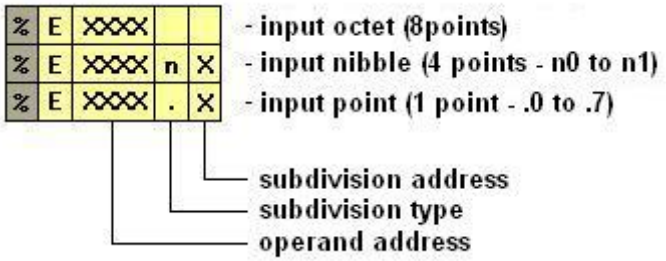


Figure 2-6. %E operands format

Examples:

- %E0018.6 - point 6 of the input octet 18
- %E0021n0 - nibble 0 of the input octet 21
- %E0025 - input octet 25

Limits:

- Minimum: 0
- Maximum: 255

%S Operands - Output Relays

%S operands are used to refer points of digital modules of output. Their quantity is determined through the number of E/S modules which are arranged behind the scenes in the system.

The %S operands are used in movement and binary instructions (contacts, coils). They use 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 the figure bellow:

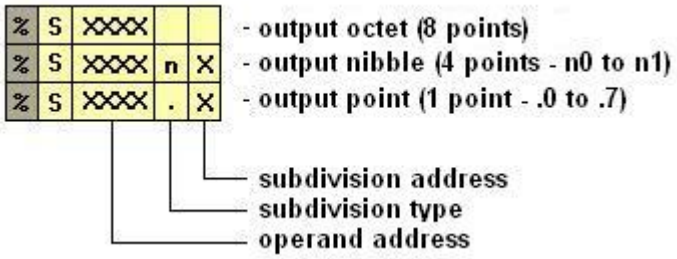


Figure 2-7. %S operands format

Examples:

- %S0011.2 - point 2 of the output octet 11
- %S0010n1 - nibble 1 of the output octet 10
- %S0015 - output octet 15

Limits:

- Minimum: 0
- Maximum: 255

%A Operands - 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 (check section **Declaration of Operands** in this same chapter).

%A operands are used in movement and binary instructions (contacts, coils). They use 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 %A operands can be seen in the figure bellow:

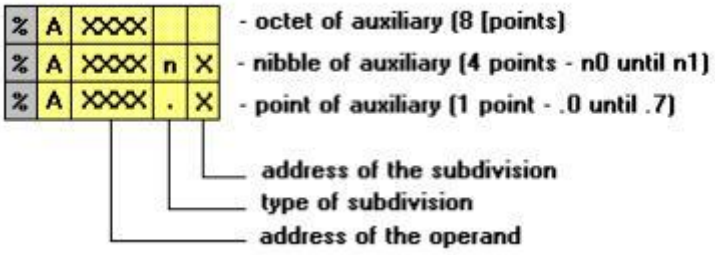


Figure 2-8. %A operands format

Examples:

- %A0032.7 - point 7 of the auxiliary output 32
- %A0087n1 - nibble 1 of the auxiliary output 87
- %A0024 - auxiliary octet 24

Limits:

- Minimum: 0
- Maximum: 255

%R Operands - Bus Addresses

Are operands used to refer points or octets in the input and output modules of the controller. Those operands represent only bus addresses, not storing values or using memory space. They are used in some instructions or functions that access modules.

The format of the %R operands can be seen in the figure bellow:

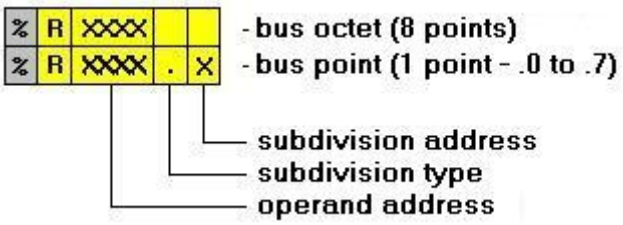


Figure 2-9. %R operands format

In the PLC AL-2004 bus 0 and 1, each bus position corresponds to 8 octets of %R operands. This way, in the 0 position there are the operands %R0000 a %R0007/; in 1 position, %R0008 a %R0015, and so on.

For obtaining the first operand of a bus position, it is necessary to calculate:

Octet's address = bus position x 8

In the AL-2004 buses from 2 to 9, each bus position corresponds to 2 octets of %R operands. So, in the 0 position there are the %R0000 e %R0001 operands; in the 1 position %R0002 e %R0003, and so on.

For obtaining the first operand of a given bus position, it is necessary to calculate:

Octet's address = bus position x 2

The addresses for each bus position are automatically showed at the declaration of the bus window, on the address column.

Examples:

- %R0026 - octet 26 of the bus
- %R0015.7 - point 7 of the octet 15 of the bus

Limits:

- Minimum: 0
- Maximum: 32767

%M Operands - Memories

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

The formats of the operands %M can be seen in the figure below:

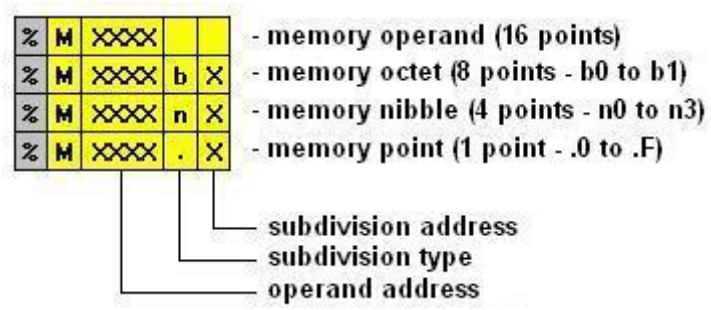


Figure 2-10. %M operands format

The amount of memory operands is configurable in the declaration of the configuration module, being the maximum limit depending on the PLC model in use (check section **Declaration of Operands** in this same chapter).

The operands %M are used in instructions of movement, comparison, arithmetic, counting, timing and conversion. They can be used in contacts, as the operands %E, %S and %A. These operands use two bytes of memory (16 bits) storing the value in two complement form (2') according to figure bellow.

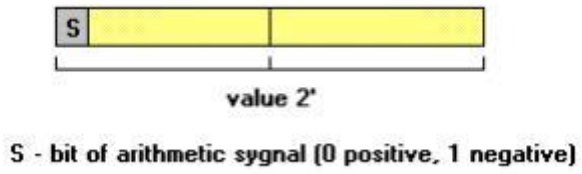


Figure 2-11. Memory operands format

Examples:

- %M0032 - memory 32
- %M0072n1 - nibble 1 of the memory 72
- %M0084.F - point 15 of the memory 84

Limits:

- Minimum: -32768
- Maximum: 32767

%D Operands - Decimals

The %D operands 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 below:

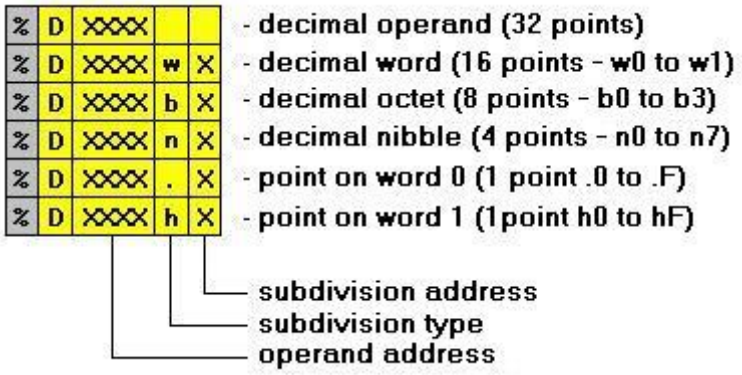


Figure 2-12. %D operands format

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

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

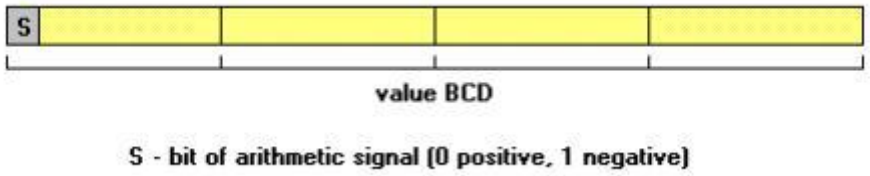


Figure 2-13. Decimal operand format

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

Limits:

- Minimum: -9999999
- Maximum: 9999999

%F Operands - Reals

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

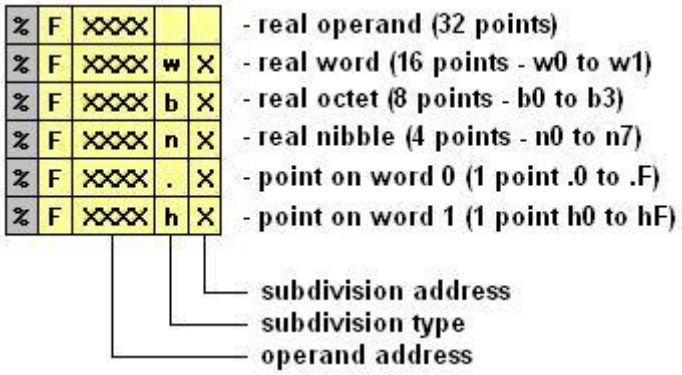


Figure 2-14. %F operands format

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

The %F operands 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 in the following figure:

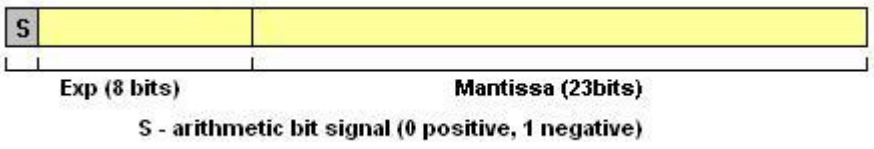


Figure 2-15. Real operand format

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

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

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

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

Example:

- %F0032 - real 32

Limits:

- Minimum: -3.4028235e+38
- Maximum: 3.4028235e+38

%I Operands - Integers

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 %I operands format can be seen in the following figure:

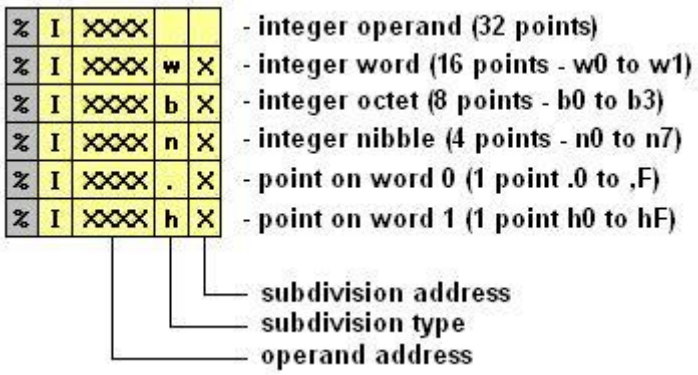


Figure 2-16. %I operands format

The quantity of integer operands can be configured on configuration module declaration, being the maximum limit depending on the PLC model that is in use (see the section **Declaration of Operands** in this same chapter).

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

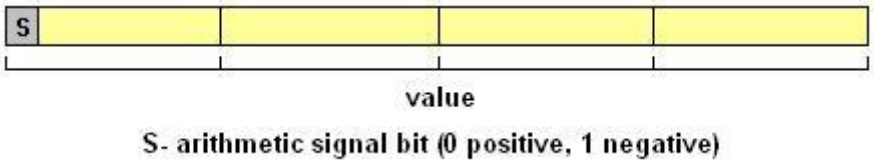


Figure 2-17. Integer operand format

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

Limits:

- Minimum: -2147483648
- Maximum: 2147483647

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

Operands are used to define the fixed values in the elaboration of the applications program. There 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:

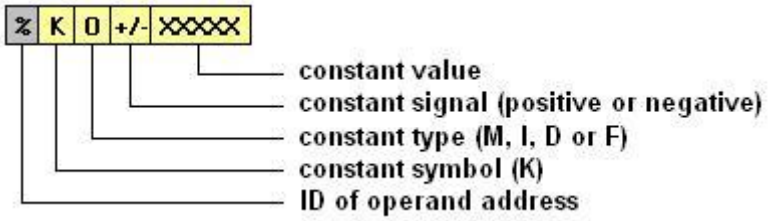


Figure 2-18. Constant operand format

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

Examples:

- %KM+00241 - memory constant + 241
- %KI+2000000000 - integer constant 2 billion or 2 x 10^9
- %KD-0019372 - decimal constant - 19.372
- %KF+0125.78 - real constant + 125.78
- %KF+3.1415E23 - real constant 3.1415 x 10^{23}

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

Tables of operands are grouped with simple operands, made up of one-dimensional arrays in order to store numerical values. Each table has a number of configurable positions, and each position can have 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:

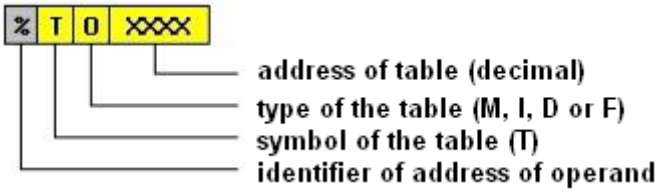


Figure 2-19. Table operands format

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

The tables are used in instructions of movement.

Limits:

- Minimum : 0
- Maximum: 254

Indirect Access

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

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

The format of indirect access can be seen in the following figure:

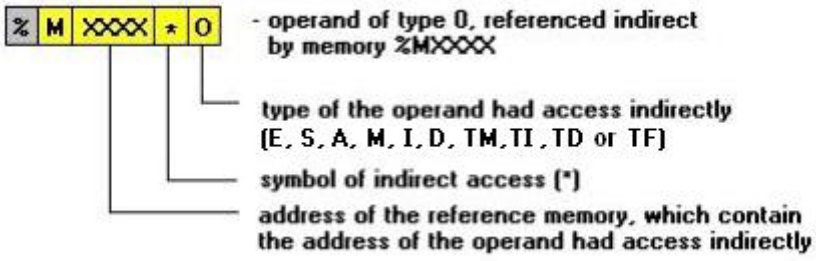


Figure 2-20. Indirect access format

In MasterTool XE, 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 will 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 the existing addresses for the 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.

The %R operands do not occupy memory space, being just addresses to the access to the buses.

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

The amount of operands %M, %I, %D, %F, %TM, %TI, %TD and %TF can be declared, occupying an area of RAM memory proper of the PLC being used. The following table shows the maximum memory capacity for the stocking of those operands in each controller. The %E, %S and %A operands do not occupy this area.

The declaration of the operands is carried out through the editing window of the configuration module of MasterTool XE, being stored in the configuration module. The number of operands declared should be tailored to the maximum capacity of the available memory. Check items Configuring Simple Operands, Configuring Table Operands and Configuring Retentive Operands on MasterTool XE Manual.

ATTENTION: A minimal quantity of memory operands (%M) able to have the diagnostic bytes used on the bus modules should be declared

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

The operands %TM, %TI, %TF and %TD are declared informing the number of necessary tables 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.

The following table shows the memory space used for each type of operands and where its values are stored.

Operand	Memory	Localization
%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 operands
%I – integer	4 bytes	RAM operands
%D – decimal	4 bytes	RAM operands
%F – real	4 bytes	RAM operands
%TM – table M	2 bytes per position	RAM operands
%TI – table I	4 bytes per position	RAM operands
%TD – table D	4 bytes per position	RAM operands
%TF – table F	4 bytes per position	RAM operands

Table 2-2. Memory and localization of operands

Retentive Operands

Retentive Operands are operands which have their values preserved when the PLC is turned OFF (disconnected). The operands non 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 input operands (%E) are retentives. When these operands are associated a local bus module, their values will be updated before the first execution cycle. If these operands is associated a remote bus (Profibus, MODBUS, etc) there is a latency time until their update.

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 bytes for numeric operands. The declaration of the operands %S and %A is carried out from 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, then the operands %M0384 to %M0511 will be considered retentive.

Instructions

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

The programming of this language, carried out through. MasterTool XE, uses a group of powerful instructions.

MasterTool XE instructions can be divided into 7 groups:

RELAYS containing the instructions:

- **RNA** contact normally open
- **RNF** contact normally closed
- **BOB** simple coil
- **BBL** coil connected
- **BBD** coil disconnected
- **SLT** jump coil
- **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
- **MES** moving inputs or outputs
- **CES** conversion of inputs or outputs
- **AES** updating inputs or outpus
- **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

CONVERSERS containing the instructions:

- **B/D** binary- decimal conversion
- **D/B** decimal - binary conversion
- **A/D** analogic - digital conversion
- **D/A** digital - analogic conversion

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
- **ECR** writing of operands on another PLC
- **LTR** reading of operands from another PLC
- **LAI** frees images update of operands
- **ECH** writing of operands on another PLC
- **LTH** reading of operands on another PLC
- **LAH** frees images update on operands

CONNECTIONS containing the instructions:

- **LGH** horizontal connection
- **LGV** vertical connection
- **NEG** denied connection

Restrictions related to the Placement of the Instructions

There are rules related to the placement of the instructions on the 8 logic columns that must be followed. Those instructions can be divided in 3 categories:

Instructions that can be edited only on column 7:

- **BOB** simple coil
- **BBL** turn on coil
- **BBD** turn of coil
- **SLT** jump coil
- **RM** master relay
- **FRM** end of master relay

Instructions that can be edicted from column 0 to 6:

- **RNA** relay usually open
- **RNF** relay usually closed
- **PLS** pulse relay
- **LGH** horizontal connection
- **LGV** vertical connection
- **NEG** denied connection
- **DIV** division
- **MOB** movement of operand blocks
- **>** greater
- **<** lesser
- **=** equal
- **SEQ** sequencer
- **CHF** call function mode
- **ECR** writing of operands in other PLC
- **LTR** reading of operands in other PLC

Instructions that can be edited in all the columns:

- **MOV** moving of simple operands
- **MOP** moving of part of operands
- **MOT** moving of table operands
- **MES** moving of inputs or outputs
- **CES** conversion of inputs or outputs
- **AES** updating of inputs or outputs
- **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** bidirectional counter
- **TEE** timer to turn on
- **TED** timer to turn off
- **B/D** binary- decimal conversion
- **D/B** decimal - binary conversion
- **CAR** load operand
- **LDI** connect or disconnect indexed points
- **TEI** test the status of indexed points

- **CHP** call procedure module
- **LAI** frees images update of operands
- **A/D** analogic - digital conversion
- **D/A** digital - analogic conversion

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 an 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, and each file contains a module, named as shown in the following figure:

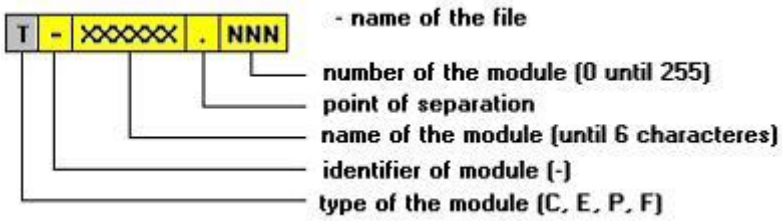


Figure 2-21. Format of the file module names

In some places in this manual and in the **Help** section, 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 XE.

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

There are 4 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 uses 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 XE (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 200 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 processing 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 P199).
- **Module F** (Function): there can be up to 229 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 used again in various different applications programs. They are similar to instructions, being able to be called for modules of execution, procedure or function. (F000 to F228).

Module C - Configuration

Module C contains the configuration parameters of the PLCs. Its creation is a pre-requisite for editing other modules of the MasterTool XE programming project. 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, check MasterTool XE Manual.

There is only one module C per programming project, having as its number 000.

Contents of a module C:

- **Declaration of the Bus of E/S modules:** specifies the configuration of the E/S modules to be used in the programmable controller, indicating the distribution of these modules and special modules in the PLCs bus. The declaration of the modules defines, in this way, the number of points and the E/S 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, check MasterTool XE Manual.
- **Declaration of Operands:** specifies the number of simple operands and tables of operands which will be 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), Decimal (%D), Integer (%I), and Real (%F). It takes place in the editing window of module C. For more information regarding how to declare simple operands, check MasterTool XE Manual.
    - **Declaration of Table Operands:** allows the definition of the number of tables of Memory operands (%TM), Integer operands (%TI), Real operands (%TF) and Decimal operands (%TD) and of the number of positions in each one. A table shows a group of operands, being defined in the editing window of Module C. For further information about how to configure table operands, check MasterTool XE 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 PLC 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, check MasterTool XE Manual.
- **Declaration of the General Parameters of the PLC**: are the generic parameters necessary for the functioning of the programmable controller, such as the type of PLC in which the applications program will be loaded, the period of calling of the modules activated 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, check MasterTool XE Manual.
- **Declaration of the Parameters of the ALNET I Network**: specifies the several parameters necessary for the functioning of communication in ALNET I Network. These parameters are configured in the editing window of Module C. For further information regarding how to configure parameters of ALNET I, check MasterTool XE Manual.
- **Declaration of the Parameters of the ALNET II Network**: specifies the various parameters necessary for the functioning of communication in ALNET II network, 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, check MasterTool XE Manual.

- **Declaration of the Parameters of the Ethernet Network**: specifies the various parameters necessary for the functioning of communication in Ethernet network, 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, check MasterTool XE Manual.
- **Declaration of the Parameters of the Synchronism Network**: specifies the various parameters necessary for the functioning of communication in synchronism network, 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, check MasterTool XE Manual.

Extended Module C – Configuration

These 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. It is due to the fact that the amount of this type of module varies according to each application, may not have any to be up 7 modules (C003 to C009).

For further information see MasterTool XE Manual.

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 E:

- **E000 - Initialization Module:** is executed once, when the PLC is turned on or in the passage of programming for execution mode 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 of the programmed 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 had been interrupted. The time continues to be counted during the call of Module E018, its execution having to be as short as possible so that to avoid an excessive increase in the time of Module E001 cycle.

ATTENTION: The E-*.018 execution time cannot be equal or higher than the time period of the call. If this happen, the PLC get in error mode, showing the message “Reentrance in E-*.018 module”, in information window (command menu Communication / State / Information).

ATTENTION: The E-*.018 Time Interrupt just will be execute for first time after finished the module E-*.000.

- **E020 – External Interrupt Module:** When occur a rise edge signal in input source, the main program is interrupted and the E-*.020 is executed. When this module finished, the execution will continue in the point where was stop. If the interrupt source input is active with many frequency, the time execution module must be as short as possible.

WARNING:
The execution time of Module E020 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 E020, in the window Information (command Communication, Status, Information)

ATTENTION:
The External Interrupt Module (E020) will be execute for first time after finished the module E-*.000.

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 200 (P000 to P199).

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

They can also be used for a better structure of the main program, dividing it into segments according to its function and declaring them in different Modules P. In this case, the continue execution module E001 only 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 application programs called from 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). The following figure represents the flow of data between instruction CHF and the function module.

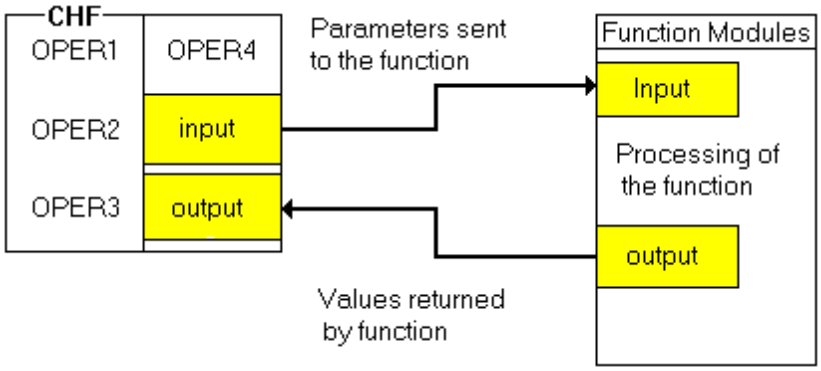


Figure 2-22. Parameters on F modules

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

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 where the programmable controller is, is indicated in the LEDs of the front panel of the PLC, and can also be consulted through MasterTool XE, through the dialogue box **Status** (**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 initializing, the command **Communication/Status/Programming**, or equivalent short cut in the tool bars can be activated, making the PLC to pass 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 PLCs watch dog circuit that is always connected, after initialization status. Executing 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 scanning 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 application program is not executed, when there is no reading of the input points, then the outputs are deactivated and the PLCs memory compacted. The PLC remains non-operating, waiting for commands from MasterTool XE. This mode is normally used to load programming project modules for MasterTool XE through the serial channel. At the passing for execution or cycling status starting from the programming status, the operands are zeroed.

