

MasterTool Programming Programming Manual

Ref. 6399-601.4

Issue. H 01/2005

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

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

- The **industrial automation equipment**, built by ALTUS are strong and reliable due to the stringent quality control it is subjected to. However the electronic industrial control equipment (programmable controllers, numerical commands, etc.) can cause damage to the machines or processes through their controllers when there are defective components and programming or installation errors. This can even put human lives at risk.
- The user should consider the possible consequences of the defects and should provide additional external installations for security so that, if necessary, the security of the system can be maintained especially during the initial installation and testing.
- It is essential to completely read the manuals and/or about the technical characteristics of the product before it's installation or use.

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

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

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

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

COPYRIGHTS

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

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



Index

Preface xvii

Description of this Manual	xvii
Related Manual	xviii
Terminology	xviii
Conventions Used	xix
Conventions for Use with the Mouse	xx
Technical Support	xxi
Issue of this Manual	xxi

Introduction 1

Programming Language	1
----------------------------	---

Diagrams of Relays Language 1

Elements of Programming	1
ALTUS PLCs Memory Organization	2
Logics 4	
Operands 5	
Identifying an Operand through Address	6
Identification of an Operand through Tag	6
Operands Used in MasterTool	7
Identification of Simple Operands	8
Identification of Constant Operands	9
Identification of Table Operands	10
Operands %E - Input Relays	10
Operands - Output Relays	11
Operands %A - Auxiliary Relays	12
Operands %R - Addresses in the Bus	13
Operands %M - Memories	15
Operands %D - Decimals	16
Operands %KM and %KD - Constants	17
Operands %TM and %TD - Tables	17
Indirect Access	18
Declaration of Operands	20



Retentive Operands.....	24
Instructions	25
Restrictions as to How Much to Use Instructions in the PLC's.....	27
Graphic Representation of Instructions.....	28
Description of Instructions Syntax.....	29
Restrictions as to Positioning of the Instructions	30
Programming Project.....	32
Structure of a Programming Project	32
Operation Status of the PLC	39
Execution of the Programming Project.....	41
Elaboration of Programming Projects.....	42
Depuration of Programming Projects.....	51
Program Execution Cycle Times	63
Protection Levels of the PLC.....	65
Interlocking of Commands in the PLC	67
Router Project.....	70
Building up a Router Project.....	70
Router Operation States	72

Instructions	1
---------------------	----------

List of Instructions	1
Conventions Used	1
Instructions of the Relays Group	2
Instructions of the Relays Group	4
Contacts 5	
Coils 6	
SLT - Jump Coil.....	7
PLS - Pulse Relay.....	9
RM, FRM - Master Relay, End of Master Relay	10
Instructions of Group Moving	10
MOV - Moving of Simple Operands.....	12
MOP - Moving of Parts (Subdivisions) of Operands	14
MOB - Moving of Blocks of Operands.....	17
MOT - Moving of Tables.....	19
MES - Moviment of Inputs/Outputs	22
CES - Conversion of Inputs/Output.....	24
AES - Update Inputs/Outputs.....	26
CAB - Load Block.....	28
Arithmetic Instructions of the Group	33
SOM - Addition.....	34
SUB - Subtraction	36
MUL - Multiplication	38
DIV - Division.....	39
AND - AND Binary between Operands	41



OR - Or Binary between Operands	43
XOR - Or exclusive between Operands	45
CAR - Load Operands	47
Instructions of Comparison of Operands - Equals, More than and Less than	48
Instructions of Group Counters	52
CON - Simple Counter	53
COB - Bidirectional Counter	55
TEE - Timer in the Powering	57
TED - Timer in the Turning Off	59
Group Converter instructions	61
B/D - Conversion Binary-Decimal	62
D/B - Conversion Decimal-Binary	62
A/D - Conversion Analog - Digital	63
D/A - Conversion Digital - Analog	66
General Group Instructions	69
LDI - Connect/Disconnect Indexed	70
TEI - Test of Indexed Status	73
SEQ - Sequencer	75
CHP - Procedure Module Call	82
CHF - Function Module Call	84
ECR - Write from Operands to another PLC	88
LTR - Reading of Operands from Another PLC	98
LAI - Free Updating of Images of Operands	100
Group Connection Instructions	101
LGH - Horizontal Connection	101
LGN - Denied Connection	101
LGV - Vertical Connection	101

Function Modules **1**

F-RELOG.000 - Function to Access the Real Time Clock	4
Introduction	4
Programming	4
F-LEDS.001 - Function to Access the LEDs Module Panel	7
Introduction	7
Programming	7
F-PT100.002 - Function to read Module Pt 100	10
Introduction	10
Programming	10
F-TERM0.003 Function to Read Termopar Module	14
Introduction	14
Programming	14
F-CONTR.004 - Function to Access the Rapid Counter Module	18
Introduction	18
Programming	18



F-CONT.005 - Function to Access the Fast Counting Inputs.....	21
Introduction.....	21
Programming.....	21
Description of Functioning.....	23
F-ANLOG.006 - Function to Convert A/D or A/D Integrated.....	24
Introduction.....	24
Programming.....	24
F-EVENT.017 - Function to Access the Module Register of Events	26
Introduction.....	26
Programming.....	27
F-ALNET2.032 - Function Read from Statistics of ALNET II.....	40
Introduction.....	40
Programming.....	40
F-PID.033 - PID Control Function	46
Introduction.....	46
Programming.....	47
F - RAIZN.034 - Square Root Function with Normalization of Scale	54
Introduction.....	54
Programming.....	54
FR-ARQ2.035 to F-ARQ31.042 - Functions Data File	57
Introduction.....	57
Programming.....	58
F-MOBT.043 - Function for Moving of blocks from Table Operands.....	63
Introduction.....	63
Programming.....	63
F-STMOD.045 - Function Status of the Buses and I/O Modules	66
Introduction.....	66
Programming.....	66
F-RELG.048 - Function to Access the Real Time Clock.....	72
Introduction.....	72
Programming.....	72
F-SINC.049 - Function to Access the Synchronized Real Time Clock	76
Introduction.....	76
Programming.....	76
F-RELOG.061 - Function to Access the Real Time Clock of QK801 and QK2000.....	82
Introduction.....	82
Programming.....	82
F-ALNET1.062 - Function Interpreter of the ALNET I Protocol for QK801	86
Introduction.....	86
Programming.....	86
F-IMP.063 Function for Printing ASCII Characters.....	91
Introduction.....	91
Programming.....	91
F-RECEP.064 - Function for Reception of ASCII Characters	95



Introduction.....	95
Programming.....	95
F-UTR_S.068 - Function to turn on UTRs outputs	99
Introduction.....	99
Programming.....	100
F-COMP.B.070 – Function to Compare Operands Blocks.....	107
Introduction.....	107
Programming.....	108
Inputs and Outputs	109
F-NORM.071 - Function for Normalization.....	111
Introduction.....	111
Programming.....	111
F-COMPF.072 - Function for Multiple Comparisons	114
Introduction.....	114
Programming.....	114
F-ALMLOG.075 – Function to Logic Alarms.....	117
Introduction.....	117
Programming.....	117
Operation	119
Inputs and Outputs	120
F-XMOV.088 – Module to Move the Data From the CPU to Memory Operands	121
Parameters:.....	121
Inputs of the function	122
Outputs of the function	122
Functioning:	122
F-ANDT.090, F-ORT.091 and F-XORT.092 - Functions of Logic Operations between Table Operands	124
Introduction.....	124
Programming.....	125
F-NEGT.093 - Function for the logic denial of Table Operands	128
Introduction.....	128
Programming.....	128

Appendix A Execution Times of the Instruction **1**

Description of Execution Times	1
Relays 2	2
Movements	4
Arithmetic.....	7
Counters 10	10
Conversor	11
General 12	12

Appendix B Execution Times of the Function Modules **1**



Description of Execution Times 1



Figures

Figure 2-1 Logic Format	4
Figure 2-2 Processing Order of the Logic Cells	5
Figure 2-3 Format of a Simple Operand	8
Figure 2-4 Format of a Constant Operand	9
Figure 2-5 Operand Table Format	10
Figure 2-6 Format of Operands %E	11
Figure 2-7 Format of operands %S	12
Figure 2-8 Formats of Operands %A	13
Figure 2-9 Formats of the Operands	13
Figure 2-10 Formats of Operands %M	15
Figure 2-11 Format of Memory Operand	15
Figure 2-12 Formats of Operands %D	16
Figure 2-13 Format of Decimal Operand	16
Figure 2-14 Format of Constant Operands	17
Figure 2-15 Format of Table Operands	18
Figure 2-16 Format of an Indirect Access	18
Figure 2-17 Format of Instructions Syntax	30
Figure 2-18 Format of Name of Modules in File	32
Figure 2-19 Parameter Passing for Module F	38
Figure 2-20 Operating Statuses of the PLC	41
Figure 2-21 Execution of Programming Project	42
Figure 2-22 Maximum Number of Levels for Call from Modules	44
Figure 2-23 Recursive Call of Modules	44
Figure 2-24 Module Call Loop	45
Figure 2-25 Care in Use of Module E018	47
Figure 2-26 Care in Use of Module E020	48
Figure 2-27 Incoherent Situation in Logic Monitoring	53
Figure 2-28 Compaction of RAM Memory	58
Figure 2-29 Compaction of RAM Memory-2	59
Figure 2-30 Compaction of RAM Memory-3	59
Figure 2-31 Format of Name of Module R in file	70
Figure 2-32 Operating Statuses of the Router	73
Figure 3-1 Example of SLT Instruction	7
Figure 3-2 Example of Instruction MOP	15



Figure 3-3 Dialogue Box CAB - Values.....	29
Figure 3-4 Dialogue Box CAB - Editing in ASCII.....	30
Figure 3-5 CAB - Initialize table	31
Figure 3-6 Example of Instructions of Comparison.....	49
Figure 3-7 Example of the Instructions of Comparison	49
Figure 3-8 Incorrect Use of the Instruction CAR.....	50
Figure 3-9 Correct Use of the Instructions CAR.....	51
Figure 3-10 Diagram of Times of the Instruction TEE.....	58
Figure 3-11 Diagram of Times of Instruction TED	60
Figure 3-12 Dialogue Box CHF - Input Parameters.....	86
Figure 3-13 Dialogue Box ECR - Parameters.....	90
Figure 3-14 Control Operand for Instruction ECR and LTR	94
Figure 4-1 Layout for the Events Registers	31
Figure 4-2 Example 1 of Use of Module F-EVENT.017.....	38
Figure 4-3 Example 2 of Use of Module F-EVENT.017.....	39
Figure 4-4 Example of Use of Module Function F-ALNET.032	44
Figure 4-5 Diagram of Times of Example of F-ALNET.032	45
Figure 4-6 Diagram in Blocks of the Function PID	47
Figure 4-7 Example of Diagram of Setting Input Times.....	75
Figure 4-8 Example of Diagram of Times of Input Set Maintained Synchronism.....	79
Figure 4-9 Example of Diagram of Times of Input Setting External Pulse	80
Figure 4-10 Example of Diagram of Input Set Times.....	85



Tables

Table 2-1 Shows the memory capacity of the applications program for each controller.....	3
Table 2-2 Operands Used in MasterTool	7
Table 2-3 Memory Capacity of the PLC's Numeric Operands	20
Table 2-4 Occupied Memory and Location of Operands	21
Table 2-5 Maximum Quantity of Operands	22
Table 2-6 Maximum Quantity of Operands	23
Table 2-7 Maximum Quantity of Operands	24
Table 2-8 Non-existent Instructions in Certain PLCs	27
Table 2-9 Braking of Commands in the PLC (loading module)	68
Table 2-10 Braking of Commands in the PLC (Compacting RAM)	69
Table 3-1 Instructions of Relays Group	4
Table 3-2 Syntax of the Instructions RNA and RNF	5
Table 3-3 Syntax of Instructions BOB, BBL and BBD	6
Table 3-4 Syntax Instruction SLT	8
Table 3-5 Syntax of PLS Instruction.....	9
Table 3-6 Instructions of Group Movements.....	11
Table 3-7 Syntax of the Instruction MOV	13
Table 3-8 Syntaxes of the Instruction MOP	16
Table 3-9 Syntax of the Instruction MOB.....	18
Table 3-10 Syntax of the Instructions MOT	21
Table 3-11 Syntaxes of the Instruction MES	23
Table 3-12 Syntaxes of the Instruction CES	25
Table 3-13 Syntaxes of the Instruction AES	27
Table 3-14 Syntax of the Instruction CAB.....	32
Table 3-15 Arithmetic Instructions of the Group	33
Table 3-16 Syntaxes of the Instruction SOM	35
Table 3-17 Syntaxes of the Instruction SUB	37
Table 3-18 Syntax of the Instruction MUL	38
Table 3-19 Syntax of the Instruction DIV.....	40
Table 3-20 Point to Point Operations	41
Table 3-21 Syntaxes of the Instruction AND	42
Table 3-22 Operations Point to Point (OR).....	43
Table 3-23 Syntaxes of the Instruction OR	44
Table 3-24 Operations Point to Point (XOR)	45



Table 3-25 Syntaxes of the Instruction XOR.....	46
Table 3-26 Syntax of the Instruction CAR.....	47
Table 3-27 Syntax of the Instructions More than, Equals and Less than.....	51
Table 3-28 Instructions of Group Counters.....	52
Table 3-29 Syntax of Instruction CON.....	54
Table 3-30 Syntax of the Instruction COB.....	56
Table 3-31 Syntax of the Instruction TEE.....	58
Table 3-32 Syntax of the Instruction TED.....	60
Table 3-33 Group Converter Instructions.....	61
Table 3-34 Syntax of the Instruction B/D.....	62
Table 3-35 Syntax of the Instruction D/B.....	63
Table 3-36 Syntax of Instruction A/D.....	65
Table 3-37 Instruction D/A - Output in Tension.....	67
Table 3-38 Instruction D/A - Output in Current.....	67
Table 3-39 Syntax of the Instruction D/A.....	68
Table 3-40 General Group Instructions.....	69
Table 3-41 Syntaxes of the Instruction LDI.....	72
Table 3-42 Syntaxes of the Instructions TEI.....	74
Table 3-43 Syntax of the Instruction SEQ.....	81
Table 3-44 Syntax of CHP Instruction.....	83
Table 3-45 Syntax of Instruction CHF.....	87
Table 3-46 Addresses of Node and Sub-network.....	89
Table 3-47 Operand for Local and Remote PLCs in ECR.....	92
Table 3-48 Occupation in Bytes of the Operands of the ECR.....	93
Table 3-49 Example of Occupation in Bytes.....	93
Table 3-50 Syntax of the Instruction ECR.....	96
Table 3-51 Syntax of the LTR Instruction.....	98
Table 3-52 Group Connection Instructions.....	101
Table 4-1 List of Function Modules Supplied through ALTUS.....	3
Table 4-2 Values read by the clock (F- RELOG.000).....	5
Table 4-3 Linearisation and Configuration of the Modules AL - 1117 and QK 1117.....	12
Table 4-4 Values Read from Modules AL-1109 and QK1109.....	16
Table 4-5 Frequency of Counting AL-600.....	23
Table 4-6 Interface Configuration Parameters.....	29
Table 4-7 Declaration of the Module AL-3130 in the Bus.....	37
Table 4-8 Description of Values of Statistics and Parameters.....	43
Table 4-9 Additional Parameters of the P/D.....	50
Table 4-10 Occupation of the Field of the Files.....	60
Table 4-11 Capacity of the Functions Data Files.....	61
Table 4-12 Format for Storing of the Status of the I/O for the AL-2002/MSP.....	67
Table 4-13 Format for storing of the I/O Status for the AL-2003.....	68
Table 4-14 Format for storing the Status of the Buses.....	70
Table 4-15 Values Read from the Clock (F-RELG.048).....	73
Table 4-16 Values of the Days of the Week (F-RELG.048).....	74



Table 4-17 Values Read from the Clock (F-SINC.049).....77

Table 4-18 Values of the Days of the Week (F-SINC.049).....78

Table 4-19 Values Read from the Clock (F-RELOG.061).....83

Table 4-20 Values of the Days of the Week (F-RELOG.061).....84

Table 4-21 Commands Executed by Module F - ALNET1.06289

Table 4-22 Commands not executed by Module F-ALNET1.062.....90

Table 4-23 Definition of ranges..... 115



Preface

Description of this Manual

This Manual gives a general description, instructions for programming, method of operating and commands of the software programmer MasterTool Programming. It was written assuming a familiarity with the use of standard IBM PC[®] microcomputers and Windows[™] operating environment.

The software programmer MasterTool Programming MT4000 or MT4100 referred to from now on as MasterTool[®] was developed for programming in the relay and blocks language of the programmable controller series ALTUS AL-600, AL-2000, AL-3000, QUARK[®] and PICCOLO as well as the configuration of the router devices AL-2400/S, AL-2401, QK2400 and QK2401.

This manual is divided into 6 chapter and two appendixes.

Chapter 1, **Introduction**, present the basic characteristics of PLCs programming and ALTUS router devices.

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

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

Chapter 4, **Function Modules**, describe objective of ALTUS sponsored function modules and its statement programming.

Appendix A, **Instruction Execution Time**, show a list of execution time of instructions .

Appendix B, **Function Modules Execution Times**, has a list of execution time of function modules .



Related Manual

For more information about the MasterTool programmer, the ALTUS series of PLCs, the programming language and the networks ALNET I and ALNET II we recommend the following manuals:

- User's Manual for AL-3830
- MasterTool User's Manual
- User's Manual AL-600
- User's Manual AL-2000/MSP
- User's Manual AL-2002/MSP
- User's Manual AL-2003
- User's Manual AL-3000
- User's Manual for PLCs in the QUARK series.
- User's Manual for PLCs in the PICCOLO series.
- User's Manual for ALNET II
- User's Manual for FOCOS
- Technical Characteristics
- NT-031: ALNET PROTOCOL

Terminology

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

The abbreviation MSP indicates “Multi Station Processor”, that is to say, refers to the PLC's capacity to carry out distributed processes in several stations.

The name MasterTool identifies the ALTUS program for the standard IBM-PC[®] microcomputer, executed in the operating environment of Windows[™] 95/98/ME (MT4000) or Windows[™] NT/2000 (only MT4100 it's compatible). It allow applications development in the PLCs series AL-600, AL-2000, AL-3000, QUARK, PICCOLO, Ponto Series besides AL-2400/S, AL-2401, QK2400 and QK2401 router devices.



Throughout manual this program will be referred to through the appropriate initials or as “MasterTool programmer”.

The word “module”, when it refers to hardware, is used for denoting each of the components of the equipment.

The word “module”, when it refers to software, is used to denote each of the components of an applications program.

Conventions Used

The symbols used throughout this manual have the following significance:

- This mark indicates a list of items or topics

CAPITAL LETTER indicate names of keys, for example ENTER.

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

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

CAPITAL LETTERS indicate file names and folder names.

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

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

Warning messages have the following format and significance.

⚠DANGER:

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



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

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

Contains important information about the product, its operation or a part of the text which should be given special WARNING.

😊HINT:
The label HINT indicates a better way of carrying out a task.

Conventions for Use with the Mouse

Despite MasterTool only being able to be executed with the use the keyboard, it's execution can be achieved more efficiently with a mouse.

Some terms are used to describe the action to be executed with the mouse in order to achieve a specific task.

Term	Significance
Click	Pressing the main button on the mouse. Normally the main button is on the left, but it can be changed for use by left-handed people through the Control Panel of Windows, by the Configurations command, Mouse .
Click twice or double click	Press the button twice with a short time interval. This time interval can be configured in the Control Panel of Windows, by the Configurations command, Mouse .
Drag	Press the main button on the mouse, move the mouse to the required position, keeping the button pressed and then releasing.



Technical Support

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

Internet address:

E-Mail: altus@altus.com.br

Home page: <http://www.altus.com.br>

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

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



information about MasterTool

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



information about the PLC or router

contains the applications program, available through the command **Communication, Modules** of the MasterTool programmer.

Issue of this Manual

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

The information which cannot be included in this manual appears in file README.WRI, which accompanies the MasterTool product. In order to consult it the Windows™ software WRITE.EXE should be used:

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

Issue A	Date 01/99
	First issue of the manual.
Issue B	Date 08/2000



PL104 and PL105 include.
Error correction.

Issue C

Date 06/02

Contents equaled with portuguese (brazilian) manual
version.

Issue H

Date 01/05

Sincronization of the manuals (Portuguese and English).



Introduction

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

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

Programming Language

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

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

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

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





Diagrams of Relays Language

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

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

Elements of Programming

An applications program is made up of 4 basic elements:

- Modules
- logics
- instructions
- operands

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

An applications program module is divided into **programming logics**. The format of an applications program logic used in PLCs in the series AL-600, AL-2000, AL-3000, QUARK and PICCOLO allows up to eight elements in the series and up to four parallel routers.

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



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

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

ALTUS PLCs Memory Organization

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

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

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

Table 2-1 shows the memory capacity of the applications program for each controller. In this table the EPROM flash memory (flash) was distinguished from the EPROM cartridge (EPROM) to indicate with greater precision the type of memory used in each PLC model, although they may be equal.



Controller		Standard Capacity of Banks				Maximum Capacity of Banks			
		0	1	2	3	0	1	2	3
AL-600	flash	-	-	-	-	32K	32K	32K	32K
	RAM	16K	-	-	-	16K	32K	32K	32K
AL-3003	EPROM	32K	-	-	-	32K	32K	-	-
	RAM	32K	-	-	-	32K	32K	-	-
AL-3004	EPROM	16K	-	-	-	32K	-	-	-
	RAM	16K	-	-	-	16K	-	-	-
AL-2000/MSP	flash	32K	32K	-	-	32K	32K	32K	32K
	RAM	32K	-	-	-	32K	32K	32K	32K
AL-2002/MSP	flash	32K	32K	-	-	32K	32K	32K	32K
	RAM	32K	-	-	-	32K	32K	32K	32K
AL-2003	flash	16 banks of 64 Kbytes				16 banks of 64 Kbytes			
	RAM	64K	64K	-	-	64K	64K	-	-
PL101, PL102, PL103	flash	-	-	-	-	-	-	-	-
	RAM	16K	-	-	-	-	-	-	-
PL104, PL105	flash	32K	-	-	-	32K	-	-	-
	RAM	32K	32K	-	-	32K	32K	-	-
QK600	flash	-	-	-	-	32K	32K	32K	32K
	RAM	16K	-	-	-	16K	32K	32K	32K
QK800	flash	32K	-	-	-	-	-	-	-
	RAM	32K	-	-	-	-	-	-	-
QK801	flash	32K	32K	-	-	32K	32K	32K	32K
	RAM	32K	-	-	-	32K	32K	32K	32K
QK2000	flash	32K	32K	-	-	32K	32K	32K	32K
	RAM	32K	-	-	-	32K	32K	32K	32K

Table 2-1 shows the memory capacity of the applications program for each controller

The values of the numeric operands (% M, %D, %TM and %TD) are stored in a separate area of the program, with different sizes according to the model of PLC. The amount of operands memory free can be checked in the editing window of module C in the operands panel. For further information about the



Editing Window of module C, c.f. section Editing Windows, in chapter 3 of the User’s Manual.

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

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

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

Logics

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

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

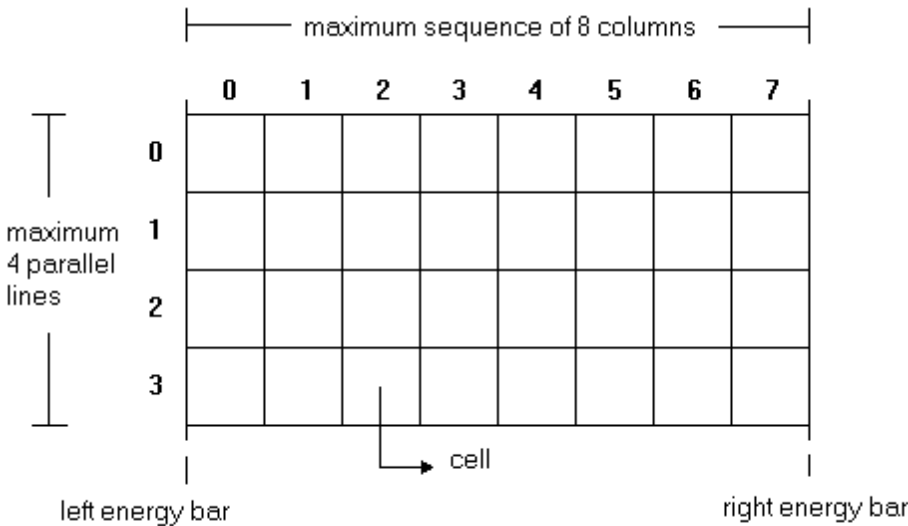


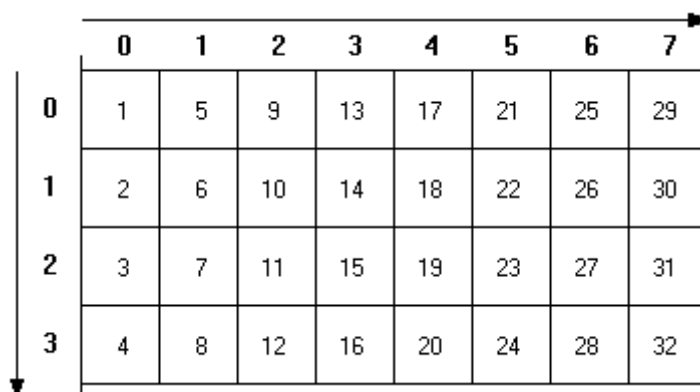
Figure 2-1 Logic Format



The two lateral lines of the logic represent energy bars between the instructions placed for execution. Symbolic instructions usually found in diagrams are available for programming, such as contracts, coils, connections and instructions shown in boxes as timers, counters and arithmetics.

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

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



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

Figure 2-2 Processing Order of the Logic Cells

Operands

Operands are elements used for MasterTool instructions in the elaboration of an applications program.

The operands can define constant values, defined at the time of programming, or variables, identified through an address or tag, with values able to be changed during the execution of an applications program.



Identifying an Operand through Address

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

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

Identification of an Operand through Tag

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

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

E.g.:

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

☺HINT:

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

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

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



For further information about creating and attributing tags to operands, c.f. sections about the command **Report, Operands**, in chapter 4 and **Inserting Tags and Comments for Operands** in chapter 5 of MasterTool User's Manual.

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

Operands Used in MasterTool

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

Type	Operand
%E	Input Relays
%S	Output Relays
%R	Bus Address
%A	Auxiliary Relays
%M	Memories
%D	Decimals
%KM	Memory Constants
%KD	Decimal Constants
%TM	Memory Tables
%TD	Decimal Tables

Table 2-2 Operands Used in MasterTool

The operands are divided into 3 groups:

- simple operands
- constant operands
- table operands



Identification of Simple Operands

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

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

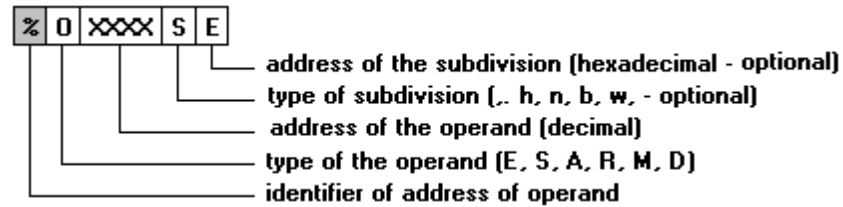


Figure 2-3 Format of a Simple Operand

Operand type:

- **%I** - input
- **%O** - output
- **%A** - auxiliary
- **%R** - bus address
- **%M** - memory
- **%D** - decimal

Subdivision type

- **.** - point of box word
- **h** - point of high word
- **n** - nibble (4 points)
- **b** - octet (8 points)
- **w** - word (16 points)



Examples of addresses:

- % E0002.3 - point 3 of input operand 2
- %S0004.7 - point 7 of output operand 4
- %A0039n I - nibble I of auxiliary operand 39
- %A0045 - auxiliary octet 45
- %M0205 - memory operand 205
- %M020560 - octet 0 of memory 205
- %D0029 - decimal operand 29
- %D0034wl - word I of decimal operand 34

Examples of tags:

- FORNO
- LIMSUP
- CHAVE1

Identification of Constant Operands

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

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

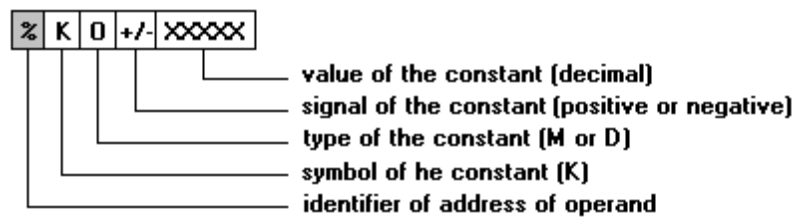


Figure 2-4 Format of a Constant Operand

Constant type:

- **%M** memory
- **%D** decimal

Examples:



- %KM+05172 - positive constant memory
- %KD-0974231 - constant negative decimal

Identification of Table Operands

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

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

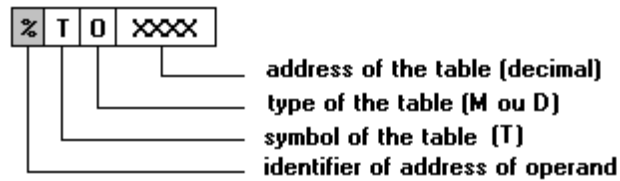


Figure 2-5 Operand Table Format

Table type:

- % **TM** memory
- % **TD** decimal

Examples:

- % TM0026 - memory table 26
- % D0015 - decimal table 15

Operands %E - Input Relays

Operands are used to reference points of digital modules of input. Their quantity is determined through the number of I/O modules which are arranged behind the scenes of the system. C.f. item **Configuring the Bus** in the section **Configuring the Module C** in chapter 5 of the MasterTool User's Manual.



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

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

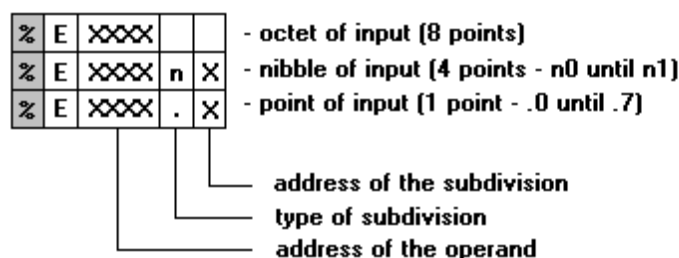


Figure 2-6 Format of Operands %E

Examples:

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

Operands - Output Relays

Operands are used to reference points of digital modules of output. Their quantity is determined through the number of I/O modules which are arranged behind the scenes in the system. C.f. item **Configuring the Bus** in the section **Configuring the Module C** in chapter 5 of the MasterTool User's Manual.

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



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

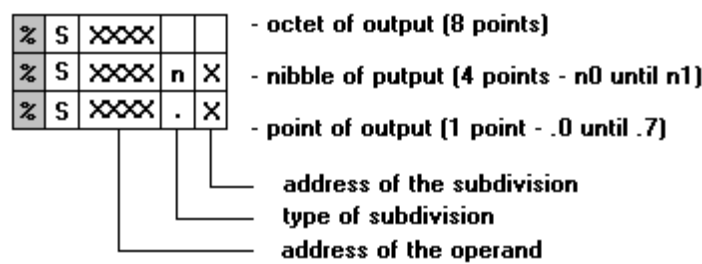


Figure 2-7 Format of operands %S

Examples:

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

Operands %A - Auxiliary Relays

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

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

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

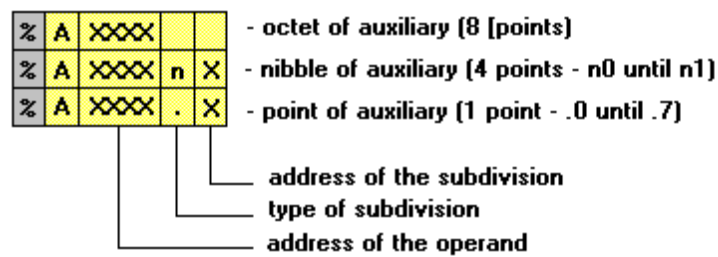


Figure 2-8 Formats of Operands %A

Examples:

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

Operands %R - Addresses in the Bus

Operands are used to reference points or octets in the bus of the input and output modules of the controller. These operands represent only addresses of the bus, not storing values not even occupying memory space. They are used in same instructions or functions which access the modules.

The formats of the operands %R can be seen in figure 2-9.

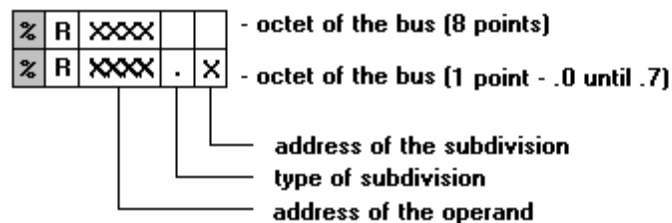


Figure 2-9 Formats of the Operands



In the PLCs **AL-3003**, **AL-3004** and the **buses 0 and 1 of AL-2002/MSP and of AL-2003**, each position in the bus corresponds to 8 octets of operands %R. In this way, in position 0 has the operands %R0000 to %R0007, in position 1, %R0008 to %R0015, and so on.

To obtain the first operand from the determined position of the bus, simply calculate:

Octet Address = Position in the Bus **X** 8

In the PLCs **PL101**, **PL102** and **PL103**, each position in the bus corresponding to the 4 octets of the operands %R. In this way, position 0 has the operands %R0000 to %R0003; in position 1, %R0004 and %R0007, and so on.

In order to obtain the first operand from the position determined in the bus, stop to calculate:

Octet Address = Position in Bus **X** 4

In the PLCs **AL-600**, **AL-2000/MSP**, **QK800**, **QK801**, **QK2000/MSP** and in the **buses from 2 to 9 of the AL-2002/MSP and of the AL-2003**, each position in the bus corresponds to 2 octets of %R operands. In this way, position 0 has the operands %R0000 and %R0001; in position 1, %R0002 and %R0003 and so on.

In order to obtain the first operand of the position determined in the bus stop to calculate:

Octet Address = Position in Bus **X** 2

The addresses for each position of the bus are automatically shown in the window of the declaration of the bus in column **Addresses** (c.f. item **Configuring the Bus** in the section **Configuring Module C** in chapter 5 of the MasterTool User's Manual.

Examples:

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



Operands %M - Memories

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

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

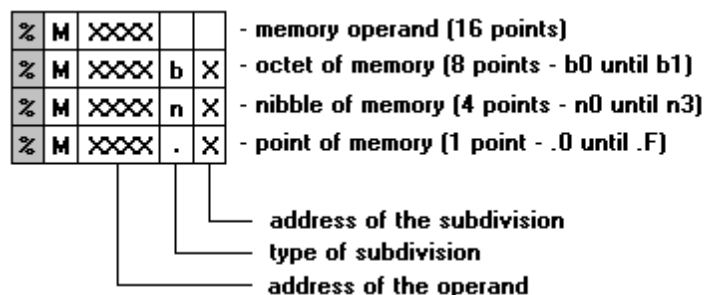
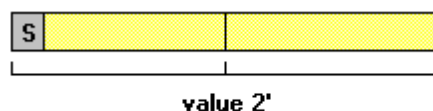


Figure 2-10 Formats of Operands %M

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

The operands %M are used in instructions of movement, comparison, arithmetic, counting, timing and conversion. They can be used in contacts, for the same form as the operands %E, %S and %A. These operands use up two bytes of memory (16 bits) storing the value in in two complement from (2) according to figure 2-11.



S - bit of arithmetic signal (0 positive, 1 negative)

Figure 2-11 Format of Memory Operand

Examples:

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



Operands %D - Decimals

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

The formats of the operands %D can be seen in figure 2-12.

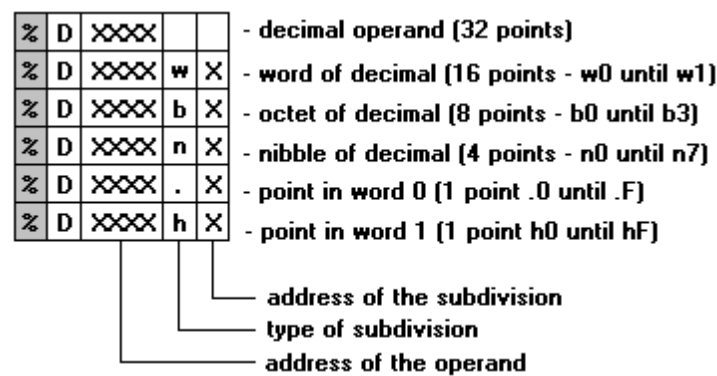


Figure 2-12 Formats of Operands %D

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

The operands %D are used in instructions of movement, comparison, arithmetic and conversion. They can be used in contacts, in the same form as the operands %E, %S and %A. These operands use up four bytes of memory (32 bits), storing the value in the format BCD (each digit occupies 4 bits), with signal, according to figure 2-13.

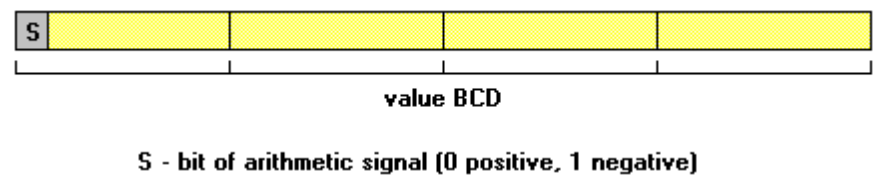


Figure 2-13 Format of Decimal Operand

Examples:

- %D0041 - decimal 41
- %D0023b2 - octet 2 of the decimal 23
- %D0059n - nibble 6 of the memory 59
- %D0172hA - point 10 of the word 1 of the memory 172

Operands %KM and %KD - Constants

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

The format of the constant operands can be seen in figure 2-14.

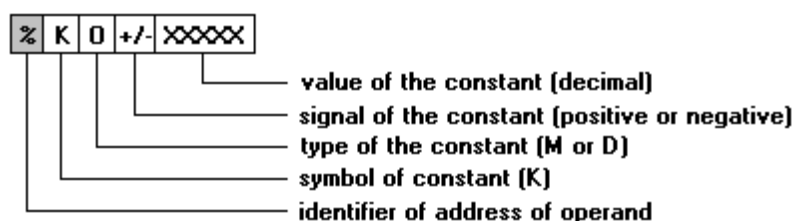


Figure 2-14 Format of Constant Operands

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

Examples:

- %KM+00241 - memory constant + 241
- %KD-0019372- decimal constant - 19.372

Operands %TM and %TD - Tables

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



The format of the table operands can be seen in figure 2-15.

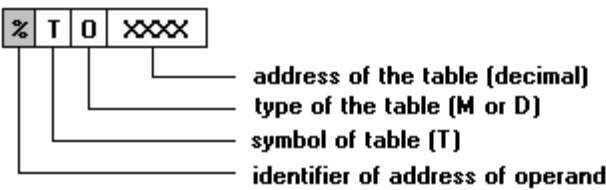


Figure 2-15 Format of Table Operands

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

The tables are used in instructions of movement.

Indirect Access

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

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

The format of indirect access can be seen in figure 2-16.

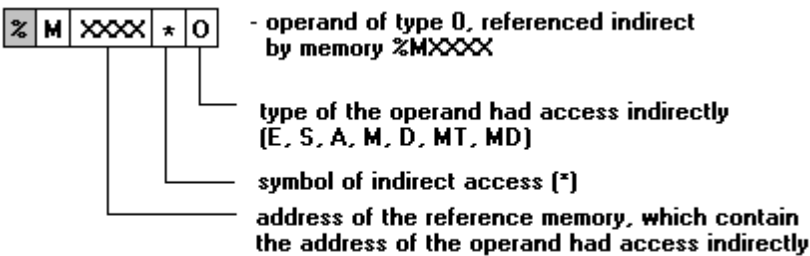


Figure 2-16 Format of an Indirect Access

In MasterTool, the indirect access to the tables is shown with the asterisk.

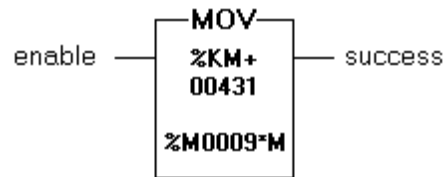


The indirect access is used in instructions of movement, comparison, counting and timing.

Example:

- %M0043*E - input octet referenced indirectly through memory 43
- %M1824*A - auxiliary octet referenced indirectly through memory 1824
- %M0371TD - table of decimals referenced indirectly through memory 371
- %M0009*M - memory operand referenced indirectly through memory 9

Example:



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

WARNING:

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

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



Declaration of Operands

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

The operands %R do not use up memory space, only being addresses to access the buses.

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

It can declare the number of operands %M, %D, %TM and %TD, these occupying their own area of RAM memory in the CPU being used. The table 2-3 shows the maximum capacity of memory for the storage of these operands in each controller. The operands %E, %S and %A do not occupy this area.

Controller	Operands Memory %M, %D, %TM and %TD
AL-600	8 Kbytes
AL-3003	11,5 Kbytes
AL-3004	11,5 Kbytes
AL-2000/MSP	15,5 Kbytes
AL-2002/MSP	15,5 Kbytes
AL-2003	48 Kbytes
QK600	8 Kbytes
QK800	15,5 Kbytes
QK801	15,5 Kbytes
QK2000	15,5 Kbytes
PL101	11,5 Kbytes
PL102	11,5 Kbytes
PL103	11,5 Kbytes
PL104 , PL105	15,5 Kbytes

Table 2-3 Memory Capacity of the PLC's Numeric Operands



The declaration of the operands is carried out through the editing window of module C of MasterTool, being stored in module C. The number of operands declared should be tailored to the maximum capacity of the available memory. C.f. items **Configuring Simple Operands**, **Configuring Table Operands** and **Configuring Retentive Operands** in the section **Configuring Module C** in chapter 5 of the MasterTool User's Manual.

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

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

Table 2-4 shows the memory space used up for each type of operands and where its values are stored.

Operand	Memory Occupied	Location
%E - input	1 byte	Microprocessor
%S - output	1 byte	Microprocessor
%A - auxiliary	1 byte	Microprocessor
%R - bus	-	-
%KM - constant M	-	-
%KD - constant D	-	-
%M - memory	2 bytes	RAM of operands
%D - decimal	4 bytes	RAM of operands
%TM - table M	2 bytes per position	RAM of operands
%TD - table D	4 bytes per position	RAM of operands

Table 2-4 Occupied Memory and Location of Operands



The tables 2-5, 2-6 and 2-7 show the number of octets for the ALTUS PLCs include inputs and output simultaneously, according to the configuration of the input and output modules used in the bus. Therefore, the sum of the number of the operands %E with %S should be smaller or equal to this limit.

