



---

# NUM

## AUTOMATIC CONTROL FUNCTION PROGRAMMING MANUAL LADDER LANGUAGE

0101938846/8

Despite the care taken in the preparation of this document, NUM cannot guarantee the accuracy of the information it contains and cannot be held responsible for any errors therein, nor for any damage which might result from the use or application of the document.

The physical, technical and functional characteristics of the hardware and software products and the services described in this document are subject to modification and cannot under any circumstances be regarded as contractually binding.

The programming examples described in this manual are intended for guidance only. They must be specially adapted to the automated system used and the safety levels required before they can be used in programs with an industrial application.

© Copyright NUM 1998.

All rights reserved. No part of this manual may be copied or reproduced in any form or by any means whatsoever, including photographic or magnetic processes. The transcription on an electronic machine of all or part of the contents is forbidden.

© Copyright NUM 1998 NUM 1020/1040/1060 CNC software.

This software is the property of NUM. Each electronic copy of this software sold confers upon the purchaser a non-exclusive licence strictly limited to the use of the said copy. No copy or other form of duplication of this product is authorised.

---

# Table of Contents

<b>1 Presentation of the Automatic Control Function</b>	1 - 1
1.1 General	1 - 3
1.2 Automatic Control Function	1 - 6
<b>2 Structure of an Application</b>	2 - 1
2.1 General	2 - 3
2.2 Structure of an Application	2 - 13
2.3 Structure of a Ladder Module - Elementary Sequences	2 - 15
2.4 Elements Common to All Types of Sequences	2 - 15
2.5 Table of Constants Sequence	2 - 15
2.6 Character String Sequence	2 - 16
2.7 Ladder Sequence	2 - 16
<b>3 Variables</b>	3 - 1
3.1 Principle of Exchanges	3 - 5
3.2 Variable % - Mnemonic	3 - 6
3.3 Variable %	3 - 6
3.4 Mnemonic	3 - 8
3.5 Common Internal Variables Saved	3 - 8
3.6 Common Internal Variables Not Saved	3 - 8
3.7 I/O Card Interface Variables %I and %Q	3 - 9
3.8 CNC I/O Interface Family %R and %W	3 - 29
3.9 %S Common Word Variables	3 - 68
3.10 %Y Local Variables- Pointers	3 - 70
3.11 Exchange Area	3 - 72
<b>4 Literal Elements of Ladder Language</b>	4 - 1
4.1 Notations Used	4 - 3
4.2 Label - Comment	4 - 3
4.3 Step	4 - 3
4.4 Literal Elements of Ladder Sequences	4 - 3
4.5 Additional Information on Literal Elements	4 - 5
<b>5 Programming in Ladder Language</b>	5 - 1
5.1 Elements Common to All Types of Sequence	5 - 3
5.2 Network Sequence	5 - 7
5.3 Function Calls	5 - 26
5.4 Parameter Check	5 - 26
<b>6 General Purpose Functions</b>	6 - 1
6.1 Convert an ASCII String to a Signed Integer of 32 Bits	6 - 3
6.2 Convert an ASCII String to a Signed Integer of 32 Bits	6 - 4
6.3 BCD → Binary Conversion	6 - 5
6.4 Binary → BCD Code Conversion	6 - 6
6.5 Separate Bits into Bytes	6 - 7
6.6 Read the Parameters Stored on the Stack	6 - 8
6.7 Copy One or More Bytes	6 - 9

6.8	Copy One or More Words	6 - 10
6.9	Copy One or More Long Words	6 - 11
6.10	Set Self-Test Period	6 - 11
6.11	Convert a Signed Integer to an ASCII String	6 - 12
6.12	Convert an Unsigned Integer to an ASCII String	6 - 12
6.13	Concatenate Bytes into Bits	6 - 13
6.14	Simulate Operator Panel Keyboard	6 - 15
6.15	Shortest Path Calculation	6 - 15
6.16	Search for the Value of a Byte	6 - 16
6.17	Search for the Value of a Word	6 - 16
6.18	Search for the Value of a Long Word	6 - 17
6.19	Return to Calling Module or Network	6 - 18
6.20	Jump to a Module Label without Return	6 - 19
6.21	Jump to a Module Label with Return	6 - 19
6.22	Flag	6 - 20
6.23	Set One or More Bytes	6 - 20
6.24	Set One or More Words	6 - 21
6.25	Set One or More Long Words	6 - 22
6.26	Call %SP Modules	6 - 22
6.27	Format a Character String	6 - 24
6.28	Integer Square Root	6 - 25
6.29	Analyse an ASCII String	6 - 25
6.30	Compare Two Character Strings	6 - 26
6.31	Copy a Character String	6 - 27
6.32	Calculate String Length	6 - 27
6.33	Swap the Even and Odd Bytes of a Word	6 - 28
6.34	Swap the Four Bytes of a Long Word	6 - 29
6.35	Change Tool Wear Offset	6 - 30
6.36	Read n Variables E42000	6 - 31
6.37	Write n Variables E42000	6 - 32
6.38	Initialise the Base Associated with the %Y Variables	6 - 33
<b>7 Task Management</b>		<b>7 - 1</b>
7.1	Introduction	7 - 3
7.2	Start a Critical Section	7 - 3
7.3	End a Critical Section	7 - 3
7.4	Suspend a %TF task	7 - 3
7.5	Start a %TF Task	7 - 4
7.6	Stop a %TF Task	7 - 4
<b>8 Transparent Mode</b>		<b>8 - 1</b>
8.1	Introduction	8 - 3
8.2	Functions Assigned to transparent Mode	8 - 7
8.3	Panel Transparent Mode	8 - 18

<b>9 Analogue Inputs/Outputs</b>	9 - 1
9.1 General	9 - 3
9.2 Configure an Analogue I/O Card	9 - 3
9.3 Write an Analogue Output	9 - 5
9.4 Read an Analogue Input	9 - 6
9.5 Reassign an Analogue Card	9 - 7
<b>10 Explicit Read/Write of Input/Output Cards</b>	10 - 1
10.1 General	10 - 3
10.2 Explicit Read of an Input Card	10 - 3
10.3 Explicit Write to an Output Card	10 - 4
<b>11 Interrupt Inputs</b>	11 - 1
11.1 General	11 - 3
11.2 Principle of Line Assignment	11 - 5
11.3 Associate an Interrupt Input with Axis Groups	11 - 5
11.4 Configure an Interrupt Input	11 - 6
11.5 Read an Interrupt Input	11 - 8
11.6 Associate a %TH Task with an IT Input	11 - 9
<b>12 Serial Lines</b>	12 - 1
12.1 General	12 - 3
12.2 Select Data Rate and Format	12 - 4
12.3 Send a Buffer	12 - 6
12.4 Reception of a Buffer	12 - 7
12.5 Read the Status of a Serial Line	12 - 10
12.6 Control the Serial Line Driver	12 - 11
12.7 Transmission Standards	12 - 12
<b>13 Timer Function</b>	13 - 1
13.1 General Description of the Timer Function	13 - 1
13.2 Use of Timer A	13 - 1
13.3 Associate a %TH Task with a Timer	13 - 1
<b>14 Date-Time Stamp Function</b>	14 - 1
14.1 General Description of the Date-Time Stamp Function	14 - 1
14.2 Read the Current Date tmget	14 - 1
14.3 Read the Current Date and Day in Week	14 - 2
<b>15 Exchanges by Protocol</b>	15 - 1
15.1 General Description of Exchanges	15 - 3
15.2 Objects Accessible by a UNITE Request	15 - 7
15.3 UNITE Requests Processed by the CNC Function	15 - 16
15.4 Programming the General Request Function	15 - 29
15.5 Exchanges With a Remote Station	15 - 34
<b>16 Programming in C Language</b>	16 - 1
16.1 General	16 - 3
16.2 Call Executable Module	16 - 3
16.3 Identify an Executable Module	16 - 4
16.4 Programming in C	16 - 5

<b>17 PLC Axes</b>		17 - 1
17.1	General	17 - 1
17.2	Programming Principle	17 - 1
<b>18 Programme Debugging</b>		18 - 1
18.1	Programme Debugging with the PLCTOOL Software Workshop	18 - 3
18.2	Debugging on the CNC	18 - 3
<b>19 Errors and Diagnostic</b>		19 - 1
19.1	List of Hardware Errors	19 - 1
19.2	List of Configuration Errors	19 - 1
19.3	List of Programming Errors	19 - 1
<b>A Lists of Functions</b>		A - 1
A.1	List by Themes	A - 3
A.2	Alphanumeric List	A - 6
<b>Index</b>		I - 1

# Record of Revisions

Date	Revision	Pages revised	Pages added	Pages deleted
01 - 98	8	Title page, 2, 3, 7, 10 Ch. 2: 13 Ch. 3: 1-4, 17, 27, 34, 37-78 Ch. 5: 13 Ch. 8: 3, 6 Ch. 9: 3 Ch. 12: 5 Ch. 15: 8, 25 Ch. 17: 1 Index: 1-4 Agencies Questionnaire	79-82	

## DOCUMENT REVISIONS

Date	Revision	Reason for revisions
07 - 92	0	Conforming to NUM 1060 software - Index D Document creation
10 - 92	1	Conforming to NUM 1060 software - Index D Miscellaneous corrections Variable %R1.B changed to %R0.W. Variable %Rg1F.B changed to %Rg1E.W. Variable %Rg7E.W changed to %Rg7C.L. Added variables %R2.7, %R2.6, %W3.7, %W3.6 Deleted variables %W15.B and %W16.B. Added functions call(), goto(), R_E42000(), W_E42000(). Modified character table in transparent mode. Procedures of utility 7. List of functions in appendix.
04 - 93	2	Conforming to NUM 1060 software - Index E Miscellaneous corrections Addition of variables - %S common words - %Y local variables - %Qrc3B.1 NC access authorisation - %R2.5 E_INTERV cycle hold - %W5.6 INIB_E33 Input/Output card write enable by part programme

Date	Revision	Reason for revisions
04 - 93	2	<ul style="list-style-type: none"> <li>- %W4.4 PRESPUIS Motor power on</li> <li>- %W15.B MSG1 Message number to be displayed on line 1</li> <li>- %W16.B MSG2 Message number to be displayed on line 2</li> <li>- %W2C.W List of bits - JOG increments inhibited</li> <li>- %W30.L List of bits - Modes inhibited</li> <li>- %Rg01.0 E_RAZ1 to E_RAZ8 Group g being reset</li> <li>- %Rg01.4 E_DGURG1 to E_DGURG8 Emergency retraction on group g</li> <li>- %Wg01.4 C_DGURG1 to C_DGURG8 Emergency retraction request for group g</li> </ul> <p>Test contact high for a list of bits      Test contact low for a list of bits      Test contact on rising edge      Test contact on falling edge      Conditional actions in test area      Multiple numerical assignments on T and F coils      Subroutine call with %Y local variables - Function spy()      Initialisation of the basis associated with %Y variables - Function y_init()      Graphic initialisation - Function inig()      Sending of request to a distance server - Function neto()      Read of a request from a distance server - Function neti()      Common word service setup - Function setcomw()      Answer to a STATUS request - Function netst_ad()      NUM library C programming function (NUM.OBJ)      Programme storage using UT7</p>
02 - 94	3	<p>Conforming to NUM 1060 software - Index F      Miscellaneous corrections and additions      Inclusion of the UCSII module (CPU time, hardware restrictions, etc.)      Mnemonic on 12 characters      Deletion of %R3.5, E_STOP      Addition of variables:</p> <ul style="list-style-type: none"> <li>- %R19.B, ID_ICB_CN, identifying whether the operator panel or CNC is active</li> <li>- %W2.0 KB_INIT, keyboard initialisation</li> <li>- %W4.6 INIBUTIL, utility inhibit</li> <li>- %W5.6 changed to SK_DISP, display of the softkey bar window</li> <li>- %W5.7, SC_SAVE, places the CNC screen on standby</li> <li>- %Rg01.5 NO_POS1 to NO_POS8, axis awaiting positioning</li> <li>- %Wg00.6 C_FAST1 to C_FAST8, operation at high speed during the cycle</li> <li>- %W900.0, INIB_E33, enables write of the output cards by part programming</li> </ul> <p>Indirect addressing and addressing by pointer (%Y variable)      Counters, CTU_n, CTD_n      Timeouts, TOF_n, TON_n, TP_n      32-24 I/O and 64-48 I/O cards      Deletion of the message() function      Unsolicited data:</p> <ul style="list-style-type: none"> <li>- \$1, nonblocking message</li> <li>- \$11, blocking message</li> </ul> <p>Serial lines: Inclusion of standards RS232, RS485, RS422</p>

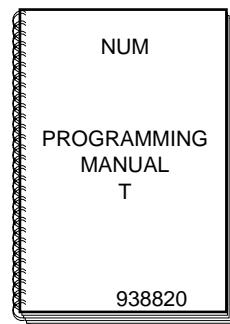
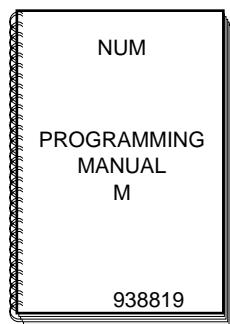
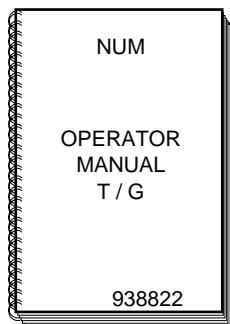
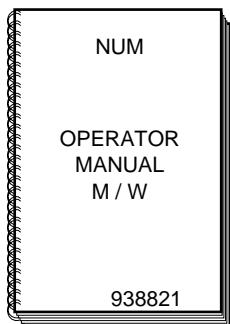
Date	Revision	Reason for revisions
08 - 94	4	<p>Conforming to NUM 1060 software - Index G.</p> <p>Miscellaneous corrections.</p> <p>Addition of variables:</p> <ul style="list-style-type: none"> <li>- %R2.1, E_NMAUTO, N/M functionality,</li> <li>- %W2.1, C_NMAUTO, N/M functionality,</li> <li>- %W34.0 to %W37.7, DISC_TRQx, Torque enable on the x axis,</li> <li>- %W38.0, DISC_SDP, Speed reference enable on QDD axes,</li> <li>- %R350.B to %R976.W, Monitor and %TS task time profile,</li> <li>- %W97a.L, Task type and number in Ladder animation,</li> <li>- %W97e.B, Number of the component to be animated.</li> </ul> <p>New programming on 32-input/32-output cards.</p> <p>Monitor and %TS task time profile under UT7.</p> <p>Ladder grid animation under UT7.</p>
04 - 95	5	<p>Conforming to NUM 1060 software - Index H</p> <p>Miscellaneous corrections</p> <p>Duplicated watchdog</p> <p>Added variables:</p> <ul style="list-style-type: none"> <li>- %R2.4, STATETRACE, backoff/return to path state</li> <li>- %R14.0, SC_USED, screen enabled in PCNC configuration</li> <li>- %R22.0 to %R22.3, STOPBR1 to STOPBR4, spindle 1 to 4 stop request</li> <li>- %R39.0, BACKWARD, backward movement requested on path</li> <li>- %R39.1, FORWARD, forward movement requested on path</li> <li>- %R39.2, INITPOS, automatic recall after maintenance.</li> </ul> <p>Reset of saved variables.</p>
11 - 95	6	<p>Conforming to NUM 1020/1040/1060 software - Index J</p> <p>Miscellaneous corrections</p> <p>Added general purpose functions:</p> <ul style="list-style-type: none"> <li>- BCD → binary code conversion</li> <li>- Binary → BCD code conversion</li> </ul> <p>Added compact panel in Chapter 3</p> <p>Added segment 235</p> <p>Added variables:</p> <ul style="list-style-type: none"> <li>- %R12.4 to %R12.7, Bx_arr, Spindle stopped</li> <li>- %R12.0 to %12.3, Bx_ROT, Spindle rotation correct</li> <li>- %R24.L, AXBLKx, Axis clamp</li> <li>- %W3A.L, STOPAXx, Feed stop per axis</li> </ul>
09 - 96	7	<p>Conforming to NUM 1020/1040/1060 software - Index K</p> <p>Miscellaneous corrections</p> <p>Added variables:</p> <ul style="list-style-type: none"> <li>- %R14.1, E_BAT, Battery status</li> <li>- %W2.2, C_INDG, Switchover between common groups/independent groups</li> <li>- %W2.3, CHG_OPDC, Dynamic operators</li> <li>- %W22.4 to %W22.7, VERBRb, Spindle b power on or off</li> <li>- %Rg00.0, E_PROG1 to E_PROG8, Active program on CNC axis group g</li> <li>- %Rg00.5, E_INTER1 to E_INTER8, Intervention state on CNC axis group g</li> <li>- %Rg00.6, E_SLASH1 to E_SLASH8, Block skip enabled on CNC axis group g</li> <li>- %Rg00.7, E_M011 to E_M018, Optional stop enabled on CNC axis group g</li> <li>- %Rg01.1, E_ARUS1 to E_ARUS8, Cycle stop on CNC axis group g</li> </ul>

Date	Revision	Reason for revisions
09 - 96	7	<ul style="list-style-type: none"><li>- %Rg01.3, E_RAX1 to E_RAX8, Axis recall on CNC axis group g</li><li>- %Rg01.7, E_OPER1 to E_OPER8, Programme stop by M00 or M01 enabled</li><li>- %Rg06.B, MODCOUR1 to MODCOUR8, Current mode on CNC axis group g</li><li>- %Wg01.1, C_ARUS1 to C_ARUS8, Request cycle stop on CNC axis group g</li><li>- %Wg01.3, C_RAX1 to C_RAX8, Select axis recall on CNC axis group g</li><li>- %Wg01.6, C_SLASH1 to C_SLASH8, Enable block skip on CNC axis group g</li><li>- %Wg01.7, C_M011 to C_M018, Enable optional programme stop (M01) on axis group g</li><li>- %WE00.B to %WE1F.B, RDUC_TRQ0 to RDUC_TRQ31, Current reduction</li></ul> <p>Added DTGET to the date-time stamp function Added user code disassembly messages to the section "Programme Debugging, CPU Command"</p>
01 - 98	8	<p>Miscellaneous corrections</p> <ul style="list-style-type: none"><li>- %Wg03.B, Independent group mode</li><li>- PLC &lt;→ CNC exchange area related to AN96 1050 function</li></ul>

## Structure of the NUM 1020/1040/1060 Product Documentation

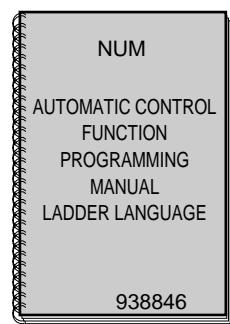
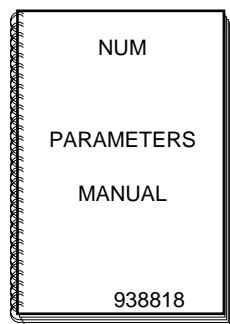
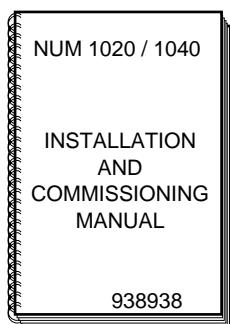
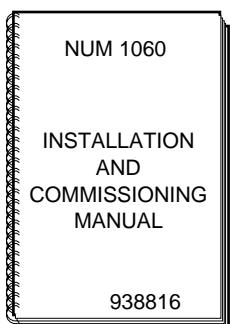
### User Documentation

These documents are designed for use of the CNC.

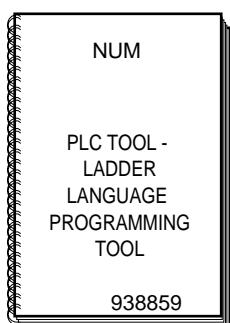


### Integrator Documents

These documents are designed for setting up the CNC on a machine.



### Special Programming Documents



## List of NUM 1020/1040/1060 Product Utilities

A series of utilities are available for the NUM products for integration and use of the system.

These utilities may be included in the basic version or available as options.

Depending on the function performed by each utility, its use is described in the integration manual or operator manual, as appropriate.

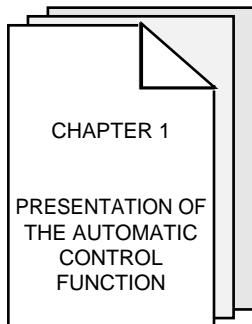
The table below lists the utilities and gives the references of the document describing them:

Utility	Name	Manual	Chapter	Application
UT0	utility management	operator manuals	8	NUM 1020/1040/1060
UT2	axis calibration	installation and commissioning manual (1020/1040 or 1060)	10 11	NUM 1020/1040 NUM 1060
UT3	resident macros	operator manuals	8	NUM 1020/1040/1060
UT5	parameter integration	parameter manual	12	NUM 1020/1040/1060
UT7	programme debugging	automatic control function programming manual - Ladder language	16	NUM 1020/1040/1060 programming in ladder language
UT12	option locking	operator manuals	8	NUM 1020/1040/1060
UT20	interaxis calibration	installation and commissioning manual (1020/1040 or 1060)	11 12	NUM 1020/1040 NUM 1060
UT22	axis parameter integration	SET_TOOL manual	8	NUM 1060

*REMARK      Utility 22 is no longer used starting from CNC software index K and SET\_TOOL software index E.*

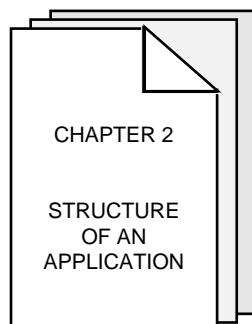
## Automatic Control Function Programming Manual

Programming of the automatic control function in Ladder language. It covers the automatic functions using the sensors and actuators located on the machine and the interfacing data with the CNC processor.



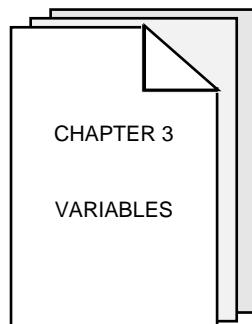
Presentation and characteristics of the automatic control function and CPU

- Block diagrams of the system and cards involved.



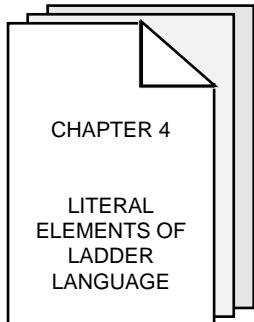
Theory of operation and organisation of a PLC application.

- System tasks.
- User tasks.
- Structure of an application.
- Modules.



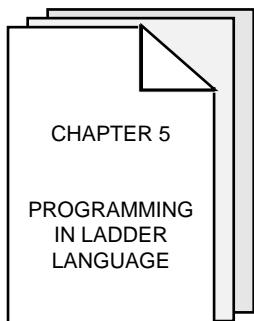
Detail of the variables involved.

- Internal variables.
- Terminal board input/output variables.
- Configuration and diagnostic variables.
- CNC interface variables.
- Common word variables.
- Local variables.



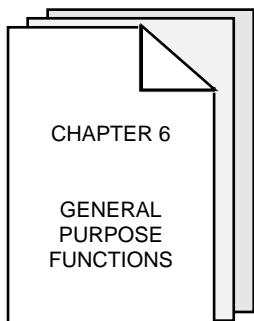
Information on the literal elements of the ladder language.

- Literal elements.
- Operators.
- Examples of computations.



Information on Ladder programming.

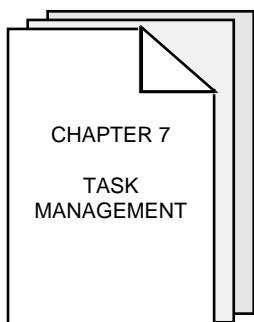
- Common elements.
- Grafset steps.
- Network sequence.
- Programming advice.



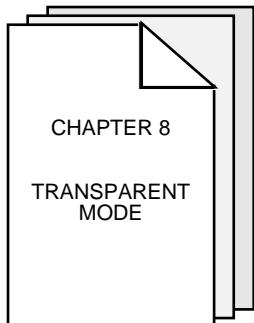
General purpose functions used in ladder language.

Syntax.

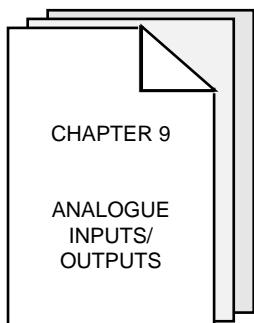
Operation.



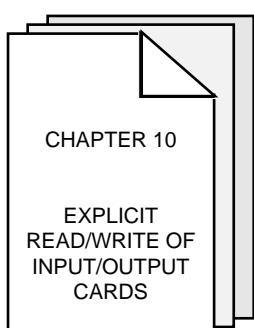
Principles and functions related to task management.



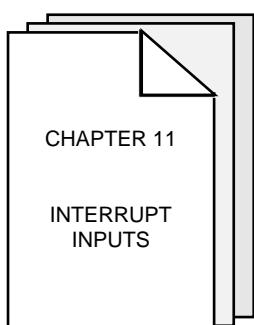
Principles and functions related to programming in transparent mode.



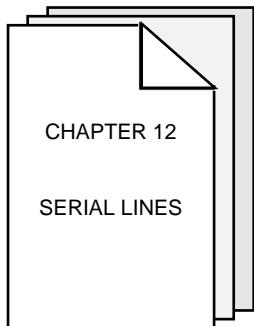
Principle and functions related to analogue input/output programming.



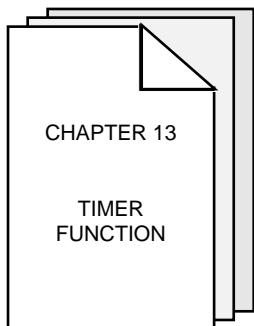
Principle and functions related to read and immediate write of input/output cards.



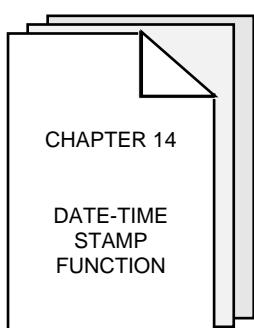
Principle and functions related to programming the interrupt inputs.



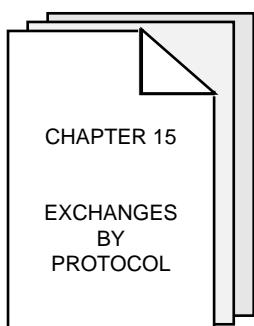
Principle and functions related to programming the serial lines.



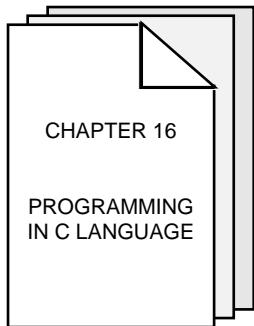
Principle and applications related to programming the timer function.



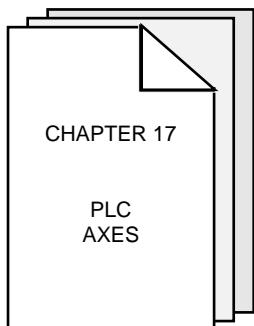
Principles and applications related to programming the date-time stamp.



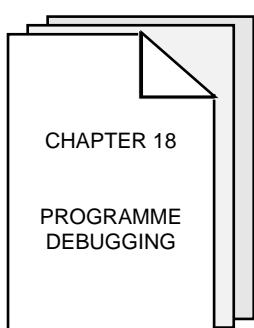
Principle and applications related to programming exchanges by PROTOCOL.



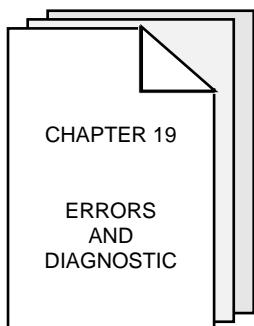
Functions processing calls to modules in C.



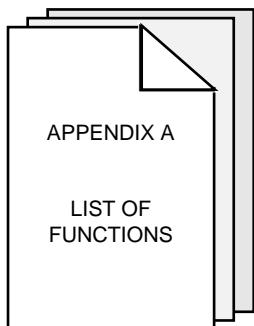
Principles and applications related to programming the PLC axes.



Programme creation and debugging tools.  
- Operating modes.



Machine processor monitoring levels and list of errors.



List of Ladder functions:

- Sorted by theme
- In alphanumeric order

## Use of the Automatic Control Function Programming Manual

### Procedures

This manual includes procedures (in particular in chapter 18).

The actions required are presented as follows:

Reset the system.



The keys to be pressed are indicated on the right. They can have two forms:



Square keys: correspond to keys on the operator panel.



Rectangular keys: correspond to softkeys located in the block at the bottom of the screen and activated by function keys (F2-F11) located under the screen.

### Index

The index at the end of the volume is convenient for looking up information related to keywords.

### Agencies

The list of NUM agencies is given at the end of the volume.

### Questionnaire

To help us improve the quality of our documentation, we kindly ask you to return the questionnaire at the end of this volume.



---

# 1 Presentation of the Automatic Control Function

1.1 General	1 - 3
1.2 Automatic Control Function	1 - 6



## 1.1 General

The NUM 1060 CNC is a multimaster multiprocessor system in which the automatic control function provides the interface with the NC machine-tool.

The automatic control function processes functions involving the sensors and actuators on the NC machine-tool and Boolean and numerical data interfacing with the NC function.

Its capabilities for accessing the CNC screen and simulating the operator panel make it very flexible to use, allowing the machine-tool OEM to customise the 1060 system for special applications.

The automatic control function is located in the central processing unit. The central processing unit includes one or more cards and provides the CNC, graphic and automatic control functions as well as the memory.

The CPU includes:

Functions	CNC	Graphic	Automatic control function	Memory
NUM 1060 Series I	CNC processor	Graphic processor	Machine processor	Memory card
NUM 1060 Series II multicard	Graphic processor	Graphic processor	Machine processor	Memory card
NUM 1060 Series II single card	UCSII	UCSII	UCSII	UCSII

Data transfers by bit or by byte with the input/output cards are via the serial bus.

Data exchanges with the system are of two types:

- communication via the exchange area,
- communication by protocol.

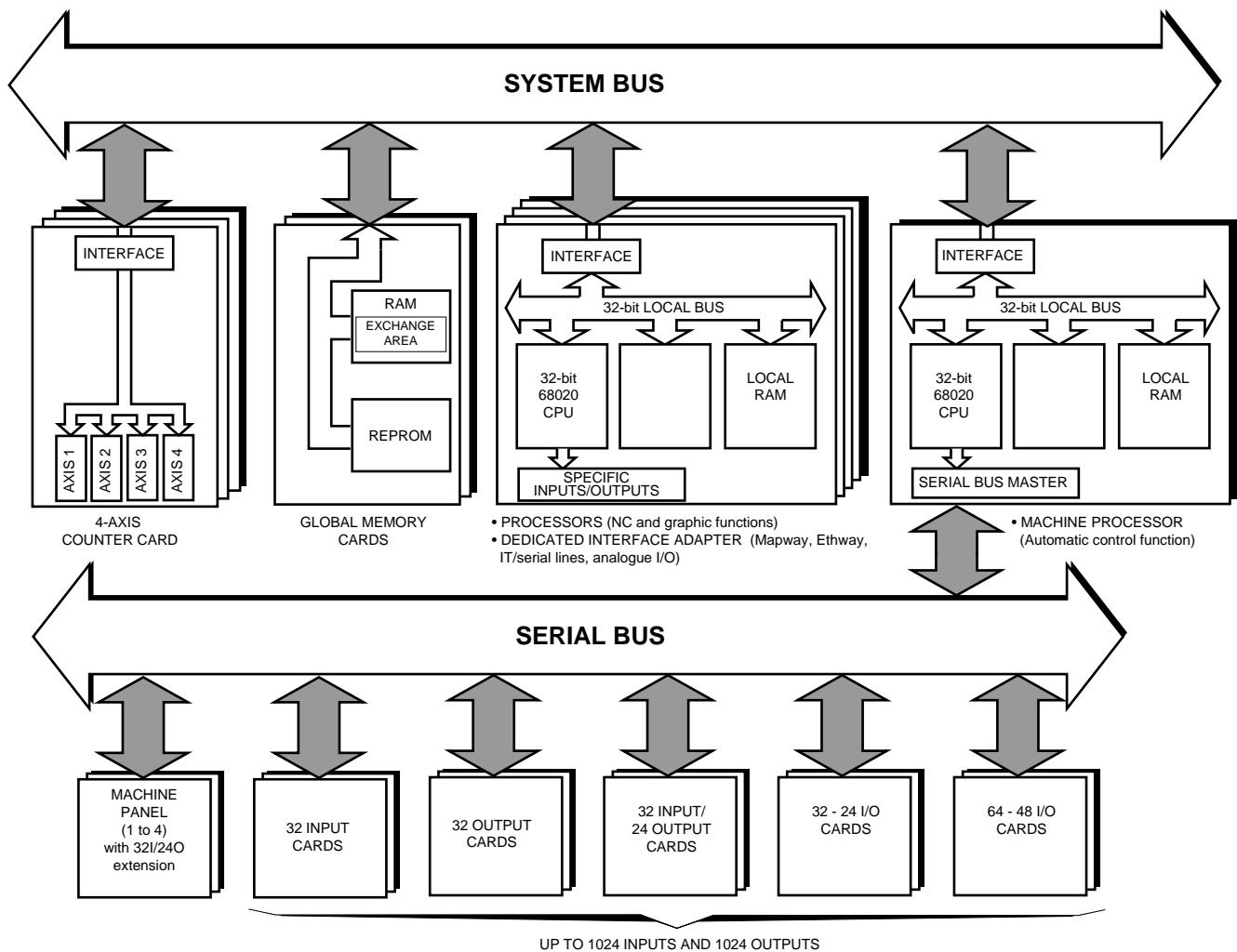


Figure 1.1 - General block diagram of a multiprocessor CPU

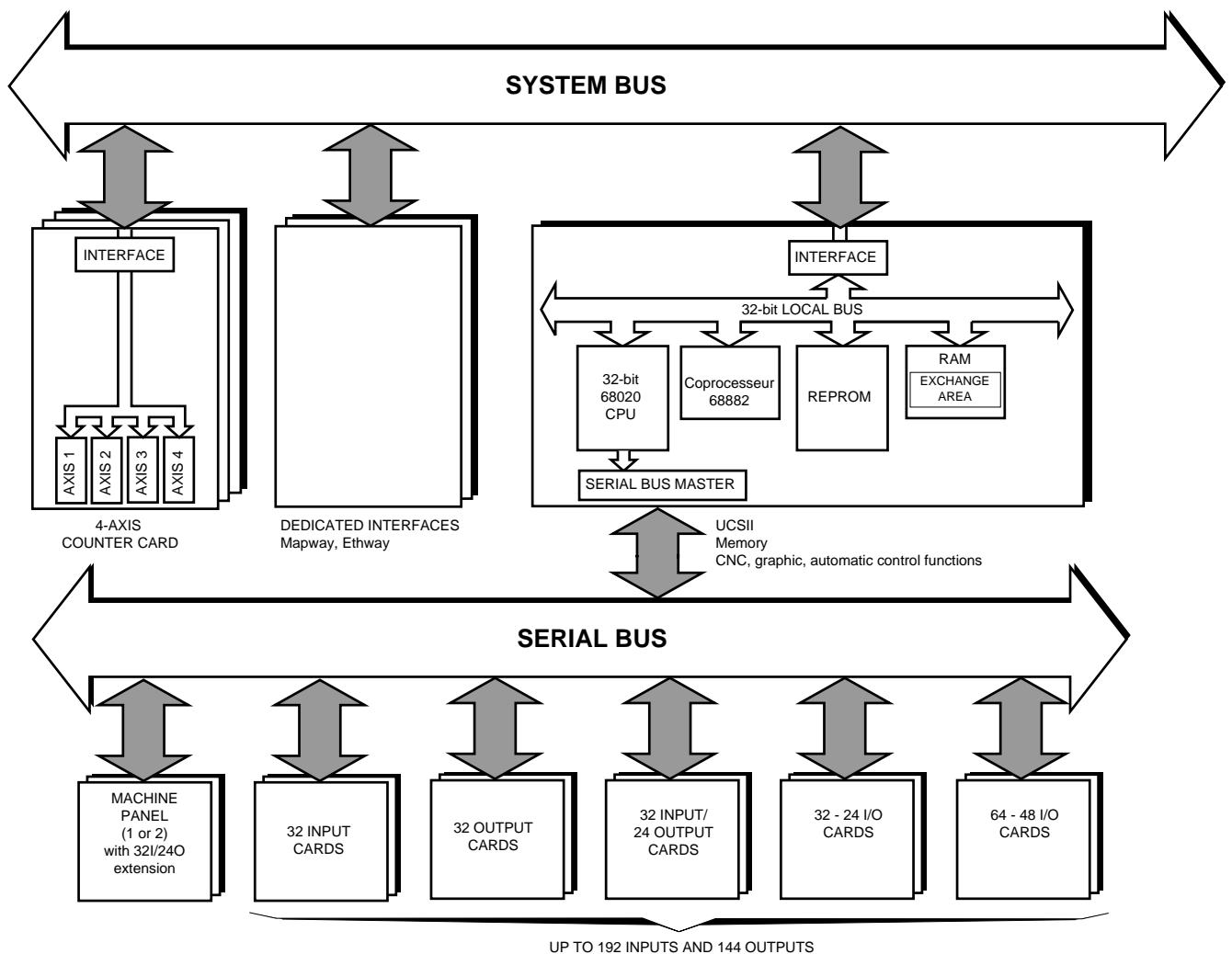


Figure 1.2 - General block diagram of a single-card CPU

## 1.2 Automatic Control Function

The automatic control function is managed by a monitor that handles a number of basic tasks such as initialisation, input/output assignment to the different racks, input/output interface, watchdog, etc.

To this systematic processing performed by the monitor is added processing of programmes called «user programmes».

The programme is executed under control of the monitor. Programme execution is clocked by the real-time clock (RTC) with a 20 ms cycle.

The machine processor memory space is organised as follows:

- 30 KB static RAM with backup,
- 32 KB dynamic RAM reset at power on,
- 180 KB of dynamic RAM occupied by the user programme on the 1 MB V1 machine processors,
- 2.5 MB of dynamic RAM occupied by the user programme on the 4 MB V1 machine processors,
- 3.5 MB of dynamic RAM occupied by the user programme on the 4 MB V2 machine processors,
- 64 KB of dynamic RAM occupied by the user programme on the UCSII modules.

The automatic control function allows:

- direct access to the DACs (12 bits),
- indirect read and write access to the ADCs and inputs/outputs. This access is via the virtual memory space (updated every 20 ms).

	No. of inputs/outputs	Maximum No. of racks
NUM 1060	1024I/1024O	1 main 6 extension
NUM 1060 Series II	192I/144O	1 main

The configuration of the inputs/outputs is set at power on. The refresh cycle requires 2 ms.

The use of the automatic control function requires availability of the PLCTOOL programming tool on PC. It is used to:

- write programmes in Ladder language,
- compile programmes,
- transfer programmes to the CPU,
- debug programmes after loading them.

Used in conjunction with the MICROTEC RESEARCH MCC68K compiler, PLCTOOL also allows writing, compilation and transfer of programmes in C.

Programmes are loaded and unloaded via one of the system serial lines.

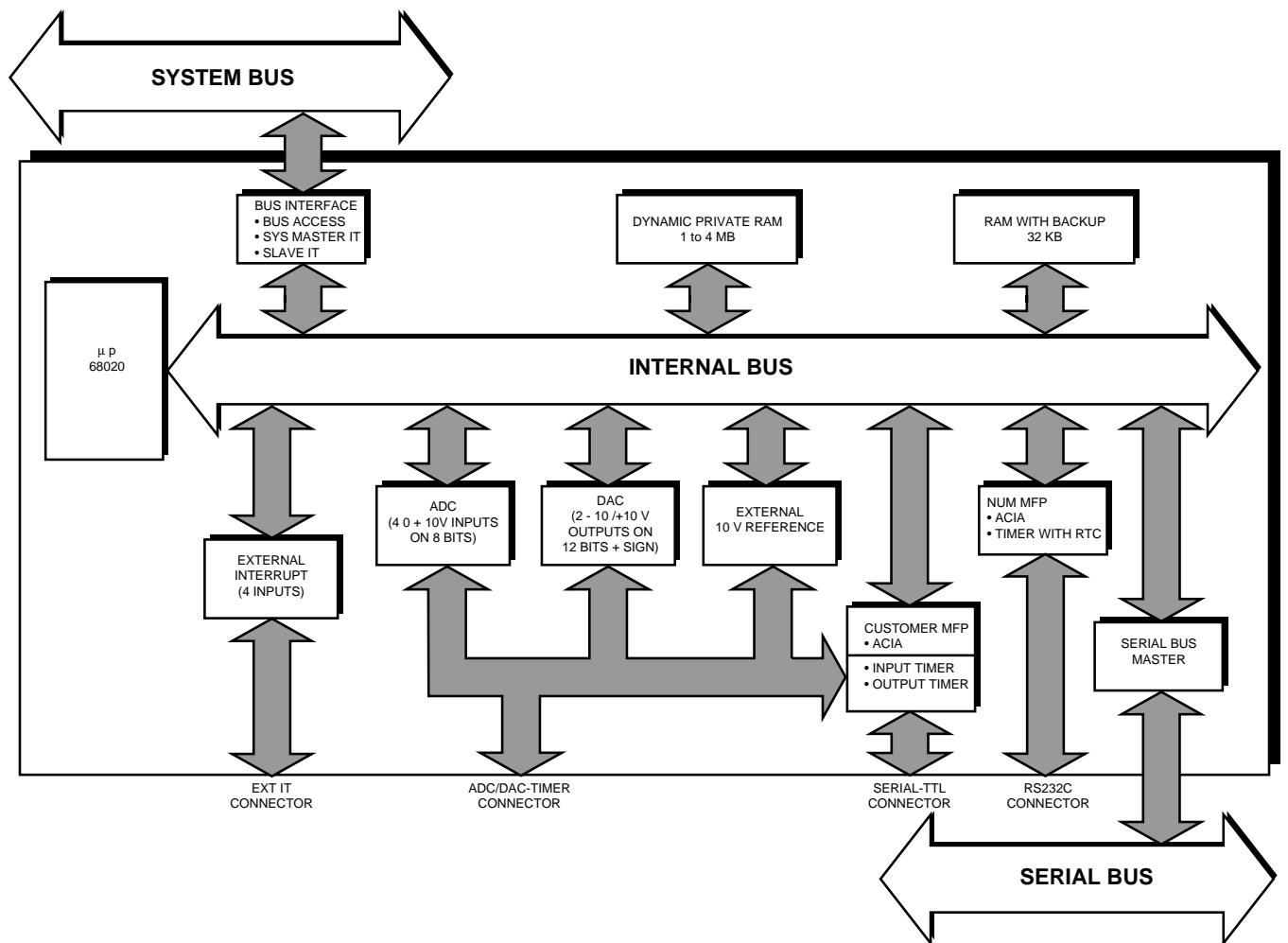


Figure 1.3 - Block diagram of the machine processor

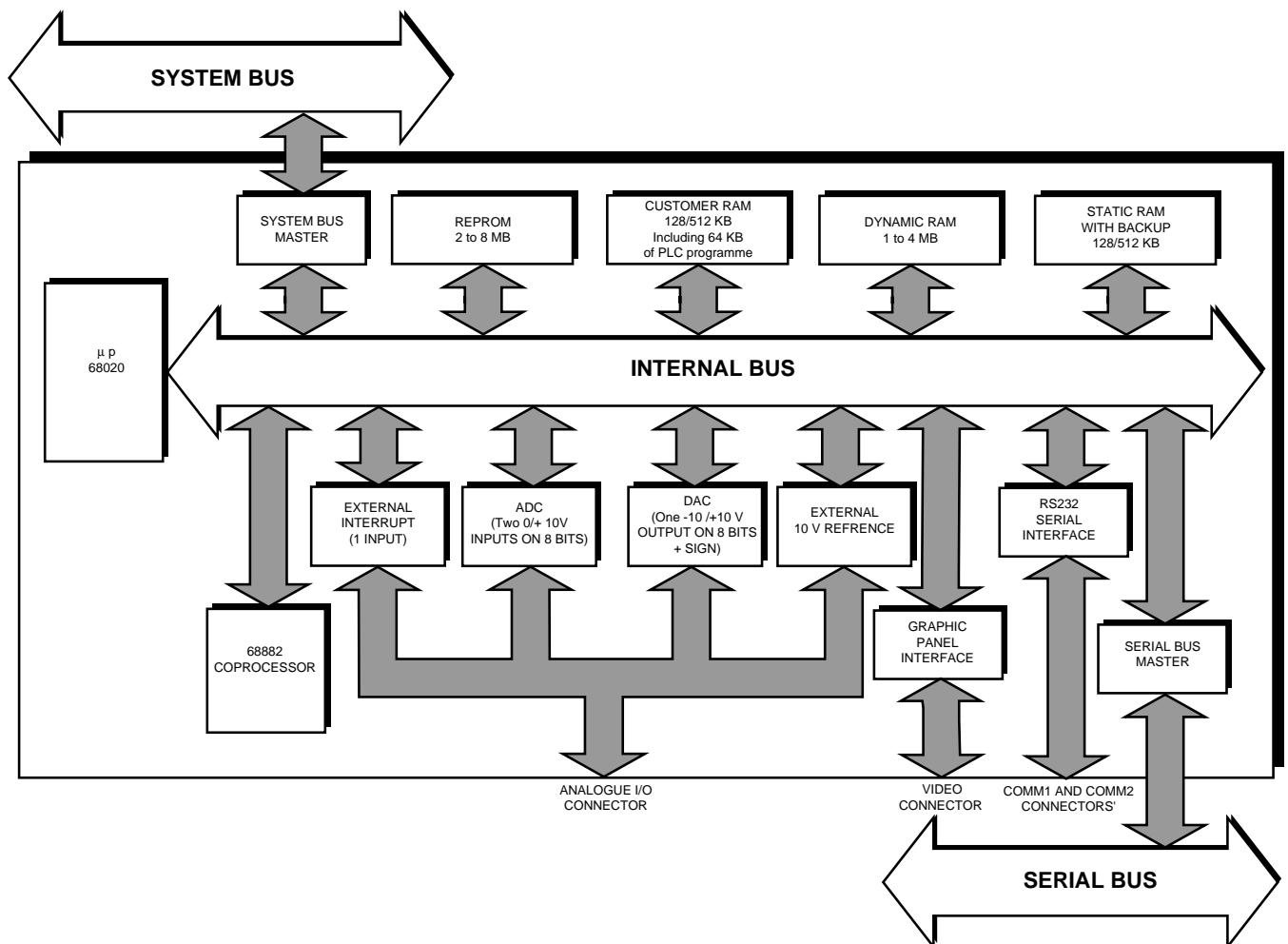


Figure 1.4 - Block diagram of the UCSII card

## 2 Structure of an Application

2

<b>2.1 General</b>	2 - 3
2.1.1 «System» Tasks	2 - 3
2.1.1.1 «System» Start up Task	2 - 3
2.1.1.2 «System» CNC I/O Refresh Task	2 - 3
2.1.1.3 «System» I/O Card Refresh Task	2 - 5
2.1.1.4 «System» UNITE Server Task	2 - 5
2.1.2 «User» Tasks	2 - 5
2.1.2.1 Start up Task	2 - 5
2.1.2.2 Cyclic Tasks	2 - 5
2.1.2.3 Background Tasks	2 - 6
2.1.2.4 Real-Time Tasks	2 - 9
2.1.3 Overrun Processing	2 - 10
2.1.3.1 1060 Series I and Series II Multicard Systems	2 - 10
2.1.3.2 1060 Series II- UCSII Systems	2 - 11
<b>2.2 Structure of an Application</b>	2 - 13
<b>2.3 Structure of a Ladder Module - Elementary Sequences</b>	2 - 15
<b>2.4 Elements Common to All Types of Sequences</b>	2 - 15
<b>2.5 Table of Constants Sequence</b>	2 - 15
2.5.1 General	2 - 15
2.5.2 Using a Table	2 - 15
2.5.3 Initialising a Table	2 - 15
<b>2.6 Character String Sequence</b>	2 - 16
2.6.1 General	2 - 16
2.6.2 Using a String	2 - 16
2.6.3 Initialising a String	2 - 16
<b>2.7 Ladder Sequence</b>	2 - 16



## 2.1 General

The machine processor card performs two types of tasks:

- «system» tasks initiated by the monitor and not programmable by the user,
- «user» tasks programmable by the user.

### 2.1.1 «System» Tasks

#### 2.1.1.1 «System» Start up Task

##### Processing Performed

During a start up, the system performs the following functions:

- self-test on the CPU resources,
- check of the integrity of the system code in the global memory,
- «System» code transfer from the global memory to the working memory,
- check of the integrity of the user code in the global memory,
- «User» code transfer from the global memory to the working memory,
- scan of the I/O cards present on the serial bus:
  - . update of the status and identification of each I/O card,
  - . read of the inputs of each I/O card and update of the image area %I.
- initiation of user task %INI.

##### Occurrence

The machine processor card is started:

- when power is applied to the CNC,
- when the Restart (RaZ) pushbutton on the front edge of the power supply card is pressed.

#### 2.1.1.2 «System» CNC I/O Refresh Task

##### Processing performed

This task carries out systematic exchanges with the CNC:

- read of %R. CNC inputs (%R. variables written by the NC function),
- write of %W. CNC outputs (%W. variables read by the NC function).

##### Occurrence

This task is executed once each real time clock (RTC) cycle (real-time clock).

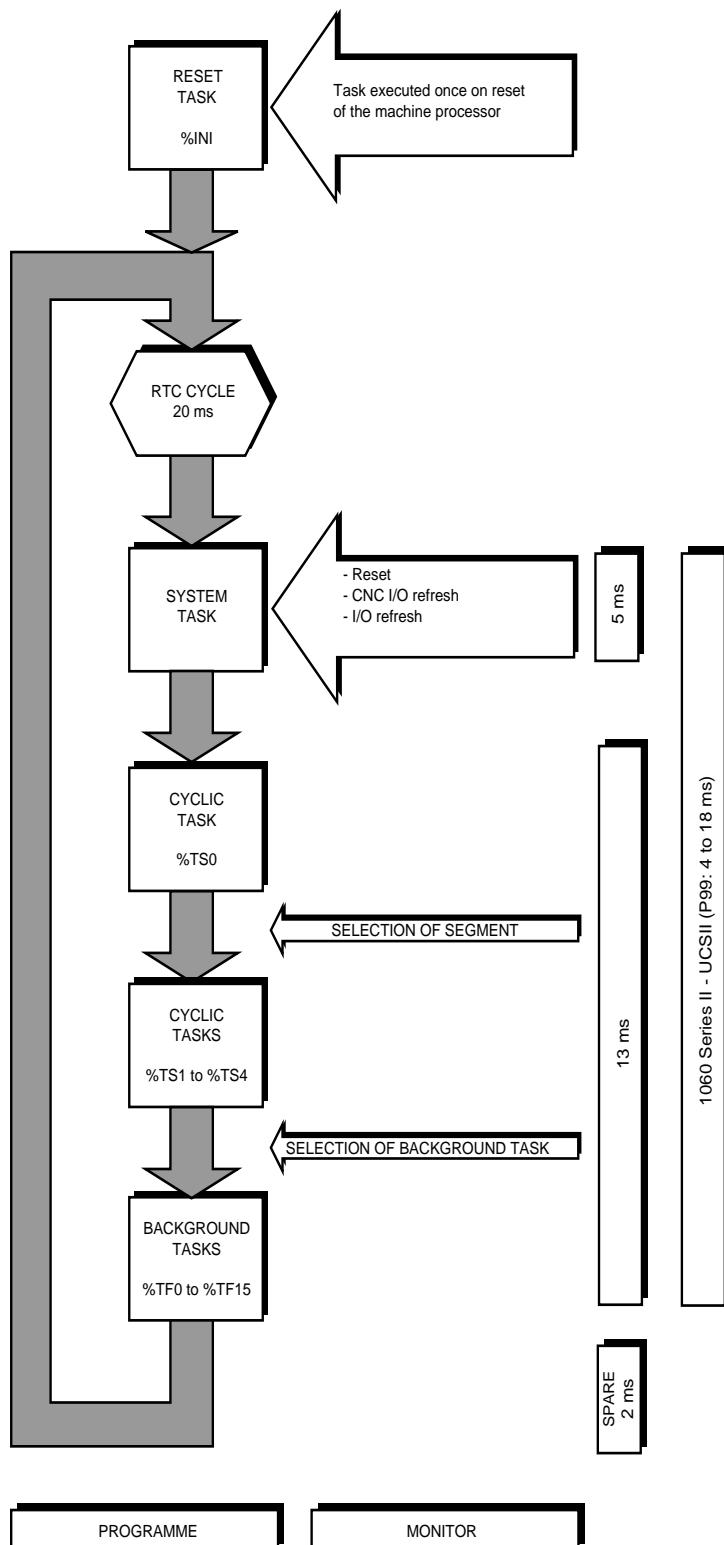


Figure 2.1 - Execution of an application

### 2.1.1.3 «System» I/O Card Refresh Task

#### Processing Performed

This task refreshes the card I/Os:

- read of %I. card inputs,
- write of %Q. card outputs,
- update of I/O card diagnostic variables.

### 2.1.1.4 «System» UNITE Server Task

#### Processing Performed

This task processes the UNITE requests sent to the automatic control function server.

#### Service Performed by the UNITE Server

The automatic control function server mainly processes the following UNITE requests:

- Read/write of variables (%I, %Q, %R, %W, %M, %V),
- load/unload of automatic control function tasks (%TS0, %SP30, etc.),
- automatic control function STOP (stopping of user tasks),
- automatic control function INIT (CPU initialisation),
- automatic control function RUN (running of user tasks).

## 2.1.2 «User» Tasks

### 2.1.2.1 Start up Task

The %INI task is called by the system before any other user tasks when the automatic control function is started.

This task can re-configure the I/O cards.



#### CAUTION

The system reads the I/O card configuration returned by %INI. Any changes made later to the configuration are ignored until the next restart.

### 2.1.2.2 Cyclic Tasks

The cyclic tasks are %TS0 to %TSn where n is incremented from 1 to 5 each RTC cycle.

The execution period of %TS0 is equal to an RTC cycle, i.e. 20 ms.

The execution period of %TS1, %TS2, %TS3 and %TS4 is equal to five RTC cycles, i.e. 100 ms (the fifth RTC cycle is used by system task %TS5).

Tasks %TS are not interruptible:

- by occurrence of the RTC (1060 series I and series II multicard),
- by the IT set by parameter P99 (1060 series II - USCI).

### 2.1.2.3 Background Tasks

These tasks (%TF0 to %TF15) are used for low priority tasks so as not to penalise cyclic tasks %TS. They also allow the use of wait functions.

Background tasks %TF have lower priority than tasks %TS and %TH.

%TF tasks have the following features:

- a task is executed on an explicit request by function tfstart(..),
- a task is not executed unless some time remains after execution of the cyclic tasks,
- a task is only executed once per RTC cycle,
- a task reset by function tfstop(..) is executed completely.

Tasks %TF are interruptible:

- by occurrence of the RTC (1060 series I and series II multicard),
- by the IT set by parameter P99 (1060 series II - USCI).

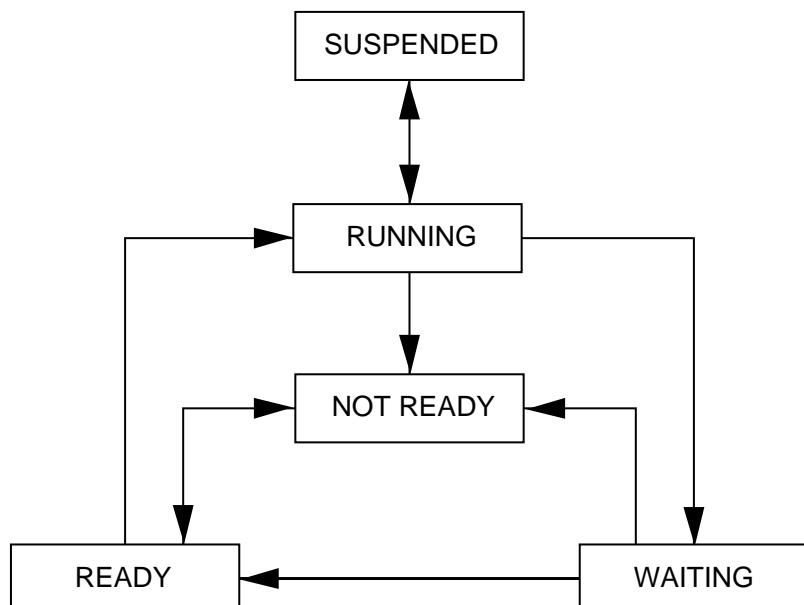


Figure 2.2 - States of a %TF task

## Operation of the Background Tasks

### NOT READY —> READY

When the system is reset, tasks %TF are in NOT READY state. The call to function tfstart(n) causes %TFn to go into READY state.

### WAITING or READY or RUNNING —> NOT READY

The call to function tfstop(n) causes task %TFn to go into NOT READY state.

### READY —> RUNNING

In READY state, a task %TF is executed as soon as no task is RUNNING and no %TF task with higher priority is in READY state.

The hierarchy of tasks %TF is determined by their number:

**%TF0 priority > %TF1 priority > ... > %TF15 priority**

### RUNNING —> SUSPENDED

Task %TF is suspended to allow execution of a task %TS or %TH. Tasks %TF are not mutually preemptible.

### SUSPENDED —> RUNNING

When no %TS or %TH task is running, the suspended %TF task is resumed.

### RUNNING —> WAITING

Task %TF calls function whtr() or ends (end of task reached).

*REMARK Function whtr(..) programmed in a background task interrupts the task during execution to allow the execution of other %TF tasks in READY state.*

### WAITING —> READY

The task was placed on wait by function whtr(..) for a given number of RTC cycles which have now elapsed.

The task is completed, and the occurrence of the RTC returns it to READY state.

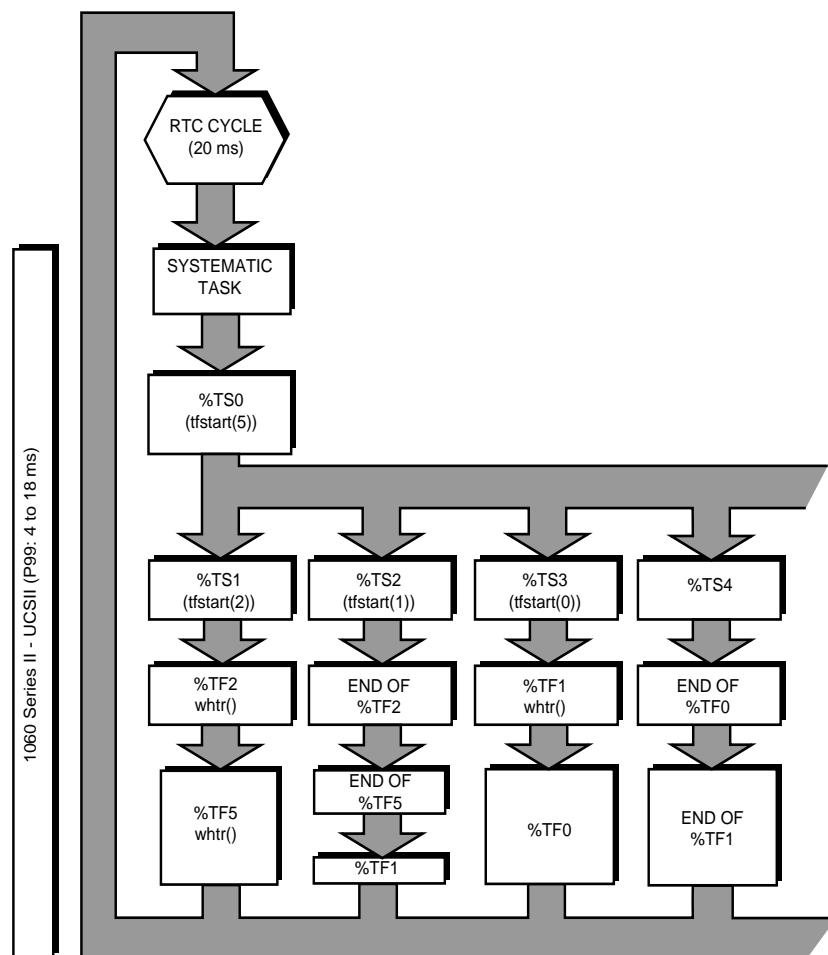


Figure 2.3 - Processing of %TS and %TF tasks

#### 2.1.2.4 Real-Time Tasks

Tasks %TH0 to %TH15 allow processing of high priority events that cannot wait for the end of the RTC cycle.

Tasks %TH therefore have higher priority than tasks %TS and %TF.

An activated %TH task cannot interrupt a running %TH task.

The hierarchy of %TH task priorities is determined by their number:

**%TH0 priority > %TH1 priority > ... > %TH15 priority.**

#### Operation of Tasks %TH

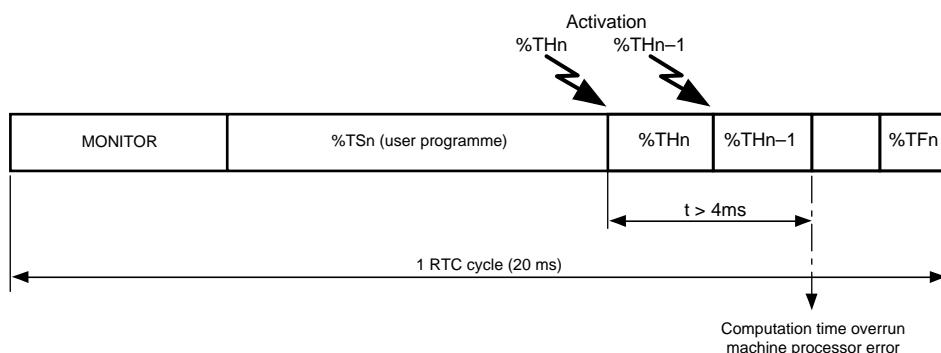
The programmer associates a task %TH with a hardware interrupt by the following functions:

thiti() probe interrupt.

thtimer() timer interrupt.

When the interrupt occurs, the system starts execution of the associated task %TH.

If several %TH tasks are activated during the same RTC cycle, the total time required for processing each interrupt routine must not exceed 4 ms. If it does, the machine processor goes into the «computation time overrun» error.



## 2.1.3 Overrun Processing

User programme operational errors are indicated by:

- increment of the RTC overrun counter %R97C.W,
- HALT\_ON\_ERROR of the CPU in critical cases.

This type of error requires debugging the user programme.

### 2.1.3.1 1060 Series I and Series II Multicard Systems

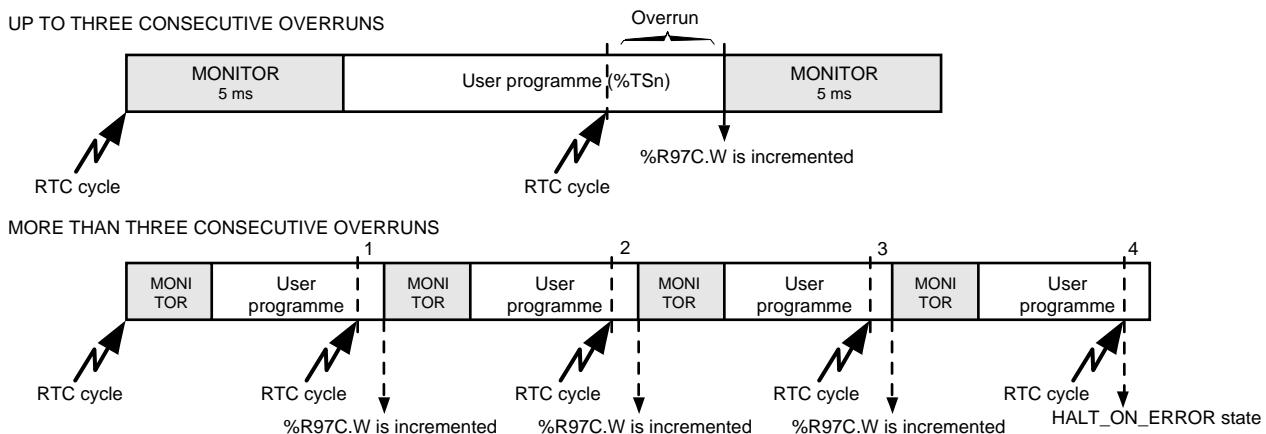
Processing of %TS tasks is clocked by the RTC with a 20 ms cycle. It should normally be completed before the end of the cycle.