Cycling Status: when in cycling mode, the programmable controller does not execute the module E001 cyclically, and waits for the commands from MasterTool XE. Each command **execute cycle**

activated in MasterTool (options **Communication/Status/Execute Cycle** starting from the main menu or equivalent shortcut) starts one single scan of the applications program (Module E001), the PLC remains waiting for a new command after executing the scan. When the PLC passes to cycled mode, the counting of time in the timer stops, being the same increments of one unit of time for each two scans executed. The calls to the module of external interruption E020 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. It is only possible to leave the error status 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 XE commands (options **Execution, Programming and Cycling** in the dialogue box **Status**, or their shortcut equivalents in the **Tool Bar)**. 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. The next figure shows the possibilities for changing status.

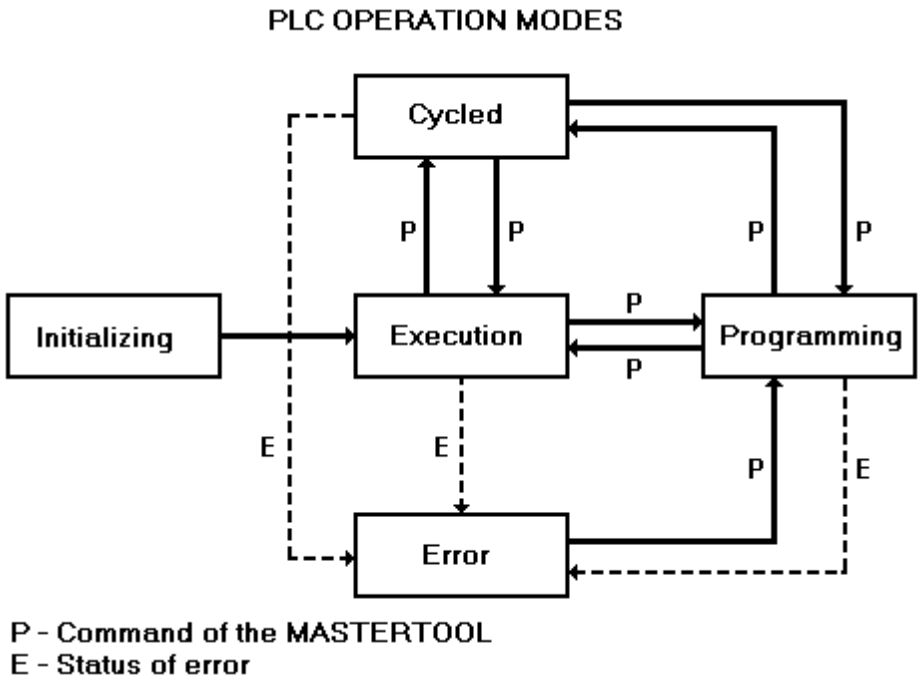


Figure 2-23. PLC operation status

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 is 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. The figure bellow shows the execution of the applications program in outline.

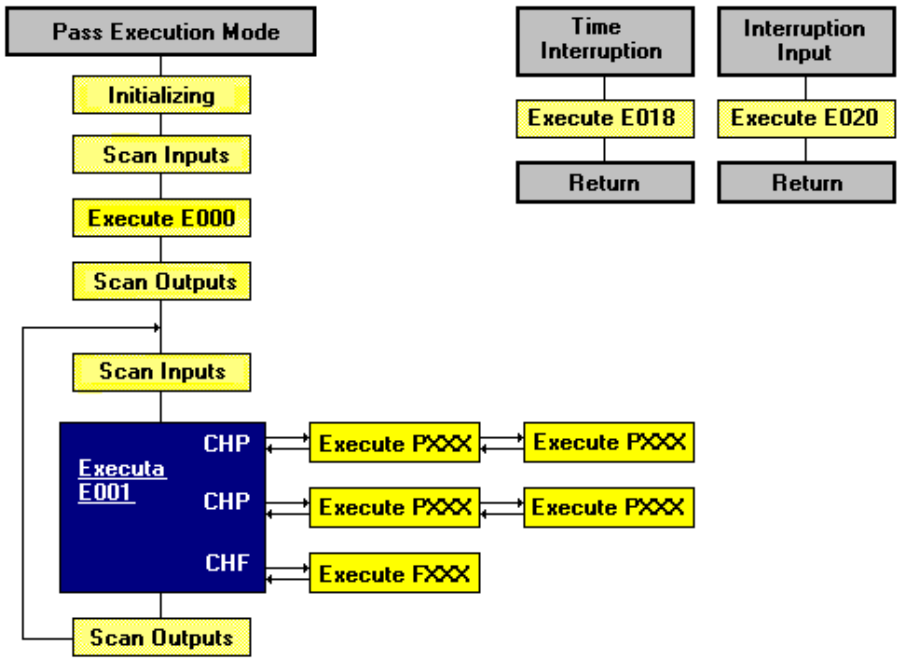


Figure 2-24. Execution of a programming project

ATTENTION: The modules E020 e E018 only will be executed after the execution of E000.

Elaboration of the Programming Project

General Considerations

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

The first step in the editing of a MasterTool XE 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, as in this module there are the declarations of the input and output modules 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 XE 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 XE allows various modules to be loaded and remain simultaneously in its memory.

After creating or reading a project, it can be edited, adding it to the existent modules, creating new modules for the project or excluding modules that are already in the project.

MasterTool XE allows several modes to be carried out and remain in its memory simultaneously.

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 must 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, any two modules can be inadvertently accessing 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, even operands used in the parameter passing. To guarantee its generic character, each Module F should use a different group of operands from the ones 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 management of modules calls, verifying the existence or not of the modules in the directory of the programmable controller, based on their types and numbers.

In the PLC AL 2004 there are 32 levels of calling, so, can be executed up to 32 consecutive callings of modules without finalizing anyone. Should be considered that the module E018 (if it exists) and the modules called also occupy call levels.

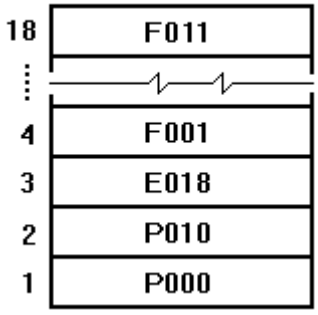


Figure 2-25. Maximum number of calling modules level

When the maximum number of accumulated calls without return is ultrapassed, 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** (**Communication/Status/Information** starting from the main menu), since these situations can cause processing errors according to the programmed logic.

It is possible the 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 occurring infinite loops of execution.

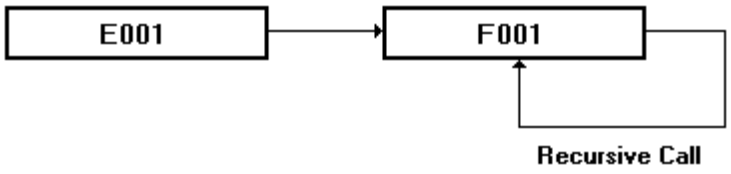


Figure 2-26. Recursive modules calling

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 cannot 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.

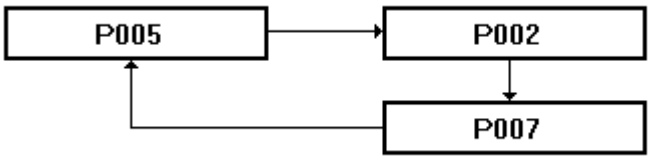


Figure 2-27. Module call loop

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 sensoring the end limit in systems of quick positioning. The instruction for updating the points of E/S (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 depending 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 PLCs processing.

Use of Module E020

When enabled, Module E020 is called for the execution, carrying out the necessary processing and updating of output points through the AES instruction.

Module E020 can also be used for security device or procedures, breaking devices or another applications that need quick acting.

If the module E020 is in the PLC, the PLC is called in every time the input is used. If the application program calls the module F-CONT. 005, it is do the reading and writing of the counting value, incrementing it in each acting of the input. If it is required, this input can be used with both functions, with the module F-CONT. 005 counting the number of times the module E020 was enabled.

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, there are only 5 ms for the execution of the main program before E018 will be called again. This situation considerably increases the time of the cycle of module E001.

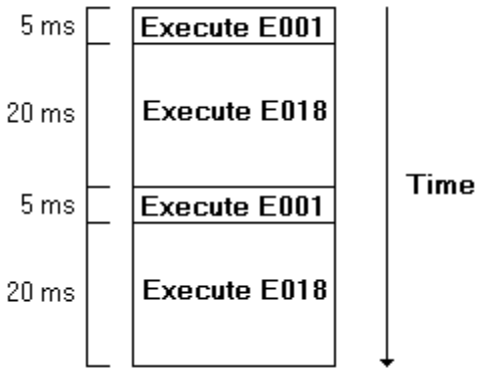


Figure 2-28. Care in using module E018

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** (**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, whatever is 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 observing the same rules of programming applying to module E018. The maximum number of levels of call from modules used in the module E018 must be added to the maximum level used in E001, and the sum must be less than the limit of the system (18 levels).

Care in the Programming of the Module E020

Some special care is necessary in programming module E020. Its processing time must be short, especially if the interruption input is enabled often, to avoid excessively risen of the total cycle time of the application program.

If the interruption input is enabled in a periodic manner in each 30ms, for example, and the execution time of E020 is 25ms, there will be only 5 ms for the execution of the main program before the calling of the module. This situation increases considerably the time of the module E001 cycle.

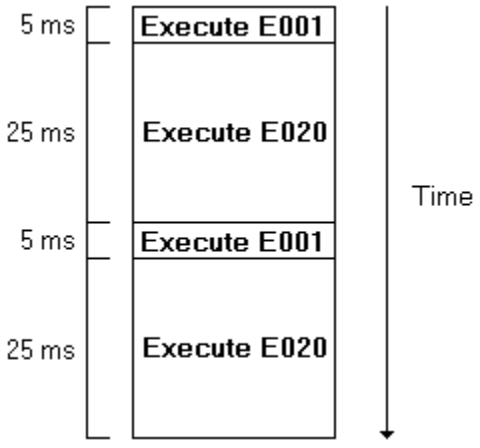


Figure 2-29. Care in using module E020

In case the module E020 is being executed and a new enable of the PLC input of interruption, this enable will not be considered, and the execution of the module will continue normally. This situation

does not provoke change for the error mode, and the PLC continues its normal execution. So, the PLC ignores enables of the quick interruption input that happen in times shorter than the E020 execution time.

The instructions keep the same behave when executed within the module E020, except in relation to some particular characteristics.

The calling of the module depends on the process that is being controlled, and does not happen in a periodic manner. This characteristic does not allow the use of timers on E020, it means, the instructions TEE and TED must not be used in the same. The pulse relay (PLS) enables its output during an execution of the module E020, and zeroes it in the next call. The instructions CHP and CHF can be used as in the main program and the enabled modules must observe the same programming rules of the module E020. The maximum number of module call levels used within a E020 module must be added to the maximum level used in E001 and E018, and this sum must be less than the system limit (18 levels).

Using of Operands in Programming Modules E018 and E020

Other care necessary is with the data sharing between the modules E018 or E020 and the other 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), also 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 or E020 any operand which is used in another programming project module, since this use, according to the case, can cause incorrect functioning. When the module e018 and E020 are used simultaneously, both must use exclusive operands.

In order to share the data between the Modules E018, E020 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 must be used in the modules of the normal execution cycle and not in modules E018 or E020.

ATTENTION: 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 must use only this last one. The MOV instruction should be in the main program, and not in the module E018.

ATTENTION: The contrary flow of data also demands the creation of image operands. If module E020 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 E020.

ATTENTION: A similar situation occurs for coil instructions. If some point of an operand is modified in the main program through a coil, it is not allowed to change any point of the whole octet of the same operand in Modules E018 or E020. 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 or E020 through a coil 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 or E020 through contacts.

Other care to be taken with respect to the updating of the inputs and outputs of Modules E018 or E020.

Preferably the inputs used in its processing should be only updated in these modules, using the instruction 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 or E020, 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 or memory operands in the beginning of the main program. It is also possible not updating input images in Modules E018 or E020 with the instruction AES, but only read directly the values of the E/S 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 or E020 (through the instruction F-AES) is possible, since the points pertaining to these octets are actioned through coils only in these modules.

In Modules E018 and E020, the values with the instruction MES in output modules declared in the bus through MasterTool XE should not be written, since the verification of output also carries out the updating of the values in these modules.

When a Module E018 or E020 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 or E020 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 and E020 continue being executed.

Simultaneous use of Modules E018 and E020

There are no priorities of execution related to the interruptions of the modules. It means that, if module E020 is being executed and the next time interruption occurs, the processing of E020 is interrupted, module E018 is executed, returning after to the execution of the E020.

In the same manner, if module E018 is being executed and the input interruption is enable, the E018 processing is interrupted, module E020 is executed, returning after to the execution of the E018.

Some care must be taken in the simultaneous use of both modules. The execution time of modules E 018 to E020 must be added, and this total must be less than the call period programmed to E018, avoiding mistakes of input in the execution of it.

The maximum number of call levels of the modules E001, E018 and E020 must also be added, and the result must be less or equals to the maximum allowed number (32 levels).

Both modules must use exclusive operands, observing the rules in the section **Using of Operands in Programming Modules E018 and E020**, in this same chapter.

Depuration of Programming Projects

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

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 XE:

- **PLC Model** - indicates the type of controller with which MasterTool XE 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 any other mode, a message indicates the existence of problems that do not cause the change to error mode (for example, the PC's battery is flat). Check MasterTool XE 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 FLASH EPROM or from FLASH EPROM 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. Check the 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 (**Communication/Status/Information**), directory of modules (**Communication/Modules**) 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. Whenever MasterTool XE is connected to any PLC, it regards the obtaining of this information as the first step to be taken.

Monitoring

Through MasterTool XE 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 further information about how to carry out the monitoring, c.f. items **Monitoring Simple Operands, Monitoring Table Operands** and **Monitoring Programs** on the MasterTool XE User's Manual.

ATTENTION: 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.

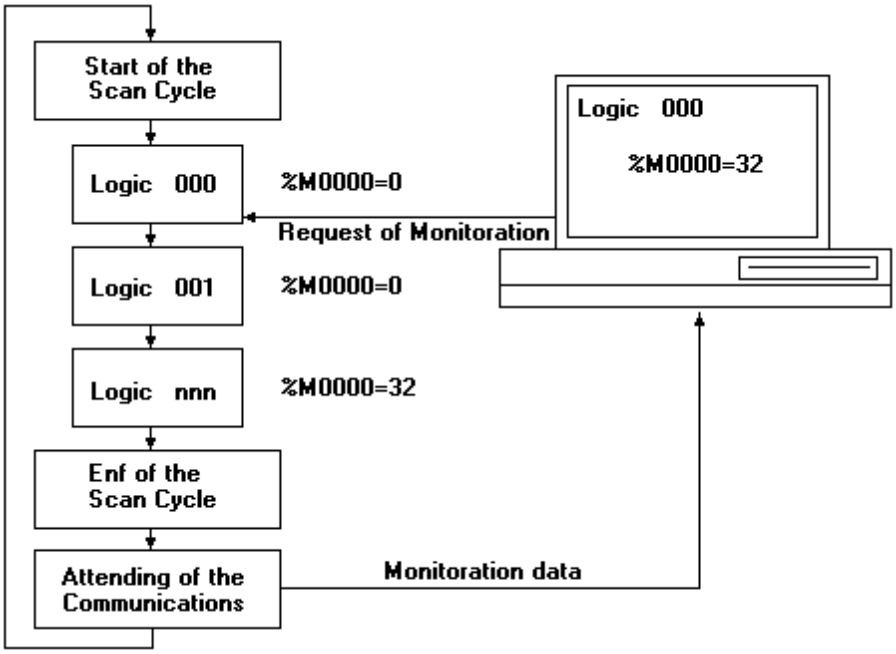


Figure 2-30. Incoherent situation on logics monitoring

Forcing

The values of the operands can also be forced with MasterTool XE, that is to say, the content of any programming project operand can be modified. The forcing of operands is permitted in any operating mode. The forcing of operands is permitted in any operating mode, except error mode.

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, the program which modifies it cannot have any instruction.

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 PLC 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 shows 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 programs.

WARNING:
Force operation does not actuate in %E or %S operands that has been updated with the instruction AES. This instruction read %E operands or write %S operands and it does not consider operands forcing effects. For this reason, it is not recommended forcing operands that have been updated by AES that are actives in the program.

Disabling the Outputs

For the "on Initialization" security when 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 checked in their values.

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

Program Modifications

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 even to reinitialize the control application program or to change the 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 is modified, it is advisable 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 be necessary the compaction of the RAM memory for reasons explained in the section **Managing Programming Project Modules**, in this chapter. This type of loading is only possible if there is enough free memory in the PLC for storing in the module to be sent.

At the end of the depuration of a program module, its transfer is suggested to the FLASH EPROM 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 is useful in the verification of the functioning of quick brakes in the applications program. The other depuration facilities 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 themselves, 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 PLC 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 XE 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.

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

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

ATTENTION:
A program module present in the PLC 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 PLC model used.

As the modules which make up the programming project are sent to the PLC through the serial channel, they occupy the first RAM 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 module, and this will be loaded in the following bank, if it 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 is named 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:

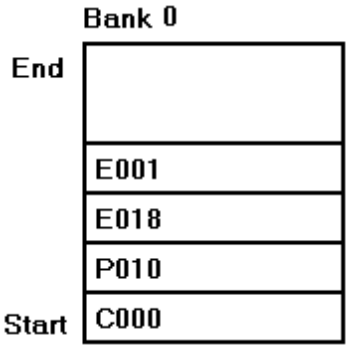


Figure 2-31. Compaction of RAM memory

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

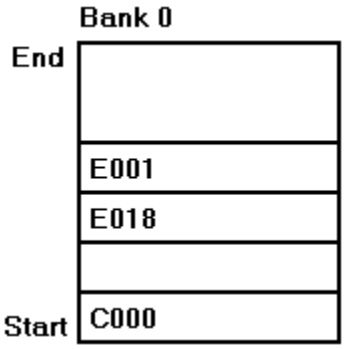


Figure 2-32. Compaction of RAM memory - 2

The space previously occupied by P010 is not taken advantage of by the programmable controller, since the loading of a new module is only possible after the last, module E001. After carrying out the compaction of the PLC's memory, bank 0 passes to the following configuration:

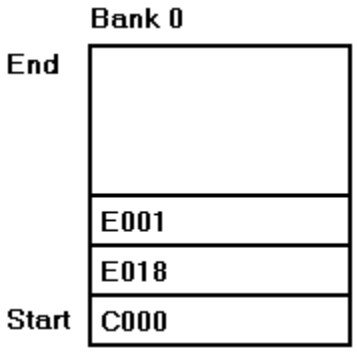


Figure 2-33. Compaction of RAM memory - 3

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, through MasterTool XE (**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 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.

ATTENTION:
FLASH compaction does not exist on MasterTool XE. The method to compact FLASH is to load the modules to the RAM, clean the FLASH and load the modules again to the FLASH.

Transference of Modules from RAM to FLASH

After being loaded into the RAM memory of the program, through the serial of the PLC, the programming project's modules can be transferred to FLASH EPROM. This command is only unable in PLCs which have FLASH memory.

It is possible to transfer one single module or a group of module seven 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 FLASH EPROM 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 for it, after the 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 FLASH to RAM:

The modules present in FLASH memory or in FLASH cartridge can be transferred to the RAM memory of the program.

It is possible to transfer one single module or a group of module, even with the PLC executing the program. The transferring into execution mode is partially carried out in each verification, being able to last 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 are modified.

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

Erasing Modules on FLASH

The erasing command can be used for modules stored in the FLASH memory of the PLC. As the erasing of FLASH'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 FLASH 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 FLASH, only the new one in RAM being shown in the directory.

Erasing the FLASH Memory

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

To erase the FLASH memory, use the options **Communications**, **Modules**, **Erase** FLASH when the PLC is in programming mode. The erasing can last several seconds, depending on the capacity of the FLASH used in the PLC.

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 PLCS information window (**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 XE. 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 XE 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.

Care should be take 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 its good functioning.

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.

ATTENTION:
Some typical procedures exist to reduce the execution time of very 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 (RM and FRM) 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.

ATTENTION:
The Initializations of values in operands or tables in Module E000 should be carried out, designed specially for this intention. The execution of module E000, for not to being 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

PLCs in AL 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 protection levels:

- **Level 0** – no protection
- **Level 1** – allows monitoring, writing, forcing operands and reading of the programming modules
- **Level 2** – allows monitoring, writing, forcing operands
- **Level 3** – only allows reading the PLC information

The change of protection level is carried out with the options **Communication/Status/Protection** in MasterTool XE, having to key in the password to achieve correct access. The PLC's protection level can be consulted with MasterTool XE 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 without 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 PLCs protection acts not only to carry out operations with MasterTool XE but also the commands received through ALNET I and ALNET II, with the same characteristics defined for each level.

Interlocking of Commands in the PLC

In the AL 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 FLASH 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, from 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, the PLC can receive one command to erase the FLASH memory while a module may be being transferred to the same memory.

To solve 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. 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 XE. While any of the signals are enabled, the LED FC of the panel in the PLC remains alight.

Operation on PLC	Blocked Command (ALNET I, ALNETII)	Enable Signal
Modules Uploading	Modules Uploading Transference from EPROM to RAM Transference from RAM to FLASH Asking Modles Uploading Erasing FLASH Compacting	CM
Transference from EPROM to RAM	Modules Uploading Transference from EPROM to RAM Transference from RAM to FLASH Asking Modles Uploading Erasing FLASH Compacting	CM
Transference from RAM to FLASH	Modules Uploading Transference from EPROM to RAM Transference from RAM to FLASH Asking Modles Uploading Erasing FLASH Compacting	CM
Erasing FLASH	Modules Uploading Transference from EPROM to RAM Transference from RAM to FLASH Asking Modles Uploading Erasing FLASH Compacting	CM
Legend: CM – Uploading Module		

Table 2-3. Braking of commands in the PLC (Uploading module)

Operation on PLC	Blocked Command (ALNET I, ALNETII)	Enable Signal
Modules Uploading	Modules Uploading Transference from EPROM to RAM Transference from RAM to FLASH Asking Modles Uploading Erasing FLASH Compacting	CR
Legend: CR – Compacting RAM		

Table 2-4. 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, transfers from RAM to FLASH, requesting to load modules, re-enabling of modules in EPROM, erasing of FLASH EPROM and compaction not be possible to execute, 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.

3. Instructions References

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.

Instructions List

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 XE, uses a group of powerful instructions shown in the following sections.

MasterTool XE 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 follow this routine.

1. The group is described with a little containing the name of the group.
2. Straight after the little, a brief description of the common characteristics of the group is given.
3. Finishing the presentation of the group, a table is displayed containing the name at 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

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 for divert the flow and control of the processing of the applications program.

Name	Name Description	Edition Sequence
RNA	Normaly Opened Contact	Alt, I, R, A
RNF	Normaly Closed Contact	Alt, I, R, F
BOB	Simple Coil	Alt, I, R, B
SLT	Jump Coil	Alt, I, R, S
BBL	Turn On Coil	Alt, I, R, L
BBD	Turn Off Coil	Alt, I, R, D
PLS	Pulse Relay	Alt, I, R, P
FRM	End of Master Relay	Alt, I, R, M
RM	Master Relay	Alt, I, R, R

Table 3-1. Instructions relays group

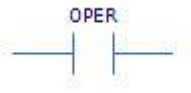
Presentation of the Instructions

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.

Contacts

RNA - Contact Normaly Open



RNF - Contact Normaly Closed

Description:

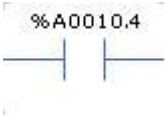
These instructions reflect, logically, the real behavior of an electrical contact of a relay in the applications program.

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.

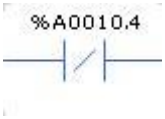
The contact normally open has a behavior opposite to normally open. If the point of the associated operand is in the logic status **1** or **0**, the contact normally closed is opened or closed respectively.

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.

Example:



In the case above, of a contact normally opened, the contact will be closed only if the operand status %A0010.4 is **1**, or it will remain opened.



In this case, the contact is normally closed, and it will be opened if the operand status is %A0010.4 is **1,** or it will remain closed.