Operand	AL-600 and QK600	AL-3003	AL-3004	AL-2000
%I input	%E0000 to %E0031	%E0000 to %E0063	%E0000 to %E0005	%E0000 to %E0063
%S output	%S0000 to %S 0031	%S0000 to %S0063	%S0006 to %S0009	%S0000 to %S0063
%A auxiliary	%A0000 to %A0095	%A0000 to %A0095	%A0000 to %A0095	%A0000 to %A0095
%R bus	%R0000 to %R0031	%R0000 to %R0127	%R0000 to %R0127	%R0000 to %R0063
%M memory	%M0000 to %M4095	%M0000 to %M5887	%M0000 to %M5887	%M0000 to %M7935
%D decimal	%D0000 to %D2047	%D0000 to %D2943	%D0000 to %D2943	%D0000 to %D3967
%TM memory table	4096 total positions	5888 total positions	5888 total positions	7936 total positions
%TD decimal table	2048 total positions	2944 total positions	2944 total positions	3968 total positions

Table 2-5 Maximum Quantity of Operands



Operand	AL-2002	AL-2003	PL101	PL102
%I input	%E0000 to %E0063	%E0000 to %E0255	%E0000 to %E0063	%E0000 to %E0063
%S output	%S0000 to %S0063	%S0000 to %S0255	%S0000 to %S0063	%S0000 to %S0063
%A auxiliary	%A0000 to %A0095	%A0000 to %A0511	%A0000 to %A0095	%A0000 to %A0095
%R bus	%R0000 to %R0511	%R0000 to %R0511	%R0000 to %R0063	%R0000 to %R0063
%M memory	%M0000 to %M7935	%M0000 to %M9984	%M0000 to %M4095	%M0000 to %M4095
%D decimal	%D0000 to %D3967	%D0000 to %D9984	%D0000 to %D2047	%D0000 to %D2047
%TM memory table	7936 total position	9999 total position	4096 total position	4096 total position
%TD decimal table	3968 total position	9999 total position	2048 total position	2048 total position

Table 2-6 Maximum Quantity of Operands



Operand	PL103	PL104 PL105	QK800	QK801 QK2000
%I input	%E0000 to %E0063	%E0000 to %E0063	%E0000 to %E0031	%E0000 to %E0063
%S output	%S0000 to %S0063	%S0000 to %S0063	%S0000 to %S0031	%S0000 to %S0063
%A auxiliary	%A0000 to %A0095	%A0000 to %A0095	%A0000 to %A0095	%A0000 to %A0095
%R bus	%R0000 to %R0063	%R0000 to %R0063	%R0000 to %R0031	%R0000 to %R0063
%M memory	%M0000 to %M4095	%M0000 to %M7935	%M0000 to %M7935	%M0000 to %M7935
%D decimal	%D0000 to %D2047	%D0000 to %D3967	%D0000 to %D3967	%D0000 to %D3967
%TM memory table	4096 total position	7936 total position	7936 total position	7936 total position
%TD decimal table	2048 total position	3968 total position	3968 total position1	3968 total position

Table 2-7 Maximum Quantity of Operands

The table still specifies the maximum possible number of operands %M, %D, %TM and %TD with the operands memory used totally for each type, with the declaration of the rest. If two or more different types of operands in an applications program are to be declared, the maximum possible number for each type will be different from the values presented.

Retentive Operands

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

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



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

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

C.f. item **Configuring Retentive Operands** in the section **Configuring Module C** in chapter 5 of the MasterTool User's Manual.

Instructions

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

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

MasterTool instructions can be divided into 7 groups:

- **RELAYS** containing the instructions:
 - **RNA** contact open normally
 - **RNF** contact closed normally
 - **BOB** simple reels
 - **BBL** reel connected
 - **BBD** reel disconnected
 - **SLT** reel jump
 - **PLS** pulse relay
 - **RM** master relay
 - **FRM** end of master relay
- **MOVEMENTS** containing the instructions:
 - **MOV** moving of simple operands
 - **MOP** moving of parts of operands
 - **MOB** moving of blocks of operands
 - **MOT** moving of tables of operands
 - **MES** moving of inputs and outputs
 - **CES** conversion of inputs and outputs
 - **AES** updating of inputs or outputs





- **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
 - **=** equals
 - **<** less than
 - **>** more than
- **COUNTERS** containing the instructions:
 - **CON** simple counter
 - **COB** bidirectional counter
 - **TEE** timer for turning on
 - **TED** timer for turning off
- **CONVERTORS** containing the instructions:
 - **B/D** conversion binary - decimal
 - **D/B** conversion decimal - binary
 - **A/D** conversion analog - digital
 - **D/A** conversion digital - analogue
- **GENERAL** containing the instructions:
 - **LDI** connect or disconnect the indexed points
 - **TEI** test the status of indexed points
 - **SEQ** sequencer
 - **CHP** call procedure module
 - **CHF** call function module
 - **ECR** write from operands in other PLC
 - **LTR** read from operands in other PLC
 - **LAI** read updating of image of operands
- **CONNECTIONS** containing the instructions:
 - **LGH** horizontal connection
 - **LGV** vertical connection
 - **LGN** denied connection



Restrictions as to How Much to Use Instructions in the PLC's

The language of relays and blocks is perfectly compatible between the PLCs programmed through MasterTool. Due to the characteristics of functioning, nevertheless, some instructions are not available in all the controllers. Table 2-8 shows the instructions and the controllers in which they cannot be used.

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

UCPs Instruction	AL-600	AL-3003, AL-3004	QK600, QK800, QK801	PL101, PL102, PL103, PL104, PL105	AL-2000, AL-2002, AL-2003, QK2000
CES					
A/D					
D/A					
ECR					
LTR					
LAI					

Table 2-8 Non-existent Instructions in Certain PLCs

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

WARNING:

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

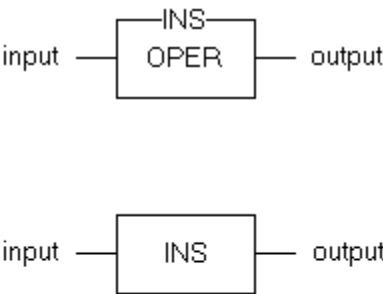


WARNING:
If is required to change the type of CPU for another, after the program to be edited, you should search and remove the instructions which cannot be used in the new type of CPU. This procedure should be carried out in all the program modules.

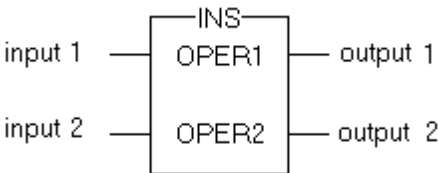
Graphic Representation of Instructions

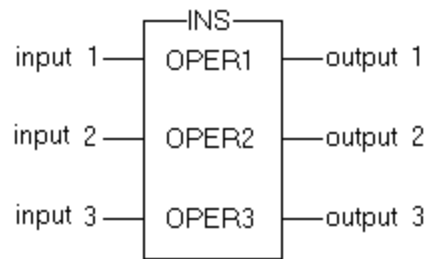
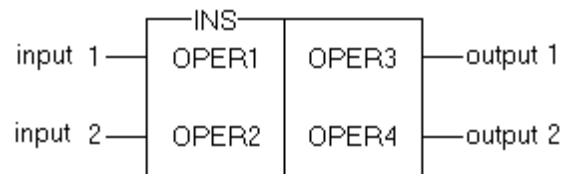
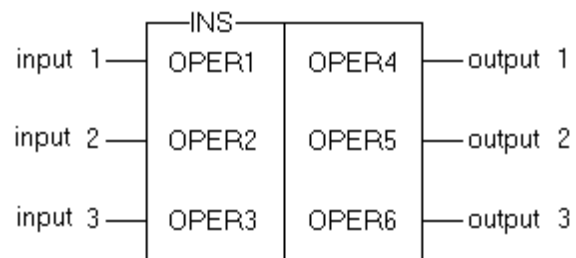
The following figures show the maximum configurations of input and outputs in each type, not being necessary all used in a certain instruction.

Instructions with a cell



Instructions with two cells



Instructions with three cells**Instructions with four cells****Instruction with six cells****Description of Instructions Syntax**

The description of the possible operands to be programmed in the cells of each instruction is carried out in accordance with the format shown in figure 2-17.



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

Figure 2-17 Format of Instructions Syntax

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

Example:

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

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

Restrictions as to Positioning of the Instructions

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

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

Instructions which can be edited in columns 0 to 6:

- **RNA** relay open normally
- **RNF** relay closed normally
- **PLS** relay pulse
- **LGH** horizontal connection
- **LGV** vertical connection
- **LGN** denied connection
- **DIV** division
- **MOB** moving of blocks of operands
- **>** more than
- **<** less than
- **=** equals
- **SEQ** sequencer
- **CHF** call function module
- **ECR** write from operands into other PLC
- **LTR** read from operands into other PLC

- Instructions which can be edited in all the columns:

- **MOV** moving of simple operands
- **MOP** moving of parts of operands
- **MOT** moving of table of operands
- **MES** moving of inputs or outputs
- **CES** conversion of inputs of 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 for turning on
- **TED** timer for turning off
- **B/D** binary conversion - decimal
- **D/B** decimal conversion - binary
- **CAR** load operand
- **LDI** connect or disconnect indexed points
- **TEI** status test for indexed points
- **CHP** call procedure module
- **LAI** free updating of image
- **A/D** analog conversion - digital
- **D/A** digital conversion - analogue



Programming Project

Structure of a Programming Project

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

The modules are called for execution through executive software (operating system of the PLC). or for other modules, through appropriate instructions. When stored on disk, the programming project corresponds to a group of files, where each file contains a module, named as shown in figure 2-18.

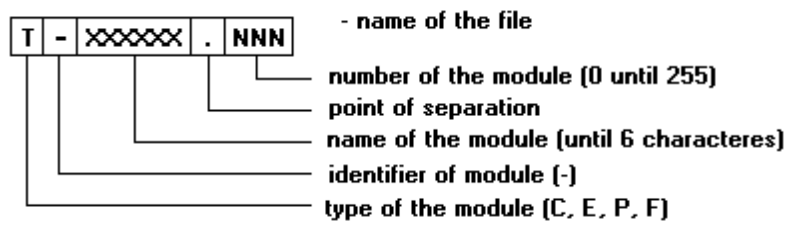


Figure 2-18 Format of Name of Modules in File

Example: F-PID. 033

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

Example: E018

WARNING:

The file name corresponds to a program module which should not be changed through another application of Windows TM. To change the name of a file, it should be read and saved with the name required through MasterTool. C.f. section **Saving a Module with Another Name** in chapter 5 of the MasterTool User’s Manual.

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

There are 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).
- **Module E** (Execution): there can be up to 4 execution modules for the project. They are only called through the operating system of the PLC (E000, E001, E018 and E020).
- **Module P** (Procedure): there can be up to 112 procedure modules per project. They contain passages of the applications program being called through instructions placed in execution modules, procedure or function. After they are executed, the returns to the following instruction of the call. The modules P act as sub-routines not allowing parameter passing for the module called (P000 to P111).
- **Module F** (Function): there can be up to 112 function modules per project. They contain passages of the applications program written in generic form, allowing parameter passing to the module called, in this way they can be reappraised in various different applications programs. They are similar to instructions, being able to be called for, modules of execution, procedure or function. (F000 to F111).

Module C - Configuration

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

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

Contents of a module C:

- **Declaration of the Bus of I/O modules:** specifies the configuration of the I/O modules to be used in the programmable controller, indicating the distribution of these modules and special modules in the PLC's bus. The declaration of the modules defines, in this way, the number of points and the I/O addresses to be used in applications program. The declaration takes place in the editing window of module C. For further information about how to configure the bus, c.f. the item **Configuring the Bus** in the



section **Configuring Module C** in chapter 5 of the MasterTool User's Manual.

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



- **Declaration of the Parameters of the ALNET I Network:** specifies the parameters necessary for the functioning of communication in ALNET I. These parameters are configured in the editing window of Module C. For further information regarding how to configure parameters of ALNET I, c.f. item **Configuring Parameters of the ALNET I Network** in the section **Configuring Module C** in chapter 5 of the MasterTool User's Manual.
- **Declaration of the Parameters of the ALNET II Network:** specifies the parameters necessary for the functioning of communication in ALNET II, for the programmable controllers which allow its use. These parameters are configured in the editing window of Module C. For further information about how to configure parameters of ALNET II c.f. item **Configuring Parameters of the ALNET II network** in the section **Configuring Module C** in chapter 5 of the MasterTool User's Manual.
- **Declaration of the Parameters of the Ethernet Network:** specifies the various parameters necessary for the functioning of communication in Ethernet, for the programmable controllers which allow its use. These parameters are configured in the editing window of Module C. For further information regarding how to configure parameters of Ethernet, c.f. item **Configuring Parameters of the Ethernet Network** in the section **Configuring Module C** in chapter 5 of the MasterTool User's Manual.
- **Declaration of the Parameters of the Synchronism Network:** specifies the various parameters necessary for the functioning of communication with the synchronism network, for the programmable controllers which allow its use. These parameters are configured in the editing window of Module C. For more information regarding how to configure parameters for the synchronism network, c.f. item **Configuring parameters of the Synchronism Network** in the section **Configuring Module C** in chapter 5 of the MasterTool User's Manual.

Module E - Execution

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

Types of Modules:



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

WARNING:

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

- **E020 - Module Activated through Input of Interruption:** the passage of the applications program placed in this Module is executed with the activating of the input of the interruption of the PLC's AL-600/4, AL-600/8, AL-600/16, QK600, PL102 or PL103. When rise into the present signal in this input occurs, the sequential execution of the applications program is interrupted and Module E020 is executed. When it is finished, the system returns to execution for the sequential processing point where the Module E0001 was interrupted. If the input is activated very often, the module's execution time has to be as short as possible, so that there is not an excessive increase in the cycle time of the Module E001.

WARNING:

The execution time of Module E020 cannot be greater than or equal to the calling period. If this happens, the PLC goes into error mode displaying the message **Recessed in Module E020**, in the window **Information** (command **Communication, Status, Information**).



The Module E020 only works in the PLCs AL-600/4, AL-600/8, AL-600/16, starting from the version 1.20 of the executive software, as well as in the PLCs QK600, PL102 and PL103. Only these PLCs have a rapid input of interruption that operates the E020.

☺HINT:

If the PLC module defined in the Module C allows the use of a certain type of module, but MasterTool does not enable its creation, it can use the generic execution module, defining its number according to its needs (E-.018 or E-.020).

Module P - Procedure

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

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

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

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

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

Examples:

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

Module F - Function

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



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

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

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

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

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

The passing of parameters is achieved through the copy of the values of the declared operands (passing of parameters for value). Figure 2-19 shows the flow of data between instruction CHF and the function module.

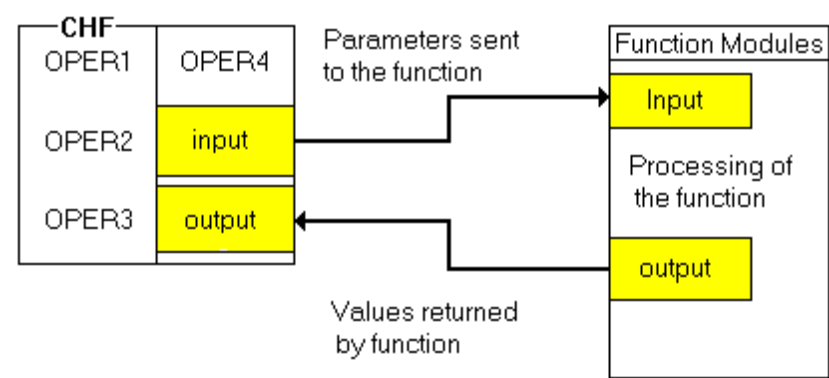


Figure 2-19 Parameter Passing for Module F

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

Examples:

- **F-LINEAR.002** - executes the linearisation of values read from a sensor

- **F-PID.033** - carries out calculations for implementing the control PID loop

Operation Status of the PLC

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

- **Status Initialization** : the PLC initializes the different data structures for use by the executive program and achieves consistency in the programming project present in the memory. This status occurs after the controller is turned on, passing after a few seconds to the execution status. If no applications program exists in memory, the PLC passes to error mode.

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

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

- **Executive Status:** normally the programmable controller is found in this status, continually sweeping away the input points and updating the output points according to the logic programmed. This status shows that the PLC is executing an applications program correctly.
- **Programming Status:** The applications program is not executed, not having the reading of the input points, the outputs being deactivated and the PLC's memory compacted. The PLC remains non-operational, waiting for commands from MasterTool. This mode is normally used to load programming project modules for MasterTool through the serial channel. At the passing for execution or cycling status starting from the programming status, the operands are zeroed.



- **Cycling Status:** when in cycling mode, the programmable controller does not execute the module E001 cyclically, remaining to wait for the commands from MasterTool. Each command **execute cycle** activated in MasterTool (options **Communication, Status, Execute Cycle** starting from the main menu or equivalent shortcut) fires one single scan of the applications program (Module E001), the PLC remaining to wait for a new command after executing the scan. When the PLC passes to cycled mode, the counting of time in the timers stops, being the same increments of one unit of time for each two scans executed. The calls to the module of interruption of time E018 are not carried out in this mode. The Module E020, activated through the input of external interruption, continues being called in this mode.
- **Error Status:** shows there was some anomaly in the PLC during the processing of the programming project. The type of error occurring can be checked through the dialogue box (options **Communication, Status, Information** starting from the main menu), while the PLC is in this status. The output of the error status is only possible passing the programmable controller to programming mode.

In normal conditions, the programmable controller can be in the modes of execution, programming and cycling, these modes being Actioned through the MasterTool commands (options **Execution, Programming and Cycling** in the dialogue box **Status**, or their shortcut equivalents in the **Tool Bars**). In the event of some functional error in these modes, the PLC passes to error status. The recovery of error mode is only possible by passing the programmable controller to programming mode. Figure 2-20 shows the possibilities for changing statuses.



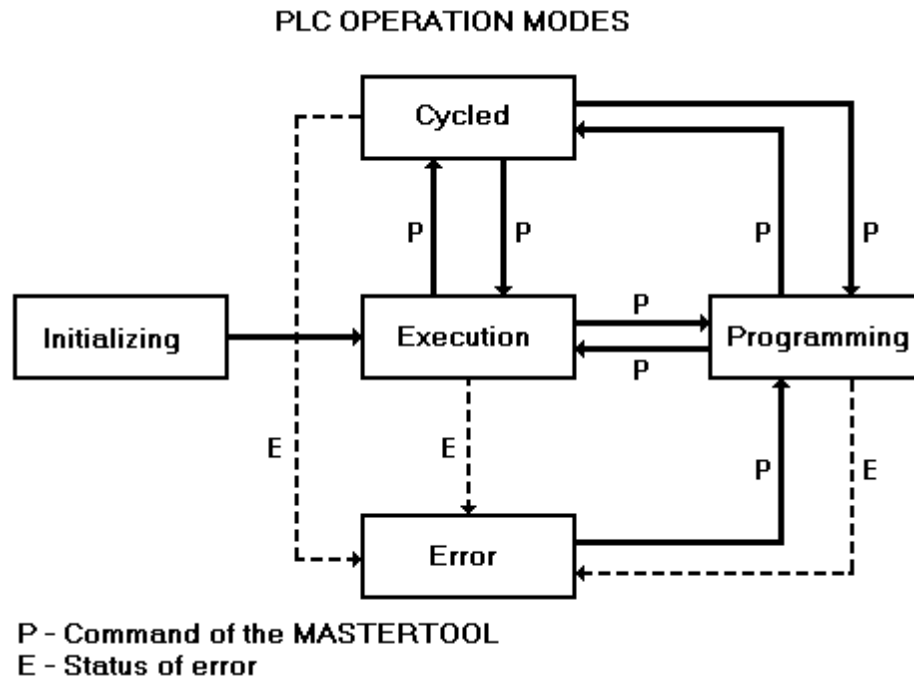


Figure 2-20 Operating Statuses of the PLC

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

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

Execution of the Programming Project

When the PLC is powered or after the passing to execution mode, the Initialization s of the system are carried out according to the contents of Module C, being straight after executing Module E000 once.

The programmable controller then passes to cyclical processing of Module E001, updating the inputs and outputs and calling the Module E018, when it exists, for each period of interruption time programmed. In PLCs AL-600/4, AL-600/8, AL-600/16, QK600, PL102 and PL103, the Module E020 is called, when existing, with the activating of the interruption input. Figure 2-21 shows the execution of the applications program in outline.



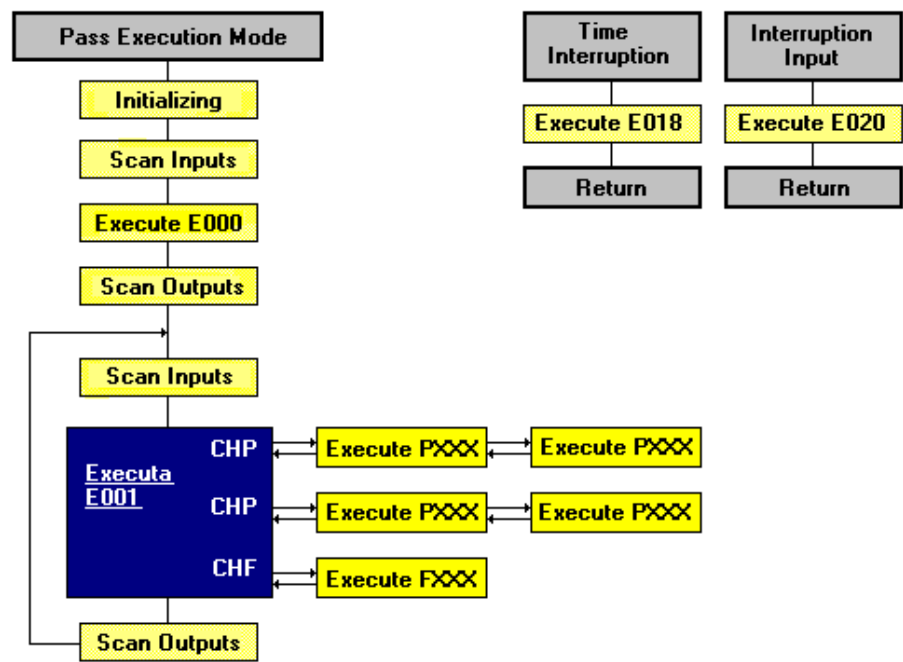


Figure 2-21 Execution of Programming Project

Elaboration of Programming Projects

General Considerations

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

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



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

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

Considering Operands

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

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

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

Use of Modules P and F

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

In the CPUs AL-600, AL-600/4, AL-600/8, AL-600/16, AL-3003, AL-3004, AL-2000/MSP, AL-2002/MSP, AL-2003, QK600, QK800, QK801, QK2000/MSP, PL101, PL102 and PL103 there are 18 call levels, since the Modules E (higher level). In PLC AL-2003 there are 32 calls levels.



That is to say up to 18 consecutive calls from modules can be executed without the execution of any being finalised. It should be considered that the module E018 (if existing) and the modules called for it also occupy call levels.

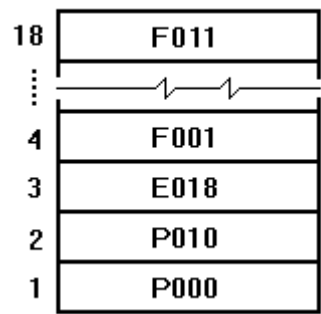


Figure 2-22 Maximum Number of Levels for Call from Modules

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

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

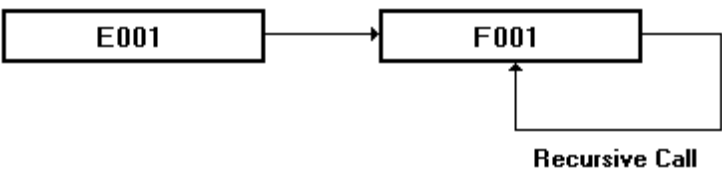


Figure 2-23 Recursive Call of Modules



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

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

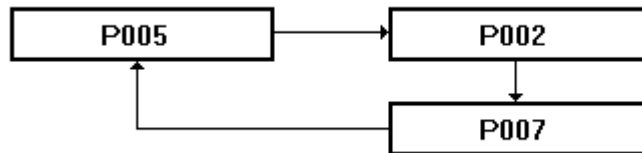


Figure 2-24 Module Call Loop

Use of Module E018

Module E018 should be used when quick processing is necessary for some points of input and output of the programmable controller, like for example, in sensing the end limit in Systems of rapid positioning. The instruction for updating the points of I/O (AES) should be used in this case, carrying out a similar process in module E018 to a complete loop of main program execution. The inputs are read, the passage of the applications program required is executed and the outputs are updated.

In this way, this module makes itself useful when it requires a response from the operations of output after a fixed time of stimulating inputs, independent from the verification time of the main program, which can vary. This characteristic is also important in position control Systems.

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

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



Use of Module E020

The rapid input of interruption of PLCs in the series AL-600 and PICCOLO can be used for immediate processing from a point of input, being useful for quick control of positionings. With its activation, the module E020 is called for execution, carrying out the processing needed and the updating of output points through the instruction AES.

Module E020 can also be used in activating the devices or security procedures, activation devices or other applications which need fast updating.

The input of interruption is also used as a second counting input in PLCs in the series AL-600 and PICCOLO and in the CPU QK600, not being necessary for any adjustment in the equipment to select its function. If module E020 is present in the PLC, this is called to each actioning of the input. If the applications program calls the module **F-CONT.005**, this carries out the reading and writing of the counting value, increasing each a each input actioning. If required, it can use this input with both the functions, with the module **F-CONT.005** counting the number of times that the module E020 was actioned.

Module E020 only acts in the PLCs AL-600/4 AL-600/8 and AL-600/16 starting from version 1.20 of the executive software, and in the PLCs QK600, PL102 and PL103. Only these PLCs have a rapid input of interruption which actions the E020.

Care in the Use of Module E018

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

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



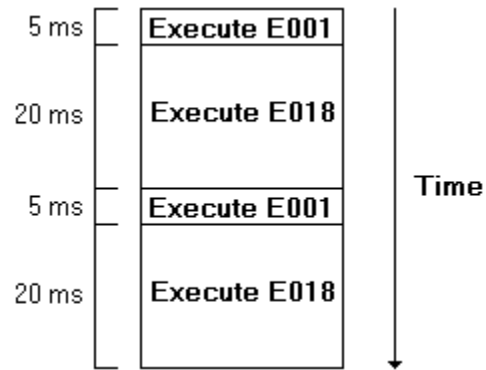


Figure 2-25 Care in Use of Module E018

If the execution of module E018 takes more than the time interval programmed for their calls, the PLC passes to error status, sending the message “Re-input in module E018” in the window. **Information** (options **Communication, Status, Information** starting from the main menu). In this situation, the period of the call used should be increased or the execution time of module E018 should be reduced so that the programming project can be executed correctly.

The instructions behave the same when executed in module E018, except in relation to some other particular characteristics. The timers (TEE and TED) continue to count the time at each 100 ms, any which is in the period of actioning 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 actioned through them obeying the same rules of programming applying to module E018. The maximum number of levels of call from modules used in the module E018 should be added to the maximum level used in E001, the sum having to be less than the limit of the system (18 levels).

Care in Programming Module E020

Some special care is needed in programming module E020. Its processing time should be short, mainly if input of interruption was actioned often, so as not to increase excessively the overall cycle time of the applications program.



If the of interruption is actioned 30 ms, for example, and the execution time of E020 is 25 ms, it restores only 5 ms for the execution of the main program before which the module is called again. This situation considerably increases the cycle time of module E001.

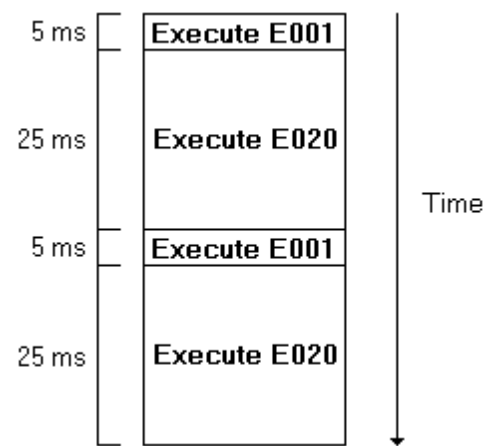


Figure 2-26 Care in Use of Module E020

If module E020 is being executed and a new actioning occurs in the PLC's interruption input, this actioning is ignored, continuing normally with the module's execution. This situation does not cause the change to error mode, the PLC remaining in normal execution. Therefore, the PLC ignores actionings of the rapid interruption input which occur in times less than the execution time of E020.

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

The call from the module depends on the process which is being controlled, not occurring in periodic form. This characteristic makes the use of the times in E020 impractical, that is to say, the instructions TEE and TED should not be used in it. The pulse relay (PLC) actions its output during an execution of module E020, 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 actioned through them obeying the same programming rules applying to module E020. The maximum number of levels of module calls used in the module E020 should be increased to the maximum level used in E001 and E018, the sum having to be less than the limit of the system (18 levels).



Use of Operands in Programming of Modules E018 and E020

Other care necessary is with the data sharing between the modules E018 or E020 and the rest present in the programmable controller. The interruptions can occur at any point of the main program of execution cycle (module E001 or modules P or F called through it), including during the processing of its instructions. As the operands are all of common use to any programming project module, care should be taken not to inadvertently use, in modules E018 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 should 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 should be used in the modules pertaining to the normal execution cycle and not in modules E018 or E020.

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

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

A similar situation occurs for reel instructions. If some point of an operand is modified in the main program through a reel, it is not permitted to change any point pertaining to the whole octet of the same operand in Modules E018 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 reel and are only tested for another module with contact instructions, for example. The opposite situation is permitted, that is to say the operand points changed in the main program through coils can be tested in Modules E018 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 operands in the rest of it. Also it cannot update input images in Modules E018 or E020 with the instruction AES, but only read directly the values of the I/O modules to memory operands through the instruction MES, and use these memories in contacts to carry out the processing in the interruption modules.

The updating of output octets in Modules E018 or E020 (through the instruction AES) is possible, since the points pertaining to these octets are action 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 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 actioned, 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 actioned for interruption. During the compaction of the rest of the modules, still, the Modules E018 or E020 continue being executed.



Simultaneous Use of Modules E018 and E020

It is possible to simultaneously use Module E018 (actioned periodically through time interruption) with the Module E020 in the PLCs AL-600/4, AL-600/8, AL-600/16, QK600, PL102 and PL103.

Execution priorities do not exist for the interruptions of the two modules. That is to say if Module E020 is being executed and the next time interruption occurs, the processing of E020 is interrupted and the Module E018 is executed, returning afterwards to execution interrupted by E020.

In this way, if Module E018 is being executed and the input interruption is actioned, the processing of E018 is interrupted, Module E020 is executed, returning after to the execution interrupted by E018.

Some care should be taken in the simultaneous use of the two modules.

The execution time of the module should add up to the maximum number of call levels of the Modules E001, E018 and E020, the result having to be less or equal to the maximum number allowed (18 levels).

The two modules should use exclusive operands, following the rules in the section **Use of Operands in the Programming of 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 follows.

Information about the Status of the PLC

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



Shortcut:



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

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



Monitoring

Through MasterTool it is possible to monitor the values of one 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 visualised directly in the PLC allowing the verification of its functioning.

For more information about how to carry out the monitoring, c.f. items **Monitoring Simple Operands**, **Monitoring Table Operands** and **Monitoring Programs** in the section **Communicating with the PLC or Router** in chapter 5 of the MasterTool User's Manual.

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

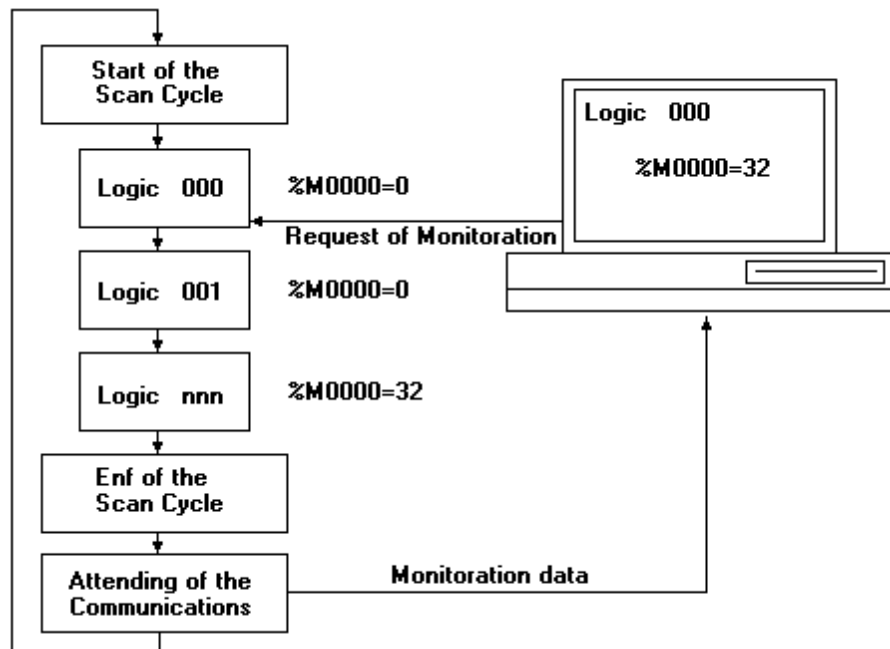


Figure 2-27 Incoherent Situation in Logic Monitoring

Forcing

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

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

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

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

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

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

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

WARNING:

Incoherent situations can be visualised 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 cancelling the forcing previously determined. The operands %E return to have their values updated according to the input modules, while the output modules receive the values obtained in the processing of the applications program.

WARNING:

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

For further information about how to free forced operands, c.f. item **Liberating Forced Operands** in the section **Communicating with the PLC or Router** in chapter 5 of the MasterTool User's Manual.

Disabling the Outputs

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

For more information regarding the disabling of outputs, c.f. item **Enabling and Disabling the Outputs** in the section **Communicating with the PLC or Router** in chapter 5 of the MasterTool User's Manual.

WARNING:

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



Modifications in the Program

The loading of the modules during the execution of the programming project (loading “on line”) makes possible successive modifications and messages from the module in the dedepuration for the programmable controller.

In this mode it is not necessary to reinitialize the control application program not even a change of status from programmable controller to each alteration carried out in a module.

WARNING:

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

WARNING:

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

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

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

Cycling Mode

The execution of the programming project in cycling mode makes use, in the verification of the functioning of rapid brakes in the applications program.

The rest of the facilities of dedepuration continue acting in the same way as in the execution mode (monitoring, forcing, loading and other operations with modules).

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



Managing Programming Project Modules

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

WARNING:

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

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

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

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

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

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

Compaction

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



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

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

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

Example:

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

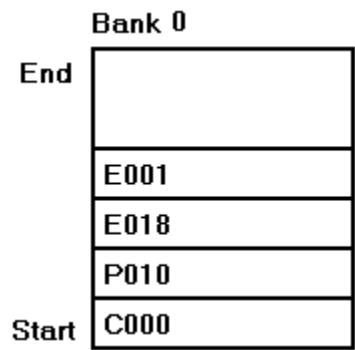


Figure 2-28 Compaction of RAM Memory



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

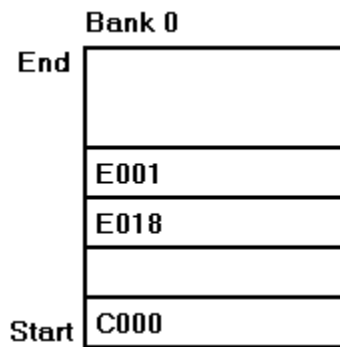


Figure 2-29 Compaction of RAM Memory-2

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

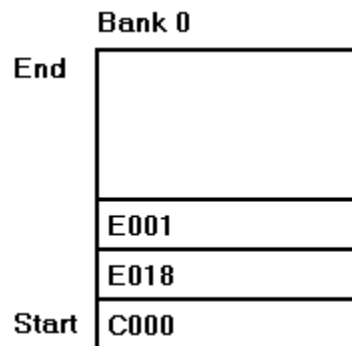


Figure 2-30 Compaction of RAM Memory-3

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 actioned manually, through MasterTool (options **Communication, Modules, Compact RAM** from the main menu).

This procedure is common when different loadings of modules in execution mode are carried out (loads “on line”), typically when a module is being purified, needing successive alterations and transmissions for the PLC.

WARNING:

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

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.

Use of EPROM or EPROM Flash Memory

The controllers can count 2 different types of to permanently record from the applications program:

- **EPROM Memory** - is shown in cartridge form, connected in the front panel of the PLC. The programming project is recorded in cartridges through the recorder AL-2860, connected to MasterTool, and removed with appropriate erasers, using ultraviolet light. Used in the controllers AL-3003 and AL-3004.
- **EPROM flash Memory** - is placed on the board of the PLC's circuit, not being necessary to remove it for recording or erasing programs. These operations are carried out through their own controller, through MasterTool commands. Used in controllers from the series AL-600, AL-2000, PL104, PL105 and QUARK.

Both types have peculiar characteristics: they can be partially recorded, however they do not allow the partial erasure of their contents. That is to say, it is only possible to erase all the contents of memory in the erasing.

Each PLC only uses the two types mentioned before. No PLC has both types of EPROM memory.



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

In this manual, in Help and Master, the name EPROM refers indistinctly to memory used to permanently record the programming project of the PLC, that is to say the type of EPROM cartridge or EPROM flash.

Transference of Modules from RAM to Flash:

After they are loaded into the RAM memory of the program, through the serial of the PLC, the programming project's modules can be transferred to EPROM flash. This command is only usable in PLC's which have flash memory. For further information about the transferring modules from RAM to EPROM Flash c.f. item **Transferring Modules from RAM to EPROM Flash** in the section **Communicating with the PLC or Router** in chapter 5 of the MasterTool User's Manual.

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

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

The EPROM memory of PL101, PL102 and PL103 make possible the carrying out of backup of RAM memory. It is only possible to transfer all the modules of RAM to EPROM or all the modules of EPROM to RAM. It is not possible to visualise the contents of the EPROM memory.

If PL101, PL102 or PL103 is turned on without there being programming modules in RAM memory all the existing program modules in EPROM memory are transferred to RAM.

In PL104 e PL105 flash EPROM memory is used to store executive program.



Transference of EPROM modules to RAM:

The modules present in EPROM flash memory or in EPROM cartridge can be transferred to the RAM memory of the program. For further information about how to transfer modules from EPROM to RAM, c.f. item **Transferring Modules from EPROM to RAM** in the section **Communicating with the PLC or Router** in chapter 5 of the MasterTool User's Manual.

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

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

Erasing and re-enabling modules in EPROM:

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

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

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

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

For further information regarding how to erase or re-enable modules, c.f. items **Erasing Modules in the PLC or Router** and **Re-enabling Modules in EPROM** in the section **Communicating with the PLC or Router** in chapter 5 of the MasterTool User's Manual.



Erasing from EPROM Memory:

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

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

To erase the EPROM flash memory, use the options **Communications, Modules, Erase Flash** which are the PLC in programming mode. The erasing can wait several seconds, depending on the capacity of the flash used in the PLC. For further information regarding how to erase the flash memory, c.f. item **Erasing the EPROM Flash Memory** in the section **Communicating with the PLC or Router** in chapter 5 of the MasterTool User's Manual.

Program Execution Cycle Times

The maximum time possible for the execution of a complete cycle of the applications program in the programmable controller is configurable for 100ms to 800ms. That is to say, the complete execution of a verification of Module E001 cannot be extended for more than the value configured, including the calls to the Modules P and F and the actioning 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 verify the execution times of the applications program through the PLC's information window (options **Communication, Status, Information** starting from the main menu) various execution cycle times being given, specified as follows:

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



- **Maximum cycle time:** shows the longest time between all the cycles carried out since the passing of the PLC into execution or cycling mode.
- **Minimum cycle time:** shows the shortest time between all the cycles carried out since the passing of the PLC to execution or cycling mode.

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

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

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

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

WARNING:

Each programming project should be examined carefully in the search for situations which will cause the longer execution times. These situations should be simulated and the times averaged, verifying if they are not excessive. This procedure should be carried the same in the programming project with cycle times well below the limit, to ensure it functions well.

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

WARNING:

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



☺HINT:

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

☺HINT:

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

Protection Levels of the PLC

CPUs in the series AL-600, AL-2000, QUARK and Piccolo 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-authorised operators.

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

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



Level 0 - all the PLC's operands are permitted.

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

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

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

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

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

WARNING:

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

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

For more information about how to alter the protection level and the password of the PLC, c.f. items **Altering the Protection Level** and **Altering the Password** in the section **Communicating with the PLC or Router** in chapter 5 of the MasterTool User's Manual.



Interlocking of Commands in the PLC

In the series AL-2000 and in the CPU QK2000/MSP in the Quark 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 PLC can receive a command to transfer from EPROM to RAM through ALNET II while the same command is being loaded in ALNET I.

Similar situations occur with the commands for transferring program modules from EPROM Memory to RAM to flash or erasing from flash memory.

The execution of these commands can be extended for several seconds, during these the PLC can receive other commands which conflict with operation in progress. For example, PLC can receive one command to erase the flash memory while a module may be being transferred to the same memory.

To resolve possible conflicts, there is a braking mechanism to execute some of the commands available in the PLC. These commands cannot be executed if the PLC is carrying out a specific operation. There are two internal signals, **loading module** (CM) and **compacting RAM** (CR) which are used for this intention. The tables 2-9 and 2-10 show the commands which use the braking and the actioning of the signals.

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



Operation carried out in PLC	Command blocked (ALNET I, ALNET II)	Signal ON
Loading of Modules	Loading of modules Transfer from EPROM to RAM Transfer from RAM to Flash Request to load modules Re-enabling of modules in EPROM Erasing of EPROM Flash Compaction	CM
Transfer from EPROM to RAM	Loading of modules Transfer from EPROM to RAM Transfer from RAM to Flash Request to load modules Re-enabling of modules in EPROM Erasing of EPROM Flash Compaction	CM
Transferência de RAM para Flash	Loading of modules Transfer from EPROM to RAM Transfer from RAM to Flash Request to load modules Re-enabling of modules in EPROM Erasing of EPROM Flash Compaction	CM
Erasing from EPROM Flash	Loading of modules Transfer from EPROM to RAM Transfer from RAM to Flash Request to load modules Re-enabling of modules in EPROM Erasing of EPROM Flash Compaction	CM
Legend: CM - Loading Module		

Table 2-9 Braking of Commands in the PLC (loading module)



Operation carried out in PLC	Command blocked (ALNET I, ALNET II)	Signal ON
Compaction	Load modules Transfer from EPROM to RAM Transfer from RAM to Flash Request loading of modules Re-enabling of modules in EPROM Removal of modules Compaction	CR
Legend: CR - Compacting RAM		

Table 2-10 Braking of Commands in the PLC (Compacting RAM)

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

WARNING:

The braking of the commands does not occurs in PLCs AL-3003 and AL-3004. The LED of the front panel is not lit when the operations in tables 2-8 and 2-9 are carried out.