#### Consecutive OVERRUNS

A slight overrun of the RTC cycle is tolerated in processing of tasks %TS. Each overrun, the system increments the RTC overrun counter %R97C.W.

The system accepts a maximum of three consecutive overruns.

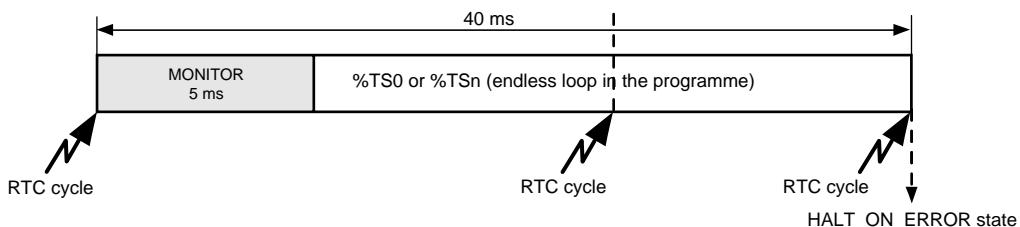
The fourth overrun increments the RTC overrun counter %R97C.W and causes the machine processor to go into HALT\_ON\_ERROR state.



**REMARK** Although a user programme may not create an overflow when operating off load (e.g. without machining parts), overflows may occur on load due to CNC hardware tasks (feedback processing, etc.) or ITs (serial lines, etc.) added in the interval between two RTCS.

#### Endless Loop in a Programme

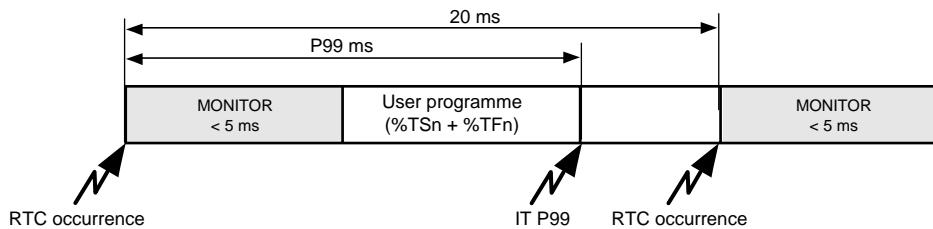
Uninterrupted execution of %TSn for more than 40 ms causes a HALT\_ON\_ERROR with the ERR\_RTC\_OVERRUN.



### 2.1.3.2 1060 Series II- UCSII Systems

#### Normal Operation

The user programme is clocked by the RTC with a 20 ms cycle. Its duration is however limited by parameter P99 (see Parameter Manual). Parameter P99 is a multiple of 2 ms between 4 and 18 ms. Processing of %TS tasks should normally be completed before occurrence of IT P99.



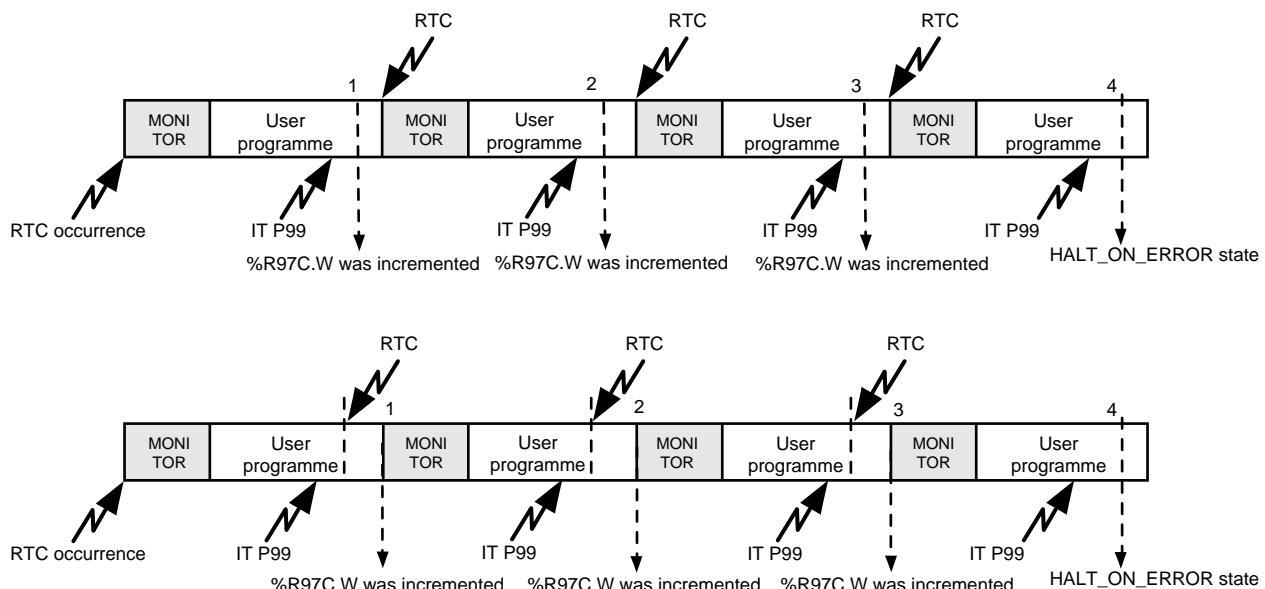
#### Consecutive OVERRUNS

A slight overrun of IT P99 is tolerated in processing of tasks %TS. Each overrun, the system increments the RTC overrun counter %R97C.W.

If the RTC interrupt occurs before the %TS tasks are completed, the monitor is immediately restarted.

The system accepts a maximum of three consecutive overruns.

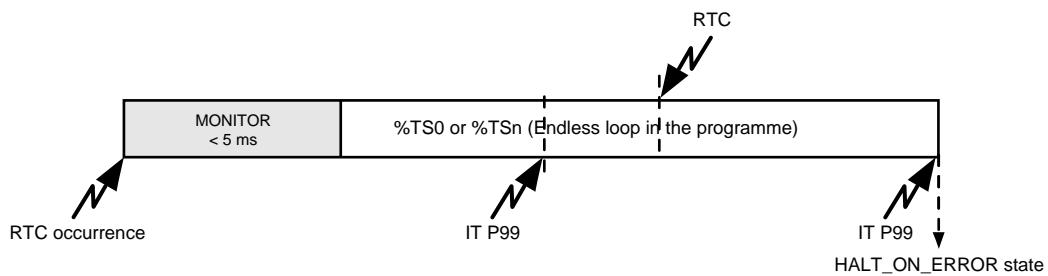
The fourth overrun increments the RTC overrun counter %R97C.W and causes the CPU to go into HALT\_ON\_ERROR state.



**REMARK** Although a user programme may not create an overflow when operating off load (e.g. without machining parts), overflows may occur on load due to CNC hardware tasks (feedback processing, etc.) or ITs (serial lines, etc.) added in the interval between two RTCS.

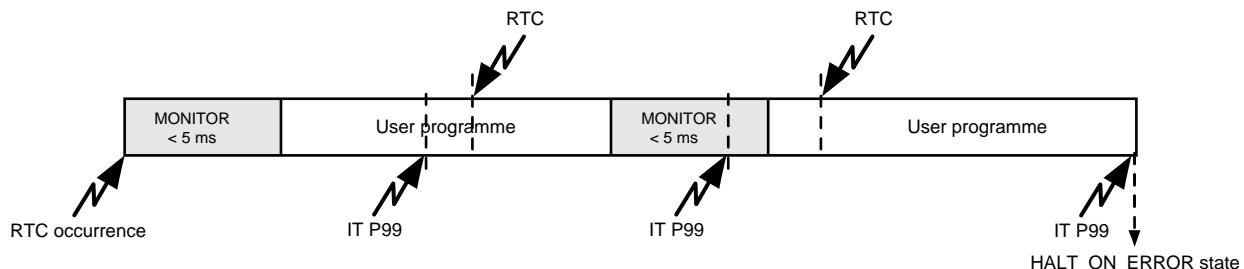
## Endless Loop in a Programme

Uninterrupted execution of %TSn for 2 x P99 ms causes a HALT\_ON\_ERROR with the ERR\_RTC\_OVERRUN state.



## Failure to Process the Monitor

Failure to process the monitor for 2 x P99 ms causes a HALT\_ON\_ERROR with the ERR\_RTC\_OVERRUN error.



## 2.2 Structure of an Application

An application includes a set of modules created under the PLCTOOL programming tool and loaded onto the NC to control the system.

### Detail of Modules

#### Ladder Task Modules

Ladder task modules are associated with:

- task %INI,
- tasks %TS0 to %TS4,
- tasks %TF0 to %TF15,
- tasks %TH0 to %TH15.

These modules are called by the system task manager. They cannot be called explicitly. The user manages these task modules by means of the task management functions (see Chapter 7).

Ladder task modules are files of the \*.XLA type.

#### Ladder subroutine modules

Ladder subroutine modules are associated with tasks %SP0 to %SP255 . These modules can be called from a task module or another subroutine module using function sp(..) or spy(..).

A PLC subroutine written in Ladder language and called by the SP function from a C module must not include ANY function calls (except functions goto 0 and call 0).

Ladder subroutine modules are files of the \*.XLA type.

#### Executable Modules

Executable modules are produced by a C compiler (MCC68K). These modules can be called from a ladder task module or a ladder subroutine module using function exec(..).

All tasks can be written in C. If the same task is written in C and in ladder language, it is the C code that is executed by the automatic control function.

Executable modules are files of the \*.XCX type.

#### Note on Initialising a C Module (.XCX)

Implementation of global memory code in the local PLC memory.

Execution of the following directives in the «main()»:

- Import(),
- Export(),
- Possible initialisation of certain variable types: global C variables of the XCX module, uninitialised variables (saved) of Ladder language.

Resolution of imports/exports (imported variables cannot be used in the «main()»)

Initialisation of initialised Ladder variables

Execution of %INI module

Normal PLC cycle start.

**REMARK** *On a PLC Stop/Start (without INIT), only the last three steps are executed.*

*The saved variables are %M, %C, %CQ; the initialised variables are %I, %Q, %R, %T, %TQ, %V. The so-called «pulse» variables %W are reset by the RESET or power off button.*

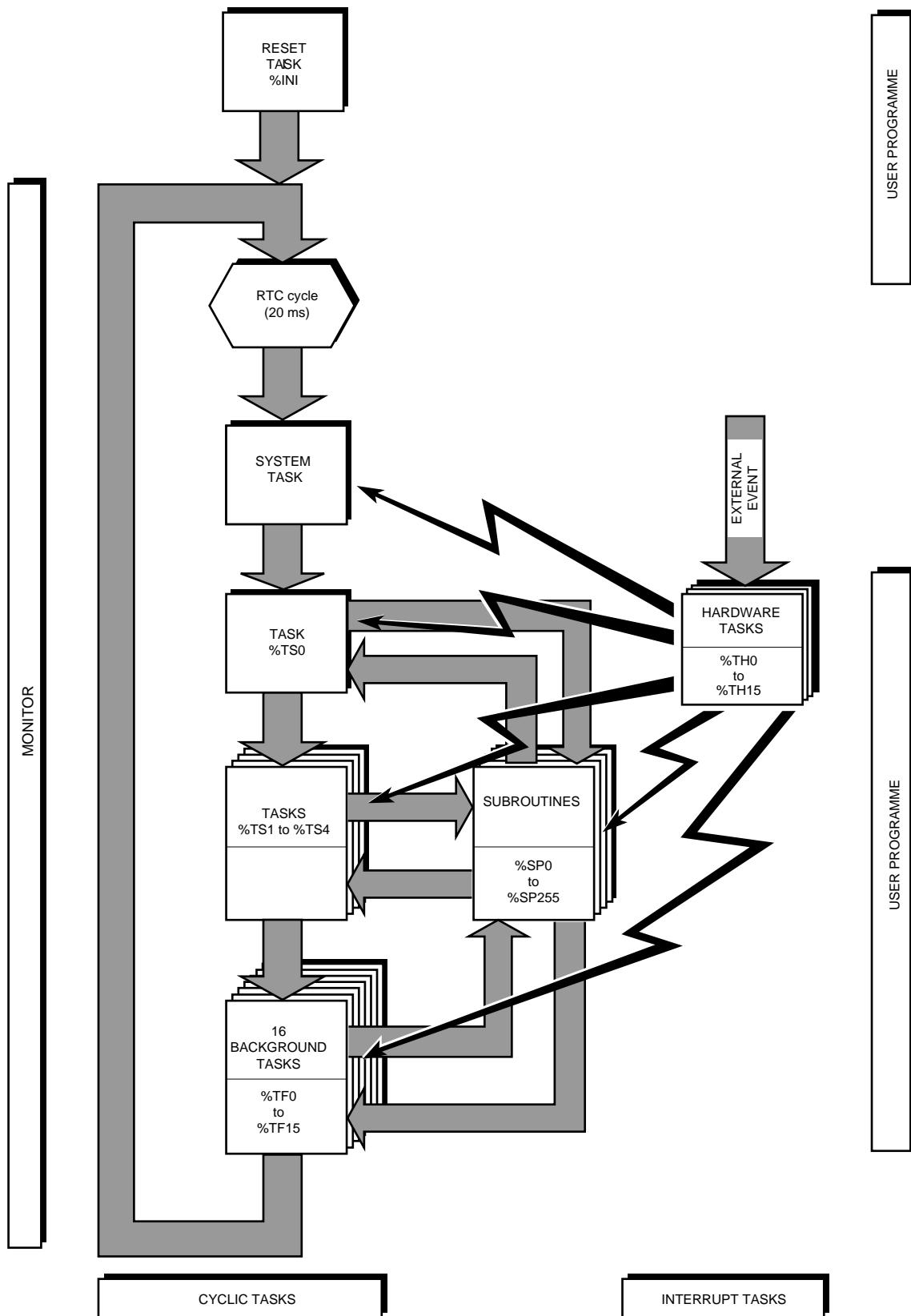


Figure 2.4 - Structure of an application

## 2.3 Structure of a Ladder Module - Elementary Sequences

A ladder module is a succession of elementary sequences. The number of sequences in a module is limited to 100.

The sequence is the basic entry and compilation unit. There are three types of elementary sequences:

- the sequence used for entry of a table of numerical constants,
- the sequence used for entry of one or more character strings,
- the sequence used for entry of a network of contacts and coils.

## 2.4 Elements Common to All Types of Sequences

Each sequence includes the following common elements:

- sequence header,
- grafct step.

## 2.5 Table of Constants Sequence

### 2.5.1 General

This type of sequence contains a data table that is processed by a buffer handling function.

This sequence includes:

- an optional label and an optional comment,
- an optional grafct step,
- an associated %Vxx.L or %Yxx.L variable that contains the table start address,
- a sequence of numerical values in which each value occupies a long word.

Each table can contain up to 500 values.

The number of tables that a module can contain is limited by the maximum module size and the number of sequences which must be less than 100.

### 2.5.2 Using a Table

Tables are accessed by the table start address contained in a %Vxx.L or %Yxx.L variable using functions of the type cpyb(..), cpyl(..), print(..), etc.

The values of a table can be accessed directly by the pointers (e.g. %Yxx -> n.L).

### 2.5.3 Initialising a Table

A table is initialised when the monitor loads variable %Vxx.L or %Yxx.L with the table start address.

A table sequence must therefore have been executed once by the system before being able to be used by a ladder sequence in the programme.

## 2.6 Character String Sequence

### 2.6.1 General

This type of sequence contains character strings that are mainly processed by the NC screen display functions.

It includes:

- an optional label and an optional comment,
- an optional grafset step,
- the definition of one to 32 character strings.

A character string definition includes:

- an associated variable %Vxx.L or %Yxx.L containing the string start address,
- a sequence of 120 characters at most.

The number of string sequences that a module can contain is limited by the maximum module size and the number of sequences which must be less than 100.

The compiler automatically adds a null byte to the end of each string.

### 2.6.2 Using a String

It is not possible to directly access the individual characters of a string.

Strings are accessed by the string start address contained in a %Vxx.L or %Yxx.L variable using functions of types printf(..), scano(..), scanu(..), etc.

The characters of a string can be accessed directly by pointers (e.g. %Yxx -> n.B).

### 2.6.3 Initialising a String

A string is initialised when the monitor loads variable %Vxx.L or %Yxx.L with the string start address.

A string sequence must therefore have been executed once by the system before being able to be used by a ladder sequence in the programme.

The same variable %Vxx.L or %Yxx can be associated with different strings in different sequences. In this case, the % variable %Vxx.L contains the address of the string located in the string sequence executed last.

## 2.7 Ladder Sequence

This type of sequence is the basic entity of the automatic control programme. A ladder sequence consists of contacts, branches and coils.

A ladder sequence includes:

- an optional label and an optional comment,
- an optional grafset step,
- a test area with six lines containing six contacts each (36 cells),
- an action area with six lines containing one coil each (six cells).

# 3 Variables

3

<b>3.1 Principle of Exchanges</b>	3 - 5
<b>3.2 Variable % - Mnemonic</b>	3 - 6
<b>3.3 Variable %</b>	3 - 6
3.3.1 Symbol Field	3 - 6
3.3.2 Logical Number Field	3 - 6
3.3.3 Size Field	3 - 7
3.3.4 Index Field	3 - 7
3.3.4.1 Indexing with a Bit Variable	3 - 7
3.4 Mnemonic	3 - 8
3.4.1 Coercion Field	3 - 8
<b>3.5 Common Internal Variables Saved</b>	3 - 8
<b>3.6 Common Internal Variables Not Saved</b>	3 - 8
<b>3.7 I/O Card Interface Variables %I and %Q</b>	3 - 9
3.7.1 Structure of Read Variables %Irc	3 - 10
3.7.1.1 Card Diagnostic Part	3 - 10
3.7.1.2 Input Image Part	3 - 10
3.7.2 Structure of Write Variables %Qrc	3 - 10
3.7.2.1 Card Configuration Part	3 - 10
3.7.2.2 Output Image Part	3 - 10
3.7.3 Card Diagnostic Variables	3 - 10
3.7.3.1 Card Identifier %Irc3E.W	3 - 10
3.7.3.2 Card Status %Irc3C.W	3 - 11
3.7.3.3 Dialogue Error Counter %Irc3A.W	3 - 11
3.7.3.4 Bus Status %Irc39.B	3 - 11
3.7.4 Card Configuration Variables	3 - 12
3.7.4.1 Card Identifier %Qrc3E.W	3 - 12
3.7.4.2 Logical/Geographic Address Option %Qrc3D.B	3 - 12
3.7.4.3 Card Priority %Qrc3C.B	3 - 13
3.7.4.4 Watchdog %Qrc3B.0	3 - 14
3.7.4.5 NC Access Enable %Qrc3B.1	3 - 14
3.7.5 Physical Organisation of Variables %I and %Q	3 - 15
3.7.5.1 Physical Organisation of Variables %I and %Q of Rack r	3 - 15
3.7.5.2 Physical Organisation of Variables %I and %Q of the Different Racks	3 - 16
3.7.6 Rack and Card Identifiers	3 - 17
3.7.6.1 Card Identifiers	3 - 17
3.7.6.2 Rack Identifiers	3 - 17
3.7.7 Image Part of 32 Discrete Input Card	3 - 18
3.7.8 Image Part of the 32 Discrete Output Card	3 - 19
3.7.9 Image Part of the 32-24 I/O Discrete I/O and 32-24 I/O Cards	3 - 20
3.7.10 Image Part of the 64-48 I/O Card	3 - 22
3.7.11 Image Part of the Basic Operator Panel Card	3 - 24
3.7.12 Image Part of the Extension Operator Panel Card	3 - 25

3.7.13	Image Part of the Compact Panel	3 - 27
3.7.13.1	Image Part of the Compact Panel in the Exchange Area	3 - 27
3.7.13.2	Image Part of the Compact Panel	3 - 27
3.7.13.3	Image of the JOG Softkeys	3 - 27
3.7.13.4	Programmable Key Image Leds	3 - 28
<b>3.8</b>	<b>CNC I/O Interface Family %R and %W</b>	<b>3 - 29</b>
3.8.1	Inputs from the CNC %R0 to %R7F	3 - 29
3.8.1.1	Keyboard Characters: %R0.W	3 - 29
3.8.1.2	Machine Status: %R2.W	3 - 29
3.8.1.3	CNC Status: %R4.W	3 - 30
3.8.1.4	Axes in Motion: %R6L	3 - 31
3.8.1.5	Axes Initialised (origin setting completed): %RA.L	3 - 32
3.8.1.6	External Parameters E10000 to E10031: %RE.L	3 - 32
3.8.1.7	Spindle Status: %R12.W	3 - 33
3.8.1.8	Type of Jog Increment: %R15.B	3 - 34
3.8.1.9	Current Mode: %R16.B	3 - 34
3.8.1.10	Other Variables	3 - 35
3.8.1.11	Spindle Speed: %R1C.W to %R22.W	3 - 36
3.8.1.12	Axis Clamp: %R24.L	3 - 36
3.8.1.13	1050 Servo-Drive Status Word	3 - 37
3.8.2	Outputs to the CNC %W0 to %W7F	3 - 38
3.8.2.1	Pulse Commands: %W2.W	3 - 38
3.8.2.2	Latching Commands: %W4.W	3 - 39
3.8.2.3	Positive JOG Commands: %W6.L	3 - 40
3.8.2.4	Negative JOG Commands: %WA.L	3 - 41
3.8.2.5	External Parameters E20000 to E20031: %WE.L	3 - 41
3.8.2.6	Value of the Jog Increment: %W13.B	3 - 42
3.8.2.7	Mode Requested: %W14.B	3 - 42
3.8.2.8	Message Display: %W15.B and W16.B	3 - 42
3.8.2.9	Axis Group Selection: %W17.B	3 - 43
3.8.2.10	Requested Programme Number: %W18.W	3 - 43
3.8.2.11	Handwheel Assignment: %W1A.B to %W1D.B	3 - 44
3.8.2.12	Spindle Potentiometer: %W1E.B to %W21.B	3 - 44
3.8.2.13	Spindle Controls: %W22.W	3 - 45
3.8.2.14	Spindle Speed Setting: %W24.W to %W2A.W	3 - 45
3.8.2.15	Inhibited Jog Increments: %W2C.W	3 - 48
3.8.2.16	Modes Inhibited: %W30.L	3 - 49
3.8.2.17	Torque Enable for QVN Axes: %W34.L	3 - 50
3.8.2.18	Speed Reference Enable for QVN Axes: %W38.0	3 - 50

3.8.2.19	Backward or Forward Movement in Path	3 - 51
3.8.2.20	Feed Stop per Axis (the bit number corresponds to the physical address of the axis): %W3A.L	3 - 51
3.8.2.21	Current Reduction: %WE00.B to WE1F.B DISC and 1050	3 - 51
3.8.2.22	1050 Servo-Drive Control Word	3 - 52
3.8.3	Inputs from the Axis Groups	3 - 53
3.8.3.1	Group Status: %Rg00.W	3 - 53
3.8.3.2	Current Machining Cycle Number: %Rg02.B	3 - 54
3.8.3.3	G Function Status: %Rg03.B	3 - 54
3.8.3.4	Encoded M Function Without Response: %Rg04.W	3 - 55
3.8.3.5	Encoded M Function With Response: %Rg1E.W	3 - 55
3.8.3.6	Decoded M Functions: %Rg20.L	3 - 56
3.8.3.7	Decoded M Functions (Spindle Status): %Rg24.W	3 - 58
3.8.3.8	Axis Clamp/Unclamp	3 - 59
3.8.3.9	Tool Number: %Rg7C.L	3 - 59
3.8.4	Outputs to the Axis Groups	3 - 61
3.8.4.1	Group Commands: %Wg00.W	3 - 61
3.8.4.2	Feed Rate Potentiometer Setting: %Wg02.B	3 - 62
3.8.4.3	Independent Group Mode: %Wg03.B	3 - 62
3.8.5	System Faults and Diagnostic	3 - 63
3.8.5.1	System or Configuration error	3 - 63
3.8.5.2	System Diagnostic	3 - 63
3.8.6	Selecting the Module to Be Animated	3 - 64
3.8.7	Output Card Write Enable: %W900.0	3 - 65
3.8.8	System Fault Management	3 - 65
3.8.9	External Parameters E30xxx, E40xxx and E42xxx	3 - 65
3.8.9.1	External Parameters E30xxx	3 - 65
3.8.9.2	External Parameters E40xxx	3 - 66
3.8.9.3	Parameters E42xxx	3 - 66
3.8.10	Physical Organisation of %R and %W Variables	3 - 67
<b>3.9 %S Common Word Variables</b>		3 - 68
3.9.1	Variable Update	3 - 68
3.9.2	Setting up the Common Words	3 - 68
3.9.3	Organisation of %S Common Word Variables	3 - 69
<b>3.10 %Y Local Variables- Pointers</b>		3 - 70
3.10.1	General	3 - 70
3.10.2	Indirect Addressing - Pointers	3 - 70
3.10.3	Examples of Use of Pointers	3 - 71

---

<b>3.11 Exchange Area</b>	<b>3 - 72</b>
3.11.1 Inputs from the CNC	3 - 72
3.11.2 CNC-PLC Exchange Area - 1050	3 - 74
3.11.3 Output to the CNC	3 - 75
3.11.4 PLC-CNC Exchange Area - 1050	3 - 79
3.11.4.1 Torque Modulation	3 - 79
3.11.4.2 Servo-Drive Control Word	3 - 79
3.11.5 Inputs from Axis Groups	3 - 80
3.11.6 Outputs to Axis Groups	3 - 81

### 3.1 Principle of Exchanges

Exchanges between the automatic control function and the NC function are via a memory space accessible to both functions called the exchange area.

Exchanges with the discrete input/output cards are processed directly by the automatic control function.

**REMARK** *The terms inputs and outputs are defined with respect to the automatic control function.*

*An input is a variable read by the automatic control function.  
An output is a variable written by the automatic control function.*

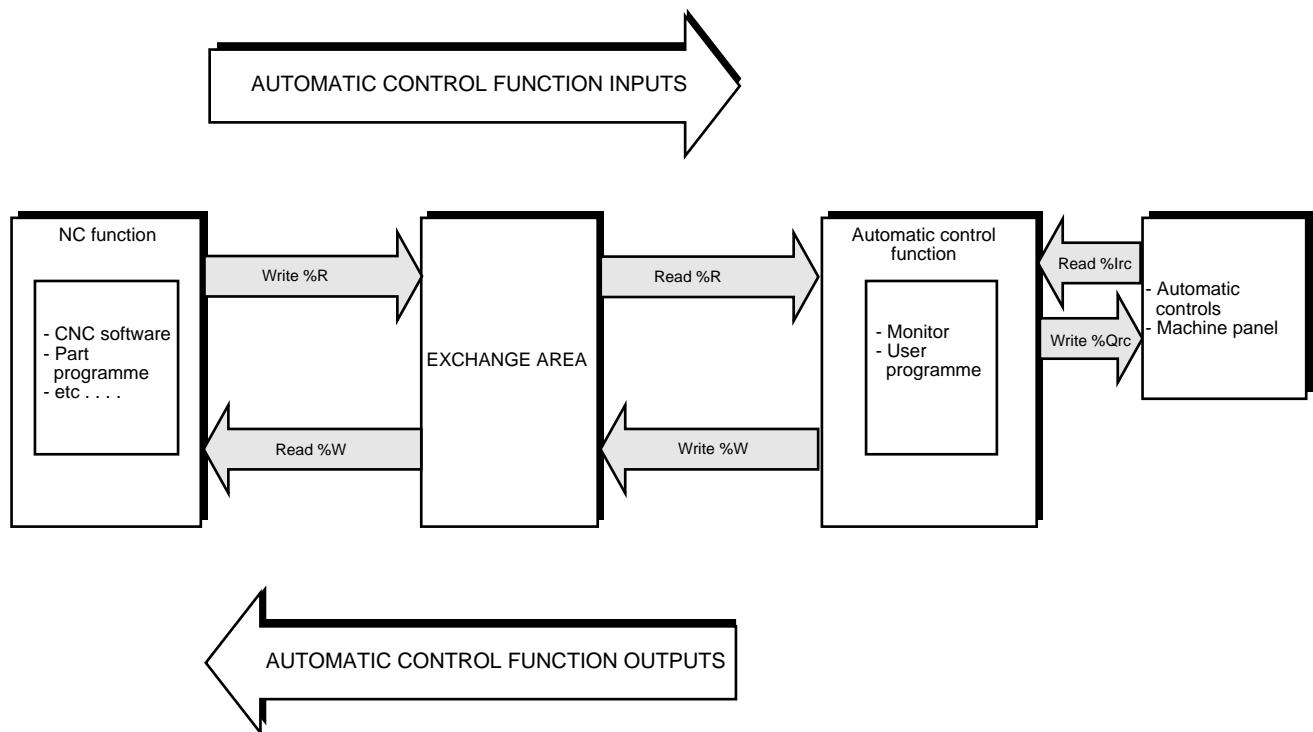


Figure 3.1 - Principle of exchanges

## 3.2 Variable % - Mnemonic

A variable can be represented in two ways:

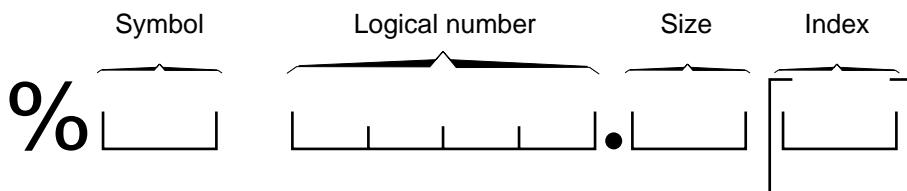
- One representation that always begins with the character %. This representation allows the compiler to determine the physical address of the variable,
- An optional user representation also called mnemonic. This representation does not have to begin with %.

The user can associate a mnemonic with a % variable in a symbol table (see the «PLCTOOL Programming Tool» manual).

## 3.3 Variable %

This type of variable always begins with % and includes the following fields:

- Symbol,
- Logical number,
- Size,
- Index.



### 3.3.1 Symbol Field

This field is mandatory.

This field indicates the family of the variable.

Field value	Definition
%M	For saved common internal variables
%V	For common internal variables that are not saved
%I	For I/O interface read variables
%Q	For I/O interface write variables
%R	For CNC I/O interface read variables
%W	For CNC I/O interface write variables
%S	For common word variables
%Y	For local variables

### 3.3.2 Logical Number Field

This field is mandatory.

This field designates an object in a family. The logical number is hexadecimal of four digits maximum.

The logical number is the logical address in BYTES as an offset from the first element of the family.

## Examples

%M9 points to byte 9 of the family of internal variables %M.

%MA points to byte 10 of the family of internal variables %M.

### 3.3.3 Size Field

This field begins with a dot (.) followed by one of the following alphanumeric characters:

Field value	Definition
.n	Designates bit n (from 0 to 7) of the byte (bit 0 is the LSB, bit 7 is the MSB)
.B	Designates a signed integer of 8 bits
.W	Designates a signed integer of 16 bits (MSB at address n, LSB at address n+1)
.L	Designates a signed integer of 32 bits (MSB at address n, LSB at address n+3)
.&	Designates the variable address. An address is encoded on 32 bits

### 3.3.4 Index Field

This field is optional.

The index is between square brackets [] after the size field.

The index is a variable %M with size .W (e.g. %M34.L[%M5.W]).

The value of the index is added to the logical number of the base variable to find the address of the indexed variable.

#### Example

If           %M2.W = 4

Then        %M8.L[%M2.W] designates %MC.L.



#### CAUTION

Indexing is prohibited with a .& variable.

#### Example

%M34.&[%M4.W] is prohibited.

### 3.3.4.1 Indexing with a Bit Variable

Indexing of bit variables affects the byte address but does not change the bit location in the byte.

#### Example

If           %M2.W = 4

Then        %M8.7[%M2.W] designates %MC.7.

## 3.4 Mnemonic

A mnemonic is a combination of up to twelve characters at most chosen from:

- the 26 capital letters (A, B, C, ..., Z),
- the 26 lower case letters (a, b, c, ..., z),
- the 10 digits (0, 1, 2, ..., 9),
- the underline character ( \_ ).

A mnemonic must always begin with a letter (the underline character is prohibited). The compiler does not differentiate between upper and lower case letters. The user can only associate a mnemonic with a variable %.

These associations are saved in the PLCTOOL symbol files (\*.XSY).

### 3.4.1 Coercion Field

When using a mnemonic, the variable size specified can be different from that indicated when associating the mnemonic with a % variable.

Coercion is specified after the mnemonic by a dot (.) followed by the symbol for the new size.

#### Example

If the mnemonic «Status\_word» is associated with variable %M3.B,

Then:                  The mnemonic «Status\_word.0» represents %M3.0  
                          The mnemonic «Status\_word.7» represents %M3.7  
                          The mnemonic «Status\_word.W» represents %M3.W  
                          The mnemonic «Status\_word.L» represents %M3.L

## 3.5 Common Internal Variables Saved

These are variables from %M0 to %M77FF (i.e. 30 kbytes).

These %M variables are saved in case of a power failure.

## 3.6 Common Internal Variables Not Saved

These are variables %V0 to %V7FFF (i.e. 32 kbytes).

These %V variables are not saved in case of a power failure or a CPU INIT. They are reset by reset of the central processing unit.

### 3.7 I/O Card Interface Variables %I and %Q

This type of variable is associated with the following elements:

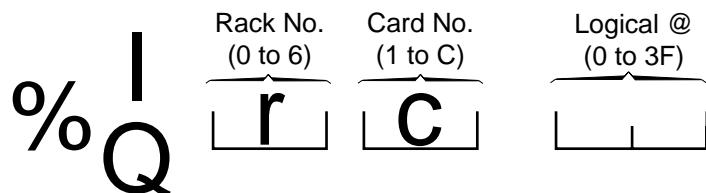
- 32-discrete-input cards,
- 32-discrete-output cards,
- 32-discrete-input/24-discrete-output card,
- 32-24 I/O cards,
- 64-48 I/O card,
- machine panel,
- machine panel 32-discrete-input/24-discrete-output extension cards.

Each interface discrete I/O card includes:

- a block of 64 bytes of %I read variables
- a block of 64 bytes of %Q write variables.

The I/O cards are addressed logically (see Sec. 3.7.4) on four digits. The default value (no reconfiguration) is:

**Logical @ = geographic @**



The card number c and rack number r are related to the type of equipment. Refer to the Installation and Commissioning Manual for rack addressing.

Equipment type	Rack number	Card number
Main 19" rack	0	5 to C
Main 12" rack	0	5 to 8
12-card extension rack	1 to 6	1 to C
2-card extension rack	1 to 6	1 and 2
Machine panel	0	1 to 4

#### Example

%I3500              Represents read byte 0 on card 5 in rack 3.

%Q652F              Represents write byte 0x2F on card 5 in rack 6.

*REMARK Logical (as opposed to geographical) addressing is also possible (see Sec. 3.7.4).*

### 3.7.1 Structure of Read Variables %Irc

The block of %Irc read variables (for cards 0 to C) is divided into two parts:

- card diagnostic part,
- card image part.

#### 3.7.1.1 Card Diagnostic Part

This part includes the diagnostic variables read by the user.

These variables are located at the high logical addresses (%Irc3F, %Irc3E, etc.).

The structure is the same for all types of cards.

#### 3.7.1.2 Input Image Part

This part contains the images of the card inputs. The input images are located at the low logical addresses (%Irc00, %Irc01, etc.). The structure depends on the type of card.

### 3.7.2 Structure of Write Variables %Qrc

The block of %Qrc write variables (for cards 0 to C) is divided into two parts:

- the card configuration part,
- the card image part.

#### 3.7.2.1 Card Configuration Part

This part includes the configuration variables written by the user.

These variables are located at the high logical addresses (%Qrc3F, %Qrc3E, etc.).

The configuration of the I/O cards must be programmed in initialisation task %INI.

The monitor will read the configuration at the end of the %INI task. Future configuration changes are therefore ignored by the monitor.

The structure is the same for all types of cards.

#### 3.7.2.2 Output Image Part

This part includes the images of the card outputs.

The output images are located at the low logical addresses (%Qrc00, %Qrc01, etc.).

The structure depends on the card type. Refer to the following sections for the structure of each card.

### 3.7.3 Card Diagnostic Variables

#### 3.7.3.1 Card Identifier %Irc3E.W

This word is written by the machine processor after interrogating the card.

%Irc3E.W == 0x700 indicates the card is not present.

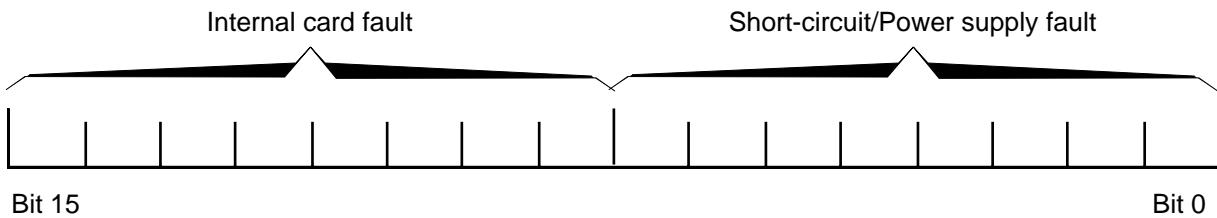
Example:

%I123E.W Contains the identifier of card 2 of rack 1.

### 3.7.3.2 Card Status %Irc3C.W

This word informs the user of the internal status of the card. This functionality is available only on the 32I/24O cards and 32-24 I/O, 64-48 I/O extension cards of the machine operator panel.

Register %Irc3C.W has the following format:



If no fault is detected, this register contains 0x00FF.

The internal card status is checked periodically. The programmer can set the period by function DIAGIQ().

If a problem is detected, the register is no longer updated. The user must force the register to 0x00FF to restart update.

**REMARK** *If an internal card fault is detected then the general I/O card error bit %R97F.2 (DEFCARTE) is set.*

### 3.7.3.3 Dialogue Error Counter %Irc3A.W

This word is incremented whenever a link error or card error is detected when polling a card. This counter is blocked at 0x7FFF.

### 3.7.3.4 Bus Status %Irc39.B

This byte informs the user of the status of the serial I/O bus link:

- 0 operation OK,
- 1 no echo frame,
- 2 check-sum error on echo frame,
- 3 no response frame,
- 4 check-sum error on response frame,
- 5 optical fibre cut,
- 6 other errors.

The card internal fault bits include the input line fault bits and output line fault bits (refer to the detailed description for each card supporting this functionality).

If the input link bits are set, the state of the corresponding power supply fault bits is not significant.

If the output link bits are set, the state of the corresponding short circuit fault bits is not significant.

**REMARK** *If four consecutive transmission errors occur on the same card, the general serial I/O bus link fault bit %R97F.0 (DEFBUS) is set and the watchdog is reset. If the transmission errors occur during machine processor initialisation, the general serial I/O bus link fault bit %R97F.0 (DEFBUS) is set and the watchdog is not set. If the transmission errors occur during a CPU reset, the general serial I/O bus link fault bit M97F.0 (DEFBUS) is set and the watchdog is disabled.*

### 3.7.4 Card Configuration Variables

#### 3.7.4.1 Card Identifier %Qrc3E.W

This field indicates the type of card the user expects to find in the location of rack r and card slot c. It must be programmed in an %INI task.

It is used to check whether the card/rack configuration of an application is correct. The check is made by comparing this identifier with the values read in %Irc3E.W.

**REMARK** *If there is a difference between the specified configuration %Qrc3E.W and the actual configuration %Irc3E.W, the general I/O card configuration error bit %R97F.1 (DEFCONF) is set, the inputs and outputs are no longer refreshed and the watchdog is reset.*

%Qrc3E.W is initialised with 0x700. This value indicates that the card is not configured. In this case, the monitor processes the card only if it is present in the rack.

#### Example

%Q123E.W == Contains the identifier of the card expected in slot 2 of rack 1.

#### 3.7.4.2 Logical/Geographic Address Option %Qrc3D.B

This option is used to choose the physical card associated with variables %Irc and %Qrc. This facilitates management of physical changes of the system since it is not necessary to change the input/output variables in the programme.

If byte %Qrc3D.B == r'c', the physical card associated with variables %Irc and %Qrc is card c' of rack r'.

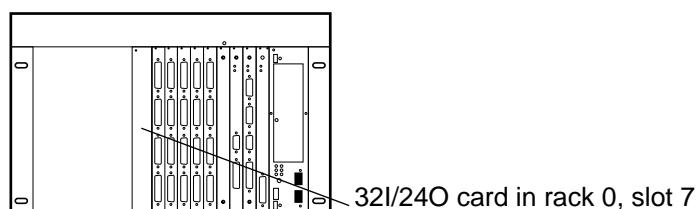
%Qrc3D.B must be initialised with r'c' in task %INI since the system only reads %Qrc3D.B on return from %INI.

When r' does not indicate a configured rack or c' does not indicate a configured card (ERROR\_CONFIG\_SBCE error), the general I/O card configuration error bit %R97F.1 (DEFCONF) is set, the inputs and outputs are no longer refreshed and the watchdog is reset.

As the geographic identification of the bus is carried out before task %INI, the user can use word %Irc3E.W (card identifier) in this task. If the logical addressing option (%Qrc3D.B) is used, the identifier read on the bus is moved accordingly in the input/output table.

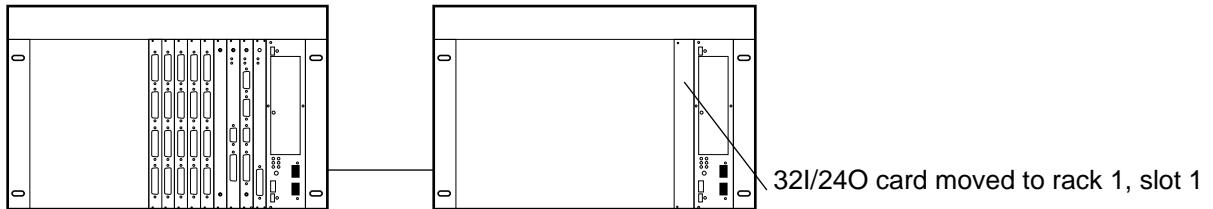
#### Example

In the basic configuration, there is a 32I/24O card in slot 7 of the main rack.



For this card, the user programme is written with variables %I07xx.x and %Q07xx.x.

A configuration change requires moving the card from rack 0, slot 7 to extension rack 1, slot 1.



To avoid having to change the user programme, programme %INI as follows:

%Q073D.B = 0x11

%Q073E.W = 0x1500 (32I/24O card identifier)

*REMARK The card identifier must be programmed.*

This means that:

- 0x07xx is the logical address
- 0x11xx is the geographic address (physical @).

#### 3.7.4.3 Card Priority %Qrc3C.B

Must be programmed in the %INI task. This byte sets the card priority. It is used to associate a card with a systematic task %TS0, %TS1 to %TS4 or %TS5.

This makes it possible to decrease the systematic processing performed each RTC.

Byte value	Processing frequency
0	The card is processed once each RTC
1	The card is processed every 5 RTCs in phase with %TS1
2	The card is processed every 5 RTCs in phase with %TS2
3	The card is processed every 5 RTCs in phase with %TS3
4	The card is processed every 5 RTCs in phase with %TS4
5	The card is processed every 5 RTCs in system task %TS5

The inputs of cards with priority  $i = 1, 2, 3, 4$  are read before the call to %TSi.

The outputs of cards with priority  $i = 1, 2, 3, 4$  are written at the end of %TSi.

If the priority byte is not between 0 and 5, the card is not periodically refreshed by the monitor. It can however be accessed by explicit read and write functions (see Sec. 10.2, read\_i(...)function and 10.3, write\_q(...)function).

The system initialises the priority byte with the default value of 0.

### 3.7.4.4 Watchdog %Qrc3B.0

Must be programmed in the %INI task. When this bit is set, it indicates that output %Qrc00.0 of this card is a watchdog output.

Two watchdogs are allowed. The monitor scans all of variables %Qrc3B.0 and selects the first two watchdogs programmed in increasing order (r,c).

If the watchdog is not initialised, the general I/O card configuration error bit %R97F.1 (DEFCONF) is set, the inputs and outputs are no longer refreshed and the watchdog is reset.

### 3.7.4.5 NC Access Enable %Qrc3B.1

Enables or disables access to the output cards (by parameters E33xxx) and the input cards (by parameters E43xxx) by the part programmes.

The variable equals 0 to inhibit access to the card by the part programme.

The variable equals 1 to enable access to the card by the part programme.

Variable %Qrc3B.1 is set to a default value of 0 by the monitor.

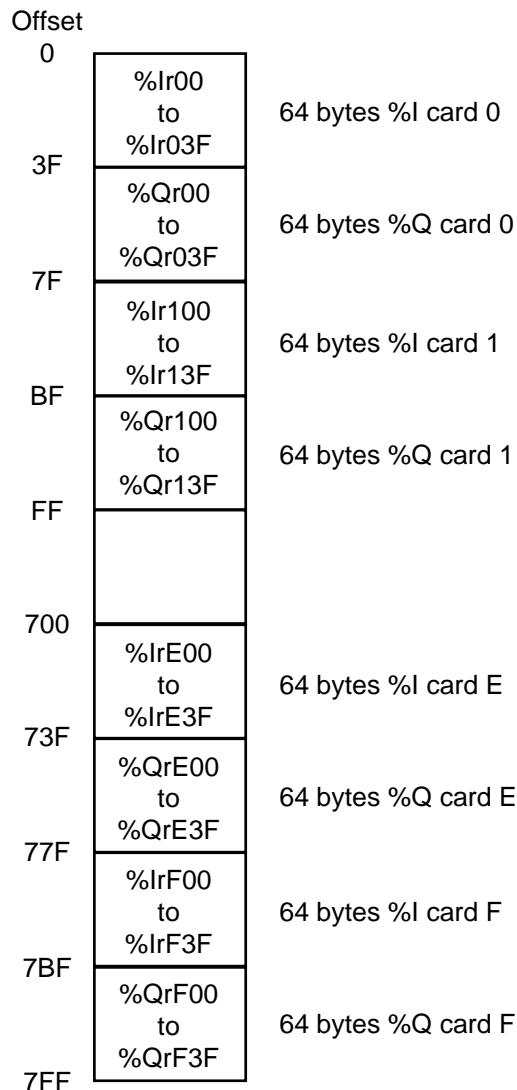
*REMARK %Qrc3B.1 must be programmed in %INI. Depending on the state of variable %W900.0, access to the outputs is enabled or inhibited by E33xxx.*

### 3.7.5 Physical Organisation of Variables %I and %Q

Variables %I and %Q are organised as memory blocks of 64 %I bytes followed by 64 %Q bytes corresponding to one card then to the next card until reaching the last card of the rack.

The racks are consecutive and contiguous (from rack 0 to rack 6).

#### 3.7.5.1 Physical Organisation of Variables %I and %Q of Rack r



### 3.7.5.2 Physical Organisation of Variables %I and %Q of the Different Racks

Offset	
0	Rack 0      16 cards ( $64 \%I + 64 \%Q$ ) = 2 kbytes
7FF	Rack 1      16 cards ( $64 \%I + 64 \%Q$ ) = 2 kbytes
FFF	Rack 2      16 cards ( $64 \%I + 64 \%Q$ ) = 2 kbytes
17FF	Rack 3      16 cards ( $64 \%I + 64 \%Q$ ) = 2 kbytes
1FFF	Rack 4      16 cards ( $64 \%I + 64 \%Q$ ) = 2 kbytes
27FF	Rack 5      16 cards ( $64 \%I + 64 \%Q$ ) = 2 kbytes
2FFF	Rack 6      16 cards ( $64 \%I + 64 \%Q$ ) = 2 kbytes
37FF	

### 3.7.6 Rack and Card Identifiers

#### 3.7.6.1 Card Identifiers

##### 1060 Rack

Card type	Value of %Irc3E.W and Qrc3BE.W
32-input card	0x0A00
V2 32-Input Card	0x0A10
32-output card	0x0100
V2 32-Output Card	0x0110
32-input/24-output card	0x1500
32-24 I/O card	0x0F00
80 mA 32-24 I/O Card	0x0F10
64-48 I/O card	0x0300
80 mA 64-48 I/O Card	0x0310
Machine panel	0x02C0
Machine panel with extension	0x0200
No card	0x0700

##### 1020/1040/1050 Cards

Card type	Value of %Irc3E.W and Qrc3E.W
80 mA 32-24 I/O Card	0x2100
80 mA 64-48 I/O Card	0x2000

#### 3.7.6.2 Rack Identifiers

##### 1060 Rack

**REMARK** The rack hardware components (power supply + metal work + bus) correspond to card 0.

Rack type	No. of cards	Power supply	Optical fibre	Value of identifier %Ir03E.W
Main	8	130w	Yes	0x0
Main	8	130w	No	0x80
Main	8	60w	Yes	0x10
Main	8	60w	No	0x90
Main	4	130w	Yes	0x3000
Main	4	130w	No	0x3080
Main	4	60w	Yes	0x3010
Main	4	60w	No	0x3090
Extension 12	12	130w		0x1000
Extension 12	12	60w		0x1010
Extension 2	2			0x2020

## 1020/1040 Rack

Optical fibre	Value of identifier %Ir03E.W
Yes	0x40B0
No	0x4030

### 3.7.7 Image Part of 32 Discrete Input Card

32-input card identifier %Irc3E.W == 0x0A00.

V2 32-input card identifier %Irc3E.W == 0x0A10.

Variable type	Input type	Variables
%Irc00	Discrete inputs 0 to 7	%Irc00.0 (Input 00.0) to %Irc00.7 (Input 00.7)
%Irc01	Discrete inputs 8 to 15	%Irc01.0 (Input 01.0) to %Irc01.7 (Input 01.7)
%Irc02	Discrete inputs 16 to 23	%Irc02.0 (Input 02.0) to %Irc02.7 (Input 02.7)
%Irc03	Discrete inputs 24 to 31	%Irc03.0 (Input 03.0) to %Irc03.7 (Input 03.7)

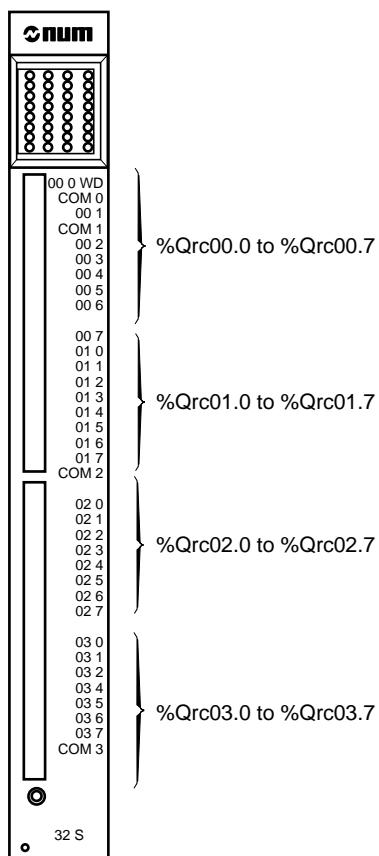


### 3.7.8 Image Part of the 32 Discrete Output Card

32-output card identifier %Irc3E.W == 0x0100.

V2 32-output card identifier %Irc3E.W == 0x0110.

Variable type	Output type	Variables
%Qrc00	Discrete outputs 0 to 7	%Qrc00.0 (Output 00.0) to %Qrc00.7 (Output 00.7)
%Qrc01	Discrete outputs 8 to 15	%Qrc01.0 (Output 01.0) to %Qrc01.7 (Output 01.7)
%Qrc02	Discrete outputs 16 to 23	%Qrc02.0 (Output 02.0) to %Qrc02.7 (Output 02.7)
%Qrc03	Discrete outputs 24 to 31	%Qrc03.0 (Output 03.0) to %Qrc03.7 (Output 03.7)



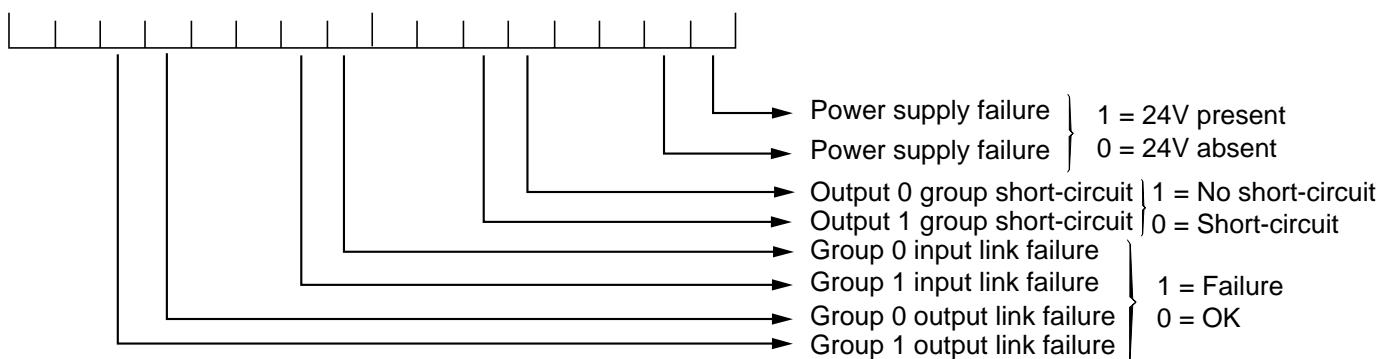
### 3.7.9 Image Part of the 32-24 I/O Discrete I/O and 32-24 I/O Cards

32I 24O card Identifier %Irc3E.W == 0x1500.

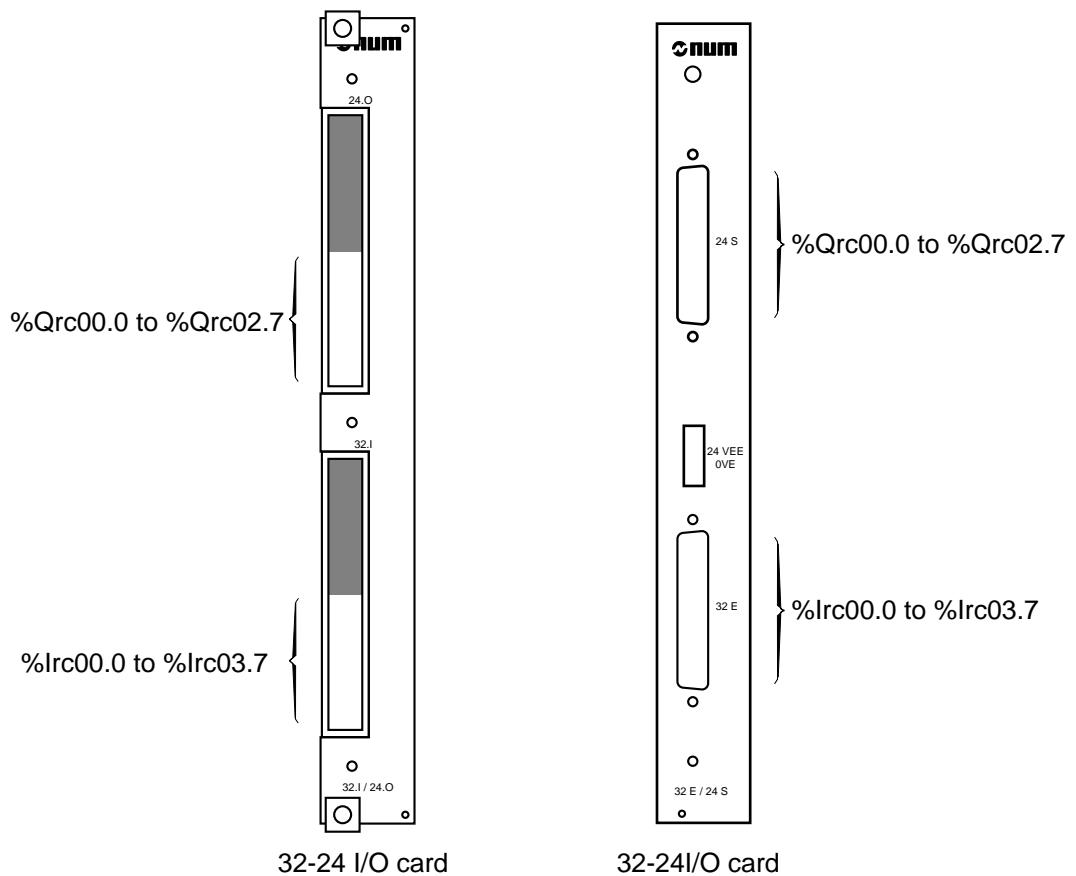
32-24 I/O card identifier %Irc3E.W == 0x1500.

80 mA 32-24 I/O card identifier %Irc3E.W == 0x0F10

#### Detail of Register %Irc3C.W



Variable type	Input or output type
%Irc00	Discrete inputs 0 to 7
%Irc01	Discrete inputs 8 to 15
%Irc02	Discrete inputs 16 to 23
%Irc03	Discrete inputs 24 to 31
%Qrc0 0	Discrete outputs 0 to 7
%Qrc01	Discrete outputs 8 to 15
%Qrc02	Discrete outputs 16 to 23

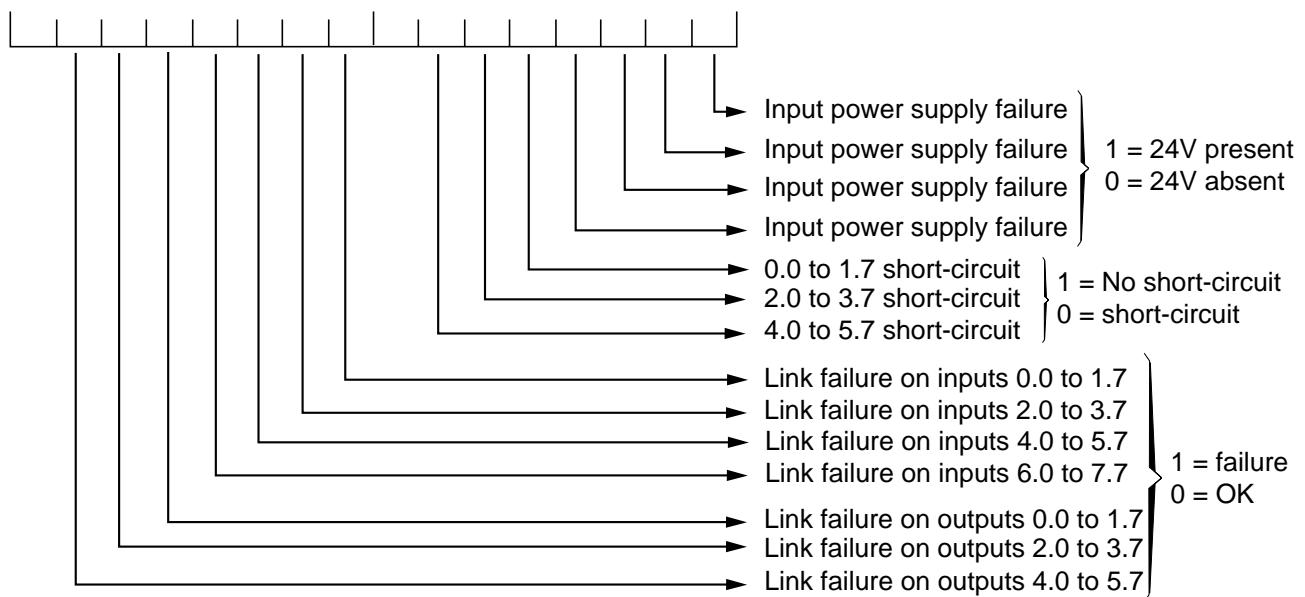


### 3.7.10 Image Part of the 64-48 I/O Card

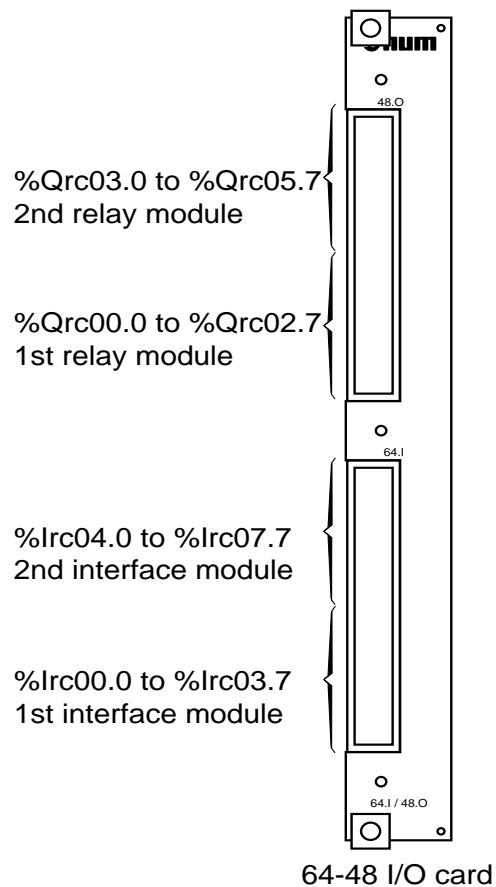
64-48 I/O card identifier %Irc3E.W == 0x0300.

80 mA 64-48 I/O card identifier %Irc3E.W == 0x0310.

#### Format of Register %Irc3C.W



Variable type	Input or output type
%Irc00	Discrete inputs TOR 0 to 7
%Irc01	Discrete inputs TOR 8 to 15
%Irc02	Discrete inputs TOR 16 to 23
%Irc03	Discrete inputs TOR 24 to 31
%Irc04	Discrete inputs TOR 32 to 39
%Irc05	Discrete inputs TOR 40 to 47
%Irc06	Discrete inputs TOR 48 to 55
%Irc07	Discrete inputs TOR 56 to 63
%Qrc00	Discrete outputs TOR 0 to 7
%Qrc01	Discrete outputs TOR 8 to 15
%Qrc02	Discrete outputs TOR 16 to 23
%Qrc03	Discrete outputs TOR 24 to 31
%Qrc04	Discrete outputs TOR 32 to 39
%Qrc05	Discrete outputs TOR 40 to 47

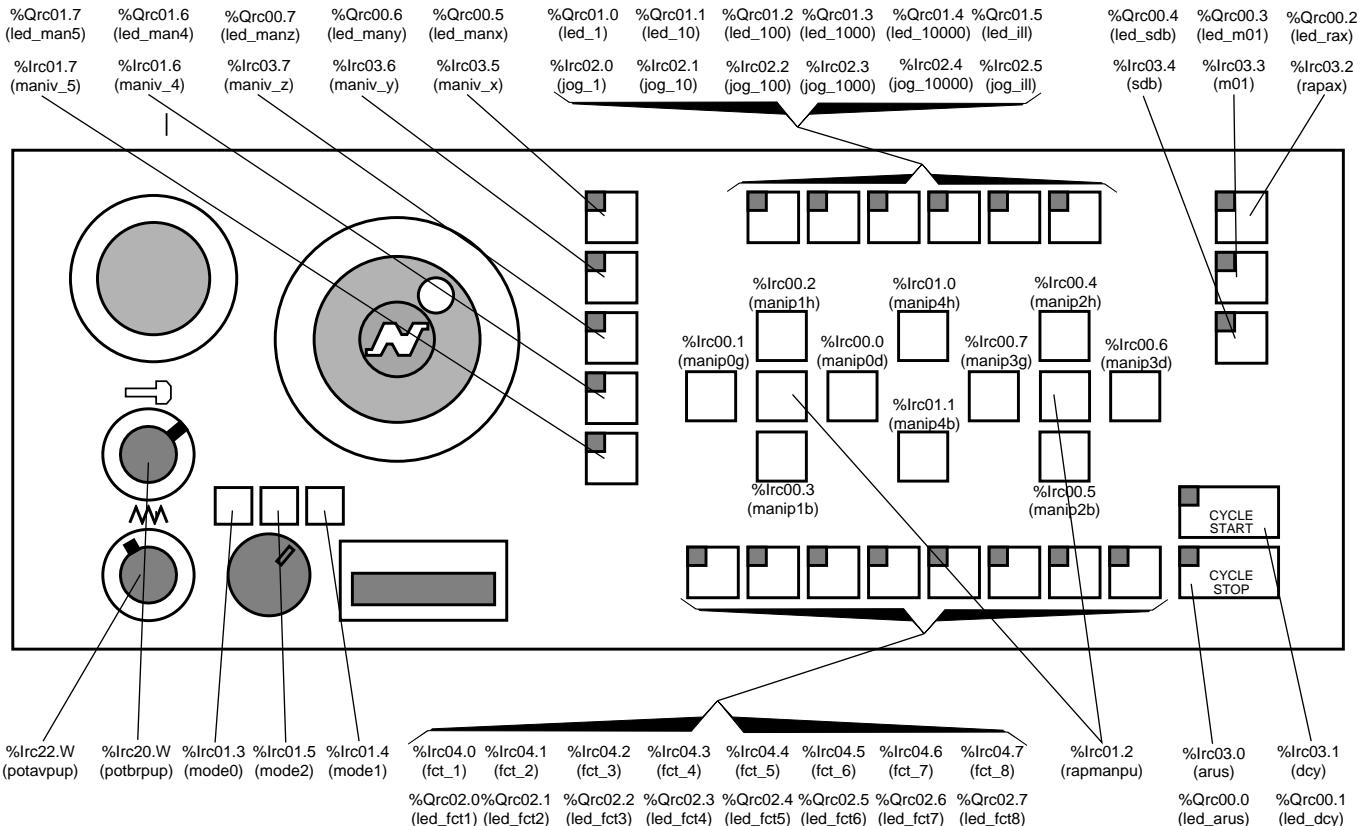


64-48 I/O card

### 3.7.11 Image Part of the Basic Operator Panel Card

Card Identifier %Irc3E.W == 0x2C0.