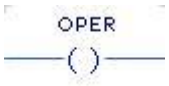
Syntax:

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

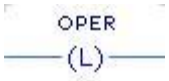
Table 3-2. Sintax of the RNA and RNF instructions

Coils

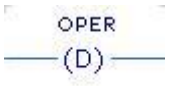
BOB - Simple Coil



BBL - Turn On Coil



BBD - Turn Off Coil



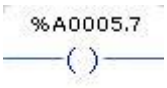
Description:

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.

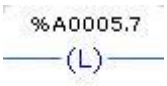
The simple coils connect or disconnect the operand point according to the enabling line, while the of type connected and of type disconnected only connect or disconnect. Operands when the line is powered ("set/reset").

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

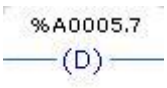
Example:



In the case above, while the coil is powered in %A005.7 it will have the logic value **1**. If not, its value will be **0**.



In this case, at the moment the coil is powered, the operand %A0005.7 will have logic value **1.** It will remain in this status even if the connection coil is not powered.



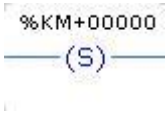
In this coil, at the moment the coil is powered, the operand %A0005.7 will have logic value **0.** It will remains in this status even if the disconnection coil is not powered.

Syntax:

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

Table 3-3. Sintax of the BOB, BBL and BBD instructions

SLT - Jump Coil



Description:

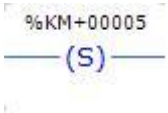
The instruction jump coil serves as a controller of execution sequence of an applications program, being used to divert its processing to a determined logic.

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 1, the execution of the applications program is diverted to logic 6 if the enabling line is powered. If the coil is not powered, the processing continues normally, that is, the next logic to be executed is logic 2.



It can be used 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.

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

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

ATTENTION:
If use a jump with a negative operand after a RLM instruction, the bar will be off to execute the instructions.

Syntax:

OPER1
%KM +XXXXX
%KM-XXXXX

Table 3-4. Sintax of the SLT instruction

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.

Example:

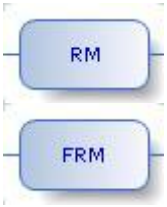
In this case, when the relay input is energized, the relay output stay energized in all program cycle. In operand %A0000.0 is stored the relay state.

Syntax:

OPER1
%AXXXX.X

Table 3-5. Syntax of the PLS instruction

RM, FRM - Master Relay, 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 then 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.

Example:

In this example, when the contact normally open, with operand %A0000.0 have 0 (open contact) the RM instruction input is not energized, disabling the next logic source bar (Logic: 001) until the logic has a FRM instruction (Logic:002).

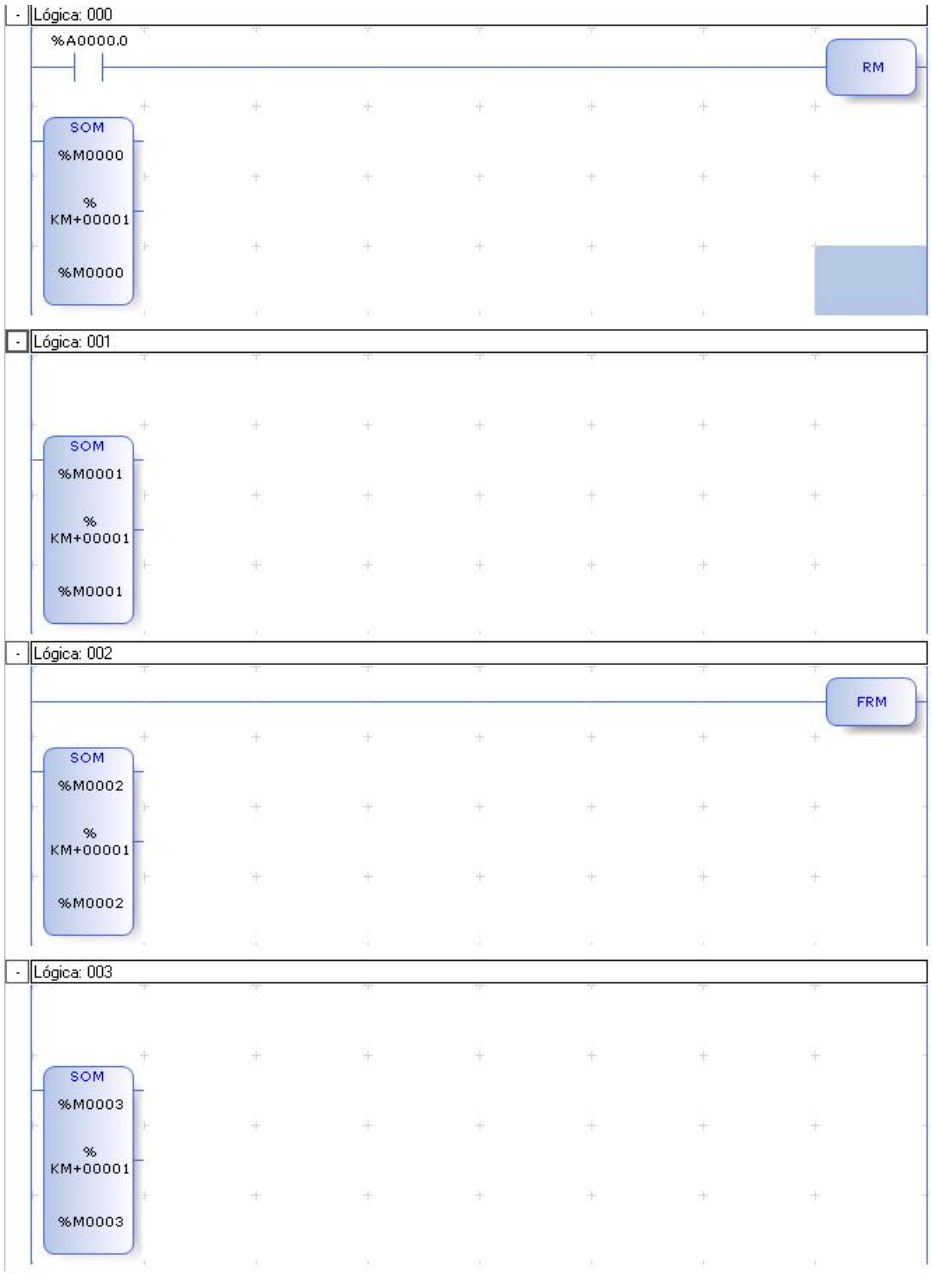


Figure 3-1. Example of use of the FRM instruction

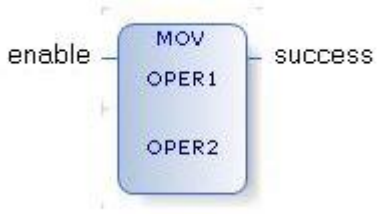
Instructions of Moving Group

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

Name	Name Description	Edition Sequence
MOV	Moving simple operands	Alt, I, M, V
MOP	Moving of parts (Subdivisions) of Operands	Alt, I, M, P
MOB	Moving of Blocks of Operands	Alt, I, M, B
MOT	Moving of Tables	Alt, I, M, T
MES	Moving of Inputs/Outputs	Alt, I, M, E
AES	Updates of Inputs/Outputs	Alt, I, M, A
CES	Conversion of Inputs/Outputs	Alt, I, M, S
CAB	Load Block	Alt, I, M, C

Table 3-6. Instructions of moving group

MOV - Moving Simple Operands



OPER1 - source operand
OPER2 - target operand

Description:

This instruction moves the contents of simple 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.

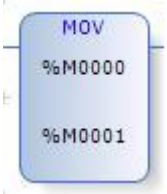
The moving of subdivisions of operands is not permitted. For this reason, the instruction MOP should be used.

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.

ATTENTION If OPER1 will be of bigger precision that OPER2 then is preserved the OPER2 precision.

Example:

```
In the instruction bellow, the operand value %M0000 is moved to operand %N1000
as the input is powered. That is, if the operand %M0000 had the value 5 and
%M10000 value 3 initially. After powering of the input, the value %M0000 and
```




```
%M10000 will be 5 and 5, respectively.
```

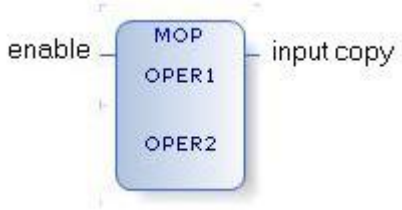
Syntax:

OPER1	OPER2
%E	
%S	%M
%A	%E
%M	%S
%D	%A
%M*E	%D
%M*S	%M*E
%M*A	%M*A
%M*M	%M*M
%M*D	%M*D
%KM	%M*S
%KD	

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

Table 3-7. Syntax of MOV instruction

MOP - Moving of parts (Subdivisions) of Operands



OPER1 - source operand
OPER2 - target 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 designation 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 designation 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.

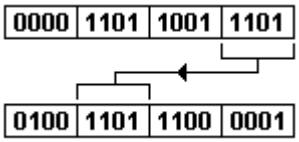


Figura 3-2. Example of the MOP instruction

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 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
%EXXXX.X	
%SXXXX.X	%EXXXX.X
%AXXXX.X	%SXXXX.X
%MXXXX.X	%AXXXX.X
%DXXXX.X	%MXXXX.X
%DXXXXhX	%DXXXX.X
%FXXXX.X	%DXXXXhX
%FXXXXhX	%FXXXX.X
%IXXXX.X	%FXXXXhX
%IXXXXhX	%IXXXX.X
%KMXXXXX	%IXXXXhX
%KDXXXXX	

OPER1	OPER2
%MXXXXbX	
%DXXXXbX	
%FXXXXbX	
%IXXXXbX	%MXXXXbX
%EXXXX	%DXXXXbX
%SXXXX	%FXXXXbX
%AXXXX	%IXXXXbX
%KMXXXXX	
%KDXXXXX	

OPER1	OPER2
%EXXXXnX	
%SXXXXnX	%EXXXXnX
%AXXXXnX	%SXXXXnX
%MXXXXnX	%AXXXXnX
%DXXXXnX	%MXXXXnX
%FXXXXnX	%DXXXXnX
%IXXXXnX	%FXXXXnX
%KMXXXXX	%IXXXXnX
%KDXXXXX	

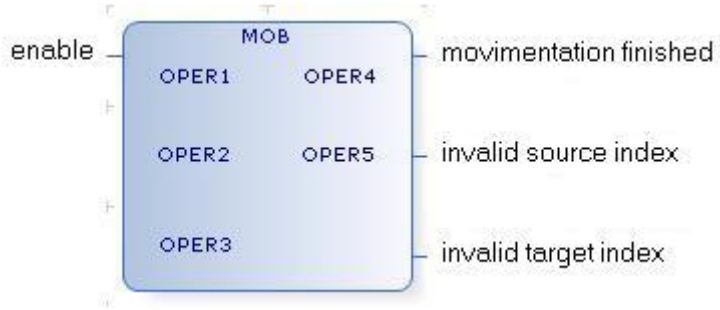
OPER1	OPER2
%MXXXXbX	
%DXXXXbX	%EXXXX
%FXXXXbX	%SXXXX
%IXXXXbX	%AXXXX

OPER1	OPER2
%DXXXXwX	
%FXXXXwX	%DXXXXwX
%IXXXXwX	%FXXXXwX
%MXXXX	%IXXXXwX
%KMXXXXX	
%KDXXXXX	

OPER1	OPER2
%DXXXXwX	
%FXXXXwX	%MXXXX
%IXXXXwX	

Table 3-8. Syntax of MOP instruction

MOB - Moving of Blocks of Operands



OPER1- first operand of source block
OPER2 - number of transfers to be carried out
OPER3 - control operand
OPER4 - first operand of target 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 format of the operand destination 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.

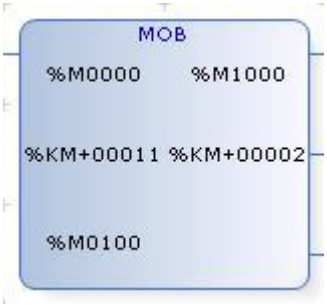
WARNING:
The control operand should not have its contents altered in any part of the applications program, under penalty of preventing the correct execution of the instruction. Each occurrence of this instruction in the program should have an operand of exclusive control, different from to rest. This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

Example:



In the example above if it is required to move the values of the operand block from %M0000 to %M0010, as OPER1 indicates the beginning of the origin block for moving and OPER2 indicates its size. In OPER4 there is the initial operand of the designation level, it means, the first operand of the designation block that is required to move the values is %M1000. OPER5 represents the number of operands that must be moved in each PLC verification that in this case will be 2 (%KM+00002). In OPER3 (%m0100) it is registered what was the last moving done by the instruction.

Syntax:

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

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

Table 3-9. Syntax of MOB instruction

Truth Table (valid only for the PO3x47):

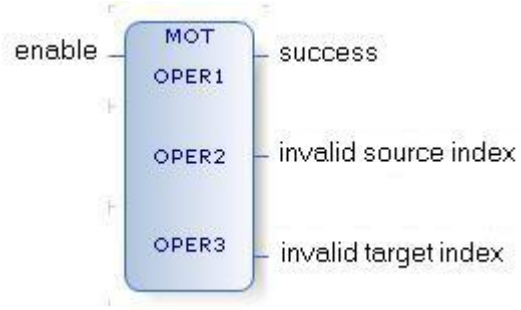
Truth Table of Instruction MOB								
	Input				Output			
Situation	Enable	Oper1 Inval	Oper3 Inval	Oper4 Inval	Oper4	Concl.	Org Inv	Dest Inv
Instruction not Enable	0	x	x	x	unchanged	0	0	0
Invalid Operand 1	1	1	x	x	unchanged	0	1	0
Invalid[1] Operand 3	1	0	1	x	unchanged	0	1	0
Invalid Operand 4	1	0	0	1	unchanged	0	0	1
Parcial copy[2]	1	0	0	0	Oper1	0	0	0
Completed operation	1	0	0	0	Oper1	1	0	0

Table 3-10. Truth table of instruction MOB

Legends:

- Inval = invalid
- Org.Inv = invalid origin
- Dest.Inv = invalid destiny
- 1 = This indication is avaiable only in PO3x47 CPUs
- 2 = Operation not yet concluded, occurs when transferring is in course

MOT - Moving of Tables



OPER1 - source table or source operand
OPER2 - table index
OPER3 - target operand or target 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. When both operands are invalids, the output **origin index invalid** is enabled.

The output **success** is enable when the moving was finished with successful.

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.

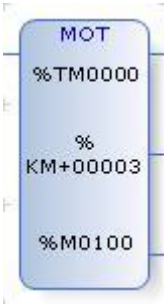
The instruction is programmed in the following way:

- 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

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.

Example:



In this example the value of the position 3 (%KM00003) of the table %TM0000 will be transferred to the memory %M0100.

Syntax:

Read:

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

Write:

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 3-11. Syntax of MOT instruction

Truth Table (valid only for the PO3x47):

Truth Table of instruction MOT: Reading of table content								
	Input				Output			
Situation	Enable	Oper1 Inval	Oper2 <0 ou > tam	Oper3 Inval	Oper3	Suces	Org Inv	Dest Inv
Instruction not Enable	0	x	x	x	unchanged	0	0	0
Invalid Operand 1	1	1	x	x	unchanged	0	1	0
Negative Operand 2 or out of table.	1	0	1	x	unchanged	0	1	0
Invalid operand 3	1	0	0	1	unchanged	0	0	1
Normal Operand	1	0	0	0	Value readed	1	0	0

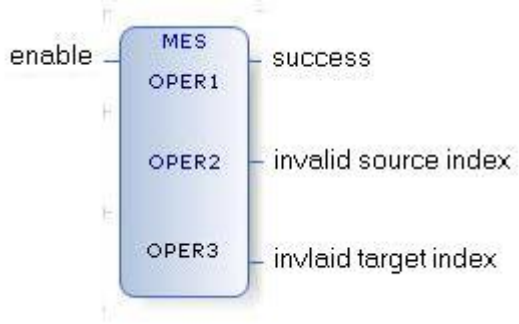
Truth Table of instruction : Writing of values in table								
Input					Output			
Situation	Enable	Oper1 Inval	Oper2 <0 ou > tam	Oper3 Inval	Oper3	Suces	Org Inv	Dest Inv
Instruction not Enable	0	x	x	x	unchanged	0	0	0
Invalid Operand 1	1	1	x	x	unchanged	0	1	0
Negative Operand 2 or out of table	1	0	1	x	unchanged	0	0	1
Invalid operand 3	1	0	0	1	unchanged	0	0	1
Normal Operand	1	0	0	0	Value readed	1	0	0

Table 3-12. Truth table true of MOT instruction

Legends:

- Inval = invalid;
- Org.Inv = invalid origin;
- Dest.Inv = invalid destiny;
- Suces = success.

MES - Moving of Inputs/Outputs



OPER1 - first source operand
OPER2 - number of octets to be transferred
OPER3 - first target operand

Description:

This instruction is used to transfer data directly between memory and octet operands of the modules bus of input and output. It is possible to do value readings of the bus octets value or writings on it, according to the operands programmed in the instruction.

The operand in the first instruction cell (OPER1) is the origin operand, which content will be moved to the destination operand specified in the third cell (OPER3). OPER 2 defines the octets number to be transferred from the first origin and destination specified.

WARNING:
The octets number to be transferred is limited in 255.

If a constant is programmed in the first cell (writing of values on the bus), its value is moved to all bus octets specified by the second and third cell operands.

Whenever the input **enable** is on, one of the instruction outputs is powered, according to the following rules:

The output **invalid source index** is powered in 3 situations:

- If the transfers number specified in OPER2 is negative, zero, more than the maximum octets number jon the PLC bus used (reading on the bus) or the configurated memory limit (writing on the bus).
- If the first reading position is more than the maximum octet number on the PLC bus used (%M*R programmed in OPER1)
- The first memory address to be written is negative or more than the last memory address configurated (%M*M programado em OPER1)

The output **invalid target index** is powered when:

- The transfers number specified in OPER2 is more than the memory limit configured (reading on the bus) or the maximum octets number on the PLC bus used (writing on the bus)
- The first written position is more than the maximum octets number on the PLC bus used (%M*R programmed in OPER3)
- The first memory address to be read is negative or more than the last memory address configurated (%M*M programado em OPER3)

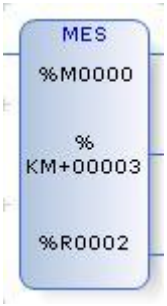
The output **success** is powered when the outputs **invalid source index and invalid target index** are not powered.

This instruction is used only to special accesses to the bus. For its use, the E/S module placed in the bus physical position read or written by MES and how to access it must be known. As the input modules are given by ALTUS they have specific instructions about how to access them. The MES instruction is not necessary in most of the application programs.

It is not possible to write values in digital input modules of octets or reading values of digital module octets of output with MES.

WARNING:
This instruction is valid only to PLCs of AL series.

Example:



In this case, the values of the memory band %M0000 to %M0002 (the operand %KM+00003 determines the band size) will be moved to the registers of the bus %R0002 to %R0004.

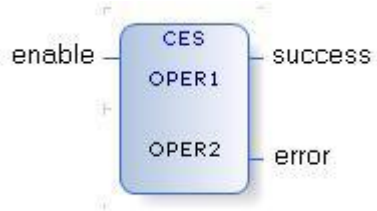
Syntax:

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

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

Table 3-13. Syntax of MES instruction

CES - Conversion of Inputs/Outputs



OPER1 - source operand
OPER2 - target operand

Description:

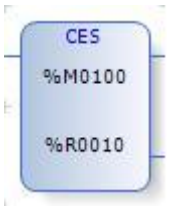
This instruction is used to transfer data directly between memory operands and octets in the bus, converting the binary values to BCD, when writing to the bus, or BCD, for binary, when reading

If it is required to convert the bus octets to a memory, the initial octet should be programmed in OPER1 and the memory to receive the converted values in OPER2. The instruction concatenates the octet value specified with the following octet, converts from the BCD format to binary and stores the converted value in the destination memory.

If it is required to convert value from one memory or constant memory to the bus, the value to be converted should be specified in OPER1, and in OPER2 the initial octet to receive the values. The instruction converts the value to BCD format end writes it to the octet specified and the following one.

If the value moved to the bus has more than 4 digits, the more significant surplus digits are discarded.

Example:



To move the contents of %M0100 to %R0010:

- Value of %M0100 = 21947, equivalent to 101010110111011 binary form
- Value of %M0100 = 21947, converted to 0010 0001 1001 0100 0111 in the BCD form
- Value moved to %R0010 = 47 in the BCD form, equivalent to 0100 0111 written in the octet
- Value moved to %R0011 = 19 in BCD format, equivalent to 0001 1001 written in the octet

A in The instruction is always executed when the input **enable** is powered. The output **success** is powered if the instruction is executed correctly.

The output **error** is powered when an invalid access is made to same operand indirectly referenced for a memory.

Syntax:

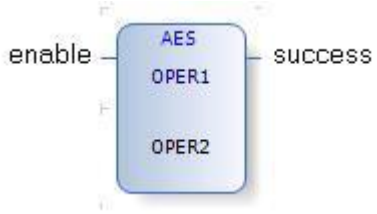
OPER1	OPER2
%R	%M
%M*R	%M*M

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

Table 3-14. Syntax of CES instruction

ATENTION: This instruction is only to PLC`s AL Series.

AES - Updates of Inputs/Outputs



OPER1 - first octet of operands to update
OPER2 - number of octets update

Description:

This instruction executes an immediate updating in the memory image for the specified operands. Its updating is identical to the scan of the E/S points carried out by the executive program at the end of each scan, however with the number of operands limited.

The first operand (OPER1) contains the first octet of operands to be updated, while the second operand (OPER2) shows the total number of octets to update. The operands % E (input) are read from the bus to the image memory and the operands % S (output) are written from the image memory to the bus when the instruction is executed.

If the number of operands to update exceeds the number of operands declared, it is only possible to update from the type declared.

If no octet is updated from the instruction, the output **success** is turned off.

The instruction AES should be used only in special processing, where there is a very fast time delay or a constant is demanded by the PLC. In relatively small applications programs, with a short scan time and common control tasks, it does not have to be used.

Example:

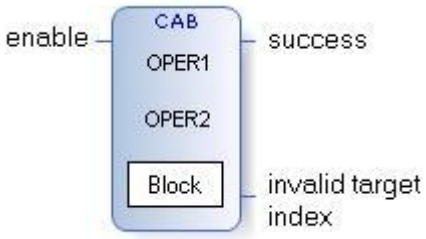
If the PLCs configuration is 16 input octets (%E0000 to %ED015) and 8 output octets (%S0016 to %S0023), the instruction shown will update only 4 octets (%E0012 to E0015). No output octet is updated.

Syntax:

OPER1	OPER2
%E %S	%KM %M

Table 3-15. Syntax of AES instruction

CAB - Load Block



OPER1 - initial operand or table to be loaded
OPER2 - number of operands or positions of table

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).

The value of the second operand should be positive, less or equal to %KM+255.

This values are declared selecting the button **Block,** then an edition window is opened in MasterTool XE. The constants edited in the block can vary according to the operand type:

- Are of type %KM if the type of the first operand is %E, %S, %A, %M, %TM
- Are of type %KD if the first operand is %D or %TD
- Are of type %KF if the first operand is %F or %TF
- Are of type %KI if the first operand is %I or %TI

If the first operand is an octect (%E< %S or %A), only the less significative octets values of each declared constant will be moved.

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

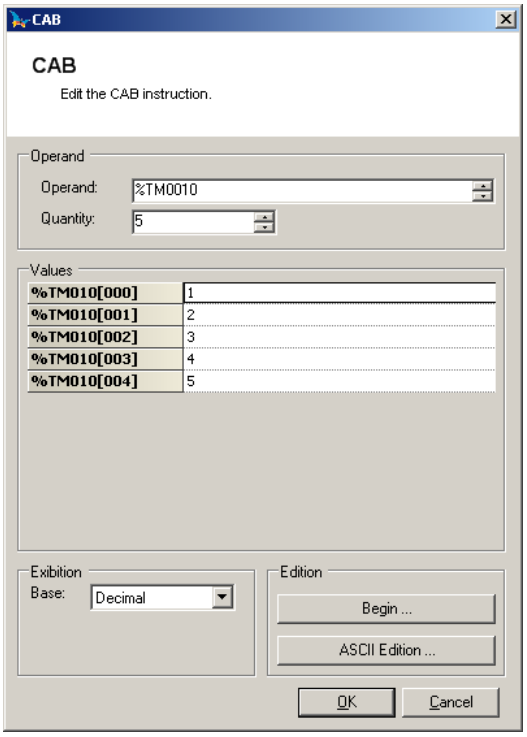


Figure 3-3. Edition CAB window

It is also possible to declare table ASCII values. Pressing the button **Edition ASCII** that opens the window **CAB – Edition in ASCII,** it is possible to type a text that will be loaded in the screen with the ASCII values related to each character. The edition of the values in ASCII is limited to the operand %TM and to the maximum size of 128 positions of the table.