Router Project

Building up a Router Project

A router project is made up of on single configuration module of networks (Module R) which have the necessary parameters to configure the routing of the network.

The network configuration module is called for execution through the executive software (operational system of the router). When stored on disk, the programming project corresponding to an archive, named as shown in figure 2-30.

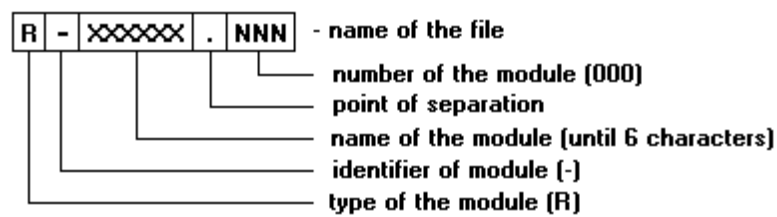


Figure 2-31 Format of Name of Module R in file

Example: R-GAT1.000

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

Example: R000

WARNING:

The file name corresponding to a program module should not be altered through another application of Windows™. To change the name of a file, it should be read and saved with the required name through MasterTool. C.f. section **Saving a Module with Another Name** in chapter 5 of the MasterTool User’s Manual.

If the file name is modified through another application of Windows™, it can be attributed with an invalid name, not being able to be read to MasterTool or loaded in the PLC.



Module R - Configuration of Router

Module R contains the parameters for configuring the router. The definition of the parameters contained in the router is carried out through the editing window of Module R, c.f. section **Configuring Module R**, in chapter 5 of the MasterTool User's Manual.

There is only one Module R for the router project, having the project name as its own name, for better identification.

Contents of Module R:

- **Declaration of CPU Model:** specifies the CPU model in which the router project is to be executed. The editing of this parameter is carried out in the editing window of Module R. For further information, c.f. item **Configuring the CPU Model** in the section **Configuring Module R** in chapter 5 of the MasterTool User's Manual.
- **Declaration of Channel Parameters:** specifies the configuration parameters relating to each channel of the router. For more information regarding the function of each router channel, c.f. the manual for the router being used. For more information about how to declare the parameters, c.f. the manual of the router used. The channel parameters are declared in the editing window of Module R. For more information about declaring the parameters c.f. item **Configuring the Channel Parameters** in the section **Configuring Module R** in chapter 5 of the MasterTool User's Manual.
- **Declaration of the Channel Routing:** specifies the routing table of the channel, that is to say, for whatever sub-network has to be revised the command for that arrives at the sub-network destination. This table is declared in the editing window of Module R. For more information about how to declare the routing table, c.f. item **Configuring the Channel Routing**, in the section **Configuring Module R** in chapter 5 of the MasterTool User's Manual.
- **Declaration of the Channel Redundancy:** specifies the parameters of redundancy of the channel: enabling or not of the redundancy, test period for the active connection and behind for commutation. They should be declared in the editing window of module R. For more information about how to declare the parameter of redundancy c.f. item **Configuring the Channel Redundancy** in the section **Configuring Module R** in chapter 5 of the MasterTool User's Manual.



Router Operation States

There are four states or operation modes of the router: Initialization, execution, programming, and error. The status in which the router finds itself is indicated in the LEDs in the front panel of the CPU, being able also to be consulted through MasterTool through the dialogue box **Status** (options **Communication, Status**, Starting from the main menu). To obtain specific information about the modes of operation, consult the user's manual of the router used.

- **Initialization Status:** the router initializes the various data structures used by the executive program and carries out consistencies in the router project present in memory. This status occurs after the powering of the router, passing after some seconds to the execution status. If a module R does not exist in memory the router passes to error mode.

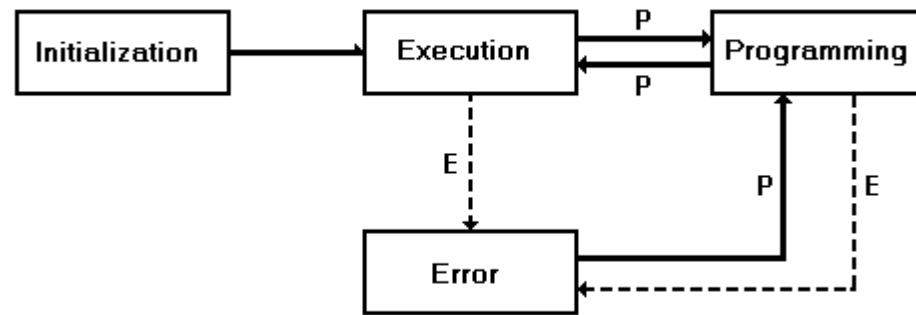
While the router is initializing, the command **Communication, Status, Programming** can be actioned, or equivalent in the tool bar, in doing this the router passes directly to programming status, instead of passing to execution mode. This procedure is useful to reinitialize the routers with modules R containing serious configuration errors.

- **Execution Status:** normally the router is found in this status executing the router according to module R. This status indicates that the router is executing the routing correctly.
- **Programming Status:** the router is not executing. The router is not executing. The router remains non-operational waiting for commands from MasterTool. This mode is normally used to load from the project module of the router to MasterTool through the serial channel.
- **Error Status:** indicates that there is some anomaly in the router during the processing of the router. The type of error occurring can be checked through the dialogue box (options **Communication, Status, Information** starting from the main menu), while the router is not in this status. The output from the error status is only possible passing the router to programming mode.

In normal conditions, the router can be in execution and programming modes, these modes being actioned through the MasterTool commands (options **Execution, Programming and Cycling** from the dialogue box **Status**, or their equivalent short cuts in the **Tool Bars**. If some incorrect functioning occurs in these modes, the router passes to error status. The recovery of the error mode is only possible by passing the router to programming mode. Figure 2-31 shows the possible ways to change status.



ROUTER OPERATION MODES



P - Command of the MASTERTOOL

E - Status of Error

Figure 2-32 Operating Statuses of the Router

In execution and programming modes, it is possible to load and read modules R to the serial channel of the router as well as monitor and force any operands used. These operations are not possible if the router is in error mode.



Instructions

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

List of Instructions

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

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

MasterTool instructions can be divided into 7 groups:

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

Conventions Used

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



Presentation of the Groups

The descriptions of each group follows this routine.










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

Example:

Instructions of the Relays Group

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

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

Name	Description of Name	Editing Sequence	Tool Bars
<u>RNA</u>	contract normally open	ALT, R, A	
<u>RNF</u>	contact normally closed	ALT, R, F	
<u>BOB</u>	simple coil	ALT, R, B	
<u>SLT</u>	jump coil	ALT, R, S	
<u>BBL</u>	connected coil	ALT, R, L	
<u>BBD</u>	disconnected coil	ALT, R, D	
<u>PLS</u>	pulse relay	ALT, R, P	
<u>FRM</u>	end of master relay	ALT, R, M	
<u>RM</u>	master relay	ALT, R, R	



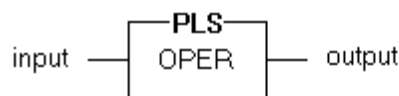
Presentation of the Instructions

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

1. The instruction is described with a little containing the name of the instruction and the description of the name. A figure presented as an instruction is visualised 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 actioned 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 behaviour. This item is only present in instructions which require major detailing of their functioning.
5. There are also other items which describe a specific characteristic of the instruction if it is necessary.

Example:

PLS - Pulse Relay



Description:

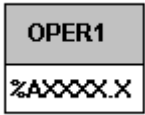
The instruction pulse relay generates a pulse from a scan on its output, that is to say, remains powered during a scan of the applications program when the status of its input may pass from turned off to powered.

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



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

Syntax:



Instructions of the Relays Group

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



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

Name	Description of Name	Editing Sequence	Tool Bars
<u>RNA</u>	contract normally open	ALT, R, A	
<u>RNF</u>	contact normally closed	ALT, R, F	
<u>BOB</u>	simple coil	ALT, R, B	
<u>SLT</u>	jump coil	ALT, R, S	
<u>BBL</u>	connected coil	ALT, R, L	
<u>BBD</u>	disconnected coil	ALT, R, D	
<u>PLS</u>	pulse relay	ALT, R, P	
<u>FRM</u>	end of master relay	ALT, R, M	
<u>RM</u>	master relay	ALT, R, R	

Table 3-1 Instructions of Relays Group



Contacts

- **RNA** contact normally open

- **RNF** contact normally closed


Description:

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

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

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

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

Syntax:

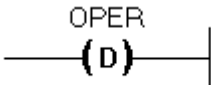
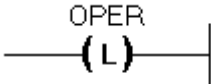
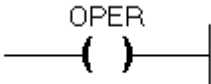
OPER1
%EXXXX.X
%SXXXX.X
%AXXXX.X
%MXXXX.X
%DXXXX.X
%DXXXXhX

Table 3-2 Syntax of the Instructions RNA and RNF



Coils

- BOB** Simple Coil
- BBL** Connected Coil
- BBD** Disconnected Coil



Description:

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

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

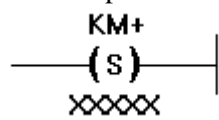
These instructions can only be positioned in column 7 of the logic.

Syntax:

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

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

SLT - Jump Coil



Description:

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

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

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

Example:

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

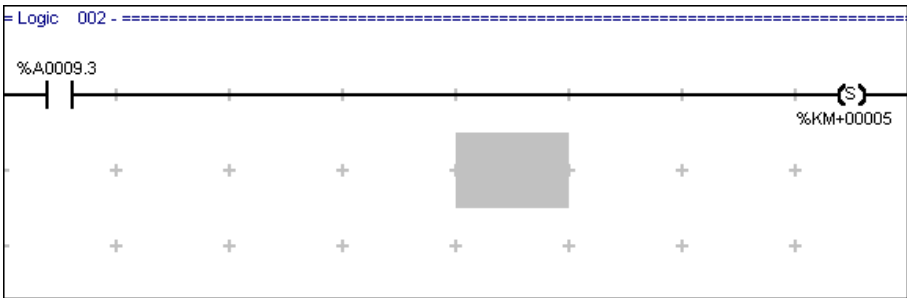


Figure 3-1 Example of SLT Instruction

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



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

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

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

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

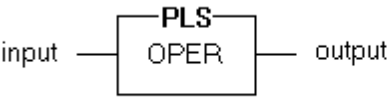
If the logic destination of a return jump is less than the first logic of the applications program, the execution is restarted starting from logic 0.

Syntax:

OPER1
%KM+XXXXX
%KM-XXXXX

Table 3-4 Syntax Instruction SLT

PLS - Pulse Relay



Description:

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

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

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

Syntax:

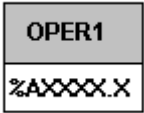


Table 3-5 Syntax of PLS Instruction



RM, FRM - Master Relay, End of Master Relay

- **RM** Master Relay



- **FRM** End of Master Relay



Description:

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

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

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

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

WARNING:

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

Instructions of Group Moving

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











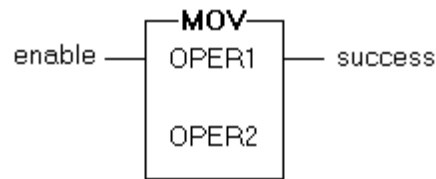
Name	Description of Name	Editing Sequence	Tool Bars
MO<u>V</u>	moving of simple operands	ALT, M, V	
MO<u>P</u>	moving of parts of operands	ALT, M, P	
MO<u>B</u>	moving of blocks of operands	ALT, M, B	
MO<u>T</u>	moving of tables of operands	ALT, M, T	
ME<u>S</u>	moving of inputs or outputs	ALT, M, E	
<u>A</u>ES	updating of inputs or outputs	ALT, M, A	
CE<u>S</u>	conversion of inputs or outputs	ALT, M, S	
<u>C</u>AB	load block	ALT, M, C	

Table 3-6 Instructions of Group Movements



MOV - Moving of Simple Operands



OPER1 - origin operand

OPER2 - destination operand

Description:

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

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 actioned.

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.



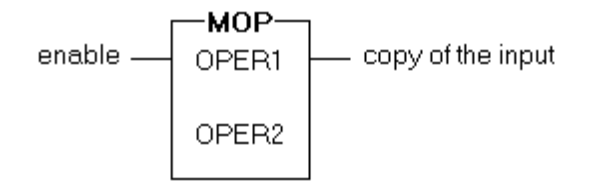
Syntax of the Instruction:

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

Table 3-7 Syntax of the Instruction MOV



MOP - Moving of Parts (Subdivisions) of Operands



OPER1 - origin operand
OPER2 - destination operand

Description:

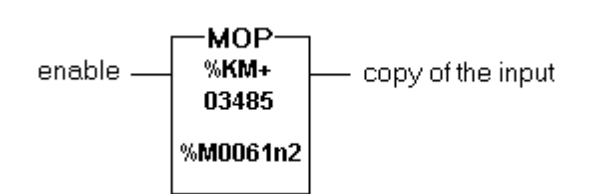
This instruction moves the contents of parts of simple operands (words, octets, nibbles, points) when the enabled input is powered. The conversion between types of operands is not carried out, only the moving of values.

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

WARNING:

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

Example:



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

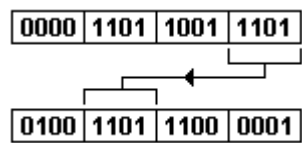


Figure 3-2 Example of Instruction MOP

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



Syntax:

OPER1	OPER2	OPER1	OPER2
%EXXXX.X	%EXXXX.X	%MXXXXbX	%MXXXXbX
%SXXXX.X	%SXXXX.X	%DXXXXbX	%DXXXXbX
%AXXXX.X	%AXXXX.X	%EXXXX	
%MXXXX.X	%MXXXX.X	%SXXXX	
%DXXXX.X	%DXXXX.X	%AXXXX	
%DXXXXhX	%DXXXXhX	%KMXXXXX	
%KMXXXXX		%KDXXXXX	
%KDXXXXX			

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

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

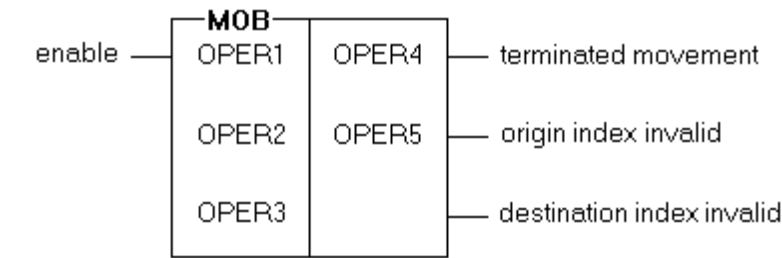
OPER1	OPER2
%DXXXXwX	%DXXXXwX
%MXXXX	
%KMXXXXX	
%KDXXXXX	

OPER1	OPER2
%DXXXXwX	%MXXXX

Table 3-8 Syntaxes of the Instruction MOP



MOB - Moving of Blocks of Operands



- OPER1 - first operand of origin block
- OPER2 - number of transfers to be carried out
- OPER3 - control operand
- OPER4 - first operand of destination block
- OPER5 - number of transfers for scan

Description:

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

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

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

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

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

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



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

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

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

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

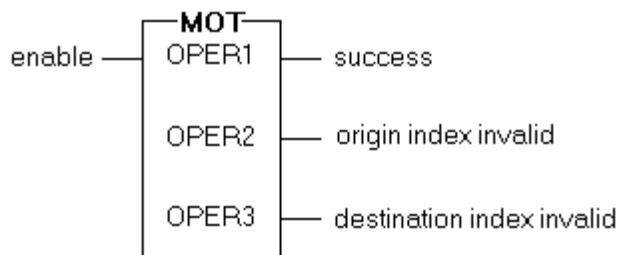
Syntax:

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

Table 3-9 Syntax of the Instruction MOB



MOT - Moving of Tables



OPER1 - origin table or origin operand

OPER2 - table index

OPER3 - destination operand or destination table

Description:

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

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

Reading from 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 (%TM, %M*TM, %TD)
- **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 (%M, %M*M, %D, %M*D)



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 actioned. 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 actioned.

Writing values into the 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 (%KM, %M, %M*M, %KD, %D, M*D)
- **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 (%TM, %M*TM, %TD, %M*TD) are transferred to

If the first operand indirectly references an undeclared the transfer of the contents is not carried out and the output **origin index invalid** is actioned. 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 actioned.

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



Syntax:

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

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

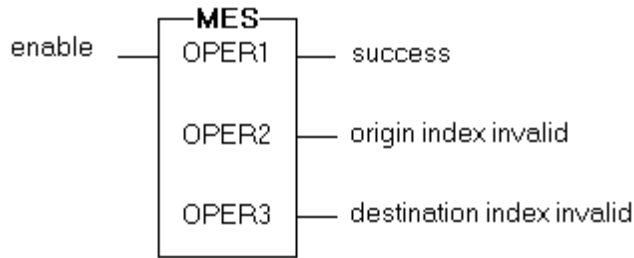
OPER1	OPER2	OPER3
%KM %M %M*M	%KM %M	%TM %M*TM

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

Table 3-10 Syntax of the Instructions MOT



MES - Moviment of Inputs/Outputs



OPER1 - first origin operand

OPER2 - number of octets to transfer

OPER3 - first destination operand

Description:

This instruction is used to transfer data directly between memory operands and octets of the input and output modules bus. It is possible to carry out readings of values of the octets of the bus or to write to it according to the operands programmed in the instruction.

The operand which occupies the first instruction cell (OPER1) is the origin operand, whose contents will be moved to the destination operand specified in the third cell (OPER3). OPER2 defines the number of octets to be transferred starting from the first origin and destination specified.

WARNING:

The number of octets to be transferred is limited to 255.

If a constant is programmed in the first cell (write the value in the bus), its value is moved to all the octets of the buses specified for operands in the second and third cells.

Always when the input **enable** is powered, one of the output of the instruction is powered, according to the following rules.

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

- the number of transfer specified in OPER2 is negative, zero, greater than the maximum number of octets in the bus of the PLC used (reading of the bus) or greater than the memories limit configured (writing to the bus)
- the first position read was greater than the maximum number of octets in the bus of the PLC used (%M*R programmed in OPER1)



- the first address of memory to be written is negative or greater than the last memory address configured (%M*M programmed in OPER1)

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

- the number of transfers specified in OPER2 is greater than the memories limit configured (reads from the bus) or greater than the maximum number of octets in the bus of the PLC used (writes to the bus)
- the first position written is greater than the maximum number of octets in the bus of the PLC used (%M*R programmed in OPER3).
- the first memory address to be read is negative or greater than the last memory address configured (%M*M programmed in OPER3)

The output **success** is powered when the outputs **origin index invalid** and **destination index invalid** are not powered.

This instruction is only used for special access to the bus. For its use, it is essential to know exactly what I/O module is placed in the physical position of the bus read or written to the MES and how it is accessed. As the module of and output supplied through ALTUS have specific instructions for their access, the MES instruction is not necessary in most applications programs.

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

Syntax:

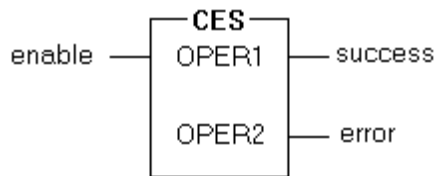
OPER1	OPER2	OPER3
%R %M*R	%KM %M	%M %M*M

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

Table 3-11 Syntaxes of the Instruction MES



CES - Conversion of Inputs/Output



OPER1 - origin operand

OPER2 - destination 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 and 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



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

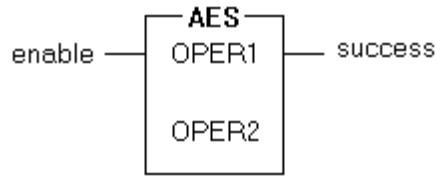
OPER1	OPER2
%KM	%R
%M	%M*R
%M*M	

Table 3-12 Syntaxes of the Instruction CES

This instruction is not available for the CPUs AL-3003 and AL-3004.



AES - Update 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 I/O 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 % I (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.

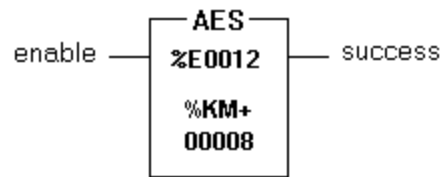
If the number of operands to update exceeds the number of operands declared, it is only possible to update from the type declared.

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 PLC’s configuration is 16 input octets (%E0000 to %ED015) and 8 output octets (%S0016 to %0023), the instruction shown will update only 4 octets (%E0012 to E0015). No output octet is updated.

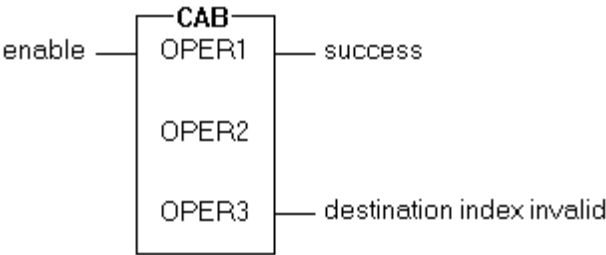
Syntax:

OPER1	OPER2
%E %S	%KM %M

Table 3-13 Syntaxes of the Instruction AES



CAB - Load Block



- OPER1 - initial operand or table to be loaded
- OPER2 - number of operands or positions of table
- OPER3 - table of constants to be loaded

Description:

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

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

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

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

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

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

Value of %M0000:00035.

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

Declaration of constant values table

	Values
0	0000
1	00000
2	00000
3	00000
4	00000
5	00000
6	00000
7	00000
8	00000
9	00000

Type: M

Base: DEC

OK

Cancel

ASCII Editing

Initialize

Press the button to do ASCII editing

Press the button to Initialize the constant values with a specific value

Figure 3-3 Dialogue Box CAB - Values

To carry out the editing of the constants

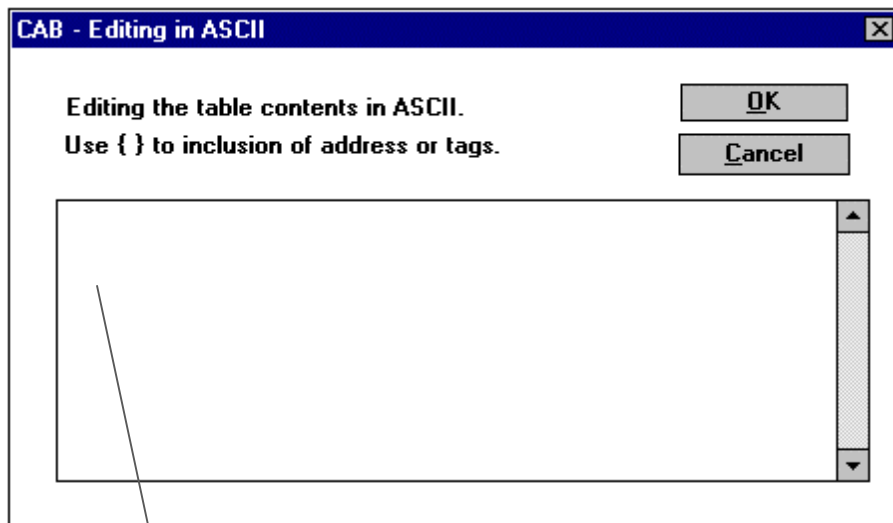
- 1. Position the cursor on the index to be edited. If it is necessary to roll the pages, the keys PAGE DOWN and PAGE UP or the vertical roll bar can be used.
- 2. Key in the constant value.

To carry out editing in ASCII

- 1. Select the button **Editing ASCII**. The dialogue box **CAB - Editing in ASCII** is shown.



2. Key in the text which it is required to be loaded in the constants of the CAB and select the Ok button.



Key in the text may be attributed at constant table of the CAB instructions.

Figure 3-4 Dialogue Box CAB - Editing in ASCII

To initialize the constants with a specific value

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



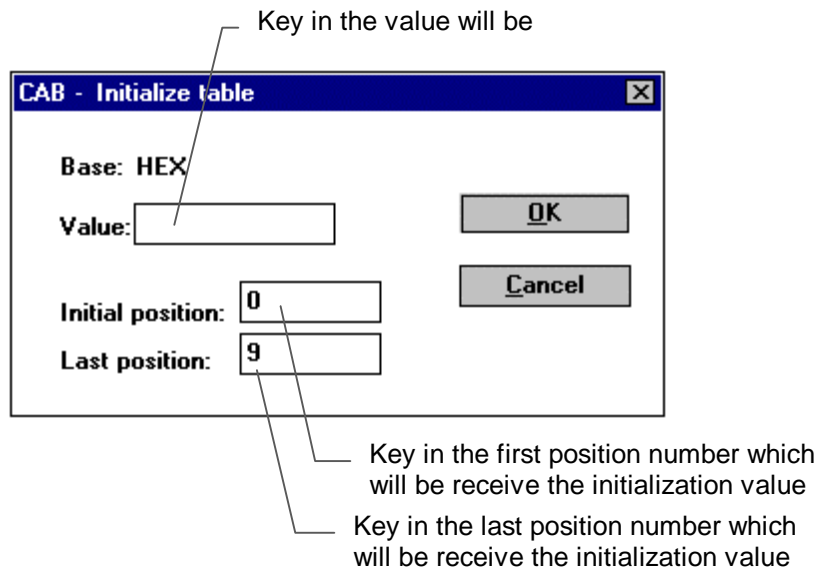


Figure 3-5 CAB - Initialize table

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

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



Syntax:

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

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

Table 3-14 Syntax of the Instruction CAB



Arithmetic Instructions of the Group

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












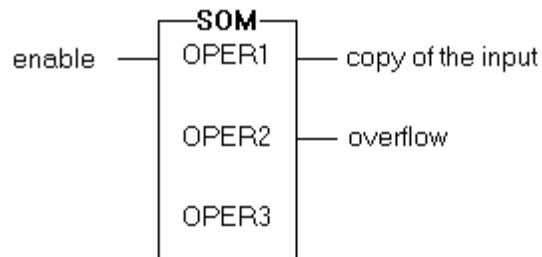
Name	Description of Name	Editing Sequence	Tool Bar
<u>S</u>OM	addition	ALT, A, S	
<u>S</u>UB	subtraction	ALT, A, B	
<u>M</u>UL	multiplication	ALT, A, M	
<u>D</u>IV	division	ALT, A, D	
<u>A</u>ND	function AND binary between operands	ALT, A, A	
<u>O</u>R	function OR binary between operands	ALT, A, O	
<u>X</u>OR	function OR EXCLUSIVE between operands	ALT, A, X	
<u>C</u>AR	load operands	ALT, A, C	
<u>I</u>GUAL	equals	ALT, A, I	
<u>M</u>ENOR	less than	ALT, A, N	
<u>M</u>AIOR	more than	ALT, A, R	

Table 3-15 Arithmetic Instructions of the Group



SOM - Addition



OPER1 - first plot

OPER2 - second plot

OPER3 - total

Description:

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

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

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



Syntax:

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

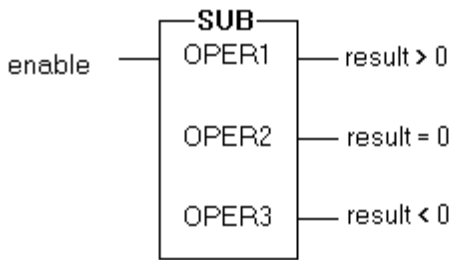
OPER1	OPER2	OPER3
%KF %F %KM %M	%KF %F %KM %M	%F %M %I

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

Table 3-16 Syntaxes of the Instruction SOM



SUB - Subtraction



OPER1 - first plot
OPER2 - second plot
OPER3 - result

Description:

This instruction carries out the subtraction arithmetic between operands. When **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 actioned according to the result of the subtraction.

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

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



Syntax:

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

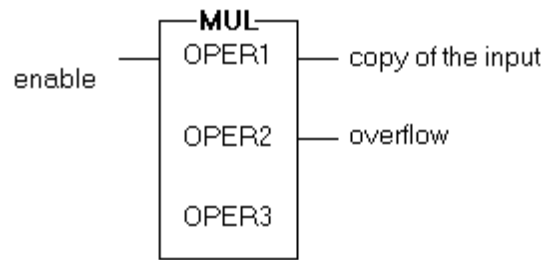
OPER1	OPER2	OPER3
%KF %F %KM %M	%KF %F %KM %M	%F %M %I

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

Table 3-17 Syntaxes of the Instruction SUB



MUL - Multiplication



OPER1 - multiplied
OPER2 - multiplier
OPER3 - product

Description:

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

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

Syntax:

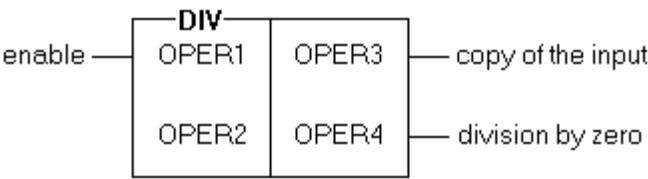
OPER1	OPER2	OPER3
%KF %F %KM %M	%KF %F %KM %M	%F %M %I

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

Table 3-18 Syntax of the Instruction MUL



DIV - Division



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

Description:

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

If the value of the second operand is zero, the output division by zero is actioned 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 actioned. If it is not actioned, OPER3 and OPER4 remain unchanged.



Syntax:

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

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

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

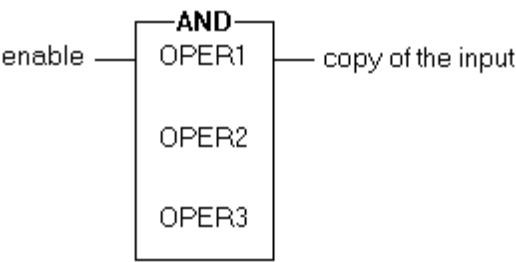
OPER1	OPER2	OPER3	OPER4
%KF %F %KM %M	%KF %F %KM %M	%F	%M(NU) %F (NU) %I (NU)

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

Table 3-19 Syntax of the Instruction DIV



AND - AND Binary between Operands



OPER1 - first operand
OPER2 - second operand
OPER3 - result

Description:

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

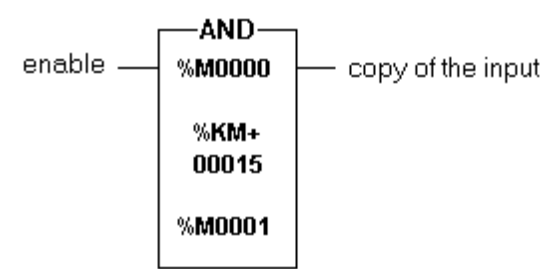
The operation is carried out point between the operands. The table to follow shows the possible combinations of the “and” point to point operation.

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

Table 3-20 Point to Point Operations



Example:



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

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

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

Syntax:

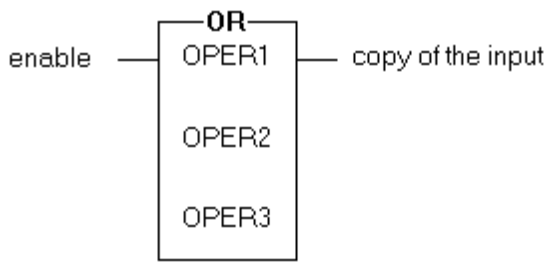
OPER1	OPER2	OPER3
%KM %M	%KM %M	%M

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

Table 3-21 Syntaxes of the Instruction AND



OR - Or Binary between Operands



OPER1 - first operand
OPER2 - second operand
OPER3 - result

Description:

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

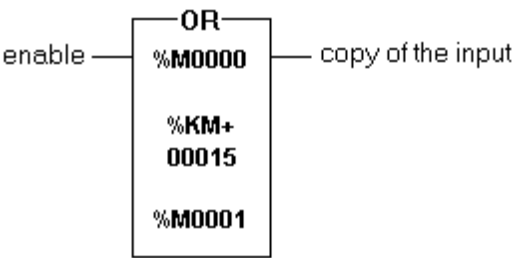
The operation is carried out point to point between the operands. The table to follow shows the possible combinations of the operation “or” point to point.

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

Table 3-22 Operations Point to Point (OR)



Example:



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

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

Syntax:

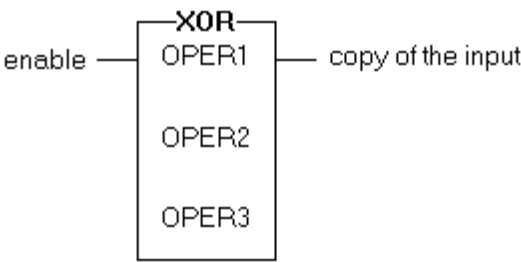
OPER1	OPER2	OPER3
%KM %M	%KM %M	%M

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

Table 3-23 Syntaxes of the Instruction OR



XOR - Or exclusive between Operands



OPER1 - first operand
OPER2 - second operand
OPER3 - result

Description:

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

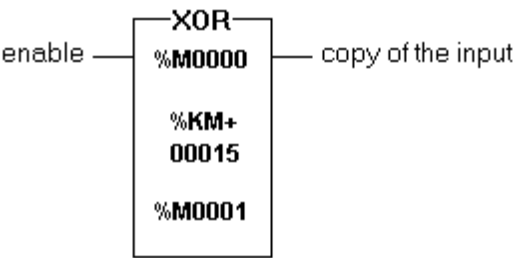
The operation is carried out point to point between the operands. The table to follow shows the possible combinations of the operation “or exclusive” point to point.

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

Table 3-24 Operations Point to Point (XOR)



Example:



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

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

Therefore, the decimal value 1603 is stored in M001.

Syntax:

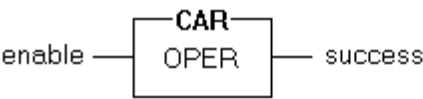
OPER1	OPER2	OPER3
%KM %M	%KM %M	%M

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

Table 3-25 Syntaxes of the Instruction XOR



CAR - Load Operands



OPER - operand to be loaded

Description:

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

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

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

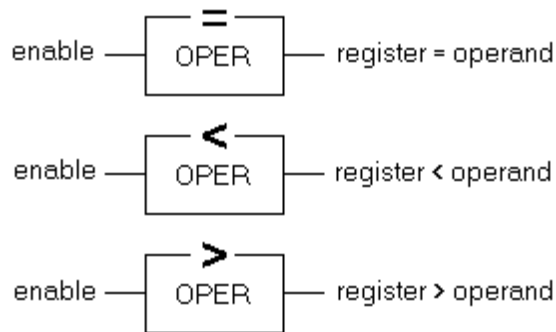
Syntax:

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

Table 3-26 Syntax of the Instruction CAR



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



OPER - operand to be compared

Description:

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

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).

WARNING:

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



Example:

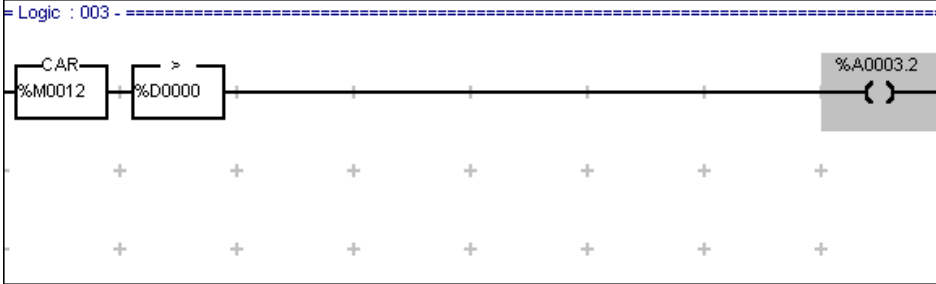


Figure 3-6 Example of Instructions of Comparison

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

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

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

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

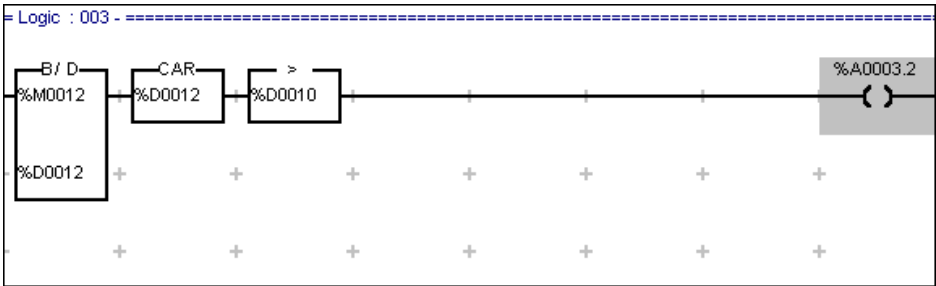


Figure 3-7 Example of the Instructions of Comparison



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

Example:

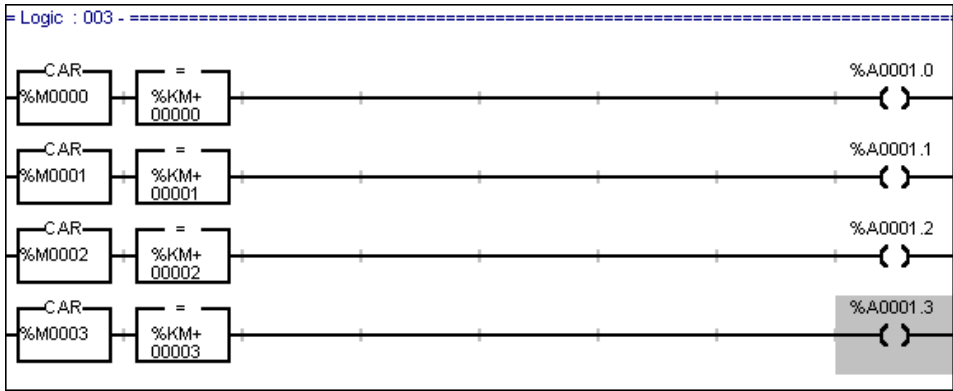


Figure 3-8 Incorrect Use of the Instruction CAR

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

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



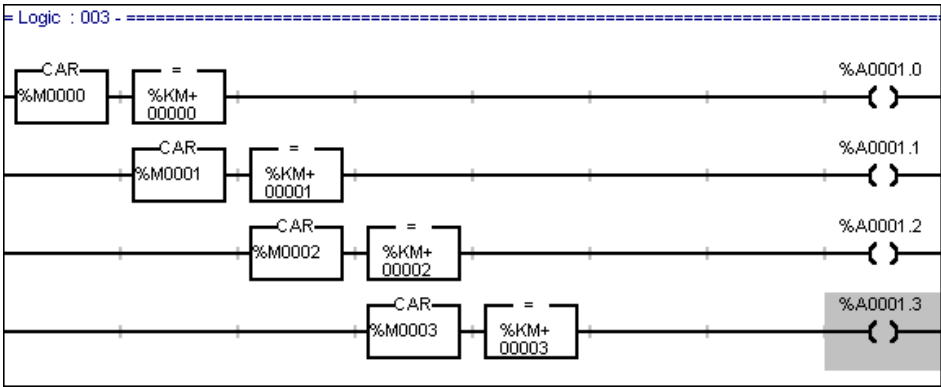


Figure 3-9 Correct Use of the Instructions CAR

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

Syntax:

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

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



Instructions of Group Counters

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





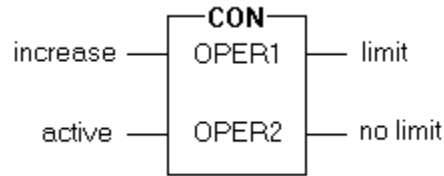
Name	Description of Name	Editing Sequence	Tool Bar
CON	simple counter	ALT, C, N	
COB	bidirectional counter	ALT, C, B	
TEE	timer in the powering	ALT, C, T	
TED	timer in the turning off	ALT, C, D	

Table 3-28 Instructions of Group Counters



CON - Simple Counter



OPER1 - counter

OPER2 - limit of count

Description:

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

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 the of the upper cell and can be of type %KM or operand %M referenced indirectly.

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

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

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

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

WARNING:

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



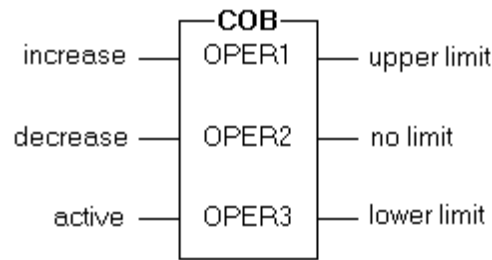
Syntax:

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

Table 3-29 Syntax of Instruction COM



COB - Bidirectional Counter



OPER1 - counter
OPER2 - count step
OPER3 - count limit

Description:

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

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

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

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

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

If the value of the first operand is equal to or less than zero, the output **lower limit** is actioned, 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 actioned. If the input **active** is not powered, the output **lower limit** is powered and the first operand is zeroed.

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



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

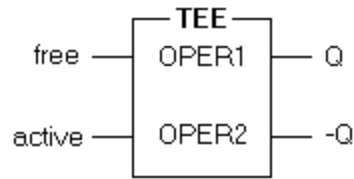
Syntax:

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

Table 3-30 Syntax of the Instruction COB



TEE - Timer in the Powering



OPER1 - time accumulator

OPER2 - time limit (tenths of seconds)

Description:

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

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

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

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

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

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

WARNING:

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



Diagram of Times:

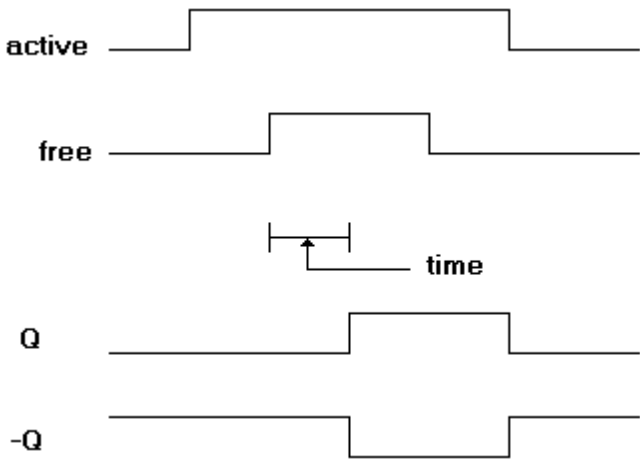


Figure 3-10 Diagram of Times of the Instruction TEE

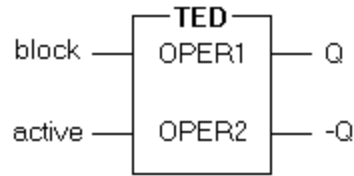
Syntax:

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

Table 3-31 Syntax of the Instruction TEE



TED - Timer in the Turning Off



OPER1 - time accumulator

OPER2 - time limit (tenths of seconds)

Description:

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

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

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

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

Actioning the input **block**, there is an interruption in the time count, while deactioning the input **active**, the time of the accumulator is zeroed and the output **Q** is deactioned.