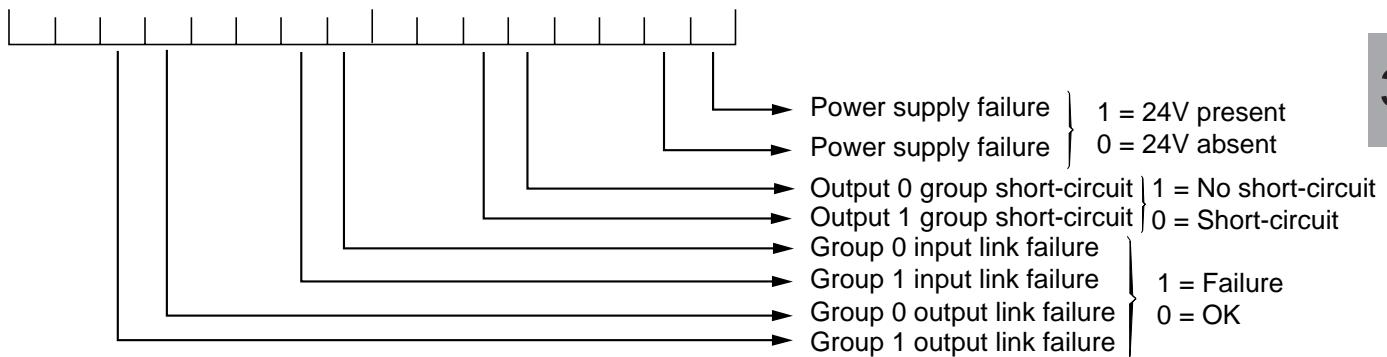
Variable type	Input or output type	Variables
%Irc00	Discrete inputs 0 to 7	%Irc00.0 (Input 0) to %Irc00.7 (Input 7)
%Irc01	Discrete inputs 8 to 15	%Irc01.0 (Input 8) to %Irc01.7 (Input 15)
%Irc02	Discrete inputs 16 to 23	%Irc02.0 (Input 16) to %Irc02.7 (Input 23)
%Irc03	Discrete inputs 24 to 31	%Irc03.0 (Input 24) to %Irc03.7 (Input 31)
%Irc04	Discrete inputs 32 to 39	%Irc04.0 (Input 32) to %Irc04.7 (Input 39)
%Irc20.W	Analogue input 0	
%Irc22.W	Analogue input 1	
%Qrc00	Discrete outputs 0 to 7	%Qrc00.0 (Output 0) to %Qrc00.7 (Output 7)
%Qrc01	Discrete outputs 8 to 15	%Qrc01.0 (Output 8) to %Qrc01.7 (Output 15)
%Qrc02	Discrete outputs 16 to 23	%Qrc02.0 (Output 16) to %Qrc02.7 (Output 23)



### 3.7.12 Image Part of the Extension Operator Panel Card

Card Identifier %Irc3E.W == 0x200.

#### Format of register %Irc3C.W



Variable type	Input or output type	Variables
%Irc00	Discrete inputs 0 to 7	%Irc00.0 (Input 0) to %Irc00.7 (Input 7)
%Irc01	Discrete inputs 8 to 15	%Irc01.0 (Input 8) to %Irc01.7 (Input 15)
%Irc02	Discrete inputs 16 to 23	%Irc02.0 (Input 16) to %Irc02.7 (Input 23)
%Irc03	Discrete inputs 24 to 31	%Irc03.0 (Input 24) to %Irc03.7 (Input 31)
%Irc04	Discrete inputs 32 to 39	%Irc04.0 (Input 32) to %Irc04.7 (Input 39)
%Irc10	Discrete inputs 40 to 47	%Irc10.0 (Input 40) to %Irc10.7 (Input 47)
%Irc11	Discrete inputs 48 to 55	%Irc11.0 (Input 48) to %Irc11.7 (Input 55)
%Irc12	Discrete inputs 56 to 63	%Irc12.0 (Input 56) to %Irc12.7 (Input 63)
%Irc13	Discrete inputs 64 to 71	%Irc13.0 (Input 64) to %Irc13.7 (Input 71)
%Irc20.W	Analogue input 0	
%Irc22.W	Analogue input 1	
%Qrc00	Discrete outputs 0 to 7	%Qrc00.0 (Output 0) to %Qrc00.7 (Output 7)
%Qrc01	Discrete outputs 8 to 15	%Qrc01.0 (Output 8) to %Qrc01.7 (Output 15)
%Qrc02	Discrete outputs 16 to 23	%Qrc02.0 (Output 16) to %Qrc02.7 (Output 23)
%Qrc10	Discrete outputs 24 to 31	%Qrc10.0 (Output 24) to %Qrc10.7 (Output 31)
%Qrc11	Discrete outputs 32 to 39	%Qrc11.0 (Output 32) to %Qrc11.7 (Output 39)
%Qrc12	Discrete outputs 40 to 47	%Qrc12.0 (Output 40) to %Qrc12.7 (Output 47)

**REMARK** The panel output LED test must not be conducted in a single operation. In the PLC programme, first test half the LEDs then test the other half.

24-output extension (pinout)

24 VS.0	19
%Qrc10.0	37
%Qrc10.1	18
%Qrc10.2	36
%Qrc10.3	17
%Qrc10.4	35
%Qrc10.5	16
COMMON	34
%Qrc10.6	33
COMMON	14
%Qrc10.7	32
%Qrc11.0	13
%Qrc11.1	31
%Qrc11.2	12
%Qrc11.3	30
%Qrc11.4	28
COMMON	9
%Qrc11.5	8
%Qrc11.6	5
%Qrc11.7	7
%Qrc12.0	4
%Qrc12.1	25
%Qrc12.2	24
%Qrc12.3	20
%Qrc12.4	21
%Qrc12.5	22
%Qrc12.6	23
%Qrc12.7	1
24 VS.1	2
COMMON	3

32-input extension (pinout)

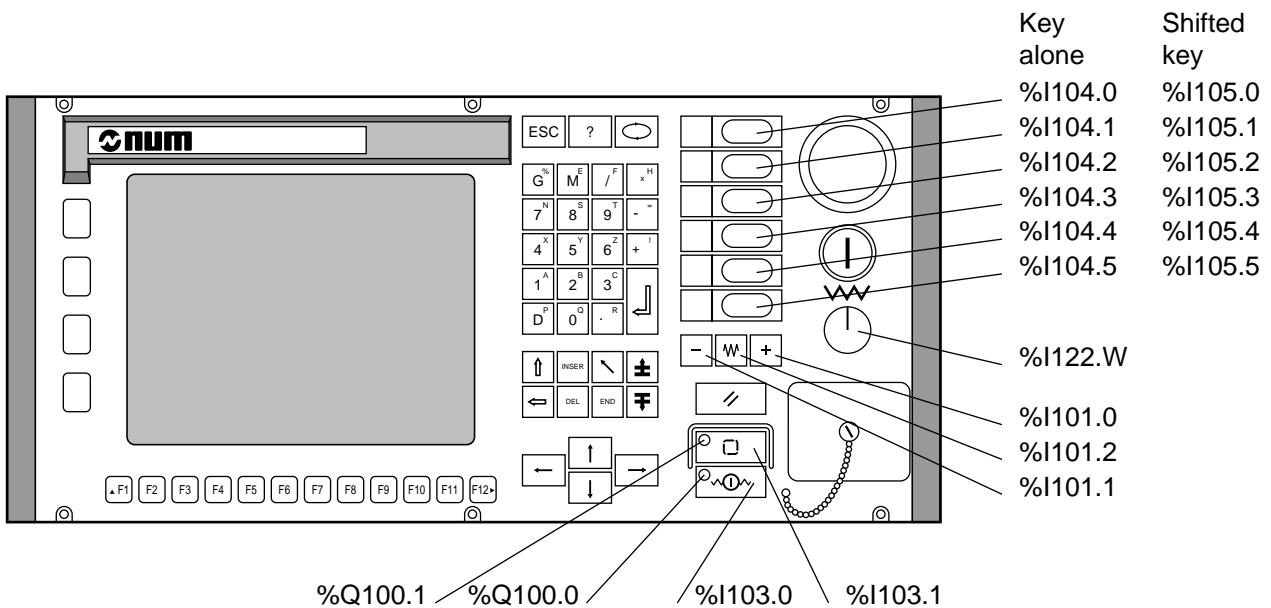
%lrc10.0	1
%lrc10.1	20
%lrc10.2	2
%lrc10.3	21
%lrc10.4	3
%lrc10.5	22
%lrc10.6	4
%lrc10.7	23
COMMON	5
%lrc11.1	24
%lrc11.2	6
%lrc11.3	25
%lrc11.4	7
%lrc11.5	26
%lrc11.6	8
%lrc11.7	27
COMMON	9
%lrc12.0	28
%lrc12.1	29
%lrc12.2	11
%lrc12.3	30
%lrc12.4	12
%lrc12.5	31
%lrc12.6	13
%lrc12.7	32
COMMON	14
%lrc13.0	33
%lrc13.1	15
%lrc13.2	34
%lrc13.3	16
%lrc13.4	35
%lrc13.5	17
%lrc13.6	36
%lrc13.7	18
COMMON	37
24 VE	19
	10

### 3.7.13 Image Part of the Compact Panel

#### 3.7.13.1 Image Part of the Compact Panel in the Exchange Area

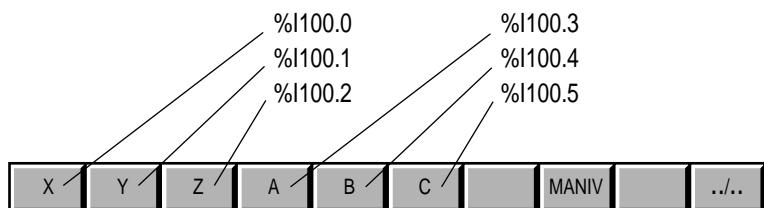
Variable type	Input or output type	Variables
%I100.B	Axis selection by JOG keys	%I100.0 to %I100.5
%I101.B	+, - and fast JOG keys	%I101.0 to %I101.2
%I103.B	Cycle Stop and Cycle Start keys	%I103.0 (Cycle Stop) and %I103.1 (Cycle Start)
%I104.B	Programmable keys 1 to 6	%I104.0 (key 1) to %I104.5 (key 6)
%I105.B	Shifted programmable keys 1 to 6	%I105.0 (key 1) to %I105.5 (key 6)
%I122.W	Potentiometer analogue input	
%Q100.B	Cycle Stop and Cycle Start LEDs	%Q100.0 (Cycle Stop) and %Q100.1 (Cycle Start)
%Q102.B	Programmable key LEDs 1 to 6	%Q102.0 (LED 1) to %Q102.5 (LED 6)
%Q103.B	Shifted programmable key LEDs 1 to 6	%Q103.0 (LED 1) to %Q103.5 (LED 6)

#### 3.7.13.2 Image Part of the Compact Panel



#### 3.7.13.3 Image of the JOG Softkeys

The compact panel has special softkeys including the new JOG keys used to select the axis controlled by the jogs:

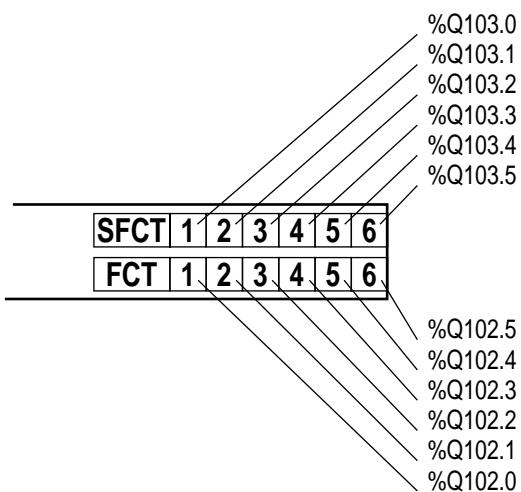


These softkeys are accessed by keys: then **JOG** (F7).

The axes whose names appear in the keys are those that were defined in machine parameter P9 (see Parameter Manual). They appear in the order in which they were defined.

### 3.7.13.4 Programmable Key Image Leds

The functions activated by the programmable keys are shown by LEDs in the Status window:



The bottom row of LEDs represents the programmable keys alone and the top row represents the shifted programmable keys.

## 3.8 CNC I/O Interface Family %R and %W

### 3.8.1 Inputs from the CNC %R0 to %R7F

#### 3.8.1.1 Keyboard Characters: %R0.W

Variable	Mnemonic	Description
%R0.W	CARCLAV	Receives the characters entered from the keyboard at a rate of %TS5, i.e. one character every 5 RTCs (see Sec. 8.1.2).

#### 3.8.1.2 Machine Status: %R2.W

Variable	Mnemonic	Description
%R2.7	E_M01	Optional programme stop enabled Image of field M01 of the NC status window. Indicates inclusion of optional programme stops in a part programme. The state of this bit can be changed by pressing the M01 key on the panel or reading C_M01 = 1.
%R2.6	E_SLASH	Block skip enabled Image of the «//» field of the NC status window. Indicates inclusion of block skips in a part programme. The state of the bit can be changed by pressing the «//» key on the panel or reading C_SLASH = 1.
%R2.5	E_INTERV	Cycle hold state After a cycle stop, the switch to cycle hold occurs the first time the AXIS RECALL key on the machine panel is pressed. Set by read of C_RAX = 1 by the NC. Reset by read of C_RAX = 0 by the NC.
%R2.4	S_RECUL	Backward/forward movement on path Set for backward or forward CNC movement Reset to cancel this state.
%R2.1	E_NMAUTO	N/M AUTO functionality Set to indicate that the N/M functionality (2/3, 3/5, etc.) is enabled.
%R3.7	E_OPER	Programme stop Indicates a programme stop caused by M00 or enabled M01. Set by M00 or M01. Reset by the CYCLE key on the machine operator panel (C_CYCLE = 1).
%R3.6	E_DEFNC	CNC fault Image of field «CN??» in the CNC status window. Indicates a machine error or a part programme error. The machine error number is contained in ERRMACH. Set by occurrence of a machine error (E30-E33, E36, E40-E71) or a part programming error. Reset by the RAZ key on the panel, C_RAZ = 1.
%R3.4	E_DGURG	General emergency retract Indicates execution of an emergency retract programme. Set by read of C_DGURG = 1 by the NC if the emergency retraction programme is enabled. Reset by detection of M00 or M02.

Variable	Mnemonic	Description
%R3.3	E_RAX	<p>General axis recall</p> <p>In CYHLD mode and at the end of SEARCH, indicates axis recall is enabled.</p> <p>Set by read of C_RAX = 1 by the NC.</p> <p>Reset by read of C_RAX = 0 by the NC.</p>
%R3.2	E_CYCLE	<p>Cycle in progress</p> <p>Set by the CYCLE key on the machine panel (C_CYCLE = 1).</p> <p>Reset by the RAZ key on the panel, C_RAZ = 1 or at the end of execution of the programme (M02).</p>
%R3.1	E_ARUS	<p>Cycle stop</p> <p>Indicates the CYHLD state of the system (programme stopped during execution and enabling of axis jogs).</p> <p>Set by the CYHLD key on the machine operator panel (C_ARUS = 1).</p> <p>Reset by the CYCLE key on the machine operator panel (C_CYCLE = 0).</p>
%R3.0	E_RAZ	<p>CNC reset in progress</p> <p>Pulse bit with a duration of 100 ms indicating a system reset.</p> <p>While this pulse is high, data from the automatic control function are ignored.</p> <p>Set by the RAZ key on the panel, a reset request from the automatic control function (C_RAZ = 1), at the end of execution of a part programme (M02) or when the NC is turned on.</p> <p>This variable is reset after 100 ms.</p>

### 3.8.1.3 CNC Status: %R4.W

Variable	Mnemonic	Description
%R5.7	E_TRANSP	<p>Transparent mode</p> <p>Gives access by the machine processor to the CNC operator panel for display of data (data tables, etc.). Machining may be in progress.</p> <p>Set by enabling the «TRANSPARENT MODE» screen page.</p> <p>Reset by clearing the «TRANSPARENT MODE» screen page.</p>
%R5.5	E_PPP	<p>Drip feed mode ready</p> <p>Indicates that the NC is ready to operate in drip feed mode or that the machining ordered by the automatic control function will be carried out in drip feed mode. In the second case, PROGDEM must be initialised with -2 (0xFFFF).</p> <p>Set after «CHOICE OF CURRENT PROGRAMME» and keyboard entry of PPR or PPL followed by ENTER.</p> <p>Reset after «CHOICE OF CURRENT PROGRAMME» and keyboard entry of -PPR or -PPL followed by ENTER.</p>
%R5.1	E_PROG	<p>Active programme</p> <p>Indicates that a part programme is being executed in AUTO, SINGLE, MDI or DRYRUN mode.</p> <p>Set by the first action on the CYCLE key on the machine operator panel (C_CYCLE = 1).</p> <p>Reset by detection of M00 (programme stop), M01 (optional stop), M02 (end of programme); the RAZ key on the operator panel, variable C_RAZ = 1 and at power on.</p>

Variable	Mnemonic	Description
%R5.0	E_CNPRET	CNC ready Indicates that power can be applied to the machine. Set at power on and by the RAZ key on the operator panel, variable C_RAZ = 1. Reset by detection of an excessive following error on an axis, poor signal or encoder complementary channel fault detected on an axis.

### 3.8.1.4 Axes in Motion: %R6L

Variable	Mnemonic	Description
%R6.7 to %R6.0	AXMVT31 to AXMVT24	axis 31 in motion to axis 24 in motion Indicates axes 24 to 31 in motion during execution of a block in a part programme or in MDI mode. Set at the start of execution of the block. Reset at the end of execution of the block if it includes M00 or M01, at the end of execution of the block in MDI mode, before carrying out axis clamping, by the RAZ key on the operator panel, variable C_RAZ = 1.
%R7.7 to %R7.0	AXMVT23 to AXMVT16	axis 23 in motion to axis 16 in motion Indicates axes 16 to 23 in motion during execution of a block in a part programme or in MDI mode. Set at the start of execution of the block. Reset at the end of execution of the block if it includes M00 or M01, at the end of execution of the block in MDI mode, before carrying out axis clamping, by the RAZ key on the operator panel, variable C_RAZ = 1.
%R8.7 to %R8.0	AXMVT15 to AXMVT8	axis 15 in motion to axis 8 in motion Indicates axes 8 to 15 in motion during execution of a block in a part programme or in MDI mode. Set at the start of execution of the block. Reset at the end of execution of the block if it includes M00 or M01, at the end of execution of the block in MDI mode, before carrying out axis clamping, by the RAZ key on the operator panel, variable C_RAZ = 1.
%R9.7 to %R9.0	AXMVT7 to AXMVT0	axis 7 in motion to axis 0 in motion Indicates axes 0 to 7 in motion during execution of a block in a part programme or in MDI mode. Set at the start of execution of the block. Reset at the end of execution of the block if it includes M00 or M01, at the end of execution of the block in MDI mode, before carrying out axis clamping, by the RAZ key on the operator panel, variable C_RAZ = 1.

### 3.8.1.5 Axes Initialised (origin setting completed): %RA.L

Variable	Mnemonic	Description
%RA.7 to %RA.0	AXINI31 to AXINI24	axis 31 initialised to axis 24 initialised Indicates the axes on which origin setting is completed. Reset when origin setting is completed on the corresponding axis. Set by a system restart (origin setting not completed).
%RB.7 to %RB.0	AXINI23 to AXINI16	axis 23 initialised to axis 16 initialised Indicates the axes on which origin setting is completed. Reset when origin setting is completed on the corresponding axis. Set by a system restart (origin setting not completed).
%RC.7 to %RC.0	AXINI15 to AXINI8	axis 15 initialised to axis 8 initialised Indicates the axes on which origin setting is completed. Reset when origin setting is completed on the corresponding axis. Set by a system restart (origin setting not completed).
%RD.7 to %RD.0	AXINI7 to AXINI0	axis 7 initialised (origin setting completed) to axis 0 initialised (origin setting completed) Indicates the axes on which origin setting is completed. Reset when origin setting is completed on the corresponding axis. Set by a system restart (origin setting not completed).

### 3.8.1.6 External Parameters E10000 to E10031: %RE.L

External parameters E100xx are written by the part programme. They are visible to the m/c processor programme.

The parameters are used to make part programme status available to the m/c processor.

Variable	Mnemonic	Variable	Mnemonic
%R11.0	E10000	%RF.0	E10016
%R11.1	E10001	%RF.1	E10017
%R11.2	E10002	%RF.2	E10018
%R11.3	E10003	%RF.3	E10019
%R11.4	E10004	%RF.4	E10020
%R11.5	E10005	%RF.5	E10021
%R11.6	E10006	%RF.6	E10022
%R11.7	E10007	%RF.7	E10023
%R10.0	E10008	%RE.0	E10024
%R10.1	E10009	%RE.1	E10025
%R10.2	E10010	%RE.2	E10026
%R10.3	E10011	%RE.3	E10027
%R10.4	E10012	%RE.4	E10028
%R10.5	E10013	%RE.5	E10029
%R10.6	E10014	%RE.6	E10030
%R10.7	E10015	%RE.7	E10031

### 3.8.1.7 Spindle Status: %R12.W

Variable	Mnemonic	Description
%R12.7	B4_ARR	Bit set when spindle 4 is stopped, i.e. its rotation speed is less than the setting of parameter E90343 (see Programming Manual)
%R12.6	B3_ARR	Bit set when spindle 3 is stopped, i.e. its rotation speed is less than the setting of parameter E90342 (see Programming Manual)
%R12.5	B2_ARR	Bit set when spindle 2 is stopped, i.e. its rotation speed is less than the setting of parameter E90341 (see Programming Manual)
%R12.4	B1_ARR	Bit set when spindle 1 is stopped, i.e. its rotation speed is less than the setting of parameter E90340 (see Programming Manual)
%R12.3	B4_ROT	Bit set to indicate that the rotation of spindle 4 is correct, i.e. that its rotation speed is within the speed tolerance interval specified in parameter E90353 (see Programming Manual)
%R12.2	B3_ROT	Bit set to indicate that the rotation of spindle 3 is correct, i.e. that its rotation speed is within the speed tolerance interval specified in parameter E90352 (see Programming Manual)
%R12.1	B2_ROT	Bit set to indicate that the rotation of spindle 2 is correct, i.e. that its rotation speed is within the speed tolerance interval specified in parameter E90351 (see Programming Manual)
%R12.0	B1_ROT	Bit set to indicate that the rotation of spindle 1 is correct, i.e. that its rotation speed is within the speed tolerance interval specified in parameter E90350 (see Programming Manual)
%R13.3	POSBR4	<p>Spindle 4 in position</p> <p>For a spindle indexing or synchronisation request, indicates that spindle 4 is in position or synchronised.</p> <p>Set when the required position is reached.</p> <p>Reset after leaving the required position, by oscillations and by cancellation of function M19.</p>
%R13.2	POSBR3	<p>Spindle 3 in position</p> <p>For a spindle indexing or synchronisation request, indicates that spindle 3 is in position or synchronised.</p> <p>Set when the required position is reached.</p> <p>Reset after leaving the required position, by oscillations and by cancellation of function M19.</p>
%R13.1	POSBR2	<p>Spindle 2 in position</p> <p>For a spindle indexing or synchronisation request, indicates that spindle 2 is in position or synchronised.</p> <p>Set when the required position is reached.</p> <p>Reset after leaving the required position, by oscillations and by cancellation of function M19.</p>
%R13.0	POSBR1	<p>Spindle 1 in position</p> <p>For a spindle indexing or synchronisation request, indicates that spindle 1 is in position or synchronised.</p> <p>Set when the required position is reached.</p> <p>Reset after leaving the required position, by oscillations and by cancellation of function M19.</p>

### 3.8.1.8 Type of Jog Increment: %R15.B

Variable	Mnemonic	Description
%R15.B	E_INCJOG	<p>Current jog increment  The value of this variable is the image of the current jog increment:</p> <ul style="list-style-type: none"> <li>0x0A Manual movement by <math>10^{-6}</math> inches</li> <li>0x09 Manual movement by <math>10^{-2}</math> <math>\mu\text{m}</math> or <math>10^{-5}</math> inches</li> <li>0x00 Manual movement by <math>10^1</math> <math>\mu\text{m}</math> or <math>10^{-4}</math> inches</li> <li>0x01 Manual movement by <math>1 \mu\text{m}</math> or <math>10^{-3}</math> inches</li> <li>0x02 Manual movement by <math>10 \mu\text{m}</math> or <math>10^{-2}</math> inches</li> <li>0x03 Manual movement by <math>100 \mu\text{m}</math> or <math>10^{-1}</math> inches</li> <li>0x04 Manual movement by <math>1000 \mu\text{m}</math> or 1 inches</li> <li>0x05 Manual movement by <math>10000 \mu\text{m}</math> or 1 inch</li> <li>0x06 Continuous jog</li> <li>0x08 Movement by handwheel</li> </ul> <p>The increment is in <math>\mu\text{m}</math> or inches depending on the value of variable C_UNIT.</p>

### 3.8.1.9 Current Mode: %R16.B

Variable	Mnemonic	Description
%R16.B	MODCOUR	<p>Current Mode  The value of the variable is the image of the current NC mode:</p> <ul style="list-style-type: none"> <li>0x00 Auto mode «AUTO»</li> <li>0x01 Single step mode «SINGLE»</li> <li>0x02 Manual data input mode «MDI»</li> <li>0x03 DRYRUN mode «DRYRUN»</li> <li>0x04 Sequence number search mode «SEARCH»</li> <li>0x05 Edit mode «EDIT»</li> <li>0x06 Test mode «TEST»</li> <li>0x07 Manual mode "MANUAL"</li> <li>0x08 Home mode «HOME»</li> <li>0x09 Origin shift mode «SHIFT»</li> <li>0x0A Tool setting mode «TL SET»</li> <li>0x0B No mode active</li> <li>0x0D Load mode «LOAD»</li> <li>0x0F Unload mode «UNLOAD»</li> <li>0x10 Mode specifying independent groups</li> </ul>

### 3.8.1.10 Other Variables

Variable	Mnemonic	Description
%R14.1	E_BAT	Battery status E_BAT = 0 Batteries OK E_BAT = 1 Replace batteries
%R14.0	SC_USED	Screen enabled in PCNC configuration Variable set to indicate that the screen is used by a user application (transparent mode inhibited). Variable reset to indicate that the screen is used by the NUM CNC application (transparent mode possible).
%R17.B	PGVISU	Displayed Page Number This variable is the image of the page displayed on the NC screen:  0x01 «DIR.» directory page 0x03 «PROG.» programme page 0x04 «INFO» information page 0x05 «L/@» programme variable page 0x06 «AXIS» current point page 0x07 «TOOLS» tool correction page 0x08 «PROCAM» graphic programming page 0x19 Load page during machining 0x1A Unload page during machining 0x09 «I/O» input/output page 0x0A «UTIL» utility page 0x15 «SHIFT» shift page 0x17 Edit mode page 0x0E Load mode page 0x11 Unload mode page 0x1B Transparent mode called directly by PUTKEY
%R18.B	ERRMACH	Machine Error Number This variable contains the machine error number detected by the system (errors 18, 30-33, 35, 36, 39-71, 210-241, 245, 300-331) as a decimal code.
<i>REMARK Refer to the Operator Manual for the list of machine errors.</i>		
%R19.B	ID_KB_CN	Operator panel or CNC active identifier In a multipanel configuration, gives the number of the active panel (0 to 7) In a multi CNC configuration, gives the number of the active CNC (0 to 4)
%R1A.W	PROGCOUR	Active Programme Number This variable contains the number of the active programme. The value 0xFFFF (-1) indicates there is no active programme. The value 0xFFFFE (-2) indicates selection of the drip feed mode.

### 3.8.1.11 Spindle Speed: %R1C.W to %R22.W

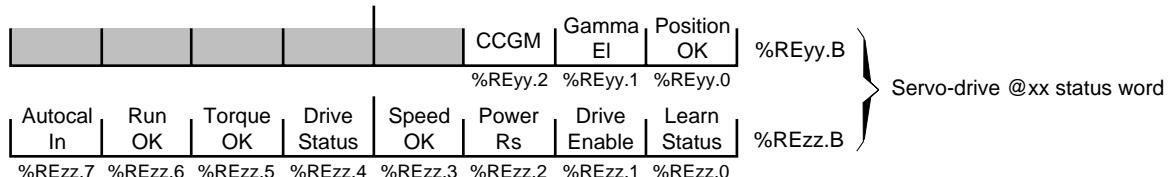
Variable	Mnemonic	Description
%R1C.W	VITBR1	Spindle 1 Speed Reference Contains the hexadecimal code of the spindle 1 servo-drive reference in the programmed speed range. The absence of functions M3 and M4 in the part programme forces the variable to zero.
%R1E.W	VITBR2	Spindle 2 Speed Reference Contains the hexadecimal code of the spindle 2 servo-drive reference in the programmed speed range. The absence of functions M3 and M4 in the part programme forces the variable to zero.
%R20.W	VITBR3	Spindle 3 Speed Reference Contains the hexadecimal code of the spindle 3 servo-drive reference in the programmed speed range. The absence of functions M3 and M4 in the part programme forces the variable to zero.
%R22.W	VITBR4	Spindle 4 Speed Reference Contains the hexadecimal code of the spindle 4 servo-drive reference in the programmed speed range. The absence of functions M3 and M4 in the part programme forces the variable to zero.

### 3.8.1.12 Axis Clamp: %R24.L

Variable	Mnemonic	Description
%R24.7 to %R24.0	AXBLK31 to AXBLK24	Axis 31 to axis 24 Bit set if the axis can be clamped Bit reset if the axis cannot be clamped A reset places the axes in the configuration specified in machine parameter P8
%R25.7 to %R25.0	AXBLK23 to AXBLK16	Axis 23 to axis 16 Bit set if the axis can be clamped Bit reset if the axis cannot be clamped A reset places the axes in the configuration specified in machine parameter P8
%R26.7 to %R26.0	AXBLK15 to AXBLK8	Axis 15 to axis 8 Bit set if the axis can be clamped Bit reset if the axis cannot be clamped A reset places the axes in the configuration specified in machine parameter P8
%R27.7 to %R27.0	AXBLK7 to AXBLK0	Axis 7 to axis 0 Bit set if the axis can be clamped Bit reset if the axis cannot be clamped A reset places the axes in the configuration specified in machine parameter P8

### 3.8.1.13 1050 Servo-Drive Status Word

For the digital servo-drive at address xx (xx between 00 and 31), the status word format is as follows:



Bit	Meaning	Value
%REzz.0	Learn Status	Reserved
%REzz.1	Drive Enable	0: servo-drive not enabled 1: servo-drive enabled
%REzz.2	Power Rs	0: bus voltage not present 1: bus voltage present
%REzz.3	Speed OK	0: speed not reached 1: speed reached
%REzz.4	Drive Status	0: servo-drive stopped 1: servo-drive start
%REzz.5	Torque OK	0: torque threshold not reached 1: torque threshold reached
%REzz.6	Run OK	0: motor stopped 1: motor running
%REzz.7	Autocalibration In	0: self-calibration completed 1: self-calibration in progress
%REyy.0	Position OK	0: position not reached 1: position reached
%REyy.1	Gamma EI	0: low speed range 1: high speed range
%REyy.2	CCGM	0: mechanical speed range not requested 1: mechanical speed range requested

### 3.8.2 Outputs to the CNC %W0 to %W7F

#### 3.8.2.1 Pulse Commands: %W2.W

Variable	Mnemonic	Description
%W2.3	CHG_OPDC	If CHG_OPDC is set, the dynamic operators are reloaded in C by a general CNC reset.
%W2.2	C_INDG	Common group/independent group switchover This command is latched. A change of state of C_INDG is detected only on a common reset requested by the PLC. C_INDG = 0: Common groups C_INDG = 1: Independent groups
%W2.1	C_NMAUTO	N/M AUTO functionality Set to enable the AUTO N/M (2/3, 3/5, et.) functionality. This command is operative when command C_CYCLE goes low.
%W2.0	KB_INIT	Keyboard initialisation Set to enable identification of the configuration of keyboards and interconnected CNCs. Identification must be carried out whenever the configuration is modified. After identification, keyboard 1 is assigned to CNC 1.
%W3.7	C_M01	Optional stop (M01) enabled A pulse forces a change of state to enable or inhibit optional stop depending on the previous state.
%W3.6	C_SLASH	Block skip enabled A pulse forces a change of state to enable or inhibit block skip depending on the previous state.
%W3.5	C_RAZER	Cancels the following error without a reset.
%W3.4	C_DGURG	Emergency retract request This request is taken into account in the AUTO, SINGLE and DRYRUN modes. The current block is interrupted and the system branches to the last emergency retract programme declared in the part programme by function G75. If no emergency retract programme was defined, this command is processed in the same way as C_ARUS.
%W3.3	C_RAX	Axis recall selection This request is taken into account when E_ARUS = 1 and all the axis jogs are released. It is a bistable type command. The first pulse sets E_INTERV and enables the axis jogs in both directions. If at least one axis has been moved in INTERV mode, a second pulse sets E_RAX and enables a single direction of movement of the axis jogs to return the slides to the initial position.
%W3.2	C_CYCLE	CYCLE START pulse Allows execution of the AUTO, SINGLE, MDI, DRYRUN, SEARCH, TEST, LOAD and UNLOAD modes. A pulse command must be used for C_CYCLE to prevent resumption of machining after detection of M02 or a reset in the AUTO and DRYRUN modes.
%W3.1	C_ARUS	Machining stop request This request is taken into account in the AUTO, SINGLE, MDI, DRYRUN and incremental JOG modes. The first pulse stops machining. Machining is restarted by action on CYCLE.

Variable	Mnemonic	Description
%W3.0	C_RAZ	Reset request. Also resets the PLC axes in case of a machine error. Taken into account if there is no movement on the axes.

**REMARKS** For processing of C\_ARUS, C\_CYCLE and C\_RAX, refer to the Operator Manual. For processing of C\_DGURG, refer to the Programming Manual.

### 3.8.2.2 Latching Commands: %W4.W

Variable	Mnemonic	Description
%W4.7	VREDUIT	Causes a switch to low speed Set to force the low speeds of movement set in words N3 and N4 of parameter P31 (see Parameter Manual).
%W4.6	INIBUTIL	Utility inhibit Set to inhibit access to the utilities. Reset to enable access to the utilities.
%W4.5	C_UNIT	Display units (metric or inch system) Set to enable dimension entry and display in inches. Reset to enable dimension entry and display in the metric system.
%W4.4	PRESPUIS	Motor power on. This variable is reset to indicate a synchronised axis motor power failure to the NC (after a synchronisation error). It is set to indicate power return and axis synchronisation enable to the NC.
%W4.3	NARFIB	No stop at end of block Enables execution of a CYCLE in the AUTO, SINGLE, MDI, DRYRUN modes and enables block sequencing in the AUTO and DRYRUN modes. Reset of this variable causes the cycle to stop at the end of execution of the current block.
%W4.2	VITMAN2	Selection of rapid feed rates in manual modes 1 and 2
%W4.1	VITMAN1	Enable selection of rapid feed rates in the JOG and HOME modes or setting the handwheel increment multiplier. The feed rates are modulated by the feed rate potentiometer.
		VITMAN1   VITMAN2   FEED RATE   HANDWHEEL (Parameter P31)   INCREMENT
		0        0        Normal JOG      IU x 1 0        1        Slow JOG        IU x 100 1        0        Fast JOG        IU x 10 1        1        Fast JOG        UI x 10
		Where IU = internal system unit set by a machine parameter.
%W4.0	AUTAV	Feed authorised on all the axis groups Enables movements in all the modes with movement. STOP in the CNC status window indicates that this operand is reset.
%W5.7	SC_SAVE	CNC screen on standby Set to enable the CNC screen to be placed on standby after five minutes of keyboard inactivity. Reset to inhibit the switch to standby of the screen and immediately reactivate the screen.

Variable	Mnemonic	Description
%W5.6	SK_DISPL	Softkey bar window display Set to inhibit display of the softkey bar window. Reset to enable display of the softkey bar window. <i>REMARK Inhibiting of the display does not inhibit use of the softkeys.</i>
%W5.5	INIBCLAV	Keyboard inhibit Set to inhibit the alphanumeric QWERTY keyboard and function keys for the basic softkeys which are then no longer processed by the CNC. The key codes are however transmitted to the automatic control function by CARCLAV.
%W5.4	IMPULS	Operator panel pulse inputs Disable the RAZ, ARUS, CYCLE, M01, / keys and the TCOMP softkey. Set to disable the keys on the CNC operator panel and enable selection by the automatic control function.
%W5.3	CORDYN	Wear offset load enable Set to enable load of the wear offsets by the automatic control function and disable load from the operator panel.
%W5.2	JOGPUP	JOG selection from the operator panel Set to disable selection of the jog type by the CNC operator panel and enable selection by the automatic control function.
%W5.1	MODEPUP	Operator panel mode selection Set to disable mode selection by the CNC operator panel and enable mode selection by the automatic control function. The mode is selected by the PLC. The mode number is encoded in %Wg03.b. The mode codes are the same as those of %W14.b for common modes. %W5.1=0. Except for MMI, the modes can be selected from the operator panel. The mode is assigned to the group selected by %W17.b.
%W5.0	PUPABS	CNC panel absent Set to declare the CNC panel absent. All the CNC operator panel functions are disabled and can be simulated by the automatic control function.

### 3.8.2.3 Positive JOG Commands: %W6.L

Variable	Mnemonic	Description
%W6.7 to %W6.0	JOGPOS31 to JOGPOS24	Positive jog on axis No. 31 to Positive jog on axis No. 24
%W7.7 to %W7.0	JOGPOS23 to JOGPOS16	Positive jog on axis No. 23 to Positive jog on axis No. 16
%W8.7 to %W8.0	JOGPOS15 to JOGPOS8	Positive jog on axis No. 15 to Positive jog on axis No. 8
%W9.7 to %W9.0	JOGPOS7 to JOGPOS0	Positive jog on axis No. 7 to Positive jog on axis No. 0

### 3.8.2.4 Negative JOG Commands: %WA.L

Variable	Mnemonic	Description
%WA.7 to %WA.0	JOGNEG31 to JOGNEG24	Negative jog on axis No. 31 to Negative jog on axis No. 24
%WB.7 to %WB.0	JOGNEG23 to JOGNEG16	Negative jog on axis No. 23 to Negative jog on axis No. 16
%WC.7 to %WC.0	JOGNEG15 to JOGNEG8	Negative jog on axis No. 15 to Negative jog on axis No. 8
%WD.7 to %WD.0	JOGNEG7 to JOGNEG0	Negative jog on axis No. 7 to Negative jog on axis No. 0

### 3.8.2.5 External Parameters E20000 to E20031: %WE.L

External parameters E200xx are written by the user programme. They are visible in the part programme.

They are used to make machine status data available to the part programme.

Variable	Mnemonic	Variable	Mnemonic
%W11.0	E20000	%WF.0	E20016
%W11.1	E20001	%WF.1	E20017
%W11.2	E20002	%WF.2	E20018
%W11.3	E20003	%WF.3	E20019
%W11.4	E20004	%WF.4	E20020
%W11.5	E20005	%WF.5	E20021
%W11.6	E20006	%WF.6	E20022
%W11.7	E20007	%WF.7	E20023
%W10.0	E20008	%WE.0	E20024
%W10.1	E20009	%WE.1	E20025
%W10.2	E20010	%WE.2	E20026
%W10.3	E20011	%WE.3	E20027
%W10.4	E20012	%WE.4	E20028
%W10.5	E20013	%WE.5	E20029
%W10.6	E20014	%WE.6	E20030
%W10.7	E20015	%WE.7	E20031

### 3.8.2.6 Value of the Jog Increment: %W13.B

Variable	Mnemonic	Description
%W13.B	C_INCJOG	<p>JOG increment command. The variable value corresponds to the jog increment requested:</p> <ul style="list-style-type: none"> <li>0xA Manual movement by <math>10^{-6}</math> inches</li> <li>0x09 Manual movement by <math>10^2 \mu\text{m}</math> or <math>10^{-5}</math> inches</li> <li>0x00 Manual movement by <math>10^1 \mu\text{m}</math> or <math>10^{-4}</math> inches</li> <li>0x01 Manual movement by <math>1 \mu\text{m}</math> or <math>10^{-3}</math> inches</li> <li>0x02 Manual movement by <math>10 \mu\text{m}</math> or <math>10^{-2}</math> inches</li> <li>0x03 Manual movement by <math>100 \mu\text{m}</math> or <math>10^{-1}</math> inches</li> <li>0x04 Manual movement by <math>1000 \mu\text{m}</math> or 1 inches</li> <li>0x05 Manual movement by <math>10000 \mu\text{m}</math> or 1 inch</li> <li>0x06 Continuous jog</li> <li>0x08 Manual movement by handwheel</li> </ul>

The jog increment is in  $\mu\text{m}$  or inches depending on the value of variable C\_UNIT.

### 3.8.2.7 Mode Requested: %W14.B

Variable	Mnemonic	Description
%W14.B	MODEDEM	<p>Mode Requested The value of the variable corresponds to the CNC mode requested:</p> <ul style="list-style-type: none"> <li>0x00 Auto mode «AUTO»</li> <li>0x01 Single step mode «SINGLE»</li> <li>0x02 Manual data input mode «MDI»</li> <li>0x03 DRYRUN mode «DRYRUN»</li> <li>0x04 Sequence number search mode «SEARCH»</li> <li>0x05 Edit mode «EDIT»</li> <li>0x06 Test mode «TEST»</li> <li>0x07 Manual mode «MANUAL»</li> <li>0x08 Home mode «HOME»</li> <li>0x09 Origin shift mode «SHIFT»</li> <li>0x0A Tool setting mode «TL SET»</li> <li>0x0B No mode active</li> <li>0x0D Load mode «LOAD»</li> <li>0x0F Unload mode «UNLOAD»</li> </ul>

### 3.8.2.8 Message Display: %W15.B and W16.B

Variable	Mnemonic	Description
%W15.B	MSG1	<p>Message number to be displayed on line 1 The message is displayed on line 1 of the «Error Message» page. A message with that number must be included in part programme %9999.9.</p>
%W16.B	MSG2	<p>Message number to be displayed on line 2 The message is displayed on line 2 of the «Error Message» page. A message with that number must be included in part programme %9999.9.</p>

Programme %9999.9 must be structured as follows:

```
%9999 .9
N0
N1 $MESSAGE NUMBER 1
$ REST OF MESSAGE NUMBER 1
N2 $MESSAGE NUMBER 2
$ REST OF MESSAGE NUMBER 2
$ REST OF MESSAGE NUMBER 2

Nx $MESSAGE NUMBER X
```

Where:

- the bloc numbers (N..) correspond to the numbers of the messages to be displayed,
- the messages are preceded by the character \$,
- a message line can contain a maximum of 35 characters,
- unnumbered blocks are used for continuation of the messages.

### 3.8.2.9 Axis Group Selection: %W17.B

Variable	Mnemonic	Description
%W17.B	SELECGR	Axis Group Selection Assigns all the data relative to one axis group (part programme, programme variable, etc.) for display. The data entered from the CNC keyboard in MDI mode is assigned to the axis group selected.
	0	Selection of axis group 1
	1	Selection of axis group 2
	2	Selection of axis group 3
	3	Selection of axis group 4
	4	Selection of axis group 5
	5	Selection of axis group 6
	6	Selection of axis group 7
	7	Selection of axis group 8

*REMARK Used only for axis multigroup machine-tools.*

### 3.8.2.10 Requested Programme Number: %W18.W

Variable	Mnemonic	Description
%W18.W	PROGDEM	Requested Programme Number Used to load the requested programme number as active programme or request machining in drip feed mode. The programme number or drip feed machining request is read by the system on the rising edge of the reset flag C_RAZ = 1.
	0	No programme number requested by the automatic control function.
	1 to 0x270F (9999)	Programme number specified by the automatic control function.
	-2 (0xFFFF)	Machining in drip feed mode requested by the automatic control function.

**REMARK** *The programme requested must be present in the CNC memory to be installed as the active programme. If it is not present, the system cancels the old active programme and the message «NO ACTIVE PROGRAMME» is displayed on the current programme page.*

### 3.8.2.11 Handwheel Assignment: %W1A.B to %W1D.B

Variable	Mnemonic	Description
%W1A.B	AFMAN1	Handwheel 1 Assignment Contains the physical address of the axis to be moved. Refer to the Installation and Commissioning Manual for assignment of the physical axis addresses.
%W1B.B	AFMAN2	Handwheel 2 Assignment Same as AFMAN1 for handwheel 2.
%W1C.B	AFMAN3	Handwheel 3 Assignment Same as AFMAN1 for handwheel 3.
%W1D.B	AFMAN4	Handwheel 4 Assignment Same as AFMAN1 for handwheel 4.

 **CAUTION**

Variables AFMAN1, AFMAN2, AFMAN3 and AFMAN4 must contain the physical address of a measured axis.

The assignment of the handwheel to an axis must precede manual movement via the handwheel.

The JOG, JOGPOSn and JOGNEGn commands (with n from 0 to 31) must be enabled for the axis concerned.

### 3.8.2.12 Spindle Potentiometer: %W1E.B to %W21.B

Variable	Mnemonic	Description									
%W1E.B	POTBR1	Spindle 1 Potentiometer A hexadecimal value corresponding to the ADC input.									
		<table> <thead> <tr> <th>Hexadecimal code</th> <th>ADC input (anai(..) function)</th> <th>Spindle speed override percentage</th> </tr> </thead> <tbody> <tr> <td>0x0</td> <td>0 Volts</td> <td>50%</td> </tr> <tr> <td>0xFF</td> <td>10 Volts</td> <td>100%</td> </tr> </tbody> </table>	Hexadecimal code	ADC input (anai(..) function)	Spindle speed override percentage	0x0	0 Volts	50%	0xFF	10 Volts	100%
Hexadecimal code	ADC input (anai(..) function)	Spindle speed override percentage									
0x0	0 Volts	50%									
0xFF	10 Volts	100%									
%W1F.B	POTBR2	Spindle 2 Potentiometer Same as POTBR1 for spindle 2.									
%W20.B	POTBR3	Spindle 3 Potentiometer Same as OTBR1 for spindle 3.									
%W21.B	POTBR4	Spindle 4 Potentiometer Same as OTBR1 for spindle 4.									

### 3.8.2.13 Spindle Controls: %W22.W

Variable	Mnemonic	Description
%W22.7	VERBR4	Spindle 4 power VERBR4 = 0: Indicates to the CNC that spindle 4 power is on VERBR4 = 1: Indicates to the CNC that spindle 4 is manually inhibited or locked
%W22.6	VERBR3	Spindle 3 power Same as VERBR4 for spindle 3
%W22.5	VERBR2	Spindle 2 power Same as VERBR4 for spindle 2
%W22.4	VERBR1	Spindle 1 power Same as VERBR4 for spindle 1
%W22.3	STOPBR4	Spindle 4 stop requested by the PLC function Latched command. The spindle remains stopped as long as this bit remains set. When the bit is reset, the spindle can begin rotating again.
%W22.2	STOPBR3	Spindle 3 stop requested by the PLC function Same as STOPBR4 for spindle 3
%W22.1	STOPBR2	Spindle 2 stop requested by the PLC function Same as STOPBR4 for spindle 2
%W22.0	STOPBR1	Spindle 1 stop requested by the PLC function Same as STOPBR4 for spindle 1
%W23.3	COMBR4	Spindle 4 control Set to enable spindle control by the automatic control function. The setting is transmitted to the axis card by C_VITBR4.
%W23.2	COMBR3	Spindle 3 control Set to enable spindle control by the automatic control function. The setting is transmitted to the axis card by C_VITBR3.
%W23.1	COMBR2	Spindle 2 control Set to enable spindle control by the automatic control function. The setting is transmitted to the axis card by C_VITBR2.
%W23.0	COMBR1	Spindle 1 control Set to enable spindle control by the automatic control function. The setting is transmitted to the axis card by C_VITBR1.

### 3.8.2.14 Spindle Speed Setting: %W24.W to %W2A.W

Variable	Mnemonic	Description
%W24.W	C_VITBR1	Spindle 1 Speed Setting Used to send the spindle servo-drive reference in binary code on 14 bits plus sign. Bit 15 of C_VITBR1 gives the sign of the setting.
%W26.W	C_VITBR2	Spindle 2 Speed Setting Same as C_VITBT1 for spindle 2.
%W28.W	C_VITBR3	Spindle 3 Speed Setting Same as C_VITBR1 for spindle 3.
%W2A.W	C_VITBR4	Spindle 4 Speed Setting Same as C_VITBR1 for spindle 4.

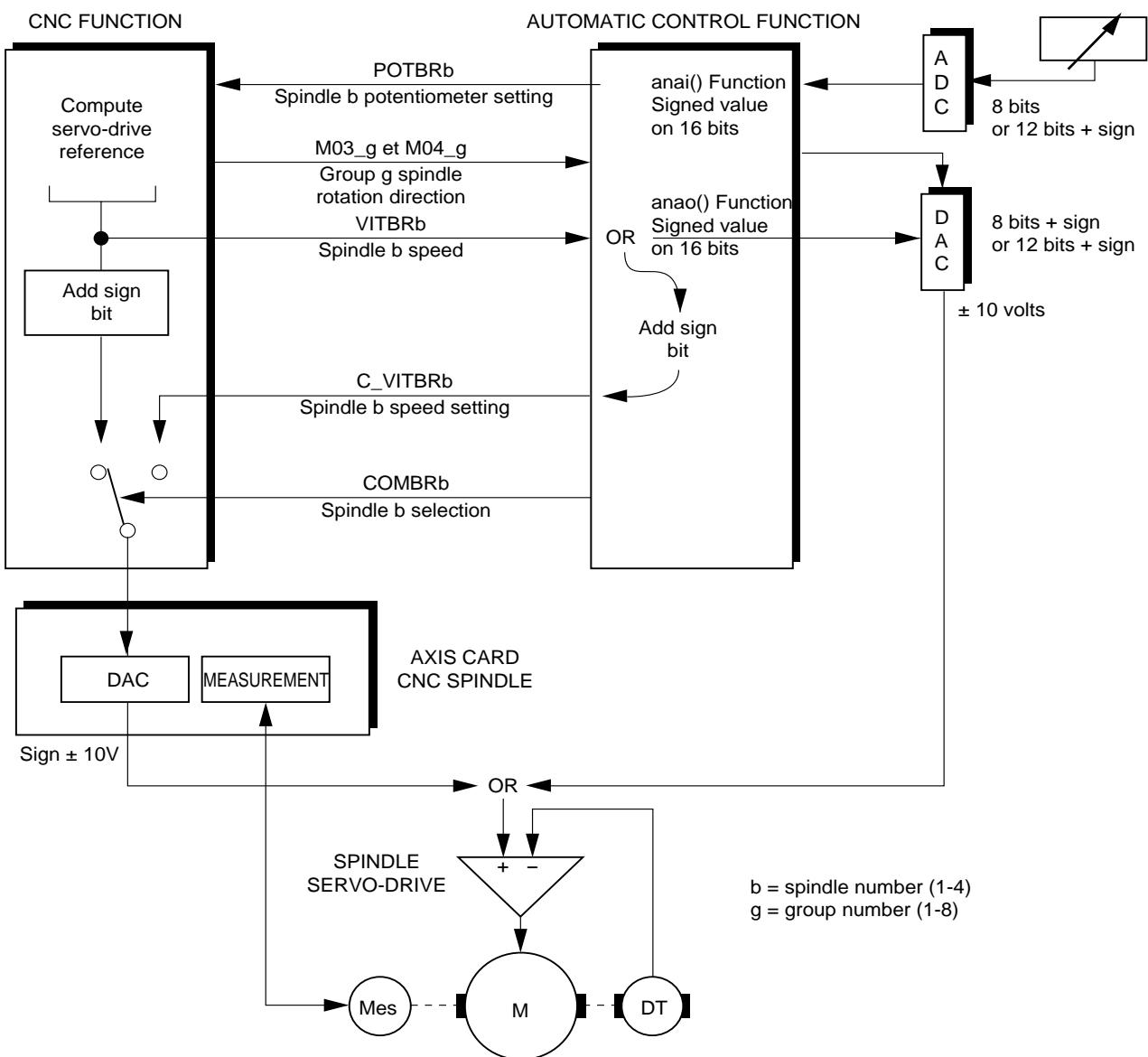


Figure 3.2 - Organisation of a spindle

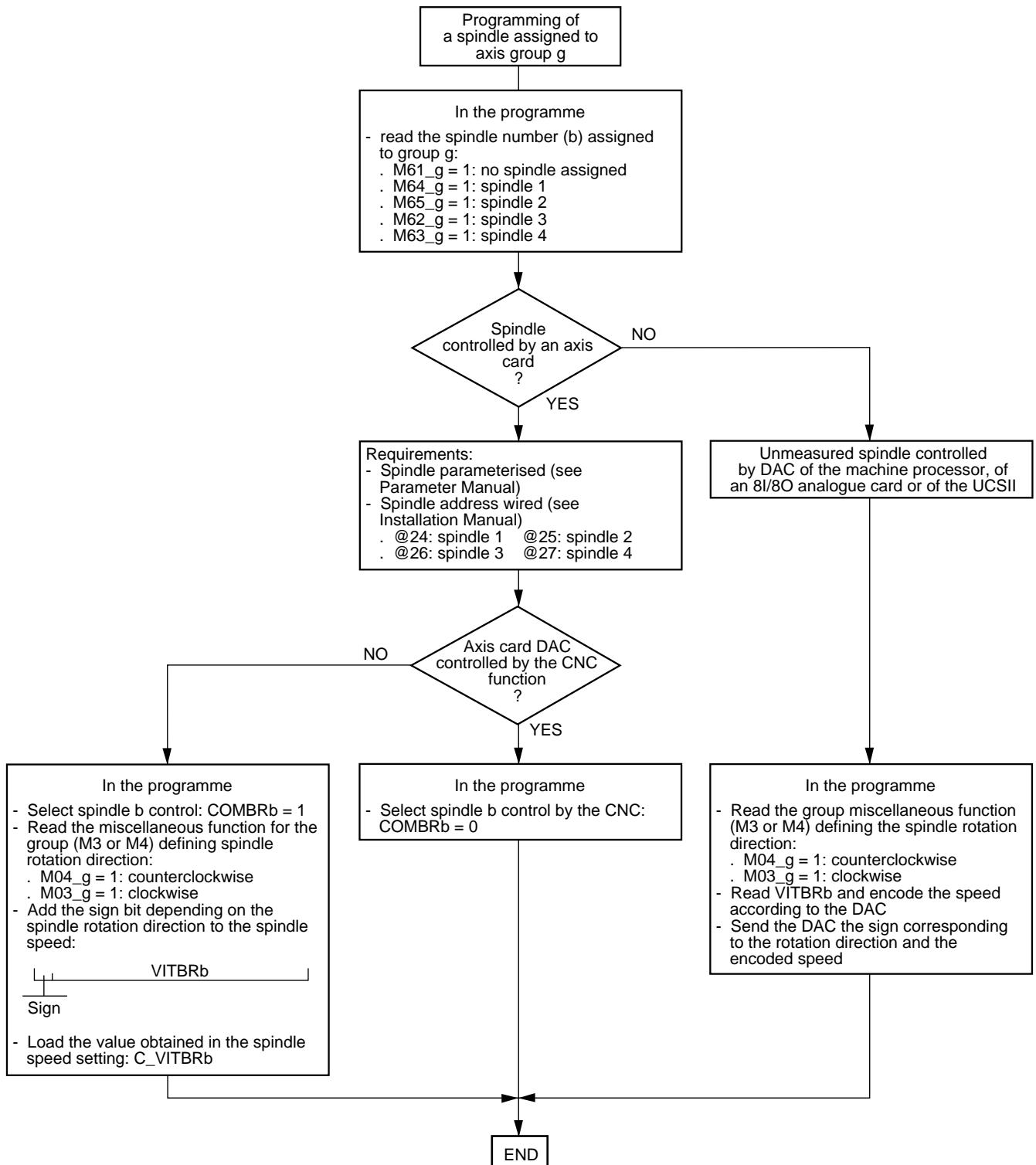


Figure 3.3 - Programming of a spindle

### 3.8.2.15 Inhibited Jog Increments: %W2C.W

Variable	Mnemonic	Description
%W2C.1	NJGMANIV	Inhibits handwheel selection. Inhibits the HAND softkey in the jog key bar. Set to inhibit the key. Reset to enable the key.
%W2C.0	NJG0001	Inhibits selection of the 0.001 mm increment. Inhibits the .001 softkey in the jog key bar. Set to inhibit the key. Reset to enable the key.
%W2D.7	NJG001	Inhibits selection of the 0.01 mm increment. Inhibits the .01 softkey in the jog key bar. Set to inhibit the key. Reset to enable the key.
%W2D.6	NJG01	Inhibits selection of the 0.1 mm increment. Inhibits the .1 softkey in the jog key bar. Set to inhibit the key. Reset to enable the key.
%W2D.5	NJG1	Inhibits selection of the 1 mm increment. Inhibits the 1 softkey in the jog key bar. Set to inhibit the key. Reset to enable the key.
%W2D.4	NJG10	Inhibits selection of the 10 mm increment. Inhibits the 10 softkey in the jog key bar. Set to inhibit the key. Reset to enable the key.
%W2D.3	NJG100	Inhibits selection of the 100 mm increment. Inhibits the 100 softkey in the jog key bar. Set to inhibit the key. Reset to enable the key.
%W2D.2	NJG1000	Inhibits selection of the 1000 mm increment. Inhibits the 1000 softkey in the jog key bar. Set to inhibit the key. Reset to enable the key.
%W2D.1	NJG10000	Inhibits selection of the 10000 mm increment. Inhibits the 10000 softkey in the jog key bar. Set to inhibit the key. Reset to enable the key.
%W2D.0	NJGILLIM	Inhibits continuous jog selection. Inhibits the FREE softkey in the jog key bar. Set to inhibit the key. Reset to enable the key.

### 3.8.2.16 Modes Inhibited: %W30.L

Variable	Mnemonic	Description
%W30.7	I_POM	Inhibits selection of the homing mode. Inhibits the HOME softkey in the mode key bar. Set to inhibit the key. Reset to enable the key.
%W30.6	I_PREF	Inhibits selection of origin shift mode. Inhibits the SHIFT softkey in the mode key bar. Set to inhibit the key. Reset to enable the key.
%W30.5	I_REGOUT	Inhibits selection of automatic tool setting mode. Inhibits the TLSET softkey in the mode key bar. Set to inhibit the key. Reset to enable the key.
%W30.2	I_CHARG	Inhibits selection of the load mode. Inhibits the LOAD softkey in the mode key bar. Set to inhibit the key. Reset to enable the key.
%W30.0	I_DCHG	Inhibits selection of the unload mode. Inhibits the UNLOAD softkey in the mode key bar. Set to inhibit the key. Reset to enable the key.
%W31.7	I_CONT	Inhibits selection of the automatic mode. Inhibits the AUTO softkey in the mode key bar. Set to inhibit the key. Reset to enable the key.
%W31.6	I_SEQ	Inhibits selection of the single step mode. Inhibits the SINGLE softkey in the mode key bar. Set to inhibit the key. Reset to enable the key.
%W31.5	I_IMD	Inhibits selection of the manual data input mode. Inhibits the MDI softkey in the mode key bar. Set to inhibit the key. Reset to enable the key.
%W31.4	I_RAPID	Inhibits selection of the dry run mode. Inhibits the DRYRUN softkey in the mode key bar. Set to inhibit the key. Reset to enable the key.
%W31.3	I_RNS	Inhibits selection of the sequence number search mode. Inhibits the SEARCH softkey in the mode key bar. Set to inhibit the key. Reset to enable the key.
%W31.2	I_MODIF	Inhibits selection of the edit mode. Inhibits the EDIT softkey in the mode key bar. Set to inhibit the key. Reset to enable the key.

Variable	Mnemonic	Description
%W31.1	I_TEST	Inhibits selection of the test mode. Inhibits the TEST softkey in the mode key bar. Set to inhibit the key. Reset to enable the key.
%W31.0	I_JOG	Inhibits selection of the manual mode. Inhibits the MANUAL softkey in the mode key bar. Set to inhibit the key. Reset to enable the key.

### 3.8.2.17 Torque Enable for QVN Axes: %W34.L

The bits of %W34.L are initialised with 0.

Variable	Mnemonic	Description
%W34.7 to %W34.0	DISC_TRQ31 to DISC_TRQ24	Torque enabled on QVN axis No. 31 to Torque enabled on QVN axis No. 24 Set to enable torque Reset to inhibit torque
%W35.7 to %W35.0	DISC_TRQ23 to DISC_TRQ16	Torque enabled on QVN axis No. 23 to Torque enabled on QVN axis No. 16 Set to enable torque Reset to inhibit torque
%W36.7 to %W36.0	DISC_TRQ15 to DISC_TRQ8	Torque enabled on QVN axis No. 15 to Torque enabled on QVN axis No. 8 Set to enable torque Reset to inhibit torque
%W37.7 to %W37.0	DISC_TRQ7 to DISC_TRQ0	Torque enabled on QVN axis No. 7 to Torque enabled on QVN axis No. 0 Set to enable torque Reset to inhibit torque

### 3.8.2.18 Speed Reference Enable for QVN Axes: %W38.0

Variable	Mnemonic	Description
%W38.0	DISC_SDP	Speed reference enable for QVN axes Set to enable normal operation of QVN axes. Reset to cancel the speed reference immediately on the QVN axes to enable braking at maximum torque.

If the speed references are inhibited, they are forced low.

At power on, the speed references are inhibited.

In the case of detection of a CNC error causing E\_CNPRET to go low, the speed reference is forced low for the QVN axes. Cancellation of the error by a reset again allows the automatic control function to enable or inhibit the speed references.