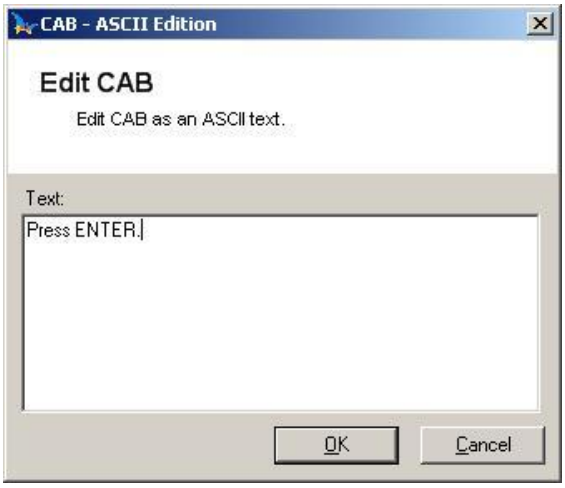


Figure 3-4. CAB on ASCII edition

The value can be initialized with an only value through the window **CAB – Initialize Block**, accessed through the **Initialize** button. In this window it is possible to edit the value and the position to where the value will be moved.

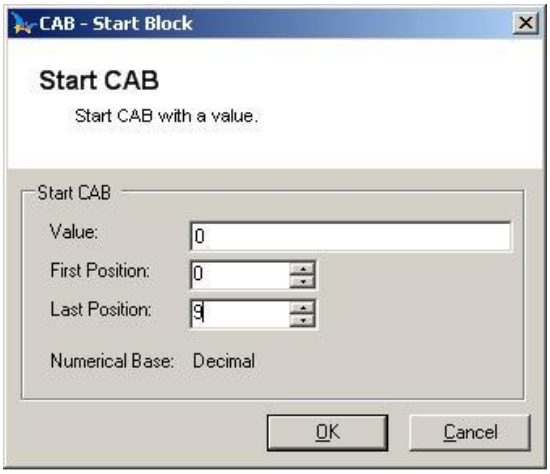


Figure 3-5. CAB start window

The output **invalid target index** 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 **invalid target index** 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 (E000) or that it is enabled only at the necessary loading times.

Example:

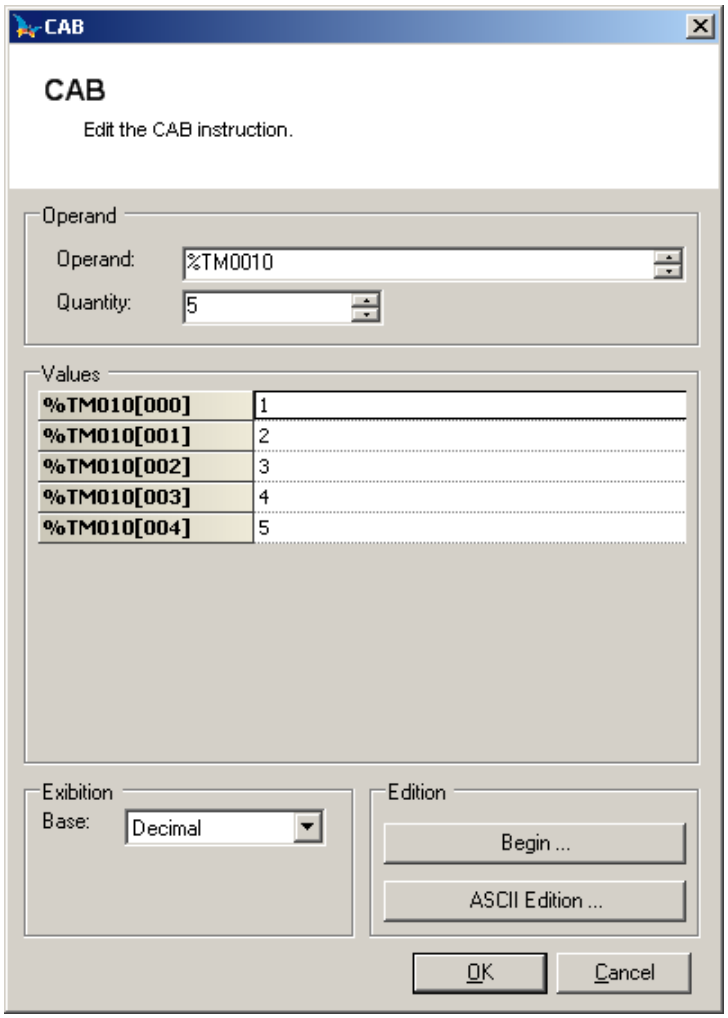


Figure 3-6. Example of setting the instruction CAB

This example shows how the CAB instruction is configured to copy value 1, 2, 3, 4 and 5 to the first five positions of the operand %TM0010.

Syntax:

OPER1	OPER2	OPER3
%E %S %A %M %TM %M*E %M*S %M*A %M*M %M*TM	%%KM	Table of values memory

OPER1	OPER2	OPER3
%D %TD %M*D %M*TD	%KD	Table of values decimals

OPER1	OPER2	OPER3
%F %TF %M*F %M*TF	%KF	Table of values float

OPER1	OPER2	OPER3
%I %TI %M*I %M*TI	%KI	Table of values Integers

Table 3-16. Syntax of CAB instruction

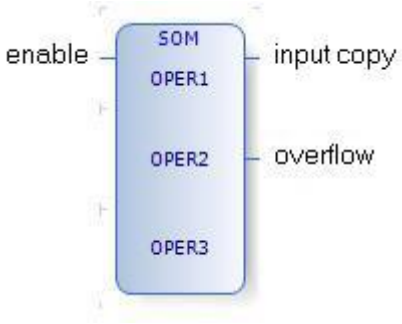
Arithmetic Group Instructions

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	Name Description	Edition Sequence
SOM	Sum	Alt, I, A, S
SUB	Subtraction	Alt, I, A, B
MUL	Multiplication	Alt, I, A, M
DIV	Division	Alt, I, A, D
AND	**And** Binary between operands	Alt, I, A, A
OR	**Or** binary between operands	Alt, I, A, O
XOR	**Or Exclusive** between operands	Alt, I, A, X
CAR	Load Operand	Alt, I, A, C
EQUAL	Equal	Alt, I, A, I
LESSER	Lesser	Alt, I, A, N
GREATER	Greater	Alt, I, A, R

Table 3-17. Arithmetic group instructions

SOM - Sum



OPER1 - first plot
OPER2 - second plot
OPER3 - total

Description*:*

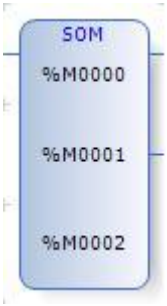
This instruction carries out the arithmetic sum of operands. When the input **enabled** is powered, the values of the specified operands in the first two cells are added and the result stored in the operand of the third cell.

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.

When a instruction destination is an integer (%M) and at least one of the other operands of the instruction is real (%F) the stored result will be truncated, it means, it will be stored at the operand M only the integer part of the operation result.

Example:



If the operand %M0000 has the value 100, and operand %M0001 has the value 34, after the enabled input is powered, the maximum value on the operand %M0002 will be 134 and output copies of the entry also powered.

If the operand value %M0000 is 30000 and %M0001 is 150000, for example, the output **overflow** will be powered, as it has passed the store limit of the operands %M.

Syntax:

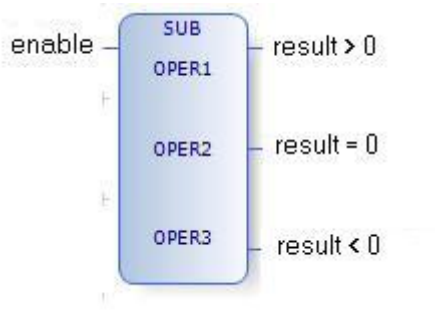
OPER1	OPER2	OPER3
%KD %D	%KD %D	 %D

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

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

Table 3-18. Syntax of SOM instruction

SUB - Subtraction



OPER1 - first plot
OPER2 - second plot
OPER3 - result

Description:

This instruction carries out the subtraction arithmetic between operands. When **enables** 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.

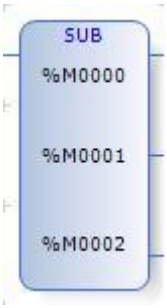
The lines of output **result > 0, result = 0** and the **result < 0** can be used for comparisons and are enabled according to the result of the subtraction.

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.

When the destination operand of the instruction is an integer (%M) and at least one of the other operands of the instructions is a real (%F) the stored result will be truncated, it means, it is stored in the operand M only the integer part of the operation result.

Example:



If the operand %M has the value 100, and the operand %M001 has the value 34, after the enabled input is powered, the value on the operand %M0002 will be 66 and the output **result > 0** (more than zero) will be powered.

If the operand value %M0000 is 1000 and %M0001 is 1500, for example, the output **result < 0** (less than zero) will be powered.

And finally, if the operand value %M0000 is 10 and %M0001 is also 10, the output **result = 0** (equals to zero) will be enabled.

Syntax:

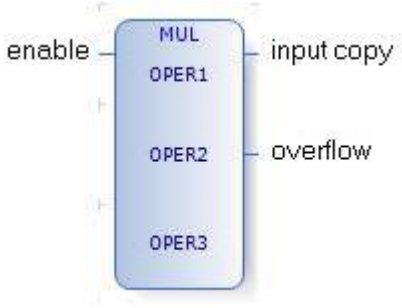
OPER1	OPER2	OPER3
%KD %D	%KD %D	 %D

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

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

Table 3-19. Syntax of SUB instruction

MUL - Multiplication



OPER1 - multiplied
OPER2 –multiplier
OPER3 - result

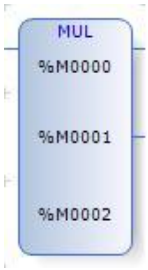
Description:

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.

When the destination operand of the instruction is an integer (%M) and at least one of the other operands of the instruction is a real (%F) the stored result will be truncated, it means, only the integer part of the result is stored in the operand.

Example:



In the case above, if the value of %M0000 is 10 and the value of %M0001 is 6, after the input enables being powered, the value in %M0002 will be 60, and the output copies to the input will be enabled. However, if the value %M0000 is 10000 and the value of %M0001 is 5, the output **overflow** will be powered.

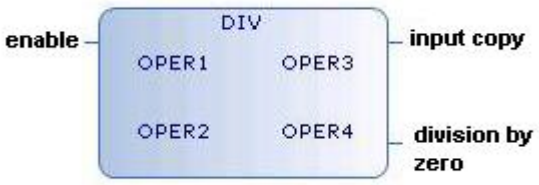
Syntax:

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

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

Table 3-20. Syntax of MUL instruction

DIV - Division



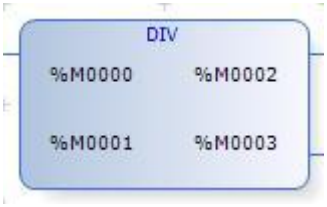
OPER1 - divided
OPER2 - divider
OPER3 - quotient
OPER4 - remainder

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.

Example:



If in the instruction above the value %M0000 is 11, and the value %M0001 is 10, after the powering of the input enables, the result of the division will be shown in the operand %M0002 with the value 1, and in the operand %M0003 the value 1 will be shown, which is the remainder of the division. If it is carried out normally, the copy output of the enter will also be enabled, and in case there is division by zero, the output with the same name will be unabled.

Syntax:

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

OPER1	OPER2	OPER3	OPER4
%KF %F %KM %M	%KF %F %KM %M	%M	%M

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

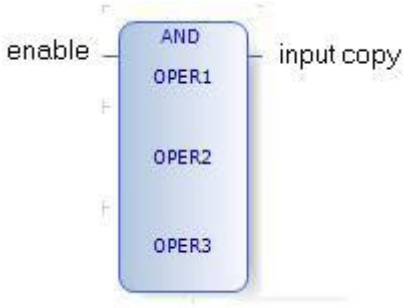
OPER1	OPER2	OPER3	OPER4
%KF	%KF		
%F	%F		
%KM	%KM	%I	%I
%M	%M		

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

Table 3-21. Syntax of DIV instruction

NU= Operating not used for the instruction, being able to be informed any declared address of operating in the CP.

AND - And Binary between operands



OPER1 - first operand
OPER2 - second operand
OPER3 - result

Description:

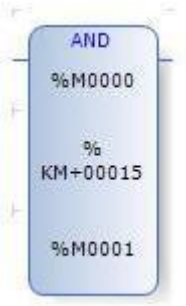
This instruction carries out the operation “and” binary between the first two operands, storing the result in the third.

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 3-22. Operations point to point (AND)

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 (0000011 binary).

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

Table 3-23. Example of an AND operation

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

Syntax:

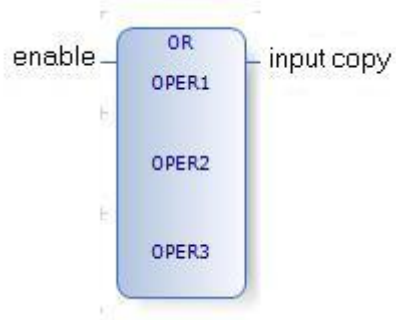
OPER1	OPER2	OPER3
%KM %M	%KM %M	 %M

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

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

Table 3-24. Syntax of AND instruction

OR - Or binary between operands



OPER1 - first operand
OPER2 - second operand
OPER3 - result

Description:

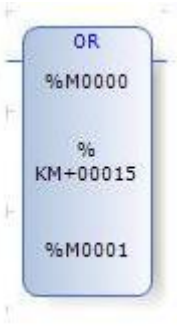
This instruction carries out the operation “or” binary between the values of the first two operands, storing the result in the third.

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 3-25. 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 (011011100111111 binary)

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

Table 3-26. Example of an OR operation

Syntax:

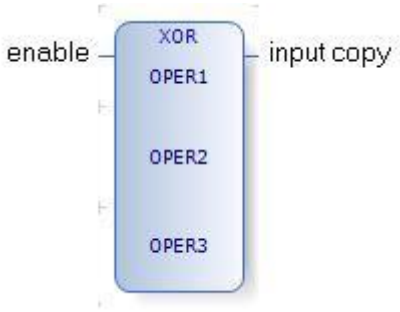
OPER1	OPER2	OPER3
%KM %M	%KM %M	%M

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

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

Table 3-27. Syntax of OR instruction

XOR - Or Exclusive between operands



OPER1 - first operand
OPER2 - second operand
OPER3 - result

Description:

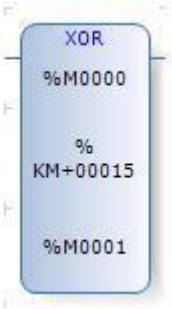
This instruction carries out the operation “or exclusive” binary between the two first operands, storing the result in the third.

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 points.

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

Table 3-28. 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 de %M0000)
XOR 15	XOR 00000000 00001111 (value de %KM+00015)
1603	00000110 01000011 (result em %M0001)

Table 3-29. Example of an XOR operation

Therefore, the decimal value 1603 is stored in M0001.

Syntax:

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

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

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

Table 3-30. Syntax of XOR instruction

CAR - Load Operand

OPER1 - operand to be loaded

Description:

The instruction load 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

ATTENTION: The comparing of decimal operands and real operands is not allowed

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

Example:

Check examples of use of this instruction in Instructions for Comparing Operands – Equals, Less than and More than.

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 3-31. Syntax of CAR instruction

ATTENTION:
- If in the case, an instruction CAR will be qualified and soon after of this, one another indexed instruction CAR with operating invalid, instruction CAR remains with the previously loaded value, exactly if a new instruction CAR will be set in motion with operating invalid.
- The comparison instructions will be executed instruction CAR after to have been executed, contrary case the comparison instructions will not return true.
- If to place instructions of comparation in module E000 and a CAR in module E001, the instruction of comparison will not be executed after the POWER-ON command, only after change the mode programming/execution.

Instructions of Comparison of Operands - Equals, Greater and Lesser

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). If any of the operands is different of the real type, the less precise operand is converted to real and after the same comparison is done point to point.

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. Check the following example.

ATTENTION:
The comparing of decimal operands and real operands is not allowed

Example:

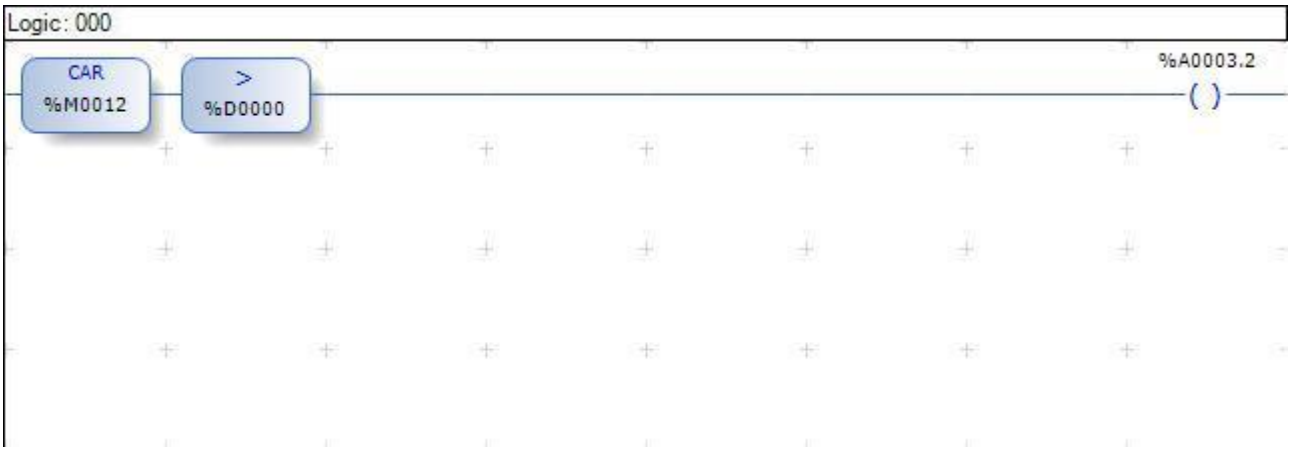


Figure 3-7. Example of comparing instructions

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, even showing it as a negative value.

Figure 3-8. Example of comparing instructions

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

Example:

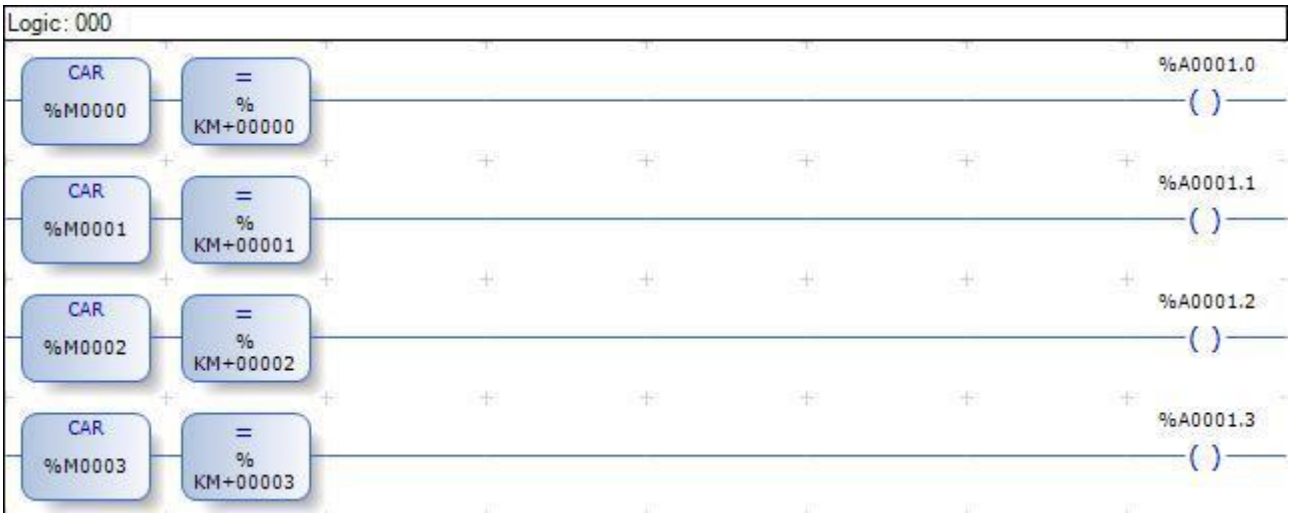


Figure 3-9. Incorrect using of CAR instruction

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 occurs in a different form than the visual form suggests. As the processing of the logic takes places 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 fact, 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 manner:

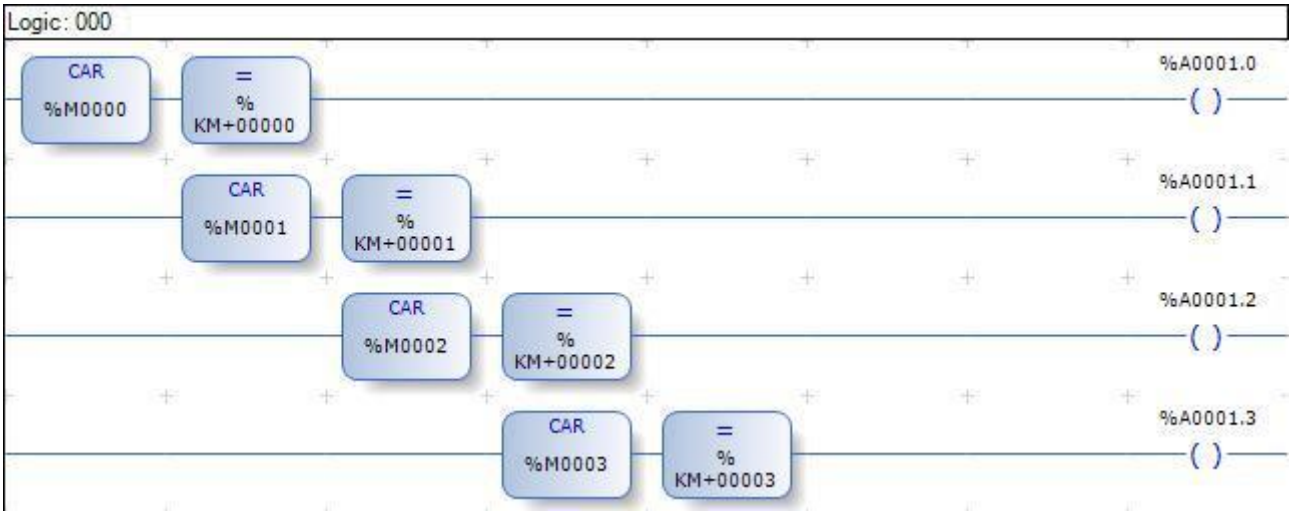


Figure 3-10. Correct using of CAR instruction

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 3-32. Syntax of Greater, Lesser and Equal instructions

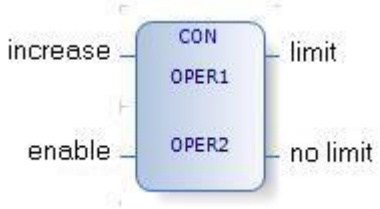
Instructions of Counters Group

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

Name	Name Description	Edition Sequence
CON	Simple Counter	Alt, I, C, N
COB	Bidirectional Counter	Alt, I, C, B
TEE	Timer on enabling	Alt, I, C, T
TED	Timer on disabling	Alt, I, C, D

Table 3-33. Instructions of counters group

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

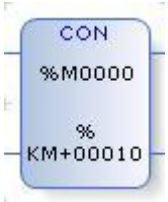
When the input **enable** is powered, each transition of connection in the input **increase** raises the value of the operand counter (OPER1) by one unit.