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

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

WARNING:

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



Diagram of Times:

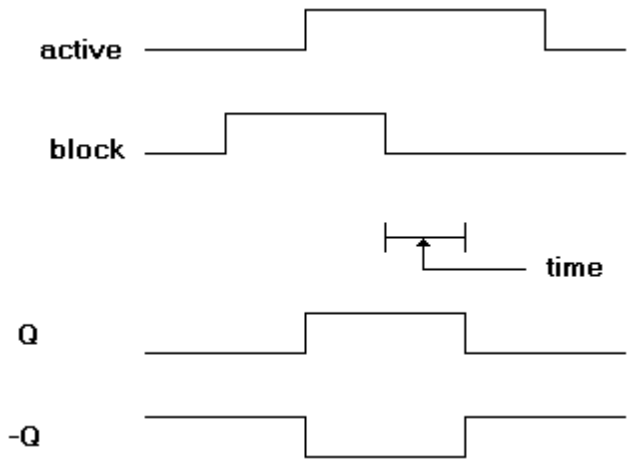


Figure 3-11 Diagram of Times of Instruction TED

Syntax:

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

Table 3-32 Syntax of the Instruction TED



Group Converter instructions

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





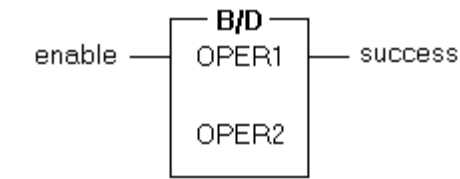
Name	Description of Name	Editing Sequence	Tool Bar
<u>B</u>IN/DEC	conversion binary-decimal	ALT, V, B	
<u>D</u>EC/BIN	conversion decimal-binary	ALT, V, D	
<u>A</u>NA/DIG	conversion analogue-digital	ALT, V, A	
<u>D</u>IG/ANA	conversion digital-analogue	ALT, V, G	

Table 3-33 Group Converter Instructions



B/D - Conversion Binary-Decimal



OPER1 - origin
OPER2 - destination

Description:

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

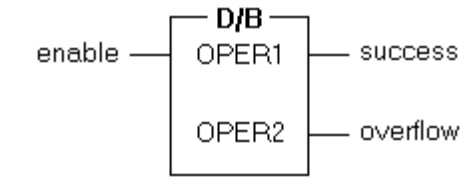
The binary value contained in the first operand (OPER1) is converted to decimal value and stored in the second operand (OPER2). The output **success** is actioned and the conversion is carried out correctly. If any invalid indirect access happens to the operand, the output **success** is not powered.

Syntax:

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

Table 3-34 Syntax of the Instruction B/D

D/B - Conversion Decimal-Binary



OPER1 - origin

OPER2 - destination

Description:

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

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

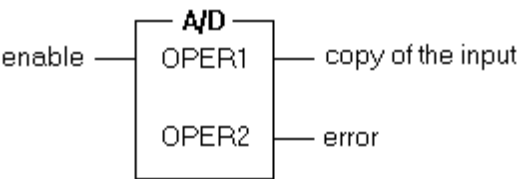
If the value converted results in a value greater than the maximum storable in operands %M, the output **success** is not powered, the limit value being stored in the destination operand. In this case, the output **overflow** is powered.

Syntax:

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

Table 3-35 Syntax of the Instruction D/B

A/D - Conversion Analog - Digital



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

Description:

This instruction converts the values read from an analog input module to numerical values stored in operands.



It is possible to read 1 or 8 channels by changing only the specification of the first operand, which shows the address in the bus occupied by module A/D. This module should be specified in the declaration of the bus, carried out in MasterTool. The address to be programmed in OPER1 can be obtained directly in MasterTool. The values converted are placed in operands of type memory, defined in OPER2.

The conversion is carried out only if the input **enable** is powered.

If OPER1 is with subdivision of type point (%RXXXX.X), the conversion is only carried for the channel of the module relative to the point. The points .0 to .7 of the operand correspond to the channels of the module, respectively. In this format, the execution time for the instruction is significantly less than the conversion of the 8 channels, being suitable, for example, for use in modules of program E018, actioned for time interruption.

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

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

The modules available to carry out the conversion A/D are shown as follows. The values converted for the instruction belong to a track related to the characteristic of each module:



AL-1103 (10 bits): values from 0000 to 1023

- AL-1116 (12 bits): values from 0000 to 4095
- AL-1119 and QK1119 (12 bits): values from 0000 to 4095
- QK1136 (12 bits): values from 0000 to 3999, with indication of over flow (4000 to 4095)
- AL-1139 e QK1139 (12 bits): values from 0000 to 3999, with indication of over flow (4000 to 4095)

The output of error for the instruction is activated in some of he following situations:

- module declared in the bus is invalid for the instruction (it is not one of the modules previously related)
- attempt to access the operands not declared
- conversion error (except AL-1103)

Syntax:

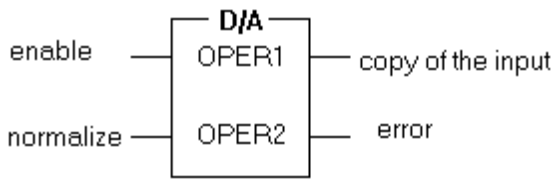
OPER1	OPER2
%RXXXX %RXXXX.X	%M

Table 3-36 Syntax of Instruction A/D

This instruction is not available for the CPUs AL-3003, AL-3004, PL101, PL102, PL103, PL104 and PL105.



D/A - Conversion Digital - Analog



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

Description:

This instruction converts the numerical values of memories to analog signals. The values are converted through cards of analog output AL-1203, AL-1214, AL-1222 or QK1222 being possible to convert from 1 or 4 channels using only one D/A instruction.

The first operand specifies the first memory with the value to convert.

The second operand indicates the address of the D/A module in the module bus. The module should be specified in the declaration of the bus, carried out in MasterTool. The address to be programmed in OPER2 can be obtained directly through MasterTool.

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

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

If OPER2 is specified as %RXXXX (conversion of 4 channels), the values to be converted are obtained from the memory declared in OPER1 and 3 subsequent ones.

The modules available to carry out the D/A conversion are shown as follows. The values converted by the instruction belong to a track related to the characteristic of each module:



Output in Tension

Module	Resolution	Normalization	Track
AL-1203	10 bits	not used	0000 to 1000
AL-1214	10 bits	not used	0000 to 1000
AL-1222, QK1222	12 bits	disconnected	0000 to 4000
QK1222, QK1222	12 bits	connected	-2000 to +2000

Table 3-37 Instruction D/A - Output in Tension

Output in Current

Module	Resolution	Normalization	Track
AL-1203	10 bits	not used	0000 to 1000
AL-1214	10 bits	not used	0000 to 1000
AL-1222, QK1222	11 bits	disconnected	0000 to 2000

Table 3-38 Instruction D/A - Output in Current

The values of AL-1222 and QK1222 converted still depend the input **normalize**, which converts symmetrical values when powered. This becomes useful when it is necessary to work with negative values, for example in the tension track of +/-10V.

There is no Normalization for modules AL-1203 and AL-1214, only for AL-1222 and QK1222. However, the Normalization is only possible for operation in tension mode.

WARNING:

In current mode the input **normalize** should not be powered.

The AL-1222 and QK1222 can work with the 4 outputs in tension mode or current mode, or the two modes simultaneously. The selection of the operation mode is achieved by the user through the programming of the addressing of the module in OPER2:

- If %RXXXX is even, convert current.
- If %RXXXX is odd, convert tension.



Example:

If the module is placed in the address %R0024 of the bus, if the instruction is programmed with %R0024, the AL-1222 and QK1222 will operate in current mode. If it is programmed with %R0025, it will operate in tension mode.

WARNING:
The instruction cannot be jumped during the execution of the applications program under penalty of the values sampled being incorrect.

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

- module declared in the bus is invalid for the instruction (is not one of the modules previously related)
- attempt to access the operands not declared

Syntax:

OPER1	OPER2
%M	%RXXXX %RXXXX.X

Table 3-39 Syntax of the Instruction D/A

This instruction is not available for the CPUs AL-3003, AL-3004, PL102, PL103, PL104 and PL105.



General Group Instructions

The general group instructions allow the testing and actionings of points indirectly, implementations of status machines, calls for procedures and functions, and writing and reading of operations in ALNET II.









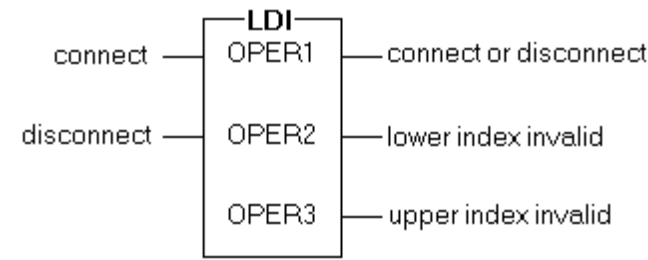
Name	Description of the Name	Editing Sequence	Tool Bar
<u>L</u>DI	connect or disconnect indexed points	ALT, G, L	
<u>T</u>EI	test of status of indexed points	ALT, G, T	
<u>S</u>EQ	sequencer	ALT, G, S	
<u>C</u>HP	call procedure module	ALT, G, P	
<u>C</u>HF	call function module	ALT, G, F	
<u>E</u>CR	write operands to another PLC	ALT, G, E	
<u>L</u>TR	read operands from other PLC	ALT, G, T, T	
<u>L</u>AI	free updating of images	ALT, G, I	

Table 3-40 General Group Instructions



LDI - Connect/Disconnect Indexed



OPER1 - address of point to be connected or disconnected
OPER2 - address lower limit
OPER3 - address upper limit

Description:

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

The first operand specifies the memory whose contents reference the auxiliary operand, input or output to be connected or disconnected. It should be declared as the operand of indirect access to the operand %E or %A (%MXXXX*E or %MXXXX*A). The same when the instruction is used to connect or disconnect points of output (%0), the representation in this operand will be as indirect access to the input (%MXXXX*E).

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

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

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

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

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

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

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

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

Example:

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

$$\text{OCTET} = 10$$

$$\text{POINT} = 5$$

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

The value to be loaded in OPER1 is 85.

WARNING:

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



Syntax:

OPER1	OPER2	OPER3
%M*E	%E	%E

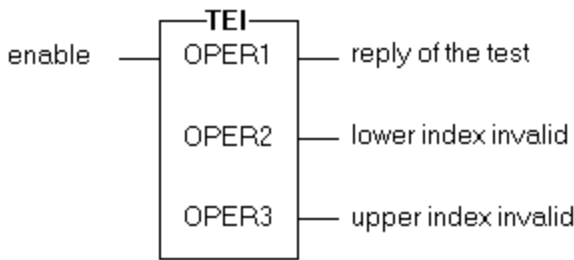
OPER1	OPER2	OPER3
%M*S	%S	%S

OPER1	OPER2	OPER3
%M*A	%A	%A

Table 3-41 Syntaxes of the Instruction LDI



TEI - Test of Indexed Status



OPER1 - address of point to be tested
OPER2 - address lower limit
OPER3 - address upper limit

Description:

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

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

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

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

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

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



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

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

Syntax:

OPER1	OPER2	OPER3
%M*E	%E	%E

OPER1	OPER2	OPER3
%M*S	%S	%S

OPER1	OPER2	OPER3
%M*A	%A	%A

Table 3-42 Syntaxes of the Instructions TEI



SEQ - Sequencer



- OPER1 - table of conditions or first table of statuses
- OPER2 - index of the table(s) (current status)
- OPER3 - operand base of the first series of conditions
- OPER4 - operand base of the second series of conditions

Description:

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

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

1000 Mode:

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

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

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

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

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



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

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

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

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

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

Example:

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

Address of operand = 30

Address of subdivision = 2

$$\text{VALUE} = (30 * 8) + 2 = 242$$

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).



3000 Mode:

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

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

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

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

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

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

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



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

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

ESTADO = contents of operand OPER2 (current status)

END3 = address of OPER3

END4 = address of OPER4

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

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

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

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

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

M<END1>.<SUB1> or A<SUB1> (first table) and 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:

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



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

%M0010 = 00001

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

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

Then the evolution starting from status 1 are:

For the first table:

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

For the second table:

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



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

%M0100.1	%A0020.1	%M0010	Observation
0	0	00001	Change of status Does not have a Change of status
0	1	00003	Change of status according to %TM001
1	0	00002	Change of status according to %TM000
1	1	00002	Change of status according to TM000 (OPER 3 has priority over OPER4).

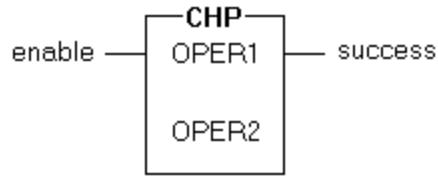
Syntax:

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

Table 3-43 Syntax of the Instruction SEQ



CHP - Procedure Module Call



OPER1 - name of module to call

OPER2 - number of module to call

Description:

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

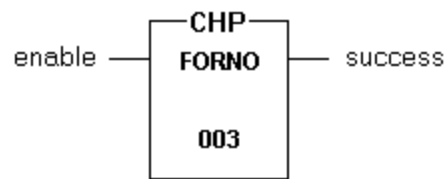
The first operand (OPER1) is documental and specifies the name of the module to be called. The second operand (OPER2) specifies the number of this module, the fact that the module called is of type procedure being implicit.

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

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



Example:



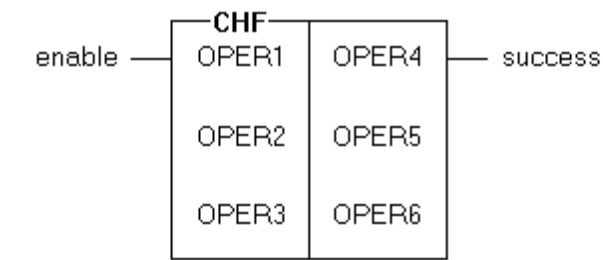
Syntax:

OPER1	OPER2
NOME	NÚMERO

Table 3-44 Syntax of CHP Instruction



CHF - Function Module Call



- OPER1 - name of module to call
- OPER2 - number of parameters to send
- OPER3 - number of parameters to return
- OPER4 - number of module to call
- OPER5 - list of parameters to send
- OPER6 - list of parameters to return

Description:

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

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

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

After calling for the execution of module F, the values of the operands declared in the list of parameters of output (option MasterTool **Parameters** in its programming) are copied to the operands declared in the list of operands to

return from the instruction CHF (OPER6). Having finalised the copy of the return, the processing continues in the instruction following the call.

WARNING:

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

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

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

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

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



To carry out the editing of parameters

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

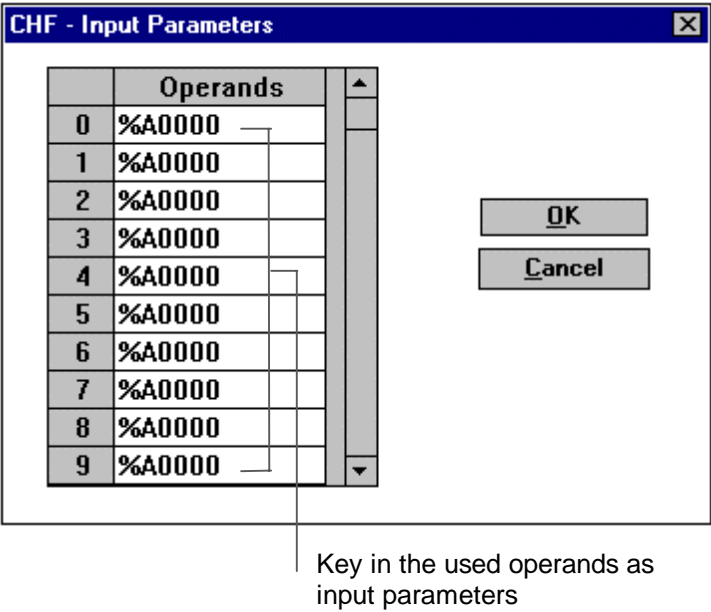


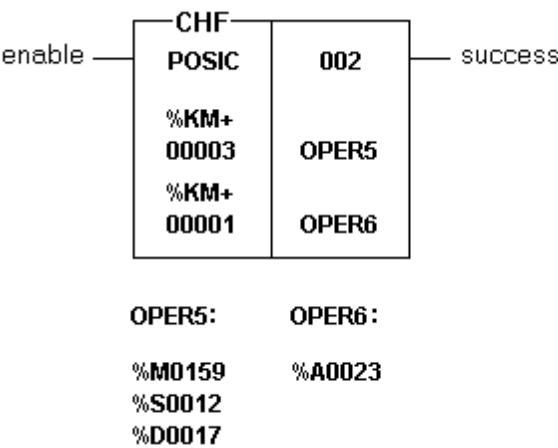
Figure 3-12 Dialogue Box CHF - Input Parameters

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

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



Example:



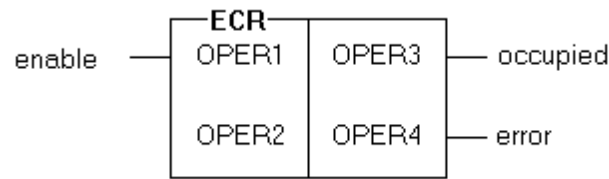
Syntax:

OPER1	OPER2	OPER3	OPER4	OPER5	OPER6
NOME	%KM	%KM	NÚMERO	%KM %KD %TM %TD %M %D %E %S %A %R	%KM %KD %TM %TD %M %D %E %S %A %R

Table 3-45 Syntax of Instruction CHF



ECR - Write from Operands to another PLC



- OPER1 - address of remote controller node
- OPER2 - address of remote controller sub-network
- OPER3 - control operand of instruction
- OPER4 - editing window of operands

Description:

This instruction carries out the reading of values of operands of the controller where it is being executed in operands present in other PLCs, through the communication network ALNET II. For its use, therefore, it is essential that the controller that executes is connected to other PLCs through ALNET II.

Through ECR it possible to transfer values of individual operands or of groups of operands, being possible to program up to 6 different communications in one instruction.

The ECR can be programmed for priority, send an urgent communication, processed through “bridges” and for the PLC destination of the common communications. The ECR priority allows only one communication, being useful for the signalling of alarms or emergency situations between PLCs.

To program the instruction, it should declare in the first and second cells (OPER1 and OPER2), the address of the node and sub-network of the programmable controller destination which will receive the values written.

These operands are programmed as constants of memory type (%KM) and have the same significance as the address configured in the options **Communication, Address, Sub-network and Communication, Address, Node.**

The table 3-46 shows the values possible for node addresses in the sub-network.

Sub-network	Node	Type of Communication
000	000	point-to-point
000	001 a 255	ALNET I
001 a 063	001 a 031	ALNET II with one node
100	001 a 015	ALNET II with multicast group in all the sub-networks
101 a 163	001 a 015	ALNET II with multicast group in a specific sub-network
200	xxx	ALNET II in broadcast for all the sub-network
201 a 263	xxx	ALNET II in broadcast for a specific sub-network

Table 3-46 Addresses of Node and Sub-network

The address of the Sub-network equal to 000 show that the communication is carried out using ALNET I and that the value contained in option **Node** shows the node that receives the communication.

The address of node 000 determines that all the PLCs in the network may hear and respond to the command sent. The specification of the address of the node in the band from 001 to 254 ensures that only the corresponding PLC identifies and responds to the command.

The address of the sub-network between the values 001 and 063 shows that the communication is carried out using ALNET II and that it is aimed at a single node indicated in the option **Node** (global multicast).

The sub-network address 100 shows that the communication is carried out using ALNET II and that it is aimed at all the nodes of the sub-network shown in the option **Sub-network less than 100** which belongs to the multicast group specified in option **Node** (local multicast).

The address of the sub-network 200 shows that the communication is carried out using ALNET II and is aimed at all the nodes of all the sub-networks (global broadcast). The value contained in the option **Node** is not relevant in this option.



The address of the sub-network between 201 and 263 shows that the communications is carried out using the ALNET II and aimed at all the nodes of the sub-network shown in the option **Sub-network less 200** (local broadcast). The value contained in option **Node** is not relevant in this option.

In the third cell (OPER3) a decimal operand (%D) should be declared to be used for its own instruction in the control of its processing.

WARNING:

The operand %D programmed in OPER3 cannot have its value modified at no other point of the applications program for the correct functioning of the ECR. Consequently, each new instruction ECR or LTR inserted in the applications program should use an operand %D different from the rest. This operand cannot be retentive .

To carry out the editing of the ECR parameters

- 1. Select button **PLC**. The dialogue box **ECR - Parameters** is displayed.

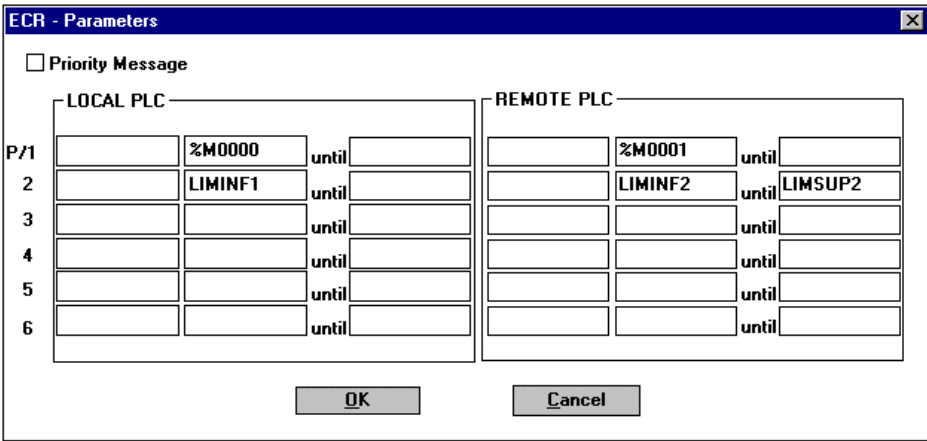


Figure 3-13 Dialogue Box ECR - Parameters

This dialogue box is divided in two parts: LOCAL PLC and REMOTE PLC, which each has three columns. In the three columns which make up the local PLC it is possible to define the operand or the group of operands whose values are sent to the programmable controller destination. The operands which will receive the value in the destination controller are declared, being able to be different from the local PLC. The dialogue box has six lines, allowing the definition of up to six different communications in the same ECR instruction for the same controller destination.



The item **Priority Message** allows the editing of a priority ECR when it is selected.

In the editing of a priority ECR, only the line for the editing of the communication, recognised by the initial **P/1** is valid, while in a non-priority ECR the communications **P/1, 2, 3, 4, 5** and **6** are valid. If during the change between priority ECR and non-priority there are already are edited operands, the communication of number **P/1** passes to be the communication of the priority ECR and vice-versa. The remaining operands are edited in the same way as in a non-priority ECR.

The operands specified for the local PLC exist in MasterTool according to the constant declarations in module C present in it, for they belong to the applications program which is being edited. The operands declared to the remote PLC do not have consistency in the types and addresses, for they belong to an applications program from another programmable controller. However, the number of bytes occupied by the block of operands declared in the local PLC should be equal to the number of bytes occupied by operands of the remote PLC in each communications so that the writing is carried out correctly. The maximum number of bytes possible to be occupied by a block of operands in each communication is limited to 220.

The following are the types of operands possible to be programmed for the local and remote PLC, with the correct arrangement of them in the editing columns and their respective significance.



LOCAL PLC or REMOTE PLC	Significance
%EXXXX	Individual operand %EXXXX
%SXXXX	Individual operand %SXXXX
%AXXXX	Individual operand %AXXXX
%MXXXX	Individual operand %MXXXX
%DXXXX	Individual operand %DXXXX
%EXXXX .. %EYYYY	Operands Group %EXXXX at %EYYYY
%SXXXX .. %SYYYY	Operands Group %SXXXX at %EYYYY
%AXXXX .. %AYYYY	Operands Group %AXXXX at %EYYYY
%MXXXX .. %MYYYY	Operands Group %MXXXX at %EYYYY
%DXXXX .. %DYYYY	Operands Group %DXXXX a %EYYYY
%TMXXXX YYY	Table %TMXXXX position YYY
%TDXXXX YYY	Table %TDXXXX position YYY
%TMXXXX III .. FFF	Table %TMXXXX position III a FFF
%TDXXXX III .. FFF	Table %TDXXXX position III a FFF

Table 3-47 Operand for Local and Remote PLCs in ECR

MasterTool allows the free editing of operands in the same line, making possible the change columns with the help of the arrow keys of horizontal movement. The consistencies are achieved in the attempt to change the line (vertical arrows) or confirmation of the contents editing in the window with the ENTER key. It is possible to give up the alterations carried out by actioning the ESC key, the instruction remaining with the previous contents at the opening of the editing window.

The following table shows the number of octets occupied by each type of operand possible to be programmed in the definitions to be written.



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

Table 3-48 Occupation in Bytes of the Operands of the ECR

The calculation of the number of bytes occupied in the declaration of the local and remote PLC is carried out by multiplying the number of operands declared by the octets of the corresponding type. In the table to follow some examples are shown.

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

Table 3-49 Example of Occupation in Bytes



In order to action the input **enable**, the communication is sent form the first writing present in the ECR, the output **occupied** being powered by it. At the moment when the communication is complete, the instruction sends the next writing, independently of the status of the enabling input, repeating this procedure for the rest of the communications existing in this instruction. At the end of the last writing, the output **occupied** of the ECR is turned off, with the sending of a pulse with the duration of one scan in the **error** output if it has not been possible to carry out some communication.

The statuses of the six communications of the instruction are placed in the first six nibbles of the operand D programmed in OPER3. The last two nibbles are used to control their processing.

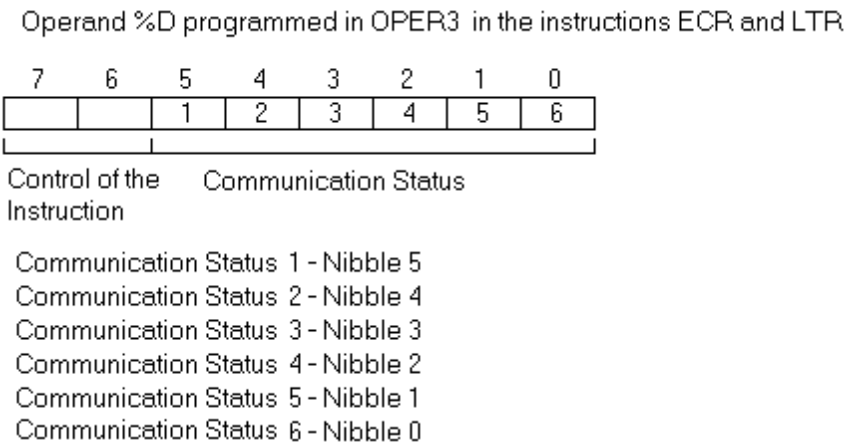


Figure 3-14 Control Operand for Instruction ECR and LTR

The communication status stored in each nibble is coded in the following way:

- 0 - communication with success
- 1 - operand not defined
- 2 - address of local controller equal to remote controller (communication for its own PLC)
- 3 - operand block invalid
- 4 - type of operand invalid
- 5 - time-out for transmission of packet
- 6 - not enough room in line



- 7 - buffer transmission fault
- 8 - time-out for requisition
- 9 - hardware error
- 10- remote PLC protégé

Briefly, in firing the execution of an ECR instruction all the communications existing in it are carried out, the same when its enabling input is turned off. When all the writings are completed, the next ECR or LTR instruction found in the applications program with the input **enable** powered become active, starting to process their communications.

WARNING:

The applications program cannot out jumps over the active ECR instruction or stop to execute the module which it contains, to ensure correct processing.

In an applications program being executed in the PLC, in a data moment, only one access instruction for ALNET II (ECR or LTR) is considered active, even if there are several instructions with enable actioned. The output engaged determines which instruction is active, be able to be used to synchronise the communications with the applications program. To avoid overloading in the information traffic of the network, it is advisable if possible to send the ECR instructions periodically, avoiding keeping them permanently enabled in the applications program. One procedure recommended is to disconnect the input **enable** as soon as the output **occupied** is powered, avoiding a new firing of the instruction after its termination.

The priority ECR does not follow the processing order of the non-priority ECRs being processed and transmitting their data as quickly as possible, to be enabled. For this reason a priority should not remain permanently enabled, having to be fired only in alarm situations or periodically. If the opposite is the case, it can prevent the remaining ECRs of the program from carrying out their communications or cause the exhaustion of the reception buffers of the destination PLC.

If the instruction is programmed by specifying the node address equal to the address of its own controller which it executes (writing the values of its own), the output error is powered.

If no operand has been defined in OPER4, the output **error** and **occupied** are turned off.

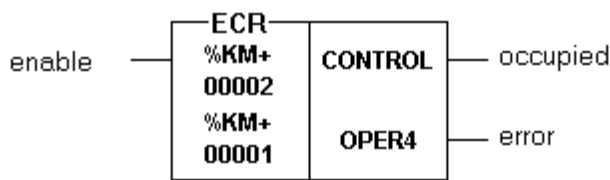


Syntax:

OPER1	OPER2	OPER3	OPER4
%KM	%KM	%D	communications

Table 3-50 Syntax of the Instruction ECR

Example:



Contents of the editing window in OPER4 of non priority ECR:

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

This instructions carries out writing into the programmable controller with the node address equal to 2 in the sub-network 1. Six communications are defined for it, transferring different types of data types between the PLCs. The communication 0 sends the contents of a memory operand into the local PLC for two auxiliary operands in the remote PLC, 2 octets being transferred. The communications 1, 2, 3, 4 and 5 transfer, respectively, 4, 12, 2, 8 and 10 octets between the controllers.



Contents of the editing window in OPER4 in a priority ECR:

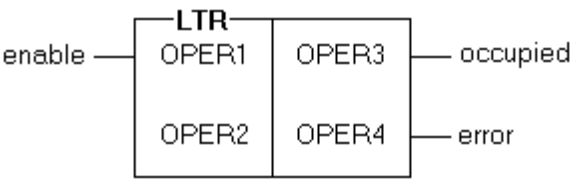
COM	Local PLC	Remote PLC
P/1	%M0004	%A0014 .. %A0015

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

This instruction can only be used in the CPUs AL-2000/MSP, AL-2002/MSP, AL-2003 and QK2000/MSP.



LTR - Reading of Operands from Another PLC



- OPER1 - node address of the remote controller
- OPER2 - sub-network address of the remote controller
- OPER3 - control operand of the instruction
- OPER4 - editing window of the operands

Description:

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

Through the LTR individual operand values or groups of operands can be read, being possible to program up to 6 different communications of reading in one instruction.

The programming of the instruction LTR is identical to ECR, observing the same restrictions. In the LTR, the transfer occurs of values of operands declared in the remote PLC to the local PLC, this being the one difference between the two.

WARNING:

The instructions LTR differs from the ECR in the possibility of priority messages, that is to say, it is not possible to edit a priority LTR.

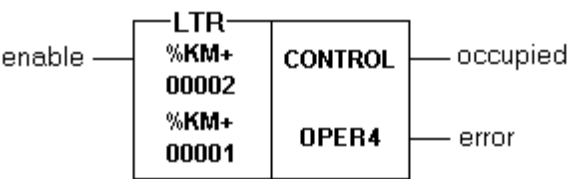
Syntax:

OPER1	OPER2	OPER3	OPER4
%KM	%KM	%D	communications

Table 3-51 Syntax of the LTR Instruction



Example:



Contents of the editing window in OPER4 in na LTR:

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

This instruction carries out readings in the programmable controller with the mode address equal to 2 in the sub-network 1. Six communications are defined for it, transferring different types of data between the PLCs. The communication 0 reads the contents of two auxiliary operands in the remote PLC for a memory operand in the local PLC, 2 octets being transferred. The communications 1, 2, 3, 4 and 5 transfer, respectively, 4, 12, 2, 8 and 10 octets between the programmable controllers.

This instruction can only be used in CPU AL-2000/MSP, AL-2002/MSP, AL-2003 and QK2000/MSP.



LAI - Free Updating of Images of Operands



Description:

The instruction frees the updating of the image of the operands and carries out the processing of the pending communications of ALNET II for the local PLC.

To return to the processing of the executive software, at the end of each scan, the PLC processes the requisitions for reading and other services which have been requested for other PLCs present in the network, during the execution of the applications program.

The programmable controller has an area of memory reserved for the storing of up to 32 communications received during the execution loop of the applications program, while the executive software does not process them. If the applications program has a relatively high execution time and the programmable controller receives many request for services from the network, it is possible that the PLC does not get answered, reaching the limit of 32 pending communications waiting for processing. In this case, the PLC returns an answer to the request indicating the impossibility of waiting for its communication.

The instruction LAI executes the processing of pending receptions and transmissions in the PLC, reducing the possibility of the situation described previously occurring and reducing the service times of the requests. Their use is recommended in applications programs with a long cycle time, having to be inserted in intermediate points of the modules, dividing them into passages with approximately 20 ms execution time.

WARNING:

The values of the operands of the applications program can be modified after the execution of an LAI, since other equipment connected to the network can be requesting writing in them. The influence of this fact should be considered when inserting this instruction in the applications program.

This instruction can only be used in CPUs AL-2000/MSP, AL-2002/MSP, AL-2003 and QK2000/MSP.



Group Connection Instructions

The group connection instructions allow the building of paths in series and in parallel as well as the inversion of the signal.




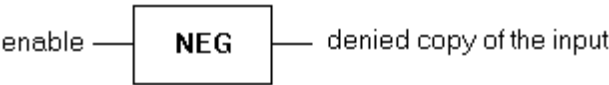
Name	Description of the Name	Editing Sequence	Tool Bar
LGH	horizontal connection	ALT, L, H	
LGN	denied connection	ALT, L, N	
LGV	vertical connection	ALT, L, V	

Table 3-52 Group Connection Instructions

LGH - Horizontal Connection



LGN - Denied Connection



LGV - Vertical Connection



Description:

The connections are auxiliary elements in the construction of the diagrams of relays, to interconnect the remaining instructions.

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



Function Modules

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

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

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

Each CPU module has a group of F modules of its own, contained in corresponding subdirectories in MasterTool. The table 4-1 shows the functions existing for each CPU, as well as the special modules of I/O which are accessed through them.

The CPUs PL102 and PL103 of the PICCOLO series have only the function modules **F-CONT.005**, **F-ANLOG.006** and **F-PID.033**.

The CPUs PL104 of the PICCOLO series have only the function modules **F-CONT.005**, **F-ANLOG.006**, **F-PID.033** and **F-relg.048**.

The CPUs PL105 of the PICCOLO series have only the function modules **F-PID.033** and **F-relg.048**.



Function	AL-600, QK600	AL-2000	AL-2002, AL-2003	QK80 0	QK80 1	PL104	PL105	QK2000	I/O Module
F-RELOG.000									AL-1420
F-LEDS.001									AL-1460
F-PT100.002									AL-1117, QK1117
F-TERMO.003									AL-1109, QK1109
F-CONTR.004									AL-1440, AL-1450, QK1450
F-CONT.005									AL-600
F-ANLOG.006									AL-600
F-EVENT.017									AL-3130, AL-3132
F-ALNET2.032									
F-PID.033									
F-RAIZN.034									
F-ARQ2.035									
F-ARQ4.036									
F-ARQ8.037									
F-ARQ12.038									
F-ARQ15.039									
F-ARQ16.040									
F-ARQ24.041									
F-ARQ31.042									
F-MOBT.043									
F-STMOD.045									
F-RELG.048									



Function	AL-600, QK600	AL-2000	AL-2002, AL-2003	QK80 0	QK80 1	PL104	PL105	QK2000	I/O Module
F-SINC.049									
F-RELG.061									
F-ALNET1.062									
F-IMP.063									
F-RECEP.064									
F-NORM.071									
F-COMPF.072									
F-ANDT.090									
F-ORT.091									
F-XORT.092									
F-NEGT.093									

Table 4-1 List of Function Modules Supplied through ALTUS

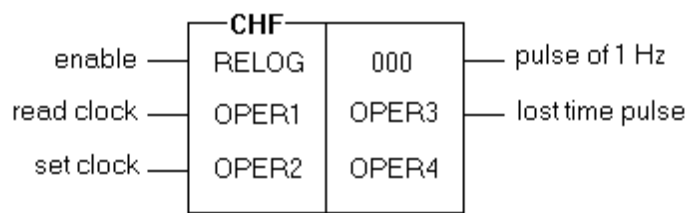
During the installation of MasterTool different modules are copied with the same name, being stored in different subdirectories, according to the type of CPU to which they are destined. The CPU having the same name, these modules differ in their contents. For example, the module **F-RELOG.000** is present in the subdirectories \AL-600, \AL-2000, \AL-2002 and \AL-2003, however being four different files, each one destined for a particular CPU.

WARNING:

The files contained in the subdirectory of a PLC should not be copied to another PLC, at the risk of losing the modules. Only the modules contained in the corresponding subdirectory of the CPU used should be loaded into the controller.

If there are doubts about the of CPU for which the module was programmed, use the file information command (command **File, Module Information** in MasterTool).

F-RELOG.000 - Function to Access the Real Time Clock



Introduction

The function **F- RELOG.000** carries out the access to the module real time clock AI-1420. This module implements a clock and calendar with great precision, allowing the development of applications programs that depend on very stable time bases. The module still continues to keep the time information with a system power failure, then it is supplied by batteries.

Programming

Operands

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

- **OPER1** - Specifies the number of parameters which are passed to the function in OPER3 this operand should compulsorily be a memory constant with value 2 (% KM + 00002).
- **OPER2** - Should be an operand of type memory constant with value 0 (%KM + 0000). Determines the number of parameters possible to be programmed in the editing window of OPER4. As this function does not need any parameter in OPER4, the value of OPER2 is 0.
- **OPER3** - Contains the parameters which are passed to the function, declared through a window visualised in MasterTool when the instruction is edited. The number of editable parameters is specified in OPER1, being set at 2 for this module:

%RXXXX - Address of bus where the module AL-142 is kept.

- **%MXXXX or %TMXXXX** - Specification of the operands for where there are read or found the clock values. If this parameter is specified as memory the values are read or found for the memory declared and the six subsequent ones. If it is specified as table, the values are placed or found starting from position 0 to 6. If the operands are not declared, the reading or finding of the time values is not achieved and the instruction outputs are disconnected. The tables can be used with more than 1 positions, with the function ignoring the surplus positions. The values are read or set in the operands in the following sequence:

Operand	Table Position	Contents	Format
%MXXXX	0	Seconds	000XX
%MXXXX+1	1	Minutes	000XX
%MXXXX+2	2	Hours	000XX
%MXXXX+3	3	Day of Month	000XX
%MXXXX+4	4	Month	000XX
%MXXXX+5	5	Year	000XX
%MXXXX+6	6	Day of the week	000XX

Table 4-2 Values read by the clock (F- RELOG.000)

The contents of these operands can be read or modified at any time, but they are update with the real hour of the module only when the instruction is executed. The 24 hour format is used in counting the time.

- **OPER4** - Not used.

Inputs and Outputs

Description of the inputs:

- **enable** - when this input is powered CHF the function is called, the parameters programmed in the instruction being analysed. If they are incorrect, all the outputs of the instruction are turned off. If they are correct, they output pulse 1 Hz is connected for one scan each second.
- **read clock** - when powered, the time values of the module are transferred to the memory operands or for the table declared as the second parameter in OPER3.



- **set clock** - when powered, the values contained in the memory or table operands declared are transferred to the module.

WARNING:

If the last two inputs are powered simultaneously, the setting of the module's values are carried out.

Description of the outputs:

- **pulse 1 Hz** - indicates if there is a change in the seconds count in the clock. The pulse last scan and be used to synchronise events in the applications program which use the Real Time Clock. This pulse can also be used to carry out the reading of the clock only when there is a change of seconds, economising on the execution time, since the reading of the insulated pulse is processed in more quickly.
- **lost time pulse** - this outputs is connected in the first scan after the powering of the PLC if the clock was fixed with the battery supply for a failure of the main supply. So that it is actioned, it is necessary that the instruction is enabled during the first scan of the controller.

WARNING:

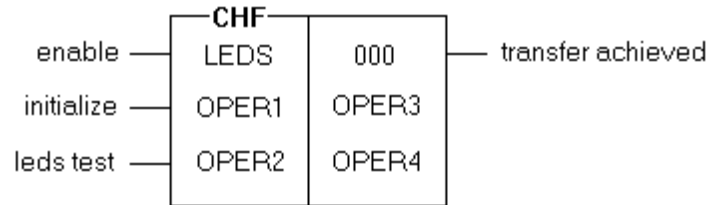
When the module is left without battery power and with the PLC disconnected, invalid values can be obtained in the hourly reading, without having counted the time and without actioning of the output pulse 1 Hz. So that the clock returns to correct functioning, it stops to carry out the setting of the time, programming a new hour.

Use

This function can be used in CPUs AL-600, AL-2000/MSP, AL-2002/MSP and AL-2003.



F-LEDS.001 - Function to Access the LEDs Module Panel



Introduction

The function **F-LEDS.001** carries out the interfacing of the applications program with the multiplexer module of the LEDs AL-1460, allowing the status of the system's digital points octets to be sent to this.

Programming

Operands

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

- **OPER1** - Specifies the number of parameters that are passed to the function in OPER3. This operand should compulsorily be a memory constant with value 4 (%KM+00004).
- **OPER2** - Should be an operand of type memory constant with value 0 (%KM+00000). It determines the of parameter possible to be programmed in the editing window of OPER4. As this function does not need any parameters in OPER4, the value of OPER2 is 0.
- **OPER3** - Contains the parameters which are passed to the function, declared through a window visualised in MasterTool when the CHF instruction is edited. The number of editable parameters is specified in OPER1, being set at 4 for this module:
 - **%RXXXX** - Address of bus where module AL-1460 is kept.
 - **%AXXXX** , **%EXXXX** or **%SXXXX** - Address of initial octet staring from the movements made to the LEDs panel.
 - **%KM+XXXXX** - Number of octets to be transferred.



- **%KM+XXXXX** - Number of initial octet of the LEDs panel where the movements are made to. Should be contained in the interval between 0 to 31.