**REMARK** *It is recommended to inhibit DISC\_SDP on an emergency stop and to activate a feed stop so as not to generate an excessive following error.*

### 3.8.2.19 Backward of Forward Movement in Path

Variable	Mnemonic	Description
%W39.2	RAP_AUTO	Automatic recall after maintenance. Set to enable recall. Reset to cancel recall.
%W39.1	B_RECUL	Forward movement requested on path. Set to enable the request. Reset to inhibit the request.
%W39.0	B_RECUL	Backword movement requested on path. Set to enable the request. Reset to inhibit the request.

### 3.8.2.20 Feed Stop per Axis (the bit number corresponds to the physical address of the axis): %W3A.L

Variable	Mnemonic	Description
%W3A.7 to %W3A.0	STOPAX31 to STOPAX24	Axis 31 to axis 24 In machining mode or JOG mode, setting a bit that addresses one of the axes in movement stops the axes of the group to which it belongs. In machining mode, if this axis does not move in the block being executed but its movement is programmed in the next block, an end-of-block stop request is generated and execution of the movement at the beginning of the next block remains suspended as long as an axis programmed in this block is stopped by setting this bit. In nmauto mode, action on the jogs or handwheel is ignored as long as the bit corresponding to the axis directly controlled is set.
%W3B.7 to %W3B.0	STOPAX23 to STOPAX16	Axis 23 to axis 16 Same as above
%W3C.7 to %W3C.0	STOPAX15 to STOPAX8	Axis 15 to axis 8 Same as above
%W3D.7 to %W3D.0	STOPAX7 to STOPAX0	Axis 7 to axis 0 Same as above

### 3.8.2.21 Current Reduction: %WE00.B to WE1F.B DISC and 1050

The current reduction function reduces the current on the digital axes and spindles according to the value of the corresponding byte.

Variable	Mnemonic	Description
%WE1F.B to %WE00.B	RDUC_TRQ31 to RDUC_TRQ0	Axis 31 to axis 0 Axis 7 to axis 0

Let  $I_{\text{maximum}}$  be the maximum current taking into account static current limiting and  $\alpha$  be the value of the byte:

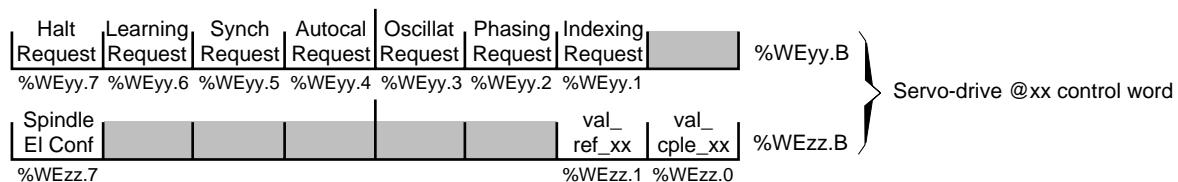
- If  $\alpha$  is negative or zero (\$00, \$80-\$FF), there is no current reduction
- If  $\alpha$  is positive (\$01-\$7F), the maximum permissible current is:  $I_{\text{maximum}} = I_{\text{max\_stat}} \times [(127 - \alpha)/127]$

The dynamic current reduction applied to a master digital servo-drive is transmitted to the associated slave digital servo-drives.

When operating in antibacklash configuration, the dynamic reduction applied to a master digital servo-drive has no effect on the master and slave preload currents.

### 3.8.2.22 1050 Servo-Drive Control Word

For the digital servo-drive at address xx (xx between 00 and 31), the control word format is as follows:



Bit	Meaning	Value
%WEzz.0	Torque enable	0: torque enable not requested 1: torque enable requested
%WEzz.1	Reference enable	0: reference not enabled 1: reference enabled
%WEzz.7	Spindle Electrical Configuration	0: low range 1: high range
%WEyy.1	Indexing Request	0: indexing not requested 1: indexing requested
%WEyy.2	Phasing Request	0: sensor phasing not requested 1: sensor phasing requested
%WEyy.3	Oscillation Request	0: oscillation not requested 1: oscillation requested
%WEyy.4	Autocalibration Request	0: self-calibration not requested 1: self-calibration requested
%WEyy.5	Synchronization Request	Reserved
%WEyy.6	Learning Request	Reserved
%WEyy.7	Halt Request	0: halt not requested 1: halt requested

### 3.8.3 Inputs from the Axis Groups

The inputs from the axis groups are contained in eight 128-byte blocks. These are variables %Rg00 to %Rg7F where g equals 1 to 8 for groups 1 to 8.

#### 3.8.3.1 Group Status: %Rg00.W

**REMARK** All these variables apply to independent CNC axis groups. Only variables E\_RAZ1 to E\_RAZ8, E\_CYCL1 to E\_CYCL8, E\_DEGURG1 to E\_DEGURG8, NO\_POS1 to NO\_POS8 and E\_DEF1 to E\_DEF8 apply to PLC axis groups (see Chapter 17).

Variable	Mnemonic (group 1 to 8)	Description
%Rg00.7	E_M011 to E_M018	Optional programme stop enabled on group g of independent CNC axes Indicates that optional programme stops in a part programme are enabled.
%Rg00.6	E_SLASH1 to E_SLASH8	Block skip enabled on group g of independent CNC axes Indicates that block skips in a part programme are enabled.
%Rg00.5	E_INTER1 to E_INTER8	Intervention status on group g of independent CNC axes.
%Rg00.0	E_PROG1 to E_PROG8	Active programme on group g of independent CNC axes. Indicates that a part programme is active in one of AUTO, SINGLE, DRYRUN, SEARCH, TEST or MDI modes
%Rg01.7	E_OPER1 to E_OPER8	Indicates that a programme stop caused by M00 or M01 is enabled.
%Rg01.6	E_DEF1 to E_DEF8	Fault on group g Indicates a programming error or the absence of a part programme on the group. Set to indicate that the group is faulty.
%Rg01.5	NO_POS1 to NO_POS8	Axis on wait for positioning When accurate positioning is required by programming (functions G09, M00, M02 or M10) in MDI or JOG mode whenever movement is stopped, signal NO_POSg is transmitted while the axis is on wait for positioning. Set to indicate that the axis is on wait for positioning.
%Rg01.4	E_DGURG1 to E_DGURG8	Emergency retraction in progress on group g Indicates execution of an emergency retraction programme. Set after read of C_DGURGg = 1 by the CNC if the emergency retraction programme is enabled. Reset by detection of M00 or M02.
%Rg01.3	E_RAX1 to E_RAX8	Axis recall on group g of independent CNC axes Indicates that axis recall is enabled.
%Rg01.2	E_CYCL1 to E_CYCL8	Cycle in progress on group g Indicates that the group is executing a part programme block. Reset: the CNC is waiting for flag C_CYCLEg = 1 to execute the part programme or the next block. Set: indicates that a block is being executed.

Variable	Mnemonic (group 1 to 8)	Description
%Rg01.1	E_ARUS1 to E_ARUS8	Exit from cycle stop on group g of independent CNC axes Indicates system intervention status (programme stopped during execution and axis jogs enabled).
%Rg01.0	E_RAZ1 to E_RAZ8	Reset in progress on group g Pulse bit with a duration of 100 ms that indicates a reset on the group. While this pulse is set, the data from the automatic control function are ignored. Set by the RAZ key on the operator panel, by a reset request from the automatic control function C_RAZg = 1, at the end of execution of a part programme (M02) or at CNC power on. This variable is reset after 100 ms.
%Rg06.B	MODCOUR1 to MODCOUR8	Current mode on CNC independent axis group g. The value of this variable is the image of the current CNC mode on independent CNC axis group g.

### 3.8.3.2 Current Machining Cycle Number: %Rg02.B

Variable	Mnemonic (group 1 to 8)	Description
%Rg02.B	NUMCYC1 to NUMCYC8	Current machining cycle number on group g. Used to read the machining cycle subroutine number from %10000 to %10255 (0 for %10000 to 0xFF for %10255).

### 3.8.3.3 G Function Status: %Rg03.B

Variable	Mnemonic (group 1 to 8)	Description
%Rg03.1	FILET1 to FILET8	Thread cutting on group g Indicates execution of a thread cutting cycle: G31 (thread chasing), G33 (thread cutting) or G38 (sequenced thread cutting), G84K (rigid tapping). Set by execution of function G31, G33, G38 or G84. Reset by cancellation of the function.
%Rg03.0	RAPID1 to RAPID8	Rapid positioning (G00) on group g Indicates execution of G0 in the current block of the part programme. Set by execution of function G0. Reset by cancellation of function G0.

### 3.8.3.4 Encoded M Function Without Response: %Rg04.W

Variable	Mnemonic (group 1 to 8)	Description
%Rg04.W	MSSCR1 to MSSCR8	<p>Encoded M Function without Response from Group g</p> <p>This variable is used to read «on the fly» encoded M functions without report from M200 to M899 (e.g. M210 sends MSSCRg == 210 to the automatic control function).</p> <p>These functions are considered pre-move and modal by the system.</p> <p>The part programme is continued without waiting for an acknowledgement.</p> <p>Used in part programmes, they are accessible for read by the automatic control function and must be decoded in the user programme.</p> <p>Only one modal encoded M function can be included in a part programme block.</p> <p>One modal and one nonmodal encoded M function can be included in the same part programme block.</p> <p>M functions must always be decoded by sequential task TS0.</p>

«On the fly» encoded M  
function MSSCRg

Figure 3.4 - «On the fly» encoded M functions.

### 3.8.3.5 Encoded M Function With Response: %Rg1E.W

Variable	Mnemonic (group 1 to 8)	Description
%Rg1E.W	MCODCR1 to MCODCR8	<p>Encoded M Function with Response Received from Group g</p> <p>This variable is used to read the encoded M functions with report up to M199 (e.g. M92 sends %MCODCRg == 92 to the automatic control function).</p> <p>These functions are considered post move and nonmodal by the system.</p> <p>The automatic control function determines whether or not they are modal. Used in part programmes, they are accessible for read by the automatic control function and must be decoded in the user programme.</p> <p>Only one non-modal encoded M function can be included in a part programme block.</p> <p>One modal and one nonmodal encoded M function can be included in the same part programme block.</p> <p>M functions must always be decoded by sequential task TS0.</p>

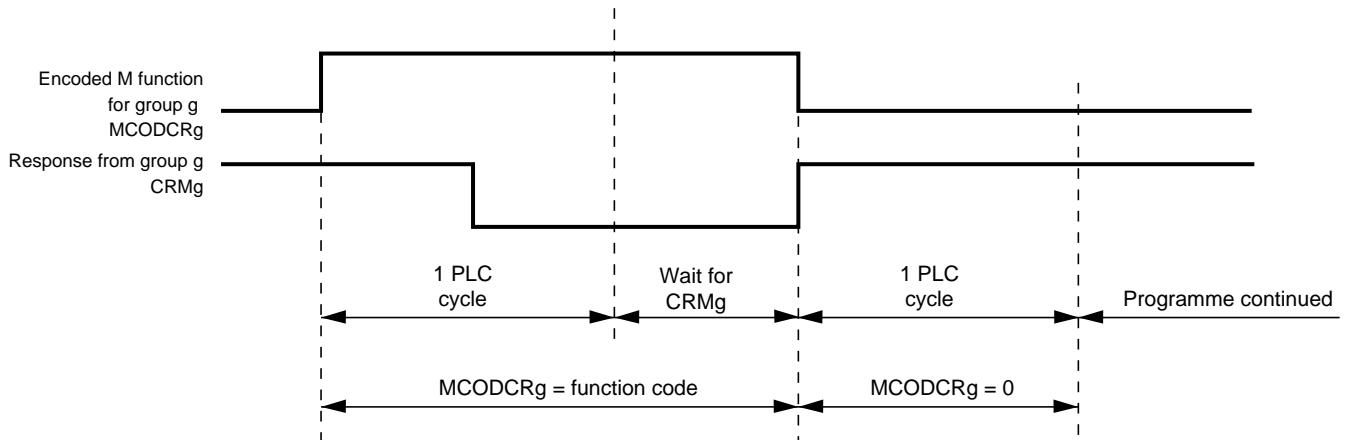


Figure 3.5 - Encoded M functions with response.

**REMARK** If CRM1 to CRM8 remains high, the part programme is continued after one PLC cycle.

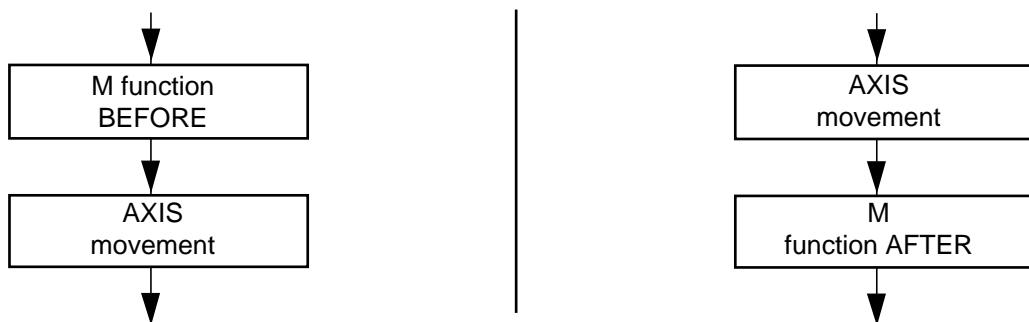
### 3.8.3.6 Decoded M Functions: %Rg20.L

These functions used in part programmes are accessible for read by the automatic control function.

These functions are defined by and known to the system (e.g. axis clamping, spindle speed range, etc.).

The automatic control function reads the function on a bit (%Rg2n.i) assigned to a decoded M function.

A distinction is made between:



#### Modal functions

A modal function remains stored and enabled during the execution of several part programme blocks, until it is cancelled.

Example (On group 1)

N100 M3 M40 S1000      M3 and M40 sent to the machine processor, i.e. %R122.0 = 1 and %R121.0 = 1.

N110 X100      Movement on X. %R122.0 = 1 and %R121.0 = 1 remain enabled in the machine processor.

N120 M5      M5 sent to the machine processor, cancelling M3, i.e. %R122.2 = 1 and %R122.0 = 0.

### Nonmodal functions

A nonmodal function is enabled only during execution of the part programme block containing it.

Example (On group 1)

N100 X100 Z200 M6      M6 sent to the machine processor, i.e. %R122.3 = 1.

N100 X50      M6 cancelled by acknowledgement of the previous block, i.e. %R122.3 = 0.

#### **⚠ CAUTION**

All the decoded M functions are acknowledged by a response (CRM1 to CRM8)

The status of CRM1 to CRM8 determines whether the part programme is continued or waits for the end of execution of the block

The automatic control function must manage CRM1 to CRM8 for the programmed functions and the functions cancelled or reset (by a reset or INIT).

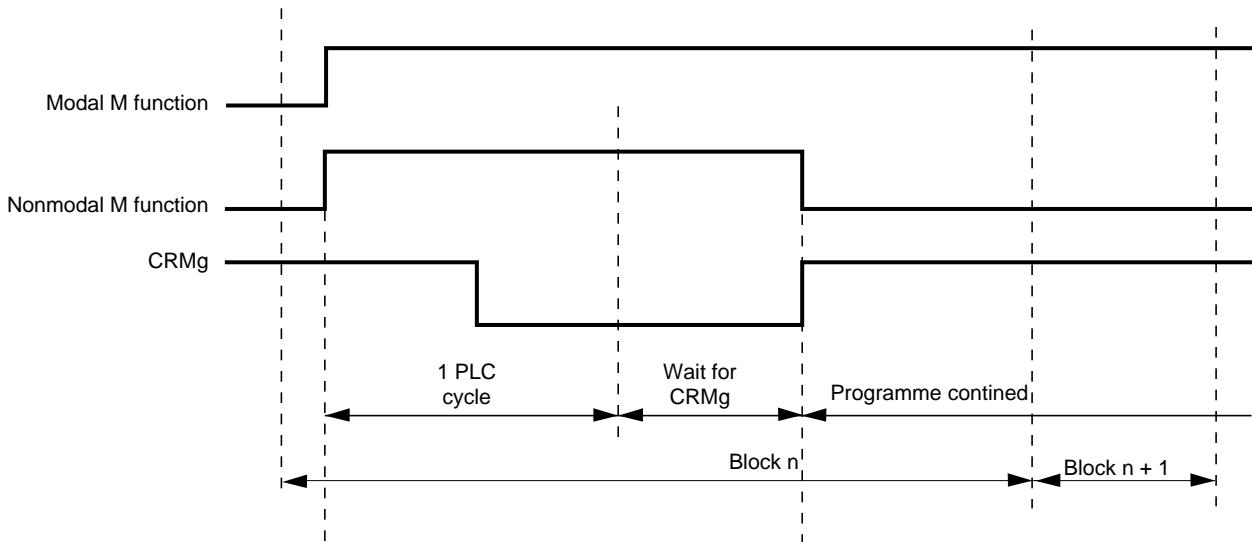


Figure 3.6 - Processing of decoded M functions.

**REMARK** If CRM1 to CRM8 remains high, the part programme is continued after the PLC cycle.

Variable	M function (Group 1 to 8)	Definition	Cancelled by	Type of function			Non modal
				Pre	Post	Modal	
%Rg20.7	M999_1	M999_8	Programmed inhibit of MDI and EDIT modes and subroutine calls by the machine processor	M997, M998, M2	X		X
%Rg20.6	M998_1*	M998_8*	Programmed enable of MDI and EDIT modes and subroutine calls by the machine processor	M999, M997	X		X
%Rg20.5	M997_1	M997_8	Forced block continuation	M998, M999, M2	X		X
%Rg20.3	M49_1	M49_8	Feed rate and spindle potentiometers forced to 100 percent	M48, M2	X		X
%Rg20.2	M48_1*	M48_8*	Feed rate and spindle potentiometers enabled	M49		X	X
%Rg20.1	M11_1	M11_8	Axis unclamp	M10	X		X
%Rg20.0	M10_1	M10_8	Axis clamp	M11		X	X
%Rg21.7	M12_1	M12_8	Forcing of CYHLD mode	C_CYCLE = 1		X	X
%Rg21.5	M45_1	M45_8	Spindle ranges	Cancelled by one another or by M02	X		X
%Rg21.4	M44_1	M44_8			X		X
%Rg21.3	M43_1	M43_8			X		X
%Rg21.2	M42_1	M42_8			X		X
%Rg21.1	M41_1	M41_8			X		X
%Rg21.0	M40_1	M40_8			X		X
%Rg22.7	M19_1	M19_8	Indexed spindle stop	M0, M2, M3, M4, ARUS		X	X
%Rg22.6	M09_1*	M09_8*	Coolant stop	M7, M8		X	X
%Rg22.5	M08_1	M08_8	Coolant 1	M9, M2	X		X
%Rg22.4	M07_1	M07_8	Coolant 2	M9, M2	X		X
%Rg22.3	M06_1	M06_8	Tool change	CRM1 to CRM8		X	
%Rg22.2	M05_1*	M05_8*	Spindle stop	M3, M4		X	X
%Rg22.1	M04_1	M04_8	Anticlockwise spindle rotation	M3, M5, M19, M0, M2	X		X
%Rg22.0	M03_1	M03_8	Clockwise spindle rotation	M4, M5, M19, M0, M2	X		X
%Rg23.7	M61_1	M61_8	Current spindle disabled in a group	M64, M65, M62, M63		X	X
%Rg23.2	M02_1	M02_8	End of part programme	RAZ		X	
%Rg23.1	M01_1	M01_8	Optional stop	C_CYCLE key		X	
%Rg23.0	M00_1	M00_8	Programme stop	C_CYCLE key		X	

(\*) Function initialised at power on by a reset or by function M02.

### 3.8.3.7 Decoded M Functions (Spindle Status): %Rg24.W

Variable	M function (Group 1 to 8)	Definition	Cancelled by	Type of function			Non modal
				Pre	Post	Modal	
%Rg24.3	M63_1	M63_8	Spindle reference applied to spindle 4.	M61, M62, M64, M65	X		X
%Rg24.2	M62_1	M62_8	Spindle reference applied to spindle 3.	M61, M63, M64, M65	X		X
%Rg24.1	M65_1	M65_8	Spindle reference applied to spindle 2.	M61, M62, M63, M64	X		X
%Rg24.0	M64_1	M64_8	Spindle reference applied to spindle 1.	M61, M62, M63, M65	X		X

Variable	M function (Group 1 to 8)	Definition	Cancelled by	Type of function			
				Pre	Post	Modal	Non modal
%Rg25.3	M69_1	M69_8	Spindle 4 measurement enabled	M66, M67, M68, M02	X		X
%Rg25.2	M68_1	M68_8	Spindle 3 measurement enabled	M66, M67, M69, M02	X		X
%Rg25.1	M67_1	M67_8	Spindle 2 measurement enabled	M66, M68, M69, M02	X		X
%Rg25.0	M66_1	M66_8	Spindle 1 measurement enabled	M67, M68, M69, M02	X		X

### 3.8.3.8 Axis Clamp/Unclamp

The axes can be clamped by miscellaneous function M10 and unclamped by miscellaneous function M11. The list of axes with clamps is defined by machine parameter P8 (see Parameter Manual).

If function M10 is present (axis clamp if there is no movement), the system detects the change of state of variables AXMVTaxis (where axis = 0 to 31) on the axes with clamps.

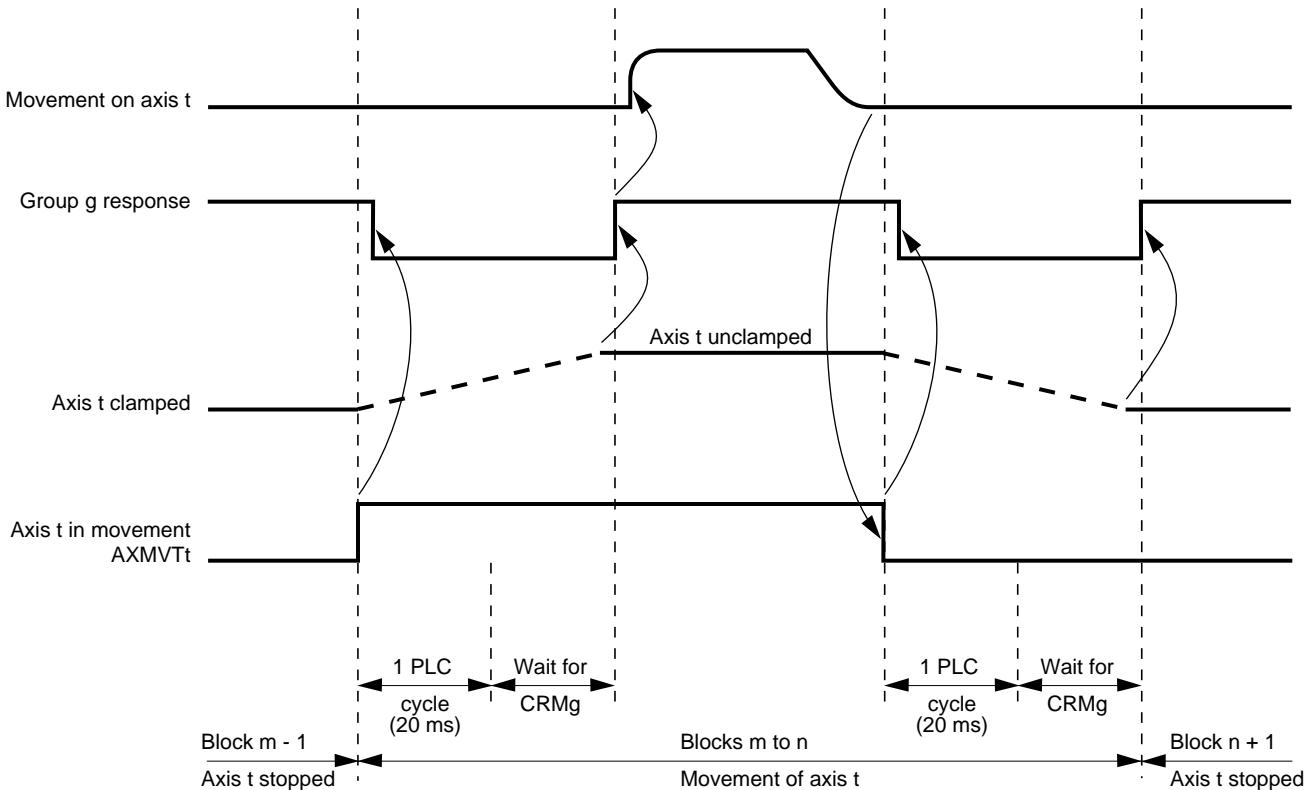


Figure 3.7 - Principle of axis clamping/unclamping

**REMARK** If the axis is still moving at block  $n + 1$ , variable AXMVTt (where  $t = 0$  to 31) remains high and the blocks are sequenced.

### 3.8.3.9 Tool Number: %Rg7C.L

Variable	Mnemonic (group 1 to 8)	Description
%Rg7C.L	OUTIL1	Tool Number Requested by Group g
	OUTIL8	Contains the tool numbers (decimal values from 0 to 65535). T functions are considered as pre-move and modal by the system, which does not wait for a response.

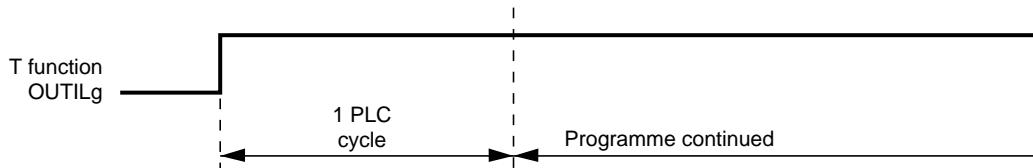


Figure 3.8 - Processing of T functions

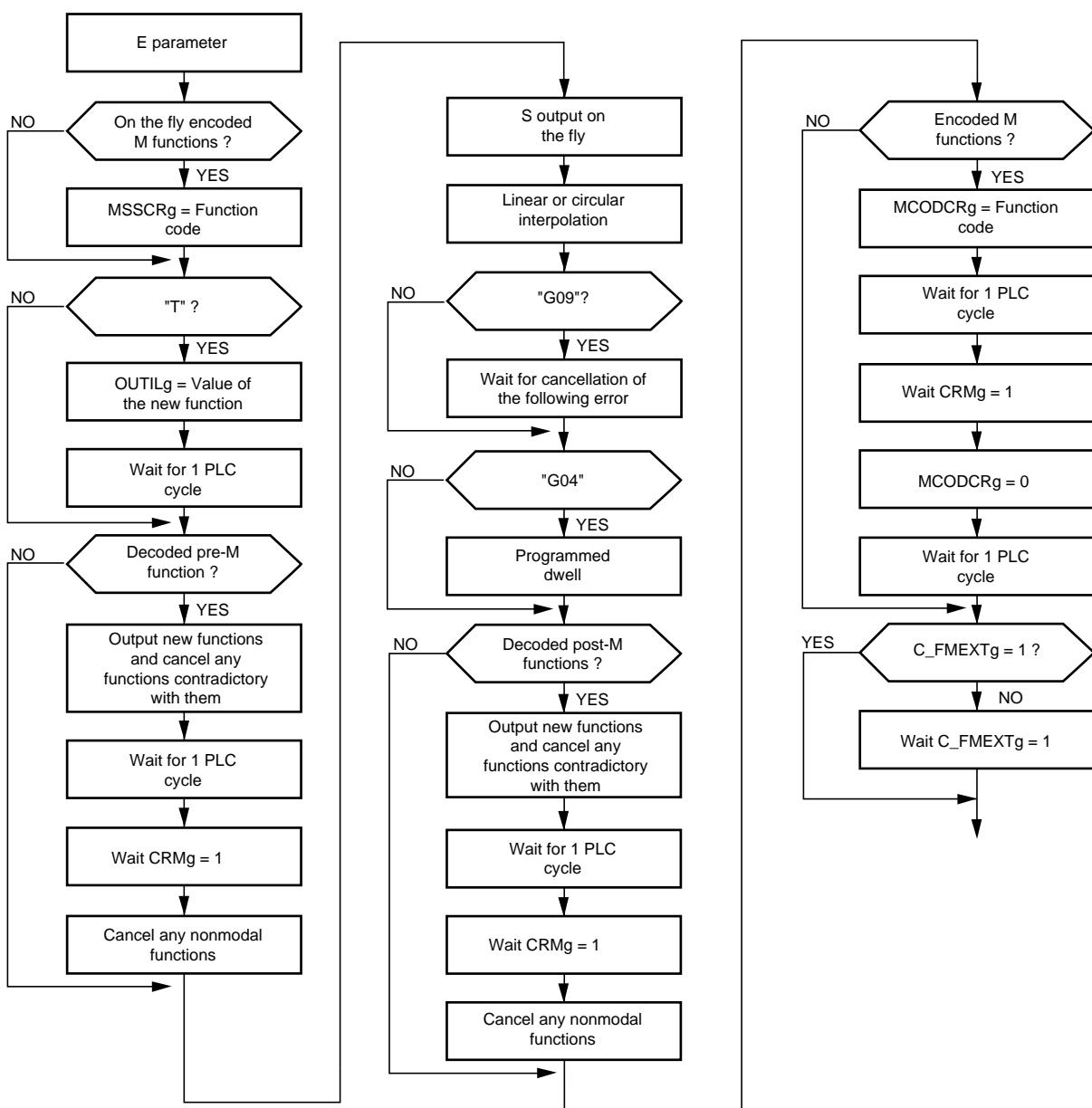


Figure 3.9 - Flow chart of functions programmed in a part programme block

### 3.8.4 Outputs to the Axis Groups

The outputs to the axis groups are included in eight 128-byte blocks:

Concerns variables %Wg00 to %Wg7F where g equals 1 to 8 for axis groups 1 to 8.

#### 3.8.4.1 Group Commands: %Wg00.W

**REMARK** Variables C\_MODE1 to C\_MODE8 apply only to PLC axis groups (see Chapter 17).

Variables C\_ARUS1 to C\_ARUS8, C\_RAX1 to C\_RAX8, C\_SLASH1 to C\_SLASH8 and C\_M011 to C\_M018 apply only to CNC axis groups.

Variable	Mnemonic (group 1 to 8)	Description
%Wg00.7	C_MODE1 to C_MODE8	Activates AUTO or SINGLE mode on PLC axis groups g.  Reset: AUTO mode enabled on the next block. Set: SINGLE mode enabled for the current block.  This flag is meaningful only if the group is valid.
%Wg00.6	C_FAST1 to C_FAST8	Latched high speed command during a cycle  This command is used during a cycle (C_CYCLEg = 1). Set to enable movement at the highest possible speed. Reset to enable movement at the work rate.
%Wg00.5	CRM1 to CRM8	M function response for group g.  Reset: places the system on wait without processing the next functions in the block being executed. Set: enables processing to continue.
%Wg00.4	APPSS1 to APPSS8	Subroutine call for group g.  During execution of a part programme, APPSS is set for a branch to subroutine %9999.g (where g is the group number). Latching of the bit or a new subroutine call is ignored during execution of the subroutine. No response is sent by the system during execution of the subroutine. The subroutine should send the machine processor a response to cancel the bit (M function, external parameter, etc.). If only one CNC axis group is declared, programme %9999 is called (i.e. %9999.0).
%Wg00.3	ARBUT1 to ARBUT8	Block interrupt on group g.  Set to stop movement on the axis group, then go to the next block or jump to another block. Function G10 associated with its arguments must be present in the part programme.
%Wg00.2	VALID1 to VALID8	Group g enable  Set to enable use of the axis group. Enabling or inhibiting are effective only after a reset or M02.
%Wg00.1	C_FMEXT1 to C_FMEXT8	End of external movement control on group g.  Reset to inhibit cancellation of CYCLE in SINGLE and MDI modes or sequencing to the next block in AUTO and DRYRUN modes. Set to allow normal execution of the mode.  This variable is tested at the end of execution of each block.

Variable	Mnemonic (group 1 to 8)	Description
%Wg00.0	C_AUTAV1 to C_AUTAV8	Feed authorisation on group g  This variable is active if general feed authorisation bit AUTAV = 1. Reset to stop movement on the axis group in all modes with movements. Movement is resumed when C_AUTAVg = 1.
%Wg01.7	C_M011 to C_M018	Enable optional programme stop (M01) on group g of independent CNC axes. A pulse enables or inhibits the optional programme stop by toggling from the previous state.
%Wg01.6	C_SLASH1 to C_SLASH8	Enable block skip on group g of independent CNC axes. A pulse enables or inhibits block skip by toggling from the previous state.
%Wg01.4	C_DGURG1 to C_DGURG8	Emergency retraction request for group g.  This request is accepted in AUTO and SINGLE modes. The current block is interrupted and the system branches to the last emergency retraction programme declared in the part programme by function G75. If no emergency retraction programme is specified, this signal is processed in the same way as C_ARUS.
%Wg01.3	C_RAX1 to C_RAX8	Select axis recall on group g of independent CNC axes.  This request is accepted in AUTO, SINGLE and DRYRUN modes.
%Wg01.2	C_CYCLE1 to C_CYCLE8	Cycle start request on PLC axis group g or independent group  Allows execution of the AUTO and SINGLE modes for the PLC axis groups. A pulse command must be used for C_CYCLEg to prevent resumption of machining after detection of M02 or a reset in the AUTO mode.  This flag is ignored unless the group is valid.
%Wg01.1	C_ARUS1 to C_ARUS8	Request cycle stop on group g of independent CNC axes.  This request is accepted in AUTO, SINGLE, DRYRUN, SEARCH, TEST and MDI modes.
%Wg01.0	C_RAZ1 to C_RAZ8	Reset request on PLC axis group g or independent group  Taken into account if there is no movement on the axes. It is during a reset on a group that flag VALIDg is taken into account and the presence of the part programme assigned to the PLC group is detected.

### 3.8.4.2 Feed Rate Potentiometer Setting: %Wg02.B

Variable	Mnemonic (group 1 to 8)	Description												
%Wg02.B	POTAV1 to POTAV8	Feed Rate Potentiometer Setting Required on Group g  Hexadecimal code corresponding to the ADC input value.  <table style="margin-left: 200px;"> <tr> <th>Hexadecimal</th> <th>ADC input</th> <th>Feed rate override</th> </tr> <tr> <th>code</th> <th>(function anai(.))</th> <th>percentage</th> </tr> <tr> <td>0x0</td> <td>0 Volt</td> <td>0%</td> </tr> <tr> <td>0xFF</td> <td>10 Volt</td> <td>120%</td> </tr> </table>	Hexadecimal	ADC input	Feed rate override	code	(function anai(.))	percentage	0x0	0 Volt	0%	0xFF	10 Volt	120%
Hexadecimal	ADC input	Feed rate override												
code	(function anai(.))	percentage												
0x0	0 Volt	0%												
0xFF	10 Volt	120%												

### 3.8.4.3 Independent Group Mode: %Wg03.B

Variable	Mnemonic (group 1 to 8)	Description
%Wg03.B	MOD-GR1 to MOD-GR8	Mode requested on independent group

### 3.8.5 System Faults and Diagnostic

#### 3.8.5.1 System or Configuration error

The following variables inform the user in case of system or configuration errors.

Variable	Mnemonic	Description
%R97C.W	DEFHTR	Computation time (or RTC) overrun error counter (*)
%R97F.2	DEFCARTE	General I/O card error bit (**)
%R97F.1	DEFCONF	General I/O card configuration error bit (**)
%R97F.0	DEFBUS	General serial I/O bus link error bit (**)

(\*) *This counter is incremented by the system whenever an overrun is detected. It is reset by the user programme.*

(\*\*) *These bits are set by the system whenever an error is detected.*

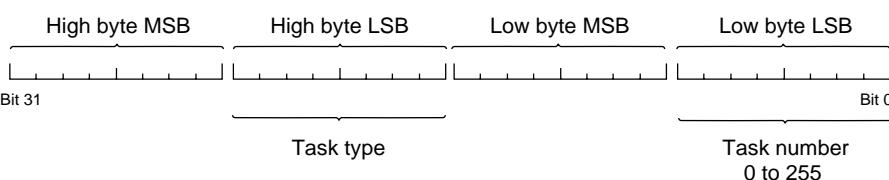
#### 3.8.5.2 System Diagnostic

The following variables give the time occupied (as a percentage of CPU time) by the monitor and each automatic control task.

Variable	Mnemonic	Description
%R950.B	Sys_avr1	Mean time occupied by the monitor on cycle %TS1
%R951.B	Sys_max1	Maximum time occupied by the monitor on cycle %TS1
%R952.B	Ts0_avr1	Mean time occupied by task %TS0 on cycle %TS1
%R953.B	Ts0_max1	Maximum time occupied by task %TS0 on cycle %TS1
%R954.B	Ts1_avr	Mean time occupied by task %TS1
%R955.B	Ts1_max	Maximum time occupied by task %TS1
%R956.W	Overrun1	Computation time overrun on cycle %TS1
%R958.B	Sys_avr2	Mean time occupied by the monitor on cycle %TS2
%R959.B	Sys_max2	Maximum time occupied by the monitor on cycle %TS2
%R95A.B	Ts0_avr2	Mean time occupied by task %TS0 on cycle %TS2
%R95B.B	Ts0_max2	Maximum time occupied by task %TS0 on cycle %TS2
%R95C.B	Ts2_avr	Mean time occupied by task %TS2
%R95D.B	Ts2_max	Maximum time occupied by task %TS2
%R95E.W	Overrun2	Computation time overrun on cycle %TS2
%R960.B	Sys_avr3	Mean time occupied by the monitor on cycle %TS3
%R961.B	Sys_max3	Maximum time occupied by the monitor on cycle %TS3
%R962.B	Ts0_avr3	Mean time occupied by task %TS0 on cycle %TS3
%R963.B	Ts0_max3	Maximum time occupied by task %TS0 on cycle %TS3
%R964.B	Ts3_avr	Mean time occupied by task %TS3
%R965.B	Ts3_max	Maximum time occupied by task %TS3
%R966.W	Overrun3	Computation time overrun on cycle %TS3
%R968.B	Sys_avr4	Mean time occupied by the monitor on cycle %TS4
%R969.B	Sys_max4	Maximum time occupied by the monitor on cycle %TS4
%R96A.B	Ts0_avr4	Mean time occupied by task %TS0 on cycle %TS4

Variable	Mnemonic	Description
%R96B.B	Ts0_max4	Maximum time occupied by task %TS0 on cycle %TS4
%R96C.B	Ts4_avr	Mean time occupied by task %TS4
%R96D.B	Ts4_max	Maximum time occupied by task %TS4
%R96E.W	Overrun4	Computation time overrun on cycle %TS4
%R970.B	Sys_avr5	Mean time occupied by the monitor on cycle %TS5
%R971.B	Sys_max5	Maximum time occupied by the monitor on cycle %TS5
%R972.B	Ts0_avr5	Mean time occupied by task %TS0 on cycle %TS5
%R973.B	Ts0_max5	Maximum time occupied by task %TS0 on cycle %TS5
%R974.B	Ts5_avr	Mean time occupied by task %TS5
%R975.B	Ts5_max	Maximum time occupied by task %TS5
%R976.W	Overrun5	Computation time overrun on cycle %TS5

### 3.8.6 Selecting the Module to Be Animated

Variable	Mnemonic	Description
%W97A.L		Task type and number %W97A.L gives the task type and number of the module to be animated.
		

The low byte gives the task number from 0 to 255.

The high byte gives the task type.

The task type codes are as follows:

- 1 for a %TS task
- 2 for a %TF task
- 3 for an %SP task
- 4 for a %TH task
- 5 for an %INI task

%W97E.B	Component number %W97E.B gives the component number to be animated in the module.
---------	--

If these two variables are mutually consistent, the component of the module specified is opened and animated. Otherwise, the list of all the modules loaded on the PLC is proposed.

#### Example

%W97A.L = 0x00300F0

%W97E.B = 2

Component 2 of module SP240 is opened and animated.

### 3.8.7 Output Card Write Enable: %W900.0

Variable	Mnemonic	Description
%W900.0	INIB_E33	Write of output cards by part programme enabled. Variables %Qrc3B.1 must already be programmed in %INI. Set to inhibit write of variables %Qrc in a part programme by parameters E33xxx or by dynamic operators. Reset to enable write.

### 3.8.8 System Fault Management

The following variables control actions of the monitor when system or configuration faults are detected.

The system fault management variables will be covered later.

### 3.8.9 External Parameters E30xxx, E40xxx and E42xxx

#### ⚠ CAUTION

Parameters E30xxx and E40xxx are not saved. They are reset at power on.

Parameters E42xxx are saved.

#### 3.8.9.1 External Parameters E30xxx

128 words x 32 bits are addressed by E30000 to E30127.

Parameters E300xx are written by the part programme. They contain significant signed numerical values. They can be read from and written to by the user programme.

Mnemonic	MSB			LSB
E30000 to E30031	%RA00	%RA01	%RA02	%RA03
E30032 to E30063	%RB00	%RB01	%RB02	%RB03
E30064 to E30095	%RC00	%RC01	%RC02	%RC03
E30096 to E30127	%RD00	%RD01	%RD02	%RD03
	%RD7C	%RD7D	%RD7E	%RD7F

### 3.8.9.2 External Parameters E40xxx

128 words x 32 bits are addressed by E40000 to E40127.

Parameters E400xx are written by the user programme. They are used to include signed numerical values which can be dimensions, offset, etc. for use in the part programme.

External parameter	MSB			LSB
E40000 to E40031	%WA00 %WA7C	%WA01 %WA7D	%WA02 %WA7E	%WA03 %WA7F
E40032 to E40063	%WB00 %WB7C	%WB01 %WB7D	%WB02 %WB7E	%WB03 %WB7F
E40064 to E40095	%WC00 %WC7C	%WC01 %WC7D	%WC02 %WC7E	%WC03 %WC7F
E40096 to E40127	%WD00 %WD7C	%WD01 %WD7D	%WD02 %WD7E	%WD03 %WD7F

### 3.8.9.3 Parameters E42xxx

128 words addressed from E42000 to E42127. These parameters can be read from and written to by the user programme (functions R\_E42000(..) and W\_E42000(..)) and by the part programme. They are accessible for read and write using dynamic operators.

**REMARK** *There is no guarantee of coherence of the exchanges at system level (for instance, read by the machine processor can be interrupted by write by the CNC processor). It is therefore up to the user to provide a secure exchange mechanism.*

### 3.8.10 Physical Organisation of %R and %W Variables

The %R and %W variables are organised in blocks of 128 %R bytes followed by 128 %W bytes then again 128 %R bytes and so forth until the end of the family.

#### Reserved Unassigned Variables

Input variables %RE00 to %RE7F and %RF00 to %RF7F are reserved but not assigned.

Output variables %WE20 to %WE7F and %WF00 to %WF7F are reserved but not assigned.

#### Table

Physical organisation of the %R and %W variables (total 4 kbytes).

Variables	Description
%R0 to %R7F	128 input bytes from the CNC
%W0 to %W7F	128 output bytes to the CNC
%R100 to %R17F	128 input bytes from axis group 1
%W100 to %W17F	128 output bytes to axis group 1
%Rg00 to %Rg7F	6 groups of 128 input bytes from axis groups 2 to 7
%Wg00 to %Wg7F	6 groups of 128 output bytes to axis groups 2 to 7
%R800 to %R87F	128 input bytes from axis group 8
%W800 to %W87F	128 output bytes to axis group 8
%R900 to %R97F	128 input bytes (internal faults)
%W900 to %W97F	128 output bytes (internal faults)
%RA00 to %RA7F	128 input bytes for parameters E30000 to E30031
%WA00 to %WA7F	128 output bytes for parameters E40000 to E40031
%RB00 to %RB7F	128 input bytes for parameters E30032 to E30063
%WB00 to %WB7F	128 output bytes for parameters E40032 to E40063
%RC00 to %RC7F	128 input bytes for parameters E30064 to E30095
%WC00 to %WC7F	128 output bytes for parameters E40064 to E40095
%RD00 to %RD7F	128 input bytes for parameters E30096 to E30127
%WD00 to %WD7F	128 output bytes for parameters E40096 to E40127
%WE00 to %WE1F	32 output bytes to the CNC, current reduction
%WF20 to %WF7F	reserved, unassigned
%RF00 to %RF7F	reserved, unassigned

## 3.9 %S Common Word Variables

When the CNC is connected to MAPWAY or ETHWAY networks, it allows access to the common words of the Telemecanique TSX PLCs. The set of common words forms a data base shared by the stations of a network in which each station can be a TSX PLC or a numerical control.

The stations included in the common word service share a common memory of 256 words x 16 bits.

Depending on the configuration, each station is allocated from 4 to 64 common words (accessible for write) of the common memory. It has read-only access to the words assigned to the other stations.

### 3.9.1 Variable Update

The %S variables are updated automatically by the system at the rate of sequential task %TS0, without action by the user programme.

At the start of %TS0, the automatic control function reads all the common words updated in the other stations in the interface associated with the network processor.

At the end of %TS0, the automatic control function writes the common words of its station in the interface associated with the network processor.

The network controller compares these values with the values sent before. It only sends a frame if at least one of the values has been updated or after 30 RTC cycles if it has not transmitted since.

### 3.9.2 Setting up the Common Words

Setting up consists of:

- defining the network and station numbers in machine parameter P100 (see Parameter Manual),
- programming the station activity and the number of common words per station in task %INI by calling function setcomw(..).

### 3.9.3 Organisation of %S Common Word Variables

The %S variables are organised in 64 128-byte blocks independently of the common word setup.

The number of an %S variable is encoded on four hexadecimal digits. The two low digits indicate the byte number in the station (from 0x0 to 0x7F) and the two high digits indicate the station number (from 0x0 to 0x3F). For instance, %S21F.B represents byte 31 of station 2.

Block	Variables	Size
Station 0	%S0 to %S7F	128 bytes
Station 1	%S100 to %S17F	128 bytes
Stations 2-61		59 blocks of 128 bytes
Station 62 (0x3E)	%S3E00 to %S3E7F	128 bytes
Station 63 (0x3F)	%S3F00 to %S3F6F	112 bytes
Diagnostic	%S3F70 to %S3F7F	16 bytes

Variables %S3F70.B to %S3F77.B contain the station update flags:

Variables	Description
%S3F70.0 to %S3F70.7	Flags for stations 0 to 7
%S3F71.0 to %S3F71.7	Flags for stations 8 to 15
%S3F72.0 to %S3F72.7	Flags for stations 16 to 23
%S3F73.0 to %S3F73.7	Flags for stations 24 to 31
%S3F74.0 to %S3F74.7	Flags for stations 32 to 39
%S3F75.0 to %S3F75.7	Flags for stations 40 to 47
%S3F76.0 to %S3F76.7	Flags for stations 48 to 55
%S3F77.0 to %S3F77.7	Flags for stations 56 to 63

These flags are set by the system after update of the %S variables for the corresponding station. They can be reset by the programmer to check for correct transfer operation.

Byte %S3F79B contains the number of its own station when the common word service is active.

Word %S3F7E.W is reserved for NUM product support.

**REMARK** *If the common word service is not active, the %S variables can be used as common unsaved variables.*

## 3.10 %Y Local Variables- Pointers

### 3.10.1 General

A data base of the microprocessor is available to the programmer. This base is used for %Y variables.

The %Y variables are used in two different ways:

- as local variables when associated with an %SP module. In this case, the base is initialised by the system when an %SP module is called by function spy(..). The %Y variables are created in the stack during the call to module %SP and are deleted on return to the calling programme. They number 128 bytes (from %Y0.B to %Y7F.B). They can be used to write relocatable, reentrant modules.
- as variables that can be substituted for any global variables (%M, %V, %I, %Q, %R and %W). In this case, the programmer must index the base on the beginning of the area concerned by function y\_init(..). The %Y variables give access to a 32767-byte field (%Y0.B to %Y7FFF.B). They are useful, for instance, when the same processing must be carried out on blocks of different variables.

In addition, the %Y variables allow indirect addressing or addressing by pointers.

**REMARKS** *The %Y variables are not essential for programming and their use is reserved for experienced programmers.*

*The %Y variables cannot be displayed on the CNC screen and the PLCTOOL programming tool.*

*The %Y variables are not accessible by a UNITE request.*

*Use of function y\_init(..) inhibits visibility of any local variables of the modules.*

### 3.10.2 Indirect Addressing - Pointers

Indirect addressing by pointer is allowed whenever a simple variable can be used, except for indexes.

#### CAUTION

Before using addressing by pointer %Yi ->, it is necessary for:

the %Y variables to be defined, i.e. to be in a %SP called with function spy() or for the base register of variables %Y to be defined by function y\_init()  
pointer %Yi.L to be loaded with a valid address.

A variable pointed to can be associated with a mnemonic (see PLCTOOL - Programming Tool in Ladder Language Manual).

To optimise speed, it is recommended to use numbers that are multiples of 4 for the pointers (e.g. %Y0 ->, %Y4 ->, %Y8 ->, %YC ->, etc.).

#### Syntax

```
<pointer> -> <post-offset>.<size>
```

Language element	Includes	Comment
<pointer>	%Y0 to %Y7C	%Y variable of size .L (the size is not specified)
<post-offset>	0 to ff	Immediate value (hexadecimal)
<size>	.0 to .7, .B, .W or .L	To access a bit, byte, word or long word variable

### Example

%Y4 -> 0.5                                  The address of the variable pointed to is equal to the address contained in the pointer + post-offset «0».

%Y7C -> ff.B                                  The address of the variable pointed to is equal to the address contained in the pointer + post-offset «0xff».

### 3.10.3 Examples of Use of Pointers

#### Processing of a Character String

```
%V500.L = «ABCDEF»                        // %V500.L contains the start address of string «ABCDEF»
%Y8.L = %V500.L                            // Pointer initialised with string start address
%Y8 -> 0.B == «A»                        // Access to the first character in the string
%Y8 -> 5.B == «F»                        // Access to the sixth character in the string
%Y8.L += 1                                // Pointer increment
%Y8 -> 0.B == «B»                        // Access to the second character in the string
```

#### Management of Four Machine Panels

**REMARK:** *The sample programme PUPITREP available under PLCTOOL illustrates the use of pointers.*

##### In a %TS

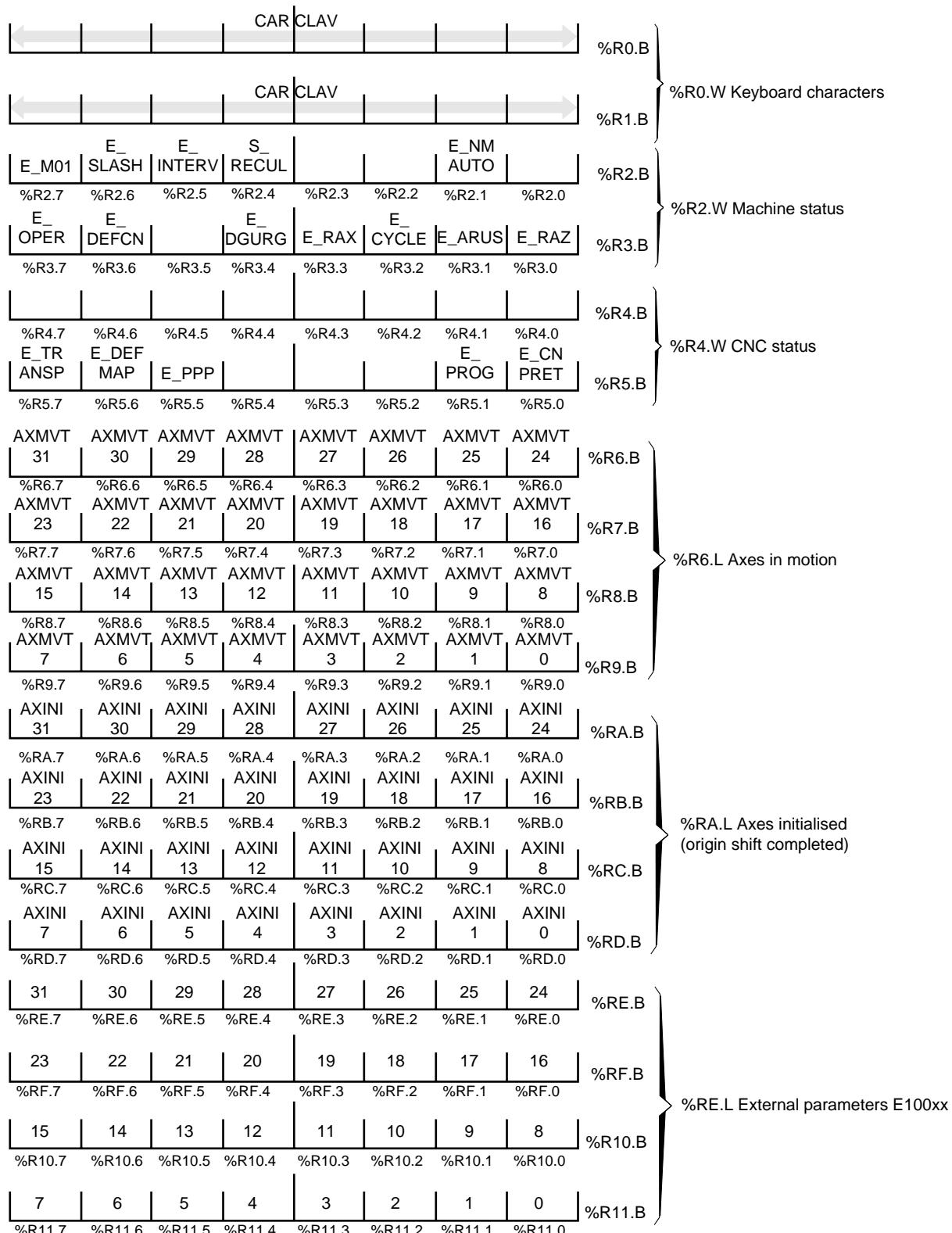
```
spy(0, %IrcOO.&, %Qrc00.&)    // Branch to %SP0 (where rc == panel number from 1 to 4)
```

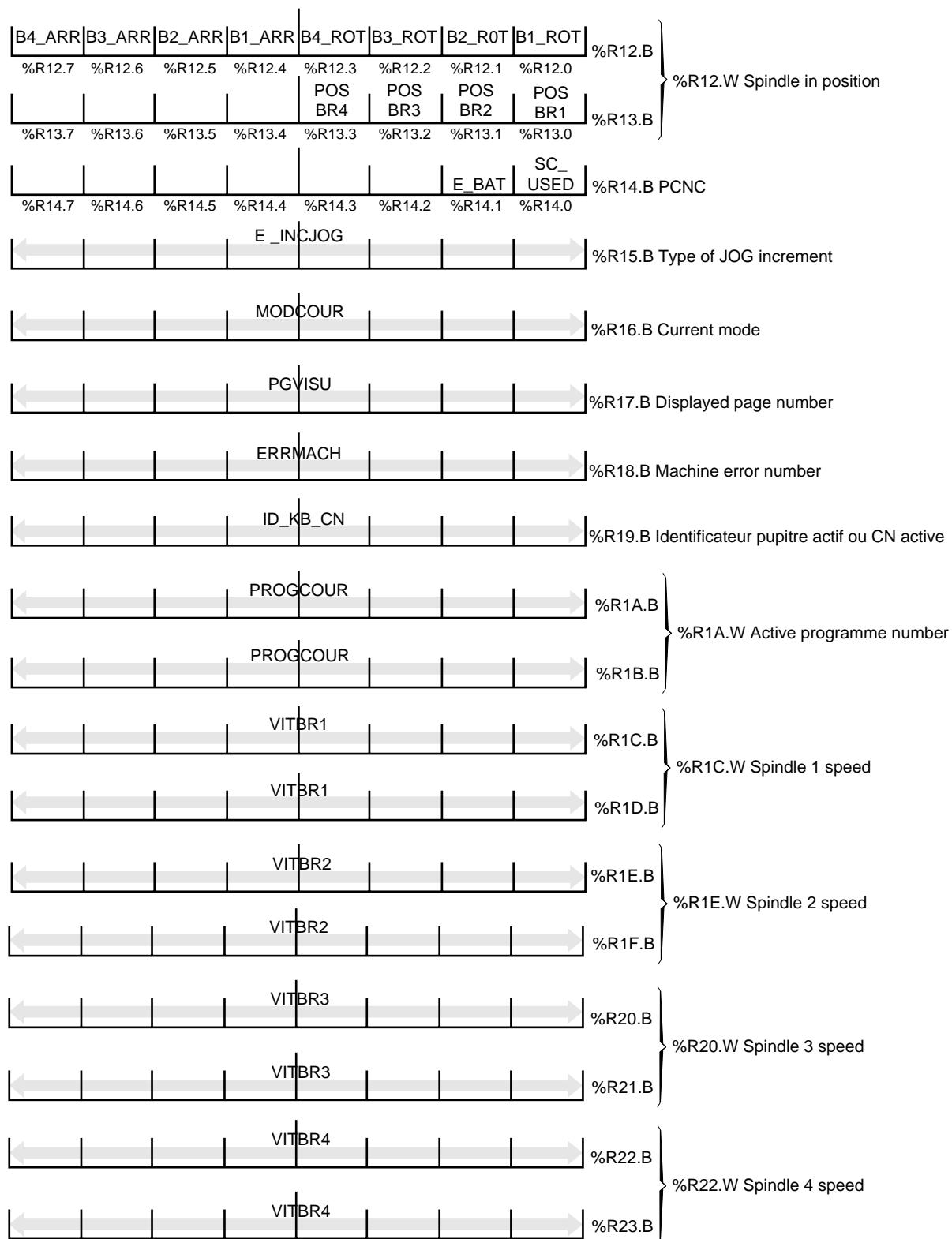
##### In %SP0

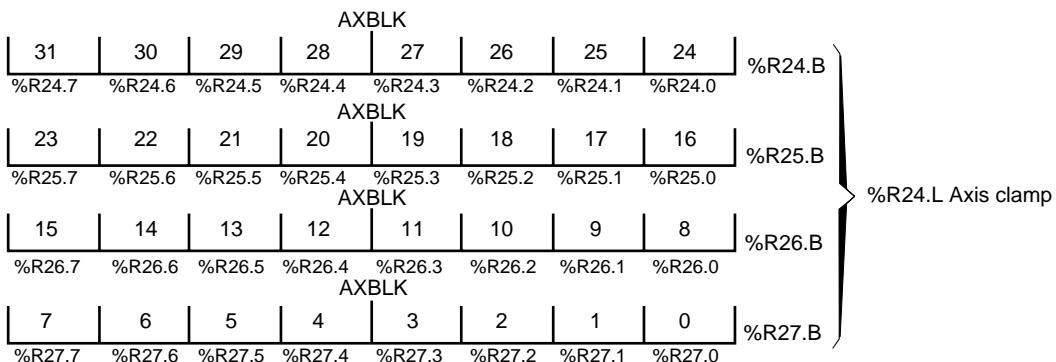
```
// contains the address of the first input %Irc00
// contains the address of the first output %Qrc00
%Y0 -> 2.0                                <==> %Irc2.0      Jog_1
%Y0 -> 2.1                                <==> %Irc2.1      Jog_10
%Y0 -> 2.2                                <==> %Irc2.2     Jog_100
%Y0 -> 20.W                               <==> %Irc20.W    Spindle potentiometer
%Y0 -> 22.W                               <==> %Irc22.W    Feed rate potentiometer
%Y4 -> 0.0                                <==> %Qrc0.0     Led_arus
%Y4 -> 0.1                                <==> %Qrc0.1     Led_dcy
%Y4 -> 1.0                                <==> %Qrc1.0     Led_1
%Y4 -> 1.1                                <==> %Qrc1.1     Led_10
```

## 3.11 Exchange Area

### 3.11.1 Inputs from the CNC

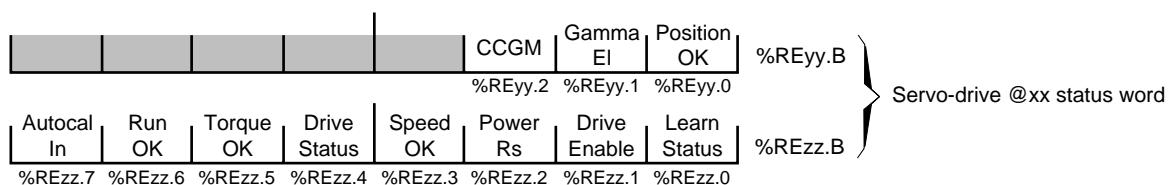






### 3.11.2 CNC-PLC Exchange Area - 1050

For the digital servo-drive at address xx (xx between 00 and 31), the status word format is as follows:

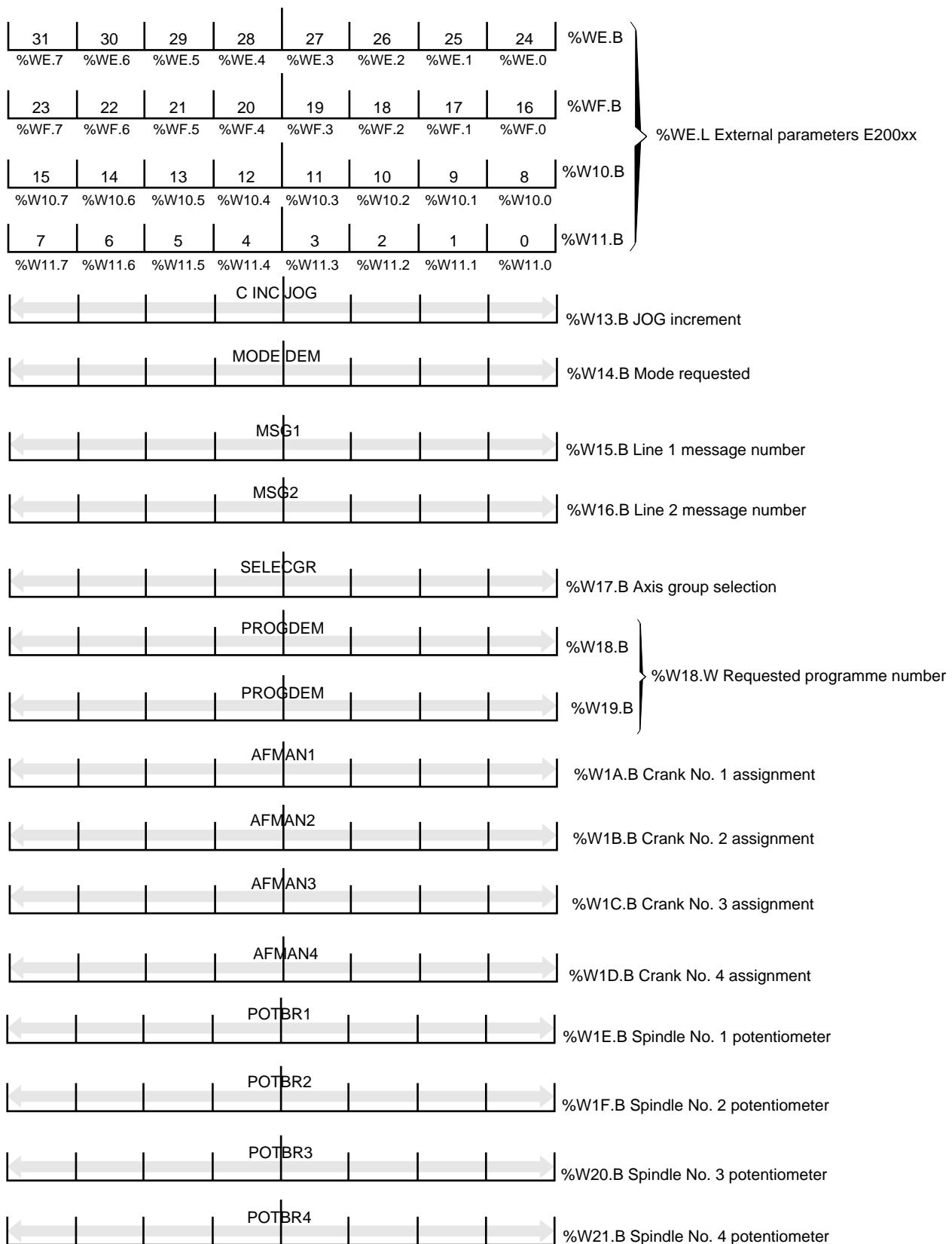


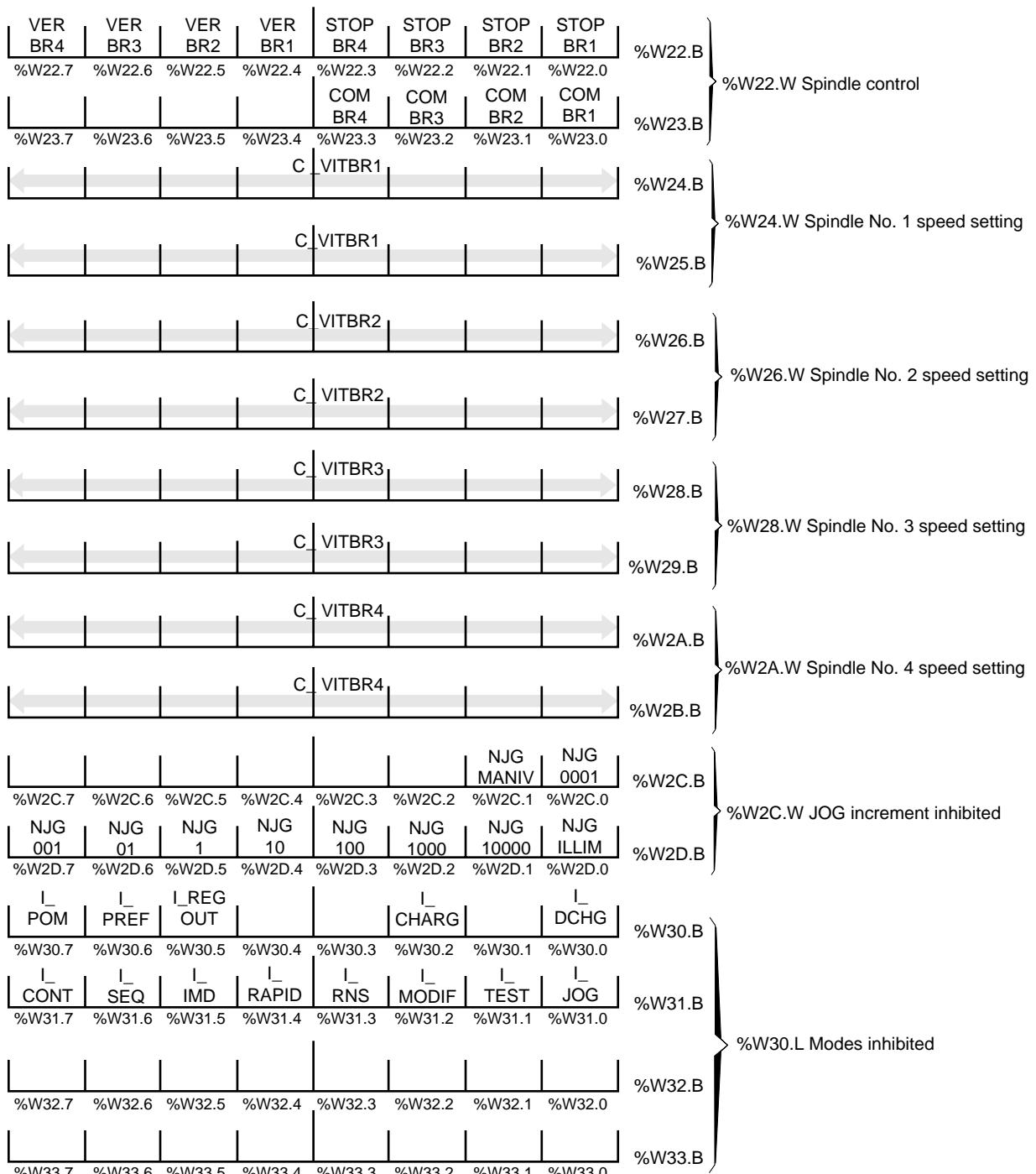
Values of yy and zz according to the address of servo-drive xx:

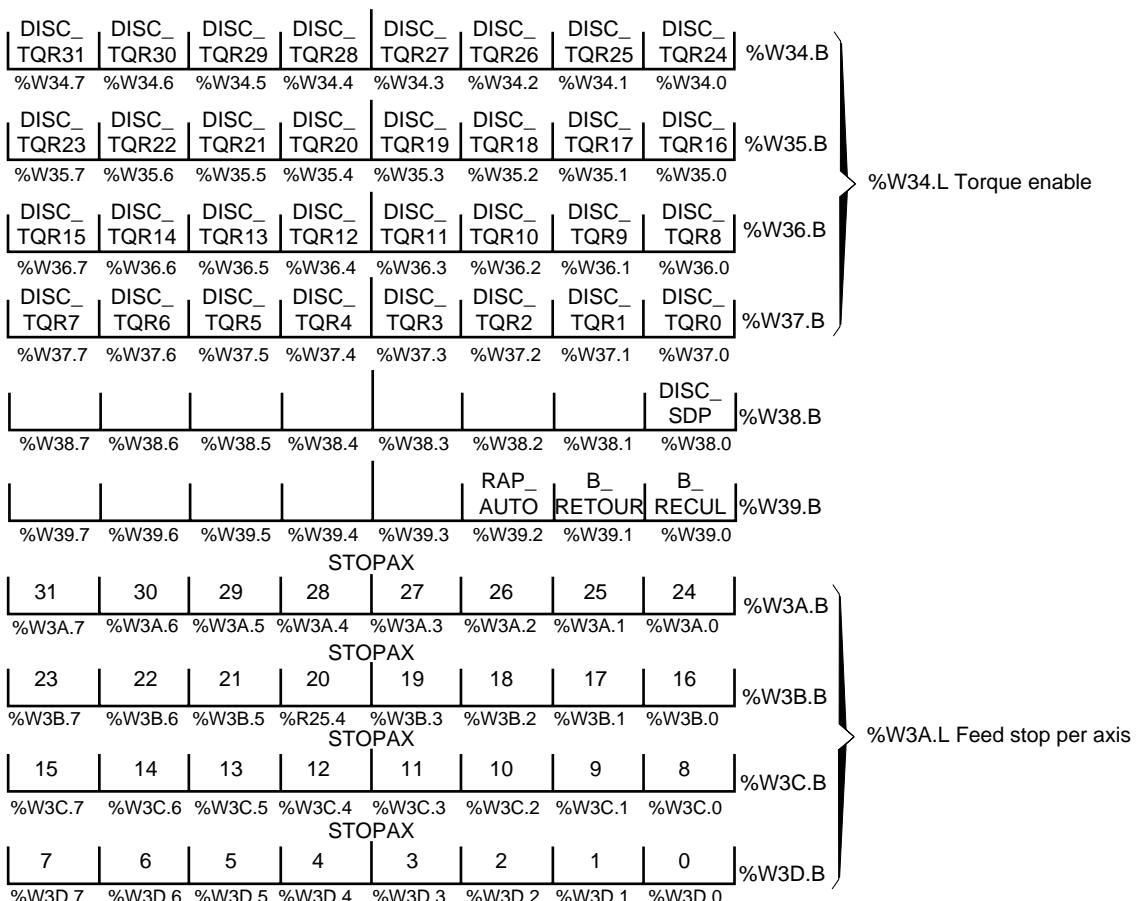
xx	00	01	02	03	04	05	06	07	08	09	10
yy	20	22	24	26	28	2A	2C	2E	30	32	34
zz	21	23	25	27	29	2B	2D	2F	31	33	35
xx	11	12	13	14	15	16	17	18	19	20	21
yy	36	38	3A	3C	3E	40	42	44	46	48	4A
zz	37	39	3B	3D	3F	41	43	45	47	49	4B
xx	22	23	24	25	26	27	28	29	30	31	
yy	4C	4E	50	52	54	56	58	5A	5C	5E	
zz	4D	4F	51	53	55	57	59	5B	5D	5F	

### 3.11.3 Output to the CNC

				CHG_OPDC	C_INDG	C_NM_AUTO	KB_INIT	%W2.B
%W2.7	%W2.6	%W2.5	%W2.4	%W2.3	%W2.2	%W2.1	%W2.0	
C_M01	C_SLASH	C_RAZER	C_DGURG	C_RAX	C_CYCLE	C_ARUS	C_RAZ	%W3.B
%W3.7	%W3.6	%W3.5	%W3.4	%W3.3	%W3.2	%W3.1	%W3.0	
V_REDUIT	INIB_UTIL	C_UNIT	PRES_PUIS	NAR_FIB	VIT_MAN2	VIT_MAN1	AUT_AV	%W4.B
%W4.7	%W4.6	%W4.5	%W4.4	%W4.3	%W4.2	%W4.1	%W4.0	
SC_SAVE	SK_DISPCLAV	INIB_CLAV	IM_PULS	COR_DYN	JOG_PUP	MOD_PUP	PUP_ABS	%W5.B
%W5.7	%W5.6	%W5.5	%W5.4	%W5.3	%W5.2	%W5.1	%W5.0	
JOG_POS31	JOG_POS30	JOG_POS29	JOG_POS28	JOG_POS27	JOG_POS26	JOG_POS25	JOG_POS24	%W6.B
%W6.7	%W6.6	%W6.5	%W6.4	%W6.3	%W6.2	%W6.1	%W6.0	
JOG_POS23	JOG_POS22	JOG_POS21	JOG_POS20	JOG_POS19	JOG_POS18	JOG_POS17	JOG_POS16	%W7.B
%W7.7	%W7.6	%W7.5	%W7.4	%W7.3	%W7.2	%W7.1	%W7.0	
JOG_POS15	JOG_POS14	JOG_POS13	JOG_POS12	JOG_POS11	JOG_POS10	JOG_POS9	JOG_POS8	%W8.B
%W8.7	%W8.6	%W8.5	%W8.4	%W8.3	%W8.2	%W8.1	%W8.0	
JOG_POS7	JOG_POS6	JOG_POS5	JOG_POS4	JOG_POS3	JOG_POS2	JOG_POS1	JOG_POS0	%W9.B
%W9.7	%W9.6	%W9.5	%W9.4	%W9.3	%W9.2	%W9.1	%W9.0	
JOG_NEG31	JOG_NEG30	JOG_NEG29	JOG_NEG28	JOG_NEG27	JOG_NEG26	JOG_NEG25	JOG_NEG24	%WA.B
%WA.7	%WA.6	%WA.5	%WA.4	%WA.3	%WA.2	%WA.1	%WA.0	
JOG_NEG23	JOG_NEG22	JOG_NEG21	JOG_NEG20	JOG_NEG19	JOG_NEG18	JOG_NEG17	JOG_NEG16	%WB.B
%WB.7	%WB.6	%WB.5	%WB.4	%WB.3	%WB.2	%WB.1	%WB.0	
JOG_NEG15	JOG_NEG14	JOG_NEG13	JOG_NEG12	JOG_NEG11	JOG_NEG10	JOG_NEG9	JOG_NEG8	%WC.B
%RC.7	%WC.6	%WC.5	%WC.4	%WC.3	%WC.2	%WC.1	%WC.0	
JOG_NEG7	JOG_NEG6	JOG_NEG5	JOG_NEG4	JOG_NEG3	JOG_NEG2	JOG_NEG1	JOG_NEG0	%WD.B
%WD.7	%WD.6	%WD.5	%WD.4	%WD.3	%WD.2	%WD.1	%WD.0	



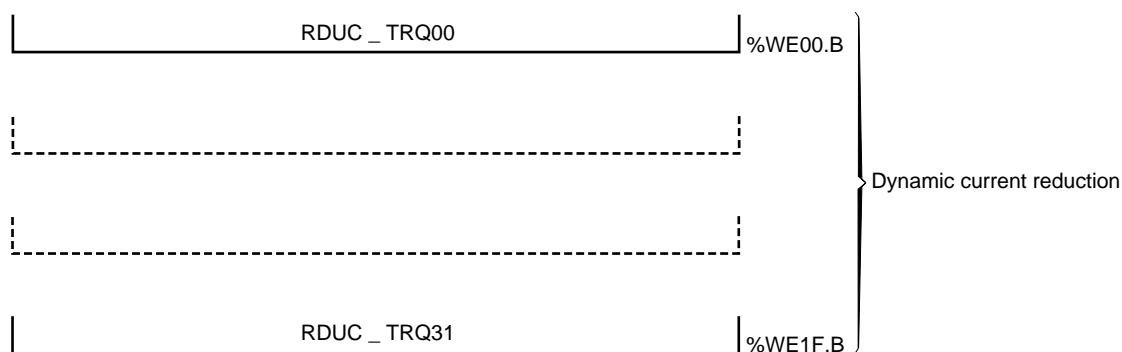




### 3.11.4 PLC-CNC Exchange Area - 1050

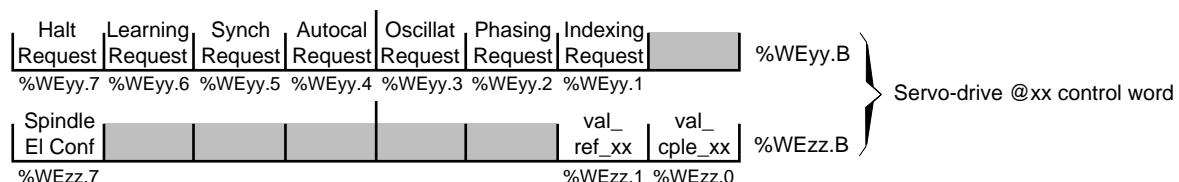
#### 3.11.4.1 Torque Modulation

The PLC can dynamically reduce the maximum current selectively for each digital servo-drive.



#### 3.11.4.2 Servo-Drive Control Word

For the digital servo-drive at address xx (xx between 00 and 31), the control word format is as follows:

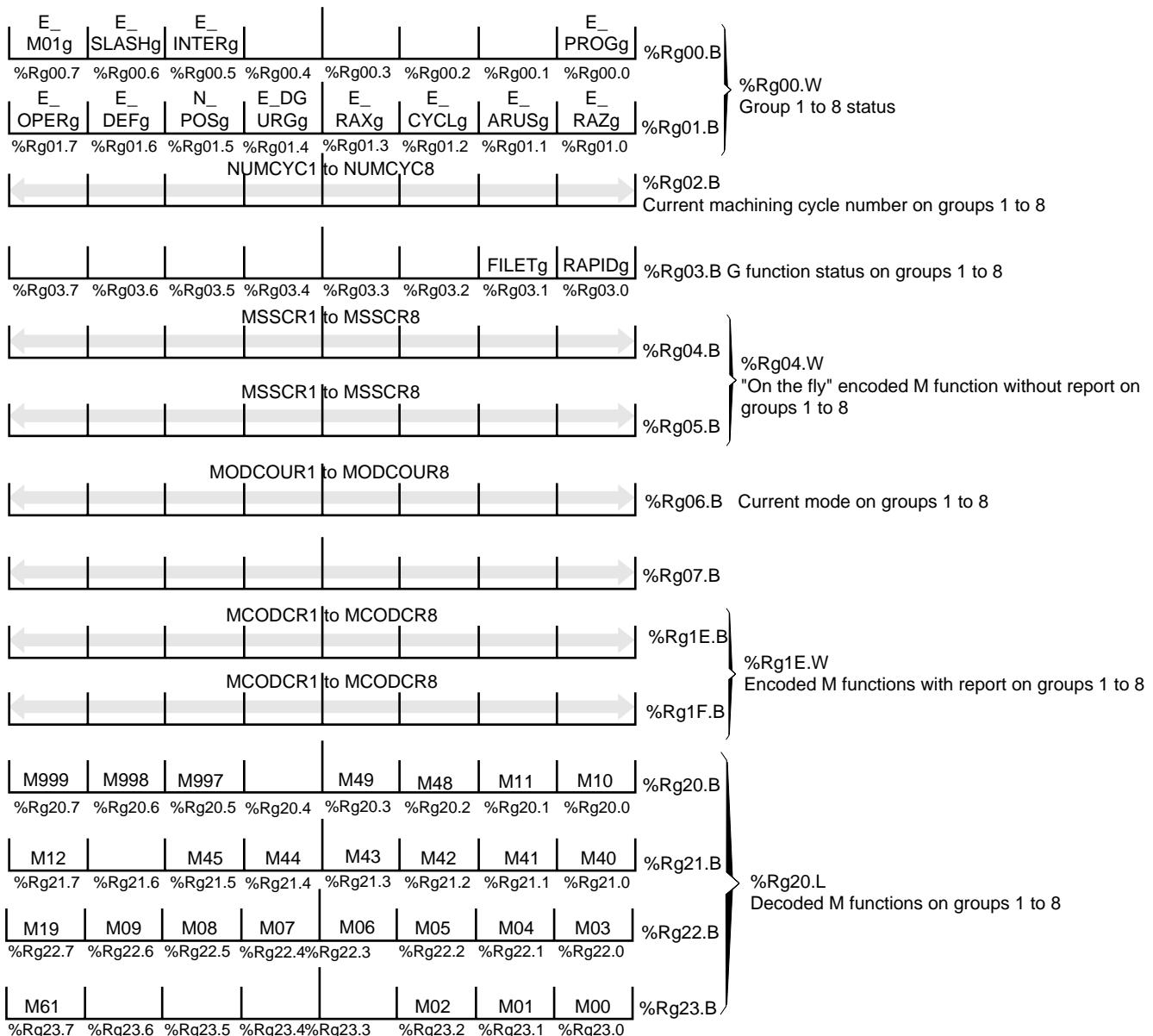


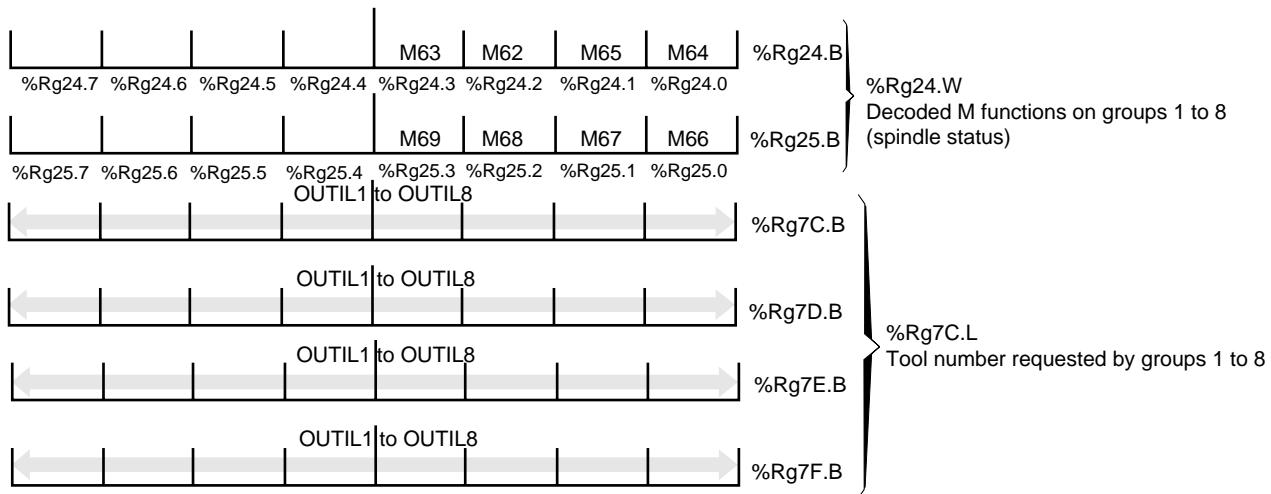
Values of yy and zz according to the address of servo-drive xx:

xx	00	01	02	03	04	05	06	07	08	09	10
yy	20	22	24	26	28	2A	2C	2E	30	32	34
zz	21	23	25	27	29	2B	2D	2F	31	33	35
xx	11	12	13	14	15	16	17	18	19	20	21
yy	36	38	3A	3C	3E	40	42	44	46	48	4A
zz	37	39	3B	3D	3F	41	43	45	47	49	4B
xx	22	23	24	25	26	27	28	29	30	31	
yy	4C	4E	50	52	54	56	58	5A	5C	5E	
zz	4D	4F	51	53	55	57	59	5B	5D	5F	

### 3.11.5 Inputs from Axis Groups

In this chart, g takes on the value of the axis group number (1 to 8)





### 3.11.6 Outputs to Axis Groups

In this chart, g takes on the value of the axis group number (1 to 8)



\* Valid only for PLC axis groups

\*\* Valid only for CNC axis groups





## 4 Literal Elements of Ladder Language

<b>4.1 Notations Used</b>	4 - 3
<b>4.2 Label - Comment</b>	4 - 3
<b>4.3 Step</b>	4 - 3
<b>4.4 Literal Elements of Ladder Sequences</b>	4 - 3
4.4.1 Literal Entities Authorised in the Test Part of a Ladder	4 - 3
4.4.2 Literal Entities Authorised in the Action Area of a Ladder	4 - 4
4.4.3 Grammar of the Literal Elements	4 - 4
<b>4.5 Additional Information on Literal Elements</b>	4 - 5
4.5.1 Priority of Operators	4 - 5
4.5.1.1 Priority of the Unary Operators	4 - 5
4.5.1.2 Priority of Binary and Comparison Operators	4 - 5
4.5.2 Comparison Operators	4 - 6
4.5.3 >> and << Operators	4 - 6
4.5.4 Assignment Operators	4 - 6
4.5.4.1 Operator =	4 - 6
4.5.4.2 Combined Operators +=, -=, &-, ^=,  =	4 - 6
4.5.5 Order of Evaluation of the Expressions	4 - 7
4.5.6 Immediate Integers	4 - 7
4.5.7 Propagation of Variables - Format of Internal Calculations	4 - 7
4.5.8 Overflow - Sign Change	4 - 9
4.5.9 Examples of Literal Entities	4 - 9
4.5.10 Maximum Length of a Literal Entity	4 - 10
4.5.11 Maximum Number of Operands in a Numerical Expression	4 - 10



## 4.1 Notations Used

The notations used to describe the literal elements of the ladder language are as follows:

Character	Function
[ ]	At most one element between square brackets can be chosen
< >	Surround nonterminal elements of the language
{ } n	At most n elements between parentheses can be chosen.

**REMARK** An element not surrounded by < and > is a terminal symbol, a keyword or a separator.

## 4.2 Label - Comment

Language element	Includes	Remark
<label>	<letter> or <number> or _	Limited to 8 characters
<comment>	<character> or <space>	Limited to 64 characters

## 4.3 Step

Language element	Includes	Remark
<step>	<step_variable><step_number> <step_variable> <step_number>	%M variable with size .W, %V or % Y Positive integer on 16 bits

## 4.4 Literal Elements of Ladder Sequences

### 4.4.1 Literal Entities Authorised in the Test Part of a Ladder

Language element	Includes	Remark
<bit_variable>	Variable % .0 to .7	Example: %V3.0
<comparison>	<numerical_expression><comparison_operator><numerical_expression>	
<numerical_assignment>	<numerical_variable><assignment operator> <numerical_expression>	
<function_call>	[<numerical_variable><assignment_operator>] <function>	

**REMARK** The evaluation of <bit\_variable> and <comparison> supplies a Boolean result [1/0].

#### 4.4.2 Literal Entities Authorised in the Action Area of a Ladder

Language element	Includes	Remark
<bit_variable>	Variable % .0 to .7	Example: %V3.0
<numerical_assignment>	<numerical_variable><assignment_operator> <numerical_expression>	
<function_call>	[<numerical_variable><assignment_operator>] <function>	
<goto_label>	goto(<label>)	Jump to label (internal in module) without possible return
<call_label>	call(<label>)	Jump to label (internal in module) with return
<return>	return([<numerical_expression>])	Return to calling module or call(<label>)

#### 4.4.3 Grammar of the Literal Elements

Language element	Includes	Remark
<function>	<function_name>() or <function_name>(<numerical_expression>) or <function_name>({<numerical_expression>},)6 <numerical_expression>	
<function_name>		Example: printf(....)
<numerical_expression>	<signed_number>{<binary_operator> <signed_number>}n	For determination of n, see Sec. 4.5.
<signed_number>	[<unary_operator>]<unsigned_number>	
<unsigned_number>	<numerical_variable> or <immediate_integer> or (<numerical_expression>)	
<immediate_integer>	<digit>{<digit>}9 or 0x<hexdigit>{<hexdigit>}7	base ten base sixteen
<digit>	0, 1, 2, 3, 4, 5, 6, 7, 8 or 9	
<hexdigit>	<number>, a, b, c, d, e, f, A, B, C, D, E or F	
<numerical_variable>	Variable %B or .W or .L or .&	Example: %V3.L
<comparison_operator>	== != >= <= > <	Equal Unequal Greater than or equal (signed comparison) Less than or equal (signed comparison) Greater (signed comparison) Less (signed comparison)
<unary_operator>	- ~	Negation of the operand that follows inversion bit by bit of the operand that follows

Language element	Includes	Remark
<binary_operator>	*	Multiplication (signed)
	/	Division (signed)
	+	Addition
	-	Subtraction
	<<	Arithmetic shift left
	>>	Arithmetic shift right
	&	AND bit by bit
	^	EXCLUSIVE OR bit by bit
		OR bit by bit
<assignment_operator>	=	Simple assignment
	+=	Add and assign
	-=	Subtract and assign
	&=	AND bit by bit and assign
	^=	EXCLUSIVE OR bit by bit and assign
	=	OR bit by bit and assign

## 4.5 Additional Information on Literal Elements

### 4.5.1 Priority of Operators

#### 4.5.1.1 Priority of the Unary Operators

Unary operators have higher priority than binary operators.

Priority	Operator	Description
Highest	[]	Indexing
	.&	«Address of» operator
	-	Negation
Lowest	~	Bit-by-bit inversion

#### 4.5.1.2 Priority of Binary and Comparison Operators

The binary and comparison operators have higher priority than the assignment operators.

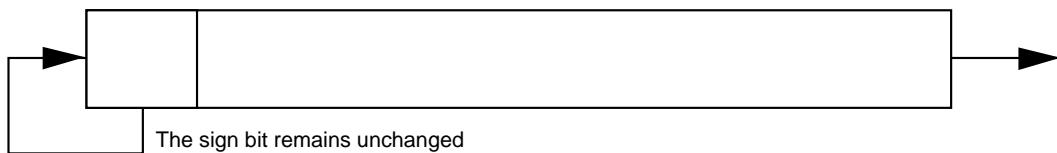
Priority	Operator	Description
Highest	P0      * /	Multiplication Division
	P1      + -	Addition Subtraction
	P2      >> <<	Arithmetic shift left Arithmetic shift right
	P3      &	AND bit by bit
	P4      ^	EXCLUSIVE OR bit by bit
	P5	OR bit by bit
Lowest	P6      ==!=>=<=><	Comparison operators

#### 4.5.2 Comparison Operators

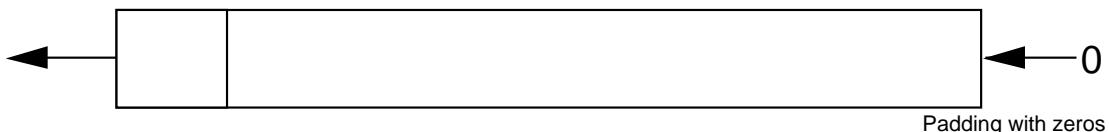
Comparisons are signed, i.e. the variable MSB is considered as sign bit (variables whose sign bit is set are lower than variables whose sign bit is a zero).

#### 4.5.3 >> and << Operators

>> Arithmetic Shift Right by N Modulo 64 bits



<< Arithmetic Shift Left by N Modulo 64 bits



**REMARKS** Allows divisions by powers of 2 more rapidly than with the «/» operator:  
/ (Var\_1 /  $2^n$  == Var\_1 >> n).

Allows multiplications by powers of 2 more rapidly than with the «\*» operator:  
/ (Var\_1 \*  $2^n$  == Var\_1 << n).

#### 4.5.4 Assignment Operators

Assignment operators have the lowest priority. Assignment is therefore carried out last.

##### 4.5.4.1 Operator =

Simple assignment loads the variable on the left with the result of the numerical expression or function on the right of assignment operator =.

##### 4.5.4.2 Combined Operators +=, -=, &=, ^=, |=

These operators combine an operation between the variable on the left and the results of the expression on the right followed by assignment of the final result in the variable on the left.

##### Examples:

Var\_1 += <numerical\_expression> is equivalent to Var\_1 = Var\_1 + <numerical\_expression>

Var\_1 -= <numerical\_expression> is equivalent to Var\_1 = Var\_1 - <numerical\_expression>

Var\_1 &= <numerical\_expression> is equivalent to Var\_1 = Var\_1 & <numerical\_expression>

Var\_1 ^= <numerical\_expression> is equivalent to Var\_1 = Var\_1 ^ <numerical\_expression>

Var\_1 |= <numerical\_expression> is equivalent to Var\_1 = Var\_1 | <numerical\_expression>

Combined operators are recommended since the code they generate is optimised as regards speed of execution and size.

#### 4.5.5 Order of Evaluation of the Expressions

In an expression, the highest priority operations are executed before the lower priority operations.

Operations with the same priority are executed from left to right.

Brackets are used to modify the order of evaluation of the expressions by forcing evaluation of the expression in brackets first.

#### 4.5.6 Immediate Integers

Immediate integers are limited to 32 bits.

The system considers integers to be signed. Bit 31 is the sign bit.

An immediate integer must be between:

Type	Value
Negative hexadecimal integer	from -2147483648 to -1
Negative decimal integer	from 0x80000000 to 0xFFFFFFFF
Positive decimal integer	from 0 to 2147483647
Positive hexadecimal integer	from 0x0 to 0x7FFFFFFF

#### 4.5.7 Propagation of Variables - Format of Internal Calculations

The system considers all variables to be signed.

##### Byte Variables

Bit 7 is the sign bit.  $-128 \leq \text{byte value} \leq +127$ .

##### Word Variables

Bit 15 is the sign bit.  $-32768 \leq \text{word value} \leq +32767$ .

##### Long Word Variables

Bit 31 is the sign bit.  $-2147483648 (-2^{31}) \leq \text{long word value} \leq +2147483647 (2^{31} - 1)$ .

##### Operation

When a variable is used in a calculation, it is first loaded into a microprocessor register.

If the variable loaded is a byte, the system propagates bit 7 of the register onto bits 8 to 31.

If the variable loaded is a word, the system propagates bit 15 of the register onto bits 16 to 31.

The calculations are then made with the 32-bit registers and generate a result on 32 bits.

The result is then loaded in the destination variable:

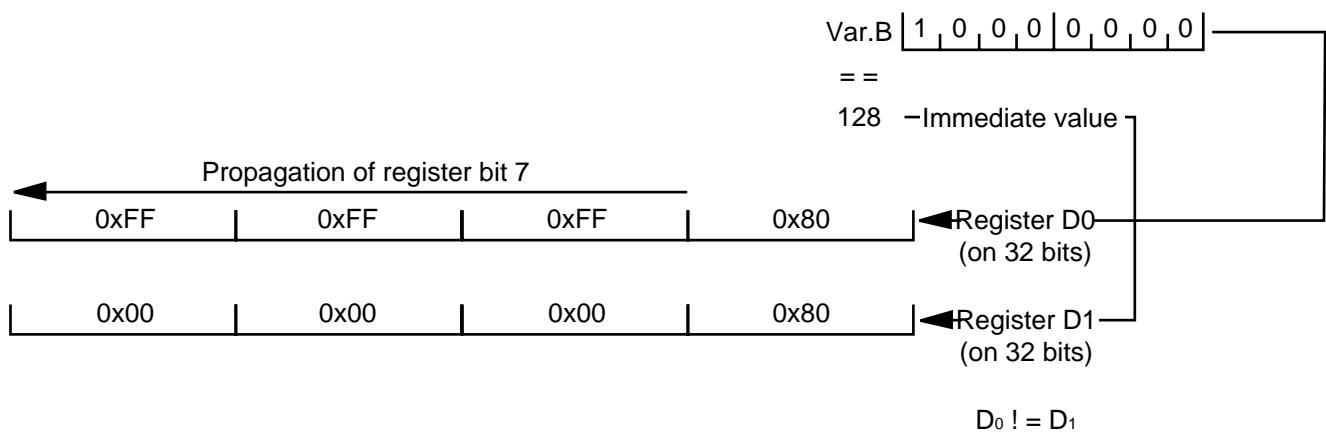
- if the destination variable is a long word, the 32 bits of the result register are loaded into it,
- if the destination variable is a word, the 16 LSBs of the result register are loaded into it,
- if the destination variable is a byte, the 8 LSBs of the result register are loaded into it.

## Pitfall to be Avoided

Comparisons between variables (bytes and signed words) and immediate values are a frequent source of errors.

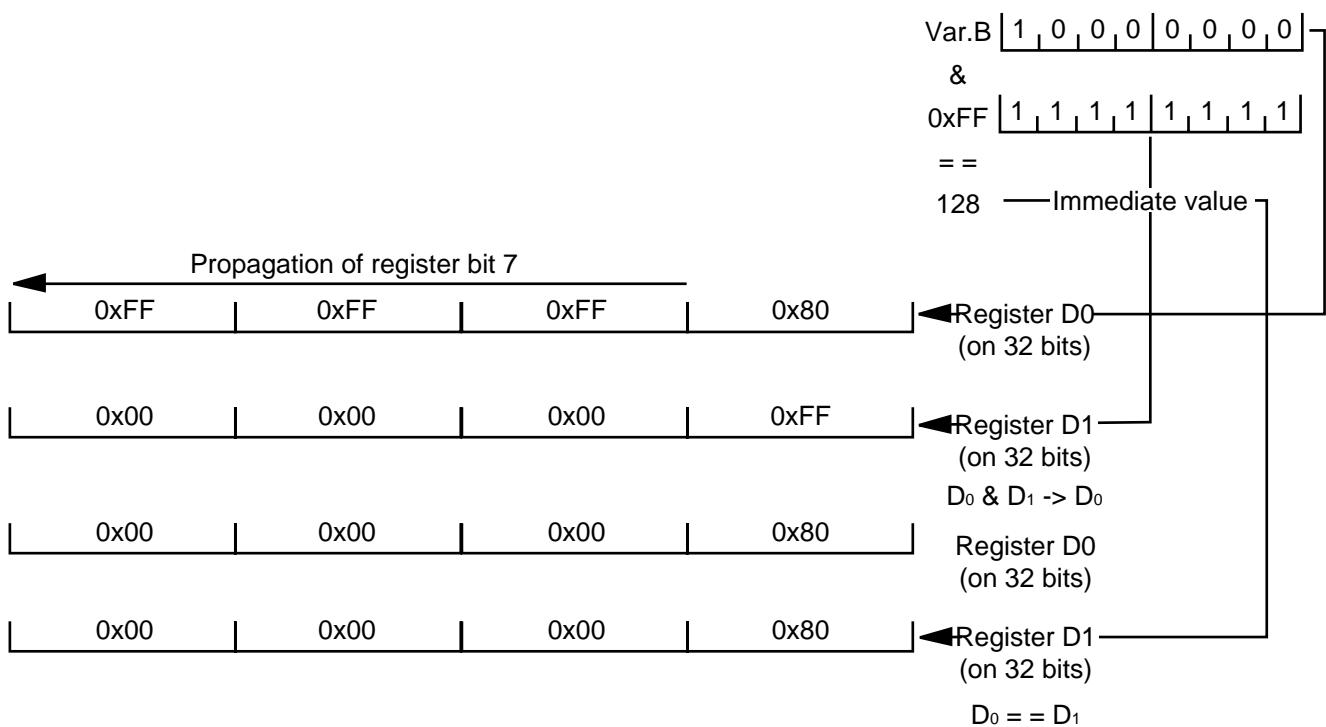
### Example

Var.B == 128.



Variable Var.B is never equal to immediate value 128.

The equality can be achieved using a mask and writing Var.B & 0xFF == 128.



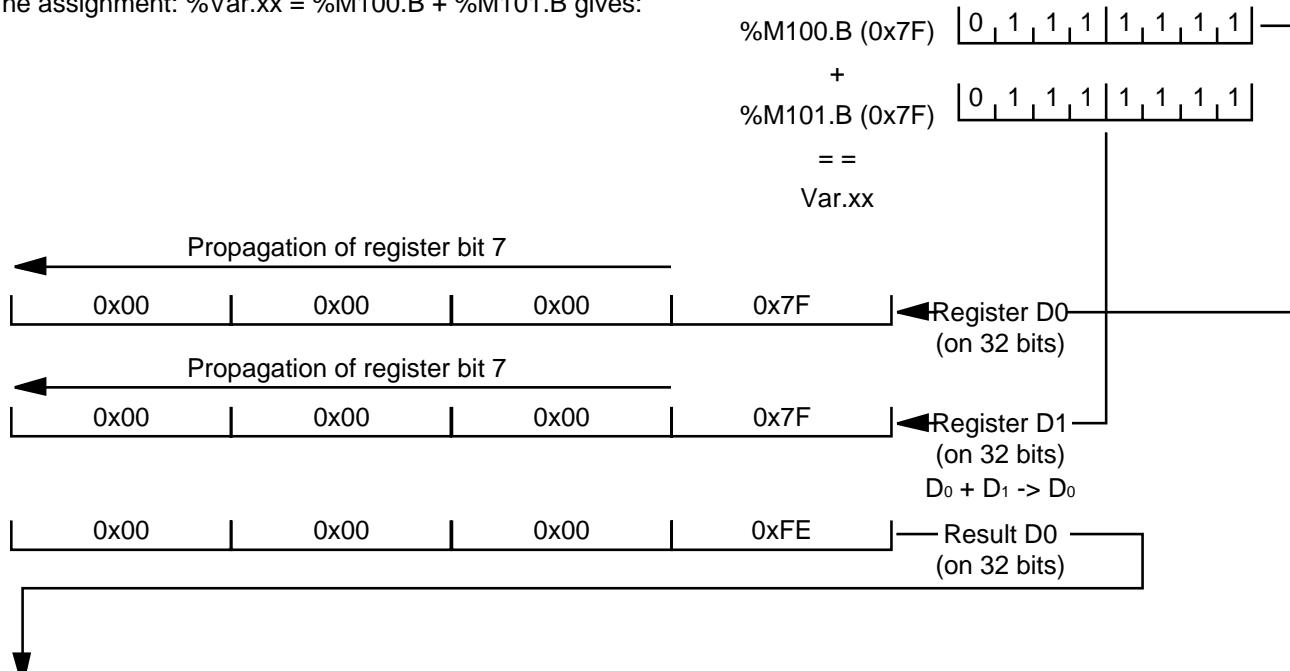
#### 4.5.8 Overflow - Sign Change

The system does not perform overflow control. It is therefore up to the programmer to take the necessary precautions.

##### Example of Sign Change:

%M100.B and %M101.B are two byte variables both equal to 0x7F (i.e. +127).

The assignment: %Var.xx = %M100.B + %M101.B gives:



Var.B is loaded with 0x00FE. Since bit 7 (sign bit) is a 1, Var.B == -2 (wrong answer) 1 1 1 1 | 1 1 1 0

Var.W is loaded with 0x00FE. Since bit 15 (sign bit) is a 0, Var.W == 254 (right answer) 0 0 0 0 | 0 0 0 0 | 1 1 1 1 | 1 1 1 0

Var.L is loaded with 0x000000FE. Since bit 31 (sign bit) is a 0, Var.L == 254 (right answer) 0x00 | 0x00 | 0x00 | 0x00 | 0xFE

#### 4.5.9 Examples of Literal Entities

##### Comparisons:

- %M5.B + %V33.L == %M10.W,
- (Var\_1 << 4 ^ 0x3) << %M10.B >= -( (Var\_6 & 0xF5) + (Var\_3 & %M5.W))
- %I900.W & 1 << %V100.B != 0 // Test of bit %V100.B of %I900.W (see Sec. 5.2.7.4).

##### Numerical Assignments:

- %M5.B = %M33.L - %M10.W,
- Var\_1 = -%M10.B + ~(0xF5 - (Var\_3 << %M5.W)),
- Remainder = Dividend - Dividend/Divisor\*Divisor//calculation of the remainder of an integer division.

**Valid Function Calls:**

- Var\_1.L = printf(STRING\_1, %M45.W << 4, %M100.L, (Var\_4 & 0x33) + %M200.W),
- printf( %M250.L, %M45.W << 4, %M100.L, (Var\_4 & 0x33) + %M200.W).

#### 4.5.10 Maximum Length of a Literal Entity

The maximum length of a literal entity is LGM\_LITTERAL (set to 120 characters).

#### 4.5.11 Maximum Number of Operands in a Numerical Expression

Independently of the maximum length of an expression, the maximum number of operands authorised in an expression is limited by another criterion: the maximum number of registers NBM\_DATA\_REG (set to 5).

An overrun is indicated during compilation by the message «Error Maximum no. of data registers».

**Example:**

The numerical expression: Var\_1 | Var\_2 ^ Var\_3 >> 8 + Var\_4 \* Var\_5 which generates the following stack is rejected by the compiler because it requires more than five registers for its evaluation.

Var_1	register 1
Var_2	register 2
Var_3	register 3
8	register 4
Var_4	register 5
Var_5	register 6    *error, more than 5 registers
*	register 5
+	register 4
>>	register 3
^	register 2
	register 1

The above expression can be evaluated by reorganising it. The equivalent expression Var\_3 >> Var\_5 \* Var\_4 + 8 ^ Var\_2 | Var\_1 which generates the following stack is accepted by the compiler.

Var_3	register 1
Var_5	register 2
Var_4	register 3
*	register 2
8	register 3
+	register 2
>>	register 1
Var_2	register 2
^	register 1
Var_1	register 2
	register 1

# 5 Programming in Ladder Language

<b>5.1 Elements Common to All Types of Sequence</b>	5 - 3
5.1.1 Sequence Header	5 - 3
5.1.2 Grafcet Step	5 - 3
5.1.2.1 General	5 - 3
5.1.2.2 How the System Processes Grafcet Steps	5 - 3
5.1.2.3 Activation/Deactivation of Grafcet Steps	5 - 4
5.1.2.4 Examples of Programming	5 - 4
<b>5.2 Network Sequence</b>	5 - 7
5.2.1 General	5 - 7
5.2.2 Structure of the Test Zone	5 - 7
5.2.2.1 General	5 - 7
5.2.2.2 Contacts	5 - 7
5.2.2.3 Conditional Actions in the Test Zone	5 - 9
5.2.2.4 Timeouts	5 - 10
5.2.2.5 Up/Downcounters	5 - 12
5.2.2.6 Branches	5 - 14
5.2.2.7 Execution of a Test Zone	5 - 14
5.2.3 Structure of the Action Zone	5 - 15
5.2.3.1 General	5 - 15
5.2.4 Execution of an Action Zone	5 - 16
5.2.5 Rules for Constructing a Ladder Network	5 - 18
5.2.6 Example of Ladder Network Sequences	5 - 18
5.2.7 Programming Hints	5 - 21
5.2.7.1 Optimising Networks	5 - 21
5.2.7.2 Bit List in the Test Zone	5 - 22
5.2.7.3 Multiple Numerical Assignments	5 - 24
5.2.7.4 Testing the Bits of a Byte, Word or Long Word	5 - 25
<b>5.3 Function Calls</b>	5 - 26
<b>5.4 Parameter Check</b>	5 - 26



## 5.1 Elements Common to All Types of Sequence

### 5.1.1 Sequence Header

Sequences such as tables of constants, character strings and networks have a common header including:

- an optional sequence identifier called a label (see Sec. 4.2),
- an optional comment (see Sec. 4.2),
- an optional grafset step (see Sec. 4.3).

### 5.1.2 Grafset Step

Grafset steps increase the speed of execution of a programme because inactive sequences are not executed. Grafset steps are used to specify the software using a grafset methodology.

If it is not possible to programme all the actions of a grafset step in the same sequence, the programmer can write as many sequences as he desires with the same step number.

#### 5.1.2.1 General

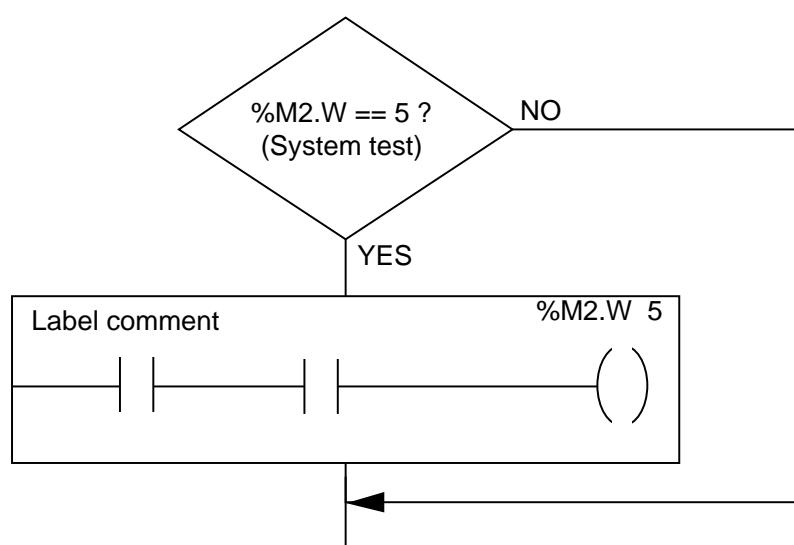
A sequence with a grafset step has two possible states:

- active when  $\langle \text{step\_variable} \rangle == \langle \text{step\_number} \rangle$ ,
- inactive when  $\langle \text{step\_variable} \rangle != \langle \text{step\_number} \rangle$ .

#### 5.1.2.2 How the System Processes Grafset Steps

When a sequence with a grafset step is active, the system executes it like a sequence without a step.

When a sequence with a grafset step is inactive, it is not executed.

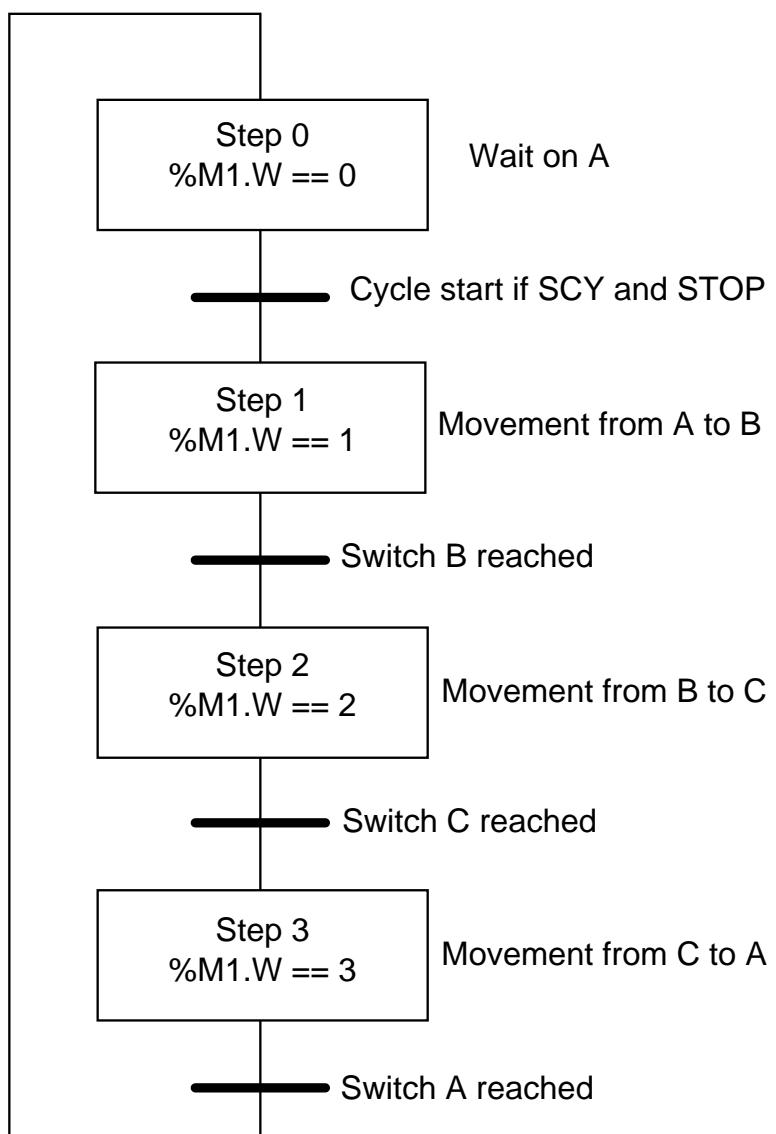


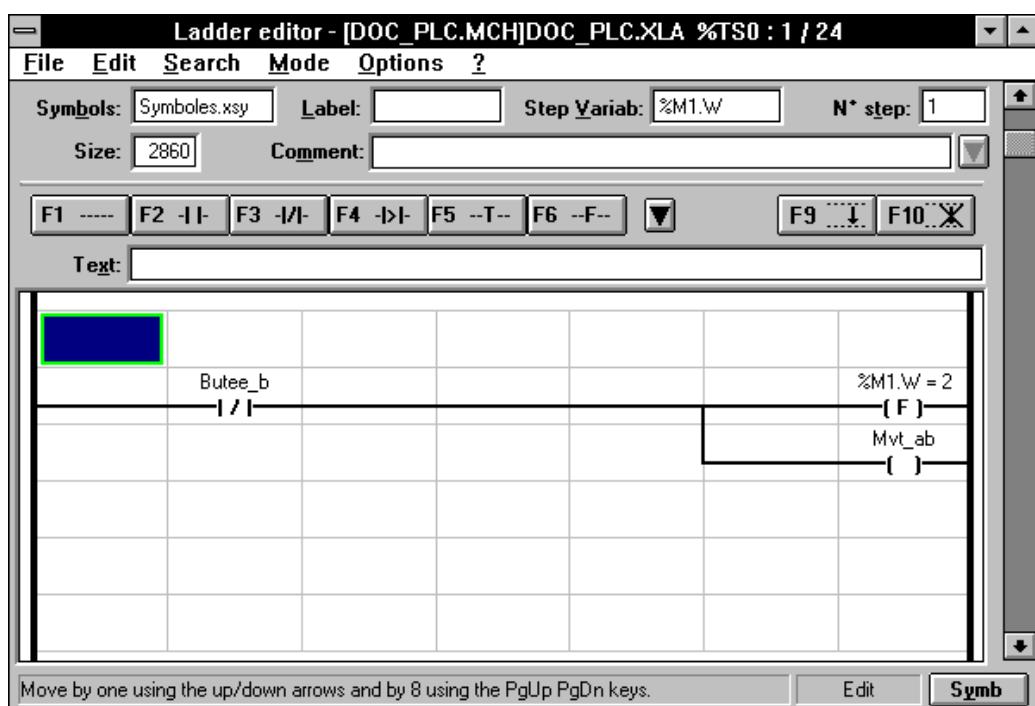
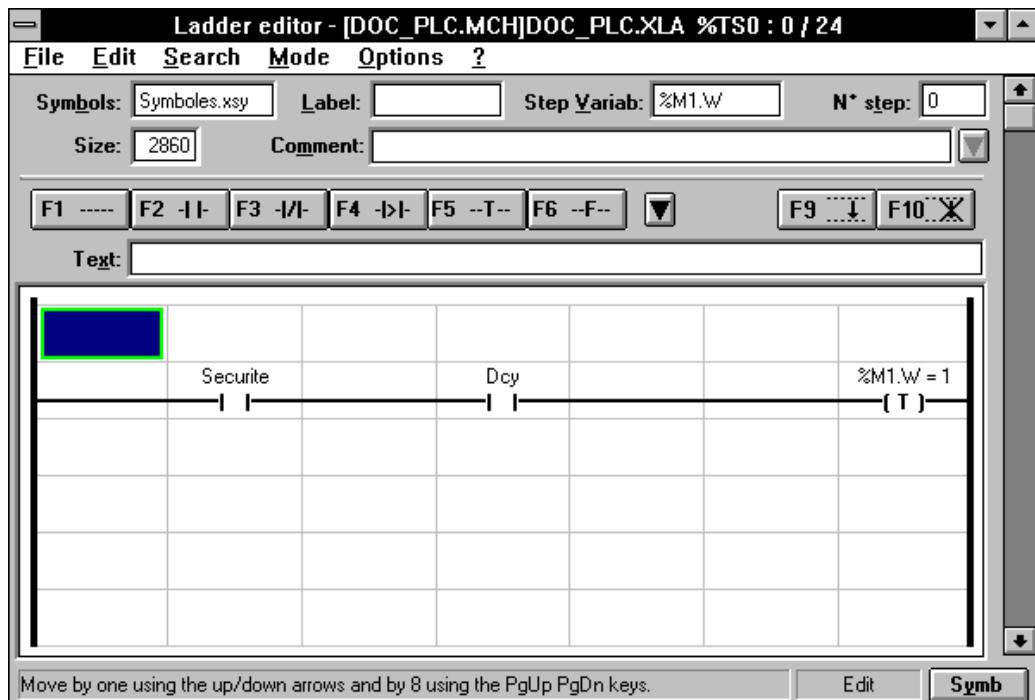
### 5.1.2.3 Activation/Deactivation of Grafset Steps

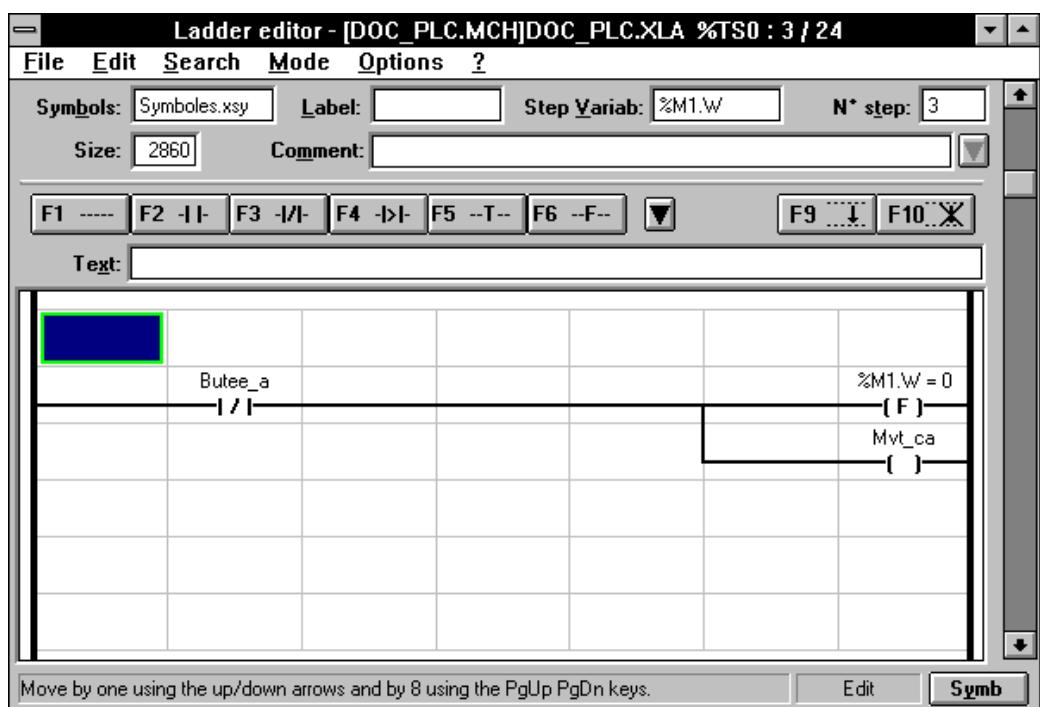
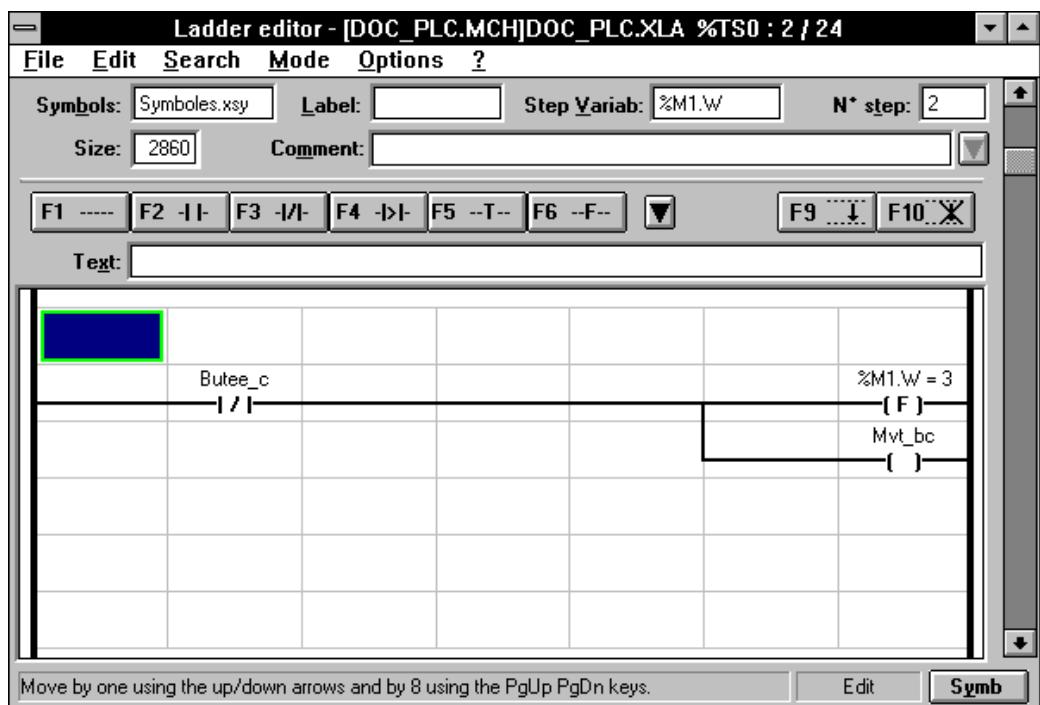
Sequences with grafset steps are activated (deactivated) from the user programme by loading the <step\_variable> with the integer corresponding to the sequence(s) to be activated.

### 5.1.2.4 Examples of Programming

Specification using the grafset methodology







## 5.2 Network Sequence

### 5.2.1 General

A ladder network includes:

- an optional label and an optional comment,
- an optional grafct step,
- a test zone,
- an action zone.

### 5.2.2 Structure of the Test Zone

#### 5.2.2.1 General

The test zone occupies the left-hand side of the network.

It is used to enter logical equations.

A logical equation is a combination of contacts in parallel or in series.

A contact is a Boolean resulting from:

- the test of one or more bit variables,
- the test of the rising or falling edge of an input rung,
- the comparison between two numerical expressions.

The test zone includes six rungs on which six contacts can be connected.

The state of the rung on the contact output depends on the state of the rung on the contact input and the result of the test.

If the result is FALSE, the corresponding rung is set to ZERO. Otherwise the rung state is not changed.

Conditional actions are allowed in the test zone. These actions are determined by the state of the input rung and do not modify the output rung.

It is possible to branch rungs.

A branch is symbolised by a vertical bar.

#### 5.2.2.2 Contacts

There are five types of contacts:

Contact type	Description
<code>&lt;bit variable&gt;{&lt;bit variable&gt;}7 —] [—</code>	Test for a ONE state of a list of bit variables. IF all the bits are ONE, input rung = output rung. ELSE, the output rung is reset.
<code>&lt;bit variable&gt;{&lt;bit variable&gt;}7 —] / [—</code>	Test for ZERO state of a list of bit variables. IF all the bits are ZERO, output rung = input rung. ELSE, the output rung is reset.
<code>&lt;bit variable&gt; — R_T —</code>	Detects the rising edge of the input rung (RISING TRIG). Used to store the state of the input rung. IF the input rung is a ONE and <bit variable> is a ZERO, the output rung is set to ONE. ELSE, the output rung is set to ZERO. <bit variable> = input rung (storage of the input rung).

Contact type	Description
<bit variable> — F_T —	Detects the falling edge of the input rung (FALLING TRIG). Used to store the state of the input rung. IF the input rung is a ZERO and <bit variable> is a ONE, the output rung is set to ONE. ELSE, the output rung is set to ZERO. <bit variable> = input rung (storage of the input rung).
<numerical_comparison> —[ > ]—	Used to compare two numerical expressions. IF the numerical comparison is TRUE, the output rung is unchanged. ELSE, the output rung is set to ZERO.

### Operation of the R\_T Cell (Rising trig)



If (Input == 1 and Stored value == 0) then Output = 1

Else Output = 0

Stored value = Input

### Operation of the F\_T Cell (Falling trig)

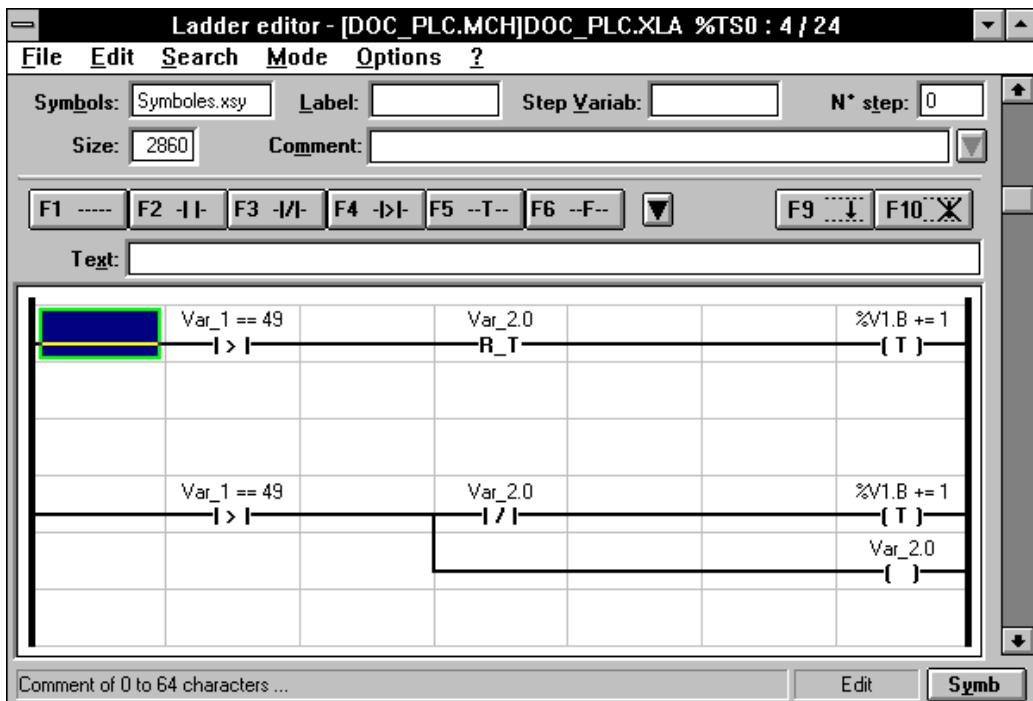


If (Input == 0 and Stored value == 1) then Output = 1

Else Output = 0

Stored value = Input

## Example

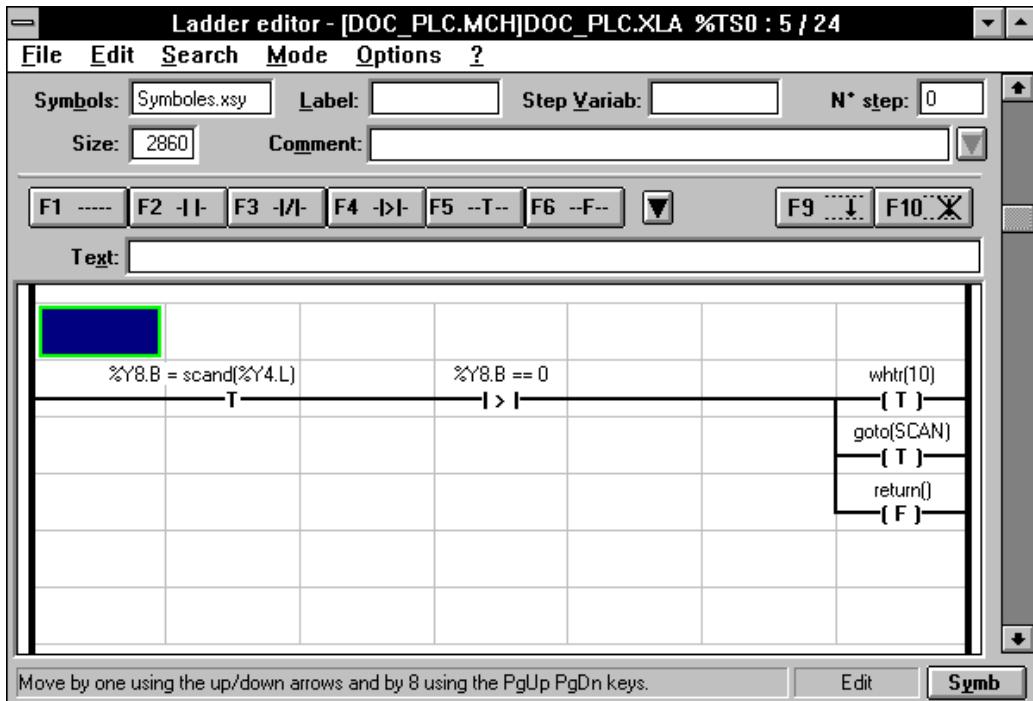


### 5.2.2.3 Conditional Actions in the Test Zone

There are two types of actions:

Contact type	Description
<numerical_assignment> {;<numerical_assignment>}7 <function_call> — T —	Action executed if the input rung is a ONE. The possible actions are: - <numerical assignment>, e.g. %M10.B = %V34+3, - <function_call>, e.g. setb(%M100.&, 0, 100).
<numerical_assignment> {;<numerical_assignment>}7 <function_call> — F —	Action executed if the input rung is a ZERO. The possible actions are: - <numerical assignment>, e.g. %M10.B = %V34+3, - <function_call>, e.g. setb(%M100.&, 0, 100).