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 **increase**, 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 enable **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.**

Example:



In this case, when a pulse is generated in the increment input, the counter value inside the memory %M0000 rises from a unity. For that, the active input must always be powered. While the counting does not reaches the limit stipulated by %KM+00010, the output non-limit remains abled. When the counting reaches its limit, the output non-limit is enabled and the limit output is powered. It is necessary to turn off the active input and turn it on again to reinitialize the counter and start a new counting.

Syntax:

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

Table 3-34. Syntax of CON instruction

Truth Table (valid only for the PO3x47):

Truth Table of CON instruction							
Input					Output		
Situation	Inc	Active	Oper1 < 0	Oper1 >= Oper2	Operand	Lim	NL
Invalid Oper1	x	x	x	x	unchanged	0	1
Input not active	x	0	x	x	0	0	1
Oper2. Invalid or negative	x	1	0	x	unchanged	0	1
Oper2. Invalid or negative	x	1	1	x	0	0	1
Without transition	nT	1	0	0	Oper1	0	1
Without transition, superior limit	nT	1	0	1	Oper2	1	0
transition	T	1	0	0	Oper1 + 1	0	1
transition, superior limit	T	1	0	1	Oper2	1	0

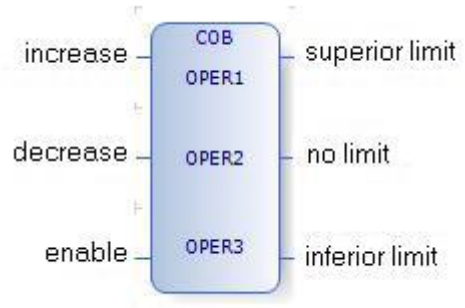
Table 3-35. Truth table of CON instruction

Legends:

- T = transition
- nT = not transition
- x = don’t care
- Inval = invalid
- Lim = limit
- NL = not limit
- Inc = increments

When some operand is invalid or operand 2 is negative output is not zeroed.

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 **enable** 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 **superior limit** is powered, not being increased.

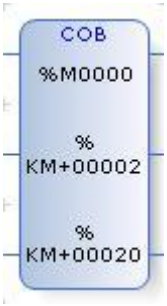
If the value of the first operand is equal to or less than zero, the output **inferior limit** is enabled, zero being stored in the first operand.

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

WARNING:
With the input enable **deactivated, the output** inferior 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.**

Example:



In the case above the counting value is stored in %M0000. To enable the counter the input must be powered. To increment the counting, a pulse in the increment input must be given, and to decrease, the same should be done on the decrease input. The counting in this example will be two by two, due to the value of the operand %KM+00002, and the limit is 20 (%KM00020). While the counting does not reach its limits, the output no limit remains enabled. If the superior or inferior limit is reached, its respective outputs are powered.

Syntax:

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

Table 3-36. Syntax of COB instruction

Truth Table (valid only for the PO3x47):

Truth Table of instruction COB										
Input							Output			
Situation	Inc	Dec	Aticve	Oper3 < 0	Oper1 < 0	Oper1 >= Oper3	Operand	Lim+	NL	Lim-
Oper 1. Invalid	x	x	x	x	x	x	unchanged	0	0	1
Input 2 not active	x	x	0	x	x	x	0	0	0	1
Oper 2 or Oper 3 invalid	x	x	1	x	1	x	0	0	0	1
Oper 2 or Oper 3 invalid	x	x	1	x	0	x	unchanged	0	0	1
Oper 3 negative	x	x	1	1	1	x	0	0	0	1
Oper 3 negative	x	x	1	1	0	X	unchanged	0	0	1
Inferior limit	x	x	1	0	0	0	0	0	0	1
Without transition, superior limit	nT ou T	nT ou T	1	0	0	1	Oper3	1	0	0
Without transition	nT ou T	nT ou T	1	0	0	0	Oper1	0	1	0
Positive transition	T	nT	1	0	0	0	Oper1 + Oper2	0	1	0
Positive transition, inferior limit	T	nT	1	0	1	0	Oper2	0	1	0
Positive transition, superior limit	T	nT	1	0	0	1	Oper3	1	0	0
Negative transition	nT	T	1	0	0	0	Oper1 – Oper2	0	1	0
Negative transition, inferior limit	NT	T	1	0	1	0	Oper2	0	1	0
Negative transition, superior limit	nT	T	1	0	0	1	Oper3-Oper2	0	1	0

Table 3-37. Truth table of instruction COB

Legends:

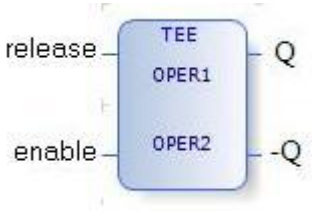
- T = transition
- nT = not transition
- x = don’t care

- Inval = invalid
- Lim = limit
- NL = not limit
- Inc = increments

When both inputs have a transition, there are not increment or decrement.

When some operand is invalid or operand 3 is negative, output is not modified.

TEE - Timer on enabling



OPER1 - time accumulater
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 **release** and **enable** 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 **release**, there is an interruption in the count time, OPER1 keeping the same value. Disabling the input **enable**, 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 enable **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:

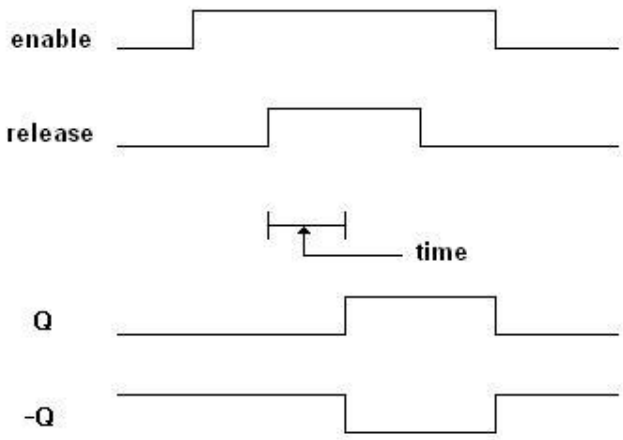


Figure 3-11. Diagram of times from TEE instruction

Example:

In the example above, if the active input is on, when the input free is activated, the output Q will be powered after 10 (%KM+00100) seconds. On the output –Q the value of the output Q is inverted. On the operand %M0000 the value of the time counting is stored.

Syntax:

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

Table 3-38. Syntax of TEE instruction

WARNING:
This instruction it is not be enable for functioning in modules of external interruption E020.

Truth Table (valid only for the PO3x47):

Truth Table of TEE instruction							
	Input				Output		
Situation	Free	Aticve	Oper1 < 0	Oper1 >= Oper2	Operand	Q	- Q
Oper1 invalid	x	x	x	x	unchanged	0	1
Input not active	x	0	x	x	0	0	1
Oper2. Invalid or negative	x	1	0	x	unchanged	0	1
Oper2. Invalid or negative	x	1	1	x	0	0	1
Time = T	1	1	x	0	Oper1+T	0	1
Time = T, superior Limit	1	1	x	1	Oper2	1	0
Time < T	1	1	x	0	Oper1	0	1
Time < T, superior Limit	1	1	x	1	Oper2	1	0
Not free	0	1	x	0	Oper1	0	1
Not free, superior Limit	0	1	x	1	Oper2	1	0

Table 3-39. Truth table of TEE instruction

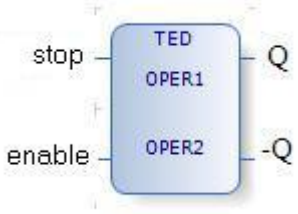
Legends:

- T = scan time in 0,1s units
- x = don't care

When Oper1 is negative it is evaluated as zero.

ATTENTION:
This instruction has some issues that needs attention:
- TEE instruction is not enabled for working into external interrupt modules E020
- Don't execute this instruction in one cycle
- Execute this instruction more than one time in the same cycle
- This instruction must be not skipped

TED - Timer on disabling



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 **enable** is powered and the input **stop** 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 **enable** is powered and OPER1 is less than OPER2. Enabling the input **stop**, there is an interruption in the time count, while disabling the input **enable**, 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**, if the instruction is deactivated.

WARNING:
With the input enable **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:

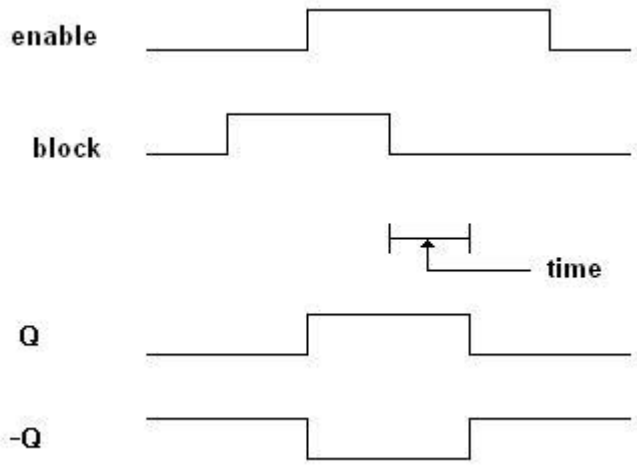
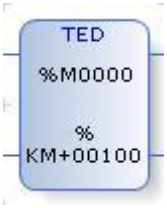


Figure 3-12. Diagram of times from TED instruction

Example:



In the example above, if the active input is on, when the block input is not powered the Q output will be turned off after 10 (%KM+00100) seconds. On the -Q output there is the value of the denied Q output. On the operand %M0000 the value of the time counting is stored.

Syntax:

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

Table 3-40. Syntax of TED instruction

Truth Table (valid only for the PO3x47):

Truth Table of TED instruction							
Input					Output		
Situation	Blocks	Active	Oper1 < 0	Oper1 >= Oper2	Operand	Q	- Q
Oper1 invalid	x	x	x	x	unchanged	0	1
Input not active	x	0	x	x	0	0	1
Oper2. Invalid or Negative	x	1	0	x	unchanged	1	0
Oper2. Invalid or Negative	x	1	1	x	0	1	0
Tempo = T	0	1	x	0	Oper1+T	1	0
Tempo = T, superior limit	0	1	x	1	Oper2	0	1
Tempo < T	0	1	x	0	Oper1	1	0
Tempo < T, superior limit	0	1	x	1	Oper2	0	1
Blocks	1	1	x	0	Oper1	1	0
Blocks, superior limit	1	1	x	1	Oper2	0	1

Table 3-41. Truth table of TED instruction

Legends:

- T = scan time in 0,1s units
- x = don’t care

When Oper1 is negative it is evaluated as zero.

ATTENTION:
This instruction has some issues that needs attention:
- TED instruction is not enabled for working into external interrupt modules E020
- Don’t execute this instruction in one cycle
- Execute this instruction more than one time in the same cycle
- This instruction must be not skipped

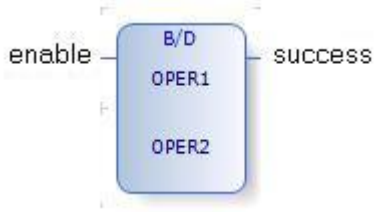
Instructions of the Conversion Group

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	Name Description	Edition Sequence
BIN/DEC	Conversion Binary-Decimal	Alt, I, V, B
DEC/BIN	Conversion Decimal-Binary	Alt, I, V, D
ANA/DIG	Conversion Analog-Digital	Alt, I, V, A
DIG/ANA	Conversion Digital-Analog	Alt, I, V, G

Table 3-42. Instructions of the conversion group

B/D - Conversion Binary-Decimal



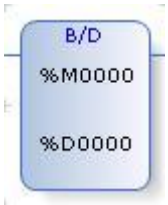
OPER1 - source
OPER2 - target

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.

Example:



If the value on the operand %M0000 is 10 (in binary 0000 0000 0000 1010), after the powering of the enable input, the operand value %D0000 will be 10 (in binary considering only the low part 0000 0000 0001 0000), that is, the value of %M0000 was converted to the BCD format.

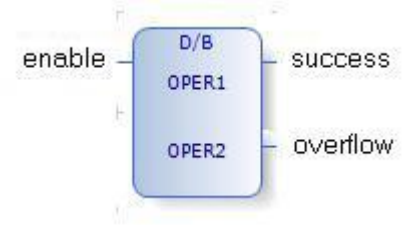
WARNING:
If the instruction used for this type of operation is MOV, discrepancies can happen in the value of the %D operands.

Syntax:

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

Table 3-43. Syntax of B/D instruction

D/B - Conversion Decimal-Binary



OPER1 - source
OPER2 - target

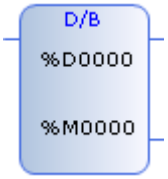
Description:

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.

Example:



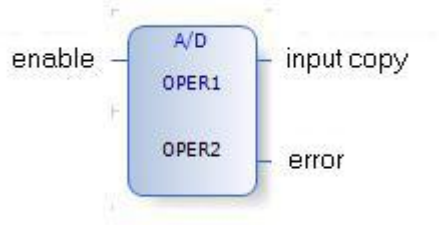
If in the operand %D0000 there is the value 23 (in binary considering only the low part 0000 00000010 0011), after the powering of the enable input, the operand value %M0000 will be 23 (in binary 0000 0000 0001 0111), that is, the value of %D0000 in DCD was converted to the operand format %M0000.

Syntax:

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

Table 3-44. Syntax of D/B instruction

A/D - Conversion Analogic-Digital



OPER1 - address of the module on the bus / number of channels to convert
OPER2 - first operand to receive the converted value

Description:

This instruction converts the read values of an input analogic module to numerical values stored on the operands.

It is possible to read from 1 or 8 channels altering only the specification of the first operand that indicates the address on the bus occupied by the module A/D. This module must be specified on the declaration of the bus, achieved on MasterTool XE. The address to be programmed in OPER1 can be obtained on MasterTool XE. The converted values are placed on memory operands, defined in OPER2.

The conversion is achieved only if the input **enable** is powered.

If OPER1 is specified with subdivision of the point type (%RXXXX.X), the conversion is achieved only to the module channel related to the point. The points 0 to 7 of the operand correspond to channels 0 to 7 of the module, respectively. In this format, the execution time of the instruction is significantly shorter than the conversion of the 8 channels, being appropriate, for example, to the use in programming modules E18, enabled by time interruption.

If OPER1 is specified as %RXXXX (8 channels conversion), the converted value are placed in the declared memory in OPER2 and in the 7 subsequent memories.

If OPER1 is specified as %RXXXX.X (1 channel local conversion), the converted value is placed in the declared memory in OPER2.

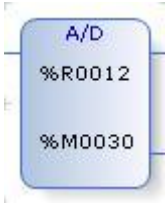
The available modules to achieve the conversion A/D are shown bellow. The converted values by the instruction belong to a related band with each module characteristic:

- AL-1103 (10 bits): value from 0000 to 1023
- AL-1116 (12 bits): value from 0000 to 4095
- AL-1119 e QK1119 (12 bits): value from 0000 to 4095
- AL-1139: value from 0000 to 3999, with overflow indication (4000 to 4095)

The output **error** of the instruction is enabled in any of the following situations:

- Module declare don the bus is invalid to the instruction (it is not one of the modules listed before)
- Try of assessment of operands not declared
- Conversion error (except AL-1103)

Example:



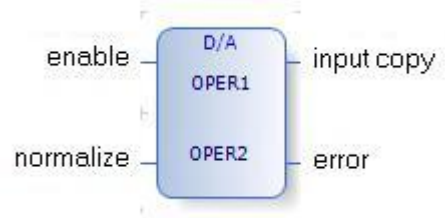
If in the register of operand %R0012 an analogic input is declared, these data will be transferred to operand %M0000.

Syntax:

OPER1	OPER2
%RXXXX %RXXXX.X	%M

Table 3-45. Syntax of A/D instruction

D/A – Conversion Digital-Analogic



OPER1 - first operand with values to be converted
OPER2 - module address on the bus / number of channels to convert

Description:

This instruction converts the numerical values of memories to analogic signals. The values are converted through cards of analogic output AL-1203, AL-1214 or AL-1222. The conversion of 1 or 4 channels is possible, using only one D/A instruction.

The first operand specifies the first memory with value to be converted.

The second operand indicates the address of the D/A module on the module bus. The module must be specified on the bus declaration, achieved on MasterTool XE. The address to be programmed in OPER2 can be obtained directly through MasterTool XE.

The conversion is only achieved if the input **enable** is powered.

If OPER2 is specified with subdivision of the point type (%RXXXX.X), the conversion is achieved from the declared operand in OPER1 to the module channel that corresponds to the point. The points 0 to 3 of the operand correspond to channels 0 to 3 of the module, respectively.

If OPER2 is specified as %RXXXX (4 channel conversion), the value to be converted are obtained from the declared memory in OPER1 and 3 subsequent memories.

The available modules to achieve the D/A conversion are shown bellow. The converted value by the instruction belong to a module related to the characteristic of each module:

Voltage Output:

Module	Resolution	Normalization	Band
AL-1203	10 bits	Not Used	0000 to 1000
AL-1214	10 bits	Not Used	0000 to 1000
AL-1222	12 bits	Off	0000 to 4000
AL-1222	12 bits	On	-2000 to +2000

Table 3-46. Instruction D/A – Voltage output

Current Output:

Module	Resolution	Normalization	Band
AL-1203	10 bits	Not Used	0000 to 1000
AL-1214	10 bits	Not Used	0000 to 1000
AL-1222	11 bits	Off	0000 to 4000

Table 3-47. Instruction D/A – Current output

The converted value of AL-1222 depend on the input **normalize,** that converts symmetric values when powered. It becomes useful when working with negative values is necessary, for example on the voltage band ±10V

There is no normalization to the modules AS-1203 and AL-1214, only to AL-1222. However, the normalization is only possible to the operation on the voltage mode.

WARNING:
In CORRENTE mode the input normalize must not be powered.

AL-1222 can work with its 4 outputs in voltage or current mode, or both modes simultaneously. The selection on the operation point is made by the user through addressing programming of the module on OPER2:

- If %RXXXX is even, converts current
- If %RXXXX is odd, converts voltage

Example:

If the module is placed on the address %R0024 on the bus, and the instruction is programmed with %R0024, the AL-1222 will work in current mode. If it is programmed with %R0025, it will work on voltage mode.

WARNING:
The instruction cannot be jumped during the execution of the applications program, or the value can be shown not correctly.

The output **error** of the instruction is activated in the following situations:

- Declared module on the bus is not valid to the instruction (it is not one of the modules listed before)
- Try to access non-declared operands

Syntax:

OPER1	OPER2
%M	%RXXXX %RXXXX.X

Table 3-48. Syntax of D/A instruction

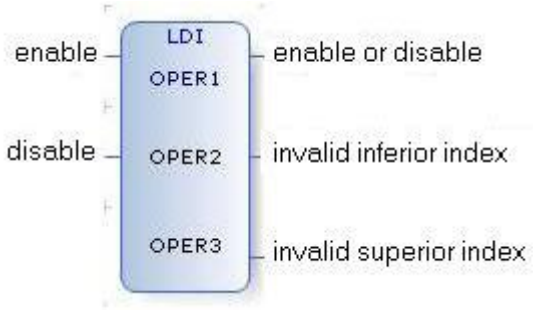
Instructions of the General Group

The general group instructions allow the testing and enabling of points indirectly, implementations of status machines, calls for procedures and functions and writing and reading of operands on ALNET II network.

Name	Name Description	Edition Sequence
LDI	Enable/Disable indexed points	Alt, I, G, L
TEI	Test of Indexed Point Status	Alt, I, G, T
SEQ	Sequencer	Alt, I, G, S
CHP	Call the Procedure Module	Alt, I, G, P
CHF	Call Function Module	Alt, I, G, F
ECR	Writing of operands in other PLC	Alt, I, G, E
LTR	Reading of Operands from Another PLC	Alt, I, G, T, T
LAI	Free Updating of Operand Images	Alt, I, G, I
ECH	Writing of Operands in Another PLC to Ethertnet	Alt, I, G, E
LTH	Reading of Operands from Another PLC to Ethernet	Alt, I, G,

Table 3-49. Instructions of the general group

LDI - Enable/Disable indexed points



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). Even when the instruction is used to connect or disconnect points of output (%S), 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 must 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 **enable** or **disable** 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 **invalid superior index** or **invalid inferior index** 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 outputs are turned off.

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

VALUE OPER1 = (OCTET*8) + POINT

Example:

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

- OCTET = 10
- POINT = 5
- 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:

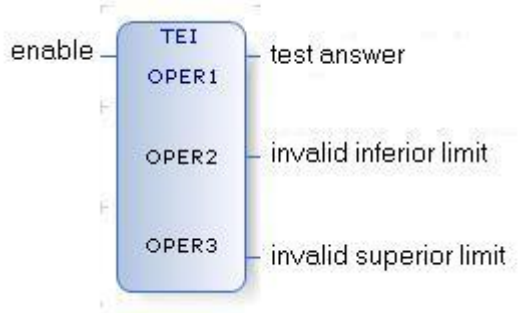
OPER1	OPER2	OPER3
%M*E	%EXXXX.X	%EXXXX.X

OPER1	OPER2	OPER3
%M*S	%SXXXX.X	%SXXXX.X

OPER1	OPER2	OPER3
%M*A	%AXXXX.X	%AXXXX.X

Table 3-50. Syntax of LDI instruction

TEI - Test of Indexed Point 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. Even 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).

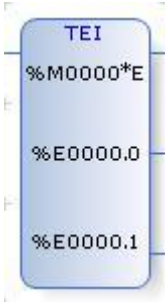
If the input **enable** is powered, the status of the relay or auxiliary specified for the value contained in the memory index (OPER1) is examined. According to whether they are 1 or 0, the output **answer** is connected or not.

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 **invalid superior limit** or **invalid inferior limit** 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 of the required point, is the same specified in the instruction **LDI**

Example:



If operand %M0000 has value 1, the test will be done in the operand %E0000.1 is active or not. If %M0000 has 0 value, the tested operand will be %E0000.0.

Syntax:

OPER1	OPER2	OPER3
%M*E	%EXXXX.X	%EXXXX.X

OPER1	OPER2	OPER3
%M*S	%SXXXX.X	%SXXXX.X

OPER1	OPER2	OPER3
%M*A	%AXXXX.X	%AXXXX.X

Table 3-51. Syntax of TEI instruction

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 XE 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 XE 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.

In the table specified in OPER1 values in decimal format must be loaded, that specify auxiliary operands points that must be tested as evolution conditions. The calculation of these values is specified through the equation:

VALUE = (address of the operand *8) + 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.

There is, however, less freedom to choose the evolution conditions in relation to the 1000 mode, besides it is necessary the use of more memory (tables) in mode 3000.

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.

Besides that, 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 is 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 is 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<END1>.<SUB1> or A<SUB1> (first table)
- M<END2>.<SUB2> or A<END2>.<SUB2> (second table)

where:

- END1 = END3 + STATUS/16 (if operand %M)
- END1 = END3 + STATUS/8 (if operand %A)
- SUB1 = REST (STATUS/16) (if operand % M)
- SUB1 = REST (STATUS/8) (if operand % A)
- END2 = END4 +STATUS/16 (if operand % M)
- END2 = END4 + STATUS/8 (if operand % A)
- SUB2 = REST (STATUS/16) (if operand % M)
- SUB2 = REST (STATUS/8) (if operand % A)

Example:

They may be:

- OPER1 = %TM000
- OPER2 = %M0010
- OPER3 = %M0100
- OPER4 = %A0020

Where:

%TM0000	Posição	Valor
	000	00001
	001	00002
	002	00004
	003	00001
	004	00000

%TM0001	Posição	Valor
	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...	...

Table 3-52. Operandos and values used in the example

Then the evolution conditions starting from status 1 will be:

For the first table:

- 100 + 1/16 = 100
- Rest(1/16) = 1
- Point to be tested = %M0100.1

For the second table:

- 20 + 1/8 = 20
- 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 change
0	1	00003	Status change as %TM001
1	0	00002	Status change as %TM000
1	1	00002	Status change as %TM000 (OPER3 has priority on OPER4)

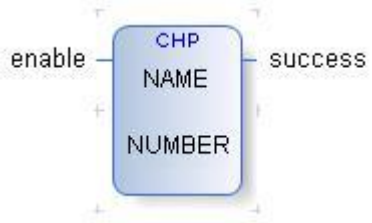
Table 3-53. States found

Syntax:

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

Table 3-54. Syntax of SEQ instruction

CHP - Call the Procedure Module



OPER1 - name of the module to be called
OPER2 - number of the module to be called

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.