WARNING:

The space for addressing the auxiliary relays (%AXXXX) is found next to the input and output relays (%EXXXX and found next to the input output relays (%EXXXX and %SXXXX). In this way, if the initial octet to be transferred is % S 0062 and the number of transfers is 5, the octets transferred are % S0062, % S0063, % A0000, % A0001 and % A0002.

- **OPER4** - Not used.

Inputs and Outputs

Description of inputs:

- **enable** - when this input is powered, the function carries out the transfer octets to the LEDs panel. The octets to be transferred are defined by the second and third parameters of OPER3, and the destination of the transfer through the fourth parameter.
- **initialize** - when this input is powered all the LEDs in the panel are put out. The input can be used during the execution of the applications program to put out all the LEDs without needing to carry out modifications in the values of the octets.

WARNING:

In order to visualise the values in the panel, the input initialized should be turned off.

- **leds test** - when this input powered, the instruction connects all the LEDs in the panel, allowing the verification of their functioning. This input can be powered during the execution of the applications program with the being the lighting of all the LEDs without needing to carry out modifications in the values of the octets.

WARNING:

If the initialize and leds test inputs are powered out the same time, the Initialization of the module is carried out, resulting in the putting out of the LEDs in the panel.

Description of outputs:

- **transfer achieved** - after all the octets are transferred or if the input leds test is activated, this output is powered, this output is powered. If the



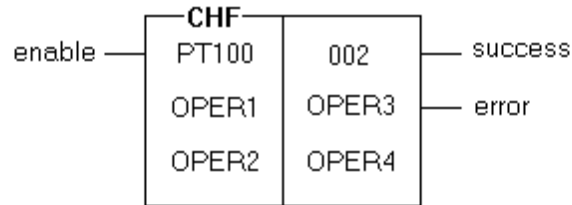
number of transfers and the initial address of the octets source or destination will be incompatible, the instruction is not carried out and the output is turned off.

Use

This function can be used in CPUs AL-600, AL-2000/MSP, AL - 2002/MSP, AL-2003, QK800 and QK2000/MSP.



F-PT100.002 - Function to read Module Pt 100



Introduction

The function **F-PT100.002** carries out the reading of the temperatures supplied by modules AL-1117 and QK1117, module interface with up to 4 sensors of type PT - 100. The values read can be linearised or not, the reading of 1 or 4 channels being possible, changing only the programming of the parameters used in its call.

Programming

Operands

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

- **OPER1** - Specifies the number of parameters that are passed to the function in OPER3. This operand should compulsorily be constant with value 4 (%KM + 00004).
- **OPER2** - Should be an operand of memory constant type with value 0 (%KM + 0000). Determines the number of parameters possible to be programmed in the editing window of OPER4. As this function does not need any parameter in OPER4, the value of OPER2 is 0.
- **OPER3** - Contains the parameters which are passed to the function, declared through a window visualised in MasterTool, when the CHF instruction is edited. The number of editable parameters is specified in OPER1, being set at 4 for this module:
 - **%RXXXX or %RXXXX.X** - Address of the bus where the module AL-1117 or QK1117 is kept. If it is specified with subdivision of point (%RXXXX.X), the reading is only carried out for the module channel corresponding to the point (points .0 to .3 of the corresponding operand to the channels 0 to 3 of the module



respectively). If the specification is not made with subdivision of point (%RXXXX) all 4 channels are read (0 to 3).

- **%KM+XXXXX** - Specification of type linearisation to be executed (c.f. table 6-2 for adjustment of modules AL-1117 and QK1117. The following types are valid:
 - **%KM+00000** - the function does not execute any linearisation showing the converter output with values between 0 and 4095 as a result.
 - **%KM+00001** - the function executes linearisation for the temperature band from -30.00 to +50.00 °C, represented in values from +0000 to +8000 (Value stored = $(T + 30) \times 100$).
 - **%KM+00002** - the function executes linearisation for the temperature band -30.00 to +370.00 °C, represented in values of +0000 to +4000 (Value stored = $(T + 30) \times 10$).
 - **%KM+00003** - the function executes linearisation for the temperature band from -30.00 to +770.00 °C represented in values from +0000 to 8000 (value stored = $(T + 30) \times 10$).
 - **%KM+00004** - the function executes linearisation for the temperature band from -30.00 to +50.00 °C represented in values from -3000 to +5000.
 - **%KM+00005** - the function executes linearisation for the temperature band from -30.00 to +370.00 °C, represented in values from -0300 to +3700.
 - **%KM+00006** - the function executes linearisation for the temperature band from -30.00 to +770.00 °C represented in values from -0300 to +7700.



Linearisation Constant	Measurement Band	Value Stored	PA5	PA6
%KM+00000	any	0000 to +4095	0/1	0/1
%KM+00001	-30 °C to +50 °C	0000 to +8000	0	0
%KM+00002	-30 °C to +370 °C	0000 to +4000	1	1
%KM+00003	-30 °C to +770 °C	0000 to +8000	2	1
%KM+00004	-30 °C to +50 °C	-3000 to +5000	0	0
%KM+00005	-30 °C to +370 °C	-0300 to +3700	1	1
%KM+00006	-30 °C to +770 °C	-0300 to +7700	2	1

Table 4-3 Linearisation and Configuration of the Modules AL - 1117 and QK 1117

WARNING:

If the sensor temperature exceeds the measurement band, the value 9999 will be stored in the corresponding channel

- **%MXXXX** - Specification of the operand where the values of the channels are stored after the reading and linearisation. If the first parameter is specified as %RXXXX.X (reading from a channel), only the memory position declared in parameter 3 is updated. If the first parameter is specified as %RXXXX (reading from 4 channels), the memory declared in parameter 3 is used and the next three the same.
- **%MXXXX** - Operand used for function for the internal control of its processing.

WARNING:

The control operand should not have its contents altered in any part of the applications program, under, penalty of threatening the correct execution of the function. Each CHF for this module should have a control operand, different from the rest. The control operand should not be retentive .

- **OPER4** - Not used.



Inputs and Outputs

Description of the inputs:

- **enable** - When this input is powered the function is called, the parameters programmed in the CHF instruction being analysed.

Description of outputs:

- **success** - is powered when the function is correctly executed.
- **error** - this output is always powered when one of the following errors occurs:
 - break in connection with the sensor Pt - 100
 - short circuit in the connection with the sensor Pt - 100
 - the module declared in the bus is not AL-1117 or QK1117
 - error in specification of operands or attempt to access operands not declared

In the first two errors, the value of the operand corresponding to the channel receive the value 9999.

WARNING:

The error output is implemented starting from version 1.10 of F-PT100.002.

Use

This function can be used in CPUs AL-600, AL-2002/MSP, AL-2002/MSP, AL-2003, QK800, QK801 and QK2000/MSP.

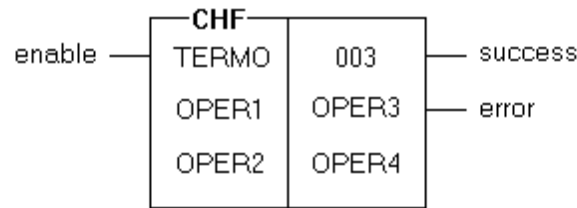
WARNING:

The updating time for each channel is 400 ms. This time is written up by its own function. In this way, the CHF instruction used for the call of module F should not be jumped, under, penalty of increasing the time.

The function cannot be for another channel before closing the conversion of the current channel.



F-TERM0.003 Function to Read Termopar Module



Introduction

The function **F-TERM0.003** carries out the temperature readings supplied by modules AL-1109 and QK1109. The values read can be linearised or not being possible to read from 1 or 4 channels, changing only the programming of the parameters used in its call.

Programming

Operands

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

- **OPER1** - Specifies the number of parameters which are passed to the function in OPER3. It is compulsory for this operand to be a memory constant with value 5 (%KM+00005).
- **OPER2** - Should be an operand of type memory constant with the value 0 (%KM+00000). It determines the number of parameters possible to be programmed in the editing window of OPER4. As this function does not need any parameter in OPER4, the value of OPER2 is 0.
- **OPER3** - Contains the parameters which are passed to the function, declared through a window visualised in MasterTool when the CHF is edited. The number of editable parameters is specified in OPER1, being set at 5 for this module:



%RXXXX or %RXXXX.X - Address of bus where the module AL-1109 or QK1109 is kept. If it is specified with subdivision of point (%RXXXX.X), the reading is only carried out for the channel of the module corresponding to the point (points .0 to .3 of the operand, corresponding to the channels 0 to 3 of the module, respectively). If the specification is not made with subdivision of point (%RXXXX), all 4 channels are read (0 to 3).

- **%KM+XXXX** - Specification of type of termopar:
 - **%KM+00000** - value supplied by module without linearisation
 - **%KM+00001** - termopar type J
 - **%KM+00002** - termopar type R
 - **%KM+00003** - termopar type S
 - **%KM+00004** - termopar type K
 - **%KM+00005** - termopar type B
- **%KM+XXXXX** - Defines the value normalized, represented (0000 to 1000) or in degrees (c.f. table 4-3):
 - **%KM+00000** - result in degrees Celsius
 - **%KM+00001** - result normalized



Type of Termopar	Result in degrees Celsius	Normalized Result
J	0000 to 1000	0000 to 1000
R	0000 to 1500	0000 to 1000
S	0000 to 1500	0000 to 1000
K	0000 to 1250	0000 to 1000
B	0000 to 1800	0000 to 1000

Table 4-4 Values Read from Modules AL-1109 and QK1109

WARNING:

If the temperature of the sensor exceeds the measurement band, the value 9999 will be stored in the corresponding channel.

- **%MXXXX** - Specification of the operand where the values of the channel are stored after the reading and Normalization . If the first parameter is specified as %RXXXX.X (reading of a channel) only the memory position declared in as %RXXXX (reading of 4 channels), the memory declared in parameter 3 and the following 3 are used.
- **%MXXXX** - Operand used by function the internal control of its processing.

WARNING:

The control operand should not have its contents altered in any part of the applications program, under penalty of endangering the correct execution of the function. Each CHF for this module F should have an exclusive control operand, different from the rest.

- **OPER4** - Not used.

Input and Outputs

Description of inputs:

- **enable** - when this input is powered the function is called, the parameters programmed in instruction CHF being analysed.



Description of the outputs:

- **success** - is powered when the function is correctly executed.
- **error** - this output is always powered when one of the following errors occurs:
 - break of the connection with the termopar sensor
 - the module declared in the bus is not AL-1109 or QK1109
 - error in the specification of the operands or attempt to access the operands not declared

In the first error, the value of the operand corresponding to the channel receives the value 9999.

WARNING:

The error output is implemented starting from version 1.10 from **F-TERM0.003**.

Use

This function can be used in the CPUs AL-600, AL-2000/MSP, AL-2002/MSP, AL-2003, QK800, QK801 and QK2000/MSP.

WARNING:

Starting from version 1.10 of module **F-TERM0.003**, the error output is connected and the value 9999 is placed in the channels of the first scan of the program execution. The updating time for a channel was reduced from 400ms to 100ms starting from this version.

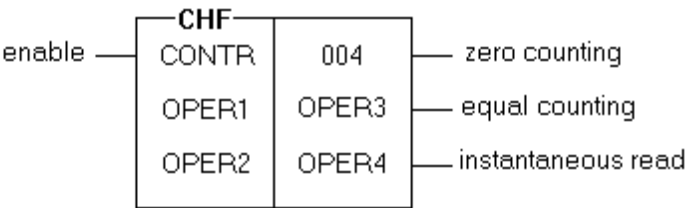
WARNING:

The updating time for each channel is 100ms. This time is written up by its own function. In this way the CHF instruction used for the call of module F should not be jumped, under penalty of increasing the conversion time.

The function cannot be called for another channel before closing the conversion of the current channel.



F-CONTR.004 - Function to Access the Rapid Counter Module



Introduction

the function **F-CONTR.004** carries out the interfacing of the applications program with the rapid counter module AL-1440 and rapid counter with interface for optical transducers AL-1450 and QK1450.

Programming

Operands

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

- **OPER1** - Specifies the number of parameters which are passed to the function in OPER3. It is compulsory for this operand to be a memory constant with value 4 (%KM+00004).
- **OPER2** - Should be an operand of type memory constant with the value 0 (%DM+00000). It determines the number of parameters possible to be programmed in the editing window of OPER4. As this function does not need any parameter in OPER4, the value of the OPER2 is 0.
- **OPER3** - Contains the parameters which are passed to the function, declared through a window visualised in MasterTool when the instruction CHF is edited. The number of editable parameters is specified in OPER1 being set at 4 for this module:
 - **%RXXXX** - Address of bus where the module AL-1440, AL-1450 or QK1450 is kept.

- **%DXXXX** - Operand with the value to be read or written to the counting register of the module.
- **%DXXXX** - Operand with the value to be read or written to the comparison register of the module.
- **%AXXXX** - Octet which contains a group of instructions for the module, described as follows:
 - **%AXXXX.0** - inhibit counting
 - **%AXXXX.1** - zero count register
 - **%AXXXX.2** - enable output relays (comparison for hardware)
 - **%AXXXX.3** - enable reference input
 - **%AXXXX.4** - execute reading of count
 - **%AXXXX.5** - execute writing of count
 - **%AXXXX.6** - execute writing of comparer
 - **%AXXXX.7** - not used
- **OPER4** - not used.

Inputs and Outputs

Descriptions of the inputs:

- **enable** - when this input is powered the function is called, the parameters programmed in the CHF instruction being analysed. If they are incorrect, all the outputs of the instruction are turned off. If they are correct, the instructions contained in the auxiliary octet %AXXXX are executed, carrying out the operations of reading and writing according to the specification. When the input enable is turned off, the previous instructions sent to the module are kept, not executing any reading or writing operation in it.

Description of the outputs:

- **zero counting** - is activated when the count register reaches the value zero. Its activation is carried out at least during one scan of the applications program, if the count passes for zero and does not remain with this value.
- **equal counting** - is activated when the count value reaches the value equal to the comparer register. Like this as the previous output, it is activated for at least one scan of the applications program.



- **instantaneous read** - is activated only during the scan following the request to read, if this is not disabled for the modules point of adjustment. In this case, the next reading of the count supplies the value of this at the moment when the instantaneous read was asked for.

WARNING:

If the reading and writing in the counting module are requested simultaneously, only the writing is executed. Independently of this situation, the writing in the register is always executed when requested, if the instruction is enabled. It should be taken into account that each reading or writing operation is relatively slow. In this way, unnecessary operations should be avoided so as not to involve the scan time of the applications program.

WARNING:

After the PLC is connected, the value of the count and comparison registers is random, having be initialized appropriately.

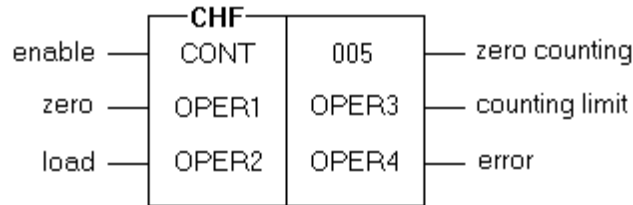
Further information can be obtained in the Technical Characteristics of the modules AL-1440, AL-1450 and QK1450.

Use

This function can be used in the PLCs AL-600, AL-2000/MSP, AL-2002/MSP, AL-2003, QK800, QK801 and QK2000/MSP.



F-CONT.005 - Function to Access the Fast Counting Inputs



Introduction

The function **F-CONT.005** carries out the interfacing of the applications program with the rapid count inputs in the CPUs AL-600, QK600, PL102 and PL103 and PL104.

The CPUs AL-600, QK600, PL102, PL103 and PL104 have two fast counting inputs in the front panel, allowing the counting of pulses with high frequency (up to 10 KHz) when an inadequate count is made through the conventional points of input.

Programming

Operands

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

- **OPER1** - Specifies the number of parameter which are passed to the function in OPER3. It is compulsory for this operand to be a memory constant with value 3 (%KM+00003).
- **OPER2** - It should be an operand of type memory constant with value 0 (%KM+00000). It determines the number of parameters possible to be programmed in the editing window of OPER4. As this function does not need any parameter in OPER4, the value of OPER2 is 0.
- **OPER3** - Contains the parameters which are passed to the function, declared through a window visualised in MasterTool when the CHF is edited. The number, of editable parameters is specified in OPER1, being set at 3 for this module:
 - **%KM+XXXXX** - Number of input counted (0 or 1).



- **%DXXXX** - Operand which stores the count value.
- **%MXXXX** - Operand used for function for the internal control of its processing.

WARNING:

The control operand should not have its contents altered in any part of the applications program, under penalty of endangering the correct execution of the function. Each CHF for this module F should have exclusive control operand, different from the rest.

- **OPER4** - Not used.

Inputs and Outputs

Description of the inputs

- **enable** - when this input is powered the function is called, the programmed parameters being analysed in the CHF instruction. If they are incorrect, the error output is powered.
- **zero** - causes the zeroing of the count value, when enabled.
- **load** - when activated, ensures that the value stored in the operand will be the new count value.

Description of the outputs:

- **zero counting** - is powered when the operand value of the count has the value zero.
- **count limit** - is powered when the operand value of the count has the value + 9999999.
- **error** - is powered if an error occurs in the specification of the operands or attempt to access the operands not declared.

When the programming PLC is to be executed, the ZERO input of the functions **F-CONT.005** should be actioned for a scan, in a way that allows the function to be referenced, seeing that its operands of control and count are zeroed in the change of status.

Use

This function can be only be used in CPUs AL-600, QK600, PL102, PL103 and PL104.



Description of Functioning

Each counter carries out an incremental count, from 0 to + 9999999 pulses, stored, in one operand %D. When the count reaches the value limit, the operand is not increased, connected to the output limit of the count in the CHF instruction.

The value in the count operand can be initialized with the actioning of the zero input of the instruction. To begin the count with a value different from zero, stop to move the value required for the operand %D and action the loading of the instruction.

The function should be called periodically, in the normal scan cycle or in the module executed for time interruption E018. The maximum frequency of the count depends on the period of the call, being shown in table 4-4.

Maximum Frequence of Count	Period of Function Call
2,5 KHz	100 ms
3,4 KHz	75 ms
5,0 KHz	50 ms
10,4 KHz	25 ms

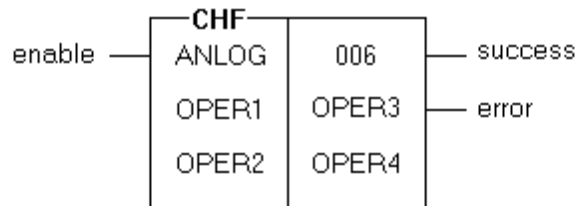
Table 4-5 Frequency of Counting AL-600

The function can be called several times in programs with a long scan cycle time, allowing the count to be more frequent.

For example, if the cycle time of the applications programs is 85ms and it is necessary to count pulses up to 7KHz, the function call should be repeated 4 times throughout the program or include it in the module E018 with an execution frequency of 25ms.



F-ANLOG.006 - Function to Convert A/D or A/D Integrated



Introduction

The function **F-ANLOG.006** carries out the conversion A/D (analogue/digital) or D/A (digital/analogue) from the available integrated analog channels in the CPUs AL-600, QK600, PL102, PL103 and PL104 (DAC 1 and DAC 2).

Using two CHF instructions, it is possible to carry out the A/D conversion in one of the channels and D/A in another or the same type of conversion in both.

Programming

Operands

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

- **OPER1** - Specifies the number of parameters which are passed to the function in OPER3. It is compulsory for this operand to be a memory constant with value 3 (%KM+00003).
- **OPER2** - It should be an operand of type memory constant with value 0 (%KM+00000). It determines the number of parameters possible to be programmed in the editing window of OPER4. As this function does not need any parameter in OPER4, the value of OPER2 is 0.
- **OPER3** - Contains the parameters which are passed to the function, declared through a windows visualised in MasterTool when the instruction is edited. The number, being set at 3 for this module:



- **%KM+XXXXX** - Specification of the channel to be converted. Should use %KM+00000 for DAC 1 and %KM1 for DAC 2.
- **%KM+XXXXX** - Type of conversion to be carried out in the channel defined by previous parameter. Should use %KM+00000 to convert A/D and %KM+00001 to convert D/A.
- **%MXXXX** - Specification of operand where the value to be written into the converter if there is a D/A conversion or value read if there is an A/D conversion.
- **OPER4** - Not used.

Inputs and Outputs

Description of the inputs:

- **enable** - when this input is powered the function is called, the parameters programmed in instruction CHF being analysed.

Description of the output:

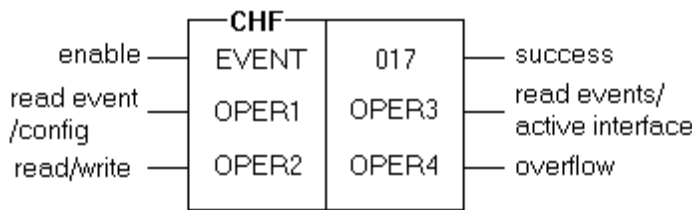
- **success** - is powered when the function was correctly executed.
- **error** - is powered if an error occurs in the specification of the operands or attempt to access the operands not declared.

Use

This function can only be used in CPUs AL-600, QK600, PL102, PL103 and PL104.



F-EVENT.017 - Function to Access the Module Register of Events



Introduction

The function **F-EVENT.017** carries out the access to interfaces AL-3130 and AL-3132, which have 32 digital inputs with register of events.

The interfaces AL-3130 and AL-3132 are equivalent to the software. In this description of the **F-EVENT** all the reference to AL-3130 is valid for AL-3132, except when there is explicit reference.

The interface AL-3130 registers the status variations in their inputs with an accuracy of 1 millisecond, allowing the monitoring of digital events in real time. The interface stores the events in a local memory, in independent form, while the CPU processes the applications program. The function **F-EVENT** transfers these events from the interface memory to a table, defined in the instruction of the CHF call.

Each AL-3130 present in the bus of the PLC contains an internal clock synchronised with the clock of AL-2002, eliminating differences in the times of events registered in different interfaces. This synchronisation occurs in transparent form in the applications program.

If different PLCs are interconnected through the synchronism network and by ALNET II, the clocks in all the AL-3130 of the system keep the hour of the clock in the PLC managing the synchronism.

The interface always registers changes in the hour of its clock, occurring due to final settings in the CPUs clock through the function **F-RELG.048** or **F-SINC.049** for the correct evaluation of the data in the registered events.

For further information regarding the synchronisation of the controllers, consult the User's Manual AL-2002 and the ALTUS Networks Manual.

Programming

Before programming the function call, the interface in the position of the bus in which it is kept should be declared, through the editing window of module C. It can declare it with two different codes:

- AL-3130/AL-3132 - events only
- AL-3131/AL-3133 - events and inputs

If declared as AL-3130 (or AL-3132), the interface is not accessed by the scan of the PLC's I/O, the operands %E not being reserved for their input octets. In this way, the interface only registers the transitions of its inputs and the applications program cannot process the values of the points as in a digital input module.

If it is declared as AL-3131 (or AL-3133), operands %E are associated to points of input, and the PLC accesses them in its scan of I/O. Like this, after the register of events, the values of module input points are used for the applications program.

WARNING:

The updating of the module input points for operands %E is only possible starting from version 1.10 of the execute software in AL-3130. In AL-3132 it is always possible.

The CHF instruction has two types of call for function **F-EVENT**:

- configuration call
- reading of events call

The configuration call allows the applications program to specify or read the interfaces parameters of functioning.

The call to read events transfers the interfaces events register to the table specified.

Configuration Call

The configuration call is used to program or read parameters of the modules functioning.

The parameters of configuration are:

- the disabling masks of the events register
- the debounce time
- overflow options



The disabling masks allow the deactivation of the events register for points used as normal input, avoiding the generation of unrequired events, which forces the program to identify and discard them. There are 4 masks, each one specifying an interface octet, each bit corresponding to an input point. The bit connected holds back the input corresponding to the events.

The debounce time specifies, in milliseconds, how long the interface should ignore the variations of an input after its change of status. The debounce is used to eliminate noise in the signals which act on the interface. If one input is actuated for a relay contact, for example, it avoids the vibration causing different events in the interface, instead of a single one. The debounce works as a filter, ignoring the contact noise to open or close. The instant of the opening or closing is registered, then the variations of the input status are ignored during the debounce time. The time can be specified between 0 and 255 milliseconds, being valid for all the interface inputs.

The debounce time always affects the digital inputs, if the interface is declared as AL-3131/AL-3133.

The overflow options in the functioning mode when new events occur after the maximum number of events storable in the interface is surpassed (interface memory totally full). Two options can be selected:

- option 0 - keep old events
- option 1 - keep new events

Option 0, automatically configured with the powering of AL-3130, keeps the older events in the interface. The new events which occur, after the memory is completely full, are ignored.

In option 1, the new events are saved, the older ones being discarded for the storing of the new ones.

WARNING:

The memory of storing the events of interfaces AL-3130 are emptied with the reading of events by the applications program, through the function **F-EVENT.017**. The program should carry out periodic readings, to prevent the interface memory becoming totally full, with possible loss of the events register.



Configuration

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

- **OPER1** - Specifies the number of parameters that are passed to the function in OPER3. It is compulsory for this operand to be a memory constant with value 2 (%KM+00002).
- **OPER2** - Should be an operand of type memory constant with value 0 (%KM+0000). Determines the number of parameters possible to be programmed in the editing window of OPER4. As this function does not need any parameter in OPER4, the value of OPER2 is 0.
- **OPER3** - Contains the parameters which are passed to the function, declared through a window visualised in MasterTool when the CHF instruction is edited. The number of editable parameters is specified in OPER1, being set at 2 for this call.
 - **%RXXXX** - Address of bus where the interface AL-3130 or AL-3132 is kept.
 - **%TMXXXX** - Specification of the memory table where the interface configuration parameters are placed. The values are read or set position 0 to 5 in the table. If the table is not declared or has less than 6 position, the reading or set of the configuration values is not carried out and all the output are disconnected. It is possible to use tables with more 6 positions, as the function ignores the surplus position. The value are read or set in the operands in the following sequence:

Table Position	Contents	Format
0	Disabling mask octet 0	00XXX
1	Disabling mask octet 1	00XXX
2	Disabling mask octet 2	00XXX
3	Disabling mask octet 3	00XXX
4	Debounce	00XXX
5	Overflow	0000X

Table 4-6 Interface Configuration Parameters

Format

Disabling masks of events register:



Mask bit: 7 6 5 4 3 2 1 0

Octet point: 7 6 5 4 3 2 1 0

Bit value 0 - events register enabled
1 - events register disabled

Debounce:

Value: 0 to 225 milliseconds Overflow

Overflow:

Value: 0 - keep old events
1 - keep new events

- **OPER4** - Not used.

Reading of events

In the call like the reading of events, the function transfer the data read from the interface AL-3130 to a memory table.

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

- **OPER1** - Specifies the number of parameters which are passed to the function in OPER3. Its is compulsory for this operand to be a memory constant with value 2 (%KM + 00002).
- **OPER2** - Should be an operand of type memory constant with value 0 (%KM + 0000). It determines the number of parameters possible to be programmed in the editing window of OPER4. As this function does not need any parameter in OPER4, the value of OPER2 is 0.
- **OPER3** - Contains the parameters which are passed to the function, declared through a window visualised in MasterTool when the CHF instruction is edited. The number of editable parameters is specified in OPER1, being set at 2 for this call:



- **%RXXXX** - Address of bus where the interface AL-3130 or AL-3132 is kept.
- **%TMXXXX** - Memory table where the registers of the events read from AL-3130 are placed. The registers of events are placed starting from the position defined for a Pointer (c.f. layout of table %TMXXXX).

If the table is not declared, the reading is not carried out and the 3 outputs of the instruction are disconnected. The table should have a minimum of 12 positions, but the recommended value to optimise the data transmission of data in the network is of 64 positions.

- **OPER4** - Not used.

Format of table and events

The layout for the registers of events follows the format:

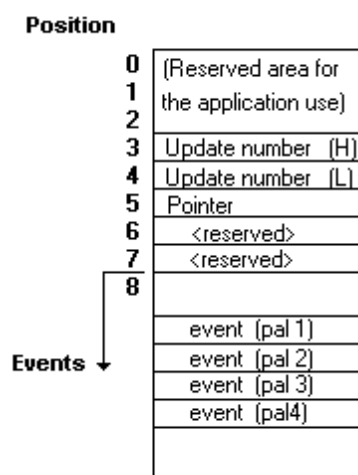


Figure 4-1 Layout for the Events Registers

- At the first position the application should inform CP identification to supervisory system: sub-network and node:



SSSSSS - sub network (1 to 64)



NNNN - node (1 to 31)

- The second and third positions are reserved for use by the application/supervisory.
- Number updating: sequential number which identifies the updating of the table. It is a 32 bit number which is increased by F-EVENT on each updating of the table. It is used for supervisors or application to validate the data read.
- Pointer: table position which points to the next available position. Its minimum value is 8, and its maximum value is the size of the table (table 100% full). It is updated by F-EVENT at each new event placed in the table. It should be zeroed for application or similar supervisor to free the table for new events.
- Events Date: date (day and month) of events in table. This position is managed by F-EVENT and must be re-initialized with zero. Date has the following format:

	12				8				4				0			
Events Date	0	0	0	0	MM	MM	0	0	0	0	D	DD	DD	DD	DD	DD

MMMM - events month (1 a 12)
DDDDDD - eventos day (1 a 31)

ATTENTION:
Date is managed by F-EVENT according to UCP events actual date, stored at AL-313X modules.

- Table size: although the function F-EVENT can work with tables from 12 to 255 position, the use of tables of size 64 bytes is recommended, appropriate for the transmission of the table by ALNET II.
- Events: the events occupy four adjacent position in the table.

The events have the format:

	12				8				4				0			
Position n	0	0	1	0	1	0	1	1	0	b	b	b	e	e	e	e
	0															
Position n+1	0	0	0	0	0	0	0	0	0	0	r	r	r	r	r	r
	0															
Position n+2					8								0			
	A	D	0	h	h	h	h	h	h	0	0	mm	mm	mm	mm	mm

	10						0							
Position n+3	s	s	s	s	s	s	1	1	1	1	1	1	1	1

FIELDS:

bbb - octet bit (0 - 1)

eeee - event status

0 - status 0 of the input

1 - status 1 of the input

14 - change in the time: previous hour

15 - change in the time: current hour

rrrrrr - address of the AL-3130 (%RXXXX) + N° octet

A - date delay (only eeee = 15)

D - date change (only eeee = 15)

hhhhh - hour of the event (0 to 23)

mmmmmm - minutes of the event (0 to 59)

sssss - seconds of the event (0 to 59)

lllllllll - milliseconds (0 to 999)



Observation: the values 14 and 15 of the status do not signify a transition from some input point, but yes that there is a setting of the clock from 3130. In this case, the value of the fields T, tttt and bbbb do not have significance (0).

Ex. 1:
An event in bit 5, status “1”, in the third octet of AL-3130 of position R024, the 14 hours, 30 minutes, 10 seconds and 456 milliseconds, produces the register:

Pal 0	0 0 1 0 1 0 1 1 0 1 0 1 0 0 0 1
Pal 1	0 0 0 0 0 0 0 0 0 0 0 1 1 0 1 0
Pal 2	0 0 0 0 1 1 1 0 0 0 0 1 1 1 1 0
Pal 3	0 0 1 0 1 0 0 1 1 1 0 0 1 0 0 0

Ex. 2:
A setting of the hour gives two events registers: the first with the previous hour and the second with the current hour. An event of “current hour” in AL-3130 of the address R016, 08:32:25, milliseconds 000, produces the register:

Pal 0	0 0 1 0 1 0 1 1 0 0 0 0 1 1 1 1
Pal 1	0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0
Pal 2	0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0
Pal 3	0 1 1 0 0 1 0 0 0 0 0 0 0 0 0 0

Inputs and Outputs

Description of the Inputs:

- **enable** - when this input is powered the function is called, the parameters programmed in the CHF instruction being analysed. If they are incorrect, all the instructions outputs are turned off. If they are correct, the output succeeded returns connected.

WARNING:

The function carries out Initialization procedures and the setting of the time of the time of the interface automatically. Due to this, each scan cycle of the applications program should be called, keeping the input enable of the CHP instruction always powered.

- **read event/config** - when powered, the function carries out the interface configuration. If turned off, it carries out the reading of the events.

IMPORTANT:

AL-3130 only generate events if receive the CPU synchronism signal. To set CPU to send synchronism you must declare CPU “generate” or “receive” synchronism (see MasterTool C module, networks frame, click on synchronism button).

- **read/write**
 - in configuration mode - when powered, the values contained in the configuration table are transferred to the interface. When turned off the configuration is read by the interface to a table.
 - in reading of events mode powered, the events registers are transferred to a specified table, if the events exist and if there are free positions. When turned off, the events are not transferred. This input can be used to hold up the reading of events while it processes the table thus, allowing the function to be called in each cycle.

Description of the outputs:

- **success** - indicates that the call parameters are correct and that the function was correctly executed.
 - in reading of events mode: indicates also that the interface is active.
- **read events /active interface**
 - in configuration mode - if powered, indicates that the interface is active.



- in reading of events mode - if powered, indicates that at least one event register was read to table in the current scan.
- **overflow** - if connected, indicates that the internal memory of the AL-3130 was completely full. Subsequent reading operations, carried out successfully, disconnect this output. The overflow output is actioned in reading of events mode.

Diagnostics

The status interface active, referred to above, is the consequence of several factors which should be verified, in case there are difficulties in programming the **F-EVENT**:

- hot swap key switched off (position RUN)
- the interface is not in error (led ERR put out)
- AL-3130 is receiving synchronism from the CPU (CPU correctly configured and connected)
- AL-3130 is declared in the bus of the PLC
- function **F-EVENT** is loaded in the CPU

If the interface is not active, the outputs **success** (events call) and **interface active** (configuration call) do not switch on. If the F_EVENT has not been loaded in the CPU, the output **succeeded** of the configuration call also is not switched on.

In case of CPU synchronism fault then ERR AL-313X module leds make blink each 2 seconds, repeatedly.

Use

This function can only be used in the CPUs AL-2002/MSP and AL-2003.

Example of Application

Declaration in the bus:

The interface can be declared in two ways: As AL-3130/AL-3132 or as AL-3131/AL-3133. If it is declared as **even**, the interface can only be used for dealing with events. If it is declared as **odd**, the interface can also be used as an input interface, without losing the events function.



Position	Model	Inputs	Output	Address
00	AL-3131	%E0000-%E0003		%R0000
01	AL-3132			%R0008

Table 4-7 Declaration of the Module AL-3130 in the Bus

Initialization:

Configuration and events tables must be initialized by application. In events table you must to do node and subnet initializing like PLC declaration and change remainder fields with zeroes.

Use in the diagram of relays:

In the above example, the interface is configured in the logic 000. It is interesting that the configuration is always made for each scan, afterwards the interface can be changed with the PLC connected (hot change).

The format of the %TM0000 in the configuration call follows that previously described.

The relay %A0000.0 always switches on when the function is called and the parameters are correct.

The relay %A0000.1 switches on when the interface is active %A0000.1 disconnect signifies inactive interface or in error.

In the logic 001 the interface configuration is switched on. The table position %TM0001 have significance identical to %TM0000.

The relay %A0001.0 always switches on when the function is called and the parameters are correct.

The relay %A0001.1 switches on when the interface is active %A0001.1 disconnected signifies interface inactive or in error.



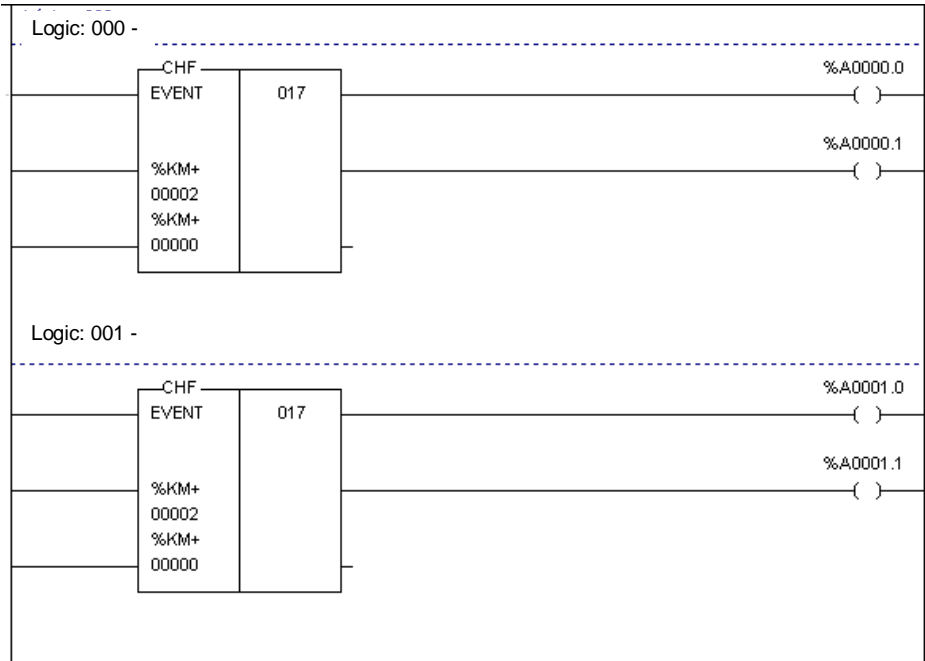


Figure 4-2 Example 1 of Use of Module F-EVENT.017

Logic 000 - Module F-EVENT.017:

Input	Output
%R0000	
%TM0000	

Logic 001 - Module F-EVENT.017:

Input	Output
%R0000	
%TM0001	

The logic 2 shows the reading of the interface points. The reading is possible since the interface may be declared as AL-3131 or AL-3133 in the bus. The reading of the points is independent of the configuration masks, but they are affected by the value of the “debounce”.



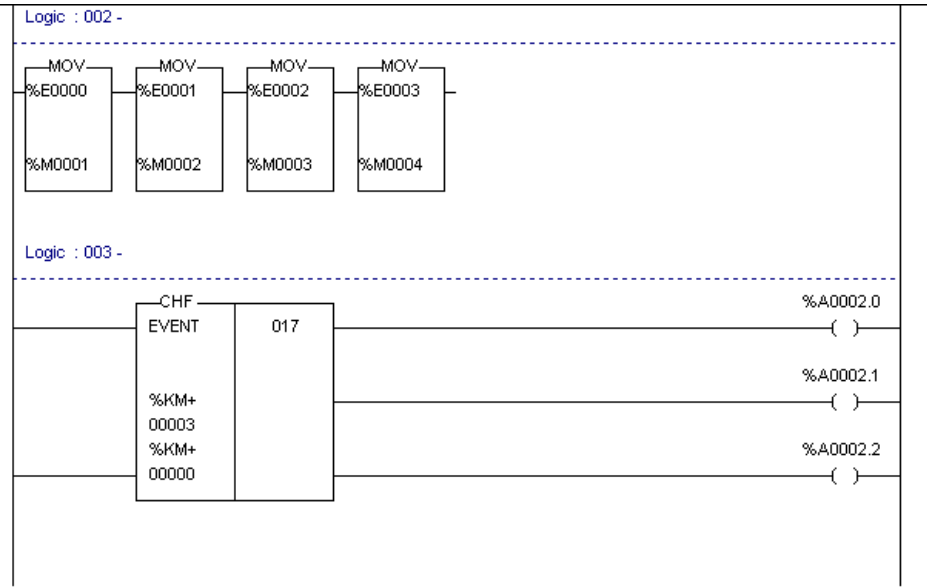


Figure 4-3 Example 2 of Use of Module F-EVENT.017

Logic 003 - Module F-EVENT.017:

Input	Output
%R0000	
%TM0002	

In the logic 3 of the example the reading of the events registered by AL-3130 is shown. The events are transferred by the AL-3130 to the table %TM0002. The position 5 of %TM0002 always points to the next available position in the table.

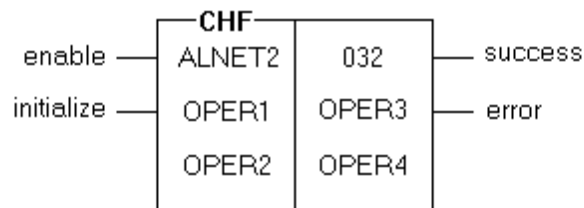
The relay %A0002.0 always switches on when the function is called, the parameters are correct and the interface is active.

The relay %A0002.1 switches on when an updating has been carried out in the events table in the scan in question.

The relay %A0002.2 switches on when overflow occurs in the internal memory of the interface. The overflow disconnects as the events are withdrawn from the interface to the table %TM0002.



F-ALNET2.032 - Function Read from Statistics of ALNET II



Introduction

The function **F-ALNET2.032** allows the reading and the writing of the values of the configuration parameters and of the statistics of the functioning of the PLC in ALNET II by the applications program.

Programming

Operands

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

- **OPER1** - Specifies the number of parameters which are passed to the function in OPER3. This It is compulsory for this operand to be a memory constant with value 1 (%KM+00001).
- **OPER2** - Should be an operand of type memory constant with value 0 (%KM+00000). Determines the number of parameters possible to be programmed in the editing window of OPER4. As this function does not need any parameter in OPER4, the value of OPER is 0.
- **OPER3** - Contains the parameters which are passed to the function, declared through a window visualised in MasterTool when the CHF instruction is edited. The number of editable parameters is specified in OPER1, being set at 1 for this module:



%MXXXX or %TMXXXX - Memory or table operand which receives the values of the statistics and parameters. If an operand %M is used it should be defined in the minimum 41 operands starting from the one declared (including this one too) so that the function is executed correctly. If it is an operand table, this should have at least 41 positions.

- **OPER4** - Not used.

Inputs and Outputs

Description of inputs:

- **enable** - when input is powered the function is called, the parameters programmed in the CHF instruction. If they are incorrect, all the outputs in the instruction are turned off. If they are correct, the values are copied, the success output being actioned.
- **initialize** - when powered, zeroes the values of the statistics.
- **write** - if powered, transfers the values of the last 3 table parameters to the PLC's variables, forcing the status of the physical connections.

Description of the outputs:

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

WARNING:

The input write is implemented starting from version 1.20 of **F-ALNET2.032**.
The error output is implemented starting from version 1.10.