## Example



### 5.2.2.4 Timeouts

Three types of timeout function blocks are available:

- Off timeouts TOF\_n
- On timeouts TON\_n
- Pulse timeouts TP\_n.

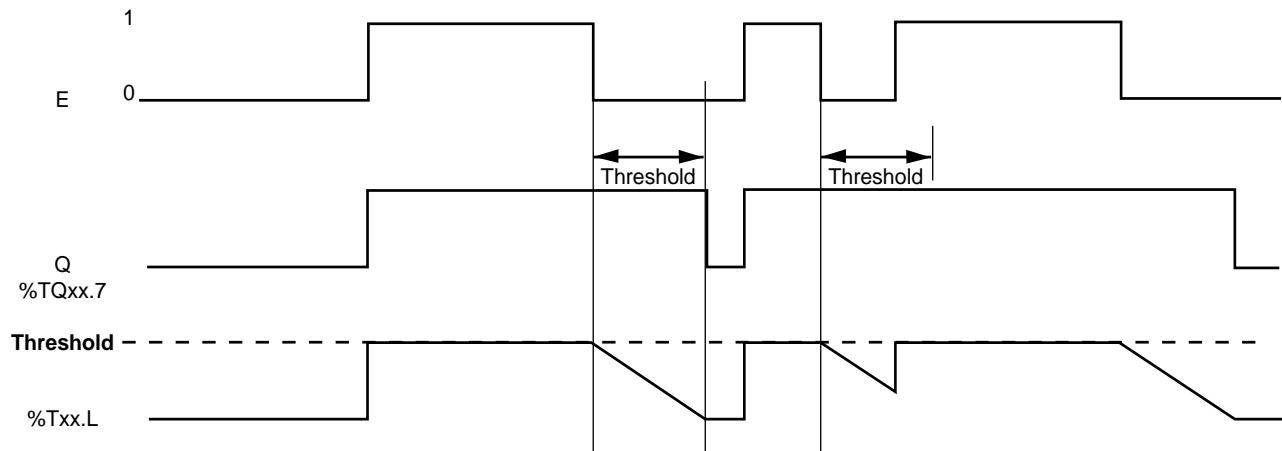
There are a total of 128 available timeouts (all types together).

Variables %T0.L to %T7F.L contain the current timeout in ms. Only size .L is allowed for read and write by UNITE.

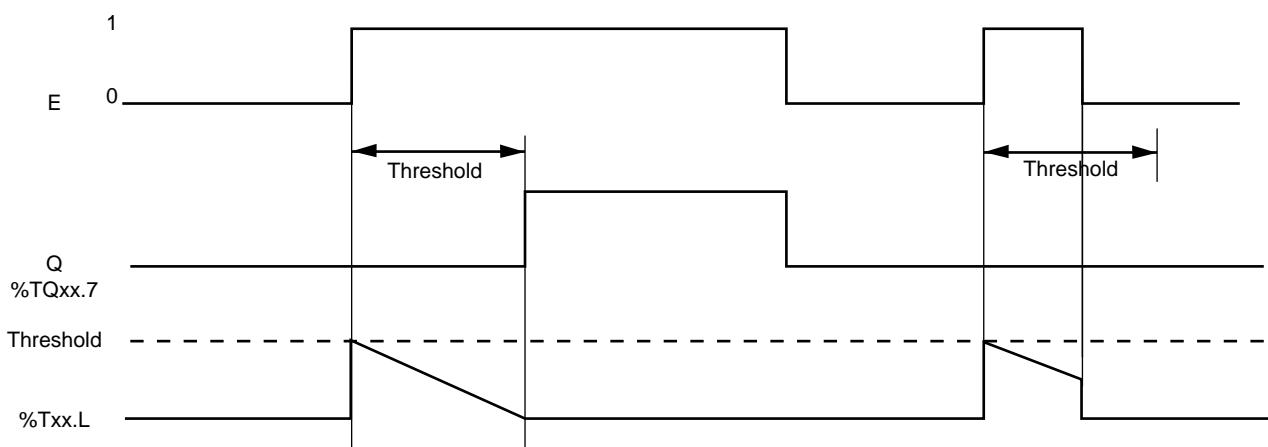
Variables %TQ0.7 to %TQ7F.7 are the image of the timeout output (Q). Only bit .7 is allowed for read and write by UNITE.

Timeout type	Description
TOF_n(<threshold>)	Off timeout (with n = 00 to 7F) Setting of E sets output Q for an indeterminate time. Reset of E resets output Q at the end of the timeout. The threshold argument is a numerical expression in ms.
TON_n(<threshold>)	On timeout (with n = 00 to 7F) Setting of E sets output Q at the end of the timeout. Q is reset as soon as E is reset. The threshold argument is a numerical expression in ms.
TP_n(<threshold>)	Pulse timeout (with n = 00 to 7F) Setting of E sets output Q for the duration of the timeout. Q is reset at the end of the timeout. The threshold argument is a numerical expression in ms.

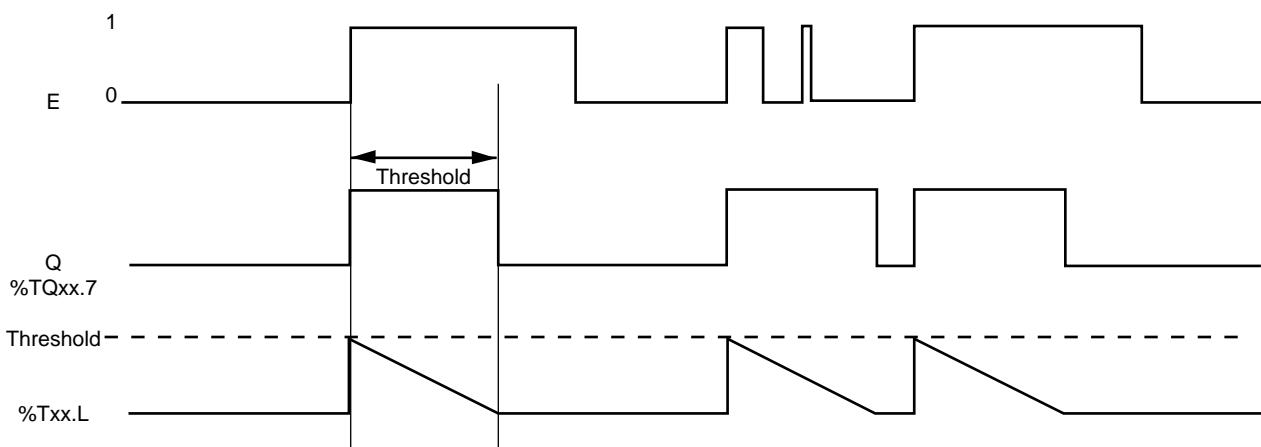
### Off Timeout «TOF\_n»



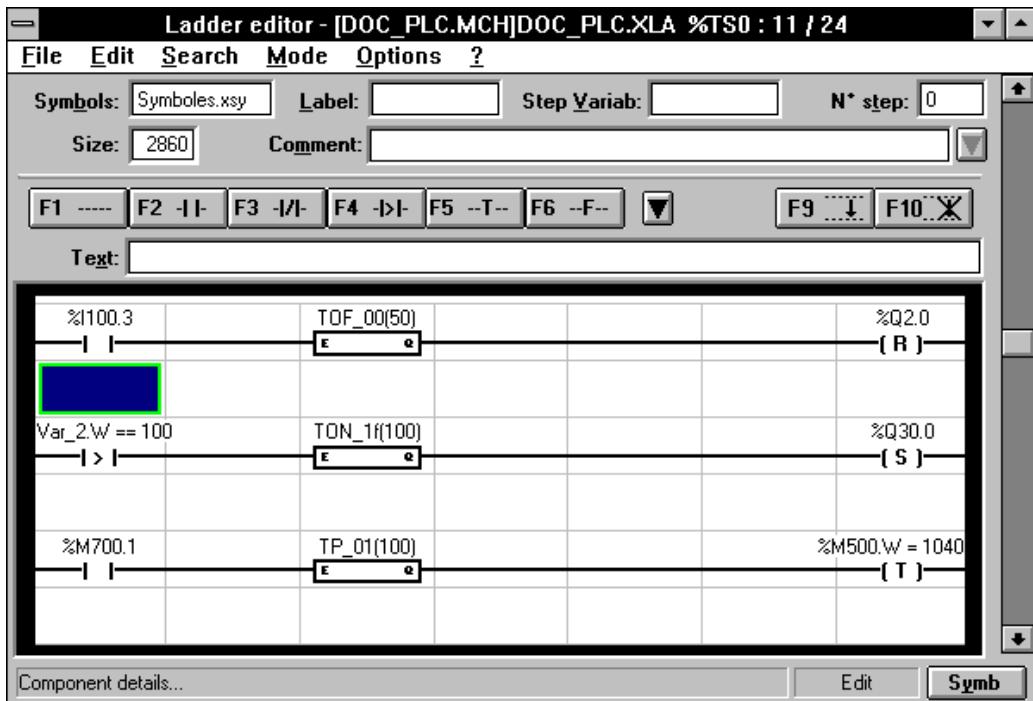
### On Timeout «TON\_n»



### Pulse Timeout «TP\_n»



## Example



### 5.2.2.5 Up/Downcounters

Two types of up/downcounter function blocks are available:

- upcounters CTU\_n
- downcounters CTD\_n.

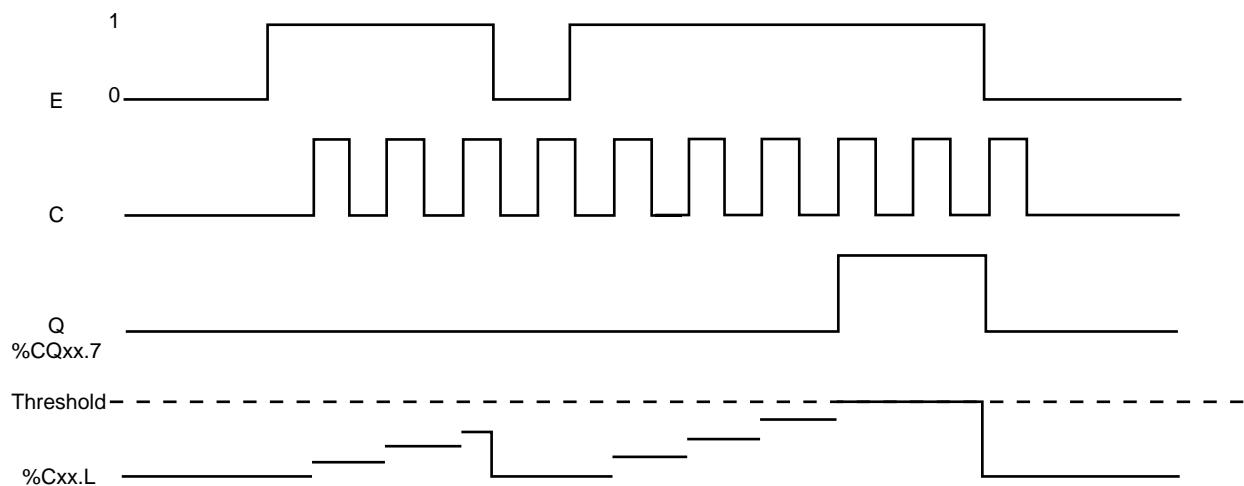
There are 128 available up/downcounters.

Variables %C0.L to %C7F.L contain the current up/down counter value. Only size .L is allowed for read and write by UNITE.

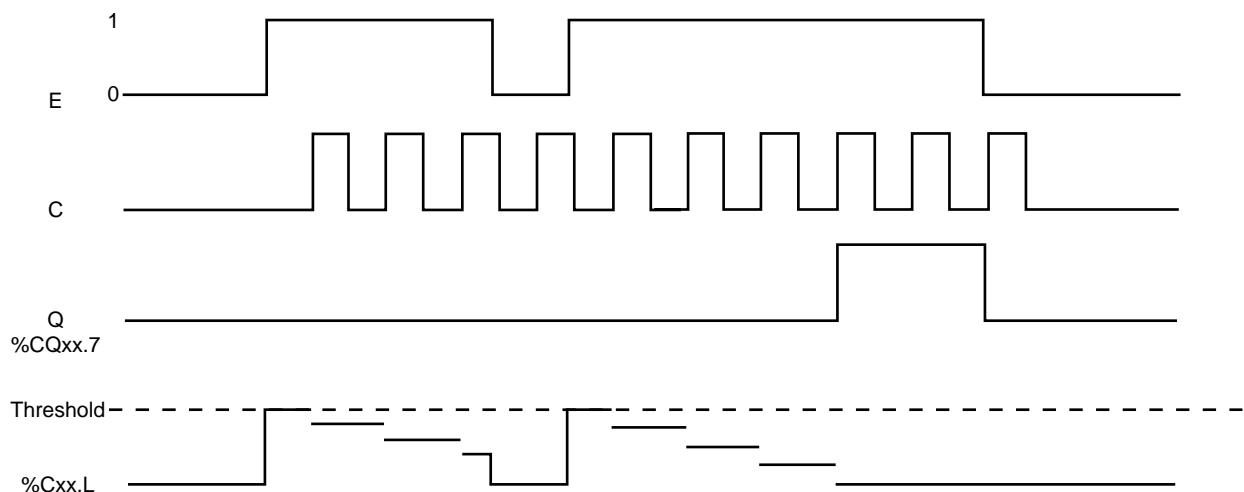
Variables %CQ0.7 to %CQ7F.7 are the image of the up/down counter (Q). Only bit .7 is allowed for read and write by UNITE.

**REMARK** *The up/down counters are reset only by a reset of the saved variables.*

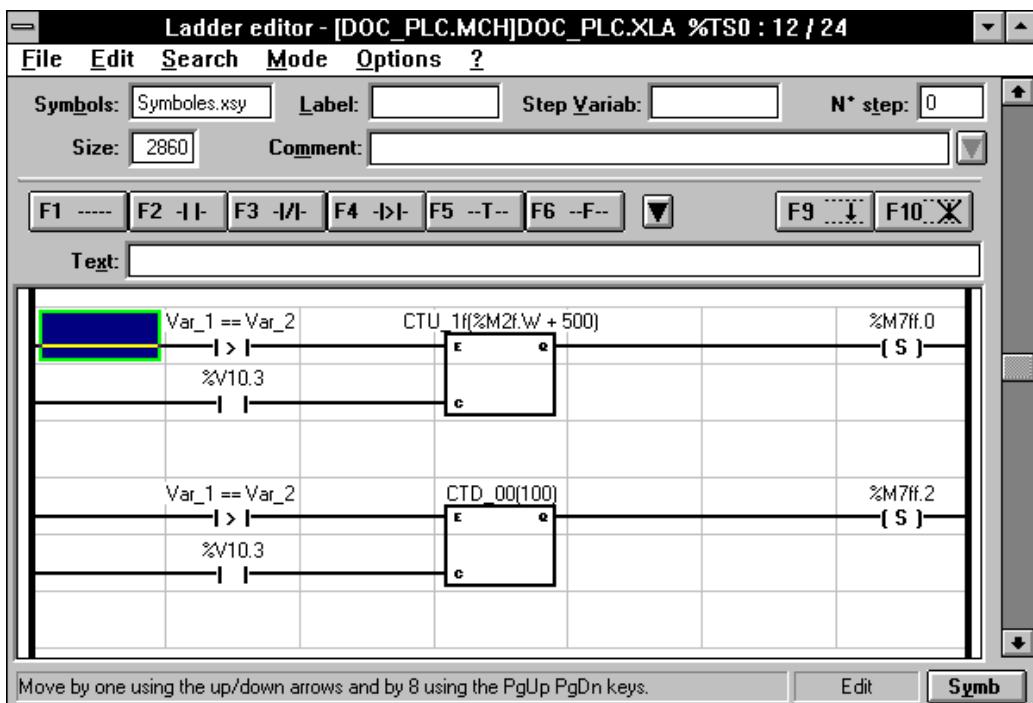
Timeout type	Description
 CTU_n(<threshold>)	Upcounter (n = 00 to 7F) Setting of E sets output Q as soon as the threshold is reached. Reset of E resets output Q. C defines the elements to be upcounted. The threshold argument is a numerical expression.
 CTU_n(<threshold>)	Downcounter (n = 00 to 7F) Setting of E sets output Q as soon as the threshold is reached. Reset of E resets output Q. C defines the elements to be downcounted. The threshold argument is a numerical expression.

**Upcounters**

5

**Downcounters**

## Example



### 5.2.2.6 Branches

It is possible to branch adjacent rungs.

A branch is shown as a vertical bar after a contact or rung.

### 5.2.2.7 Execution of a Test Zone

The test zone is scanned from top to bottom and from left to right.

On a rung, the potential propagates from left to right and never from right to left (contrary to a circuit diagram where propagation can be in both directions).

On a branch, propagation is from bottom to top and top to bottom.

## 5.2.3 Structure of the Action Zone

### 5.2.3.1 General

The action zone is located to the right of the ladder.

It allows actions to be initiated conditionally according to the logical results of the test zone.

Six actions conditioned by the six rungs of the test zone can be initiated in one sequence.

There are eight possible types of actions.

## ACTIONS

The six rungs can initiate six actions of the following types:

Contact type	Description
<bit_variable> —( )—	Set the bit to the same logic state as the rung.
<bit_variable> —( / )—	Set the bit to the inverse logic state of the rung.
<bit_variable> —( S )—	If the rung is TRUE, set the bit. Else go to the next action.
<bit_variable> —( R )—	If the rung is TRUE, reset the bit. Else go to the next action.
<numerical_assignment>{:<numerical_assignment>}7 <function_call>goto(<label>)      If the rung is TRUE, execute: call(<label>)return(...).      - a numerical assignment      e.g. %M10.B = %V34+3 —( T )—      - a function call      e.g. setb (%M100.&, 0, 100) - a jump to an internal label in the module      e.g. goto (END) - call to an internal label in the module      e.g. call (COPY) - a return to the calling module or call      e.g. return (%M10.B) Else go to the next action.	
<numerical_assignment>{:<numerical_assignment>}7 <function_call>goto(<label>)      If the rung is FALSE, then execute: call(<label>)return(...).      - a numerical assignment      e.g. %M10.B = %V34+3 —( F )—      - a function call      e.g. setb (%M100.&, 0, 100) - a jump to an internal label in the module      e.g. goto (END) - call to an internal label in the module      e.g. call (COPY) - a return to the calling module or call      e.g. return (%M10.B) Else go to the next action.	

**REMARK** Subroutines external to the module (e.g. %SP30) are called by function sp(....).

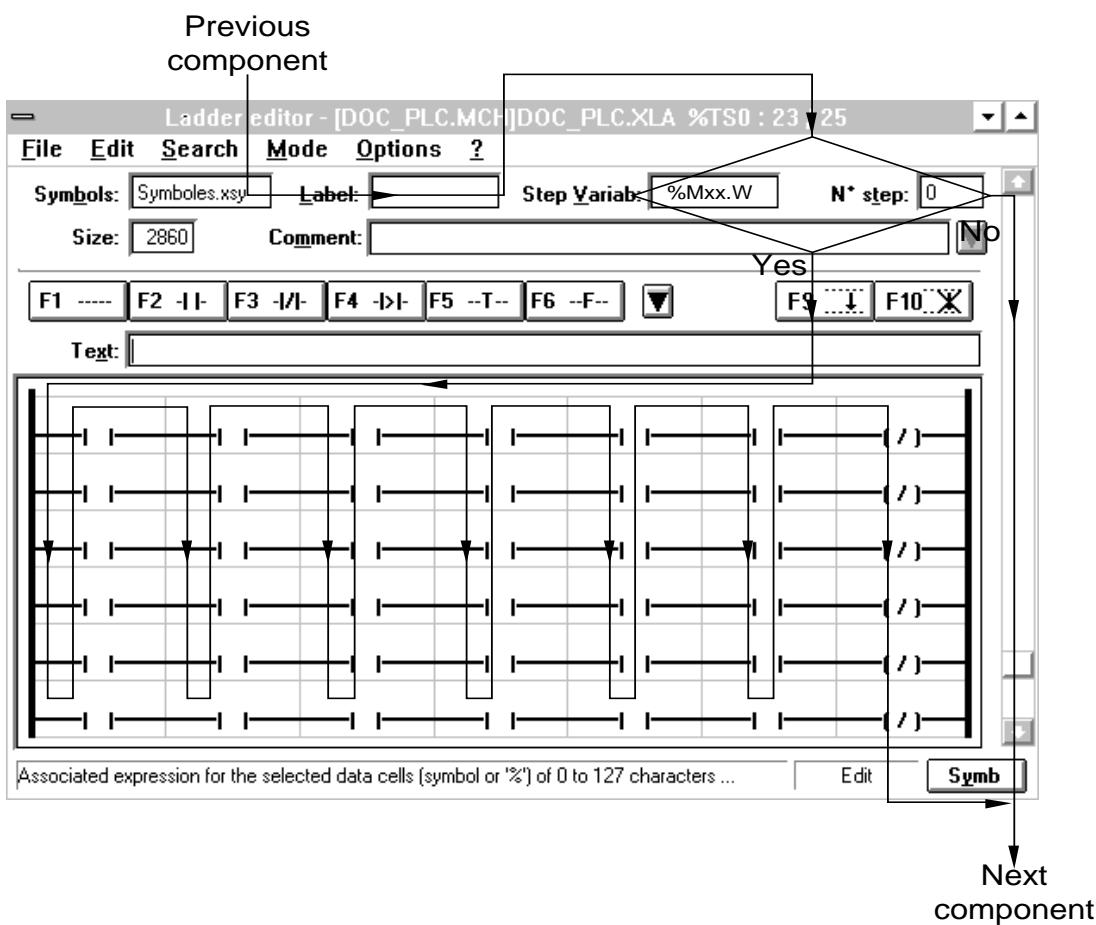
### 5.2.4 Execution of an Action Zone

The action zone is executed after the test zone from top to bottom (rung 0 to rung 5).

#### ! CAUTION

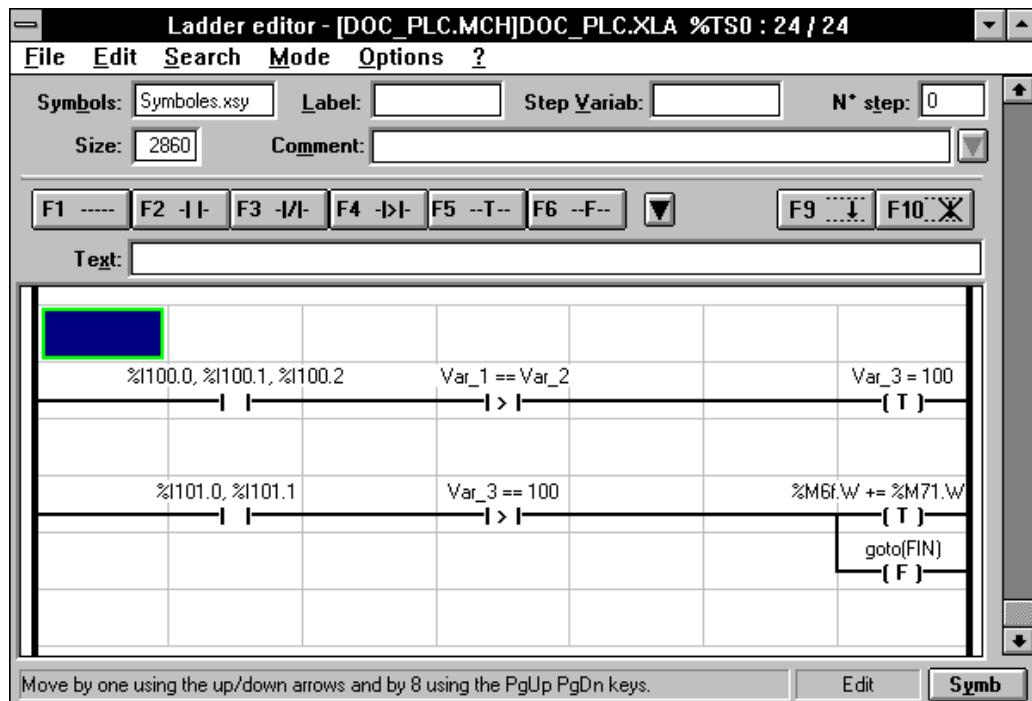
An action is always executed after complete analysis of the test zone. The change of state of a variable in an action zone is not seen until the next sequence.

#### Ladder Network Scanning Order



### Pitfall Related to Scanning

In the example below, the system reads numerical comparison «Var\_3 == 100» before writing «Var\_3 = 100» in the action zone if the conditions of the first rung are satisfied. Write of «Var\_3 = 100» and execution of the second rung are offset by one PLC cycle.



It is therefore important to make sure the scanning order does not affect execution of a programme whose instructions must be executed during the same PLC cycle.

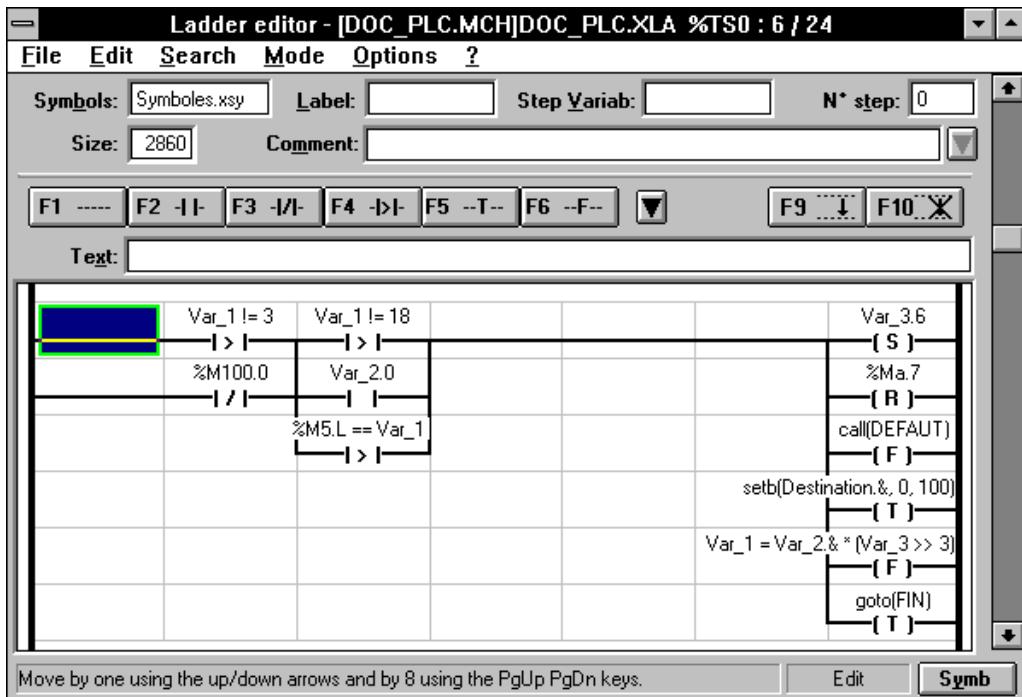
## 5.2.5 Rules for Constructing a Ladder Network

To be valid, a ladder network must comply with the following rules:

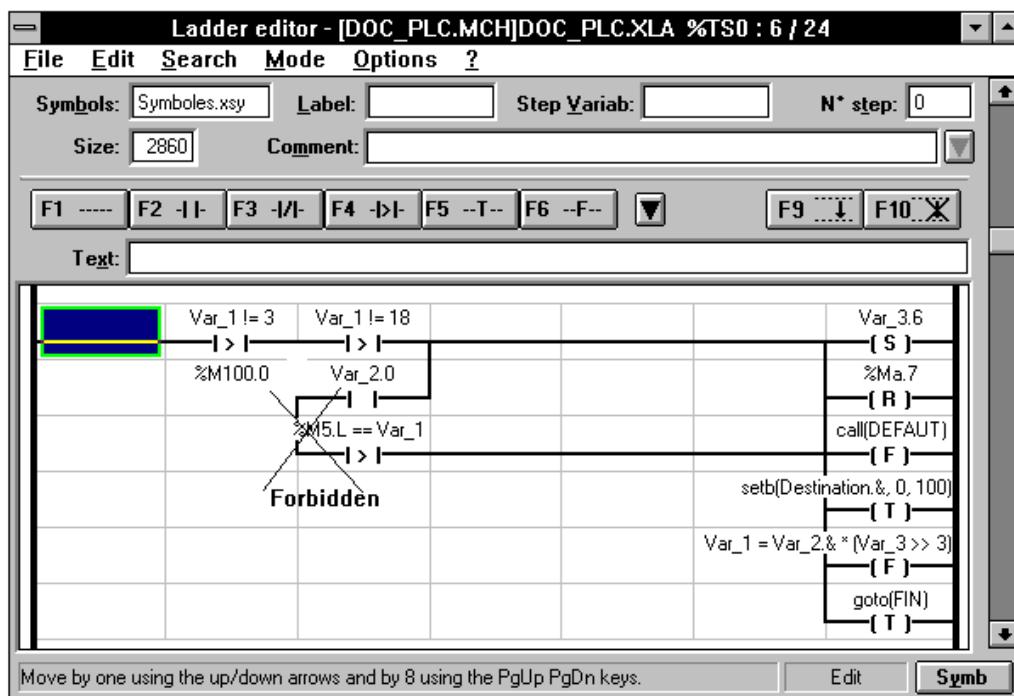
- the test zone of a ladder network must not be empty,
- a contact or a rung must be connected on the left side and right side by a contact, a rung or a branch
- a branch or a set of adjacent branches must be connected at the top and bottom to at least one contact or rung.  
In addition, it must be connected to at least one power supply, i.e. a contact or rung from the left and at least one output, i.e a contact or rung to the right,
- the action zone of a ladder network must not be empty,
- a coil must be connected on the left by a contact, a rung or a branch.

## 5.2.6 Example of Ladder Network Sequences

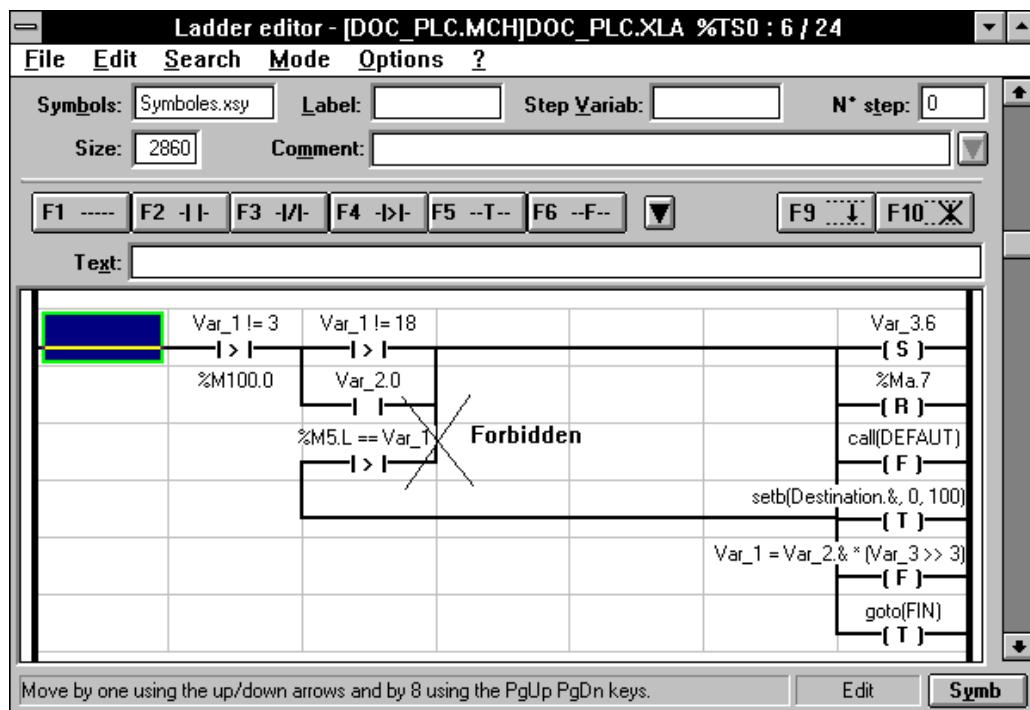
### Valid Network



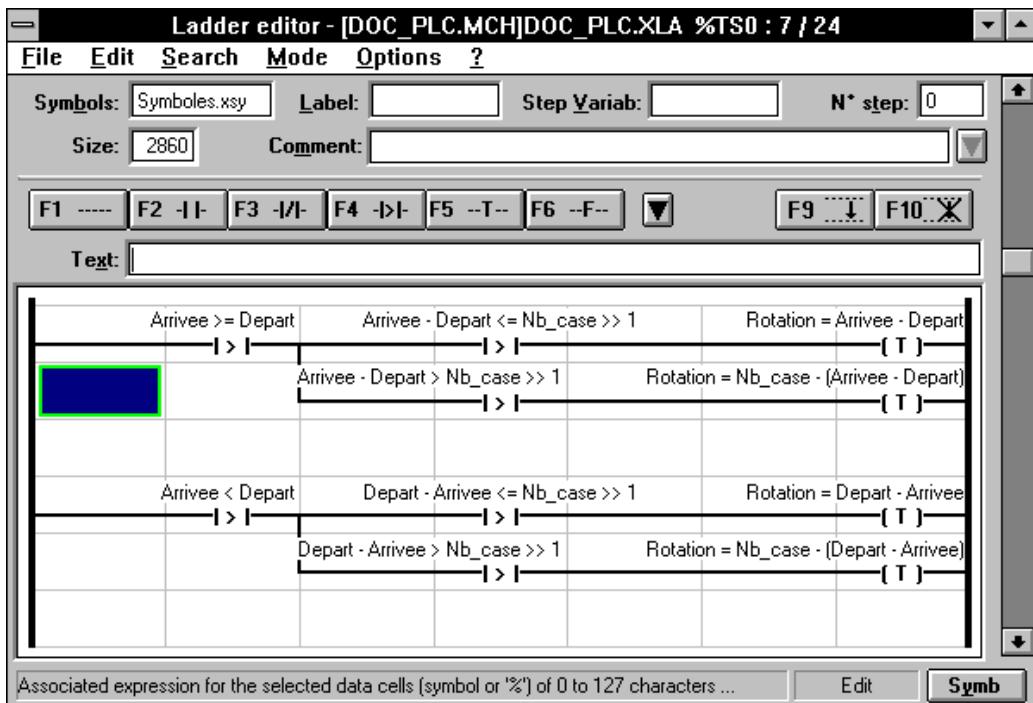
## Invalid Network - Branch without Power Supply



## Invalid network - Branch without Output



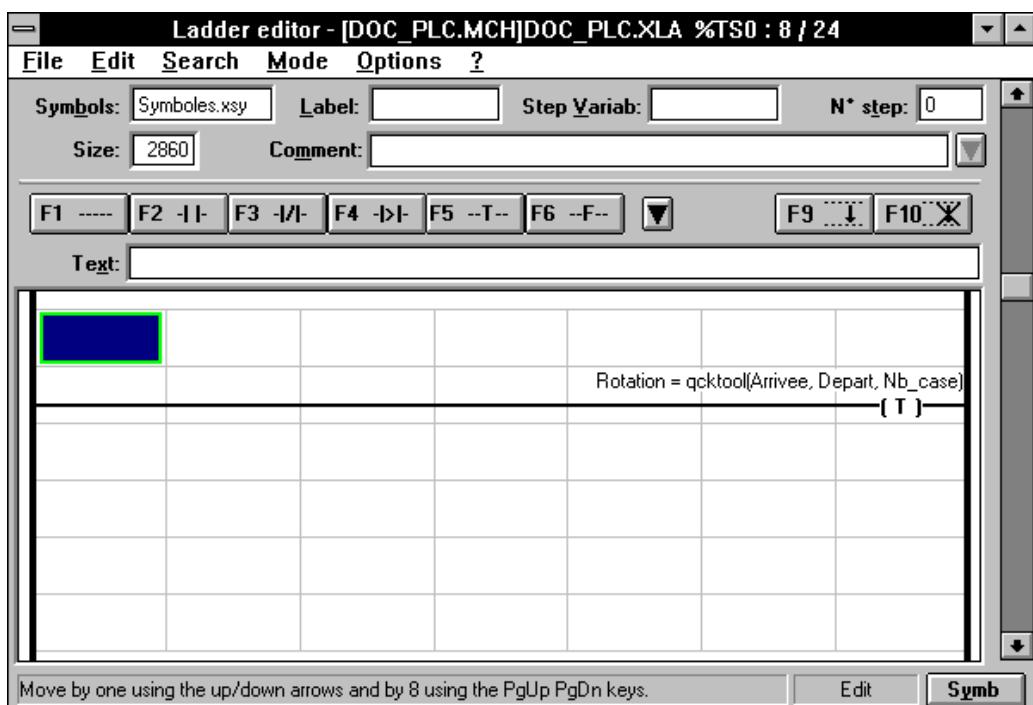
## Search for Tools in a Store



The above sequence determines the direction of rotation and number of steps required to get the tool in the «Finish» (finish) compartment from the «Start» compartment in a tool carousel with a number of compartments equal to «Comp\_num».

The absolute value of «Rotation» indicates the number of steps of the rotation and the sign of «Rotation» indicates the direction.

The following sequence uses the qcktool() function to solve this problem.



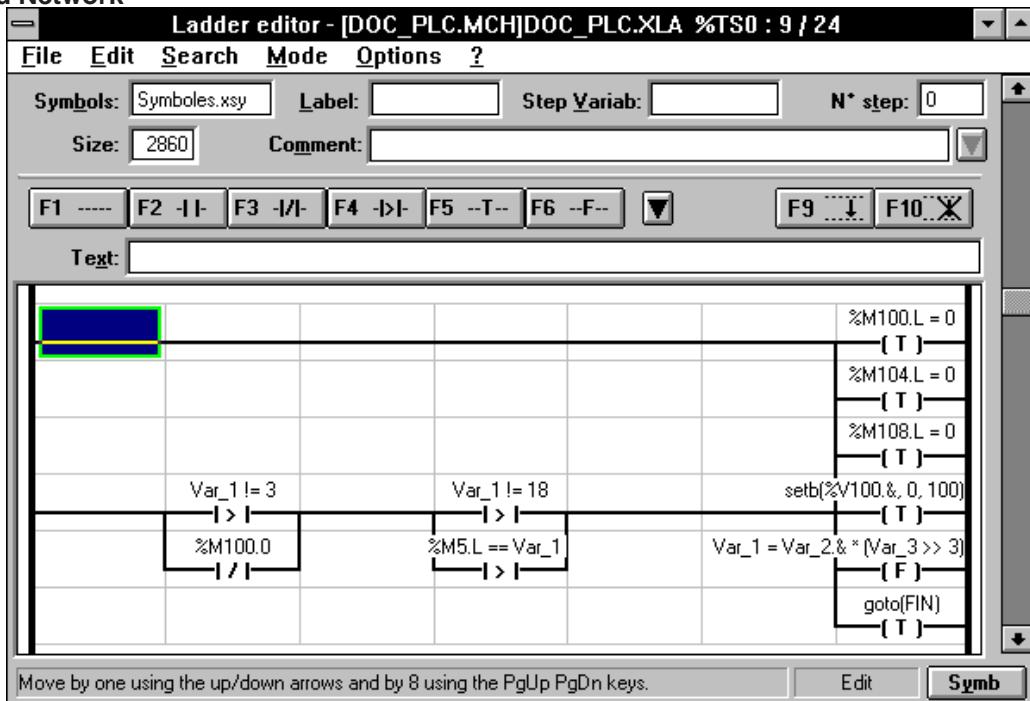
## 5.2.7 Programming Hints

### 5.2.7.1 Optimising Networks

To optimise the ladder network for code size and speed, it is necessary to minimise:

- the number of contacts,
- the number of branches (vertical bars).

#### Unoptimised Network

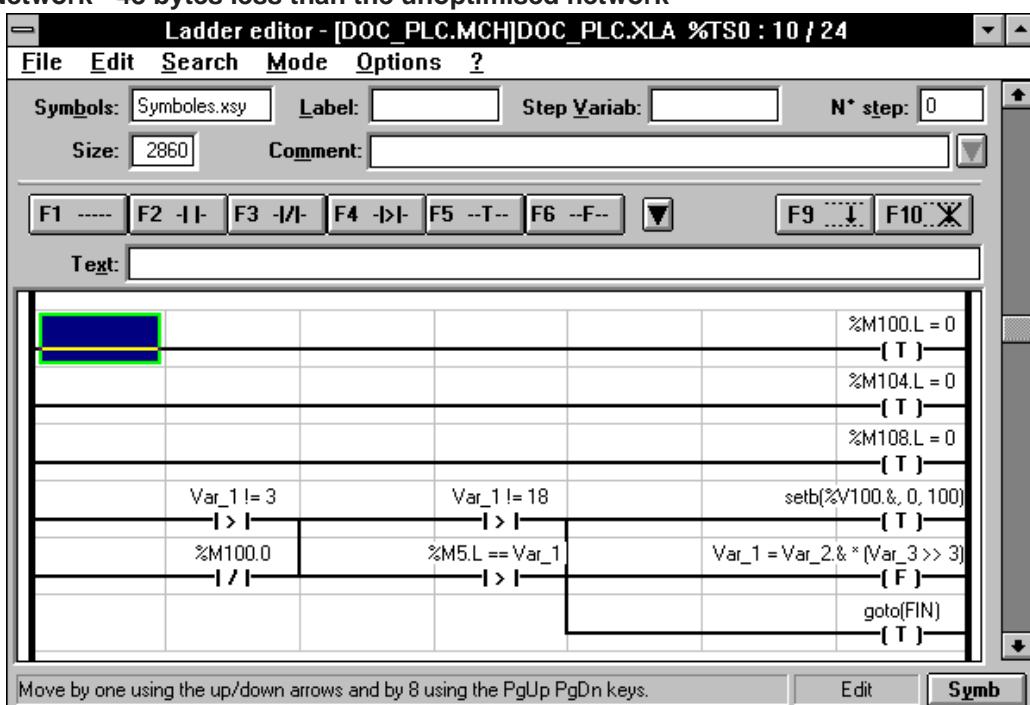


Move by one using the up/down arrows and by 8 using the PgUp PgDn keys.

Edit

Symb

#### Optimised Network - 48 bytes less than the unoptimised network



Edit

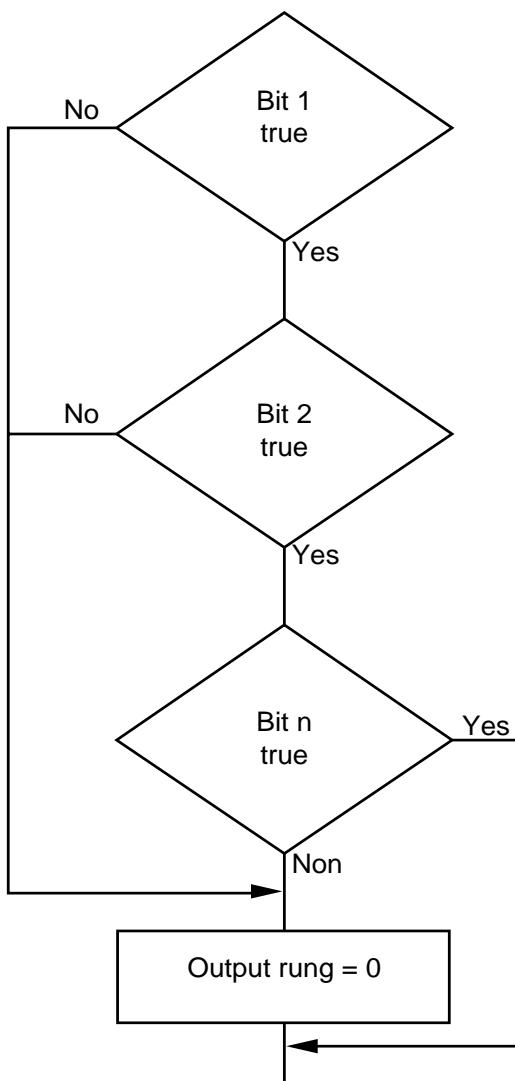
Symb

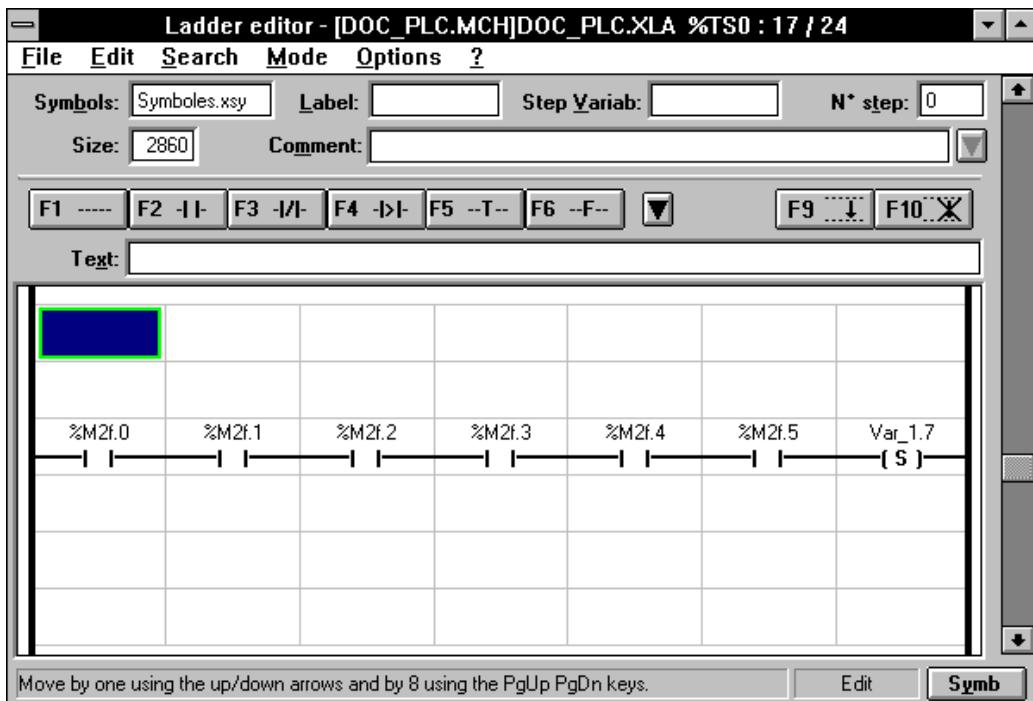
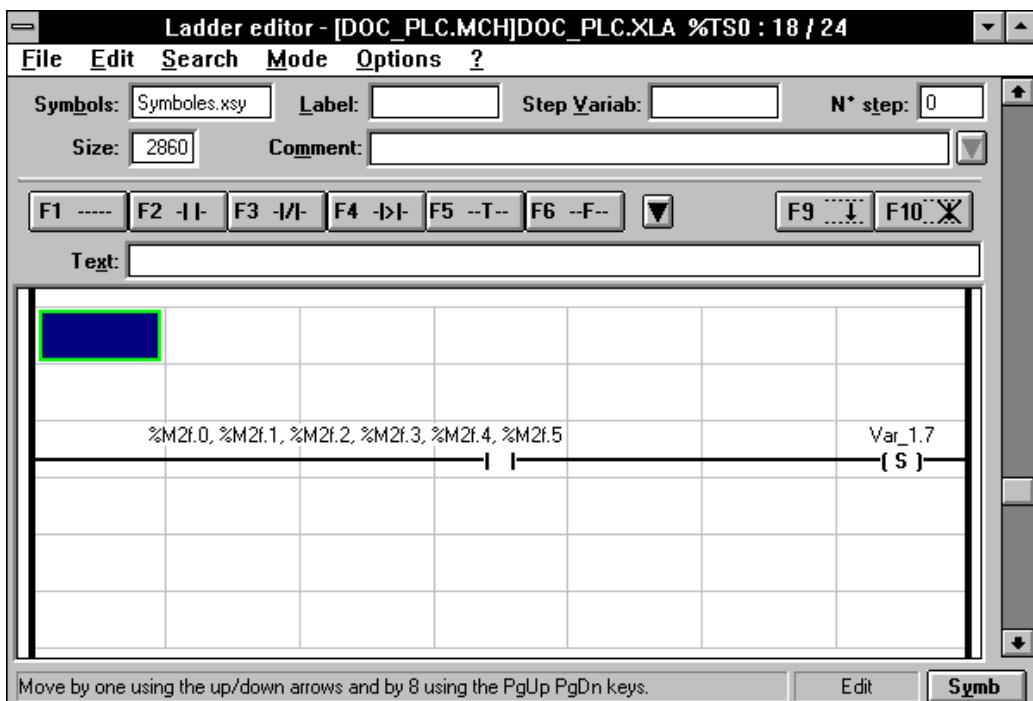
Move by one using the up/down arrows and by 8 using the PgUp PgDn keys.

### 5.2.7.2 Bit List in the Test Zone

Bit lists are used to optimise the size and speed of networks.

The flowchart below shows the principle of bit list processing by the system. Whenever a bit is false, the system skips the next bit tests.

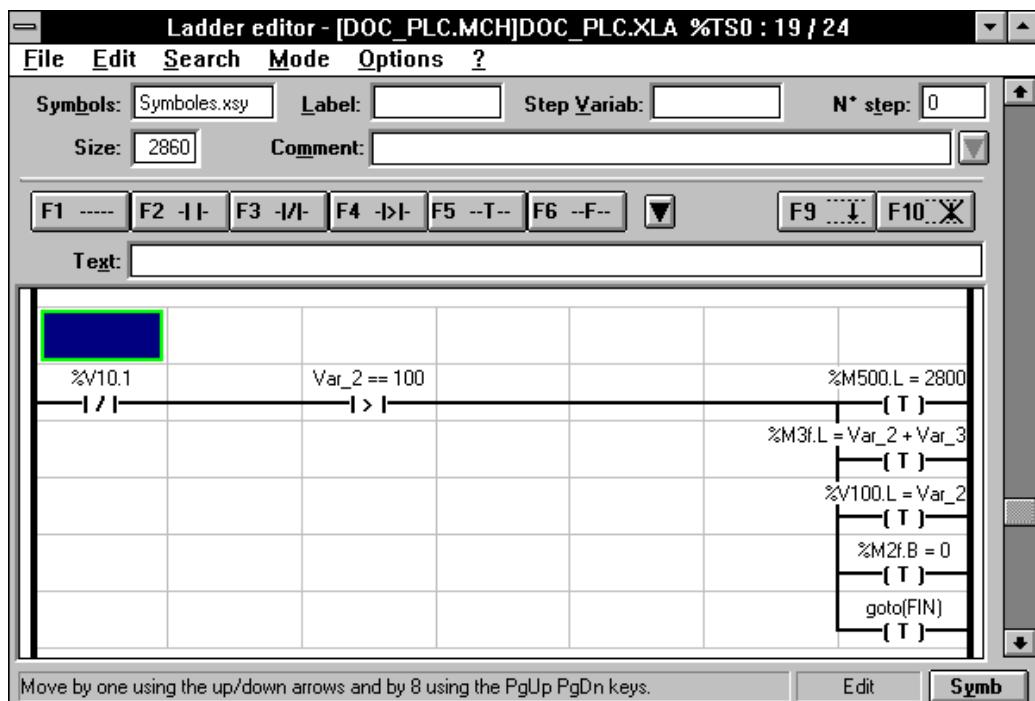


**Unoptimised Network****Optimised Network - 20 bytes less than the unoptimised network**

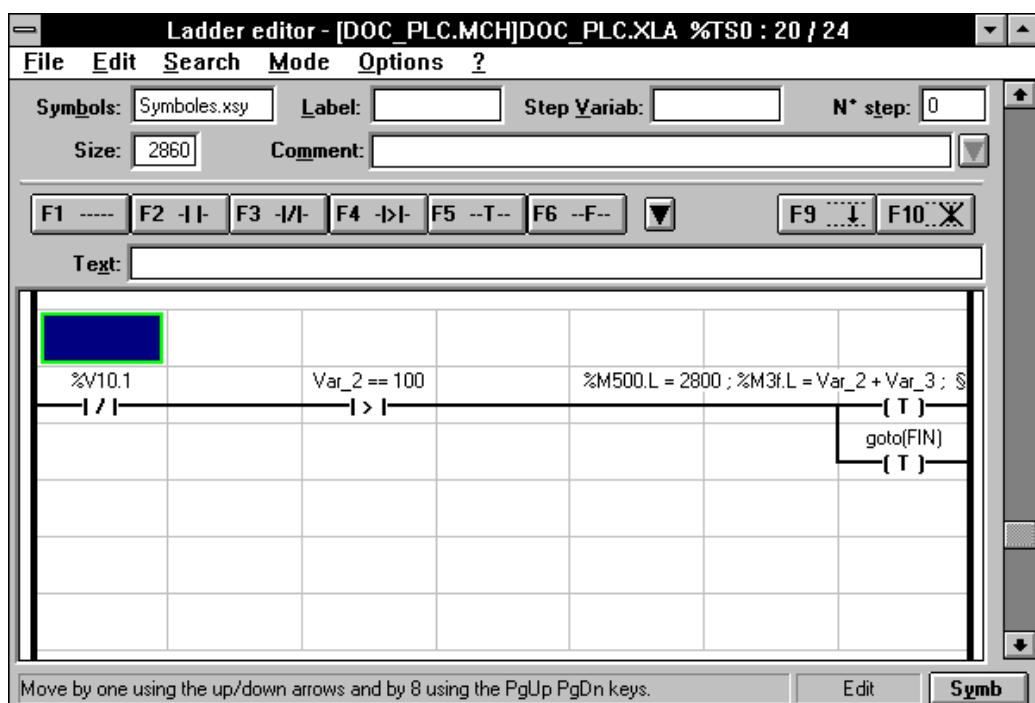
### 5.2.7.3 Multiple Numerical Assignments

Multiple numerical assignments are used to optimise the network size and speed.

#### Unoptimised Network



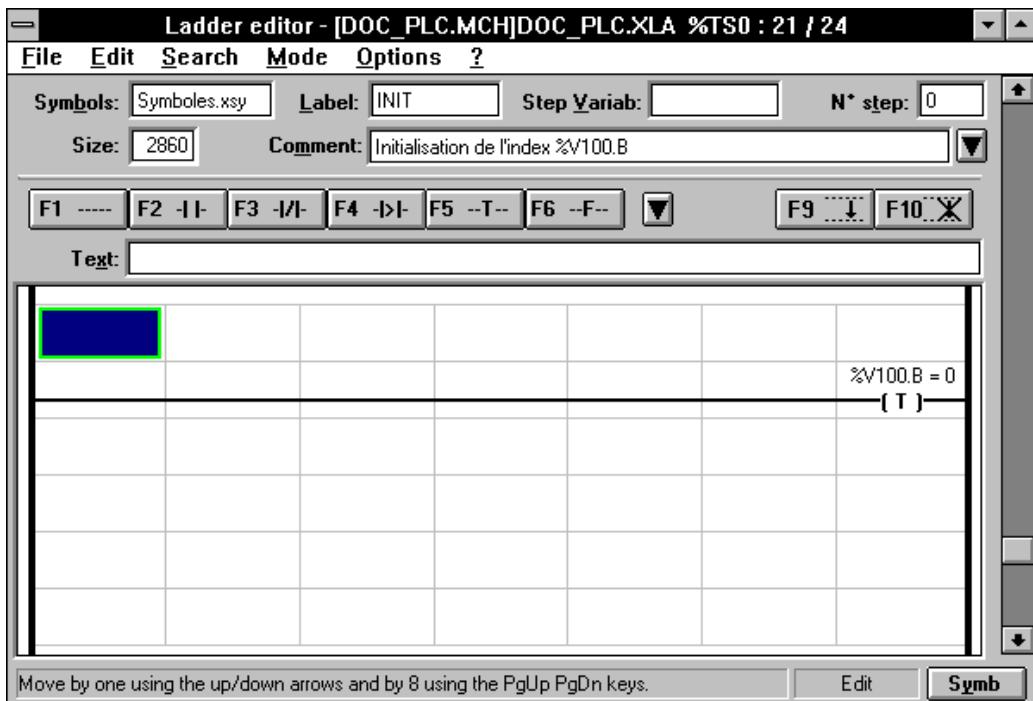
#### Optimised Network - 12 bytes less than the unoptimised network



### 5.2.7.4 Testing the Bits of a Byte, Word or Long Word

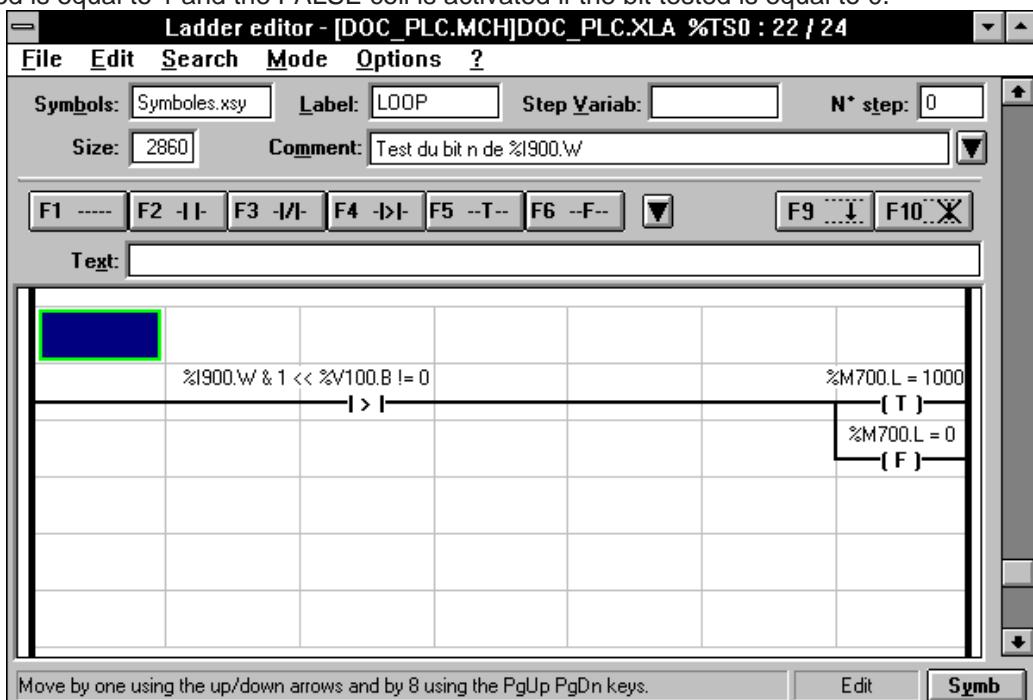
These sequences are used to test all the bits of variable %I900.W.

#### Sequence 1 - Initialise Index %V100.B

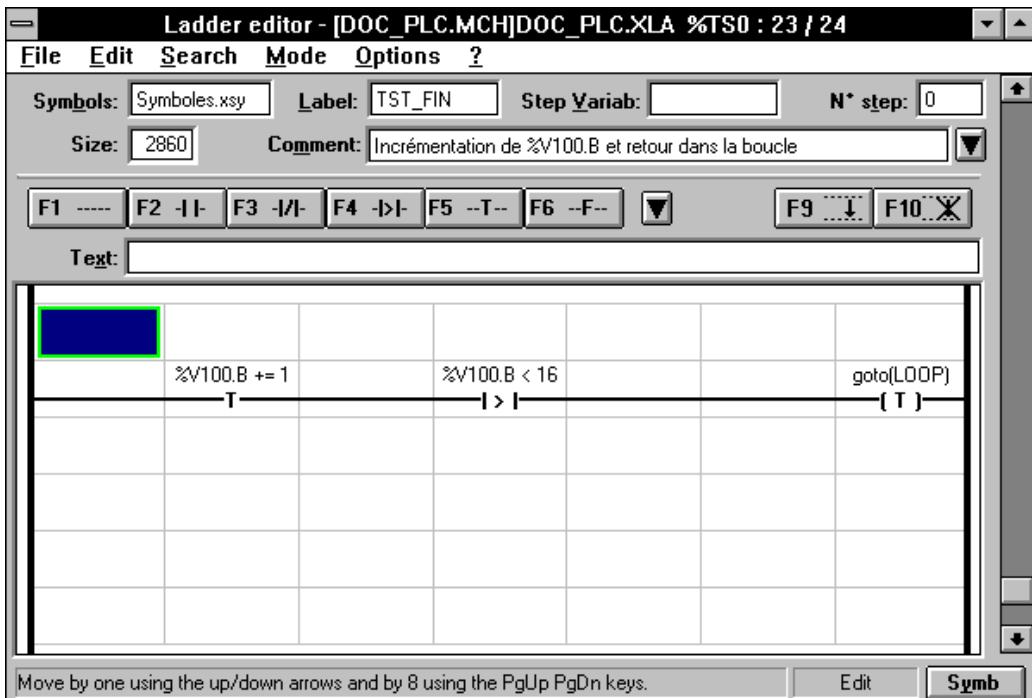


#### Sequence 2 - Test Each Bit of %I900.W

Arithmetic shift left by 1 of the value contained in %V100.B then logic AND with %I900.W (if %V100.B == 0, test bit 0 of %I900.W; if %V100.B == 1, test bit 1 of %I900.W and so forth). Result different from 0. The TRUE coil is activated if the bit tested is equal to 1 and the FALSE coil is activated if the bit tested is equal to 0.



### Sequence 3 - Increment index %V100.B and repeat loop if %V100.B < 16



## 5.3 Function Calls

Ladder language allows function calls.

The syntax is as follows:

```
[<numerical_variable> <assignment_operators>] <function_name> ( [<numerical_expression>]
{,<numerical_expression>}6)
```

The numerical assignment to the left of the function name is optional. It is used to recover the code returned by the function when the programmer wishes.

### Examples of Function Calls:

%M100.L = atoi(%M20.I)	// %M100.L receives the conversion result
bit(%M20.&, M30.&, 8);	// the return code is lost
cpyb(%V100.&, %V100.& + %M10.B, %V110.W + 10)	// the return code is lost

## 5.4 Parameter Check

The number of parameters passed is checked during compilation.

The values of the parameters passed can not be checked during compilation. The monitor checks them when calling the function, before executing it, only in the debugging mode.