Upon adding the instruction, or editing it, a list of procedure modules in the project will be opened, as in the figure bellow:

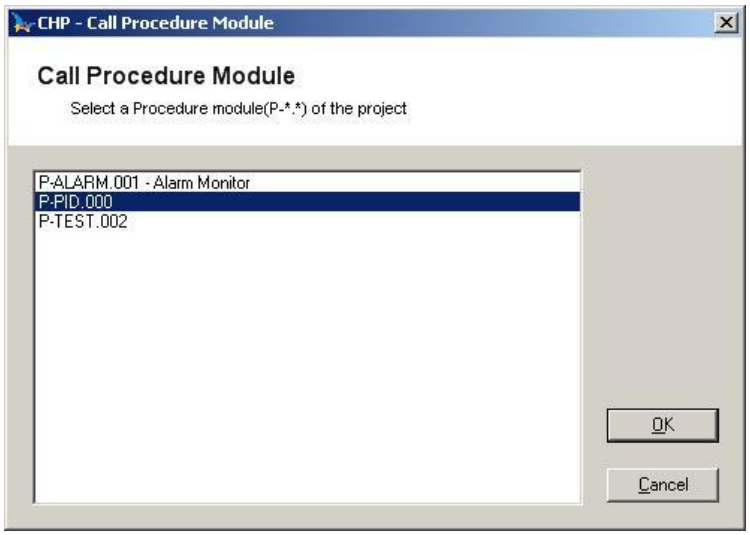


Figure 3-13. List of modules that can be called by the CHP

For selecting a mode, click twice on the module required or select it on the list and click on OK button.

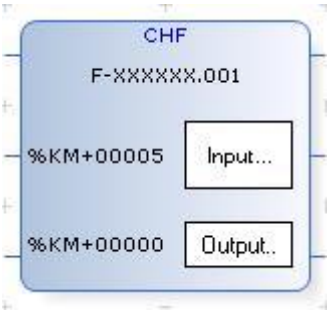
If the called module does not exist, the output **success** is turned off and the execution continues normally after the instruction.

ATTENTION: The module name is not considered by the PLC to the calling, but only its number. If there is a P module with the same number that was called, but with a different name, this module will be executed even so. However MasterTool Extended Edition will warn error on the project verification, and will not send it to the PLC.

Example:

When the instruction input of the CHP is powered, the proceeding module P-Alarm.001 will be called.

CHF - Call Function Module



F-XXXXXX.000 - name of the function module to be called
OPER1 - number of parameters to be sent
OPER2 - number of parameters to return
Input... - list of parameters to be sent
Output... - list of parameters to return

Description:

The instruction the function Module carries out deviation of the processing of the current module to the module specified, 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 CHF. The CHF instruction has the following screen for edition:

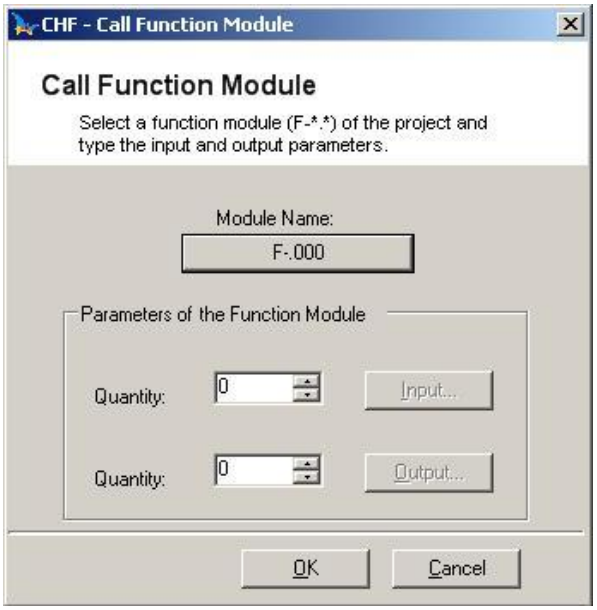


Figure 3-14. Window to fill the CHF instruction

The module function must be selected, present in the current project or a Function Module given by Altus and installed with MasterTool Extended Edition. This selection that must be done at the moment of the insertion or editing of the CHF instruction, clicking on the button with the name of the function module that will be called. So, the following window will be opened:

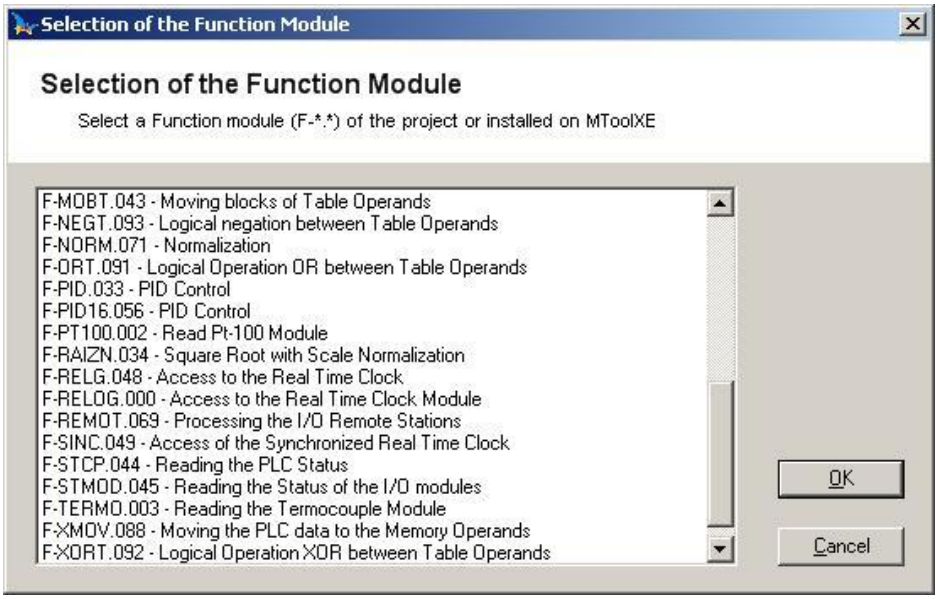


Figure 3-15. Selection list of the function modules to be called by CHF

To select a function mode, click twice on the required module or select it on the list and click on the OK button. If a function module installed on MToolXE was selected, but that is not part of the project, it will be asked if you want to add the module in the current project. This question must be answered with Yes, or the module will not be selected.

During the execution of the instruction on the PLC, if the called module does not exist on the PLC memory, the output **success** is turned off and the execution continues normally after the instruction.

ATTENTION: The name of the module is not considered by the PLC to the call, but only its number. If there is a module F with the same number called but a different name, this module is executed even so. Although MasterTool Extended Edition will warn error on the project verification, and it will not be possible to send it to the PLC.

The CHF instruction, differing from the CHP, allows the parameters passage to the function module called. It can be done in two different manners: a function module distributed by Altus, or a function mode done by the user.

If it is a module distributed by Altus and which has a standard form to be filled in, it is not necessary to define the amount of input and output parameters. In this case, upon clicking on **Input...** or **Output...** buttons a screen correspondent to the called module will be opened, to the edition of parameters, as the following example:

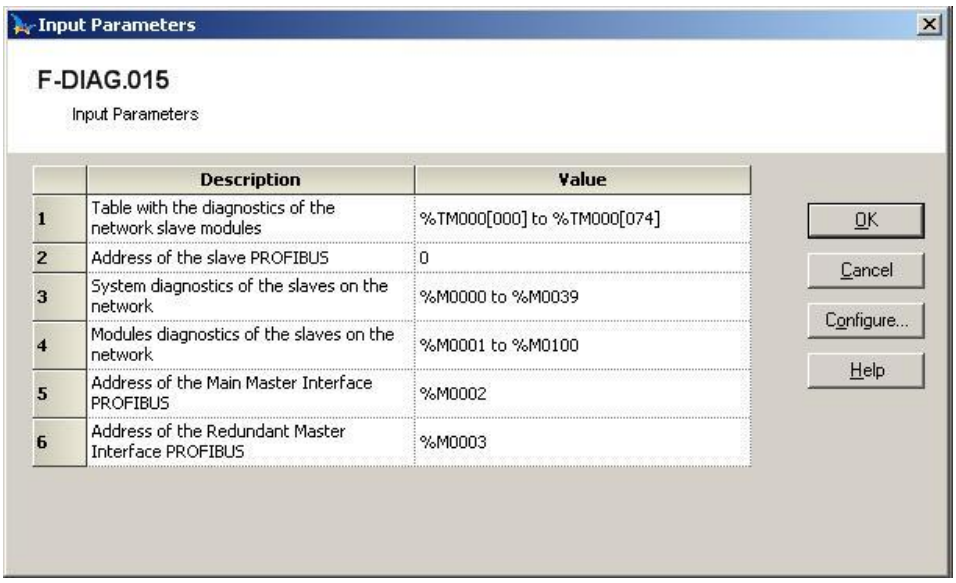


Figure 3-16. Edition of a patterned function module

In this mode, in the edition of the parameters it is informed what is allowed and what is not, through a friendly interface.

Besides some of the modules provided by Altus also can have more of one mode of configuration. And each mode has a different quantity of input parameters. When the user selects a new mode, it must reconfigure the mode. If the new mode has less operand than the previous module, the more operands will be discarded. But if the new module has more operands, the more operands will be configured with the standard operand to the module. And an error of verification will be showed if one of the parameters has different type in the new mode.

Some Function modules have a personalized configuration that is accessed through the bottom **Special Configuration…** The modules that enable this function are introduced in the next section.

But in the case of Function module selected was created by the user, or a standard call to the module does not exist, it will be parametered in a different manner. First the amount of input and output parameters should be defined and after click on **Input…** or **Output…** button**,** according to the necessity. In this case the following window will be opened:

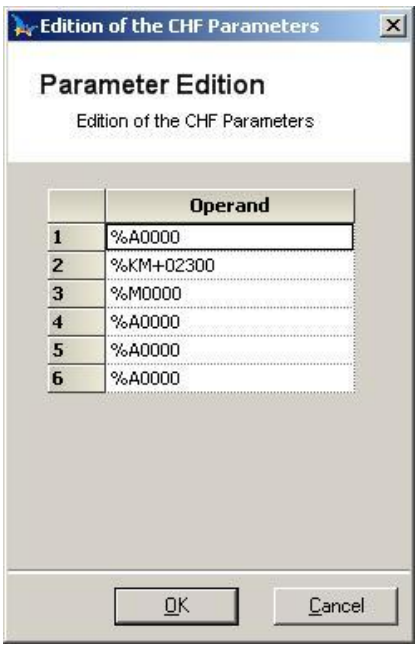


Figure 3-17. Edition of a not patterned function module

WARNING:
MasterTool XE does not realize any consistence related to the operands programmed with parameters, either on the CHF instruction nor on the Function module.

The list of operands to be sent to the module F must have the same operand number with the same type of the ones declared as input parameters module, in order to the copy of its values is correctly achieved. The copy of the operands is achieved in the same order in which they are in the lists. If one of the two lists has less operands in relation to the other, the values of the exceeding operands will not be copied. If the operands are of different types, the copy of the values must happen different by MOV instruction (moving simple operands), then there is no operand type conversion. This principle is valid also to lists of return parameters.

The passing of parameters is realized with copy of the declared operand values (parameters passing by value), although those operands can still be used globally, useful by any module present on the PLC. The F modules can be programmed in a generic form, so that they can be used again in several application programs as new instructions. It is advisable that they use their own operands that were not used for any other module present in the application program, avoiding alterations in operands used in other modules.

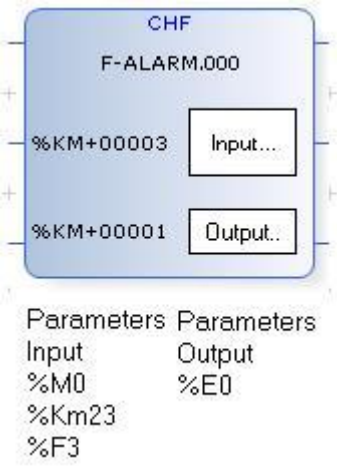
The passage of simple and constant operands to the Function mode is possible. The passage of operands with subdivision to the module Function is not allowed, as %M004.2, %A0021n1, etc. Only simple operands should be used to this passage. The maximum amount of input parameters, or output parameters, is 10.

The following operands can be used as parameters:

Operands Allowed as Parameters	
%TM	%M*M
%TD	%M*D
%TF	%M*I
%TI	%M*F
%M	%M*A
%D	%M*TM
%F	%M*TD
%I	%M*TI
%E	%M*TF
%S	%KM
%A	%KD
%R	%KF
	%KI

Table 3-55. Operands allowed as parameters

Example:



The CHF will call the Function Mode F-Alarm.000, with 3 input parameters (%M0000, %KM0023 and %F0003) and 1 output parameter (%E0000).

CHF - Call Function Module – Special Configuration F-PID16.056

The CHF instruction has a specialized interface of configuration to the module F-PID16.056. This interface, also known as PID skin, provides adjustment and configuration of the PID loop. The window of configuration has three tabs:

- Settings and Charts: monitors variables and configures basic parameters of the PID
- Configuration: allows configuring advanced items of the PID
- Operands: allows configuring the operands used by the module function F-PID16.056

WARNING:
It is necessary to read the documentation of the module F-PID16.056 in order to understand the PID skin correctly.

Tab: Settings & Chart

The Settings & Chart tab is used for parameterize the PID and allows monitoring the principles variables in a graphic representation in real time.

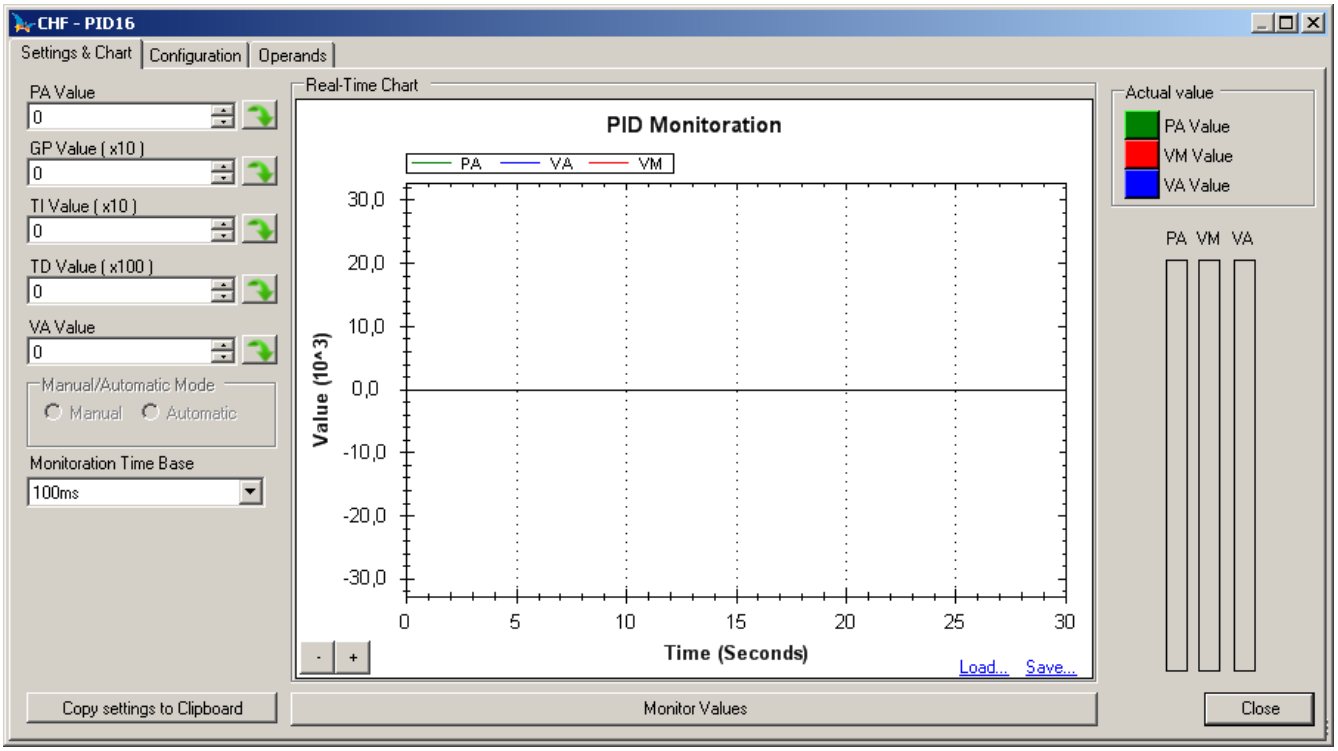


Figure 3-18. Window of the PID skin to the module F-PID16.056, tab Settings & Chart

In the left side in this window, it is possible to configure the variables following:

- PA – point of adjustment
- GP – proportional gain
- TI – integral term
- TD – derivative term
- VA – variable of acting
- Automatic or manual mode
- Direct or reverse mode

The user must digit the new value in text field in order to configure the values of PA, GP, TI, TD and VA. Thus the monitoration of the variable will stay stopped and this variable will stay in red color itself, until the user acknowledges dispatch of the value to the PLC by the bottom on the right of the text field. If the user closes the window, the values edited, are not sent, are discarded.

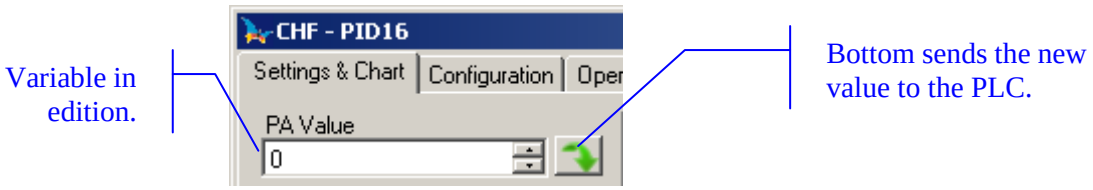


Figure 3-19. Edition of a variable of the PID

Values updated in this window are dispatched to the PLC only. If the PID loop has ladder instructions configuring the variables, they are not updated automatically. Then the user must configure this instruction manually. In order to make easy this action, the bottom Copy settings to clipboard copies the values to utilization of CAB of initiation of the table operand of control.

The interval while variables are monitored can be selected through the option "**Monitoration Time Base**". The selection of a bigger time base allows monitoring the trap by a long time. The PID skin stores the last 10,000 monitorations.

It is necessary only a click to configure the Automatic, Manual, Direct and Reverse modes and dispatch the command for the PLC.

WARNING:
The Automatic/Manual modes and the Direct/Reverse modes are enable to the module F-PID16.056 since version 1.10 only and if the CHF is configured to 7 parameters. Details about the version 1.10 and the seventh parameter can be seen in the documentation of the module F-PID16.056 in the help of the function modules.

WARNING:
The configuration of the values does not change the ladder program of the user. If the user is configuring the values through ladder instructions, these instructions will not changed.

In the central area, it is possible to visualize the values of PA, VM and VA in the graphic representation of Settings & Chart in real time. It is necessary to press the bottom Monitor Values in order to initialize the monitoration. This bottom also allows stopping the monitoration. The graph can be handled using mouse and keyboard as in the following table:

Command	Action
Left bottom	Zoom on the selected area.
Left bottom + SHIFT key or Middle bottom	Moves the graph horizontally and vertically.
Right bottom	Menu of context with options: • Un-Zoom: undoes the last zoom. • Undo All Zoom Pan: returns the graph to the initial values of zoom and axes.

Table 3-56. Commands of mouse and keyboard to Settings & Chart tab

It is possible to save the values in monitoring at the last five minutes pressing the bottom **Save**. In order to load values, it is necessary to press the bottom **Load** and to select the file.

It is possible to monitor the values of PA, VM and VA in the right side of the window. As well to monitor those values in percent scale through the graphic representation of the vertical bar. The scale of the bar graphs begins from 0 until 30,000.

Tag: Configuration

The tag Configuration is used to configure the advanced parameters of PID.

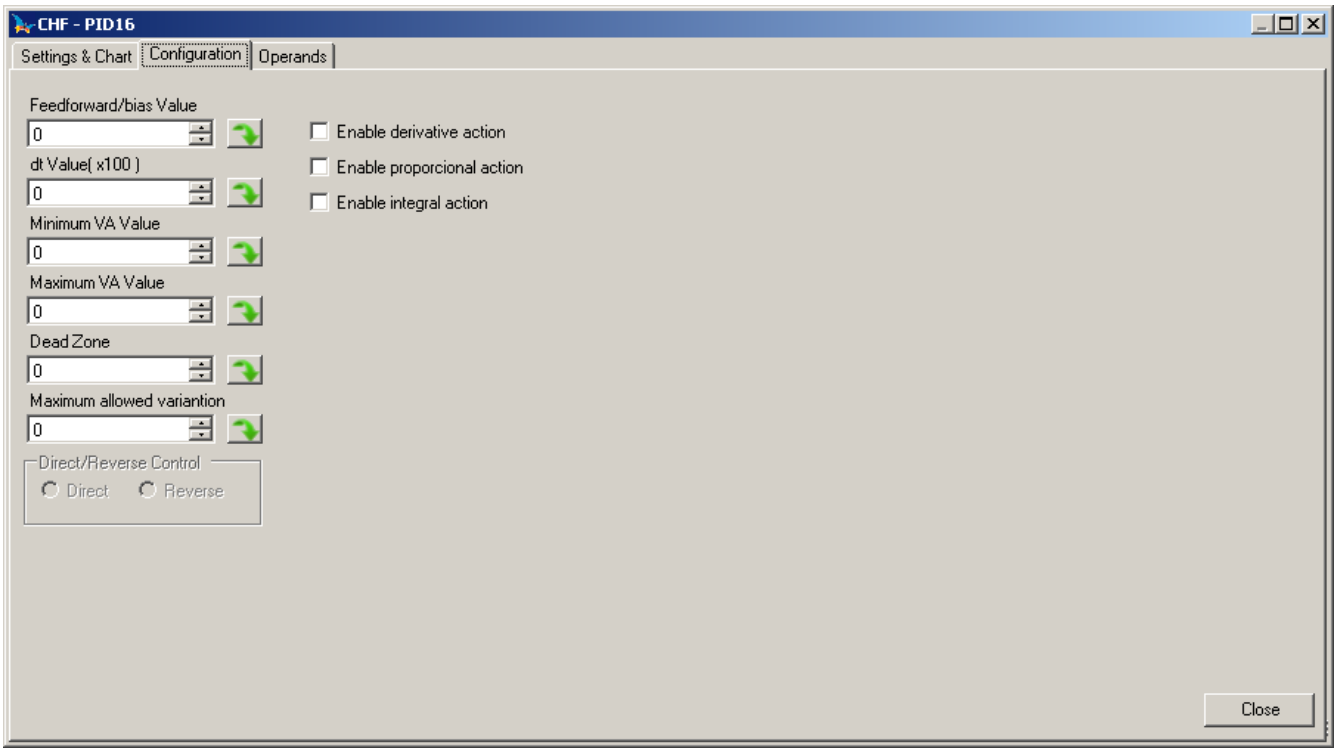


Figura 3-20. Window of the module F-PID16.056, tab Configuration.

The values that can be configured are:

- Feedfoward/bias value
- dt value
- Minimum VA value
- Maximum VA value
- Dead zone
- Maximum allowed variation
- Direct or Reverse Mode
- Enable derivation action
- Enable proportional action
- Enable integral action

The user must digit the new value within the text box in order to configure those values. Thus the monitoration of the variable will stay stopped and this variable will stay in red color itself, until the user acknowledges dispatch of the value to the PLC by the bottom on the right of the text field. If the user closes the window, the values edited, are not sent, are discarded. The enabling commands of action are sent immediately.

Tab: Operands

In the tab Operands, it is possible to configure the operands used by the module F-PID16.056. Changes on this window change the parameters loaded in the module F-PID16.056. This window edits the following entry parameters of the CHF to the module F-PID16.056. The configuration of operands is enabled only when the monitoration is stopped.

CHF Entry Parameters	Name	Type of operand accepted
1	Control operand	%TM
2	VM operand	%M
3	PA operand	%M
4	VA operand	%M
5	Feedvorward/bias operand	%M
6	Control octet	%A
7	Control operand	%M

Table 3-57. Relation of the operands with the configuration of the CHF.

The configuration of the operand of control in the seventh parameter is possible only since version 1.10 of the module F-PID16.056. When the seventh parameter is enabled, it is possible to configure the parameters of Manual and Automatic, and Direct and Reverse from the PID skin. If the user is not using this version, it is available at the link: "**Click here to update the module to the newest version and enable the control operand.**". This link is available only since version 3.05 of the Function Modules (see menu Help/About to consult the version of the Function Modules).