Description of the Values of the Statistics and Parameter

Operand	Table Position	Contents
Statistics of transmissions		
%MXXXX	0	Number of error free transmissions
%MXXXX+1	1	Number of transmissions with collision error
%MXXXX+2	2	Number of transmissions with underrun error
%MXXXX+3	3	Number of transmissions without ACK reception of hardware
%MXXXX+4	4	Number of transmission retries
%MXXXX+5	5	Number of service time-outs
%MXXXX+6	6	Number of failures of transmission buffers
%MXXXX+7	7	Not used
%MXXXX+8	8	Not used
Statistics of receptions		
%MXXXX+9	9	Number of error free receptions
%MXXXX+10	10	Number of receptions with collision error
%MXXXX+11	11	Number of receptions with overrun error
%MXXXX+12	12	Number of receptions with CRC error
%MXXXX+13	13	Number of receptions with alignment error
%MXXXX+14	14	Number of receptions of packets with size error
%MXXXX+15	15	Number of packet time-outs
%MXXXX+16	16	Number of failures of reception buffers
%MXXXX+17	17	Not used



Operand	Table Position	Contents
Configuration Parameters		
%MXXXX+18	18	Communication speed (c.f. table to follow)
%MXXXX+19	19	Station address
%MXXXX+20	20	Local sub-network address
%MXXXX+21	21	Multicast groups
%MXXXX+22	22	Address of gateway 1
%MXXXX+23	23	Address of gateway 2
%MXXXX+24	24	"Time-out" in bus (tenths of seconds)
%MXXXX+25	25	"Time-out" in bus (tenths of seconds)
%MXXXX+26	26	"Time-out" of packet (tenths of seconds)
%MXXXX+27	27	Number held back from transmission
%MXXXX+28	28	Type of physical connection (0 - electric, 1 - optical)
%MXXXX+29	29	Redundancy of physical connection (0 - without, 1 - with)
%MXXXX+30	30	Period for testing redundancy (seconds)
%MXXXX+31	31	Time for communication from physical connection (seconds)
%MXXXX+32	32	Not used
%MXXXX+33	33	Not used
%MXXXX+34	34	Physical connection selected (1 or 2)
%MXXXX+35	35	Connection status 1 (0 - normal, 1 - failure)
%MXXXX+36	36	Connection status 2 (0 - normal, 1 - failure)
%MXXXX+37	37	No used
Configuration parameter possible to be written		
%MXXXX+38	38	Physical connection forced (0 - not forced, 1 or 2)
%MXXXX+39	39	Status forced by connection 1 (0 - normal, 1 - failure)
%MXXXX+40	40	Status forced by connection 2 (0 - normal, 1 - failure)

Table 4-8 Description of Values of Statistics and Parameters



Baud Rate	0	-	2 Mbaud
	1	-	1 Mbaud
	2	-	500 Kbaud
	3	-	250 Kbaud
	4	-	125 Kbaud
	5	-	64 Kbaud
	6	-	25 Kbaud

Further information about the significance of the statistics and parameters can be found in the ALNET II User’s Manual.

Use

This function can be used in CPUs AL-2000/MSP, AL-2002/MSP, AL-2003 and QK2000/MSP.

Example of Application

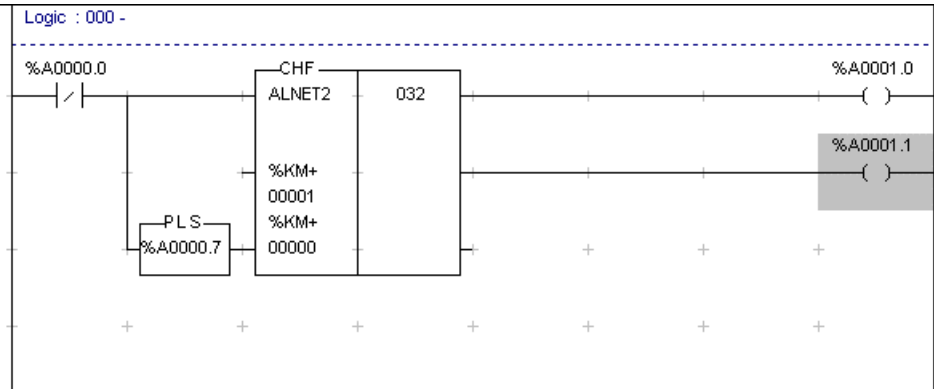
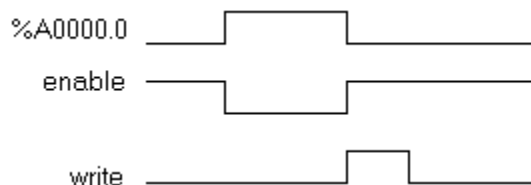


Figure 4-4 Example of Use of Module Function F-ALNET.032

Logic 000 - Module F-ALNET2.032:**Input****Output**

%TM0000

**Figure 4-5 Diagram of Times of Example of F-ALNET.032**

%TM0000 may be a table with 41 positions.

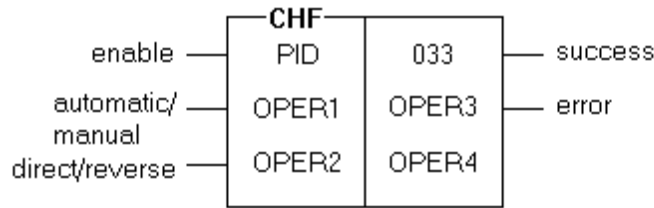
So that it may be possible to write the 3 single status parameters of the network ALNET II which can be changed by the applications program (c.f. parameters 38, 39, 40), first the enabling signal of the function should be withdrawn.

Following this, the required values are placed in the positions corresponding to the parameters in %TM0000 (positions 38, 39 and 40). At this moment, the enabling signal is activated again, when a writing pulse is generated by the scan.

The writing pulse may lead to the alteration of the required parameters. Independently of the writing pulse, while the enabling signal is active, the %TM0000 is being updated with the group of statistics and ALNET II parameters.



F-PID.033 - PID Control Function



Introduction

The function **F-PID.033** implements the proportional control algorithm, integral and derivative. Starting from a measured variable (MV) and from the required set point (SP) the function calculates the Controlled variable (CV) for the system controlled. This value is calculated periodically, taking into consideration the proportional, integral and derivative factors programmed. The function's blocks diagram is shown in figure 4-6.

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

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



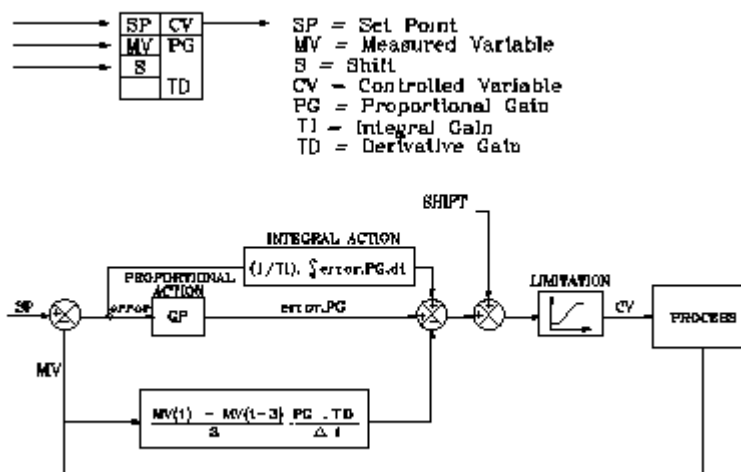


Figure 4-6 Diagram in Blocks of the Function PID

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

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

Programming

Operands

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



- **OPER1** - Specifies the number of parameters passed to the function in OPER3. It is compulsory for this operand to be a memory constant with value 5 (%KM+00005).
- **OPER2** - Specifies the number of parameters passed to the function in OPER4. It is compulsory for this operand to be a memory constant with value 0 (%KM+00000).
- **OPER3** - Contains the parameters passed to the function declared through a window visualised in MasterTool when the CHF instruction is edited. The number of editable parameters is specified in OPER1, being set at 5 for this module:
 - **%TMXXXX** - Table which contains the parameters used by the control algorithm. Should count 16 positions.
 - **%MXXXX** - Memory which contains the measure value of the process, normally obtained through an A/D instruction.
 - **%MXXXX** - Contains the set point, which is the value required for the measured variable. Its value can be modified according to the control politic required.
 - **%MXXXX** - Memory which contains the controlled variable in the process, generally actioned by a D/A instruction.
 - **%AXXXX** - Auxiliary octet which contains control points of the PID function.
- **OPER4** - Not used.



Inputs and Outputs

Description of the inputs:

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

Description of the outputs:

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

Additional Parameters

Apart from operands programmed in the CHF call instruction other parameters should be loaded into the table declared in OPER3. This table should contain 16 positions, being used to define the parameters used for the control algorithm and to store intermediate results. The table 4-9 presents the parameters which should be loaded in each table position as well as minimum and maximum values.



Po	Parameter stored	Form	Variation allowed	Table value
00	Proportional gain x 10	GP x 10	GP: 1,0 a 100,0	10 a 1000
01	Integral factor - fraction part	dt / TI	TI: 1 a 1000 s/rep	0,0001 a 10,000
02	Integral factor - integral part		dt: 0,1 a 10 s	
03	Derivative factor - integral part	TD / 3dt	TD: 1 a 1000 s	0,0333 a 3333,3333
04	Derivative factor - integral part		dt: 0,1 a 10 s	
05	Dislocated	DE	0 a 1000	0 a 1000
06	Minimum value of output		0 a 1000	0 a 1000
07	Maximum value of output		0 a 1000	0 a 1000
08	Not used			
09	Variable measure N – 1			0 a 1000
10	Variable measure N – 2			0 a 1000
11	Variable measure N – 3			0 a 1000
12	Error			0 a 1000
13	Proportional action by x 10			0 a 65535
14	Whole action - part frac x 10			0 a 65535
15	Whole action - part int x 10			0 a 65535

Table 4-9 Additional Parameters of the P/D

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

The parameters which should be pre-calculated are:

- Proportional gain x 10 (position 0) - Is calculated by multiplying the proportional gain required for 10.
- Integral multiplicative factor - Is calculated by dividing the sample interval (dt) by the whole gain required. The unit of dt is seconds, its minimum value being 0.1 seconds and maximum 10.0 seconds and should be equal to the interval in which the routine is executed. The Ti is seconds/repetition, able to vary from 1 to 1000 seconds/repetition Ti equals to 1 second/repetition signifies the maximum integral effect.



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

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

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

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

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



Characteristics of Functioning

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

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

The form of control can be direct or reverse. This selection is carried out by turning off or powering the third input of the CHF instruction. If the process is such that the measure value grows when the value of the output of the action grows, the direct action should be selected. If the measure value decreases with the increase of the output of the action, then the reverse action should be used.

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

Example of Application

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

SP = 62
PG = 5 (PG = 100 / proportional band em %)
TI = 100 seconds/repetition
TD = 5 second



dt = 1 second
S = 50%
MAX = 80%
MIN = 0%

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

Position	Value	
0	50	PG X 10 (50)
1	100	dt / TI (0,0100)
2	0	
3	6666	TD / 3dt (1,6666)
4	1	
5	500	S
6	0	MIN
7	800	MAX
8	620	SP

Use

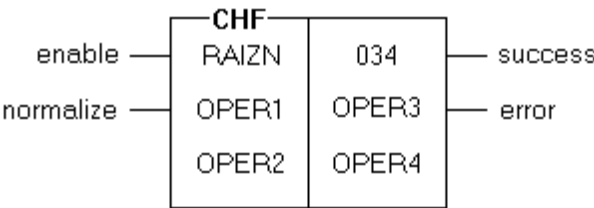
This function can be used in CPUs AL-600, AL-2000/MSP, AL-2002/MSP, AL-2003, QK800, QK801, QK2000/MSP, PL102, PL103, PL104 and PL105. For further information see F-PID16.056.

WARNING:

The module **F-PID.033** can be used in the PLC AL-2000/MSP only starting from version 1.10 or the executive software.



F - RAIZN.034 - Square Root Function with Normalization of Scale



Introduction

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

The calculation carried out corresponds to the following expression:

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

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

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

Programming

Operands

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

- **OPER1** - Specifies the number of parameters passed to the function in OPER3. It is compulsory for this operand to be a memory constant with value 3 (%KM+00003).
- **OPER2** - Should be an operand of type memory constant with value 0 (%KM+00000). It determines the number of parameters possible to be programmed in the editing window of OPER4. As this function does not need any parameter in OPER4, the value of OPER2 is 0.

- **OPER3** - Contains the parameters which are passed to the function, declared through a window visualised in MasterTool when the CHF instruction is edited. The number of editable parameters is specified in OPER1, being set at 3 for this module:
 - **%MXXXX** - Operand with the value to be extracted to the square root (source). This value should be positive so that the calculation can be carried out.
 - **%MXXXX or %KM+XXXX** - Memory or constant operand for the Normalization of button of scale of the square root extracted. The value programmed is divided by 256 and multiplied by the root of the operand supplied, giving the value of the destination operand, when the instructions second input is powered.
 - **%MXXXX** - Operand which receives the result of the normalized square root (destination).
- **OPER4** - Not used.

Inputs and outputs

Description of the inputs:

- **enable** - when this input is powered the function is called, the parameters programmed in the CHF instruction being analysed. If they are incorrect, all the outputs of the instruction are turned off. If they are correct, the calculations are carried out, the outputs success or error being actioned.
- **normalize** - when powered, carries out the adjustment of the button of scale to value of the square root obtained. If turned off, the value of the memory operand destination simply receives the square root of the source operand.

Description of the Outputs:

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



WARNING:

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

Use

This function can be used in the CPUs AL-600, AL-2000/MSP, AL-2002/MSP, AL-2003, QK800, QK801 and QK2000/MSP.

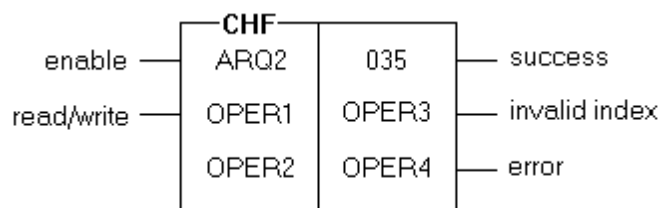
Example of Application

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

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



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



Introduction

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

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

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

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

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



Programming

Operands

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

- **OPER1** - Specifies the number of parameters passed to the function in OPER3. It is compulsory for this operand to be a memory constant with value 5 (%KM+00005).
- **OPER2** - Specifies the number of parameters passed to the function in OPER4. It is compulsory for this operand to be a memory constant with value 0 (%KM+0000).
- **OPER3** - Contains the parameters passed to the function, declared through a window visualised in MasterTool when the CHF instruction is edited. The number of editable parameters is specified in OPER1, being set at 5 for this module:
 - **%MXXXX, %DXXXX, %AXXXX, %EXXXX, %SXXXX, %TMXXXX, %TDXXXX, %KM+XXXXX or %KD+XXXXXXX** - Operand from where the data is read in the writing operations in the file or to where the data is copied into readings of the file (parameter 1).
 - **%MXXXX** - Number of register from/to which the data will be copied (parameter 2). Should contain between 0 and the total number of registers less 1.
 - **%MXXXX** - Number of field from/to which the data will be copied (parameter 3). Should contain between 0 and the total number of fields less 1.
 - **%KM+XXXXX** - Total number of registers (1 to 255) required for the file (parameter 4).
 - **%KM+XXXXX** - Total number of fields (1 to 255) required for the file (parameter 5).
- **OPER4** - Not used.



Inputs and Output

Description of the inputs:

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

Description of the outputs:

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

Use

This function can be used in the CPUs AL-600, AL-2000/MSP, AL-2002/MSP, AL-2003, QK800, QK801 and QK2000/MSP.

Description of the Functioning

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

Occupation of the file = Num. registers X Num. fields X Num. bytes per field



(parameter 4) (parameter 5)

The number of bytes per field occupied for each type of operand can be obtained from table 4-10.

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

Table 4-10 Occupation of the Field of the Files

The value obtained in the previous calculation should be less than or equal to the total capacity of the function used, according to table 4-11.



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

Table 4-11 Capacity of the Functions Data Files

WARNING:

Different CHF instruction for access to the same file can be declared in the same program. In all these instructions the operands with the values to be written or that receive the same number of bytes per field (c.f. table 4-5).

Therefore, it is possible to indistinctly read or write operands %E, %S and %A of one file or %KM, %M and %TM of another. Never the less they should not be accessed with operands %M or %D in the same file.

If the first parameter is a table (%TM or %TD), all the fields of the register indicated in the second parameter are copied, that is to say, the transfer of data is carried out between the register and the table, being that the value of the third parameter (number of field is ignored).

If the table has fewer positions than the number of fields in the register, only the fields which correspond to the existing positions are transferred. If the table has more positions than the number of fields in the register, only the existing fields are transferred.

The operation of writing the data copies it to the appropriate area of memory occupied by the function module.

WARNING:

If the module **F-ARQ** is stored in EPROM Flash, it is not possible to write data in the file, only to read data. To carry out the writing of the data into the files, the F modules corresponding to the them should be in the RAM memory of the applications program.



WARNING:

During the reading of a PLC's module data file with MasterTool or during its transferring from RAM to Flash, no writing of data to it should be carried out.

This is because the writing of data modifies the module read, being considered invalid by the programmer or by the PLC due to the altering of its checksum.

The functions data files are modules of the applications programs being able to be loaded or read by the PLC and stored on disks. For example, there may be the case of a PLC controlling a injector machine, storing different configuration parameters in an **F-ARQ8.037** module. After the parameters are stored, this module F can be read and stored on disk, to load in other equal injected machines.

Example of Application

As an example if it a file with 120 registers and with 8 fields for register to store operands %D, the occupation of memory will be:

Occupation of the file = 120 registers X 8 fields/register X 4 bytes/field

Occupation of the file = 3840

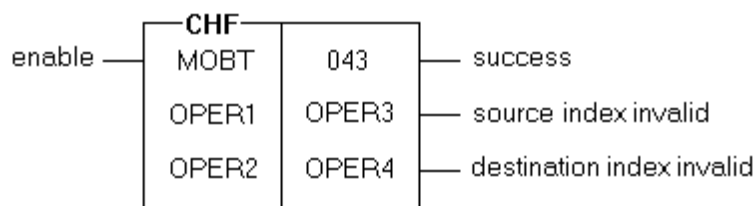
The configuration requires 3840 bytes to be occupied, the module **F-ARQ4.036** having to be used, then it allows the storing of 4096 bytes.

The parameters programmed in OPER3 of the CHF instruction for the access to the file are:

- %D0020 - operand to where it will be read or with the value to be written in the file
- %M0100 - contains the number of the register to be read or written, having to have between 0 and 119 (120 register in total).
- %M0101 - contains the number of the field to be read or written, having to have between 0 and 7 (8 register in total).
- %KM+00120 - total number of registers.
- %KM+00008 - total number of fields



F-MOBT.043 - Function for Moving of blocks from Table Operands



Introduction

The function **F-MOBT.043** carries out the copy of blocks of numeric operands (%M or %D) or positions of tables (%TM or %TD) up to 255 values of simple operands can be copied to tables and vice versa, also transferring the positions from one table to another. It is possible to specify the initial position of the block to be copied into the table supplied and into the destination table.

Programming

Operands

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

- **OPER1** - Specifies the number of parameters passed to the function in OPER3. It is compulsory for this operand to be a memory constant with value 5 (%KM+00005).
- **OPER2** - Should be an operand of type memory constant with value 0 (%KM+00000). It determines the number of parameters possible to be programmed in the editing window of OPER4. As this function does not need any parameter in OPER4, the value of OPER2 is 0.
- **OPER3** - Contains the parameters passed to the function, declared through a window visualised in MasterTool when the CHF instruction is edited. The number of editable parameter is specified in OPER1, being set at 5 for this module:
 - **%MXXXX, %DXXXX, %TMXXXX or %TDXXXX** - Initial operand from where the values are copied (source operand).



- **%KMXXXX** - Initial position to be transferred from the source operand is a simple operand (%M or %D).
- **%MXXXX, %DXXXX, %TMXXXX or %TDXXXX** - Initial operand where the values are copied to (destination operand).
- **%KMXXXX** - Initial position where the values in the destination table are copied to. This parameter is disregarded if the destination operand (%M or %D).
- **%KMXXXX** - Number of simple operands or table positions to be transferred starting from the operand or from the initial position in the parameters previously declared. It should be less than or equal to 255.
- **OPER4** - not used.

Inputs and Outputs

Description of the inputs:

- **enable** - when this input is powered the function is called, the parameters programmed in the CHF instruction being analysed. If these are incorrect, the outputs of the invalid index are actioned.

Description of the outputs:

- **success** - indicates that the moving was correctly carried out
- **source index invalid** - indicates that there was an error in the specification of the supply operand:
 - the operand is not declared in module C
 - the type of parameter 2 is not %KM
 - the initial position programmed does not exist, if the source operand is table
 - there are not enough operands or table positions to carry out the movement



destination index invalid - indicates that there was an error in the specification of the destination operand:

- the operand is not declared in module C
- the type of parameter 4 is not %KM
- the initial position programmed does not exist, if the destination operand is table
- there are not enough operand or table positions to carry out the movement

If the two outputs of the invalid index are actioned simultaneously, some of the following errors occur:

- the number of parameters programmed in OPER1 is different from five.
- the type of parameter 5 is not %KM
- the total number of position to be transferred is greater than 255

Use

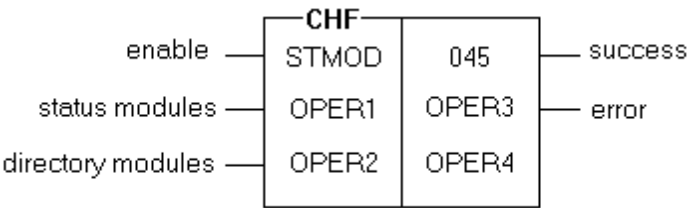
This function can be used in the CPUs AL-600, AL-2000/MSP, AL-2002/MSP, AL-2003, QK800, QK801 and QK2000/MSP.

WARNING:

This function allows the moving of a large number of operands in one scan. It should be used with care so that the maximum time of the program cycle is not exceeded.



F-STMOD.045 - Function Status of the Buses and I/O Modules



Introduction

The function **F-STMOD.045** makes possible the reading of the buses, of the octets and of the PLC's I/O modules. It allows special actions and procedures to be in the applications program, in the case of hot swap or error in some module or bus. It also places the arrangement of the applications program in the configuration of I/O modules used by the PLC.

Programming

Operands

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

- **OPER1** - Specifies the number of parameters passed to the function in OPER3. This operand must be a memory constant with value 4 (%KM+00004).
- **OPER2** - Should be an operand of type memory constant with value 0 (%KM+00000). Determines the number of parameters possible to be programmed in the editing window of OPER4. As this function does not need any parameter in OPER4, the value of OPER2 is 0.
- **OPER3** - Contains the parameters passed to the function, declared through a window visualised in MasterTool when the CHF instruction is edited. The number of editable parameters is specified in OPER1, being set at 4 for this module:

- **%MXXXX or %TMXXXX** - Memory or table operand which receives the status values of the I/O octets. If an operand %M is used, they should be defined in a minimum of 16 operands starting from the one declared (including it) so that the function may be executed correctly. If there is a table operand, it should have at least 16 positions. Each point of the operand %M or of the position of the table represents the status of an operand %E or %S (only the least significant points of its byte, .0 to .7). The value 0 in the point indicates that the octet is being updated normally by the I/O scan, while the value 1 indicates that the module or bus corresponding to the octet is in error or deactivated for hot swap. The table to follow shows the points of the operands corresponding to the status of the I/O octets.

WARNING:

For use with the PLC AL-2003 a minimum of 32 memory operands should be defined, or 32 table positions, as first parameter, for the visualisation of the statuses of the 256 I/O octets.

Operand	Table Position	I/O Octet Associated to the or Table Position the Operand							
		.7	.6	.5	.4	.3	.2	.1	.0
%MXXXX	0	%E0007	%E0006	%E0005	%E0004	%E0003	%E0002	%E0001	%E0000
%MXXXX+1	1	%E0015	%E0014	%E0013	%E0012	%E0011	%E0010	%E0009	%E0008
%MXXXX+2	2	%E0023	%E0022	%E0021	%E0020	%E0019	%E0018	%E0017	%E0016
%MXXXX+3	3	%E0031	%E0030	%E0029	%E0028	%E0027	%E0026	%E0025	%E0024
%MXXXX+4	4	%E0039	%E0038	%E0037	%E0036	%E0035	%E0034	%E0033	%E0032
%MXXXX+5	5	%E0047	%E0046	%E0045	%E0044	%E0043	%E0042	%E0041	%E0040
%MXXXX+6	6	%E0055	%E0054	%E0053	%E0052	%E0051	%E0050	%E0049	%E0048
%MXXXX+7	7	%E0063	%E0062	%E0061	%E0060	%E0059	%E0058	%E0057	%E0056

Table 4-12 Format for Storing of the Status of the I/O for the AL-2002/MSP

Operand	Table Position	I/O Octet Associated to the or Table Position the Operand							
		.7	.6	.5	.4	.3	.2	.1	.0
%MXXXX	0	%E0007	%E0006	%E0005	%E0004	%E0003	%E0002	%E0001	%E0000
%MXXXX+1	1	%E0015	%E0014	%E0013	%E0012	%E0011	%E0010	%E0009	%E0008
%MXXXX+2	2	%E0023	%E0022	%E0021	%E0020	%E0019	%E0018	%E0017	%E0016
%MXXXX+3	3	%E0031	%E0030	%E0029	%E0028	%E0027	%E0026	%E0025	%E0024
%MXXXX+4	4	%E0039	%E0038	%E0037	%E0036	%E0035	%E0034	%E0033	%E0032
%MXXXX+5	5	%E0047	%E0046	%E0045	%E0044	%E0043	%E0042	%E0041	%E0040
%MXXXX+6	6	%E0055	%E0054	%E0053	%E0052	%E0051	%E0050	%E0049	%E0048
%MXXXX+7	7	%E0063	%E0062	%E0061	%E0060	%E0059	%E0058	%E0057	%E0056
%MXXXX+8	8	%E0071	%E0070	%E0069	%E0068	%E0067	%E0066	%E0065	%E0064
%MXXXX+9	9	%E0079	%E0078	%E0077	%E0076	%E0075	%E0074	%E0073	%E0072
%MXXX+10	10	%E0087	%E0086	%E0085	%E0084	%E0083	%E0082	%E0081	%E0080
%MXXX+11	11	%E0095	%E0094	%E0093	%E0092	%E0091	%E0090	%E0089	%E0088
%MXXX+12	12	%E0103	%E0102	%E0101	%E0100	%E0099	%E0098	%E0097	%E0096
%MXXX+13	13	%E0111	%E0110	%E0109	%E0108	%E0107	%E0106	%E0105	%E0104
%MXXX+14	14	%E0119	%E0118	%E0117	%E0116	%E0115	%E0114	%E0113	%E0112
%MXXX+15	15	%E0127	%E0126	%E0125	%E0124	%E0123	%E0122	%E0121	%E0120
%MXXX+16	16	%E0135	%E0134	%E0133	%E0132	%E0131	%E0130	%E0129	%E0128
%MXXX+17	17	%E0143	%E0142	%E0141	%E0140	%E0139	%E0138	%E0137	%E0136
%MXXX+18	18	%E0151	%E0150	%E0149	%E0148	%E0147	%E0146	%E0145	%E0144
%MXXX+19	19	%E0159	%E0158	%E0157	%E0156	%E0155	%E0154	%E0153	%E0152
%MXXX+20	20	%E0167	%E0166	%E0165	%E0164	%E0163	%E0162	%E0161	%E0160
%MXXX+21	21	%E0175	%E0174	%E0173	%E0172	%E0171	%E0170	%E0169	%E0168
%MXXX+22	22	%E0183	%E0182	%E0181	%E0180	%E0179	%E0178	%E0177	%E0176
%MXXX+23	23	%E0191	%E0190	%E0189	%E0188	%E0187	%E0186	%E0185	%E0184
%MXXX+24	24	%E0199	%E0198	%E0197	%E0196	%E0195	%E0194	%E0193	%E0192
%MXXX+25	25	%E0207	%E0206	%E0205	%E0204	%E0203	%E0202	%E0201	%E0200
%MXXX+26	26	%E0215	%E0214	%E0213	%E0212	%E0211	%E0210	%E0209	%E0208
%MXXX+27	27	%E0223	%E0222	%E0221	%E0220	%E0219	%E0218	%E0217	%E0216
%MXXX+28	28	%E0231	%E0230	%E0229	%E0228	%E0227	%E0226	%E0225	%E0224
%MXXX+29	29	%E0239	%E0238	%E0237	%E0236	%E0235	%E0234	%E0233	%E0232
%MXXX+30	30	%E0247	%E0246	%E0245	%E0244	%E0243	%E0242	%E0241	%E0240
%MXXX+31	31	%E0255	%E0254	%E0253	%E0252	%E0251	%E0250	%E0249	%E0248

Table 4-13 Format for storing of the I/O Status for the AL-2003

n.b.:

- The octets are all represented as input (%E).

Starting from the determined address the outputs (%S) should be taken into consideration, depending on the configuration of modules used in the buses.

- In the PLC AL-2002, the last 8 operand %M or the last 8 table position are reserved for future use, having to be declared for correct execution of the function.

- If the PLC is an AL-2003, all the operands or 32 table position are used for storing the I/O status.

- Value of point: 0 - octet being updated in normal form with the I/O
module corresponding or not used
in the configuration of the bus
1 - octet not updated, bus or module
in error or deactivated for hot swap.

%MXXXX or %TMXXXX - Memory or table operand which receives the values of the bus status. If the operand %M is used, a minimum of 10 operands should be defined starting from and including the one declared so that the function may be executed correctly. If it is a table operand this should have at least 10 positions. Each operand or table position corresponding to the status of a bus.



Operand	Table Position	Associated Bus
%MXXXX	0	Status of bus 0
%MXXXX + 1	1	Status of bus 1
%MXXXX + 2	2	Status of bus 2
%MXXXX + 3	3	Status of bus 3
%MXXXX + 4	4	Status of bus 4
%MXXXX + 5	5	Status of bus 5
%MXXXX + 6	6	Status of bus 6
%MXXXX + 7	7	Status of bus 7
%MXXXX + 8	8	Status of bus 8
%MXXXX + 9	9	Status of bus 9

Table 4-14 Format for storing the Status of the Buses

n.b.

- Status values: %MXXXX = 0 - bus not used
 %MXXXX.4 (bit 4) - bus with error
 %MXXXX.6 (bit 6) - bus deactivated for module hot swap
 %MXXXX.7 (bit 7) - bus functioning normally

- **%MXXXX or %TMXXXX** - Memory or table operand which receives the values of the modules status. If an operand %M is used, a minimum of 160 operands should be defined starting from and including the one declared so that the function may be executed correctly. If it is a table operand, this should have at least 160 positions. Each operand %M or table position corresponds to a module of the bus, with its status represented by the following values:



%MXXXX = 0 - module non-existent
 %MXXXX.4 (bit 4) - module with error
 %MXXXX.6 (bit 6) - module de-activated for hot swap
 %MXXXX.7 (bit 7) - module functioning normally

- **%MXXXX or %TMXXXX** - Memory or table operand which receives the values from the modules directory. If and operand %M is used, a minimum of 160 operands should be defined starting from and including the one declared so that the function may be executed correctly. If it is a table operand, this should have at least 160 position. Each operand %M or table position corresponds to a bus module, containing its respective identification code. This code can be obtained in the editing window of the parameters of the I/O modules in MasterTool (options *Options Modules of I/O*) in the field type.
- **OPER4** - not used.

Inputs and Outputs

Description of inputs:

- **enable** - when this input is powered the function is called, the parameters programmed in the CHF instruction being analysed. If they are incorrect, the output success is turned off. If they are correct, the function carries out the reading of the modules directory to the operand declared in OPER3.
- **status modules** - when this input is powered, the function carries out the status reading of the modules to the operand declared in OPER3.
- **directory modules** - when this is powered, the function carries out the reading of the modules to the operand declared in OPER3.

Description of the outputs:

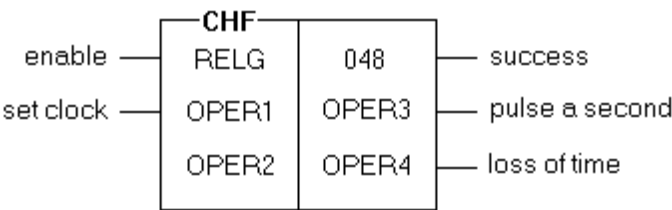
- **success** - this outputs is powered when the function was correctly executed.
- **error** - is powered when an error occurs in the specification of the operands or when there is an attempt to access the operands not declared.

Use

This function can only be used in the CPUs AL-2002/MSP and AL-2003.



F-RELG.048 - Function to Access the Real Time Clock



Introduction

The function **F-RELG.048** carries out the access of the real time clock contained in the CPU AL-2002. The clock has complete hour and calendar, allowing the development of applications programs which depend on precise time bases. The time information is kept the same when there is power failure in the system, since the CPU is powered by batteries.

This function has similar characteristics to function **F-SINC.049**, since both execute accesses to the same clock, differing only in the methods of setting. They can be used simultaneously in the same program, if necessary.

Programming

Operands

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

- **OPER1** - Specifies the number of parameters passed to the function in OPER3. This operand must be a memory constant with value 2 (%KM+00002).
- **OPER2** - Should be an operand of type memory constant with value 0 (%KM+00000). Determines the number of parameters possible to be programmed in the editing window of OPER 4. As this function does not need any parameter in OPER4, the value of OPER2 is 0.

- **OPER3** - Contains the parameter passed to the function, declared through a window visualised in MasterTool when the CHF instruction is edited. The number of editable parameters is specified in OPER1, being set at 2 for this module:
- **%MXXXX or %TMXXXX** - Specification of the operands to where the clock values are read. If this parameter is specified as memory, the values are read to the memory declared and the following six. If it is specified as table, the values are placed starting from position 0 to 6. If the operands are not declared, the reading of the time values is not carried out and the instruction outputs are disconnected. It is possible to use tables with more than 7 positions, being that values are read from operands in the following sequence:

Operand	Table Position	Content	Format
%MXXXX	0	Seconds	000XX
%MXXXX + 1	1	Minutes	000XX
%MXXXX + 2	2	Hours	000XX
%MXXXX + 3	3	Day of the month	000XX
%MXXXX + 4	4	Month	000XX
%MXXXX + 5	5	Year	000XX
%MXXXX + 6	6	Day of the week	000XX

Table 4-15 Values Read from the Clock (F-RELG.048)

The contents of these operands can be read at any time, but are updated with the real hour of the clock only when the instruction is executed. The 24 hour format is used in the time count. The days of the week are counted with values from 1 to 7.



Value	Day of the Week
1	Sunday
2	Monday
3	Tuesday
4	Wednesday
5	Thursday
6	Friday
7	Saturday

Table 4-16 Values of the Days of the Week (F-RELG.048)

- **%MXXXX or %TMXXXX** - Specification of the operands from where the clock values are set, with the actioning of some of the inputs to set the function. If this parameter is specified as memory, the values are copied from the memory declared and the following 6. If it is specified as table, the values are copied from position 1 to 6. If the operands are not declared, the setting is not carried out and the outputs of the instructions are disconnected. The values to be copied to clock should be placed in the operands in the same sequence as the operands of reading (seconds, minutes, hours, day of the month, year and day of the week).
- **OPER4** - Not used.

Inputs and Outputs

Description of the outputs:

- **enable** - when this input is powered the function is called, the parameters programmed in the CHF instruction are analysed. If they are incorrect, all the output of the instruction are turned off. If they are correct, the time values of the clock are transferred to the memory operands or to a table declared as first parameter in OPER3, the output success is powered and the output pulse a second is connected by a scan at each second.
- **set clock** - when powered, the values of the operands declared as second parameter in OPER3 are set in the clock, if the values are correct. While the input is actioned the time is not counted, the output pulse a second remaining turned off.



Example:

input set :



programmed time: 9h 35m 20s

output pulse a second:

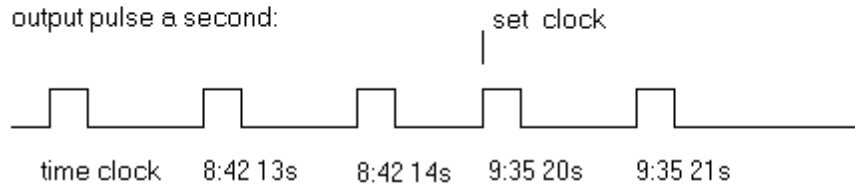


Figure 4-7 Example of Diagram of Setting Input Times

Description of the output:

- **success** - is powered when the function has been correctly executed.
- **pulse a second** - indicates if there was a change in the clock. The pulse lasts one scan and can be used to synchronise events of the applications program.
- **loss of time** - this output is connected if the clock was left without battery power during a failure of the main supply. It is deactivated with the setting of the clock.

Use

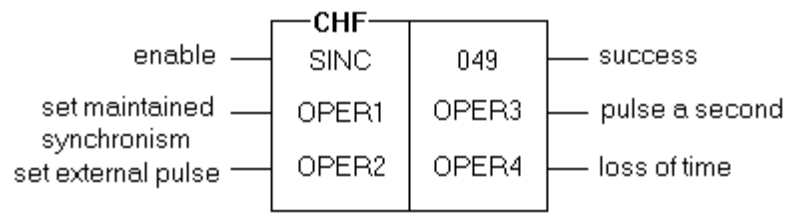
This function can only be used in the CPUs AL-2002/MSP, AL-2003, PL104 and PL105.

WARNING:

The battery power is supplied to the CPU through the bus. Therefore, the removal of the CPU AL-2002/MSP from the background results in the loss of the clock's time.



F-SINC.049 - Function to Access the Synchronized Real Time Clock



Introduction

The function **F-SINC.049** carries out access to the real time clock contained in the CPU AL-2002/MSP. The clock has complete time and calendar, allowing the development of applications programs which depend on precise time bases. The time information is kept the same with a system power failure, since the CPU is powered by batteries

The clocks of various AL-2002 controllers can operate synchronised with a precision of 1 millisecond, through the synchronism networks and ALNET II. The function **F-SINC.049** has characteristics for use in controllers which operate with their clocks synchronised, especially in the PLC's synchronism generator.

This function has similar characteristics to the function **F-RELG.048** since both execute access to the same clock, differing only in the setting methods. They can be used simultaneously in the same program if necessary.

For further information concerning the synchronisation of the controllers, consult the AL-2002 User's Manual or the ALTUS Networks Manuals.

Programming

Operands

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

- **OPER1** - Specifies the number of parameters passed to the function in OPER3. This operand must be a memory constant with value 2 (%KM+00002).
- **OPER2** - Should be an operand of type memory constant with value 0 (%KM+00000). It determines the number of parameters possible to be programmed in the editing window of OPER4. As this function does not need any parameter in OPER4, the value of OPER2 is 0.
- **OPER3** - Contains the parameters passed to the function, declared through a window visualised in MasterTool when the CHF instruction is edited. The number of editable parameters is specified in OPER1, being set at 2 for this module:
 - **%MXXXX or %TMXXXX** - Specification of the operands to where the clock values are read. If this parameter is specified as memory, the values are read to the memory declared and the following 6. If it is specified as table the values are placed starting from position 0 to 6. If the operands are not declared, the reading of the time values is not carried out and the instructions outputs are disconnected. It possible to use tables with more than 7 position, as the function disregards the surplus positions. The values are read from the operands in the following sequence:

Operand	Table Position	Content	Format
%MXXXX	0	Seconds	000XX
%MXXXX + 1	1	Minutes	000XX
%MXXXX + 2	2	Hours	000XX
%MXXXX + 3	3	Day of the month	000XX
%MXXXX + 4	4	Month	000XX
%MXXXX + 5	5	Year	000XX
%MXXXX + 6	6	Day of the week	000XX

Table 4-17 Values Read from the Clock (F-SINC.049)

The contents of these operands can be read at any time, but are updated with clock's the real time only when the instruction is executed. The 24 hour format is used in the time count. The days of the week are counted with values from 1 to 7:



Value	Day of the Week
1	Sunday
2	Monday
3	Tuesday
4	Wednesday
5	Thursday
6	Friday
7	Saturday

Table 4-18 Values of the Days of the Week (F-SINC.049)

- **%MXXXX or %TMXXXX** - Specification of the operands from where the clock value are set, with the actioning of any of the function's setting inputs. If this parameter is specified as memory, the value are copied from the memory declared and the following 6. If specified as table the values copied from position 0 to 6. If the operands are not declared, the setting is not carried out and the instruction's outputs are disconnected. The values to be copied to the clock should be placed in the operands in the same sequence of the reading operands (seconds, minutes, hours, day of the month, month, year and day of the week).
- **OPER4** - Not used.

Inputs and Outputs

Descriptions of the inputs:

- **enable** - when this input is powered the function is called, with the parameters programmed in the CHF instruction being analysed. If these are incorrect, all the instruction's outputs are turned off. If they are correct the clock's time values are transferred to the memory operands or to the table declared as first parameter in OPER3, the output success is powered and the pulse a second is connected by a scan every second.
- **set maintained synchronism** - when powered, the values of the operands declared as second parameter in OPER3 are set in the clock, if they are correct, the sequence for increasing by seconds being maintained. In this way, the sequence of the second pulses is not altered, only the time value contained in the clock. This characteristic is useful to set the PLC's synchronism generator, since it does not modify the time base of the rest of the system's controllers, only the time values.



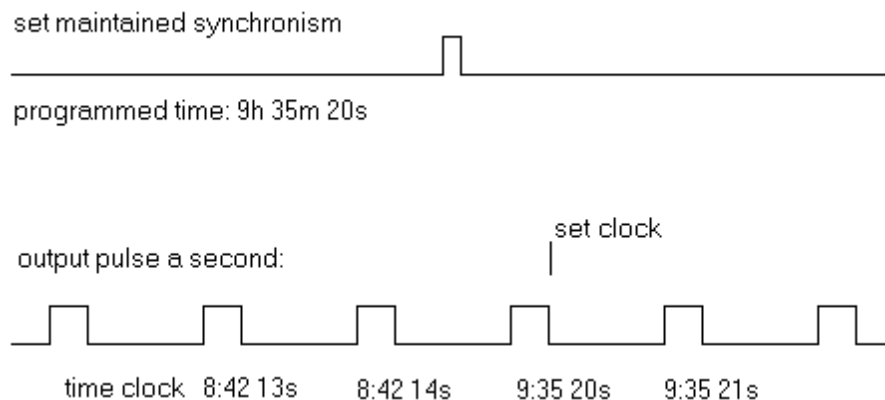


Figure 4-8 Example of Diagram of Times of Input Set Maintained Synchronism

- set external pulse** - when powered, the values of the operands declared as second parameter in OPER3 are programmed to be set in the clock in the next pulse in the synchronism input of the PLC, if the PLC is configured as synchronism generator. When the pulse occurs in the synchronism input, the time values are accepted, if the values are correct. The second count is started immediately the external pulse is actioned, changing the sequence of the second pulses generated. In this way it is possible to set the controller which manages the control system's synchronism of external clocks or other systems. It is not necessary for the input to remain powered until the actioning of the pulse for this setting to be carried out.