## 6 General Purpose Functions

6.1 Convert an ASCII String to a Signed Integer of 32 Bits	atoi	6 - 3
6.2 Convert an ASCII String to a Signed Integer of 32 Bits	atoj	6 - 4
6.3 BCD —> Binary Conversion	bcd_bin	6 - 5
6.4 Binary —> BCD Code Conversion	bin_bcd	6 - 6
6.5 Separate Bits into Bytes	bit	6 - 7
6.6 Read the Parameters Stored on the Stack	cpyarg	6 - 8
6.7 Copy One or More Bytes	cpyb	6 - 9
6.8 Copy One or More Words	cpyw	6 - 10
6.9 Copy One or More Long Words	cpyl	6 - 11
6.10 Set Self-Test Period	diagiq	6 - 11
6.11 Convert a Signed Integer to an ASCII String	itoa	6 - 12
6.12 Convert an Unsigned Integer to an ASCII String	itostr	6 - 12
6.13 Concatenate Bytes into Bits	oct	6 - 13
6.14 Simulate Operator Panel Keyboard	putkey	6 - 15
6.15 Shortest Path Calculation	qcktool	6 - 15
6.16 Search for the Value of a Byte	rchb	6 - 16
6.17 Search for the Value of a Word	rchw	6 - 16
6.18 Search for the Value of a Long Word	rchl	6 - 17
6.19 Return to Calling Module or Network	return	6 - 18
6.20 Jump to a Module Label without Return	goto	6 - 19
6.21 Jump to a Module Label with Return	call	6 - 19
6.22 Flag	sema	6 - 20
6.23 Set One or More Bytes	setb	6 - 20
6.24 Set One or More Words	setw	6 - 21
6.25 Set One or More Long Words	setl	6 - 22
6.26 Call %SP Modules		6 - 22
6.26.1 Call an %SP Module	sp	6 - 22
6.26.2 Call an %SP Module with %Y Local Variables	spy	6 - 23
6.27 Format a Character String	sprintf	6 - 24
6.28 Integer Square Root	sqrt	6 - 25
6.29 Analyse an ASCII String	sscanf	6 - 25
6.30 Compare Two Character Strings	strcmp	6 - 26
6.31 Copy a Character String	strcpy	6 - 27
6.32 Calculate String Length	strlen	6 - 27
6.33 Swap the Even and Odd Bytes of a Word	swapw	6 - 28
6.34 Swap the Four Bytes of a Long Word	swapl	6 - 29

<b>6.35 Change Tool Wear Offset</b>	<b>tooldyn</b>	6 - 30
<b>6.36 Read n Variables E42000</b>	<b>R_E42000</b>	6 - 31
<b>6.37 Write n Variables E42000</b>	<b>W_E42000</b>	6 - 32
<b>6.38 Initialise the Base Associated with the %Y Variables</b>	<b>y_init</b>	6 - 33

## 6.1 Convert an ASCII String to a Signed Integer of 32 Bits

**atoi**

### Syntax

**atoi(&source)**

&source: Address of the ASCII string to be converted.

Returns a signed integer on 32 bits resulting from conversion of the ASCII string.

### Operation

Function atoi() reads the decimal digits from left to right.

Leading spaces and tab characters are ignored.

A sign (+ or -) can be specified to obtain a signed result.

Conversion is stopped when a ZERO byte or a character other than a decimal digit is detected.

In case of overflow, atoi returns the maximum positive value of a signed integer on 32 bits, i.e. 0x7FFFFFFF.

### Return code

#### If OK

Signed integer on 32 bits resulting from the conversion.

#### Error

0x7FFFFFFF: Overflow of conversion of a signed integer on 32 bits.

### Programming error causing a CPU fault

Access to a prohibited address:

- &source parameter error,
- end of string outside authorised area.

## 6.2 Convert an ASCII String to a Signed Integer of 32 Bits

**atoj**

### Syntax

```
atoj(&&end, &source)
```

**&&end:** Address of the long word (%M or %V) to be loaded with the address of the character on which conversion was stopped.

**&source:** Address of the ASCII string to be converted.

Returns a signed integer on 32 bits resulting from conversion of the ASCII string.

### Operation

Conversion is stopped when a ZERO byte or a character other than a decimal digit is detected.

This function operates in the same way as atoi(). Function atoj() loads address **&&end** with the address of the character on which conversion was stopped or zero if the end of the string was reached.

In case of overflow, atoj() returns the maximum positive value of a signed integer on 32 bits, i.e. 0x7FFFFFFF.

The long word at address **&&end** is loaded with:

- 0 if conversion was stopped on a ZERO byte at the end of the string,
- the address of the (nonzero) character on which conversion was stopped,
- -1 in case of overflow.

### Return code

#### If OK

Signed integer on 32 bits resulting from the conversion

#### Error

0x7FFFFFFF: Overflow of conversion of a signed integer on 32 bits.

### Programming error causing a CPU fault

Access to a prohibited address:

- **&source** parameter error,
- **&&end** parameter error,
- end of string outside authorised area.

## 6.3 BCD → Binary Conversion

## bcd\_bin

### Syntax

**bdc\_bin (BCD\_code)**

Code:                    Operand or numerical expression in BCD code.

### Operation

The operand, considered as signed, is extended on 32 bits before being placed in the stack. Conversion can only be performed on an operand in which each half-byte does not exceed the value 9 (in BCD code). If an error is detected, the function returns -1.

### Example:

%V0L.= bcd\_bin(%V4.L)

%V4.L contains the value 12345678 in BCD code.

Memory representation of %V4.L: 0001-0010-0011-0100-0101-0110-0111-1000

1      2      3      4      5      6      7      8

12345678 == 0xBC614E

Memory representation of %V0.L: 0000-0000-1011-1100-0110-0001-0100-1110

0      0      B      C      6      1      4      E

6

### ⚠ CAUTION

When the BCD operand is on 8 or 16 bits and the last half-byte is > 8, it is necessary to mask the parameter with the value 0xFF or 0xFFFF so as not to propagate the sign bit.

Example: bcd\_bin(%V0.B & 0xFF) ; bcd\_bin(%V0.W & 0xFFFF)

### Return Code

#### If OK

Conversion result

#### Error

-1: operand not in BCD code, one of the half-bytes > 9.

## 6.4 Binary → BCD Code Conversion

**bin\_bcd**

### Syntax

**bin\_bcd(binary\_code)**

Binary\_code:                   Operand or numerical expression in binary code.

### Operation

The operand considered as signed can have a width of 8, 16 or 32 bits. It is extended on 32 bits before being placed in the stack. Conversion can only be performed on an operand between 0 and 99999999. If the operand is outside these limits, conversion is incorrect and the function returns -1.

### Examples:

%V0.W=bin\_bcd(1234)

1234==0x4D2	Memory representation	0000-0100-1101-0010 0      4      D      2
%V0.W	Memory representation	0001-0010-0011-0100 1      2      3      4

%V0.L=bin\_bcd(12345678)

12345678==0xBC614E	Memory representation	0000-0000-1011-1100-0110-0001-0100-1110 0      0      B      C      6      1      4      E
%V0.L	Memory representation	0001-0010-0011-0100-0101-0110-0111-1000 1      2      3      4      5      6      7      8

### Return Code

#### If OK

Conversion result

#### Error

-1: Operand not between 0 and 99999999.

**bit****6.5 Separate Bits into Bytes****Syntax**

```
bit(&dest, &source, n)
```

&dest: Address of the first destination byte.

&source: Address of the first byte to be separated.

n: Number of bytes to be separated.

Separation of n bytes starting from bit 0 at address &source into the MSBs of 8xn bytes starting at address &dest.

**Operation**

Bit 0 of the byte at address &source is copied into bit 7 of the byte at address &dest; the 7 other bits are reset.

Bit 1 of the byte at address &source is copied into bit 7 of the byte at address &dest + 1; the 7 other bits are reset.

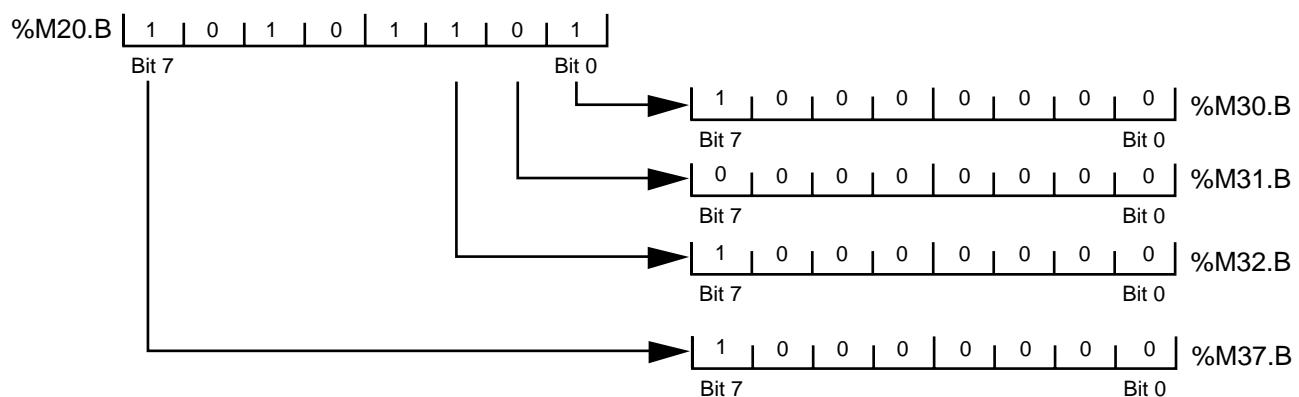
Bit 0 of the byte at address &source + 1 is copied into bit 7 of the byte at address &dest + 8; the 7 other bits are reset.

The process continues until n bytes have been separated.

**REMARK** The function *oct()* performs the reverse operation (see Sec. 6.13).

**Example**

**bit(%M30, %M20.&, 3)**



## Return code

### If OK

Not significant

### Error

-1: n negative

## Programming error causing a CPU fault

Access to a prohibited address:

- &source parameter error,
- &dest parameter error,
- &source+n outside authorised area,
- &dest+n outside authorised area.

## 6.6 Read the Parameters Stored on the Stack

# cpyarg

### Syntax

```
cpyarg(&dest, n)
```

&dest: Address of the memory block where the system copies the arguments.

n: Number of arguments to be copied (maximum 6).

Copies n arguments from the stack into local memory starting at address &dest, when the module is called by sp().

### Operation

Each argument occupies 32 bits.

Function cpyarg() must be called at the beginning of the %SP module before the stack is modified, such as would occur by a call to an internal label of the module (call(<label>)).

If the number of arguments n specified is greater than the number of arguments m passed during the call, the system does not generate an error but obviously only the first m arguments are significant.

## Return code

### If OK

Not significant.

### Error

-1: n negative, zero or greater than maximum number authorised

## Programming error causing a CPU fault

Access to a prohibited address:

- &dest parameter error,
- &dest+n outside authorised area.

## 6.7 Copy One or More Bytes

**cpyb**

### Syntax

```
cpyb(&dest, &source, n)
```

&dest: Destination address.  
 &source: Source address.  
 n: Number of bytes to be copied.

Copies n bytes from the source to the destination.

### Direction of Transfer

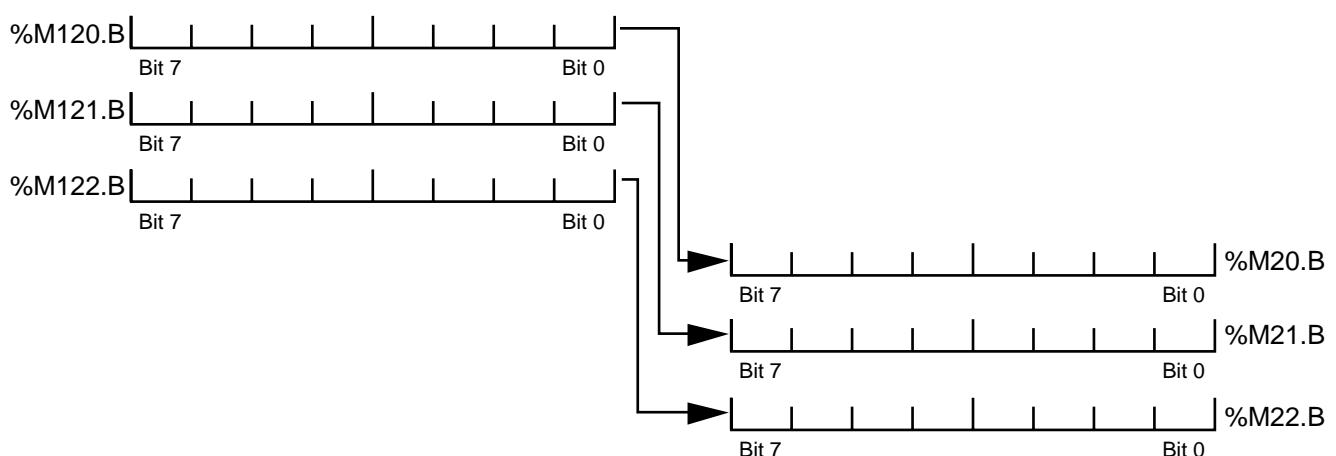
To allow transfers to memory areas which overlap, the order of copying depends on the &dest and &source addresses.

- If &dest < &source, the copy is made from the beginning to the end (increasing addresses)
- If &dest > &source, the copy is made from the end to the beginning (decreasing addresses).

### Example

cpyb(%M120.&, %M20.&, 3)

6



### Return code

#### If OK

0

#### Error

-1: n negative or zero

### Programming error causing a CPU fault

Access to a prohibited address:

- &source parameter error,
- &dest parameter error,
- &source+n outside authorised area,
- &dest+n outside authorised area.

## 6.8 Copy One or More Words

**cpyw**

### Syntax

```
cpyw(&dest, &source, n)
```

&dest: Destination address.

&source: Source address.

n: Number of words to be copied.

Copies n words from the source to the destination.

### Direction of Transfer

Refer to Section 6.5.

### Return code

#### If OK

0

#### Error

-1: n negative or zero

### Programming error causing a CPU fault

Access to a prohibited address:

- &source parameter error,
- &dest parameter error,
- &source+n outside authorised area,
- &dest+n outside authorised area.

**6.9 Copy One or More Long Words****cpyl****Syntax****cpyl(&dest, &source, n)**

&dest: Destination address.  
 &source: Source address.  
 n: Number of words to be copied.

Copies n long words from the source to the destination.

**Direction of Transfer**

Refer to Section 6.5.

**Return code**

If OK

0

6

Error

-1: n negative or zero

**Programming error causing a CPU fault**

Access to a prohibited address:

- &source parameter error,
- &dest parameter error,
- &source+n outside authorised area,
- &dest+n outside authorised area.

**6.10 Set Self-Test Period****diagiq****Syntax****diagiq(period)**

period: Self-test period (in tenths of a second).

The system cyclically reads the internal status of the cards connected to the serial bus (%I, %Q) and refreshes diagnostic word %Irc3C.W.

The default period is 400 milliseconds.

Function diagiq() is used to inhibit the self-test or modify the default period. The period parameter must be ZERO to inhibit the self-test or between 1 (0.1 second) and 10 (1 second). It should be noted that a short period involves an extra workload for the CPU.

diagiq() must be called in task %INI.

**Return code**If OK

0

Error

-1:

Period invalid (not between 0 and 10) (the default period remains valid).

## 6.11 Convert a Signed Integer to an ASCII String

**itoa****Syntax****itoa(i, &dest)**

i: Integer to be converted (the value is considered signed).

&amp;dest: Address of the ASCII\_ZERO string where the ASCII characters will be stored.

Converts a signed base 10 integer. The resulting ASCII characters are stored in the string at address &amp;dest. The string ends with a ZERO byte.

**Return code**If OK

Number of characters in the string not counting the end ZERO byte.

**Programming error causing a CPU fault**

Access to a prohibited address:

- &dest parameter error,
- end of string outside authorised area.

## 6.12 Convert an Unsigned Integer to an ASCII String

**itostr****Syntax****itostr(u, &dest, base)**

u: Integer to be converted (the value is considered unsigned).

&amp;dest: Address of the string into which the ASCII characters will be loaded.

base: Conversion base.

Converts an unsigned integer in the base specified. The resulting ASCII characters are loaded into the string at address &amp;dest. The string ends with a ZERO byte.

The base must be between 2 and 36. The default base is 10.

**Return code**If OK

Number of characters in the string not counting the end ZERO byte.

**Programming error causing a CPU fault**

Access to a prohibited address:

- &dest parameter error,
- end of string outside authorised area.

**6.13 Concatenate Bytes into Bits****oct****Syntax**

<b>oct(&amp;dest, &amp;source, n)</b>
---------------------------------------

&dest: Address of the first destination byte.

&source: Address of the first byte to be concatenated.

n: Number of destination bytes to be concatenated.

Concatenates the MSBs of  $8 \times n$  bytes from &source into the bytes starting at &dest.

6

**Operation**

Bit 7 of the byte at address &source is copied into bit 0 of the byte at address &dest.

Bit 7 of the byte at address &source + 1 is copied into bit 1 of the byte at address &dest.

.....

Bit 7 of the byte at address &source + 8 is copied into bit 0 of the byte at address &dest + 1.

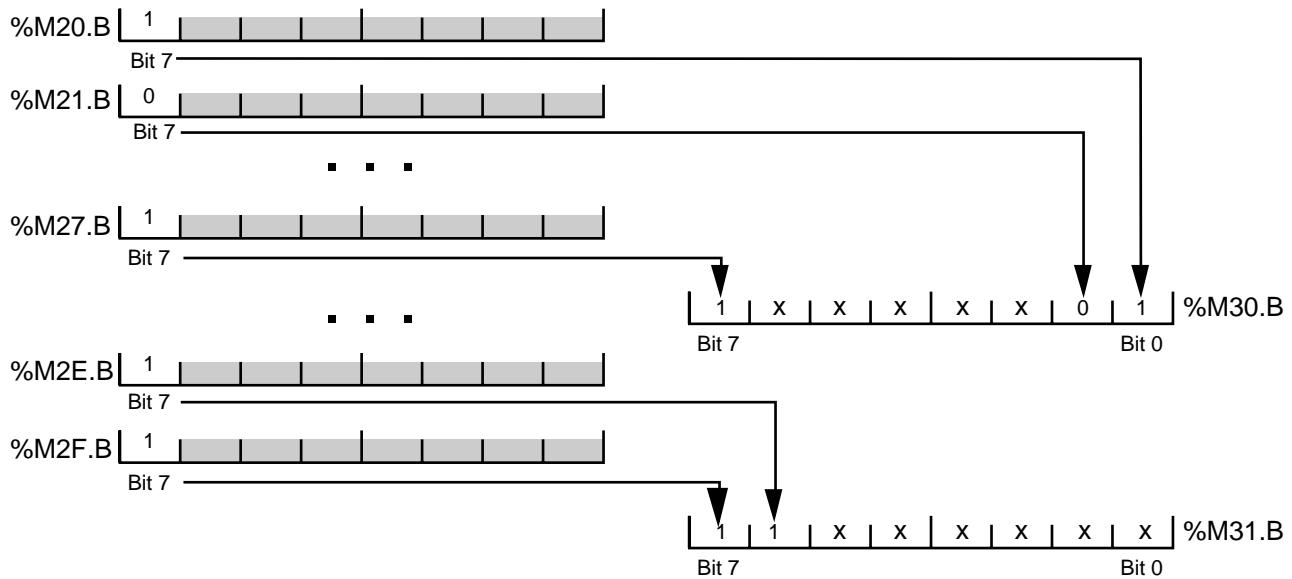
.....

Bit 7 of the byte at address &source + (n - 1) x 8 is copied into bit 0 of the byte at address &dest + (n - 1).

.....

Bit 7 of the byte at address &source + (n - 1) x 8 + 7 is copied into bit 7 of the byte at address &dest + (n - 1).

**Example:** oct(%M30.&, %M20.&,2)



#### Return code

If OK

0

Error

-1: n negative or zero

#### Programming error causing a CPU fault

Access to a prohibited address:

- &source parameter error,
- &dest parameter error,
- &source+n outside authorised area,
- &dest+n outside authorised area.

## 6.14 Simulate Operator Panel Keyboard

**putkey**

### Syntax

**putkey(key\_code)**

key\_code: ASCII code of an operator panel key.

Simulation of the operator panel keyboard by the machine processor.

### Operation

Function putkey() is enabled if the panel is absent (%W5.0 = 1).

To make sure a simulated key code is accepted by the CNC, wait for the return code to become 0 after sending the key code. This means that the key code was accepted by the CNC, but there is no way of knowing whether the code will be processed. It is therefore recommended to apply a timeout of at least 100 ms before a new call to putkey().

**REMARK** *The value 0xAF can be set in the argument «Key\_code» to call the transparent mode directly.*

### Return code

If OK

0

Error

-1: Operator panel keyboard not disabled.

1: Buffer full, repeat the call to putkey(..)

## 6.15 Shortest Path Calculation

**qcktool**

### Syntax

**qcktool(origin, destination, n)**

origin: Origin pocket number (see Remark).

destination: Destination pocket number (see Remark).

n: Number of pockets in the tool carousel.

Function qcktool() determines the number of pockets and the optimum direction of rotation to go from the origin pocket to the destination pocket in a tool carousel.

**REMARK** *The pockets are numbered from zero up (from 0 to n-1).*

## Return code

### If OK

- If > 0: The positive direction (increasing numbers) is the shortest. Indicates the number of steps.
- If < 0: The negative direction (decreasing numbers) is the shortest. The absolute value indicates the number of steps.
- If = 0: No movement is required because the carousel is already in the destination position.
- If = n: Outside carousel.

## 6.16 Search for the Value of a Byte

rchb

### Syntax

**rchb(&source, b, step, n)**

- &source: Search start address.  
b: Value of the byte to be searched for.  
step: Value of the search step in bytes.  
n: Maximum number of search steps.

Search, with a step, for the first occurrence of byte b starting from address &source.

The step can be positive or negative.

- Positive step: The search is made by increasing addresses.  
Negative step: The search is made by decreasing addresses.

### Return code

#### Value found

Positive number equal to the number of steps made to the first occurrence.

- Positive step: Return code = (occurrence address - &source)/step  
Negative step: Return code = (&source - occurrence address)/(-step)

#### Value not found

- 1: Value not found

### Programming error causing a CPU fault

Access to a prohibited address:

- &source parameter error,
- &source+step\*n outside authorised area.

**6.17 Search for the Value of a Word****rchw****Syntax****rchw(&source, w, step, n)**

&source:	Search start address.
w:	Value of the word to be searched for.
step:	Value of the search step in bytes.
n:	Maximum number of search steps.

Search, with a step, for the first occurrence of word w starting from address &source.

The step can be positive or negative.

Positive step:	The search is made by increasing addresses.
Negative step:	The search is made by decreasing addresses.

**Return code**Value found

Positive number equal to the number of steps made to the first occurrence.

Positive step:	Return code = (occurrence address - &source)/step
Negative step:	Return code = (&source - occurrence address)/(-step)

Value not found

-1: Value not found

**Programming error causing a CPU fault**

Access to a prohibited address:

- &source parameter error,
- &source+step\*n outside authorised area.

**6.18 Search for the Value of a Long Word****rchl****Syntax****rchl(&source, l, step, n)**

&source:	Search start address.
l:	Value of the long word to be searched for.
step:	Value of the search step in bytes.
n:	Maximum number of search steps.

Search, with a step, for the first occurrence of long word l starting from address &source.

The step can be positive or negative.

Positive step: The search is made by increasing addresses.

Negative step: The search is made by decreasing addresses.

#### Return code

##### Value found

Positive number equal to the number of steps made to the first occurrence.

Positive step: Return code = (occurrence address - &source)/step

Negative step: Return code = (&source - occurrence address)/(-step)

##### Value not found

-1: Value not found

#### Programming error causing a CPU fault

Access to a prohibited address:

- &source parameter error,
- &source+step\*n outside authorised area.

## 6.19 Return to Calling Module or Network

**return**

#### Syntax

```
return([numerical_expression])
```

numerical\_expression: Value returned to the calling module.

#### Operation

Returns:

- To the calling module in the case of an intermodule call with the form <variable> = sp(...). In this case, the value of the numerical expression is recovered in <variable>.
- To the calling ladder network in the case of an intra-module call with the form call(<label>). In this case, the returned value, if any, cannot be recovered.



This function cannot be called in the test zone.

#### Return code

No code is reported by the function itself.

**REMARK** An assignment with the form %M20.B = return(Var\_ 1+3) is meaningless.

## 6.20 Jump to a Module Label without Return

**goto**

### Syntax

```
goto(<label>)
```

label: Label of the ladder network called.

### Operation

Jump to a sequence without return.



**CAUTION**

This function cannot be called in the test zone.

### Return code

No code is returned.

## 6.21 Jump to a Module Label with Return

**call**

### Syntax

```
call(<label>)
```

Label: Label of the ladder network called.

### Operation

Jump to a sequence with return to the coil following the call() on the first return() encountered.



**CAUTION**

This function cannot be called in the test zone.

### Return code

No code is returned.

## 6.22 Flag

**sema**

### Syntax

**sema(&flag)**

&flag: Address of the flag byte.

Used for an uninterruptible instruction such as Test and Set to set the byte at address & flag to 0x80 (-128).

This function is used when several tasks share the same resource (e.g. keyboard, screen, etc.).

### Return code

#### Flag State

0: The flag was free.

1: The flag was already set.

#### Programming error causing a CPU fault

Access to a prohibited address:

- &flag parameter error.

## 6.23 Set One or More Bytes

**setb**

### Syntax

**setb(&dest, b, n)**

&dest: Destination address.

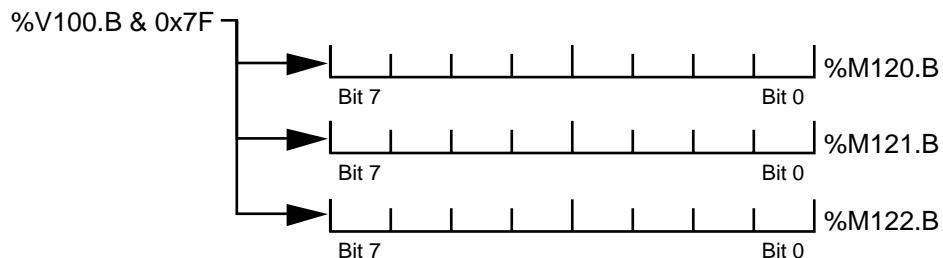
b: Value of the byte to be set.

n: Number of bytes to be set.

Sets n bytes to the value b starting at address &dest.

### Example of Use

setb(%M120.&, %V100.B & 0x7f, 3)



**Return code**If OK

Not significant.

Error

-1: n negative or zero

**Programming error causing a CPU fault**

Access to a prohibited address:

- &dest parameter error,
- &dest+n outside authorised area.

**6.24 Set One or More Words****setw****Syntax****setw(&dest, w, n)**

6

&dest: Destination address.

w: Value of the word to be set.

n: Number of words to be set.

Sets n words to the value w starting at address &dest.

**Return code**If OK

Not significant.

Error

-1: n negative or zero

**Programming error causing a CPU fault**

Access to a prohibited address:

- &dest parameter error,
- &dest+n outside authorised area.

## 6.25 Set One or More Long Words

**setl**

### Syntax

```
setl(&dest, l, n)
```

&dest: Destination address.

l: Value of the long word to be set.

n: Number of long words to be set.

Sets n long words to the value l starting at address &dest.

### Return code

#### If OK

Not significant.

#### Error

-1: n negative or zero

### Programming error causing a CPU fault

Access to a prohibited address:

- &dest parameter error,
- &dest+n outside authorised area.

## 6.26 Call %SP Modules

### 6.26.1 Call an %SP Module

**sp**

### Syntax

```
sp(moduleno{, argn}6 ...)
```

moduleno: Number of the %SP module to be called.

argn: Optional argument.

Calls an %SP module (%SP0, ..., %SP255) and passes it any arguments via the stack.

### Operation

The module number must be between 0 (call to %SP0) and 255 (call to %SP255).

The arguments are extended to 32 bits and placed on the stack. The call is then made.

The total number of arguments (including moduleno) must not exceed NBM\_PARAM (set to 7).

A call to function cpyarg() at the start of the called module allows the arguments passed via the stack to be recovered.

## Return code

### If OK

Value returned by the %SP module called using function return (<numerical\_expression>).

Not significant if the called module does not return a value.

## Example of Use of sp(), cpyarg(), return()

### Exchange of arguments during a call to an %SP module

Calling module (%TS, %TF or %SP):

%M100.W = sp(33, 10, %M20.B + %M30.B); Arguments 10 and (%M20.B + %M30.B) are extended to 32 bits and stored on the stack. The call to %SP33 is then made.

Module called %SP33:

cpyarg(M200.&, 2);	Copies the two parameters of the call to local memory starting from %M200. The value 10 is stored in %M200.L and the result of the expression (%M20.B + %M30.B) is stored in %M204.L.
return(%V100.W+25);	Return to calling module. The value of expression %V100.W + 25 is loaded in %M100.W.

## Recommendation

Passing arguments avoids inter-module communication via common variables.

This programming concept is recommended because it keeps the modules independent, thereby facilitating their reuse in other applications.

## 6.26.2 Call an %SP Module with %Y Local Variables

**spy**

### Syntax

**spy(moduleno {, argn}6 ...)**

moduleno: Number of the %SP module to be called.

argn: Optional argument.

Calls an %SP module (%SP0 to %SP255) with creation of 128 %Y local variables and passes it any arguments via the stack.

### Operation

The module number must be between 0 (call to %SP0) and 255 (call to %SP255).

128 %Y local variables are created in the stack. These variables are deleted on return to the calling programme.

The arguments are extended on 32 bits and stored on the stack except moduleno which is not stacked.

The total number of arguments (including moduleno) must not exceed NBM\_PARAM (i.e. 7).

spy(..) and %Y variables are used to write relocatable, reentrant %SP modules.

**REMARK** *The sample programme L\_E\_VAR.MCH available under PLCTOOL illustrates instruction spy().*

### Organisation of the %Y Variables Available in the %SP Modules Called:

- %Y0.L Contains the first argument if any; else don't care.
- %Y4.L Contains the second argument if any; else don't care.
- %Y14.L Contains the last argument if any; else don't care.
- %Y18.B Rest of the local variables.
- %Y7F.B Last local variable.

### Code Returned

#### If OK

Value returned by the %SP module called using function return(<numerical\_expression>).

Not significant if the called module does not return a value.

### Example of Use of spy(..) and return(..)

#### Exchange of arguments during a call to an %SP module

Calling module (%TS, %TF or %SP):

%M100.W = spy(33, 10, %M20.B + %M30.B); Creation of 128 %Y local variables in the stack.  
Arguments 10 and (%M20.B + %M30.B) are extended to 32 bits  
and stored in the stack. The call to %SP33 is made.

Module called %SP33

%Y0.L contains 10

%Y4.L contains the result of the expression (%M20.B + %M30.B).

return(%Y10.W + 25)

Return to the calling module. The local variables are deleted  
and the result of the expression (%Y10.W + 25) is loaded in  
%M100.W.

## 6.27 Format a Character String

**sprintf**

### Syntax

```
sprintf(&dest, &format {,&argn}5)
```

&dest: Destination string address.

&format: Format string address.

&argn: Possible argument.

Formats the string at address &format and copies it at address &dest. A ZERO byte is added at the end of &dest.

Function sprintf() supports the ANSI standard C language conversion specifications.

### Operation

Function sprintf() is equivalent to printf() except that the formatted string is copied starting at address &dest instead of being displayed.

For specification of the conversion formats, see function printf().

**Return code**If OK

Number of characters written in &dest not counting the terminal ZERO byte.

Error

- 1: Format string containing invalid formats.

**Programming error causing a CPU fault**

Access to a prohibited address:

- &dest parameter error,
- &format parameter error,
- end of string outside authorised area.

**6.28 Integer Square Root****sqrt****Syntax****sqrt(n)**

6

n: Positive integer.

Returns the integer square root of n.

The calculation time is less than 60 microseconds.

**Return code**If OK

Positive integer closest to the square root of n.

**6.29 Analyse an ASCII String****sscanf****Syntax****sscanf(&sourcestring, &formatstring, {, &argn}5)**

&sourcestring: Source string address.

&formatstring: Format string address.

&argn: Address of the variables to be loaded.

Analyses an ASCII string (ending with a ZERO byte) at address &sourcestring and sets the parameters according to the format string conversion specifications.

Function sscanf() supports the ANSI standard C language conversion specifications for C language conversion to ANSI standard.

## Operation

Each argument &argn must be the address of a variable %M, %V, %Q or %W.

For specification of the conversion format, refer to function printf() (see Sec. 8.2.5).

## Return code

### If OK

Number of parameters effectively loaded.

### Error

0: Unsuccessful analysis of the source string, format string containing invalid formats.

## Programming error causing a CPU fault

Access to a prohibited address:

- &sourcestring parameter error,
- &formatstring parameter error,
- &argn parameter error for %d, %E, %C, %f, %G, %g, %i, %n, %o, %P, %u, %X or %x,
- &argn parameter error for %s,
- end of string outside authorised area.

## 6.30 Compare Two Character Strings

## strcmp

### Syntax

```
strcmp(&string1, &string2)
```

&string1: String 1 address.

&string2: String 2 address.

Compares two strings ending with a ZERO byte.

## Return code

### If OK

n == 0 String 1 == string 2.

n > 0 String 1 > string 2 (byte i of string 1 > byte i of string 2).

n < 0 String 1 < string 2 (byte i of string 1 < byte i of string 2).

## Programming error causing a CPU fault

Access to a prohibited address:

- %string1 parameter error,
- %string2 parameter error,
- end of string outside authorised area.

## 6.31 Copy a Character String

**strcpy**

### Syntax

**strcpy(&dest, &source)**

**&dest:** Destination address.

**&source:** Source address.

Copies the bytes of the string starting at address **&source** into **&dest**.

The copy stops on the first ZERO byte of the source string. A ZERO byte is copied at the end of **&dest**.

### Return code

#### If OK

Pointer returned to the destination.

#### Programming error causing a CPU fault

Access to a prohibited address:

- **&source** parameter error,
- **&dest** parameter error,
- end of string outside authorised area.

## 6.32 Calculate String Length

**strlen**

### Syntax

**strlen(&string)**

**&string:** String start address.

Calculates the length of a string (number of bytes before the first ZERO byte).

### Return code

#### If OK

String length.

#### Programming error causing a CPU fault

Access to a prohibited address:

- **&string** parameter error,
- end of string outside authorised area.

## 6.33 Swap the Even and Odd Bytes of a Word

## Swapw

### Syntax

```
swapw(&dest, &source, n)
```

&dest: Destination address.

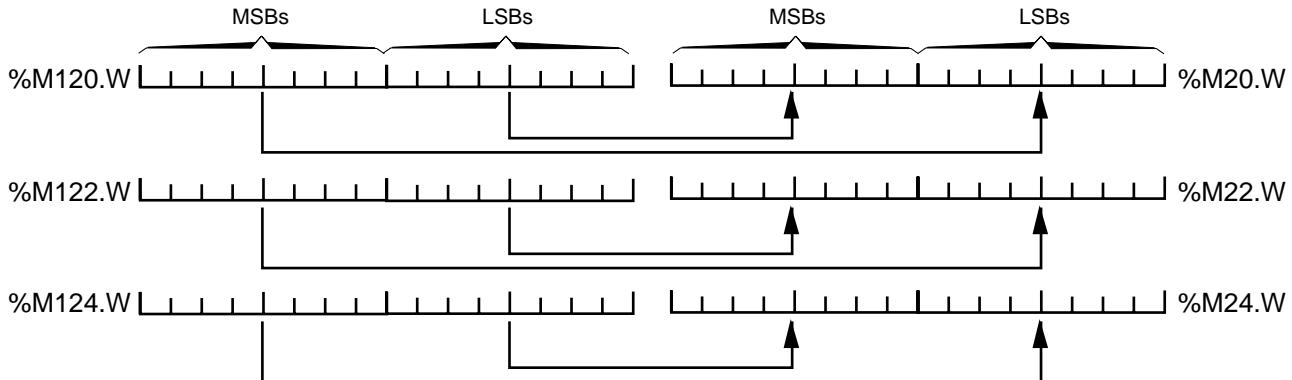
&source: Source address.

n: Number of words to be copied.

Copies n words from &source into &dest, swapping the even and odd bytes of each word.

### Example

```
swapw(%M20.&, %M120.&, 3)
```



### Return code

If OK

Not significant.

Error

-1: n negative or zero

### Programming error causing a CPU fault

Access to a prohibited address:

- &dest parameter error,
- &source parameter error,
- &dest+n outside authorised area,
- &source+n outside authorised area.

## 6.34 Swap the Four Bytes of a Long Word

**swapl**

### Syntax

```
swapl(&dest, &source, n)
```

&dest: Destination address.

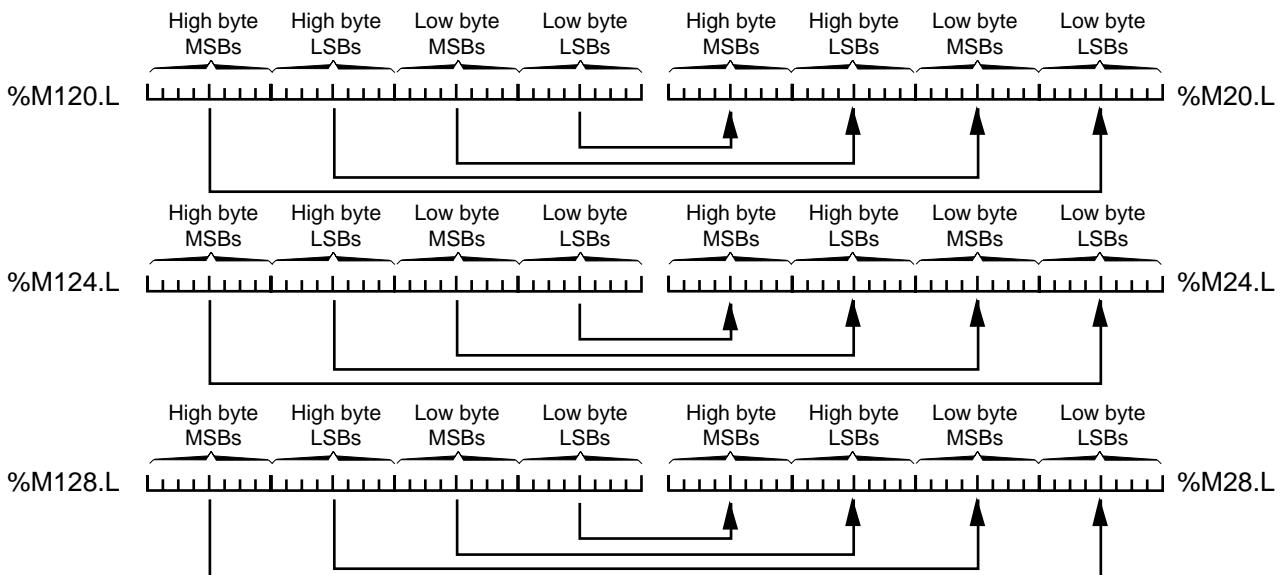
&source: Source address.

n: Number of long words to be copied.

Copies n long words from &source into &dest, swapping the four bytes of each word.

### Example

swapl(%M20.&, %M120.&, 3)



6

### Return code

#### If OK

Not significant.

#### Error

-1: n negative or zero

### Programming error causing a CPU fault

Access to a prohibited address:

- &dest parameter error,
- &source parameter error,
- &dest+n outside authorised area,
- &source+n outside authorised area.

## 6.35 Change Tool Wear Offset

**tooldyn**

### Syntax

```
tooldyn(correction, axis, toolno)
```

correction: Value of the wear offset change (signed integer on 16 bits in the internal system unit (see Parameter Manual)).

axis: Type of correction.

toolno: Tool number.

Modify a tool wear offset (the wear offsets changes are accumulated by the CNC).

### Operation

It is recommended to leave at least one RTC between processing of two tooldyn(..) functions.

Axis:

 Bit 7   Bit 0	0x1 : Wear offset increment in X (lathe) or L (milling machine)
 Bit 7   Bit 0	0x2 : Wear offset in Z (lathe) or R (milling machine)
 Bit 7   Bit 0	0x81 : Wear offset reset in X (lathe) or L (milling machine)
 Bit 7   Bit 0	0x82 : Wear offset reset in Z (lathe) or R (milling machine)
 Bit 7   Bit 0	0x83 : Wear offset reset in C and Z (lathe) or L and R (milling machine)

### Return code

#### If OK

- 0
- 1: Function refused - Queue saturated by a tooldyn(..) function sent earlier and still being processed.

**R\_E42000****6.36 Read n Variables E42000****Syntax**

```
R_E42000(&dest, number, n)
```

&dest: Destination address.  
 Number: Number of the first E42000 variable to be read (0 ... 127).  
 n: Number of bytes to be read (1 to 128).  
 Copies n bytes starting from variable E42000 + number to the area starting at &dest.

**Example**

```
R_E42000(%V100.&, 120, 7)
```



6

**Return code**If OK

0

Error

-1: number &gt; 127

number+n &gt; 128

**Programming error causing a CPU fault**

Access to a prohibited address:

- &dest parameter error,
- &dest+n outside authorised area.

## 6.37 Write n Variables E42000

**W\_E42000**

### Syntax

```
W_E42000(&source, number, n)
```

&source: Source address.

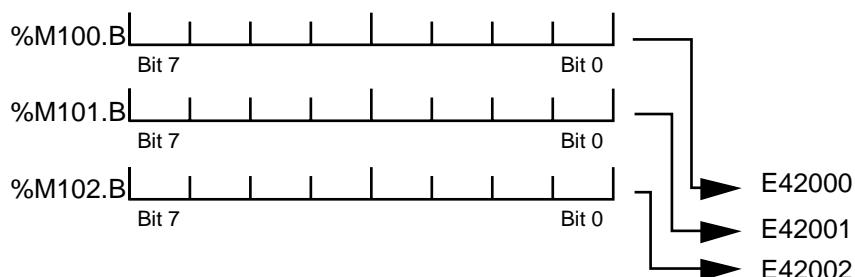
Number: Number of the first E42000 variable to be written (0 ... 127).

n: Number of bytes to be written (1 to 128).

Copies n bytes from &source to variable E42000 + number.

### Example

```
W_E42000(%M100.&, 0, 3)
```



### Return code

If OK

0

Error

-1: number > 127

number+n > 128

### Programming error causing a CPU fault

Access to a prohibited address:

- &source parameter error,
- &source+n outside authorised area.

## 6.38 Initialise the Base Associated with the %Y Variables

**y\_init**

### Syntax

```
y_init(&y_start_address)
```

y\_start\_address: Address loaded in the base associated with the %Y variables.

Loads the base associated with the %Y variables with the address passed as parameter.

### Operation

The %Y variables can replace any %M, %V, %I, %Q, %R and %W global variables. The programmer must initialise the base by function y\_init(..) before using the %Y variables.

The use of function y\_init(..) inhibits visibility of any local variables of the %SP module. To recover visibility, proceed as follows:

Var_1 = %Y0.&	Saves the base in Var_1 (e.g.: %V100.L).
y_init(Var_2 + 100)	The base points to a new variable area.
.....	Use of new %Y variables.
y_init(Var_1)	Restore the base.
.....	Use of local %Y variables.

**REMARK** Loading of a base associated with the %Y variables by function y\_init is only possible during execution of the task (%TS, %TF, %TH or %INI) using the variables.

#### Example:

- If the %Y variables are used in task %TS0, function y\_init must be called each time task %TS0 is executed
- If the %Y variables are used in a looped TF background task (iterative background task), function y\_init only needs to be called once at the beginning of the task.

### Example of Use of y\_init(..) and %Y

Processing of 8 axis groups with a single %SP0 using %Y variables

y_init(%R100.&)	%Y0.B replaces %R100.B	%Y80.B replaces %W100.B
sp(0)	Processing of axis group 1	
y_init(%R200.&)	%Y0.B replaces %R200.B	%Y80.B replaces %W200.B
sp(0)	Processing of axis group 2	
y_init(%R800.&)	%Y0.B replaces %R800.B	%Y80.B replaces %W800.B
sp(0)	Processing of axis group 8	

### Processing of a String

%V100.L = "ABCDEF"	%V100.L contains the string start address.
y_init(%V100.L)	The base points to the start of the string.
%Y0.B == A	%Y0.B corresponds to the first character in the string.
%Y1.B == B	%Y1.B corresponds to the second character in the string.
%Y5.B == F	

### **Return Code**

Not significant.

### **Programming error causing a CPU fault**

Access to a prohibited address:

- &y\_start\_address parameter error.

---

## 7 Task Management

7.1 Introduction	7 - 3
7.2 Start a Critical Section	<b>csbegin</b> 7 - 3
7.3 End a Critical Section	<b>csend</b> 7 - 3
7.4 Suspend a %TF task	<b>whtr</b> 7 - 3
7.5 Start a %TF Task	<b>tfstart</b> 7 - 4
7.6 Stop a %TF Task	<b>tfstop</b> 7 - 4



## 7.1 Introduction

For further information concerning processing of the background tasks, see Section 2.1.2.3.

## 7.2 Start a Critical Section

**csbegin**

### Syntax

```
csbegin()
```

### Operation

Inhibits pre-emption of the calling task by another %TS, %TH or %TF task.

### Return code

Always OK

0

## 7.3 End a Critical Section

**csend**

### Syntax

```
csend()
```

### Operation

Authorises pre-emption of the calling task by a higher priority task. This function cancels the effects of function csbegin().

### Return code

Always OK

0

## 7.4 Suspend a %TF task

**whtr**

### Syntax

```
whtr(n)
```

n: Number of real time clock cycles during which the %TF task is on WAIT.

### Operation

Changes the calling %TF task from EXECUTING state to WAITING state for n RTCs. At the end of this time, the %TF task goes into READY state. n must be between 0 and 255, (equivalent to 0 to 5.1 secs).

**Return code**

If OK

0

**7.5 Start a %TF Task****tfstart****Syntax**

```
tfstart(tf_number)
```

tf\_number: Number of the %TF task.

**Operation**

Sets the %TF task to READY state.

**Return code**

If OK

0

**7.6 Stop a %TF Task****tfstop****Syntax**

```
tfstop(tf_number)
```

tf\_number: Number of the %TF task.

**Operation**

Sets the %TF task to NOT READY state.

**Return code**

If OK

0

# 8 Transparent Mode

<b>8.1 Introduction</b>		8 - 3
8.1.1	Display Management	8 - 3
8.1.2	Exchange Variable	8 - 4
8.1.3	Transmission of Character Codes to the Screen	8 - 4
8.1.4	Character Codes Used by %R0.W and putkey()	8 - 5
<b>8.2 Functions Assigned to transparent Mode</b>		8 - 7
8.2.1	Position the Cursor	<b>pcur</b> 8 - 7
8.2.2	Display a Character	<b>putchar</b> 8 - 7
8.2.3	Display a String without Formatting	<b>puts</b> 8 - 8
8.2.4	Display a Buffer	<b>print</b> 8 - 8
8.2.5	Display a Character String with Formatting	<b>printf</b> 8 - 9
8.2.6	Open a Keyboard Acquisition	<b>scano</b> 8 - 12
8.2.7	Open a Numerical Keypad Acquisition	<b>scanu</b> 8 - 13
8.2.8	Read a String	<b>scans</b> 8 - 13
8.2.9	Read and Convert a Decimal Number	<b>scand</b> 8 - 14
8.2.10	Read and Convert a Hexadecimal Number	<b>scanx</b> 8 - 15
8.2.11	Close a Keyboard Acquisition	<b>scanc</b> 8 - 16
8.2.12	Position and Display an Image	<b>putimage</b> 8 - 16
8.2.13	Graphic Init	<b>inig</b> 8 - 17
<b>8.3 Panel Transparent Mode</b>		8 - 18
8.3.1	Use of the Panel Screen	8 - 18
8.3.1.1	Definition of a Window	8 - 18
8.3.1.2	Definition of the Alphanumeric Space	8 - 18
8.3.1.3	Definition of the Graphic Space	8 - 18
8.3.2	Definition of the Instructions	8 - 22
8.3.2.1	Contents of an Instruction	8 - 22
8.3.2.2	Notation Conventions	8 - 22
8.3.2.3	List of Instructions	8 - 22
8.3.3	General Purpose Instructions	8 - 23
8.3.3.1	Software Initialisation	8 - 23
8.3.3.2	Selection of a Colour	8 - 24
8.3.3.3	Window Selection	8 - 24
8.3.4	Alphanumeric Characters and Instructions	8 - 25
8.3.4.1	Alphanumeric Characters	8 - 25
8.3.4.2	Choice of Font Format	8 - 26
8.3.4.3	Character Display	8 - 27
8.3.4.4	Cursor Display	8 - 28
8.3.4.5	Cursor Movement	8 - 28
8.3.4.6	Deletion	8 - 29

8.3.5	Graphic Instructions	8 - 29
8.3.5.1	Definition of the User Reference System	8 - 29
8.3.5.2	Draw in User Reference System	8 - 31
8.3.5.3	User Drawing	8 - 32
8.3.5.4	Tool Definition	8 - 33
8.3.5.5	Animation	8 - 34
8.3.5.6	No Animation	8 - 34
8.3.5.7	Screen Drawing	8 - 34
8.3.5.8	Shift Screen Origin	8 - 35
8.3.5.9	Transfer Current Point	8 - 35
8.3.5.10	Icons	8 - 36
8.3.5.11	Screen Reference System Character String	8 - 38
8.3.5.12	User Reference System Character String	8 - 38
8.3.5.13	Filling in User Area	8 - 38
8.3.5.14	Filling in Screen Area	8 - 40
8.3.5.15	Draw Key Bar	8 - 40

## 8.1 Introduction

The programmes in transparent mode must be executed when variable %R5.7 equals 1. This variable must be used as a pre-condition for starting such programmes.

To access the «TRANSPARENT MODE» page, refer to the «OPERATOR MANUAL».

### 8.1.1 Display Management

In transparent mode, the CNC processor does not control the operator panel screen. The machine processor can then use the screen to display alphanumeric characters or draw graphics.

The cursor control commands, alphanumeric characters and graphic instructions are associated with hexadecimal codes.

The screen/keyboard management functions are valid only in transparent mode.

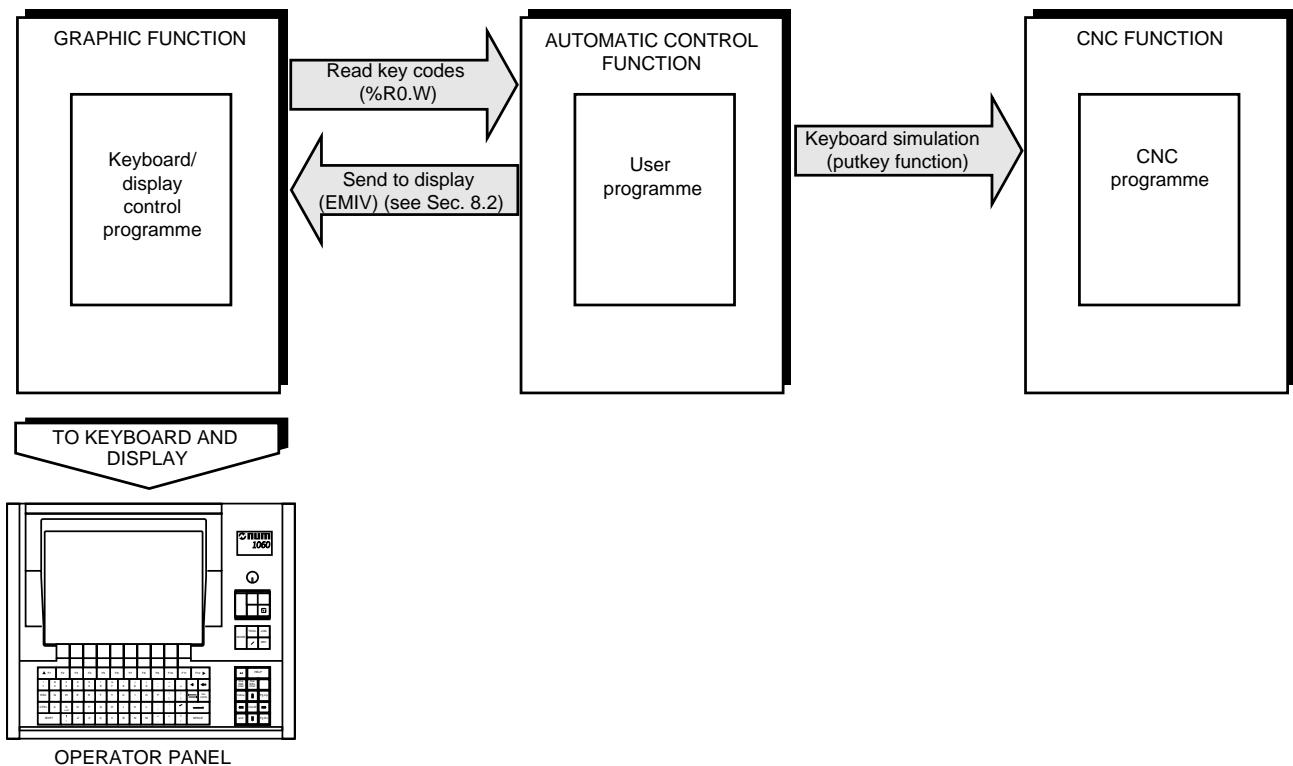


Figure 8.1 - Principle of the transparent mode

#### **CAUTION**

Setting to transparent mode is possible only with the graphic pages including the basic softkeys. To display these keys, it is necessary to add sending of code \$8D by putkey, which corresponds to the << key (in Tool or Jog mode).

### 8.1.2 Exchange Variable

Variable %R0.W «CARCLAV» is used to read the key codes sent by the operator panel keyboard at the rate of one character every 5 RTCs and process them via the user programme.

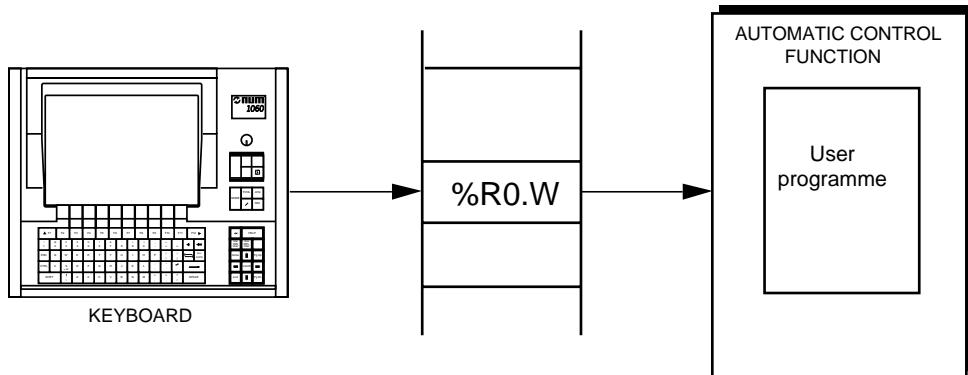


Figure 8.2 - Exchange variable %R0.W

If the operator panel keyboard is disabled (variable %W5.0 = 1) function putkey() can be used to simulate the operator panel keyboard via the user programme.

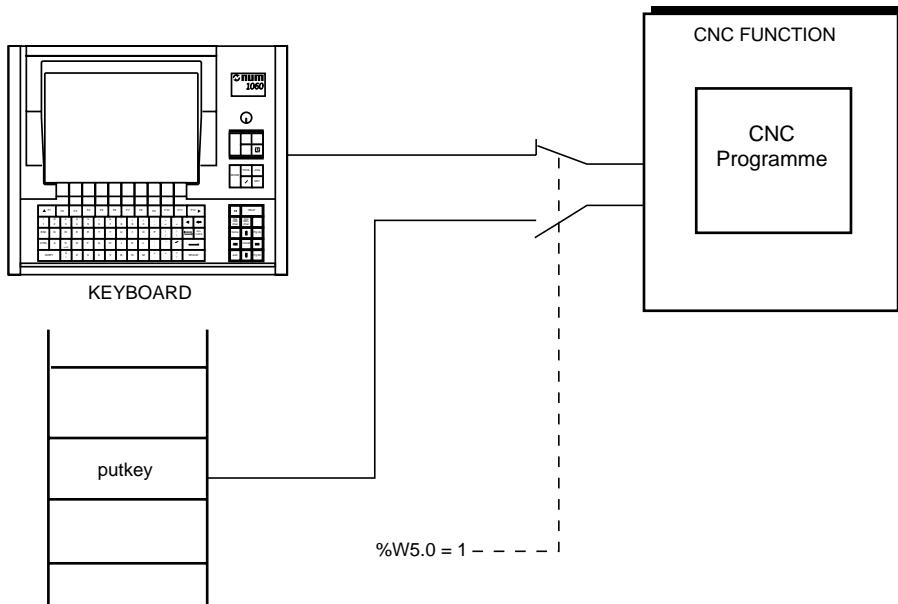


Figure 8.3 - Function putkey()

### 8.1.3 Transmission of Character Codes to the Screen

Functions putchar(), puts(), print(), printf() (see Sec. 8.2), are used to send the cursor control commands and alphanumeric characters to the operator panel screen.

### 8.1.4 Character Codes Used by %R0.W and putkey()

All the characters are read in variable %R0.W. Function putkey() simulates only the dialogue part.

HEX CODE	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	Ctrl P (DLE)	SP	0	@	P	'	p							0,1	AUTO	
1	Ctrl A	Ctrl Q (Xon)	!	1	A	Q	a	q	F2		MODE	Shift F2	Shift	1	SINGLE	
2	Ctrl B	Ctrl R (Xoff)	"	2	B	R	b	r	F3			Shift F3	Shift	TL COMP	10	MDI
3	Ctrl C	Ctrl S (Xoff)	#	3	C	S	c	s	F4		TOOL	Shift F4	Shift	100	DRYRUN	
4	Ctrl D	Ctrl T	\$	4	D	T	d	t	F5	/	Shift F5	Shift	Shift	1000	SEARCH	
5	Ctrl E	Ctrl U	%	5	E	U	e	u	F6	HOME	JOG	Shift F6	Shift	10 000	EDIT	
6	Ctrl F	Ctrl V	&	6	F	V	f	v	F7	END	M01	Shift F7	Shift	WEAR+	CONT.	TEST
7	Ctrl G	Ctrl W	'	7	G	W	g	w	F8	Pg Up	//	Shift F8	Shift	L or X	MANU	
8	Ctrl H ←	Ctrl X	(	8	H	X	h	x	F9	Pg Dn	NU_CN	Shift F9	Shift	R or Z	HANDWH	HOME
9	Ctrl I	Ctrl Y	)	9	I	Y	i	y	F10	Ins/Over (I )	Shift F10	Shift	Shift	WEAR0	SHIFT 0.01	
A	Ctrl J LF	Ctrl Z	*	:	J	Z	j	z	F11	Del car		Shift F11	Del line		TLSET	
B	Ctrl K Ctrl [	ESC	+	;	K	[	k	{	F1 s	NU_EDT	MACHI NING	Shift				
C	Ctrl L	Ctrl \	,	<	L	\			F12 ►	VALID	PRESET LF	Shift	VALID			
D	Ctrl M ← CR	Ctrl ]	-	=	M	]	m	}	◀ ◀	HELP PROGRAM EDIT					LOAD	
E	Ctrl N →	.	>	N	^	n	~			SPLIT						
F	Ctrl O →	/	?	O	-	o	←			Call to transparent mode					UNLOAD	

Do not correspond to keyboard keys. The codes are sent by the menu manager.

## Compact panel

HEX CODE	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0																
1	Ctrl A	Ctrl Q														
2	Ctrl B	Ctrl R														
3	Ctrl C	Ctrl S														
4	Ctrl D	Ctrl T														
5	Ctrl E	Ctrl U														
6	Ctrl F	Ctrl V														
7	Ctrl G	Ctrl W														
8	Ctrl H	Ctrl X	←													
9	Ctrl I	Ctrl Y														
A	Ctrl J	Ctrl Z	LF													
B	Ctrl K	Ctrl [														
C	Ctrl L	Ctrl \									F13	●				
D	Ctrl M	Ctrl ]	CR	←							F14	●				
E	Ctrl N										F15	●	●			
F	Ctrl O										F16	●	●			

### REMARK

*These key codes are accessible by the PLC spy but simulation of these codes is not taken into account.*

## 8.2 Functions Assigned to transparent Mode

 **CAUTION**

These functions are valid only in transparent mode.

They can only be programmed in a %TF task.

### 8.2.1 Position the Cursor

**pcur**

#### Syntax

**pcur(line, column)**

line: Line number.

column: Column number.

#### Operation

Positions the cursor on the line and column specified.

#### Return code

##### If OK

0

##### Error

-1: Not in transparent mode, the calling task is not a %TF task.

-2: Attempt to position the cursor outside the screen.

### 8.2.2 Display a Character

**putchar**

#### Syntax

**putchar(character)**

character: ASCII code of a character.

#### Operation

Sends a character to the system screen.

**REMARK** If the screen saver is active, the putchar command is placed on wait. To resume display in transparent mode, it is necessary to deactivate the screen saver by variable %W5.7.

**Return code**If OK

Returns the character written.

Error

- 1: Not in transparent mode, the calling task is not a %TF task.

**8.2.3 Display a String without Formatting****puts****Syntax****puts(&string)**

&string: Address of the ASCII string (ending with a ZERO byte) to be displayed.

**Operation**

Sends a string to the system screen. The system adds 0x0D9C (\n) at the end of the string.

**Return code**If OK

Returns the number of characters transmitted.

Error

- 1: Not in transparent mode, the calling task is not a %TF task.
- 2: Max. buffer size (512 bytes) exceeded.

**Programming error causing a CPU fault**

Access to a prohibited address:

- &string parameter error,
- end of string outside authorised area.

**8.2.4 Display a Buffer****print****Syntax****print(&source, n)**

&source: Address of the buffer to be displayed.

n: Number of bytes to be displayed.

## Operation

Sends a buffer of bytes to the system screen (the buffer can contain graphic commands).

The display stops according to the value of n.

If  $n == 0$ : The display stops on the first ZERO byte (ZERO byte not displayed).

If  $n > 0$ : The display stops after n bytes.

## Return code

### If OK

Returns the number of characters transmitted.

### Error

-1: Not in transparent mode, the calling task is not a %TF task.

-2: Max. buffer size (512 bytes) exceeded.

## Programming error causing a CPU fault

Access to a prohibited address:

- &source parameter error,
- &source+n outside authorised area.

## 8.2.5 Display a Character String with Formatting

**printf**

### Syntax

**printf(&format {, argn}6)**

**&format** Format string address.

**argn:** Possible argument.

Displays a string, converting any arguments (the string must not contain graphic commands).

Function printf() supports ANSI standard C language conversion specifications.

## Operation

The format string contains displayable characters and possibly specifications for argument conversions.

Function printf() analyses the characters of the format string.

If the character is displayable, printf() copies it into a work buffer.

When printf() detects the character %, it analyses the following characters which indicate the conversion required on the corresponding argument. The displayable characters resulting from conversion of the argument are stored in the work buffer.

When printf() detects the end of the format string (ZERO byte), it sends the buffer to the task handling display on the CNC screen.

## Format of the Conversion Specifications

%[flags][numbers].[numbers]][conversion\_letter]

%:	Indicates the start of a conversion specification.
[flags]:	The following optional characters: -: Indicates that the conversion result must be scaled left in the reserved field. +: Indicates that the result of a signed conversion must begin with a + or a - sign. «space»: Indicates that the result of a signed conversion must begin with a space character. This flag is ignored if the + flag is present. #: Indicates that the conversion result must be modified as follows: o: Conversion: The result must begin with a zero. x or X conversion: The result must begin with 0x or 0X. 0: Indicates that the leading zeros of the result must be displayed.
[numbers]:	Optional ASCII decimal characters indicating the minimum size of the field used to display the conversion result.
[.numbers]:	Optional ASCII decimal characters indicating the minimum number of digits to be displayed in the case of a d, o, u, x or X conversion or the maximum number of characters of the string to be displayed in the case of an s conversion.
conversion_letter:	Compulsory letter indicating the conversion to be performed on the argument. d: The argument is displayed as a signed decimal value. o: The argument is displayed as an octal value. x: The argument is displayed as a hexadecimal value using letters «abcdef». X: The argument is displayed as a hexadecimal value using letters «ABCDEF». u: The argument is displayed as an unsigned decimal value. c: The argument is taken as the code of an ASCII character and displayed without conversion. s: The argument is a pointer to a string displayed without conversion. %: % is used to display the character %.