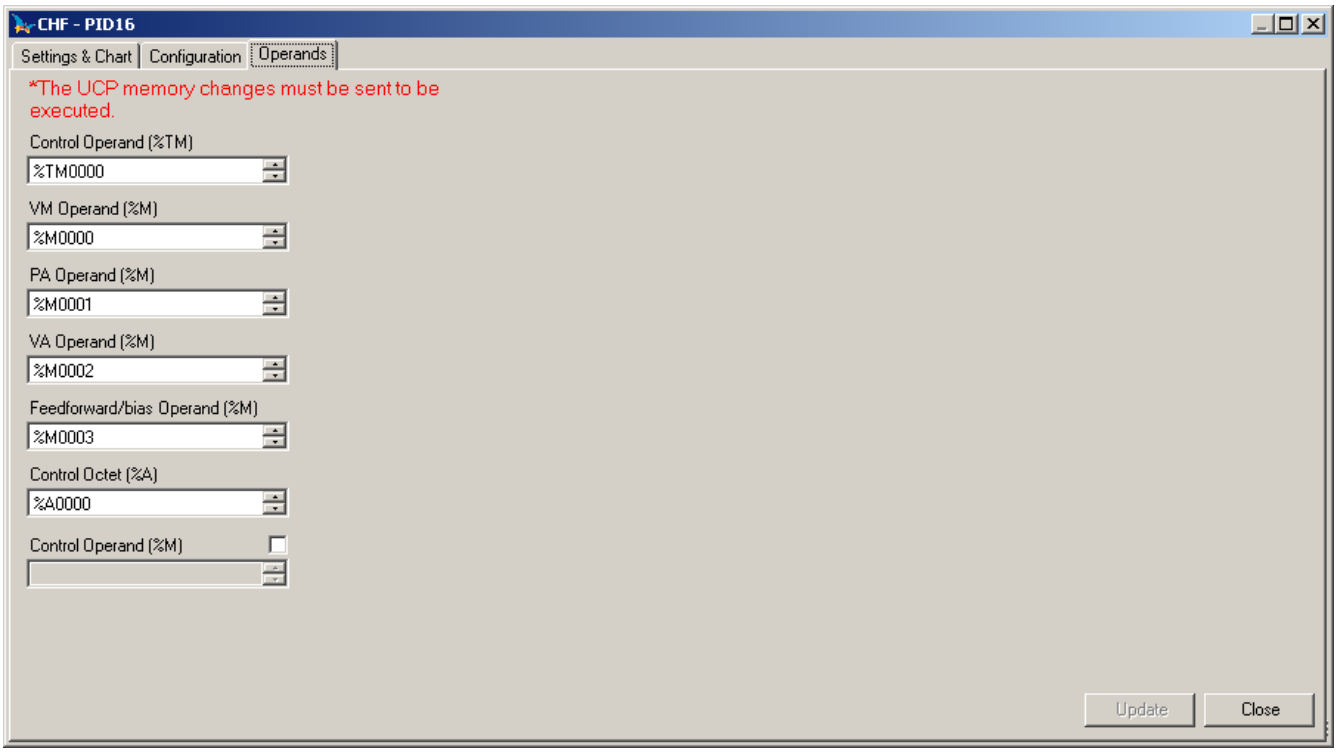


Figura 3-21. Window of the module F-PID16.056, tab Operands.

If the function module used is from version 1.10 or upper, the user will can select whether it wants to utilize the seventh parameter of the function.

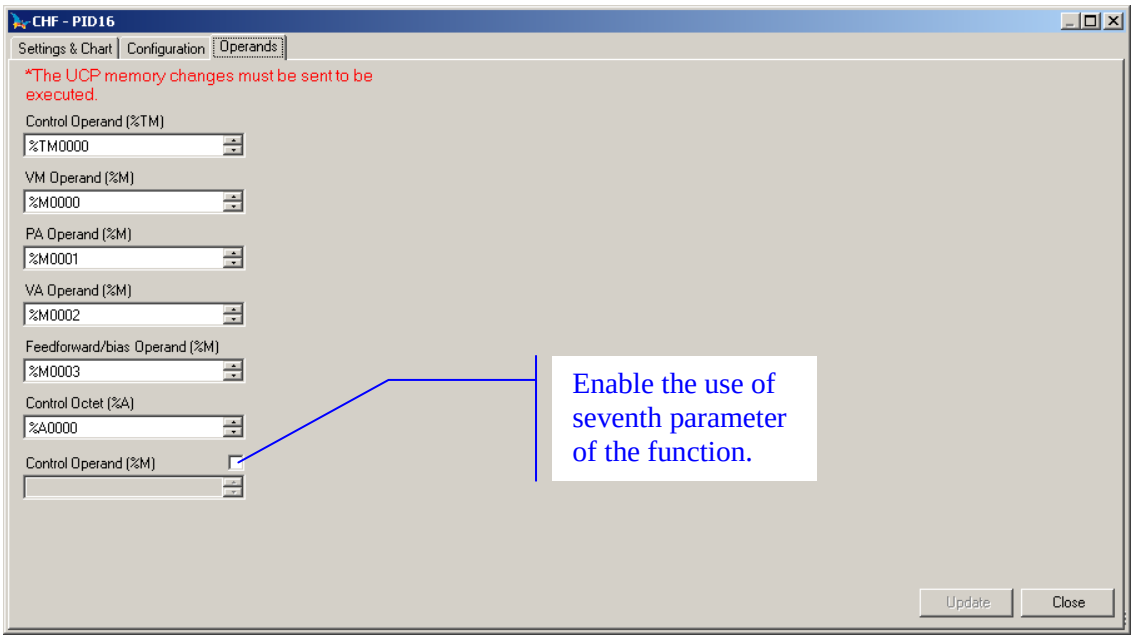


Figure 3-22. Window of the module F-PID16.056, tab Operands, since version 1.10

WARNING:
Some alteration on this window only will validate after sending of ladder module that has the CHF instruction to the PLC.

The window of configuration must be utilized to the *online* configuration of the PID. If a communication error occurs, an exclamation mark will be showed upon of the variable, as following:

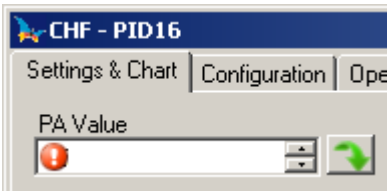


Figure 3-23. Variable with communication problem

CHF - Call Function Module – Special AL-2752

The CHF instruction also has a special configuration to the module F-2005.016, when this module is configured to use Function 100 PID Loops AL-2752. AL-2752 is a program executed by AL-2005 that calculates until 100 PID loops. The PID skin to AL-2752 is alike to PID skin to the function F-PID16.056, with few differences.

WARNING:
It is necessary to read the User's Manual of AL-2752 Function 100 PID loops in order to understand the special mode of the CHF.

To enable the use of PID skin to AL-2752, the CHF instruction must be configured correctly to the use with AL-2752. In other works:

- Must there be one CAB instruction configuring the table of configuration of AL-2752 (according to the entry parameter of the CHF instruction)
- In the CAB instruction, the position 2 must be with value 2752 (code to AL-2752)
- In the CAB instruction, the position 4, which denotes the number of PID loops, must be with value between 1 and 100

The user will must select which loop it intends to configure as soon as it opens the window.

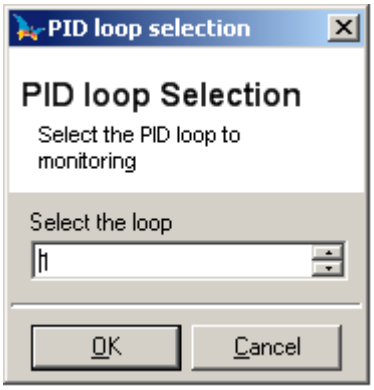


Figure 3-24. Window of selection of the PID loop

Tab: Settings & Chart

The tab Settings & Chart has the same functionalities described in the PID skin to the module F-PID16.056. But it has not a bottom to configure the CAB instruction of the PID loop, whereas this resource is exclusive of this module F.e

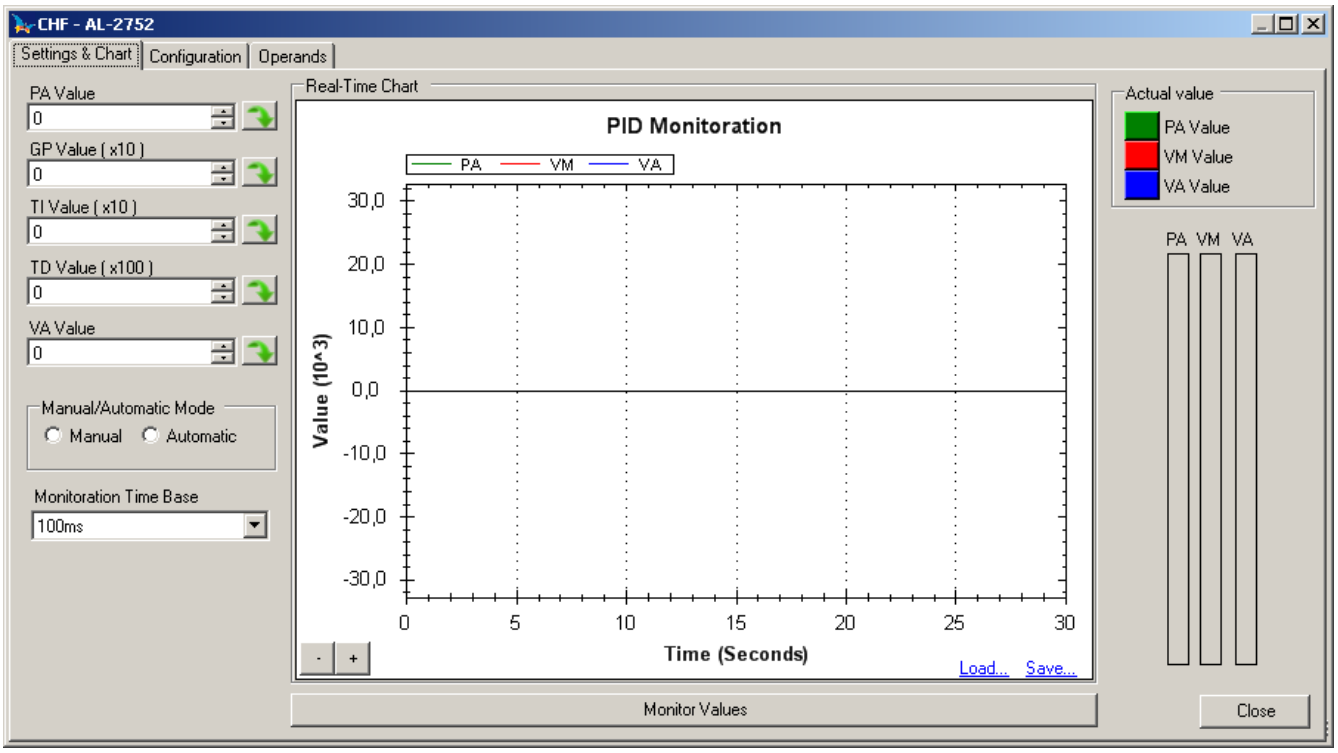


Figure 3-25. Window of the AL-2752, tab Settings & Chart

Tab: Configuration

The tab Configuration is used to configure the advanced parameters of the PID.

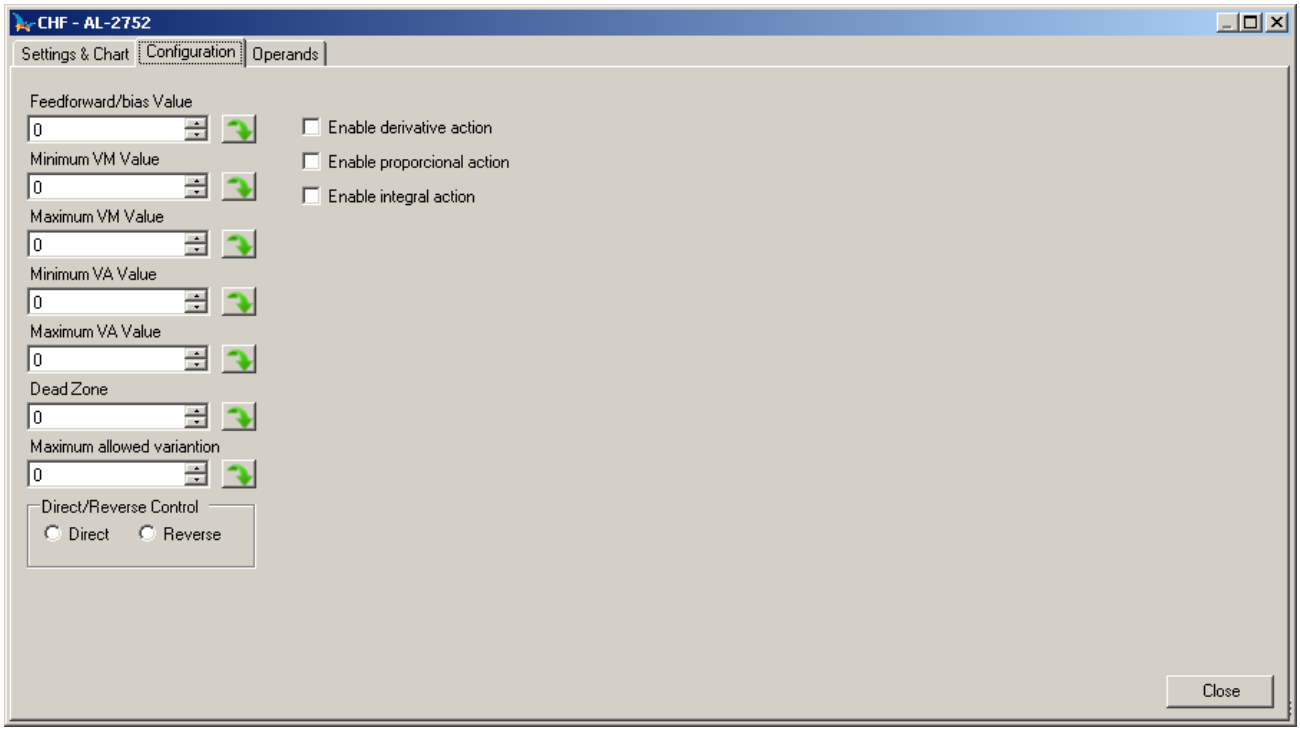


Figure 3-26. Window of the AL-2752, tab Configuration

The values that can be configured are:

- Feed forward/bias value
- Minimum VM value
- Maximum VM value

- Minimum VA value
- Maximum VA value
- Dead zone
- Maximum allowed variation
- Direct/reverse control
- Enable derivative action
- Enable proportional action
- Enable integral action

The user must digit the new value in the text field in order to configure those values. Thus the monitoration of the variable will stay stopped and this variable will stay in red color itself, until the user acknowledges dispatch of the value to the PLC through the bottom on the right of the text field. If the user closes the window, the values edited, are not sent, are discarded. The enabling commands of action are sent immediately.

Tab: Operands

In the window of the PID skin to AL-2752, it is not possible to configure the operands of the PID loop, but it is allowed to visualize which operands are used to the variables of the loop.

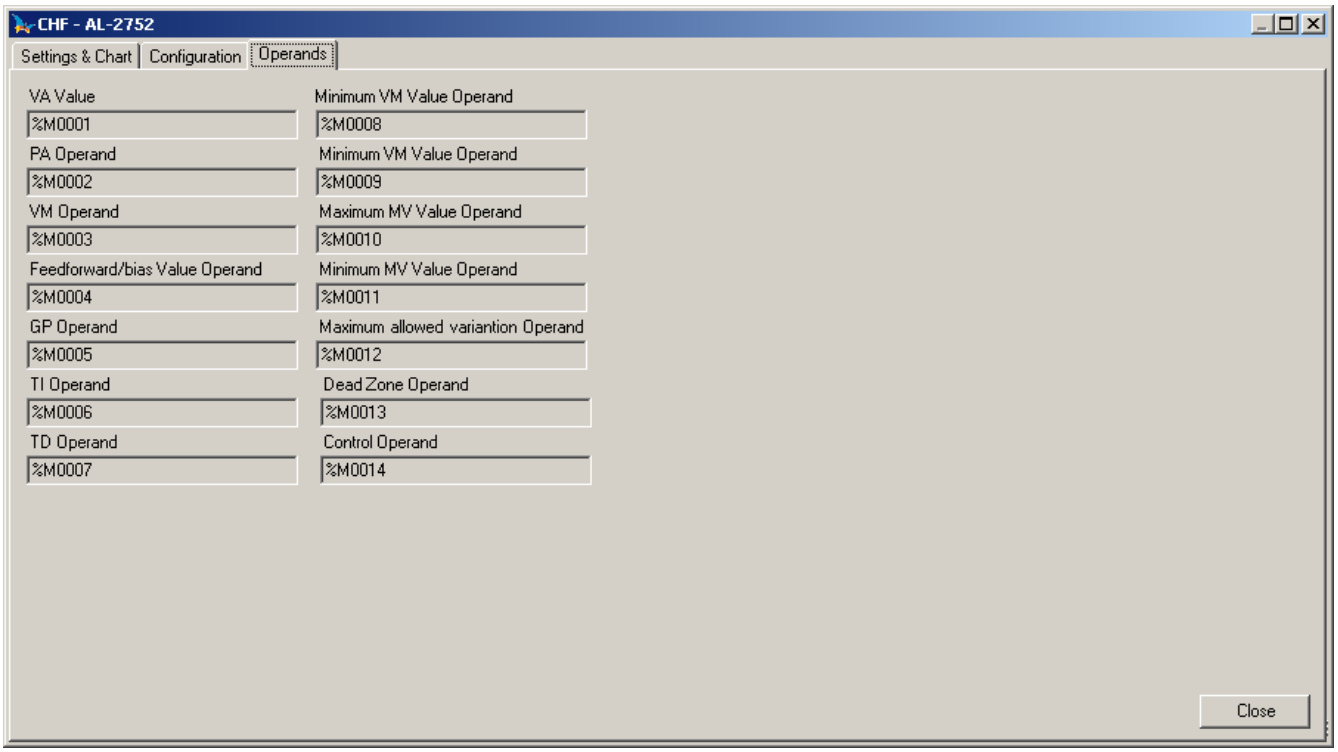


Figure 3-27. Window of the AL-2752, tab Operands

ECR - Writing of operands in other PLC

OPER1 - node address of the remote controller
OPER2 - sub-net address of the remote controller
OPER3 - instruction control operand

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 ALNETII communication network. For its use, therefore, it is essential that the controller who executes is connected to other PLCs through ALNETII.

Through the ECR 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. The ECR can be programmed to be priority, sending an urgent communication, processed by the "bridges" and by the destination PLC before the common communications. The priority ECR allows only one communication, being useful to sign alarms or emergency situations among PLCs.

To program the instruction, it must be declared in the first and second cells (OPER1 and OPER2) the node addresses of the programmable destination controller that will receive the written values. Those operands are programmed as constant type memory (%KM) and have the same address meaning of the ones configured on the options **Communication, Address, Sub-net and Communication, Address, Node.**

The following table shows the possible values to node and sub-net addresses:

Sub-net	Node	Comunication Type
001 to 063	001 to 031	ALNET II with one node
064	xxx	IP Address, where xxx specifies other IP address in the same sub-net.
100	001 to 015	ALNET II with multicast groups in all sub-nets
101 to 163	001 to 015	ALNET II with multicast group in one specified sub-net
200	xxx	ALNET II with broadcast to all sub-nets
201 to 263	xxx	ALNET II with broadcast to one specified sub-net

Table 3-58. Address of node and sub-net

The sub-net address between the values 001 and 063 indicates that the communication is carried out by using the ALNETII networks, and that it is designated to an only node indicated in the node option.

The sub-net address 100 indicates that the communication is carried out by ALNETII network and designated to all nodes of all sub-nets that belong to the multicast group specified on node option (global multicast).

The sub-net address 101 to 163 indicates that the communication is carried out by ALNETII network and designated to all nodes of all sub-nets indicated on option Sub-net less than 100 that belong to the multicast group specified on node option (global multicast).

The sub-net address 200 indicates that the communication is carried out by ALNETII network and designated to all nodes of all sub-nets (global broadcast). The value specified on node option is not relevant in this option.

The sub-net address among 201 and 263 indicates that the communication is carried out by ALNETII network and designated to all nodes of the sub-net indicated on the option Sub-net less than 200 (local broadcast). The value specified on node option is not relevant in this option.

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

WARNING:
The %D operand programmed on OPER3 cannot have its value modified in any other point of the application program for the appropriate functioning of the ECH. Consequently, each new instruction ECH, LTR, CEH or inserted LTH in the application program must use an %D operand different from the others. This operand cannot be retentive.

To carry out the editing of the ECR parameters, select the PLC… button. A window for edition of the instruction messages will be opened, as shown in the following figure:

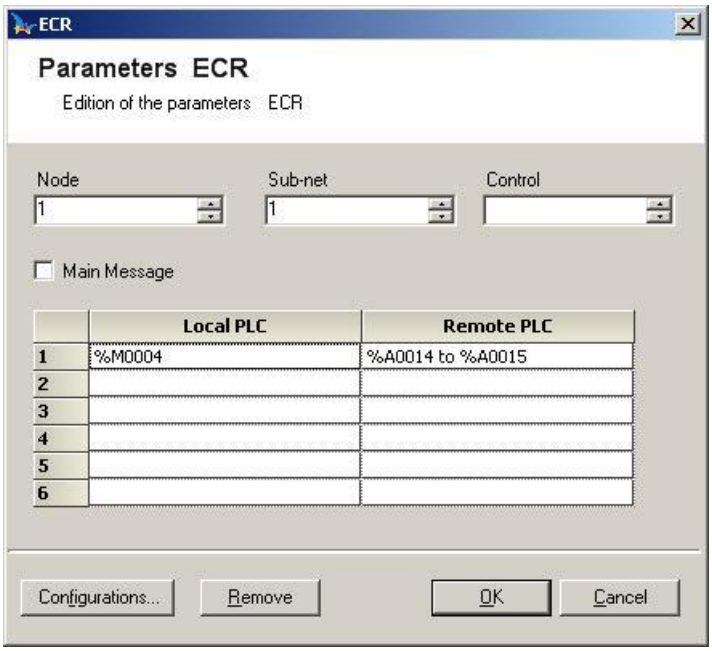


Figure 3-28. Parameters ECR

In this window, there is a table where each line represents on of the six possible communications between Local PLC and Remote PLC. In this table, the following columns are presented:

- **Local PLC:** group of operands whose values will be sent to the Remote PLC through instruction.
- **Remote PLC:** group of operands from the Remote PLC where the value coming from the instruction communication will be written.

When a line does not have any information, a communication will not be carried out. So, in the figure above, only two communications would be done by the instruction.

The **Priority Message** item allows the edition of a priority ECR whenever it is selected. In this case, only one communication will be allowed, and only the first line remains on the table.

ATTENTION:
In the verification of the MasterTool Extended Edition project, the specified operand to the local PLC are consisted by MasterTool according to the declarations of the module C, as they are part of the application program in use. However, the operands declared to the remote PLC does not suffer consistence related to their type and addresses, as they belong to an application program of another programmable controller.

For editing or adding a communication in the instruction, click twice on the correspondent line, or select the line and click on Configuration… button. The following messages edition window will be opened:

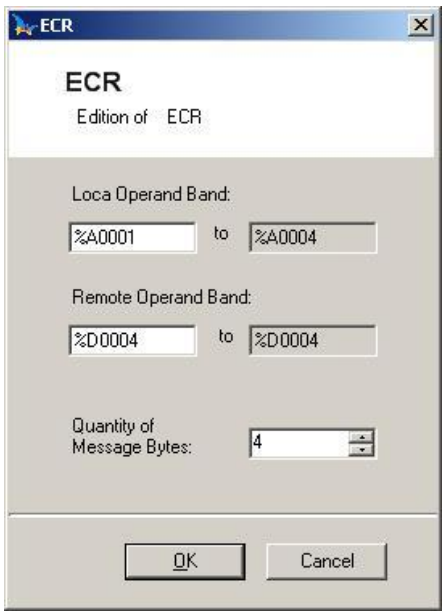


Figure 3-29. Message edition of local PLC to remote PLC

In this window, it must be informed the initial operand of the local PLC and the initial operand of the Remote PLC. The amount of bytes used in the communication also must be informed, within the limited of 220bytes. The value of this amount of bytes will be multiple of the greater amount of bytes in a single operand of the Local or Remote PLC (Check Operands in this same manual). In the figure above, the Multiple Number is 4 as the decimal operand (%D) has four bytes, while the auxiliary operand (%A) has only 1.