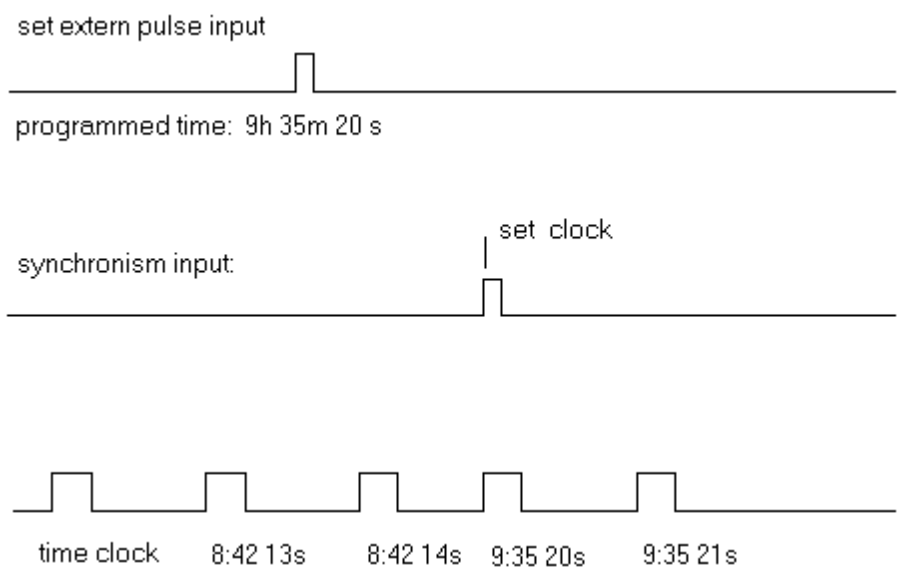


Figure 4-9 Example of Diagram of Times of Input Setting External Pulse

WARNING:
This setting is only possible for PLCs configured as synchronism generators. The external pulse should occur up to 6 seconds after the input to be actioned. After this period the time values are not changed by an external pulse, only initializing the second count.

If the PLC is configured as synchronism generator, the simultaneous actioning of the inputs sets maintained synchronism and the setting of the external pulse is done so that the PLC generates a setting of the time values in all the other PLCs connected to the synchronism network (absolute setting), not modifying the time of its clock. In order for this setting to occur, apart from the PLCs connected to the synchronism network, they should be interconnected by ALNET II. This setting is identical to that carried out automatically each minute by the PLC's synchronism generator.



Description of the outputs:

- **success** - is powered when the function has been correctly executed.
- **pulse a second** - indicates it there was a change in the clock's second count. The pulse applications program's events.
- **loss of time** - this output is connected if the clock was left without battery power during failure of the main supply. It is deactioned with the setting of the clock.

Use

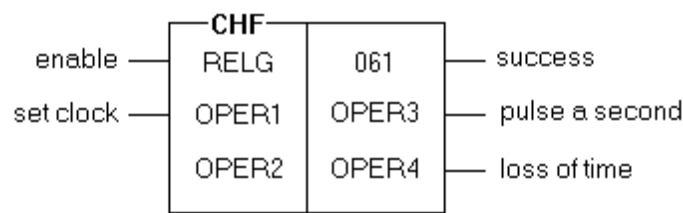
This function can only be used in CPUs AL-2002/MSP and AL-2003.

WARNING:

The battery power is supplied to the CPU through the bus. Therefore, the removal of the CPU AL-2002/MSP from the rack results in the loss of the clock's time.



F-RELOG.061 - Function to Access the Real Time Clock of QK801 and QK2000



Introduction

The function **F-RELOG.061** carries out the access to the real time clock contained in CPUs QK2000 and QK801. The clock has complete time and calendar, allowing the development of applications programs which depend on precise time bases. The time information is kept the same when there is a power failure in the CPU, since the clock remains powered by battery.

Programming

Operands

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

- **OPER1** - Specifies the number of parameters passed to the function is OPER3. This operand must be a memory constant with value 2 (%KM+00002).
- **OPER2** - Should be an operand of type memory constant with value 0 (%KM+00000). It determines the number of parameters possible to be programmed in the editing window of OPER4. As this function does not need any parameter in OPER4, the value of OPER2 is zero.
- **OPER3** - Contains the parameters passed to the function, declared through a window visualised in MasterTool when the instruction CHF is edited. The number of editable parameters is specified in OPER1, being set at 2 for this module:

- **%MXXXX or %TMXXXX** - Specification of the operands where the clock value are read to. If this parameter is specified as memory the values are read to the memory declared and the following 6. If its is specified as table, the values are placed starting from position 0 to 6. If the operands are not declared, the reading of the time values is not carried out and the instruction outputs are disconnected. It is possible to use tables with more than 7 positions, as the function disregards surplus positions. The values are read from the operands in the following sequence:

Operand	Table Position	Content	Format
%MXXXX	0	Seconds	00XX
%MXXXX + 1	1	Minutes	00XX
%MXXXX + 2	2	Hours	00XX
%MXXXX + 3	3	Day of the Month	00XX
%MXXXX + 4	4	Month	00XX
%MXXXX + 5	5	Year	00XX
%MXXXX + 6	6	Day of the Week	00XX

Table 4-19 Values Read from the Clock (F-RELOG.061)

The contents of these operands can be used at any time, but are only updated with the real time of the clock when the function is executed. The 24 hour format is used to count the time and the days of the week are counted with values from 1 to 7:



Value	Day of the Week
1	Sunday
2	Monday
3	Tuesday
4	Wednesday
5	Thursday
6	Friday
7	Saturday

Table 4-20 Values of the Days of the Week (F-RELOG.061)

- **%MXXXX or %TMXXXX** - Specification of the operands where the clock value are set from, with the actioning of the input of setting the function. If this parameter is specified as memory, the values are copied from the memory declared and the 6 following. If it specified as table, the values are copied from position 0 to 6. If the operands are not declared, the setting is not carried out and the instruction's outputs are disconnected. The values to be copied to the clock should be placed in the operands in the same sequence as the operands of the reading (seconds, minutes, hours, day or the month, month year and day of the week)
- **OPER4** - Not used.

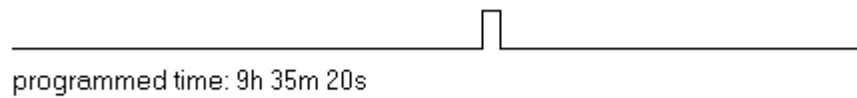
Inputs and Outputs

Description of the inputs:

- **enable** - when this input is powered the function is called, the parameters programmed in the CHF instruction being analysed. If they are incorrect all the outputs of the instruction are turned off. If they are correct, the time values of the clock are transferred to the memory operands or to the table declared as first parameter in OPER3, the output success is powered and the output pulse a second is connected by a scan every second.
- **set clock** - when powered, the values of the operands declared as second parameter in OPER3 are set in the clock, if they are with correct values. While this input is actioned the time is not counted, the output pulse a second remaining turned off.



input set :



output pulse a second:

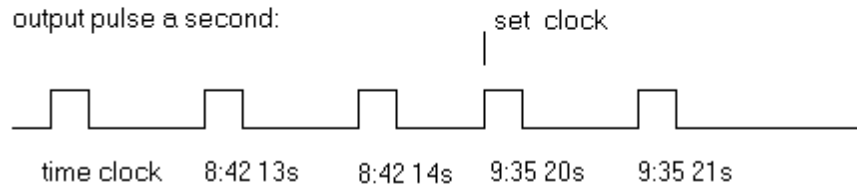


Figure 4-10 Example of Diagram of Input Set Times

Description of the outputs:

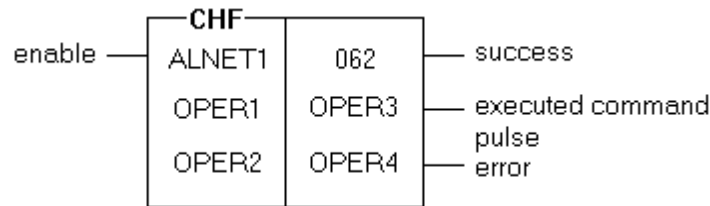
- **success** - is powered when the function was executed correctly.
- **pulse a second** - indicates if there was a change in the clock's second count. The pulse lasts for one scan and can be used to synchronise applications program events.
- **loss of time** - this output is connected if the clock was left without battery power during failure of the main supply. It is de-activated with the setting of the clock.

Use

This module can be used in CPUs QK801 and QK2000/MSP.



F-ALNET1.062 - Function Interpreter of the ALNET I Protocol for QK801



Introduction

The function **F-ALNET1.062** implements the communication in the secondary serial channel of the QK801 controller, allowing it to receive and execute ALNET I protocol commands as a slave device. In this way, the PLC QK801 can be connected to an ALNET I supervision network or to peripheral equipment through this channel

Programming

Operands

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

- **OPER1** - Specifies the number of parameters passed to the function in OPER3. This operand must be a memory constant with value 2 (%KM+00002).
- **OPER2** - Should be an operand of type memory constant with value 0 (%KM+00000). It determines the number of parameters possible to be programmed in the editing window of OPER4. As this function does not need any parameter in OPER4, the value of OPER2 is zero.
- **OPER3** - Contains the parameters passed to the function, declared through a window visualised in MasterTool when the CHF instruction is edited. The number of editable parameters is specified in OPER1, being set at 2 for this module:



- **%KMXXXX** - Specification of the baud rate of the communication in the secondary serial channel. The value of the constant corresponding directly to the number of bauds, being able to take on the values 9600, 4800, 2400, 1200, 600 or 300.
- **KMXXXX** - Specification of use of MODEM signals (RTS, CTS, DTR and DSR), when the secondary serial channel is used in standard RS-232. If programmed with the value 0, the communication does not use the MODEM signals. This constant should be programmed with the value 0 when the secondary serial channel follows the RS-485 standard .
- **OPER4** - Not used.

Input and Outputs

Description of the inputs:

- **enable** - when this input is powered the function is called, the parameters programmed in the CHF instruction being analysed. If any are incorrect, the error output is powered. If they are correct, the function interprets, executes and responds to the ALNET I protocol received in the secondary serial channel of the PLC.

WARNING:

To function correctly, the input enabled should be powered in the execution applications program's first execution cycle. The configuration of the secondary serial channel of the QK801 is executed in the first cycle.

Description of the outputs:

- **success** - is powered when the function was called (is present in the controller)
- **executed command pulse**- indicates that a command received by the serial channel has been processed, being powered by a scan.
- **error** - is permanently powered in case an error occurs in the programming of the function's constants or in the serial communication by a scan.



ALNET I Protocol

The second serial channel of the PLC QK801 is directed for use with supervisory systems or man-machine interfaces, not executing the commands referring to the modules of the applications program, nor the PLC's change of status commands.

The ALNET I commands executed by module F are shown in the table to follow, indicating the version of the protocol to which they belong. The commands of version 1.00 are used in networks which contain controllers from the series AL-1000 or AL-500. The commands of version 2.00 can be used in networks which contain only controllers from the series AL-600, AL-2000, AL-3000 or QUARK, not having any controller from the series AL-1000 or AL-500.



Num	Description of the command	V1.00	V2.00
002	Force simple operand AL-1000	x	
004	Free all those forced	x	x
006	Monitor simple operand AL-1000	x	
009	Disable digital output	x	x
010	Enable digital outputs	x	x
012	Force table position AL-1000	x	
013	Monitor table position AL-1000	x	
014	Force block of table AL-1000	x	
015	Monitor block of table AL-1000	x	
032	Receive program module		x
037	Read status		x
038	Read program's modules directory		x
039	Read program module's status		x
040	Monitor simple operands		x
041	Monitor table operands		x
042	Read status of forcings		x
129	Force simple operands		x
130	Force table operands		x
131	Free operands		x
133	Write simple operands		x
134	Change the protection level		x
135	Change the password		x

Table 4-21 Commands Executed by Module F - ALNET1.062

The table to follow shows the commands which are not executed by module **F-ALNET1.062**:



Num	Description of the command	V1.00	V2.00
005	Pass to programming mode	*	*
007	Pass to cycled mode	*	*
008	Execute a cycle	*	*
011	Pass to execution mode	*	*
031	Request to load program module		*
033	Remove program module		*
034	Transfer module from EPROM to RAM		*
035	Re-enable module in EPROM		*
036	Compact RAM memory		*
045	Transfer module from RAM to EPROM flash		*
046	Erase EPROM Flash memory		*
047	Read communication status		*
193	Load from program module		*

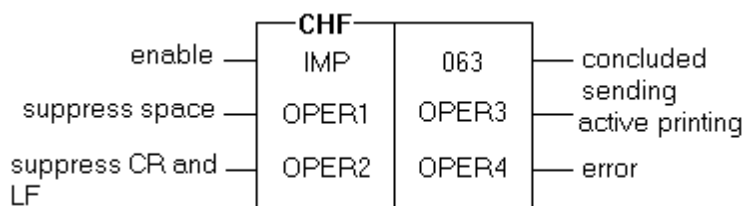
Table 4-22 Commands not executed by Module F-ALNET1.062

Use

This module can be used in the CPU QK801.



F-IMP.063 Function for Printing ASCII Characters



Introduction

The function **F-IMP.063** allows ASCII characters to be sent by the main serial channel of the programmable controllers to devices as printer or video terminals, allowing the printing or visualisation of pre-defined texts in conjunction with values of operands.

This function can work in two distinct ways: sending text and memory operands together or sending a sequence of values of up to 255 memory operands, without texts. The values of the memory operands are coded in ASCII format, for sending through the serial communication interface.

Programming

Operands

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

- **OPER1** - Specifies the number of parameters passed to the function in OPER3. This operand must be a memory constant with value 3 (%KM+00003).
- **OPER2** - Should be an operand of type constant with value D (%KM+0000). Determines the number of parameters possible to be programmed in the editing of OPER4. As this function does not need any parameter in OPER4, the value of OPER2 is zeroed.



- **OPER3** - Contains the parameters passed to the function, declared through a window visualised in MasterTool when the CHF instruction is edited. The number of editable parameters is specified in OPER1, being set at three for this module:
 - **%MXXXX or %TMXXXX** - When specified as %MXXXX, indicates the initial operand to be sent through the PLC's primary channel. When specified as %TMXXXX it indicates the table which contains the text and/or memory operands to be sent. The maximum number of characters sent is 255, allowing the use of one %TM from the maximum 128 positions.
 - **%KM+XXXXX** - Indicates the number of memory operands to be sent starting from the operand %MXXXX specified in the previous parameter. When %TMXXXX is used in the previous parameter, the value of this parameter is ignored by the function. The value of this operand should be between 1 and 255.
 - **%MXXXX** - Control operand of the function.

WARNING:

The control operand is for the exclusive use by the function, not being able to be altered in any part of the applications program, under penalty of endangering its correct execution.

- **OPER4** - Not used.

Description of the inputs:

- **enable** - when this input is powered the function is called, the programmed parameters being analysed in the CHF instruction. If they are incorrect, the output **error** is powered. This input should remain powered until the output **concluded sending** is pulsed.

WARNING:

Once started, the execution of the **F-IMP.063** function should continue until it is finished (level of the output **concluded sending** returning to 0 after to take on status in (). In this way, the CHF instruction which carries out the call to **F-IMP.063** should not be jumped nor turned off.

- **suppress space** - this output is only used when the first parameter of OPER3 is specified as %MXXXX. When powered, it suppresses the 5 spaces in white sent between the value of each operand %M.



- **suppress CR and LF** - this output is only used when the first parameter of OPER3 is specified as %MXXXX. When powered, it suppresses the sending of the CR characters (return of car) and LF (new line) after the sending of the operands has been completed.

Description of the outputs:

- **concluded sending** - this output is powered for a scan as soon as the sending of characters has been completed.
- **active printing** - this output is powered while the character are being sent to the output device. So straightaway the sending closes, the output is turned off.
- **error** - is powered if there is an error in the operands' specification, attempt to access operands not declared or time out in the signal test of MODEM (CTS and RTS).

Use

- Number of calls - all the calls belonging to **F-IMP.063** can exist in the applications program if necessary, until the last of their characters has been sent, even if other have been enabled. When the sending of the ASCII characters from the active function is concluded a scan pulse in the output **concluded sending** indicates that the communication channel is free, being activated at the next call of the **F-IMP.063** which is enabled.
- Printing of texts - if an operand %TM is used as the function's first parameter, the first table position after the last character to be sent should have the value 0. This value is recognised by the function as a marker for the end of the text. If this value is omitted, the function may send all the characters existing in the table up to a limit of 255.

The text to be sent should be stored in the table before the call module F-IMP.063 is enabled, CAB instructions being able to be used for this purpose. The text can be visualised in ASCII format during the insertion of the CAB instruction.

The ASCII characters used by the function are in the range 0 to 127.

- Use of the signals of MODEM (CTS and RTS) - in a way that allows the use of devices which use or don't use MODEM signals, the function can carry out treatment of the RTS and CTS signals or depress them. This definition is achieved via MasterTool, the treatment of the signals is only carried out in the sending of the first byte of each communication. In this way it should administer the sending of the characters into the function of the buffer size of the reception by destination device.



Being selected for use by MODEM signals, if there is no actioning of CTS signal after 200ms of CTS signal after 200ms of the actioning of the RTS signal, the output **error** is powered, with the following aspects having to be verified:

- connection of cables which connects the PLC to the output device.
- sending of character not recognised by the output device.
- incompatibility of configuration between the PLC and the output device.
- permanent over flow reception buffer of output device in function of its size X rate of sending characters.
- Configuration of serial channel - the function works with the following serial channel configuration:
 - 8 bits of data
 - without parity
 - band rate configurable (via MasterTool)

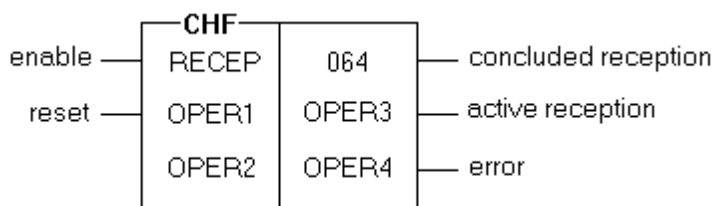
After execution is complete, the function automatically reconfigures the serial channel, allowing communication via the ALNET I protocol.

- Execution time - When the input enable is powered and no other printing was active, the function analyses the parameters, prepares the group of characters to be sent and fires its transmission. Depending on the contents to be sent, this preparation can need quite a long time for execution, significantly increasing the execution time of this cycle of the applications program. In the subsequent cycles the function only tests the end of the sending of the characters, with the execution time somewhat reduced.

This function can be used in the CPUs QK800, QK801 and QK2000/MSP.



F-RECEP.064 - Function for Reception of ASCII Characters



Introduction

The function **F-RECEP.064** allows the reception of ASCII characters through the main serial channel of the PLC.

Through this function is possible receive characteres of any devices with RS-232 serial interface having 1 start bit, 7 data bits, 1 parity bit (even) and 1 stop bit.

Through this function it is possible to receive a maximum of 255 characters, which are stored in memory operands or in a table.

Programming

Operands

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

- **OPER1** - Specifies the number of parameter passed to the function in OPER4. This operand must be a memory constant with value 4 (%KM+00004).
- **OPER2** - Should be an operand of type memory constant with value 0 (%KM+00000). Determines the number of parameter possible to be programmed in the editing window of OPER4. As this function does not need any parameter in OPER4, the value of OPER2 in zeroed.
- **OPER3** - Contains the parameters passed to the function, declared through a window visualised in MasterTool when the CHF instruction is



edited. The number of editable parameters is specified in OPER1, being set at four for this module:



- **%MXXXX or %TMXXXX** - Storing operand. When specified as %MXXXX, it indicates the first memory operand of reception, starting from the characters, received by the serial channel, which are stored. When specified as %TMXXXX it indicates the table which should receive the ASCII characters.
- **%KM+XXXXX** - Position of storing. Shows the first table position to be occupied by data received through the serial channel, if the storing operand (first parameter) is a table. If the first parameter is a memory (%MXXXX), the value of this constant is disregarded.
- **%KM+XXXXX** - Number of characters. Shows the number of characters to be received in a communication.

WARNING:

A communication can receive a maximum of 255 characters. If stored in a table an attempt should be made to ensure that the table has enough positions to receive the characters programmed. That is to say, the initial table position plus the number of table operand position.

- **%MXXXX** - Control operand of the function.

WARNING:

The control operand is for the exclusive use of the function, and is not supposed to be altered in any part of the applications program under penalty of endangering its correct execution.

- **OPER4** - Not used.

Description of the inputs:

- **enable** - when this input is powered the function is called, the parameters programmed being analysed in the CHF instruction. If they are incorrect, the output error is powered. This input should remain turned on until the output sending concluded is pulsated.

WARNING:

In order that the characters are correctly received and stored, while the call to the function **F-RECEP.064** is enabled it should not be jumped through any jump reel instruction.

- **reset** - this input is used to re-initialize the function, returning to store the characters received starting from the first position or operand programmed. While it is powered the function remains inactive.

Description of the outputs:



- **concluded reception** - this output is powered through a scan as soon as the total number of characters is received.
- **active reception** - this output is powered while the characters are being received. When the reception is finished, with the arrival of the last character programmed, this output is turned off.
- **error** - this output is powered if an error occurs in the specification of the operands, an attempt is a parity error in the characters received.

Use

All the calls of **F-RECEP.064** can be present in the applications program if necessary. However only one call remains active in the program, up to the end of the reception of the characters programmed, even if others are enabled. After the reception of active functions ASCII characters has finished, a scan pulse in the output **concluded reception** indicates that the communication channel is free, being activated at the next call of **F-RECEP.064** which is enabled.

The ASCII characters used by the function are in the range 0 to 127.

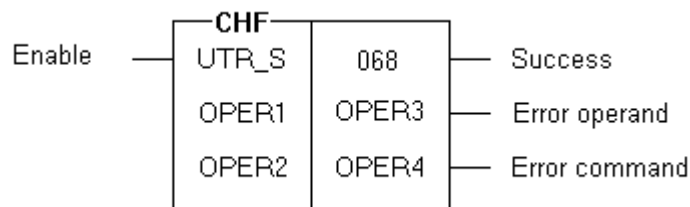
The function works with the following serial channel configuration:

- 7 data bits
- 1 parity bit (par)
- configurable baud rate (via MasterTool).

This function can be used in the CPUs QK800, QK801 and QK2000/MSP.



F-UTR_S.068 - Function to turn on UTRs outputs



Introduction

F-UTR_S.068 function make access in AL-3202 digital outputs modules, implementing special matches for PLCs in remote terminals units (UTRs).

AL-3202 module has 32 digitals outputs and work with “check before operate” principle. Its points can be configured to work like common outputs - accessed per %S operands through I/O scanning, double-stability, “trip/close” or “rise/lower” outputs.

F-UTR_S.068 function receive commands to activate supervisory station through a table. Send commands to AL-3202 output modules, read the operations states and share commands in seekings. To this operation, AL-3202 modules must be configured as double-stability, “trip/close”, “trip/close SBO” or “rise/lower”.

F-UTR_S.068 function must be used only in one program logic, the control of all AL-3202 PLC modules is done by one call.

F-CBO.018 function must be used to configure each AL-3202 module existent in bus.



Programming

Operands

CHF instruction cells are used for function calls and programmed like follow:

- **OPER1** - Specifies the number of parameter passed to the function in OPER3. This operand must be a memory constant with value 1 (%KM+00001).
- **OPER2** - Should be an operand of type memory constant with value 0 (%KM+00000). Determines the number of parameter possible to be programmed in the editing window of OPER4. As this function does not need any parameter in OPER4, the value of OPER2 is zeroed.
- **OPER3** - Contains the parameters passed to the function, declared through a window visualised in MasterTool when the CHF instruction is edited. The number of editable parameters is specified in OPER1, being set at one for this function:
 - **TMXXXX** - Table where the supervisory write commands for outputs and read the respective state. Each function interprets existents commands in its first 3 positions storing consequent operation in the follow positions, accordant the format shown in this chapt. Case the table has less than 7 positions, the commands isn't interpreted by function and its "operands error" is switched on.
- **OPER4** - Not used.

Inputs and Outputs

Description of the inputs:

- **enable** - when this input get power then function is called and analyse the parameters programmed in CHF instruction. If they are correct only success output is switched on. If any are incorrect, the success and operands error output are powered.

Description of the outputs:

- **success** - is powered when function get call (is present in the controller)
- **operands error** - is powered if occur operands specification error or if tried access operands not declared.



- **command error** - it is powered if command was rejected by AL-3202 module. In this case you must to analyze %TMXXXX table state fields to discover error reasons.

ATTENTION:

This output only get power by scanning (pulse).

Use

This function can be used only in AL-2002 and AL-2003 CPUs

Function processing

The Supervisory program write in table positions 0 to 2 declared in function, commands to action. The positions 3 to 5 needing to be read only if happen one error, if error command output get active.

The %TMXXXX table must be initialized with zeroes by CAB instruction in the first CPU scanning.

Table Format %TMXXXX

0	Command	Supervisory write
1	Address	
2	Time	
3	State 0	Supervisory can read
4	State 1	
5	State 2	
6	<function reserve>	

Table fields description:

Position 0 - supervisory command:



Code	Command
1	Turn on the double-stable point
3	Turn off the double-stable point
5	Rise
7	Lower
9	Trip
11	Close
13	Select trip
15	Select close
17	Operate trip
19	Operate close
21	Cancel

Supervisory make command match through shown codes in table above. The function switch off the 0 bit from 0 position after command executing. See format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	C	C	C	C	A

A = 1 - activate command

0 - executed command

CCCC - command code

Position 1 - actioned point address:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	R	R	R	R	0	o	o	P	P	P

RRRR - bus module position (0-15)

oo - module octet number (0-3)

PPP - Number of the point to action octet (0-7 for double-stability commands, 0-3 to other ones)

The address is useless for command 21.



Position 2 - "rise/lower" actioning time (1 to 255 seconds tenth)

This time only is needed in 5 and 7 commands.

Position 3 - command state #0:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Er	x	x	x	x	Se	x	In	To	Oc	Sa	Co	Sc	Fo	Nc	x

Er - any error

Se - Invalid Operate/Cancel

In - Inactive AL-3202

To - Time out Select

Oc - Busy Module

Sa - output Error

Co - Invalid Command

Sc - Operate/Cancel Command after configuration

Fo - switched off 24V energy source

Nc - Unconfigured Module

x - Reserved

Position 4 - command state #1:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<----- octet 3 ----->								<----- octet 2 ----->							
7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0



Position 5 - command state #2:

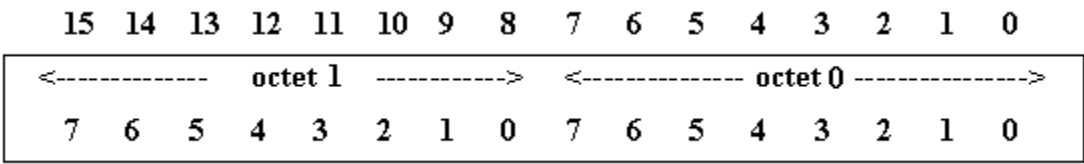


Table 4 and 5 positions has each output points state:

- "1" mean output point is damaged
- "0" mean point is OK.

Only is needed to look this positions if outputs error occur (bit Sa in #0 state)

Position 6 - reserved to function internal use.

Configuring

AL-3202 module must be configured as double-stable commands, "rise/lower" or "trip/close" to acceptance. This configuration is made module to module by F-CBO.018 function.

Configuring codes:

- | | |
|---|----------------------------------|
| 0 | S operands access (I/O scanning) |
| 1 | double- stability mode |
| 2 | "rise/lower" mode |
| 3 | "trip/close" mode |
| 4 | "trip/close" SBO mode |

Exception Situations

CBO AL-3202 output module has complex programming. It require right arquitetura understanding to comprehend exceptions situations.



Just one F-UTR_S call function can access all AL-3202 PLC boards. It does state report that has relation with board address (TMXXXX position 1).

The bits error between 1 to 7 are generated by hardware (AL-3202) besides they are specifics to one card.

Bits 8 to 10 and 15 are generated by software (CPU), corresponding to one situation that involves all the PLC.

Some bits are "latch" type, it mean error remain stored until new command is executed.

Bellow is described errors bits behaviour:

- **Er - any error** - The Er bit indicates any error situation. The other bits serve to detail the kind of occurred error. Following Er bit the second function output is turned on, and can be tested directly by application.
- **If - invalid Operate/Cancel** - Error that occur when you try to do "operate" or "cancel" command without executed previous "select" command, or the point addressed by "operate" isn't the same of selected (different address), or if selection change to "trip" having "operate" as reverse close.
- **In - AL-3202 inactive** - Occur when CPU can't access AL-3202 module: the swap key is STDBY, the AL-3202 module is removed from embroidery frame, the AL-3202 module stay without error, wrong address (RRRR field in %TMXXXX position 1), or module isn't declared in Bus.
- **To - Time out selection (latch)** - Occur when "operate" or "cancel" command arrive to AL-3202 module after selected time expired. This time must be specified in module configuration, between 0,1 to 25,5 seconds (see F-CBO.018 description).
- **Oc - Busy Module** - The Oc bit stay turned on during command execution. The Oc bit is considered error if another command was actioned in last execution. In this case, the Er bit also take switch on. Is normal Oc bit appear with power after t/c, r/l or SBO t/c command.
- **Sa - outputs Error** - This bit indicate that AL-3202 module has one or more outputs hardware damaged. You must analyze positions %TMXXXX 4 and 5 to know what output is damaged.
- **Co - Invalid Command (latch)** - AL-3202 octet addressed doesn't configured according the sent command (see F-CBO.018 - Configuration).
- **Sc - Operate/Cancel Command after configuration (latch)** - This error happen if the operate/cancel command was sent to AL-3202 after configuring command. This action can happen if input 1 F-CBO function



stay powered while SELECT and OPERATE/CANCEL execution sequence (see description of F-CBO.018).

- **Fo - 24V energy source off** - This bit tell that AL-3202 24V energy source isn't active, or tell that cable to streaming data between two AL-3202 modules isn't conected. This error can be reverted after problem correction and AL-3202 module reconfiguration, if powered like inputs 0, 1, e 2 of F-CBO (see F-CBO.018 description).
- **Nc - not configured** - Indicate module didn't receive configuration after take change, after take source energy or after PLC has changed from "programming" to "executing". It can happen in module swap, case F-CBO function didn't active (see F-CBO.018 description).

Applications example

Going ahead is presented some activation commands points examples.

In left side is placed table of written representation values per supervisory, before function processing (0 to 4 positions). In right side is shown the values monitorated by supervisory and after function processing.

1) Double-stable output 7 point of 1st %R0032 I/O Bus lodged module immediate activation.

Sup.--> PLC			Sup.<--PLC		
TMXXX			TMXXX		
0	1	- double-stable activation of lock contact	0	0	
1	279	- point address of (32 + 1)X8 + 7	1	279	
2	0	- not used	2	0	
3		- Operation status	3	0	
4		- Oct 3/2 output status	4	0	
5		- Oct 1/0 output status	5	0	
6		- internal function use	6		

2) Immediate activation per 1 second by "rise/lower" point having 2 outputs of 3rd module octet and that were lodged in %R0048 I/O Bus address.

Sup.--> PLC			Sup.<--PLC		
TMXXX			TMXXX		
0	7	- "lower" activation	0	6	
1	410	- (48 + 3) X8 + 2 point address	1	410	
2	100	- one second activation time	2	100	
3		- Operation status	3	0	
4		- Oct 3/2 output status	4	0	
5		- Oct 1/0 output status	5	0	
6		- internal function use	6		

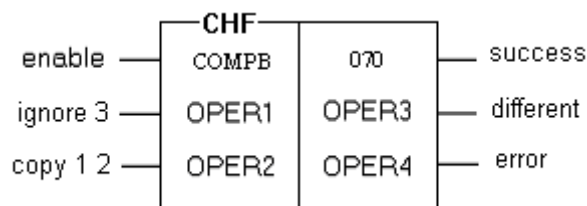
3) “trip/close” point select with 5 outputs of 0 module octet of %R0024 lodged in I/O Bus module.

Sup.--> PLC		Sup.<--PLC	
TMXXX		TMXXX	
0	13 - “trip” command selection	0	12
1	197 - (24 + 0) X8 + 5 point address	1	197
2	0 - not used	2	0
3	- Operation status	3	0
4	- Oct 3/2 output status	4	0
5	- Oct 1/0 output status	5	0
6	- internal function use	6	

4) “trip/close” point operation selected from before example.

Sup.--> PLC		Sup.<--PLC	
TMXXX		TMXXX	
0	17 - “trip” command operation	0	16
1	197 - (24 + 0) X8 + 5 point address	1	197
2	0 - not used	2	0
3	- Operation status	3	0
4	- Oct 3/2 output status	4	0
5	- Oct 1/0 output status	5	0
6	- internal function use	6	

F-COMP.B.070 – Function to Compare Operands Blocks



Introduction

The function F-COMP.B.070 compare two blocks of operands (BLC1 and BLC2), verifying if the module of the difference between the components is bigger then the value specified on a third block (BLC3). And more, it informs the relative position of the blocks that the first difference were detected.



This is useful to detect the need of transmission of messages that are not being requested, besides other needs, as the control of writes via driver AL-2730 and controllers, log of alarms and events, ...

In the case of the use to generate messages non-requested, BLC1 the actual value of a group of variables (analogicals or digitals), and BLC2 the static value of these variables since the last transmission of the messages non requested. BLC3 defines the "dead-band" of each variable of the group in the case of analogical variables, on digital variables it should be ignored.

Programming

OPER1 specifies the number of operands passed to the function, and it should be KM+00008.

OPER2 should be KM+00000.

OPER3 is the list of the 8 parameters of the function input:

- PAR1: informs the length of the blocks BLC1, BLC2 and BLC3. TAM can be informed in two ways:

Mxxxx (TAM = containing of Mxxxx)

KMxxxx (TAM = Kmxxxx)

- PAR2: first part of characterization of the block BLC1, it can be done in two ways:

Mxxxx (BLC1 = M[xxxx+OFS1]...M[xxxx+OFS1+TAM-1])

TMxxxx (BLC1 = TMxxxx[OFS1]...TMxxxx[OFS1+TAM-1])

- PAR3: informs the value of OFS1 of the block BLC1. OFS1 can be informed in two ways:

Mxxxx (OFS1 = containing of Mxxxx)

KMxxxx (OFS1 = KMxxxx)

- PAR4: first part of the characterization of the block BLC2, it can be done in two ways:

Mxxxx (BLC2 = M[xxxx+OFS2]...M[xxxx+OFS2+TAM-1])

TMxxxx (BLC2 = TMxxxx[OFS2]...TMxxxx[OFS2+TAM-1])

- PAR5: informs the value of OFS2 of the block BLC2. OFS2 can be informed in two ways:

Mxxxx (OFS2 = containing of Mxxxx)



KMxxxx (OFS2 = KMxxxx)

- PAR6: first part of the characterization of the block BLC3, it can be done in two ways:

Mxxxx (BLC3 = M[xxxx+OFS3]...M[xxxx+OFS3+TAM-1])

TMxxxx (BLC3 = TMxxxx[OFS3]...TMxxxx[OFS3+TAM-1])

- PAR7: informs the value of OFS3 of the block BLC3. OFS3 can be informed in two ways:

Mxxxx (OFS3 = containing of Mxxxx)

KMxxxx (OFS3 = KMxxxx)

- PAR8: operand that returns the relative position of the blocks BLC1 and BLC2 that the first difference occurs, or either the "n" first relative positions where the "n" first differences have occurred. This operand can be specified in two ways:

Mxxxx

TMxxxx

When an Mxxxx operand, it returns the position of the first occurrence, or the value "-1" when no difference was found.

When an TMxxxx with "T" positions, and "D" represents the number of differences between BLC1 and BLC2:

- the first D positions of TMxxxx are written with the relative positions where the differences were detected.

- the position TMxxxx[D] is filled with value "-1" to signalize the end of differences.

- if eventually D is greater than T, the algorithm is abandoned after the detection of the Tth difference, and the TMxxxx is filled with the relative positions of the T first differences.

OPER4 is an empty list.

Inputs and Outputs

The input "enable" active the execution of the function F-COMP.B.070.

The input "ignore_3" make the algorithm ignore the third block (BLC3), as if it null. In this case, it can be specified any operand in the place of PAR6 and PAR7. In this way, the function verifies if BLC1 and BLC2 are exactly equal or different, that is useful to digital variables.



The input "copy_1_2" specifies that, when the algorithm detect the need to activate the output "different", the block BLC1 will be copied over the block BLC2. This is usefull to update the last block transmited on a non-requested message (BLC2) with the values that will be transmited in function of the "different" output activation.

The output "success" is activated if the input "enable" is powered on and if the module F-COMP.B.070 is on the memory.

The output "error" indicates one of the following possibilities:

- some of the operands OPER1 or OPER2 do not have the expected value
- problems on parameters of OPER3:
 - if some of the operands on PAR1, PAR3, PAR5 or PAR8 is not of the type M or TM, or if it is not configured on the module C000. In the case of TMs, the min length must be 1.
 - if some of the operands PAR2, PAR4, PAR6 or PAR7 are not of the type M or KM, or it is not configured on module C000.
 - if the value of the parameter PAR7 is less then 1 or greater then 255.
 - if any of the blocks BLC1, BLC2 or BLC3, characterized by PAR1, PAR2, PAR3, PAR4, PAR5, PAR6 and PAR7 are not completely configured on module C000.

If the input "ignore_3" is active, PAR5 and PAR6 should not be consisted, as it will be not used. In this case, any information can be placed on PAR5 and PAR6.

The output "different" is activated if, to any position "i" the blocks (between 0 and TAM):

$$\text{abs}(\text{BLC1}[i] - \text{BLC2}[i]) > \text{BLC3}[i]$$

where "abs(x)" is the absolutvalue of "x"

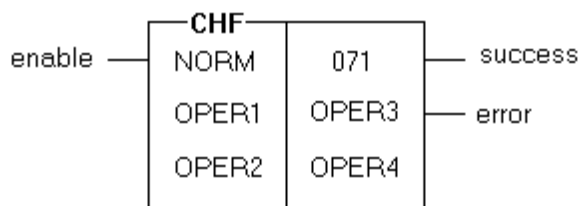
where "i" vary from 0 to TAM-1

Uses

This function can be used on the CPUs AL-2000, AL-2002, AL-2003, AL-2004, GR310, GR316, GR330, GR350, GR351, GR370, GR371, PO3045, PO3145, PO3042, PO3142, PO3242, PO3342 and QK2000.



F-NORM.071 - Function for Normalization



Introduction

The function **F-NORM.071** normalizes whole operands, implementing the function $M[\text{output}] = (M[\text{input}] - A) * C / (B - A)$, where A, B and C are constants.

Programming

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

- **OPER1** - Specifies the number of parameters passed to the function in OPER3. This operand must be a memory constant with value 6 (%KM+00006).
- **OPER2** - Should be an operand of type memory constant with value 0 (%KM+00000). It determines the number of parameters possible to be programmed in the editing window of OPER4. As this function does not need any parameter in OPER4, the value of OPER2 is 0.
- **OPER3** - Contains the parameters passed to the function, declared through a window visualised in MasterTool when the CHF instruction is edited. The number of editable parameter is specified in OPER1, being 6 for this call:
 - **%KM+XXXX** - Number of operands to process (1 to 127)
 - **%MXXXX** - Initial input operand
 - **%MXXXX** - Initial output operand
 - **%KM+XXXX** - Offset to subtract from the input operand (A)



- **%KM+XXXX** - Value reference of the input (B)
- **%KM+XXXX** - Value normalized by the output corresponding to B (C)
- **OPER4** - Not used.

Operation

A **F-NORM.071** implements the following calculation:

$$M [\text{output}] = (M [\text{input}] - A) * C / (B - A)$$

being:

M [input] - range of whole operands of input
 M [output] - range of whole operands of output
 A - offset for the input
 B - reference value of the input to normalize
 C - normalized value of the output corresponding to B

The output is the Normalization of the input and the way that for input data with the value **A** the corresponding output is **0**, and for an input value **B** the corresponding output will be **C**. If in this range, the output value will be proportional to the input, according to the formula given.

The function works with a band of up to 127 operand (1 to 127).

Inputs and Output

Description of the functions inputs:

- **enable** - when this input is powered the function is called, the parameters programmed in the CHF instruction being analysed. If they are incorrect, the output instruction error is powered, and the rest become turned off. If the parameters are correct, only the success output is powered.

Description of the functions outputs:

- **success** - indicates that the call parameters are correct and that the function has been correctly executed.
- **error** - is connected if and error occurs in the call parameters.



Use

This function can be used in CPUs AL-600, AL-2000/MSP, AL-2002, AL-2003, QK800 and QK2000.

Example

There may be a whole value originating from an A/D instruction, with variation from **0** to **4095**. It may be required to normalize the output for **0** to **100**, corresponding to input values between **800** and **4000**. We have

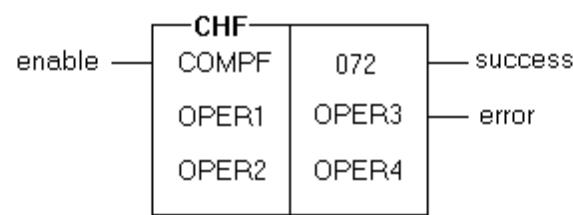
A = 800
B = 4000
C = 100

Results:

Entrada	Salida
0	-25
800	0
2400	50
4000	100
4095	102



F-COMPF.072 - Function for Multiple Comparisons



Introduction

The function **F-COMPF.072** divides an operand into specified ranges, presenting output in binary form, where the bit connected indicates the operand pertaining to the respective band.

Programming

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

- **OPER1** - Specifies the number of parameters passed to the function in OPER3. This operand must be a memory constant with value 4 (%KM+00004)
- **OPER2** - Should be an operand of type memory constant with value 0 (%KM+00000). Determines the number of parameters possible to be programmed in the editing window of OPER4. As this function does not need any parameter in OPER4, the value of OPER2 is 0.
- **OPER3** - Contains the parameters passed to the function declared through a window visualised in MasterTool when the CHF instruction is edited. The number of editable parameters is specified in OPER1, being 4 for this call:
 - **%KM+XXXX** - Number of operands %MXXXX to examine
 - **%MXXXX** - Initial input operand for comparison
 - **%MXXXX** - Initial output operand for the indicator bits

- **%TMXXXX** - Table which specifies up to 16 ranges of values to quality the input (operands/c.f. format to follow)

Table Position	Contents
0	Reserved
1	Reserved
2	Start range 0
3	End range 0
4-31	<continue the range definitions>
32	Start range 15
33	End Range 15

Table 4-23 Definition of ranges

The table should have a minimum size of 4 position (1 range). To optimise the function's execution time, it is recommended that the table is defined with the exact size to count the definitions of the necessary ranges.

- **OPER4** - Not used.

Operation

The beginning and end of each comparison range are specified as whole numbers.

The operand is considered in the range if this condition is true:
 (start of range) = %MXXXX < (end of band)

Each range in the table %TMXXXX corresponds to a bit in the operand %MXXXX being that the 0 bit of the output operand corresponds to the range 0 and so on successively. The bits correspond to the ranges not defined are always 0. The ranges can be superimposed.

The number of operands to process is given by the first parameter (%KM+XXXX), being able to be defined from 1 to 127.



Inputs and Outputs

Description of the function's inputs:

- **enable** - when this input is powered the function is called, with the parameters programmed being analysed in the CHF instruction. If they are incorrect, the error output of the instruction is powered and the rest are turned off. If the parameters are correct only the output success is powered.

Description of the function's outputs:

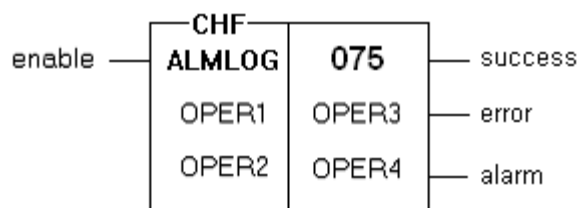
- **success** - indicates that the call parameters are correct and that the function has been correctly executed.
- **error** - is connected if an error occurs in the call parameters.

Use

This function can be used in CPUs AL-600, AL-2000/MSP, AL-2002, AL-2003, QK800, QK801, and QK2000.



F-ALMLOG.075 – Function to Logic Alarms



Introduction

The function F-ALMLOG.075 analyses a variable from 2 presets generating 1 bit to each band (alarm of high and low). The presets will determine the bands of high and low. The function allows the programming of an hysteresis to each variable too.

It can be analysed up to 255 variables on only one call of the function. The function can use simple operands (%M) or tables (%TM).

Programming

The cells of the instruction CHF used on the call are programmed as the following way:

- OPER1 – Specifies the number of parameters that are passed to the function on OPER3. This Operand should be obligatory a memory constant with value 7 (%KM+00007) or 6 (%KM+00006). The value of OPER1 is a need when the hysteresis are being programmed to the alarms, and 6 when hysteresis are not being used.

- OPER2 – It should an constant operand with value 0 (%KM+00000). Determines the number of possible parameters to be programmed on the edition window of OPER4. As this function do not need any parameter on OPER4, the value of OPER2 is 0.



- OPER3 – It contains the parameters that are passed to the function, declared through an window visualized on MasterTool when the CHF instruction is edited. The number of editable parameters is specified on OPER1:

%KM+XXXX – Number of operands or positions to analyse (1 to 255)

%MXXXX or %TMXXXX – Input Block 1 – First input operand to the analysed variables (1 to 255 operands or positions)

%MXXXX or %TMXXXX – Input Block 2 – First input operand to the definition of the presets of the high alarm (1 to 255 operands or positions)

%MXXXX or %TMXXXX – Input Block 3 – First input operand to the definition of the presets of the low alarm (1 to 255 operands or positions)

%MXXXX or %TMXXXX – Output Block 1 – First output operand to the high alarms bits (1 to 16 operands or positions). The number of operands or positions used from the first operand can be calculated dividing the number of variables by 16 + 1 if the the rest is equal to 0.

%MXXXX or %TMXXXX – Output Block 2 – First output operand to the low alarms bits (1 to 16 operands or positions). The number of operands or positions used from the first operand can be calculated dividing the number of variables by 16 + 1 if the rest is equal to 0.

%MXXXX or %TMXXXX – Input Block 4 – First input operand to the definition of hysteresiss (1 to 255 operands or positions), when OPER1 is 7.

To time optimization of the function execution, it is recommended when the table using, the right length of the table to contain programmed variables on the function.

- OPER4 – Not used.



Operation

The presets are configured with integer numbers.

The bit of high alarm is turned on when the value of the variable analysed is greater than the high preset programmed, and the bit of the alarm low is turned on when the value of the variable is lower than the low preset.

Variable analysed > preset high = bit of high alarm on 1

< preset low = bit of low alarm on 1

Hysteresis:

When the utilization of hysteresis will be programmed, the function will just turn off the alarm bits when:

Variable analysed < preset high - hysteresis = bit of high alarm on 0

> preset low + hysteresis = bit of low alarm on 0

Each alarm preset corresponds to one bit on the output operand, and the bit 0 from the first operand of the output block 1 corresponds to the high alarm of the first variable, and successively. The same occurs to the operands of the output block 2, low alarm. The bits corresponding to alarms not defined are always 0.

The number of operands to process is given by the first parameter (%KM+XXXX), and it can be defined from 1 to 255.

The programmed blocks on the function are the following:

Input 1 Input Variable	Input 2 Preset High	Input 3 Preset Low	Output 1 Alarm High	Output 2 Alarm Low	Input 4 Hysteresis
1	1	1	1	1	1
to	to	to	to	to	to
255	255	255	16	16	255



Inputs and Outputs

Description of the inputs of the function:

- **enable** – when this input is powered on the function is called, and the programmed parameters will be analysed on the instruction CHF. If the parameters are incorrect, the output error is powered on and all the other powered off. If the parameters are correct, only the output success is powered on.

Description of the outputs of the function:

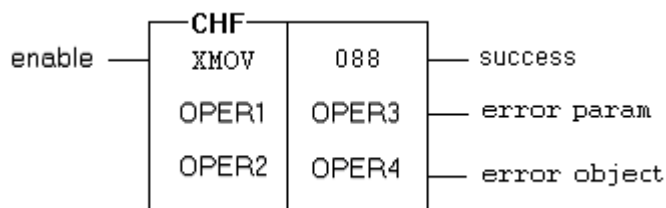
- **success** – indicates that the parameters of the function are correct and the function had been correctly executed.
- **error** – it is powered on if there is some error on the parameters of the function.
- **alarm** – it is powered on if a new alarm occurs (high or low). This output is powered on by a sweep.

Uses

This function can be used on the CLPs AL-2003, AL-2004, GR310, GR316, GR330, GR350, GR351, GR370, GR371, PO3045, PO3145, PO3042, PO3142, PO3242 and PO3342.



F-XMOV.088 – Module to Move the Data From the CPU to Memory Operands



Parameters:

OPER1

Specifies the number of parameters that are passed to the function on OPER3. This operand must be a memory constant with value 1 (KM+00001).

OPER2

Specifies the number of parameters that are passed to the function on OPER4. This operand must be a memory constant with value 0 (KM+00000).

OPER3

It contains the parameters that are passed to the function, declared through a window visualized on MasterTool programming when the instruction CHF is edited. The number of editable parameters is specified on OPER1, it is fixed on 1 to this module:

Mxxxxx or TMxxx: specification of the first memory operand, or a table memory, where the movimentations are configured, as the description below. It should exist at least two more consecutives memory operands besides the declared, or, in the case of a table, the min of 3 positions.

OPER4

Not used.



Inputs of the function

enable: when this input is powered on the function is called, and the programmed parameters will be analysed. If the number of parameters or its type are different from the needs of the function, all the outputs of the instruction are powered off.

Outputs of the function

- **success:** this output is powered on when the function is executed with success. In this case, all of the objects will be transferred to the buffer destiny of the data.
- **Error Param:** this output is powered on if some of the configuration parameters of the function are wrong: number of input parameters, type of the operands, or either, min number of operands or positions declared. This output is powered on too if the memory operand defined as buffer is not declared or if the number of objects are invalid. In this case, none of the objects is transferred to the buffer destiny of the data.
- **Error Object:** this output is powered on if some of the objects declared on the configuration operands are wrong, typically not declared operand. In this case, part of the objects can be transferred to the buffer destiny of the data.

In the case of the number of M operands declared as destiny buffer of the data is small to store the objects, the two outputs ERROR PARAM and ERROR OBJECTS are activated simultaneously. In this case, a part of the objects can be transferred to the buffer destiny of the data.

Functioning:

Each M operand of the destiny buffer of the movimentation is capable to store 2 bytes of data (high - low).

The first byte of an object always is stored from the low part of an M operand of the buffer.

The objects are stored sequentially on the buffer, from the M destiny operand specified on the configuration table.

The function does not store the data previously stored on the table if an error of object definition or the overflow of the storing buffer is found.



Configuration:

Operand (or Position)	Containing	Description of the containing
Mxxxxx	Reserved	Reserved position.
Mxxxxx + 1	Address M destiny	Address of the first memory operand of the destiny of the objects (destiny buffer of the data).
Mxxxxx + 2	Number of objects	Number of objects defined as following. Valid values: 0 to 64.
Mxxxxx + 3	First object	From this position on, the objects are defined.
Mxxxxx + 4		Each definition of the object occupy 3 operands M or positions of TM.
Mxxxxx + 5		The definitions of the objects are described following.
...		First operand/position: definition of the type of the object
	Type of the object	Second operand/position: definition of the first address
	Address	Third operand/position: definition of the quantity.
	Quantity	
Mxxxxx + N - 3	Last object	
Mxxxxx + N - 2		
Mxxxxx + N - 1		

Definition of the types of the objects:

Type of the object	Operands of Definition		Description
	Byte high	Byte low	
M Operand	0	0	Movement of values on memory operands.
		First operand	Address of the first M operand.
		Quantity	Quantity of M operands to be transferred.
D Operand (Obs. 1)	0	2	Movement of values present on decimal operands.
		First operand	Address of the first operand D.
		Quantity	Quantity of D operands to be transferred.
E/S Operand (Obs. 2)	0	8	Movement of values present on E/S operands.
		First operand	Address of the first E/S operand.
		Quantity	Quantity of E/S operands to be transferred.
A Operand (Obs. 2)	0	9	Movement of the values on A operands.
		First operand	Address of the first A operand.
		Quantity	Quantity of operands A to be transferred.
KM Constant	0	16	Movement of values on decimal operands.
		Value	Value to be transferred.
		Quantity	Quantity of constants to be transferred.
TM Operand	0	32	Movement of values on memory tables.
	First position	Table	Address of TM and the first position to be transferred.
		quantity	Quantity of positions to be transferred.
Operand TD (Obs. 1)	0	34	Movement of the values on decimal tables.
	First position	Table	Address of the TD and the first position to be transferred.
		Quantity	Quantity of positions to be transferred.

Observations:

- Each operand D or position of TD will occupy 2 M operands of the buffer:
o byte less significative of the first M operand will store the byte less



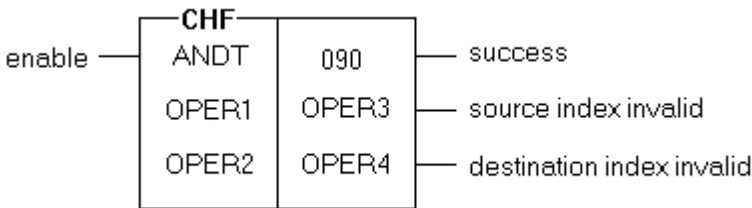
significative of the D operand., and the byte more significative of the second M operand will store the byte more significative of the D operand.

2.
- Odd number of bytes of an object are stored on ([number of bytes + 1] / 2) M operands, where the last byte will be stored on the byte less significative of the M operand, and the byte more significative should be unvalued (the high part of the M operand is not affected).

Uses

This function can be used on the CLPs AL-2003, AL-2004, GR310, GR316, GR330, GR350, GR351, GR370, GR371, PO3045, PO3145, PO3042, PO3142, PO3242 and PO3342.

F-ANDT.090, F-ORT.091 and F-XORT.092 - Functions of Logic Operations between Table Operands



Introduction

The function **F-ANDT.090**, **F-ORT.091** and **F-XORT.092** allow the carrying out of logical operations AND/and), OR (or) or XOR (or exclusive), respectively, between simple operands (M or D) and or tables (TM or TD). Up to 255 logic operations in one single function call It is necessary that the three operands (supply, supply 2 and destination) are of the same type (memory or decimal).

Programming

The cells of the CHF instruction used to call the programs in the following way:

- **OPER1** - Specifies the number of parameters passed to the function i OPER3. This operand must be a memory constant with value 3 (KM+00003).
- **OPER2** - Should be an operand of type memory constant with value 0 (KM+00000). Determine the number of parameters possible to be programmed in the editing window of OPER4. As this function does not need any parameter in OPER 4, the value of OPER is 0.



- **OPER3** - Contains the parameters passed to the function, declared through a window visualised in AL-3830 when the CHF instruction is edited. The number of editable parameters is specified in OPER1, being set at 3 for this module:
 - **MXXXX, DXXXX, TMXXXX or TDXXXX** - Simple or table operand whose values are denied (source operand).
 - **MXXXX, DXXXX, TMXXXX or TDXXXX** - Simple or table operand where the denied values are stored (destination operand).
 - **KMXXXX** - Number of simple operands or table positions to be denied. Should be less than or equal to 255.
- **OPER4** - Not used.

Inputs and Outputs

Description of the function's inputs:

- **enable** - when this input is powered the function is called, the parameters programmed in the CHF instruction being analysed in the CHF instruction. If they are incorrect, the outputs of the invalid index are actioned.

Description of the function's outputs:

- **success** - indicates that the moving was carried out correctly.
- **source index invalid** - indicates that there was an error in the specification of the supply operand:
 - the operand is not declared in the module C
 - there are not enough positions to carry out the logic
- **destination index invalid** - indicates that there was an error in the specification of the destination operand:
 - the operand is not declared in module C
 - there are not enough positions to carry out the logic

If the two outputs of the invalid index are actioned simultaneously, some of the following errors occurs:

- - the number of parameter programmed in OPER is different from three
- - the type of one the parameters in OPER3 is not valid
- - the type of destination operand is different from the source operand
- - the total number of positions to be transferred is more than 255



Use

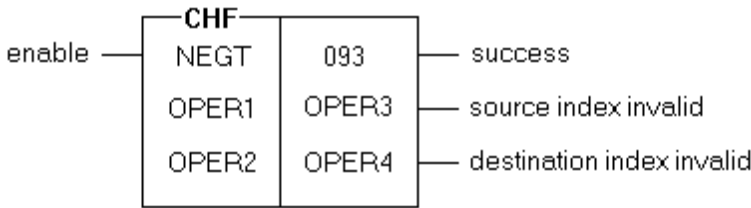
This function can be used in the CPUs AL-600, AL-2000/MSP, AL-2002/MSP, QK800 and AL-2000/MSP.

WARNING:

This function allows the denial of a large number of operands in a single scan. It should be used with care so that the maximum time of the program cycle is not exceeded.



F-NEGT.093 - Function for the logic denial of Table Operands



Introduction

The function **F-NEGT.093** carries out the logic denial of simple (M or D) or table operands (TM or TD). Up to 255 positions can be denied in one single function call. The result of the alteration can be stored in this operand, substituting the original value, or in another operand, since it may be of the same type as the first (memory or decimal).

Programming

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

- **OPER1** - Specifies the number of parameters passed to the function in OPER3. This operand must be a memory constant with value 4 (KM+00004).
- **OPER2** - Should be an operand of constant memory type with value 0 (KM+00000). It determines the number of parameters possible to be programmed in the editing window of OPER4. As these functions do not need any parameter in OPER4, the value OPER2 is 0.
- **OPER3** - Contains the parameters passed to the function, declared through a window visualised in AL-3830 when the CHF instruction is edited. The number of editable parameters is specified in OPER1, being set at 4 for these modules:

- **MXXXX, DXXXX, TMXXXX or TDXXXX** - Simple or table whose value will be used to carry out the logic (supply operand 1).
- **MXXXX, DXXXX, TMXXXX or TDXXXX** - Simple or table whose value will be used to carry out the logic (supply operand 2).
- **MXXXX, DXXXX, TMXXXX or TDXXXX** - Simple operand where the value resulting from the logic will be stored (destination operand).
- **KMXXXX** - Number of simple operands or table positions with which the logic operation will be.
- **OPER4** - Not used.

Inputs and Outputs

Description of the function's inputs:

- **enable** - when this input is powered the function is called, the parameters programmed being analysed in the CHF instruction. If they are incorrect, the outputs of the invalid index are actioned.

Description of the function's outputs:

- **success** - indicates that the moving was correctly carried out.
- **source index invalid** - indicates that there was an error in the specification of the source operand:
 - operand is not declared in module C
 - there are not enough positions to carry out the logic
- **destination index invalid** - indicates that there was an error in the specification of the destination operand:
 - the operand is not declared in module C
 - there are not enough positions to carry out the logic



If the two outputs of the invalid index are actioned simultaneously, some of the following errors occur:

- - the number of parameter programmed in OPER1 is different from four
- - the type of one or more parameters in OPER4 is not valid
- - the parameters in OPER4 are of different types (memory and decimal)
- - the total number of positions to be transferred is more than 255

Use

This function can be in the CPUs AL-600, AL-2000/MSP, AL-2002/MSP, QK800, QK801 and AL-2000/MSP.

WARNING:

These functions allow the execution of logic operations of a large number of operands in a single scan. It should be used with care so that the maximum cycle time of the program is not exceeded.



Appendix A

Execution Times of the Instruction

The execution times shown to follow are valid for the CPU AL-600, AL-600/4, AL-600/8, AL-600/16, AL-2000/MSP, AL-2002/MSP, QK600, QK800, QK801, QK2000/MSP, PL101, PL102 and PL103

The CPUs AL-3003 and AL-3004 have a processing speed 25% slower. To obtain the execution time the time presented should be multiplied by 1.25.

The CPU AL-3003/U2 has a 12.5% faster processing speed. To obtain the execution time the time presented should be multiplied by 1.25.

Description of Execution Times

The execution times of the instructions are described in a table for each instruction having the following items.

- **Header** - name of instruction or instructions for which the time measurements are.
- **Situation** - describes the situation in which the time was measured
E.g.: enabled, disabled, direct access, indirect access.
- **Execution Time** - describes execution time measured for each situation of the instruction.
E.g.: 2.4 μ s



Relays

RNA - Contact Normally Open

Situation		Execution Time
Maximum Time	(%E0000.0 to %E0015.7)	2,4 µs
Average Time	(%E0016.0 to %E0063.7) (%A0000.0 to %A0095.7)	4,8 µs
Minimum Time	(%M0000.0 to %M0127.F)	8,0 µs

RNF - Contact Normally Closed

Situation		Execution Time
Maximum Time	(%E0000.0 to %E0015.7)	2,4 µs
Average Time	(%E0016.0 to %E0063.7) (%A0000.0 to %A0095.7)	4,8 µs
Minimum Time	(%M0000.0 to %M0127.F)	8,0 µs

BOB - Simple Reels

Situation		Execution Time
Maximum Time	(%S0000.0 to %S0015.7)	2,4 µs
Average Time	(%S0001.5 to %S0063.7) (%A0000.0 to %A0095.7)	6,4 µs
Minimum Time	(%M0000.0 to %M0127.F)	9,6 µs



BBL - Reel Connected

Situation		Execution Time
Minimum Time	(%S0000.0 to %S0015.7)	4,0 μ s
Average Time	(%S0016.0 to %S0063.7) (%A0000.0 to %A0095.7)	8,0 μ s
Maximum Time	(%M0000.0 to %M0127)	11,2 μ s

BBD - Reel Disconnected

Situation		Execution Time
Minimum Time	(%S0000.0 to %S0015.7)	4,0 μ s
Average Time	(%S0016.0 to %S0063.7) (%A0000.0 to %A0095.7)	8,0 μ s
Maximum Time	(%M0000.0 to %M0127)	11,2 μ s

STL - Jump Reel

Situation		Execution Time
Disabled		26 μ s
Enabled		32 μ s

PLS - Pulse Reel

Situation		Execution Time
Disabled, Enabled		51 μ s



RM - Master Relay

Situation	Execution Time
Disabled, Enabled	26 μ s

FRM - End of Master Relay

Situation	Execution Time
Disabled, Enabled	15 μ s

Movements

MOV - Moving of Simple Operands

Situation	Execution Time
Disabled	35 μ s
Average time (direct access)	88 μ s
Maximum time (indirect access)	128 μ s

MOP - Moving of Parts of Operands

Situation	Execution Time
Disabled	35 μ s
Enabled	120 μ s



MOB - Moving of Blocks of Operands

Situation	Execution Time
Disabled	42 μ s
Moving of 8 operands %M / positions %TM for scan	365 μ s
Moving of 128 operands %M / positions %TM for scan	2400 μ s
Moving of 255 operands %M / positions %TM for scan	4600 μ s
Moving of 8 operands %D / positions %TD for scan	480 μ s
Moving of 128 operands %D / positions %TD for scan	4100 μ s
Moving of 255 operands %D / positions %TD for scan	7900 μ s

MOT - Moving of Tables

Situation	Execution Time
Disabled	38 μ s
Average time (direct access)	160 μ s
Maximum time (indirect access)	230 μ s

MES Moving of Inputs or Outputs

Situation	Execution Time
Disabled	32 μ s
Moving of 1 operand %M direct access	150 μ s
Moving of 8 operand %M direct access	470 μ s
Moving of 8 operand %M indirect access	4990 μ s



AES - Updating of Inputs or Outputs

Situation	Execution Time
Disabled	32 μ s
Updating of 1 operand %E or %S	150 μ s
Updating of 8 operand %E or %S	242 μ s

CES - Conversion of Inputs or Outputs

Situation	Execution Time
Disabled	32 μ s
Conversion from reading with direct access	260 μ s
Conversion from writing with direct access	245 μ s
Conversion from reading with indirect access	270 μ s
Conversion from writing with indirect access	260 μ s

CAB - Load Block of Operands

Situation	Execution Time
Disabled	52 μ s
Load of 8 operands %A direct access	145 μ s
Load of 8 operands %M direct access	220 μ s
Load of 8 operands %M indirect access	285 μ s



Arithmetic

SOM - Addition

Situation	Execution Time
Disabled	35 μ s
Average time (operands %M)	90 μ s
Maximum time (operands %D)	128 μ s

SUB - Subtraction

Situation	Execution Time
Disabled	35 μ s
Average time (operands %M)	110 μ s
Maximum time (operands %D)	170 μ s

MUL - Multiplication

Situation	Execution Time
Disabled	35 μ s
Enabled without overflow	130 μ s
Enabled with overflow	120 μ s



DIV - Division

Situation	Execution Time
Disabled	46 μ s
Value less than 128 in dividing	140 μ s
Value greater than 128 in dividing and less than 128 in the divider	258 μ s
Value more than 128 in dividing and in the divider	460 μ s

AND - And Binary between Operands

Situation	Execution Time
Disabled	35 μ s
Average time (operands %M)	92 μ s
Maximum time (operands %D)	110 μ s

OR - Or Binary between Operands

Situation	Execution Time
Disabled	35 μ s
Average time (operands %M)	92 μ s
Maximum time (operands %D)	110 μ s

XOR - Or Exclusive Binary between Operands Situation

Situation	Execution Time
Disabled	35 μ s
Average time (operands %M)	92 μ s
Maximum time (operands %D)	110 μ s



CAR - Load Operand

Situation	Execution Time
Disabled	34 μ s
Average time (direct access)	75 μ s
Maximum time (indirect access)	100 μ s

= - Equals

Situation	Execution Time
Disabled	34 μ s
Average time (direct access)	80 μ s
Maximum time (indirect access)	100 μ s

< - Less Than

Situation	Execution Time
Disabled	34 μ s
Average time (direct access)	95 μ s
Maximum time (indirect access)	130 μ s

> - More Than

Situation	Execution Time
Disabled	34 μ s
Average time (direct access)	95 μ s
Maximum time (indirect access)	130 μ s



Counters

CON - Simple Counter

Situation	Execution Time
Disabled	115 μ s
Average time (direct access)	120 μ s
Maximum time (indirect access)	130 μ s

COB - Bidirectional Counter

Situation	Execution Time
Disabled	170 μ s
Average time (direct access)	180 μ s
Maximum time (indirect access)	230 μ s

TEE - Timer in the Powering

Situation	Execution Time
Disabled	85 μ s
Average time (direct access)	90 μ s
Maximum time (indirect access)	110 μ s

TED - Timer in the Turning Off

Situation	Execution Time
Disabled	85 μ s
Average time (direct access)	90 μ s
Maximum time (indirect access)	110 μ s



Conversor

B/D - Conversion Binary - Decimal

Situation	Execution Time
Disabled	34 μ s
Average time (direct access)	115 μ s
Maximum time (indirect access)	155 μ s

D/B - Conversion Binary - Decimal

Situation	Execution Time
Disabled	34 μ s
Average time (direct access)	135 μ s
Maximum time (indirect access)	170 μ s

A/D - Conversion Analog - Digital

Situation	Execution Time
Disabled	34 μ s
Conversion of 1 channel AL-1103	315 μ s
Conversion of 8 channel AL-1103	1570 μ s
Conversion of 1 channel AL-1116 ou AL-1119	350 μ s
Conversion of 8 channel AL-1116 ou AL-1119	2140 μ s

D/A - Conversion Digital - Analogue

Situation	Execution Time
Disabled	34 μ s
Conversion of 1 channel AL-1103	315 μ s
Conversion of 8 channel AL-1103	1570 μ s



General

LDI - Connect or Disconnect Indexed

Situation	Execution Time
Disabled	40 μ s
Enabled	65 μ s

TEI - Test of Indexed Status

Situation	Execution Time
Disabled	40 μ s
Enabled	85 μ s

SEQ - Sequencer

Situation	Execution Time
Disabled	40 μ s
Enabled (mode AL-1000)	95 μ s



CHP - Call Procedure Module

Situation	Execution Time
Disabled	35 μ s
Enabled	138 μ s

CHF - Call Function Module

Situation	Execution Time
Disabled	45 μ s
Call of module in machine language	80 μ s
Call of module in relays language without passing parameters	240 μ s
Call of module in relays language with the passing of 4 parameters	550 μ s
Call of module in relays language with the passing of 8 parameters	710 μ s

ECR - Writing of Operands in another PLC

Situation	Execution Time
Disabled	150 μ s
Enabled with other ECR or LTR communicating (inactive)	150 μ s
First processing cycle of a communication with block of 8 operands	900 μ s
First processing cycle of a communication with block of 110 operands	2370 μ s
Enabled awaiting response	150 μ s



LTR - Reading of Operands from another PLC

Situation	Execution Time
Disabled	150 μ s
Enabled with other ECR or LTR communicating (inactive)	150 μ s
First processing cycle of a communication with any number of operands	790 μ s
Enabled awaiting response	150 μ s

LAI - Free Updating of the Operands' Images

Situation	Execution Time
Disabled	32 μ s
Enabled inactive	150 μ s
Maximum processing time of requisitions or responses to commands	2100 μ s
Maximum processing time of responses to commands without data, only confirmations	450 μ s



Appendix B

Execution Times of the Function Modules

The execution times shown here are valid for CPUs AL-600, AL-600/4, AL-600/8, AL-600/16, AL-2000/MSP, AL-2002/MSP, QK800, QK801 and QK2000/MSP.

The CPUs AL-3003 and AL-3004 have a 25% slower processing speed. To obtain the execution time the time shown should be multiplied by 1-25.

The CPU AL-3003/V2 has a 12.5% faster processing speed. To obtain the execution time the shown should be multiplied by 0-875.

Description of Execution Times

The execution times of the instructions are described in a table for each instruction or group of instructions which have the same times having the following items:

- **Header** - name of function module to which the time measures were for
- **Situation** - describes the situation in which the time was measured.
E.g. enabled, disabled, direct access, indirect access.
- **ENT0, ENT1 and ENT2** - describes the values of the inputs.
 - **x** - value is not relevant
 - **0** - input turned off
 - **1** - input powered



- **Execution Time** - describes execution time measured for each situation of the instruction.

E.g.: 2,4 μ s

F-RELOG.000

Situation	ENT0	ENT1	ENT2	Execution Time
Disabled	0	x	x	50 μ s
Reading of pulse	1	0	0	250 μ s
Reading of time	1	1	0	940 μ s
Setting of time	1	0	1	1050 μ s
Reading and setting of time	1	1	1	1050 μ s

- **ENT0** = input enabled
- **ENT1** = input read clock
- **ENT2** = input set clock

F-LEDS.001

Situation	ENT0	ENT1	ENT2	Execution Time
Disabled	0	x	x	50 μ s
LEDs test	1	0	1	2100 μ s
Initialization	1	1	0	55 μ s
Transfer of 8 octets	1	0	0	925 μ s
Transfer of 32 octets	1	0	0	2360 μ s

- **ENT0** - input enable
- **ENT1** - input initialise
- **ENT2** - input leds test



F-PTT100.002

Situation	ENT0	Execution Time
Disabled	0	50 µs
Conversion of 1 channel without linearisation	1	425 µs
Conversion of 1 channel with linearisation	1	430 µs
Conversion of 4 channels without linearisation	1	405 µs
Conversion of 4 channels with linearisation	1	420 µs

- **ENT 0** = input enable

F-TERMO.003

Situation	ENT0	Execution Time
Disabled	0	50 µs
Conversion of 1 channel without lin. in degrees	1	260 µs
Conversion of 1 channel with lin. normal	1	260 µs
Conversion of 4 channels without lin. in degrees	1	260 µs
Conversion of 4 channels with lin. normal	1	260 µs

- **ENT 0** = input enable



F-CONTR.004

Situation	ENT0	Execution Time
Disabled	0	50 μ s
Activate comparison relays	1	500 μ s
Inhibition of the counting	1	756 μ s
Write count	1	785 μ s
Read count	1	940 μ s

- **ENT 0** = input
- **ENT 1** = input
- **ENT 2** = input

F-CONT.005

Situation	ENT0	ENT1	ENT2	Execution Time
Disabled	0	x	x	50 μ s
Read count	1	0	0	360 μ s
Zero count	1	1	0	295 μ s
Load count	1	0	1	285 μ s

- **ENT0** = input enable
- **ENT1** = input zero
- **ENT2** = input load



F-ANLOG.006

Situation	ENT0	Execution Time
Disabled	0	50 µs
Functioning as A/D	1	555 µs
Functioning as D/A	1	280 µs

ENT 0 = input enable

F-EVENT.017

Situation	ENT0	ENT1	ENT2	Execution Time
Disabled	0	x	x	50 µs
Configuration	1	1	x	1100 µs
Events	1	0	1	1000 µs + 500 ms per event

- **ENT0** = input enable
- **ENT1** = input read event/configure
- **ENT2** = input read/write

F-ALNET2.032

Situation	ENT0	ENT1	Execution Time
Disabled	0	x	50 µs
Reading of values	1	0	640 µs
Initialization of values	1	1	790 µs

- **ENT0** = input enable
- **ENT1** = input initialise



F-PID.033

Situation	ENT0	ENT1	ENT2	Execution Time
Disabled	0	x	x	50 µs
Calculation of PID factors with automatic mode	1	0	x	1600 µs
Integral action inhibited with automatic mode	1	0	x	1330 µs
Derivative action inhibited with automatic mode	1	0	x	1240 µs
Adjustments I and D inhibited with automatic mode	1	0	x	950 µs
Calculation of PID factors with manual mode	1	1	x	1160 µs
Integral action inhibited with manual mode	1	1	x	470 µs
Derivative action inhibited with manual mode	1	1	x	780 µs
Adjustments I and D inhibited with manual mode	1	1	x	470 µs

- **ENT0** = input enable
- **ENT1** = input automatic/manual mode
- **ENT2** = input direct/reverse action



F-RAZN.034

Situation	ENT0	ENT1	Execution Time
Disabled	0	x	50 μ s
Root 0 to 127 without Normalization	1	0	330 μ s
Root 128 to 32767 without Normalization	1	0	440 μ s
Root 0 to 127 with Normalization	1	1	380 μ s
Root 128 to 32767 with Normalization	1	1	490 μ s

- **ENT0** = input enable
- **ENT1** = input normalize

F-ARQ2.035 to F-ARQ31.042

Situation	ENT0	ENT1	Execution Time
Disabled	0	x	50 μ s
Reading of value in file	1	0	520 μ s
Writing of value in file	1	1	430 μ s
Attempt to access invalid	1	x	190 μ s

ENT 0 = input enable

ENT 1 = input read/write



F-MOBT.043

Situation	ENT0	Execution Time
Disabled	0	50 µs
Moving of 8 operands %M positions %TM	1	595 µs
Moving of 128 operands %M positions %TM	1	2520 µs
Moving of 255 operands %M positions %TM	1	4620 µs
Moving of 8 operands %D positions %TD	1	720 µs
Moving of 128 operands %D positions %TD	1	4410 µs
Moving of 255 operands %D positions %TD	1	8360 µs

- **ENT 0** = input enable

F-STDMOD.045

Situation	ENT0	ENT1	ENT2	Execution Time
Disabled	0	x	x	50 µs
Reading of status of I/O octets from the bus	1	0	0	650 µs
Reading of module directory	1	0	1	2350 µs
Reading of module status	1	1	0	2350 µs
Reading of directory and status of modules	1	1	1	4060 µs

ENT 0 = input enable

ENT 1 = input modules status

ENT 2 = input modules directory



F-RELG.048

Situation	ENT0	ENT1	Execution Time
Disabled	0	x	50 μ s
Reading of clock	1	0	475 μ s
Setting of clock	1	1	690 μ s

- **ENT0** = input enable
- **ENT1** = input set clock

F-SINC.049

Situation	ENT0	ENT1	ENT2	Execution Time
Disabled	0	x	x	50 μ s
Reading of clock	1	0	0	475 μ s
Setting of clock in the next pulse	1	0	1	690 μ s
Setting of clock in the next second	1	1	0	690 μ s

- **ENT 0** = input enable
- **ENT 1** = input set synchronism maintained
- **ENT 2** = input set external pulse



F-ALNET1.062

Situation	ENT0	Execution Time
Disabled	0	50 µs
Configuration of second serial channel (first program cycle)	1	360 µs
Enabled without reception of command	1	300 µs
Enabled processing command for monitoring of 1 operand %M	1	330 µs
Enabled processing command for monitoring of 48 operands %D	1	1700 µs

- **ENT 0** = input enable

F-IMP.063

Situation	ENT0	Execution Time
Disabled	0	50 µs
Enabled with other transmission active	1	100 µs
Enabled waiting for end of transmission	1	105 µs
Preparing transmission of the text with 1 character (a cycle of program)	1	170 µs
Preparing transmission of the text with 100 character (a cycle of program)	1	1520 µs
Preparing transmission of the text with 255 character (a cycle of program)	1	3625 µs
Preparing transmission of 1 operand %M (a cycle of program)	1	279 µs
Preparing transmission of 100 operand %M (a cycle of program)	1	14000 µs
Preparing transmission of 255 operand %M (a cycle of program)	1	35000 µs



- **ENT0** = enable

F-RECEP.064

Situation	ENT0	Execution Time
Disabled	0	50 µs
Enabled awaiting reception	1	70 µs
Receiving characters	1	80 µs

- **ENT** = input enable



Remissive Index

—A—

A/D, 3-64
description, 3-64
syntax, 3-65
AES, 3-27
description, 3-27
example, 3-28
syntax, 3-28
AND, 3-41
description, 3-41
example, 3-42
syntax, 3-42

—B—

B/D, 3-62
description, 3-62
syntax, 3-62
BOB, BBL and BBD, **3-6**
description, 3-6

—C—

CAB, 3-29
description, 3-29
syntax, 3-33
CAR, 3-47
description, 3-47
CES, 3-25
description, 3-25
example, 3-25
syntax, 3-26
CHF, 3-84
description, 3-84
example, 3-87

syntax, 3-87
CHP, 3-82
description, 3-82
example, 3-83
syntax, 3-83
COB, 3-55
description, 3-55
syntax, 3-56
CON, 3-53
description, 3-53
syntax, 3-54

—D—

D/A, 3-66
description, 3-66
example, 3-68
syntax, 3-68
D/B, 3-63
description, 3-63
syntax, 3-63
DIV, 3-40
description, 3-40
syntax, 3-40

—E—

ECR
description, 3-88
example, 3-96
syntax, 3-96
Equals, More than and Less than, 3-48
description, 3-48
example, 3-49; 3-50
syntax, 3-51
ECR, 3-88



—F—

- F - RAIZN.034, 4-53
 - example of application, 4-55
 - inputs and outputs, 4-54
 - introduction, 4-53
 - operands, 4-53
 - programming, 4-53
 - use, 4-55
- F-ALNET1.062, 4-85
- F-ALNET2.032, 4-39
 - description of the values, 4-41
 - example of application, 4-43
 - inputs and outputs, 4-40
 - introduction, 4-39
 - operands, 4-39
 - programming, 4-39
 - use, 4-43
- F-ANDT.090, F-ORT.091 and F-XORT.092, 4-7
 - inputs and outputs, 4-8
 - introduction, 4-7
 - programming, 4-7
 - use, 4-9
- F-ANLOG.006, 4-24
 - inputs and outputs, 4-25
 - introduction, 4-24
 - operands, 4-24
 - programming, 4-24
 - use, 4-25
- F-COMPF.072, 4-4
 - inputs and outputs, 4-6
 - introduction, 4-4
 - operation, 4-5
 - programming, 4-4
 - use, 4-6
- F-CONT.005, 4-21
 - description of functioning, 4-23
 - inputs and outputs, 4-22
 - introduction, 4-21
 - operands, 4-21
 - programming, 4-21
- F-CONTR.004, 4-18
 - inputs and outputs, 4-19
 - introduction, 4-18
 - operands, 4-18
 - programming, 4-18
 - use, 4-20
- F-IMP.063, 4-90
 - introduction, 4-90
 - operands, 4-90
 - programming, 4-90
 - use, 4-92
- F-LEDS.001, 4-7
 - inputs and outputs, 4-8
 - introduction, 4-7
 - operands, 4-7
 - programming, 4-7
 - use, 4-9
- F-MOBT.043, 4-62
 - inputs and outputs, 4-63
 - introduction, 4-62
 - operands, 4-62
 - programming, 4-62
 - use, 4-64
- F-NEGT.093, 4-10
 - inputs and outputs, 4-11
 - introduction, 4-10
 - programming, 4-10
 - use, 4-12
- F-NORM.071, 4-1
 - example, 4-3
 - inputs and output, 4-2
 - introduction, 4-1
 - operation, 4-2
 - programming, 4-1
 - use, 4-3
- F-PID.033, 4-45
 - additional parameters, 4-48
 - characteristics of functioning, 4-51
 - example of application, 4-51
 - inputs and outputs, 4-48
 - introduction, 4-45
 - operands, 4-47
 - programming, 4-47
 - use, 4-52
- F-PT100.002, 4-10
 - inputs and outputs, 4-13
 - introduction, 4-10
 - operands, 4-10
 - programming, 4-10
 - use, 4-13
- FR-ARQ2.035, 4-56
 - description of the functioning, 4-58
 - example of application, 4-61
 - inputs and output, 4-58
 - introduction, 4-56
 - operands, 4-57
 - programming, 4-57



use, 4-58
F-RECEP.064, 4-94
introduction, 4-94
operands, 4-94
programming, 4-94
use, 4-96
F-RELG.048, 4-71
example, 4-74
inputs and outputs, 4-73
introduction, 4-71
operands, 4-71
programming, 4-71
use, 4-74
F-RELOG.000
inputs and outputs, 4-5
introduction, 4-4
operands, 4-4
programming, 4-4
use, 4-6
F-RELOG.061, 4-81
ALNET I protocol, 4-87
input and outputs, 4-86
inputs and outputs, 4-83
introduction, 4-81; 4-85
operands, 4-81; 4-85
programming, 4-81; 4-85
use, 4-84; 4-89
F-SINC.049, 4-75
inputs and outputs, 4-77
introduction, 4-75
operands, 4-75
programming, 4-75
use, 4-80
F-STMOD.045, 4-65
inputs and outputs, 4-70
introduction, 4-65
operands, 4-65
programming, 4-65
use, 4-70
F-TERM0.003, 4-14
input and outputs, 4-16
introduction, 4-14
operands, 4-14
programming, 4-14
use, 4-17
Function Modules, 4-1

—|—

Instructions, 3-1
arithmetic, 3-34
coils, 3-6
comparison of operands, 3-48
connection, 3-101
contacts, 3-5
converter, 3-61
counters, 3-52
general, 3-69
list of instructions, 3-1
relays group, 3-2

—L—

LAI, 3-100
description, 3-100
LDI, 3-70
description, 3-70
example, 3-71
syntax, 3-72
LGH, LGN and LGV, 3-101
description, 3-101
LTR, 3-98
description, 3-98
example, 3-99
syntax, 3-99

—M—

MES, 3-23
description, 3-23
syntax, 3-24
MOB, 3-18
description, 3-18
syntax, 3-19
MOP, 3-15
description, 3-15
example, 3-15
syntax, 3-17
MOT, 3-20
description, 3-20
syntax, 3-22
MOV, 3-13
description, 3-13
syntax, 3-14
MUL, 3-39
description, 3-39
syntax, 3-39



—O—

OR, 3-43
description, 3-43
example, 3-44
syntax, 3-44

—P—

PLS, 3-10
description, 3-3
syntax, 3-4
description, 3-10
syntax, 3-10
PLS , 3-3
Programming Project, 2-32
depuration of programming projects, 2-51
elaboration of programming projects, 2-42
execution of the programming project, 2-41
module C - configuration, 2-33
module E - execution, 2-35
module F - function, 2-37
module P - procedure, 2-37
operation status of the PLC, 2-39
program execution cycle times, 2-63
structure, 2-32

—R—

RM, FRM, 3-11
description, 3-11
RNA and RNF
description, 3-5
RNA e RNF, 3-5
syntax, 3-6
Router Project, 2-70
building up a router project, 2-70
module R, 2-71

operation statues of the router, 2-72

—S—

SEQ, 3-75
description, 3-75
example, 3-76; 3-79
syntax, 3-81
SLT, 3-8
description, 3-8
example, 3-8
syntax, 3-9
SOM, 3-35
description, 3-35
syntax, 3-36
SUB, 3-37
description, 3-37
syntax, 3-38

—T—

TED, 3-59
description, 3-59
diagram of times, 3-60
syntax, 3-60
TEE, 3-57
description, 3-57
diagram of times, 3-58
syntax, 3-58
TEI, 3-73
description, 3-73
syntax, 3-74

—X—

XOR, 3-45
description, 3-45
example, 3-46
syntax, 3-46