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

Operands Allowed as Parameters
%M
%D
%F
%E
%S
%A
%TM[x]
%TD[x]
%TF[x]
%TI[x]

Table 3-59. Operands allowed as parameters

Enabling the **enable** input the communication of the first present writing in the ECR is turned on, 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 ECR is not powered, with the application of a pulse with duration of one verification in the **error** output in case any communication could not be carried out.

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.

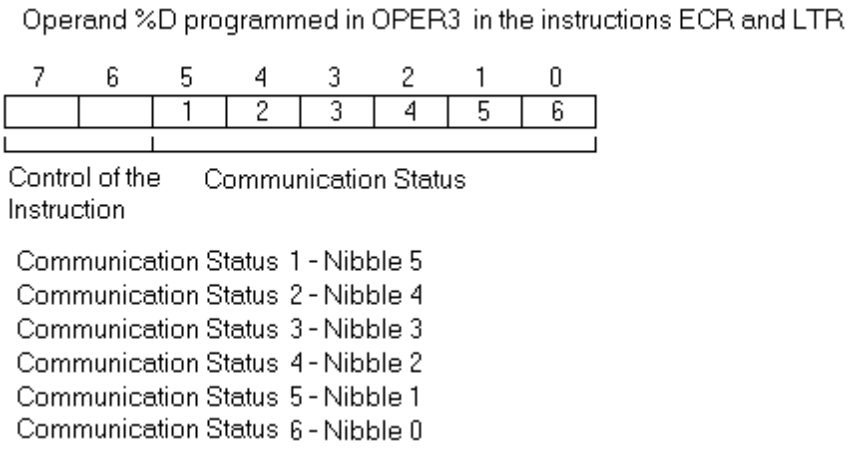


Figure 3-30. Control operand of ECR and LTR instructions

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

- 0 - communication with success
- 1 - non defined operand
- 2 - local controller address equals to the remote (communication with the PLC)
- 3 - invalid operand block
- 4 - invalid operand type
- 5 - timeout of package transmission
- 6 - no space on the transmition queue
- 7 - lack of transmition buffer
- 8 - solicitation timeout
- 9 - hardwares error
- 10 - protected remote PLC

In summary, executing an ECR instruction all the existing communications are carried out, even if the enabling input is not powered. When all the writings are completed, the next ECR or LTR instruction found in the application program with the **enable** input powered becomes active, starting to process its communications.

WARNING:
The application program can neither carry out jumps on the active ECR instruction nor stop executing the module that contains it, to assure its correct processing.

In an application program being executed in the PLC, only one instruction of access to the ALNET II network (ECR or LTR) is considered active, even if some other instructions with input **enable** exist. The **busy** output determines what is the active instruction, being able to be used to synchronize the communications with the application program. To prevent overloads in the traffic of information in the net, it is advised turn on the ECR instructions periodically, preventing to permanently keep its enabled in the applicatory program, if possible. A recommended procedure is disconnect the **enable** input after the **busy** output is powered, preventing a new enabling of the instruction after its ending.

The priority ECR does not follow the processing order of the non priority ECRs, being processed transmitting their data as fast as possible, upon being enabled. For this reason a priority ECR must not be permanently enabled, and should be used only in alarm situations or periodically, or it can prevent the other program ECRs from carrying out their communications or use up the reception buffer of the destination PLC.

If the instruction is programmed specifying that the node address is equal to the address of the controller that executes it (written of values), the **error** output is powered.

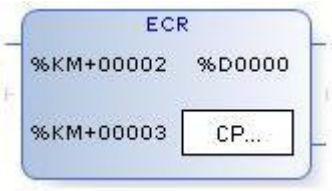
If any operand has been defined on OPER4, the **error** and **busy** output are not powered.

Syntax:

OPER1	OPER2	OPER3	OPER4
%KM	%KM	%D	COMUNICAÇÕES

Table 3-60. Syntax of ECR instruction

Example:



Low priority on messages content.

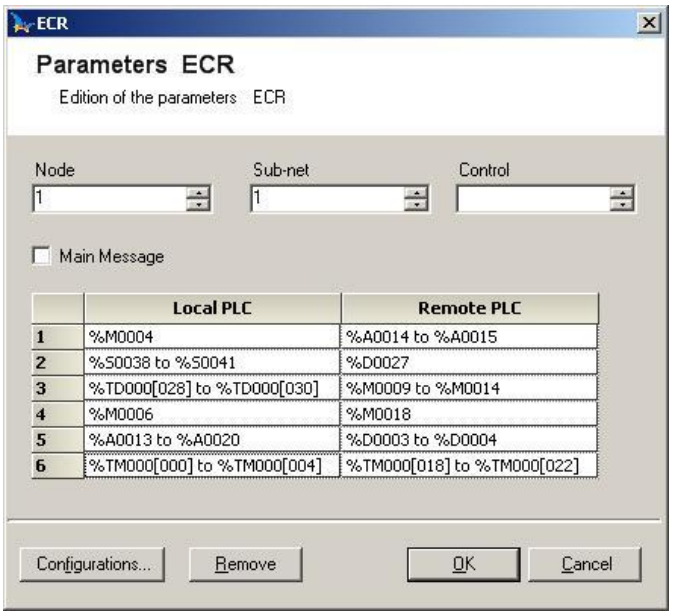


Figure 3-31. ECR parameters configuration with message not priority

This instruction carries out writing in the programmable controller with node address 2 on sub-net 1. Six communications are defined, transferring data of different types between the PLCs. Communication 0 sends the content of a memory operand in the local PLC for two auxiliary operands in the remote PLC, being transferred 2 octets. Communications 1, 2, 3, 4 and 5 transfer, respectively, 4, 12, 2, 8 and 10 octets between the controllers.

Content of ECR main messages:

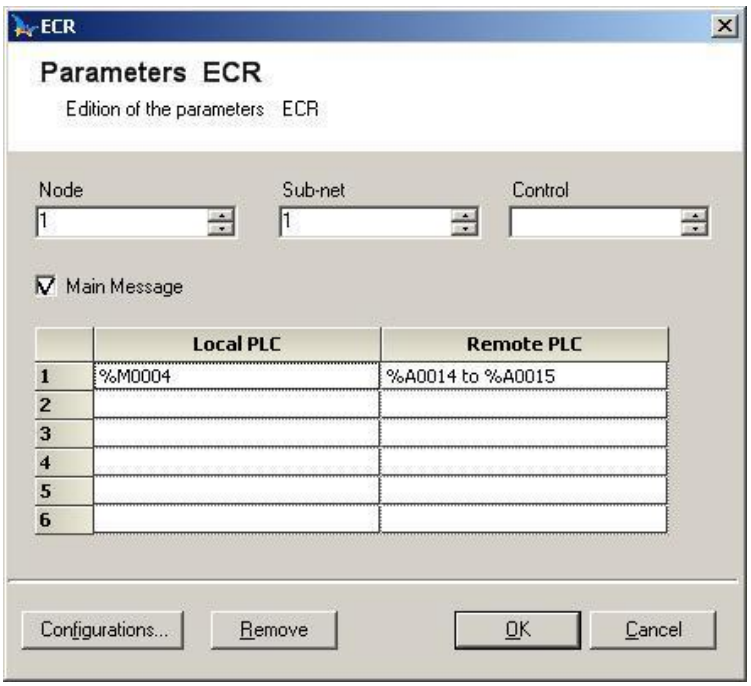


Figure 3-32. ECR parameters configuration with message priority

This instruction carries out writing on the programmable controller with the node address 1 on sub-net 1. A priority communication is defined to it. The P/1 communication sends the content of a memory operand in the local PLC to two auxiliary operands in the remote PLC, being 2 octets transferred.

ATTENTION: This instruction can only be used on PLCs AL-2004.

LTR - Reading of Operands from Another PLC

OPER1 - node address of the remote controller
OPER2 - sub-net address of the remote controller
OPER3 - operand of the instruction control

Description:

This instruction carries out the reading of values of operands presented in other programmable controllers for operands of the programmable controller where it is being executed, through the ALNET II communication net. For its use, therefore, it is essential that the PLC that executes it is connected to other PLCs by ALNET II.

Through the LTR 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 LTR is identical to the ECR, observing the same restrictions. In the LTR, the transference of the values occurs from the declared operands in the remote PLC to the local PLC , being this the only difference between them.

WARNING:
The LTR differs from ECR in the possibility of priority messages, it means, it is not possible to edit a priority LTR.

Syntax:

OPER1	OPER2	OPER3	OPER4
%KM	%KM	%D	COMUNICAÇÕES

Table 3-61. Syntax of LTR instruction

Example:

LTR Messages content:

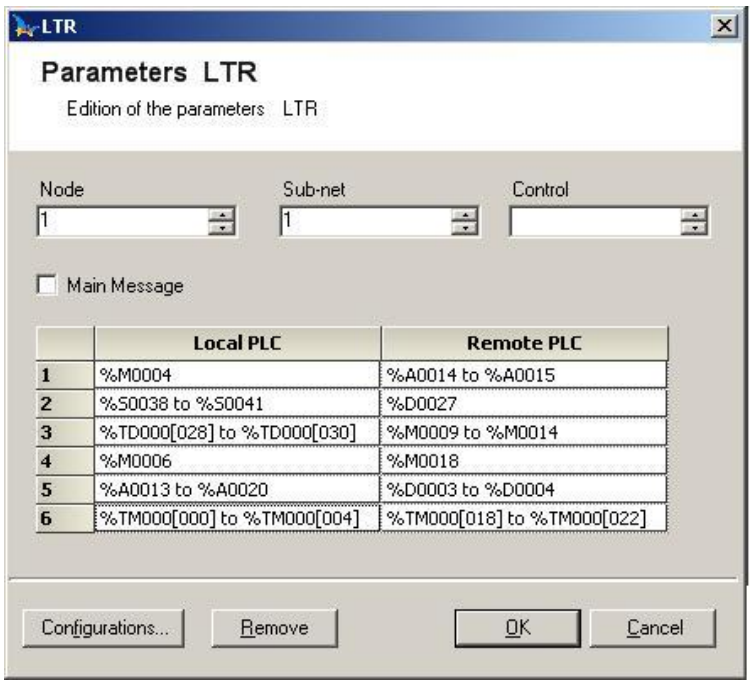
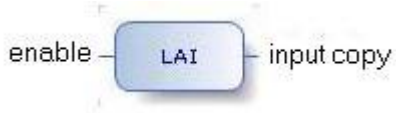


Figure 3-33. LTR parameters configuration

This instruction carries out readings in the programmable controller with the node address 2 in the sub-net 1. Six communications are defined for it, transferring data of different types between the PLCs. Communication 0 reads the content of two auxiliary operands in the remote PLC for one memory operand in the local PLC, being transferred 2 octets. Communications 1, 2, 3, 4 and 5 transfer, respectively, 4, 12, 2, 8 and 10 octets between the programmable controllers.

ATTENTION: This instruction can only be used in the AL-2004.

LAI - Free Updating of Operand Images



Description:

The instruction LAH carries through the processing of the hanging communications of the ALNETII net for the local PLC.

When returning for the processing of executive software, on each verification end, the PLC processes the solicitations of reading and other services that have been requested to it by other PLCs in the net, during the execution of the application program.

The programmable controller has a 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 application program have relatively high time of execution and the programmable controller receives many service requests of the net, can occur the situation that the PLC cannot take care of it, arriving at the limit of 32 hanging communications waiting for processing. In this case, the PLC returns a reply to the one that requests indicating the impossibility to take care of its communication.

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

WARNING:
The values of the application program operands can be modified after the execution of a LAI, as another equipment plugged to the net can request to write on it. It must be considered the influence of this fact if inserting this instruction in the application program.

ATTENTION:
This instruction can only be used in PLCs AL-2004.

ECH – Writing of Operands in Another PLC to Ethertnet

IP - IP address of the remote controller
OPER1 - operand of instruction control

Description:

This instruction carries out the writing of operand values of the controller where it is being executed in operands present in other PLCs, through Ethernet communication net. To its use, it is necessary that the controller that executes it is connected to other PLCs through Ethernet.

Through ECH individual operand values or set of operands values can be transferred, being possible to program up to 6 different communications in the same instruction.

To program the instruction, the IP address of the destination programmable controller that will receive the written values must be declared. In OPER1 a decimal operand (%D) must be declared to be used for the instruction in the control of its processing.

WARNING:
The operand %D programmed in OPER1 cannot have its value modified in any other point of the application program for the proper working of the ECH. Consequently, each new instruction ECH, LTH, ECR or LTR added to the application program must use a %D operand different from the others. This operand cannot be retentive.

The ECH instruction has the same behavior and syntaxes of the ECR instruction, related to control operand and edition of messages. The basic difference is the net where the instruction works, that in this case is Ethernet. So, for further details about how to use the operands of this instruction, check ECR instruction.

LTH – Reading of Operands from Another PLC to Ethernet

IP - endereço IP do controlador remoto
OPER1 - operando de controle da instrução

Description:

This instruction carries out the reading of operand values in other programmable controllers to operands of the programmable controller where it is being executed, through Ethernet communication net. To its use it is necessary that the PLC that executes it is connected to other PLCs through Ethernet.

To program the instruction, the IP address of the destination programmable controller that will receive the written values must be declared. In OPER1 a decimal operand (%D) must be declared to be used for the instruction in the control of its processing.

WARNING:
The operand %D programmed in OPER1 cannot have its value modified in any other point of the application program for the proper working of the LTH. Consequently, each new instruction ECH, LTH, ECR or LTR added to the application program must use a %D operand different from the others. This operand cannot be retentive.

Through LTH individual operand values or set of operands values can be transferred, being possible to program up to 6 different reading communications in the same instruction.

The programming of LTH instruction is identical to LTR, observing the same restrictions. The only difference is that this instruction is used in Ethernet networks.

LAH – Free Updating of Operand Images to Ethernet

Description:

The instruction free updating of operand images carries out the processing of hanging communications from Ethernet network to local PLC.

Upon returning to the processing of executive software, in the end of each verification, the PLC processes the reading requests and other services that have been requested to it for other PLCs present in the network, during the execution of the application program.

The programmable controller has a 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 application program have relatively high time of execution and the programmable controller receives many service requests of the net, can occur the situation that the PLC can not take care of it, arriving at the limit of 32 hanging communications waiting for processing. In this case, the PLC returns a reply to the one that requests indicating the impossibility to take care of its communication.

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

WARNING:
The values of the application program operands can be modified after the execution of a LAH, as another equipment plugged to the net can request to write on it. It must be considered the influence of this fact if inserting this instruction in the application program.

Instructions of the Connections Group

The Instructions of the connection group allow the constructions of series and parallel ways as well as the inversion of the signal

Name	Name Description	Edition Sequence
LGH	Horizontal Connection	Alt, I, L, H
NEG	Denied Connection	Alt, I, L, N
LGV	Vertical Connection	Alt, I, L, V

Table 3-62. Instructions of the connections group

LGH – Horizontal Connection

NEG – 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.

.

4. Glossary

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.

Jumpers Small connector to shortcut pins located on a circuit board. Used to set addresses or configuration.

Algorithm Finite and well defined sequence of instructions with the goal to solve problems

Altus Relay and Blocks Language Set of rules, conventions and syntaxes used when building an application program to run in an Altus PLC.

Application Program Program downloaded into the PLC and has the instructions that define how the machinery or process will work.

Arrestor Lightning protection device using inert gases.

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. It is not controlling the system, but is ready to take control if the main CPU fails.

Bit Basic information unit, it may be at 1 or 0 logic level.

BT Battery test.

Bus Set of electrical signals that are part of a logic group with the function of transferring data and control between different elements of a subsystem

Byte Information unit composed by eight bits.

C-Module See Configuration Module.

Commercial Code Product code formed by the letters PO and followed by four digits.

Commissioning Final verification of a control system, when the application programs of all CPUs and remote stations are executed together, after been developed and verified individually.

Configuration Module Also referred to as 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 I/O modules in the bus

CPU Central Processing Unit. It controls the data flow, interprets and executes the program instructions as well as monitors the system devices.

Diagnostic Procedures to detect and isolate failures. It also relates to the data set used for such tasks, and serves for analysis and correction or problems.

E2PROM Electrically Erasable Programmable Read-Only Memory. Non-volatile memory that may be electrically erased by the electronic circuit.

E-Module See Execution Module

Encoder Normally refers to position measurement transducer.

EPROM Erasable Programmable Read Only Memory. Memory for read only that may be erased and programmed out of the circuit. The memory doesn't loose its contents when powered off.

ER Acronym used on LEDs to indicate error

ESD Electrostatic Discharge.

Execution Module Application program modules. May be one of 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.

Firmware 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 electrically erased and programmed..

F-Module See Function Module.

FMS Fieldbus Message System.

Function Module Application software module called from the main module (E-module) or from another function module or procedure module. It passes parameters and return values. Works as a subroutine.

Hardkey Connector normally attached to the parallel port of a microcomputer to avoid the use of illegal software copies

Hardware Physical equipment used to process data where normally programs (software) are executed

I/O See Input/Output.

I/O Module Hardware module that is part of the Input/Output (I/O) subsystem.

I/O Subsystem Set of digital or analog I/O modules and interfaces of a PLC

IEC 61131 Generic international standard for operation and use of programmable controllers.

IEC Pub. 144 (1963)	International standard for protection of accidental access and sealing the equipment from water, dust and other foreign objects.
IEC-536-1976	International standard for electrical shock protection.
IEC-801-4	International standard for tests of immunity against interference by pulses burst
IEEE C37.90.1 (SWC)	SWC stands for Surge Withstand Capability. This is the international standard for oscillatory wave noises protection.
Input/Output	Also known as I/O. Data input or output devices in a system. In PLCs these are typically the digital or analog modules that monitor or actuate the devices controlled by the system.
Interface	Normally used to refer to a device that adapts electrically or logically the transferring of signals between two equipments.
Interruption	Priority event that temporarily halts the normal 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 peripheral, while the later is caused within a program
ISOL.	Acronym used to indicate isolation or isolated.
kbytes	Memory size unit. Represents 1024 bytes.
LED	Light Emitting Diode. Type of semiconductor diode that emits light when energized. It's used for visual feedback.
Logic	A graphic matrix in Altus Relay and Blocks Language where are inserted the relay diagram language instructions that are part of an application program are inserted. A set of sequentially organized logics makes up a program module.
MasterTool	The Altus WINDOWS® based programming software that allows application software development for PLCs from the Ponto, PX, Grano, Piccolo, AL-2000, AL-3000 and Quarks series. Throughout this manual, this software is referred by its code or as MasterTool Programming.
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 with 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 with other modules through information sharing by parameters.
Module address:	Address used by the CPU in order to access a specific I/O module.
Nibble	Information unit composed of four bits.
Not-operant CPU	In a redundant system this is the CPU that is neither active nor backup. May not take control of the system.
Operands	Elements on which software instructions work. They may represent constants, variables or set of variables.
PA	See Jumpers.
PLC	See Programmable Controller.
P-Module	See Procedure Module.
Procedure Module	PLC application software module called from the main module (E-module) or from another procedure module or function module that does not have parameters.
PROFIBUS PA	Means PROFIBUS Process Automation.
Programmable Controller	Also know as PLC. Equipment controlling a system under the command of an application program. It is composed of a CPU, a power supply and I/O modules.
Programming Language	Set of rules, conventions and syntaxes utilized when writing a program.
RAM	Random Access Memory. Memory where all the addresses may be accessed directly and in random order at the same speed. It is volatile, in other words, its content is erased when powered off, unless there is a battery to keep its contents.
Redundant CPU	The other CPU in a redundant system. For instance, the redundant CPU of CPU2 is CPU1 and vice versa.
Redundant system	System with a backup or double elements to execute specific tasks. Such system may suffer certain failures without stopping the execution of its tasks.
Ripple	Oscillation present in continuous voltages.
RX	Acronym used to indicate serial reception.
Scan Cycle	A complete execution of the PLC application program.
Sockets	Part 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
Supervisory Station	Equipment connected to a PLC network with the goal of monitoring and controlling the process variables
Tag	Name associated to an operand or to a logic that identifies its content.
Toggle	Element with two stable states that are switched at each activation.
Hot swap	Procedure of replacing modules in a system without powering it off. It is a normal procedure for I/O modules.
TX	Acronym used to indicate serial transmission.
Upload	Reading a program or configuration from the PLC.
Varistor	Protection device against voltage spikes.
Watchdog timer	Electronic circuit that checks the equipment operation integrity.

WD	Acronym for watchdog. See Watchdog timer
Word	Information unit composed by 16 bits.

Ponto Series Glossary

Fieldbus Head Address	The node address in the fieldbus. It is adjusted in the fieldbus head terminal base.
Bus	Set of I/O Modules connected to a CPU or fieldbus head
Bus Expander	Module that connects one segment to another.
Bus Segment	Part of a bus. A local or remote bus may divide in up to four bus segments.
Bus termination	Module that must be connected to the last module in a bus.
DIN Rail	Metallic element with standardized shape accordingly to the DIN50032 standard. It is also called TS35 rail.
Expansion cable	Cable that connects bus expanders
Field cabling	Cables connecting sensors, actuators and other process devices to the Ponto Series I/O modules terminal bases.
Fieldbus Cable	Cable that connects the nodes in a fieldbus, such as the Fieldbus Interface and the Fieldbus Head
Fieldbus Head	Slave module of a fieldbus (field network). It is responsible for the exchange of data between the modules and the fieldbus master.
Fieldbus Interface	Master module for the fieldbus, located in the local bus and performing the communication with the fieldbus heads.
Local Bus	Set of I/O Modules connected to a CPU.
Mechanical Switch Code	Two decimal digits defined by the base terminal programmable mechanical switches with the goal of blocking the assembly of incompatible modules. Thus avoiding potential damage caused by assembly and/or maintenance operations.
Remote Bus	Set of I/O Modules connected to the fieldbus head.
Terminal Base	Component where the IO modules, CPUs, power supplies and remaining Ponto Series modules are inserted. Connection to bus signals and field signals are made through the terminal base.

Network Glossary

Autoclear	PROFIBUS parameter that switches the master status into Clear when there is a network error.
Backoff	Time that a node in a CSMA/CD network takes before transmitting data after a collision has occurred in the physical medium.
Baud rate	Rate in which information bits are transmitted through a serial interface or communication network (measured in Bits/second, bps)
Bridge	Device to connect two communication networks with the same protocol.
Broadcast	Information sent simultaneously to all the nodes in a communication network.
Communication Network	Set of devices (nodes) interconnected by communication channels.
CSMA/CD	Way to control the access to the physical medium based on data collisions. It is used on Ethernet networks.
Deterministic communication network	Communication network where the transmission and reception of information among the nodes is guaranteed to occur within a maximum determined time period.
EIA RS-485	Industrial standard for physical layer on data communication.
EN 50170	European standard defining the PROFIBUS fieldbus.
Frame	Information unit transmitted in the network.
Freeze	PROFIBUS network status where input data is frozen.
Gateway	Device to connect two communication networks with different protocols.
Master	Device connected to a communication network originating all the command requests to other network units.
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.
Media access	Method used by all nodes in a network to synchronize data transmission and solve possible conflicts in simultaneous transmissions.
Monomaster	PROFIBUS network with only one master
Multicast	Simultaneous communication with a group of nodes connected to a network.
Multimaster	PROFIBUS network with more than one master.
Multimaster communication	Communication network where the data transfer are initiated by any node connected to the data bus.

network	
Node	Any station in a network with the capacity to communicate using a determined protocol.
Peer to peer	Type of communication where two nodes exchange data without relying on the master..
Protocol	Procedures and formats rules that allow data transmission and error recovery among devices with the use of control signals
Serial Channel	Unit interface that transfers data serially.
Slave	Device connected to a communication network that only transmits upon the master requests.
Sub network	Segment of a communication network that connects a group of devices (nodes) with the goal of isolating the local data traffic or using different protocols or physical media.
Time-out	Maximum preset time to a communication to take place. When exceeded then retry procedures are started or diagnostics are activated.
Token	It is a mark that indicates who is the bus master in a moment.