**Example 1**Consider the following variables

```
%V100.L = «Tool number:%5d Type:%2c%2c Time of use:      %2d hours %2d minutes»
%M50.W = 255
%M52.B = 0x55          (0x55 ASCII code for t, 0x57 ASCII code for v)
%M54.B = 2
%M55.B = 57
```

The instruction printf(%V100.L, %M50.W, %M52.B, 0x57, %M54.B, %M55.B) displays:

Tool number: 255 Type: tv Time of use: 2 hours 57 minutes

**Example 2**Displaying a Simple String

```
%V200.L =           «Do you want to know the time? (Y/N)»
printf(%V200.L)    displays: Do you want to know the time? (Y/N)
```

Displaying a String with a Display Format for the Arguments

```
If %M10.B = 3;
If %M11.B = 15;
%V200.L =           «It is %2d:%2d»
printf(%V200.L, %M10.B, %M11.B)  displays: It is 3:15
```

The two characters \n cause a jump to the next line when displaying the string (the compiler replaces the characters \n by bytes 0xd 0xa).

**Example 3**

```
%V200.L = «1 - Read \n 2 - Write»
```

printf(%V200.L) displays

1 - Read  
2 - Write

**Return code**If OK

Number of characters transmitted for display.

Error

- 1: Not in transparent mode, the calling task is not a %TF task.
- 2: Max. formatting buffer size (255 bytes) exceeded.
- 3: Format error in the format string.

## Programming error causing a CPU fault

Access to a prohibited address:

- &format parameter error,
- end of string outside authorised area.

### 8.2.6 Open a Keyboard Acquisition

**scano**

#### Syntax

```
scano(&question, width)
```

&question: Address of a character string (ending with a ZERO byte).

width: Maximum width of the entry field.

Opens a keyboard acquisition.

#### Operation

The system displays the string pointed to by &question at the bottom of the screen and opens the dialogue after the string.

If parameter &question == 0, no string is displayed.

Character entry is controlled by the system line editor.

The line editor checks that the number of characters entered is less than «width».

The line editor commands are the conventional commands of the part programme editor:

- cursor movement left and right, start and end of line,
- character insertion and deletion,
- entry is ended by the line feed key.

#### CAUTION

The putchar() and printf() display functions are prohibited during a keyboard acquisition.

#### Return code

##### If OK

0

##### Error

- 1: Not in transparent mode, the calling task is not a %TF task.
- 2: Resource already in use (a keyboard acquisition is already in progress).

## Programming error causing a CPU fault

Access to a prohibited address:

- &question parameter error,
- end of string outside authorised area.

**8.2.7 Open a Numerical Keypad Acquisition****scetu****Syntax****scetu(&question, width)**

**&question:** Address of a character string (ending with a ZERO byte).

**width:** Maximum width of the entry field.

Opens a keypad acquisition.

**Operation**

Scetu() operates in the same way as scano() except that the line editor inhibits entry of characters others than decimal digits (0, 1 ... 9).

 **CAUTION**

This function can only be used with the 12 line x 40 column font. Entry is on the 11th line, with deletion of the line just above.

**Return code**If OK

0

Error

-1: Not in transparent mode, the calling task is not a %TF task.

-2: Resource already in use (a keyboard acquisition is already in progress).

**Programming error causing a CPU fault**

Access to a prohibited address:

- &question parameter error,
- end of string outside authorised area.

**8.2.8 Read a String****scans****Syntax****scans(&dest)**

**&dest:** Address of the memory area (%M or %V) where the characters entered from the keyboard will be stored.

Reads a keyboard entry. This function must be called after a scano() or scetu() dialogue opening function.

## Operation

This function is used to receive the string entered at the end of an operator dialogue.

The system ends the string with a ZERO byte.

If the dialogue is still in progress (line feed key not pressed), the code 0 is returned. It is therefore necessary to call scans() cyclically until the end of the dialogue.

## Return code

### If OK

0: Dialogue in progress.

n > 0: Number of characters transferred into &dest (the dialogue is finished).

### Error

-1: Not in transparent mode, the calling task is not a %TF task.

-2: No dialogue in progress.

## Programming error causing a CPU fault

Access to a prohibited address:

- &dest parameter error,
- end of acquisition field outside authorised error.

## 8.2.9 Read and Convert a Decimal Number

**scand**

### Syntax

**scand(&lvariable)**

&lvariable: Address of an .L variable (e.g. %V100.L) where the ASCII -> signed integer conversion of the string entered from the keyboard will be loaded.

Reads and converts a decimal value entered from the keyboard. This function must be called after a scano() or scanu() dialogue opening function.

The conversion stops on the first non-decimal character. If no decimal characters are detected, %lvariable is loaded with 0.

## Operation

This function is used to receive the value of a decimal number at the end of an operator dialogue.

If the dialogue is still in progress (line feed key not pressed), the code 0 is returned. It is therefore necessary to call scand() cyclically until the end of the dialogue.

**Return code**If OK

- 0: Dialogue in progress.
- 1: Number successfully read and converted. The result is transferred into variable .L pointed to by &variable (the dialogue is finished).  
The conversion stops on the first non-decimal character.

Error

- 1: Not in transparent mode, the calling task is not a %TF task.
- 2: No dialogue in progress.

**Programming error causing a CPU fault**

Access to a prohibited address:  
- &variable parameter error.

**8.2.10 Read and Convert a Hexadecimal Number****scanx****Syntax****scanx(&variable)**

&variable: Address of an .L variable (e.g. %V100.L) where the ASCII -> signed integer conversion of the string entered from the keyboard will be loaded.

Reads and converts a hexadecimal value entered from the keyboard. This function must be called after a scan() or scanu() dialogue opening function.

The conversion stops on the first non-hexadecimal character. If no hexadecimal characters are detected, %variable is loaded with 0.

**Operation**

This function is used to receive the value of a hexadecimal number at the end of an operator dialogue.

If the dialogue is still in progress (line feed key not pressed), the code 0 is returned. It is therefore necessary to call scanx() cyclically until the end of the dialogue.

**Return code**If OK

- 0: Dialogue in progress.
- 1: Number successfully read and converted. The result is transferred into variable .L pointed to by &variable (the dialogue is finished).  
The conversion stops on the first non-hexadecimal character.

### Error

- 1: Not in transparent mode, the calling task is not a %TF task.
- 2: No dialogue in progress.

### **Programming error causing a CPU fault**

Access to a prohibited address:

- &variable parameter error.

## **8.2.11 Close a Keyboard Acquisition**

**scanc**

### **Syntax**

```
scanc()
```

### **Operation**

This function cancels a dialogue in progress (started by function scano() or scanu()).

### **Return code**

#### If OK

0

#### Error

- 1: Not in transparent mode, the calling task is not a %TF task.
- 2: No dialogue in progress.

## **8.2.12 Position and Display an Image**

**putimage**

### **Syntax**

```
putimage(x, y, &image, n)
```

- x: Start x value.
- y: Start y value.
- &image: Address of a graphic command buffer (0x9b...).
- n: Number of bytes to be sent.

Sends a buffer containing graphic commands with prior positioning of the cursor at x,y. The display stops according to the value of n.

## Operation

`putimage()` operates like `print()` but with prior positioning in (x, y).

`putimage()` is used to duplicate the same image with different (x, y) values.

If `n == 0`: The display stops on the first ZERO byte (ZERO byte not displayed).

If `n > 0`: The display stops after n bytes.

## Return code

### If OK

Returns the number of characters transmitted.

### Error

-1: Not in transparent mode, the calling task is not a %TF task.

-2: Max. buffer size (512 bytes) exceeded, attempt to position the cursor off the screen.

## Programming error causing a CPU fault

Access to a prohibited address:

- &image parameter error,
- &image+n outside authorised area.

## 8.2.13 Graphic Init

inig

8

### Syntax

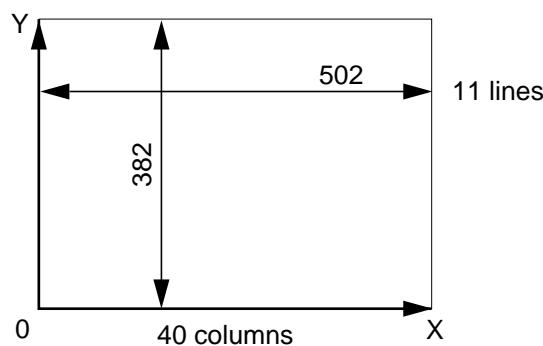
```
inig(..)
```

### Description

Initialises the graphic display and defines the reference system. Programming is in pixels.

The reference system is defined as follows:

- the X axis from 0 to 502 pixels
- the Y axis from 0 to 382 pixels.



## 8.3 Panel Transparent Mode

### 8.3.1 Use of the Panel Screen

The screen, with a definition of 640 x 480 pixels, is divided into four windows. A communication channel and a context are associated with each window. The display manager uses all the channels and saves the contexts.

*REMARK Each window is considered as a screen.*

#### 8.3.1.1 Definition of a Window

A window is defined by its size and its position in the screen.

For each window, there is an alphanumeric space and a graphic space which have their own context (colour, font, etc.). These spaces are positioned in the displayable area of the window.

*REMARKS The windows overlap.  
The spaces are overlaid.*

#### 8.3.1.2 Definition of the Alphanumeric Space

This space is used to display ASCII characters (in pixels) on lines and columns and to control the cursor.

*REMARK Display of an alphanumeric character overwrites the element previously displayed, whatever its space.*

#### 8.3.1.3 Definition of the Graphic Space

This space is used to display texts (in pixels) and graphic data.

Two reference systems (screen and user) are available. It is possible to combine the two reference systems in the same space.

*REMARK An element is displayed in the graphic space overlays the element previously displayed, whatever its space.*

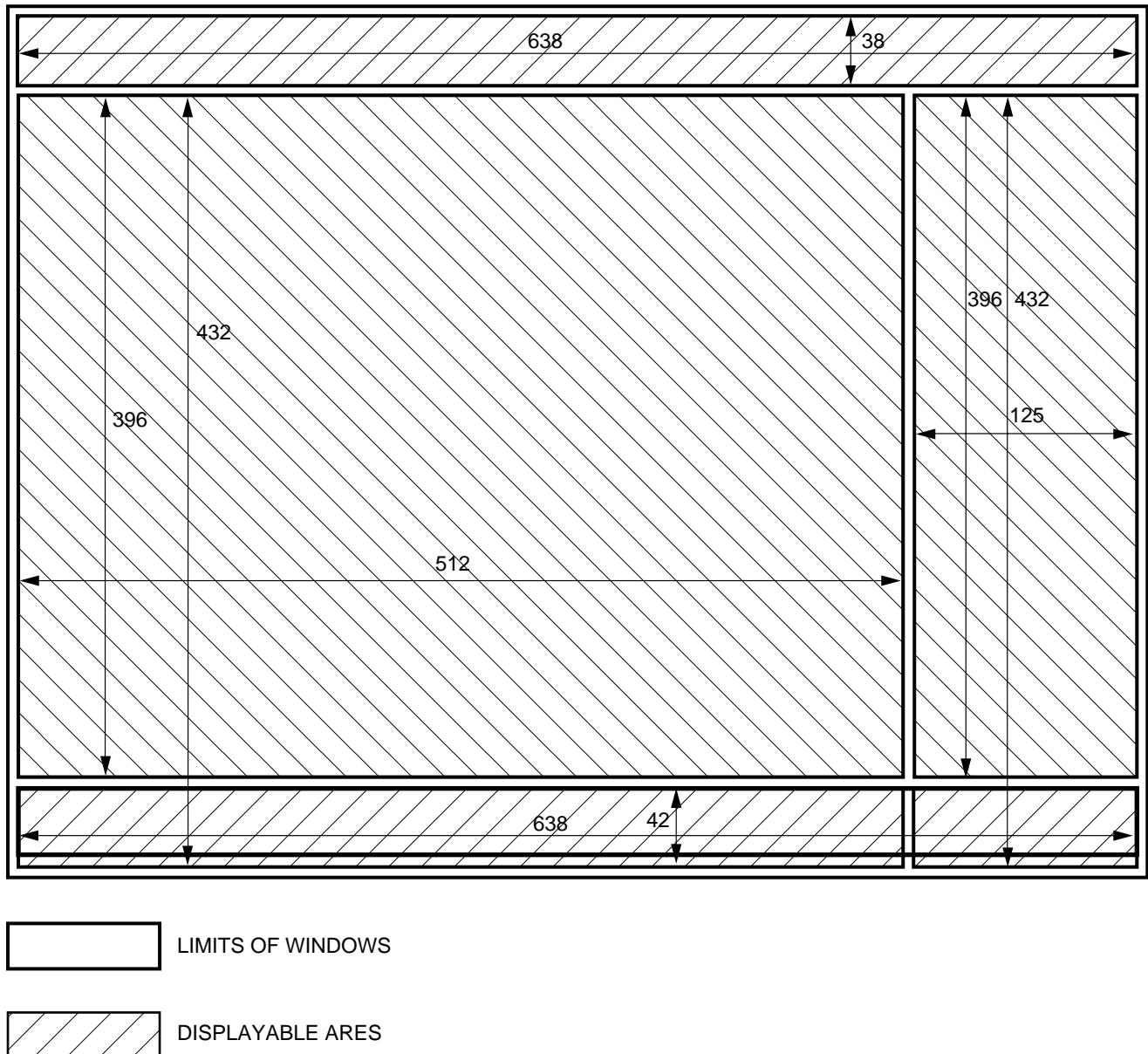


Figure 8.4 - Dimension of the windows

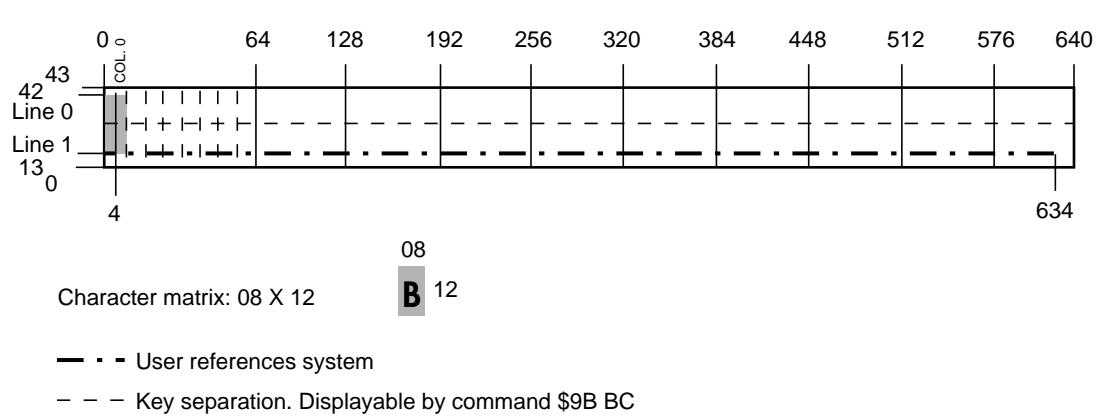
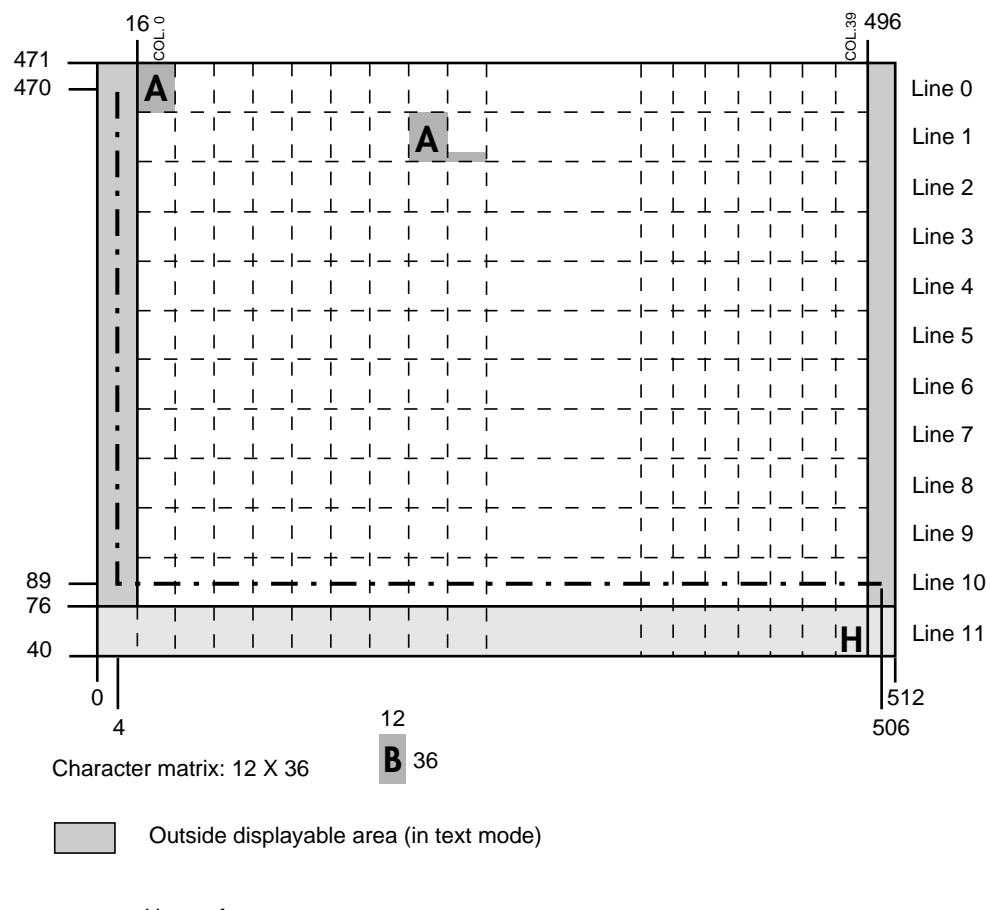


Figure 8.5 - Positioning of the main «window and message window spaces»

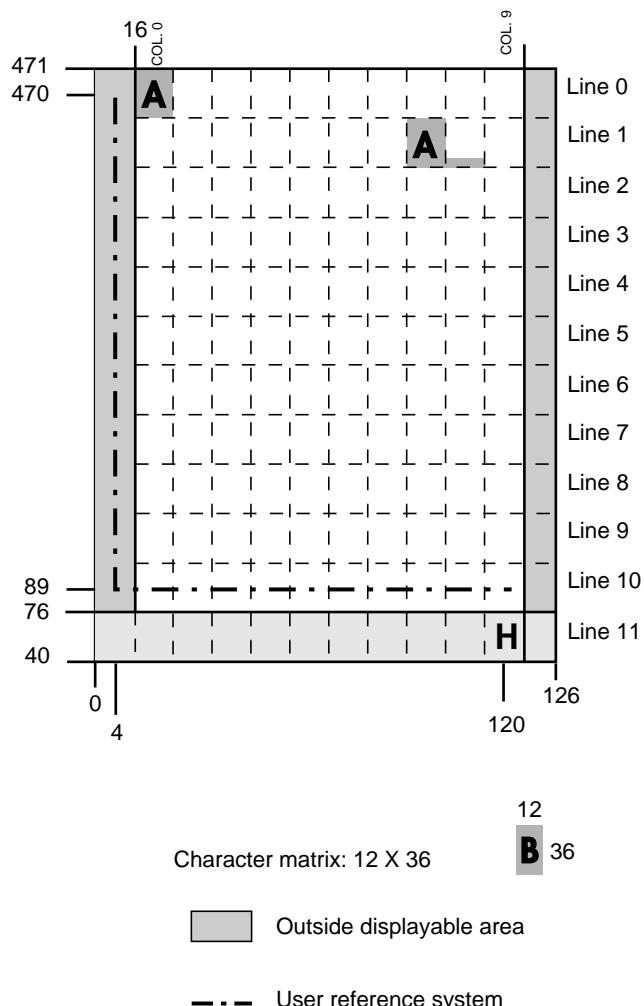


Figure 8.6 - Positioning of side «window spaces»

### 8.3.2 Definition of the Instructions

#### ! CAUTION

All the instructions can be processed in the main window, the message window and the side window.

The instruction must be programmed in a %TF task.

#### 8.3.2.1 Contents of an Instruction

An instruction consists of an operation code and may be followed by expressions.

An expression may be a sequence of expressions or arguments.

An argument is an ASCII character.

Instructions end with character «LF».

#### 8.3.2.2 Notation Conventions

Expressions are in capital letters and arguments are in quotes.

Notation	Definition
XX YY ZZ	Mandatory order of the expressions
{XX YY ZZ}	Expressions in any order
XX YY ZZ	Only one of the expressions is necessary
(XX) ...	Expression that can be repeated several times
[YY]	Optional expression
Default value	Default value for optional expressions
LF	Instruction end character (hex code 0x8A)
«0»	ASCII character 0 (hex code 0xB0)

#### 8.3.2.3 List of Instructions

Description	Instruction	See
Soft instruction	0x9BDD	8.3.3.1
Colour selection	0x9BBD	8.3.3.2
Window selection	0x9B2D	8.3.3.3
Normal character	0x9BC8	8.3.4.3
Highlighted character	0x9BC9	8.3.4.3
Character not underlined	0x9BCA	8.3.4.3
Character underlined	0x9BCB	8.3.4.3
Flashing colour selection	0x9BBD	8.3.4.3
Expanded character	0x9BDB	8.3.4.3
Cursor in any position	0x9BBF	8.3.4.5
Definition of the user reference system	0x9BB0	8.3.5.1
Draw the user reference system	0x9BD8	8.3.5.2

Description	Instruction	See
User drawing	0x9BB2	8.3.5.3
Tool definition	0x9BB1	8.3.5.4
Animation	0x9BDF 0x9BE7	8.3.5.5
No animation	0x9BDE	8.3.5.6
Draw screen	0x9BB6	8.3.5.7
Shift screen origin	0x9BB7	8.3.5.8
Transfer current point	0x9BE4	8.3.5.9
Icons	0x9BB4	8.3.5.10
Screen reference system character string	0x9BA8	8.3.5.11
User reference system character string	0x9B98	8.3.5.12
Fill in user area	0x9BA9 0x9BAA	8.3.5.13
Fill in screen area	0x9BAB 0x9BAC	8.3.5.14
Draw key bar	0x9BBC	8.3.5.15

### 8.3.3 General Purpose Instructions

#### 8.3.3.1 Software Initialisation

Instruction 0x9BDD rapidly initialises the display.

#### Syntax

**0x9BDD**

This instruction initialises the alphanumeric and graphic spaces and deletes the user reference system.

In the graphic space, it:

- clears the screen,
- clears saved areas,
- sets the default hexadecimal coordinate,
- sets the default decimal coordinate,
- set white as default colour.

In the alphanumeric space, it:

- selects format A,
- sets the default colour,
- inhibits display of the cursor,
- places the cursor on the first line and first column,
- sets normal video,
- selects no underlining.

### 8.3.3.2 Selection of a Colour

Instruction 0x9BBD selects one colour out of the 16 available.

#### Syntax

#### 0x9BBD COLOUR

COLOUR	Colour code (see Table below)
0	0xB0
1	0xB1
2	0xB2
3	0xB3
4	0xB4
5	0xB5
6	0xB6
7	0xB7
8	0xB8
9	0xB9
10	0xBA
11	0xBB
12	0xBC
13	0xBD
14	0xBE
15	0xBF

Colour code	HEX code	Colour	RGB MIX		
			%R	%G	%B
0	0xB0	Dark blue	0	0	50
1	0xB1	Red	100	0	0
2	0xB2	Blue	24	75	100
3	0xB3	Pink	100	50	100
4	0xB4	Green	0	100	0
5	0xB5	Yellow	100	100	0
6	0xB6	Cyan	0	100	100
7	0xB7	Black	0	0	0
8	0xB8	White	100	100	100
9	0xB9	Brown	75	24	0
10	0xBA	Light blue	50	75	100
11	0xBB	Light grey	75	75	75
12	0xBC	Dark grey	33	33	33
13	0xBD	Orange	100	75	0
14	0xBE	Red/white flashing	100/100	24/100	0/100
15	0xBF	Light grey/white flashing	75/100	75/100	75/100

### 8.3.3.3 Window Selection

Instruction 0x9B2D selects the window accessible for programming. This instruction is modal.

#### Syntax

#### 0x9B2D NUMBER

NUMBER:   «0x1», «0x3», «0x4»	Default value: «0x1» (main window)
«0x1»	Main window
«0x4»	Message window
«0x3»	Side window

HEX code	Window type
«0x1»	Main window
«0x4»	Message window
«0x3»	Side window

### 8.3.4 Alphanumeric Characters and Instructions

#### 8.3.4.1 Alphanumeric Characters

HEX code	0	1	2	3	4	5	6	7
0			SP	0	@	P	`	p
1			!	1	A	Q	a	q
2	CHARACTER NOT FLASHING	[	"	2	B	R	b	r
3		]	#	3	C	S	c	s
4	CURSOR FLASHING	-	\$	4	D	T	d	t
5	CURSOR STEADY	_	%	5	E	U	e	u
6	CURSOR NOT VISIBLE	_	&	6	F	V	f	v
7	CHARACTER FLASHING		'	7	G	W	g	w
8	MOVE CURSOR RIGHT	H	(	8	H	X	h	x
9	MOVE CURSOR LEFT	→	)	9	I	Y	i	y
A	MOVE CURSOR DOWN (LF)	←	*	:	J	Z	j	z
B	MOVE CURSOR UP		+	;	K	[	k	{
C	HOME	CLEAR WINDOW	,	<	L	\	l	
D	MOVE CURSOR TO START OF LINE (CR)	FORMAT A	-	=	M	]	m	}
E	DELETE TO END OF LINE	FORMAT B	.	>	N	^	n	~
F	DELETE TO END OF PAGE	FORMAT C	/	?	O	-	o	

**REMARK** Character codes 0x10 to 0x18 can only be used in format A in the main and side windows. Character codes 0x19 and 0x1A can only be used in format D in the main and side windows.

#### 8.3.4.2 Choice of Font Format

Selection of a new format causes:

- clearing of the previous cursor,
- display of the new cursor with the previous attributes (lit steady, flashing, not visible).

#### Format A

##### 0x9D

	Font size	MAX. display authorised
Main window	12x36	12 lines x 40 characters (last line not in displayable area)
Message window	16x24	1 line x 40 characters
Side window	12x36	12 lines x 10 characters (Last line not in displayable area)

#### Format B

##### 0x9E

	Font size	MAX. display authorised
Main window	06x18	24 lines x 80 characters (last two lines not in displayable area)
Message window	08x12	2 lines x 80 characters
Side window	06x18	24 lines x 20 characters (Last two lines not in displayable area)

#### Format C

##### 0x9F

	Font size	MAX. display authorised
Main window	24x56	7 lines x 20 characters (last line not in displayable area)
Message window	9x12	2 lines x 71 characters
Side window	24x56	7 lines x 5 characters (Last lines not in displayable window)

**Format D (Expanded characters)**

0x9BDB		
	Font size	MAX. display authorised
Main window	12x18	24 lines x 40 characters (the last two lines are not in the displayable area)
Message window	16x12	02 lines x 40 characters
Side window	12x18	24 lines x 10 characters (the last two lines are not in the displayable area)

**8.3.4.3 Character Display**

These instructions are modal and valid regardless of the format selected.

**Normal character**

0x9BC8

**Highlighted character**

0x9BC9

**Character not underlined**

0x9BCA

**Character underlined**

0x9BCB

**Flashing colour selection**

0x9BBD COLOUR

**COLOUR**

Colour code (see Sec. 8.3.3)

Flashing of the characters is produced by colour codes 14 and 15.

#### 8.3.4.4 Cursor Display

**Cursor steady**

0x85

**Cursor not visible**

0x86

#### 8.3.4.5 Cursor Movement

**Move one character right**

0x88

**Move one character left**

0x89

**Move one character down**

0x8A

**Move one character up**

0x8B

**Move home**

0x8C

**Move to start of line**

0x8D

**Move to a specific position**

0x9BBF LINE COLUMN

#### Description

«LINE» and «COLUMN» are defined by two hex codes.

<b>LINE</b>	Code for real position + 0x20 = Value to be programmed
<b>COLUMN</b>	Code for real position + 0x20 = Value to be programmed

**Example**

To position the cursor on LINE 2 COLUMN 34

LINE 2 (third line):  $0x2 + 0x20 = 0x22$

COLUMN 34 (35th column):  $0x22 + 0x20 = 0x42$

Command to be programmed: 0x9BBF 0x22 0x42

**8.3.4.6 Deletion**

The deletion instruction can be used in any space and format.

**Delete to end of line**

**0x8E**

**Delete to end of page**

**0x8F**

**Clear page**

**0x9C**

**8.3.5 Graphic Instructions****8.3.5.1 Definition of the User Reference System**

Instruction 0x9BB0 allows the user to define his own reference system and the applicable display characteristics (colour, legend, etc.).

*REMARK The limits are recomputed to obtain the same conversion factor on both axes.*

**Syntax**

**0x9BB0 AXIS<sub>1</sub> AXIS<sub>2</sub> AXIS<sub>3</sub> AXIS<sub>4</sub> { [FORMAT] [LINE] [COLOUR] } LF**

**AXIS<sub>1</sub> : NAME [SIGN] VALUE** Name of the horizontal axis and value of the left limit.

**NAME** Name of the axis  
Defined by characters «A» to «Z», upper or lower case (generally X and Y).

**SIGN** Sign of the limit value.  
Algebraic «+» or «-» sign.  
Default value: «+».

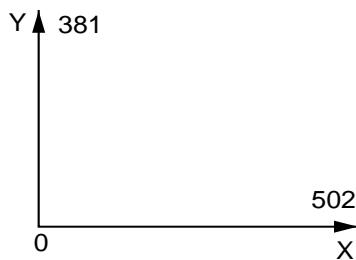
**VALUE** Value of the axis limit. (Decimal value in pixels).

<b>AXIS<sub>2</sub></b>	Name of the horizontal axis and value of the right limit.
<b>REMARK</b>	<i>AXIS<sub>2</sub> must have the same name as AXIS<sub>1</sub>.</i>
<b>AXIS<sub>3</sub></b>	Name of the vertical axis and value of the low limit.
<b>REMARK</b>	<i>Same syntax as AXIS<sub>1</sub>.</i>
<b>AXIS<sub>4</sub></b>	Name of the vertical axis and value of the high limit.
<b>REMARK</b>	<i>AXIS<sub>4</sub> must have the same name as AXIS<sub>3</sub>.</i>
<b>FORMAT: «.» VALUE</b>	Format of the decimal numbers for display of the reference system limits. (e.g.: If format = .3, the value 10000 is displayed as 10.000).
<b>VALUE</b>	Number of decimal digits. This argument is decimal. Default value: 0.
<b>LINE : «M»   «1», «2», «3», «4», «5»  </b>	Characteristic of the line used to draw the axes. Does not modify the current type of line. Default value: «1» (solid line).
<hr/>	
<b>COLOUR: «C» VALUE</b>	Colour of the axes. Does not modify the current colour.
<b>VALUE</b>	Colour code (see Sec. 8.3.3). Expressed in decimal or hexadecimal. Default value: current colour at the time of drawing.

### Examples

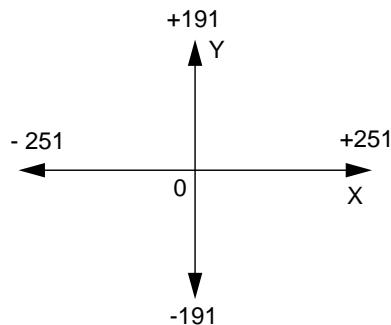
Definition of a Reference System Chosen by Function inig(..)

0x9BB0 X0 X502 Y0 Y381 (LF)

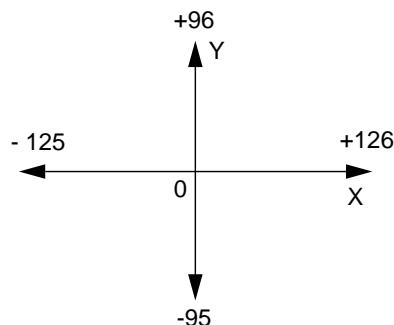


Definition of a Reference System Centred with Scale 1

0x9BB0 X-251 X251 Y-191 Y191 (LF)

Definition of a Reference System Centred with Scale 2

0x9BB0 X-125 X126 Y-95 Y96 (LF)

**8.3.5.2 Draw in User Reference System**

Instruction 0x9BD8 is used to display the user reference system. The coordinate indications are defined by the arguments of instruction 0x9BB0.

**Syntax**

0x9BD8

### 8.3.5.3 User Drawing

Instruction 0x9BB2 is used to draw a line or an arc in the user reference system. Drawing is with or without display of the tool (see instructions 0x9BB1, 0x9BDE, 0x9BDF).

#### Syntax

**0x9BB2 { [LINCIR] [DECHEX] } { [LINE] [STYLUS] [X] [Y] [I] [J] } LF**

**LINCIR : «G» | «1» «2» «3» |**

Defines the type of drawing (nonmodal instruction).  
Default value: line drawing.

Character	Type of drawing	HEX code
«1»	Line	0xB1
«2»	Clockwise arc	0xB2
«3»	Anticlockwise arc	0xB3

**DECHEX: «G» | «4» «5» |**

Current coordinate type (modal instruction).

Character	Coordinate type	HEX code
«4»	Decimal	0xB4
«5»	Hexadecimal	0xB5

**REMARK** This command applies to drawing of the screen reference system (9BB6).

**LINE: «M» | «1», «2», «3», «4», «5» |**

Characteristic of the line used for drawing (see Sec. 8.3.5.1).

**STYLUS: «M» | «6» «7» «10» |**

Current stylus type (modal instruction).

Character	Current stylus type	HEX code
«6»	Stylus (The colours are added)	0xB6
«7»	Eraser (Colour 0 is forced)	0xB7
«10»	Stylus (The requested colour is forced)	0xB1B0

**X: «X» [VALUE]**

Movement on the horizontal axis;  
Default value: no movement on this axis.

**VALUE**

X coordinate of the end point.  
Default value: «0».

**Y: «Y» [VALUE]**

Movement on the vertical axis.  
Default value: no movement on this axis.

**VALUE**

Y coordinate of the end point.  
Default value: «0».

**I: «I» [VALUE]**

Current X coordinate of the centre (modal instruction).

**VALUE**

X coordinate of the centre.  
Default value: «0».

**J: «J» [VALUE]** Current Y coordinate of the centre (modal instruction)

**VALUE** Y coordinate of the centre.  
Default value: «0».

#### 8.3.5.4 Tool Definition

Instruction 0x9BB1 is used to define the tool used for animation.

##### Syntax

**0x9BB1 | CROSS HAIRS INSERT CUTTER | LF**

**CROSS HAIRS: «R» VALUE [COLOUR]** Definition of a cross hair shape.

**VALUE** Dimension of one hair in the user reference system. Decimal value.  
**COLOUR: «C» VALUE** Colour of the tool (see Sec. 8.3.3.2).  
Default value: 8.

**INSERT: «P» VALUE [COLOUR]** Definition of a tool with an insert shape.

**VALUE** Tip radius of the insert in the user reference system. Decimal value.  
**COLOUR: «C» VALUE** Colour of the tool (see Sec. 8.3.3.2).  
Default value: 8.

**CUTTER: «F» VALUE DIRECTION  
[VALUE] {[HEIGHT] [COLOUR]}** Definition of a milling tool.

**VALUE** Radius of the cutter in the user reference system. Decimal value.  
**DIRECTION: «P» , «Q» , «R» , «S»** Cutter direction.

8

Character	Direction	HEX code
«P»	X increasing	0xD0
«Q»	Y increasing	0xD1
«R»	X decreasing	0xD2
«S»	Y decreasing	0xD3

**VALUE** Torus radius in the user reference system. Decimal value.

Default value: «0».

**HEIGHT: «H» VALUE** Cutter height in the user reference system. Decimal value.

Default value: 4 times the cutter radius.

**COLOUR: «C» VALUE** Colour of the tool (see Sec. 8.3.3.2).

Default value: 8.

<b>TOOL: (MOVEMENT (XY) ...) ...</b>	Definition of a tool of unspecified type.
<b>[COLOUR]</b>	
<b>MOVEMENT: [RAISE XY] LOWER</b>	Movement without drawing.
<b>RAISE: «M5»</b>	Raise stylus.
<b>XY : { X Y }</b>	Coordinates of the start point of the tool in the user reference system. Decimal values.
<b>X</b>	X coordinate of the point.
<b>Y</b>	Y coordinate of the point.
<b>LOWER: «M1»</b>	Lower the stylus.
<b>XY</b>	Decimal coordinates of remaining points of the tool in the user reference system.
<b>COLOUR: «C» VALUE</b>	Colour of the tool (see Sec. 8.3.3.2). Default value: 0xB8.

### 8.3.5.5 Animation

Instructions 0x9BDF and 0x9BE7 select user drawing with display of the tool. The display characteristics are defined by instruction 0x9BB1 (see Sec. 8.3.5.4).

#### Syntax

**0x9BDF or 0x9BE7**

### 8.3.5.6 No Animation

Instruction 0x9BDE selects user drawing without display of the tool.

#### Syntax

**0x9BDE**

### 8.3.5.7 Screen Drawing

Instruction 0x9BB6 is used to draw a line or an arc in the screen reference system.

#### Syntax

**0x9BB6 { [LINCIR] [DECHEX] } { [LINE] [STYLUS] [X] [Y] [I] [J] } LF**

*REMARK The syntax is exactly the same as for instruction 0x9BB2 (see Sec. 8.3.5.3).*

### 8.3.5.8 Shift Screen Origin

Instruction 0x9BB7 is used to shift the origin of the screen reference system.

*REMARK The other limit is recomputed so as not to modify the size of the reference system.*

#### Syntax

```
0x9BB7 [DECHEX] { [X] [Y] } LF
```

#### DECHEX

Selection of decimal or hexadecimal coordinates. Does not modify the current coordinate type.

Default value: decimal.

#### X

Value of the horizontal offset of the screen reference system.

Default value: Preserves the previous horizontal offset..

#### Y

Value of the vertical offset of the screen reference system.

Default value: Preserves the previous vertical offset.

#### Example

To shift the origin of the reference system by 100 pixels on the X axis and 200 pixels on the Y axis

0x9BB7 X100 Y200 (LF)

### 8.3.5.9 Transfer Current Point

Instruction 0x9BE4 is used to align the current point of the screen reference system to the current point of the user reference system.

#### Syntax

```
0x9BE4
```

### 8.3.5.10 Icons

Instruction 0x9BB4 is used to draw an icon (symbol) of constant or parameterisable dimension in a reference frame oriented like the user reference frame whose origin is the user current point.

#### Syntax

```
0x9BB4 NUMBER { [PARAM_SEQUENCE] [LINE] [STYLUS] [COLOUR] } LF
```

**NUMBER:** «N» VALUE                  Icon number.

**REMARK**        *Not all values are significant (see Fig. 8-7).*

**PARAM\_SEQUENCE:** (PARAM)...      Parameters of the icon. The number of parameters is variable depending on the icon number. The order in which the parameters are written is important (P0, P1, P2, ..., Pn).  
Default value: table of parameters in screen coordinates.

**PARAM string:** «P» [VALUE]        String of parameters (from P0 to Pn).

**VALUE**                                  Decimal value of the parameter in the user reference system.  
Default value: «0».

**LINE:** «M» | «1», «2», «3», «4», «5» |    Characteristic of the line used for drawing (see Sec. 8.3.5.1).

**STYLUS:** «M» | «6» «7» «10» |        Current stylus type (see Sec. 8.3.5.3).

**COLOUR:** «C» VALUE                  Icon colour (see Sec. 8.3.3.2).  
Default value: current colour.

#### Example

To draw a red circle with a radius of 20

0x9BB4 N38 P20 M1 M6 C1 (LF)

#### CAUTION

Before drawing icons, it is necessary to define the user reference system (Command 0x9BB0 or function inig(..)).

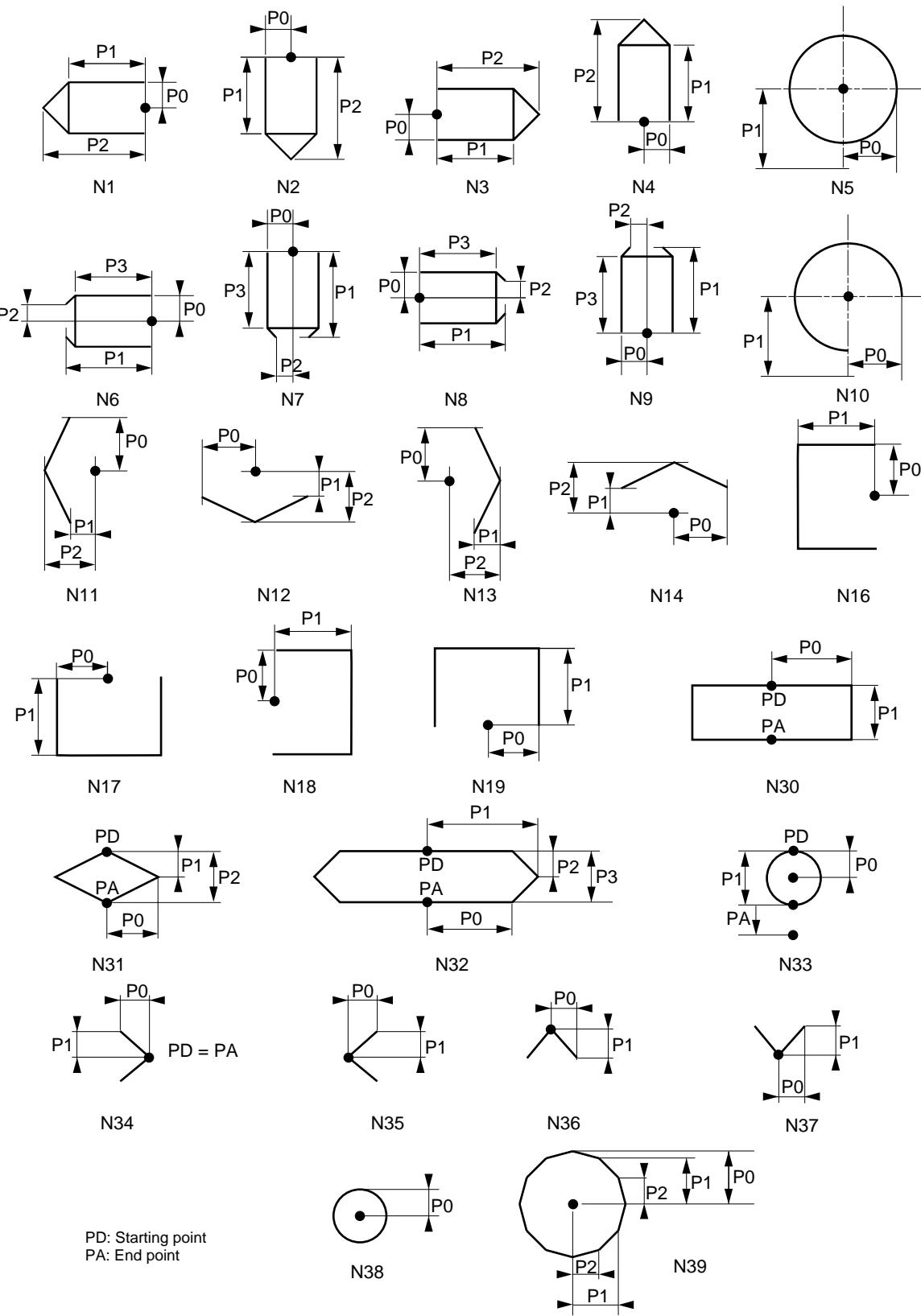


Figure 8.7 - Definition of icons

### 8.3.5.11 Screen Reference System Character String

Instruction 0x9BA8 is used to display an alphanumeric character string from the current point of the screen.

#### Syntax

```
0x9BA8 FONT STRING LF
```

##### FONT

Font number.

Character	Font	HEX code
«0»	Font 6 x 18	0xB1
«1»	Font 12 x 18	0xB2
«2»	Font 12 x 36	0xB3
«3»	Font 24 x 56	0xB4
«4»	Font 8 x 12	0xB5
«5»	Font 9 x 12	0xB6
«6»	Font 6 x 12	0xB7
«7»	Font 16 x 24	0xB8

##### STRING

All the alphanumeric characters are allowed in the font.

### 8.3.5.12 User Reference System Character String

Instruction 0x9B98 is used to display an alphanumeric character string in the current user position.

#### Syntax

```
0x9B98 FONT STRING LF
```

##### FONT

Font number (see Sec. 8.3.5.11).

##### STRING

All the alphanumeric characters allowed in the font.

### 8.3.5.13 Filling in User Area

Instructions 0x9BA9 and 0x9BAA are used to fill in an area of the user reference system.

Filling stops if:

- the window edge is reached,
- the clipping limit is reached,
- the filling colour is encountered.

#### Syntax

```
0x9BA9 { [CLIP] [CLIP] [CLIP] [CLIP] [CLIP] [X] [Y] [CONT-COL] } LF  
0x9BAA { [CLIP] [CLIP] [CLIP] [CLIP] [CLIP] [X] [Y] [CONT-COL] } LF
```

 **CAUTION**

Arguments [CLIP], [X] and [Y] are signed values.  
 The «+» sign is encoded by ASCII character «0».  
 The «-» sign is encoded by ASCII character «?».

**CLIP : | «G» «D» «H» «B» | [VALUE]**

Selection of the «clipping» limits.  
 Default value: No clipping.

Character	Limits selected	HEX code
«G»	Left	0xC7
«B»	Bottom	0xC2
«H»	Top	0xC8
«D»	Right	0xC4

**VALUE**

Hexadecimal value of the limit in the user reference system.  
 Default value: «0».

**X**

Hexadecimal X coordinate of a point in the area in the user reference system.  
 Default value: X coordinate of the current point.

**Y**

Hexadecimal Y coordinate of a point in the area in the user reference system.  
 Default value: Y coordinate of the current point.

**CONT-COL | «C» «c» | [VALUE]**

Contour colour.

Character	Definition	HEX code
«C»	Search for the contour in the four planes (Stop on the exact colour).	0xC3
«c»	Search for the contour in the planes relative to the colour (Stop on a component of the colour).	0xE3

Default values: Current colour and «C» selected.

**REMARK** *The colours are encoded on four bits. A component of the colour selected is a colour for which the same bits are set as for the colour selected (e.g. If the colour selected is Yellow «c5» (i.e. 0101 binary), the components of the colour are black (i.e. 0111 binary), orange (i.e. 1101 binary) and light grey/white (i.e. 1111 binary)).*

**Example**Filling a red rectangle

0x9BA9 G09 D0100 B0120 H0120 X050 Y0110 C1

### 8.3.5.14 Filling in Screen Area

Instructions 0x9BAB and 0x9BAC are used to fill in an area of the screen reference system.

#### Syntax

```
0x9BAB { [CLIP] [CLIP] [CLIP] [CLIP] [X] [Y] [CONT-COL] } LF  
0x9BAC { [CLIP] [CLIP] [CLIP] [CLIP] [X] [Y] [CONT-COL] } LF
```

**REMARK** *The syntax is exactly the same as for command 9BA9 (see Sec. 8.3.5.13). The coordinates are in the screen reference system.*

### 8.3.5.15 Draw Key Bar

Instruction 0x9BBC is used to separate the window vertically into ten fields.

#### Syntax

```
0x9BBC LF
```

**REMARK** *Although this instruction can be used in all the windows, it is only meaningful in the softkey message window.*

---

## **9 Analogue Inputs/Outputs**

<b>9.1 General</b>		<b>9 - 3</b>
<b>9.2 Configure an Analogue I/O Card</b>	<b>anas</b>	<b>9 - 3</b>
<b>9.3 Write an Analogue Output</b>	<b>anao</b>	<b>9 - 5</b>
<b>9.4 Read an Analogue Input</b>	<b>anai</b>	<b>9 - 6</b>
<b>9.5 Reassign an Analogue Card</b>	<b>anaa</b>	<b>9 - 7</b>



## 9.1 General

A maximum of 18 DACs and 20 ADCs are available on NUM 1060 systems. The analogue inputs and outputs are accessible to the user programme, part programme and dynamic operators. They are distributed as follows:

	Machine processor card	8I/8O analogue card (max. 2 cards)	UCSII card
1060 series I	4 ADCs - 2 DACs	8 ADCs/8 DACs	
1060 series II	4 ADCs - 2 DACs	8 ADCs/8 DACs	
1060 series II			2 ADCs/1 DAC

The analogue inputs and outputs are geographically identified by the card slot number and the channel number on the card. The code is one byte.

Bits 0 to 3 of the byte contain the channel number (0 to 7) and bits 4 to 7 the card number.

The CPU is number 1.

### Rate of variation

The analogue inputs and outputs are signed values on 16 bits. The rate of variation for these values is:

- for positive values:  
from 0 --> 0x7FFF  
for 0 --> n volts (where n is the full scale value)
- for negative values:  
from 0xFFFF --> 0x8000  
for 0 --> n volts (where n is the full scale value).

This is true regardless of the resolution of the ADC or DAC (8 or 12 bits).

The full scale value depends on the characteristics of the card used (see Installation and Commissioning Manual).

The accuracy depends on the DAC or ADC format (8 bits, 8 bits + sign or 12 bits + sign).

## 9.2 Configure an Analogue I/O Card

anas

9

### Syntax

anas(cch, wconfig)

cch: Card address code (the channel is not significant).  
config: Configuration, encoded on 16 bits.

### Operation

This function is used to configure the number of analogue inputs used (1 or 8) and the gain of each input (1 or 10).

The internal refresh period of each analogue input is 1.36 ms when all the 8 inputs are configured and 0.170 ms when only one input is configured (input 0).

This function is optional. At system initialisation, the cards are configured with 8 inputs and a gain of 1.

**REMARK** This function concerns only the analogue inputs and outputs on the additional cards.

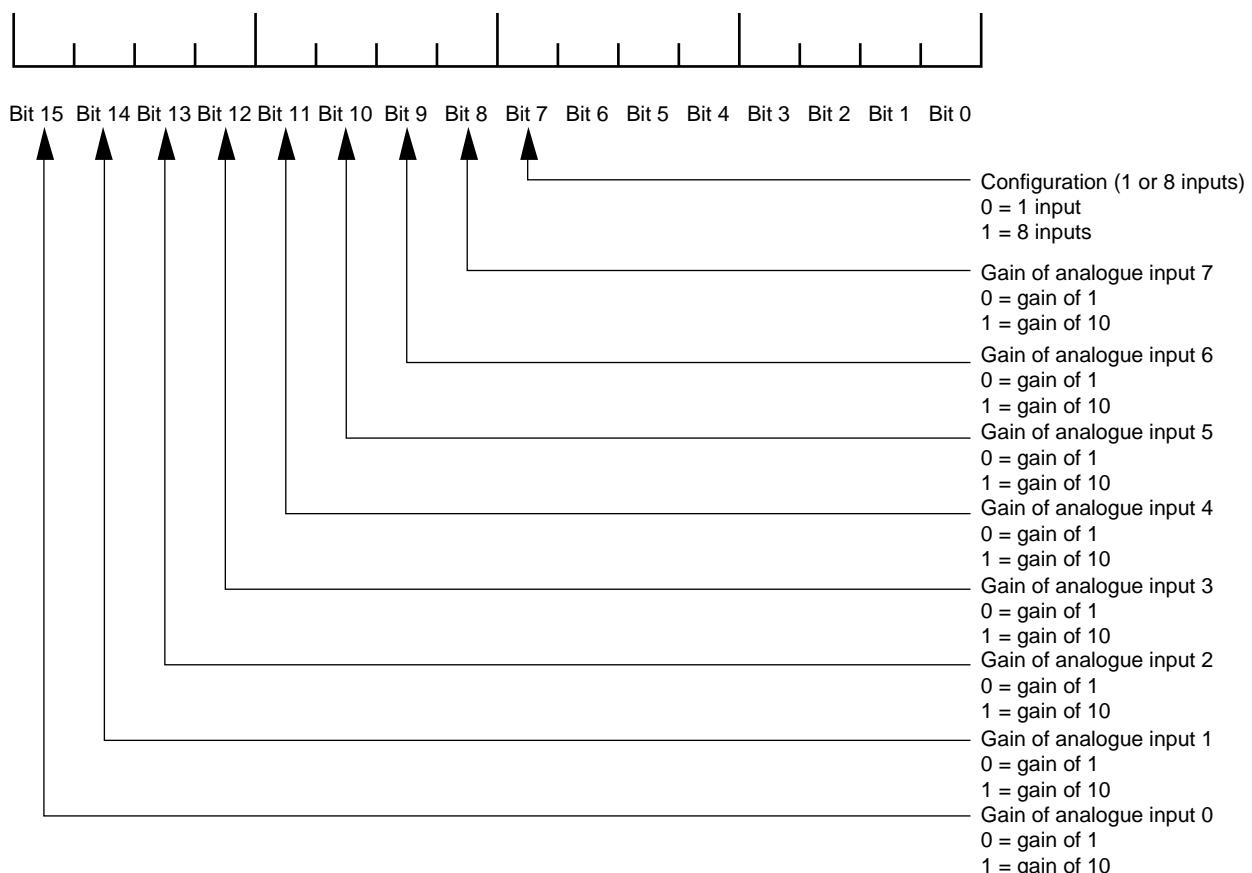
### Detail of argument «cch»



Refer to the Installation and Commissioning Manual for the card location on the system bus.

The values 0 to 7 correspond to the channel numbers of the eight input registers or eight output registers.

### Format of the configuration word



### **Return code**

#### If OK

0

#### Error

- 1: Card absent.  
2: Card parameter error.

## 9.3 Write an Analogue Output

**anao**

### Syntax

**anao(cch, woutput)**

cch: Output address encoded on one byte.  
woutput: Signed integer on 16 bits to be written.

### Operation

Writes DAC channel ch of card c.

#### Detail of argument «cch»



Refer to the Installation and Commissioning Manual for the card location on the system bus.

The values 0 to 7 correspond to the channel numbers of the eight input registers or eight output registers.

**Example:** Spindle speed programming controlled by the PLC

Read the miscellaneous function of the group (M3 or M4) specifying the spindle direction of rotation:

- M03\_g = 1:clockwise
- M04\_g = 1:counterclockwise

Read the spindle speed modulus (VITBRb). Its value ranges from 0 (zero speed) to 0x7FFF (maximum speed).

Send the DAC the signed encoded value on 16 bits. The sign depends on the spindle servo-drive wiring and the miscellaneous function of the group:

- If the value is positive or zero, ANAO(cch, VITBRb),
- If the value is strictly negative, ANAO(cch, ~VITBRb).

### Return code

#### If OK

0

#### Error

- |    |                          |
|----|--------------------------|
| 1: | Card absent.             |
| 2: | Card parameter error.    |
| 3: | Channel parameter error. |

## 9.4 Read an Analogue Input

**anai**

Syntax

```
anai(cch, &winput)
```

cch:

Input address encoded on one byte.

&winput:

Address of a variable (%Mxxx.W or %Vxxx.W) to be loaded with the value of the input on 16 signed bits.

### Operation

Reads an analogue input.



Refer to the Installation and Commissioning Manual for the card location on the system bus.

The values 0 to 7 correspond to the channel numbers of the eight input registers or eight output registers.

### Example

anai(0x37, %V100.&)

Reads input 7 from card 3 and loads the result into %V100.W.

### Return code

#### If OK

0

#### Error

- |    |                          |
|----|--------------------------|
| 1: | Card absent.             |
| 2: | Card parameter error.    |
| 3: | Channel parameter error. |

### Programming error causing a CPU fault

Access to a prohibited address:

- &winput parameter error.

## 9.5 Reassign an Analogue Card

**anaa**

### Syntax

```
anaa(initial_cch, future_cch)
```

initial\_cch: Byte containing the card No. (the channel number is not significant).

future\_cch: Byte containing the card No. (the channel number is not significant).

### Operation

Reassigns an analogue card.

This function is optional. It is used to reassign functions anas(initial\_cch, ...), anao(initial\_cch, ...), anai(initial\_cch, ...) to card future\_cch.



### CAUTION

This function is authorised only in PLC task %INI.

### Detail of the «initial\_cch» and «future\_cch» arguments



Refer to the Installation and Commissioning Manual for the card location on the system bus.

The values 0 to 7 correspond to the channel numbers of the eight input registers or eight output registers.

### Return code

9

#### If OK

0

#### Error

1: Final card absent.

2: Card parameter error.

4: Function called in a task other than an %INI task.



---

# 10 Explicit Read/Write of Input/Output Cards

<b>10.1 General</b>	10 - 3
<b>10.2 Explicit Read of an Input Card</b>	<b>read_i</b> 10 - 3
<b>10.3 Explicit Write to an Output Card</b>	<b>write_q</b> 10 - 4

10



## 10.1 General

The programmer can directly access the inputs/outputs on the serial bus (SB) without waiting for them to be refreshed by the PLC monitor.

*REMARK This function must be reserved for high priority cases, since it is costly in CPU time.*

## 10.2 Explicit Read of an Input Card

**read\_i**

### Syntax

```
read_i(rcmch, n)
```

rcmch: Word encoding the rack, card, module and channel.

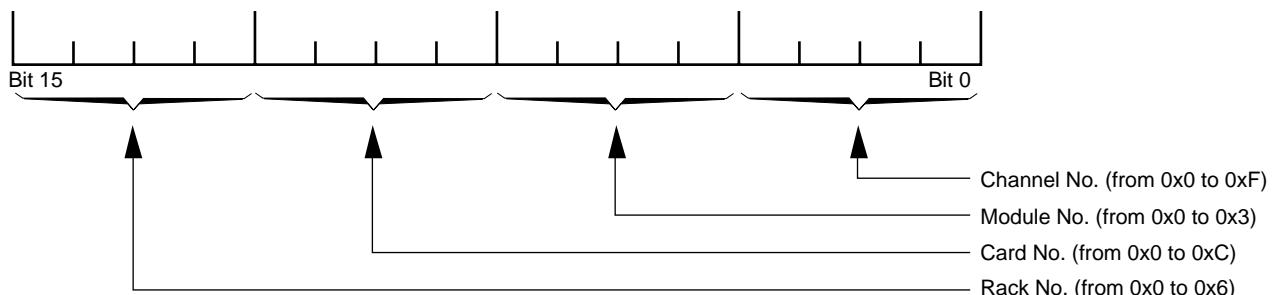
n: Number of bytes to be read.

### Operation

The system checks the coherence of parameters rcmch and n.

The system immediately reads card rc and updates image area %lrcmch to %lrcmch+n.

### Detail of argument «rcmch»



10

### Examples

- |                   |                            |
|-------------------|----------------------------|
| read_i(0x6b10, 1) | Causes refresh of %l6b10.B |
| read_i(0x6b10, 2) | Causes refresh of %l6b10.W |
| read_i(0x6b10, 4) | Causes refresh of %l6b10.L |

**Return code**If OK

0: Read OK.

Error

- |     |   |
|-----|---|
| 1:  | The variables requested exceed the card limits. The exchange does take place after truncating to the limits authorised by the card. |
| 2:  | Request to access an absent card.   |
| 3:  | Parameter rcmch outside the limits.   |
| -1: | Bus exchange error. The exchange does not take place.   |

**10.3 Explicit Write to an Output Card****write\_q****Syntax****write\_q(rcmch, n)**

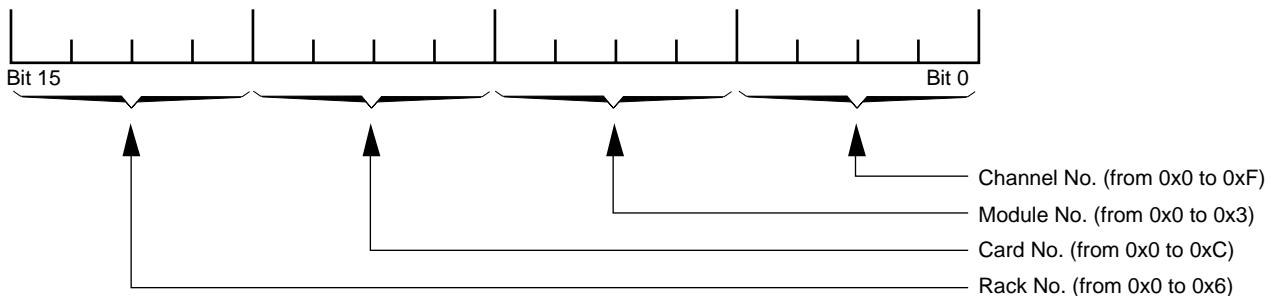
rcmch: Word encoding the rack, card, module and channel.

n: Number of bytes to be read.

**Operation**

The system checks the coherence of parameters rcmch and n.

The system immediately writes image area %Qrcmch to %Qrcmch+n in card rc.

Detail of argument «rcmch»**⚠ CAUTION**

This instruction causes rapid write of the outputs selected provided they were previously written.

### Examples:

%QB04.B=0xAA	
write_q(0xb04, 1)	Causes immediate write of 0xAA in %Qb04.B
%QB04.W=0xAAFC	
write_q(0xb04, 2)	Causes immediate write of 0xAAFC in %Qb04.W
%QB04.L=0xAAFC0000	
write_q(0xb04, 8)	Causes immediate write of 0xAAFC0000 in %Qb04.L and %Qb08.L.

### Return code

#### If OK

0: Read OK.

#### Error

1:	The variables requested exceed the card limit. The exchange does not take place after truncating to the limits authorised by the card.
2:	Request to access an absent card.
3:	Parameter rcmch outside the limits.
-1:	Bus exchange error. The exchange does not take place.



# 11 Interrupt Inputs

<b>11.1 General</b>		11 - 3
11.1.1	On-the-Fly Dimension Measurement	11 - 3
11.1.2	Interrupts Assigned to a %TH Hardware Task	11 - 3
<b>11.2 Principle of Line Assignment</b>		11 - 5
<b>11.3 Associate an Interrupt Input with Axis Groups</b>	<b>iti_gr</b>	11 - 5
<b>11.4 Configure an Interrupt Input</b>	<b>itictl</b>	11 - 6
<b>11.5 Read an Interrupt Input</b>	<b>itiget</b>	11 - 8
<b>11.6 Associate a %TH Task with an IT Input</b>	<b>thiti</b>	11 - 9



## 11.1 General

The automatic control function processes priority interrupts on lines:

	Machine processor card	IT/serial line card (max. 2 cards)	USCII card
1060 series I	iti0 to iti3	iti4 to itiB	
1060 series II	iti0 to iti3	iti4 to itiB	
1060 series II			iti0

Priority interrupts are associated with:

- function iti\_gr for on-the-fly dimension measurement,
- function thiti to suspend the cyclic tasks of the user programme and execute an interrupt routine programmed in a %TH hardware task.

### 11.1.1 On-the-Fly Dimension Measurement

These interrupts are processed by G10 in the part programme (see «PROGRAMMING MANUAL»).

An interrupt sent on one of lines iti0 to itiB is processed by the user programme. As soon as the IT is sent, the monitor instructs the CNC function to update external parameters E70001 to E78001 (position reference of an axis or group for on-the-fly measurements).

These external interrupts, dedicated to probing, can be parameterised by function iti\_gr().

Interrupt management must be programmed in TS0.

**REMARKS** *With multiple axis groups, if two interrupts arrive at the same time, the interrupt assigned to line iti0 has the highest priority and that assigned to itiB has the lowest. If an interrupt assigned to axis group 1 arrives at the same time as an interrupt assigned to axis group 8 on the same line, the interrupt assigned to group 1 is processed first by the monitor.*

### 11.1.2 Interrupts Assigned to a %TH Hardware Task

The assignment of a line iti0 to itiB to a hardware task causes execution of the interrupt routine programmed in the %TH task.

 **CAUTION**

If a hardware task and a gauging task are assigned and programmed on the same line, the hardware task is suspended during dimension measurement by the monitor.

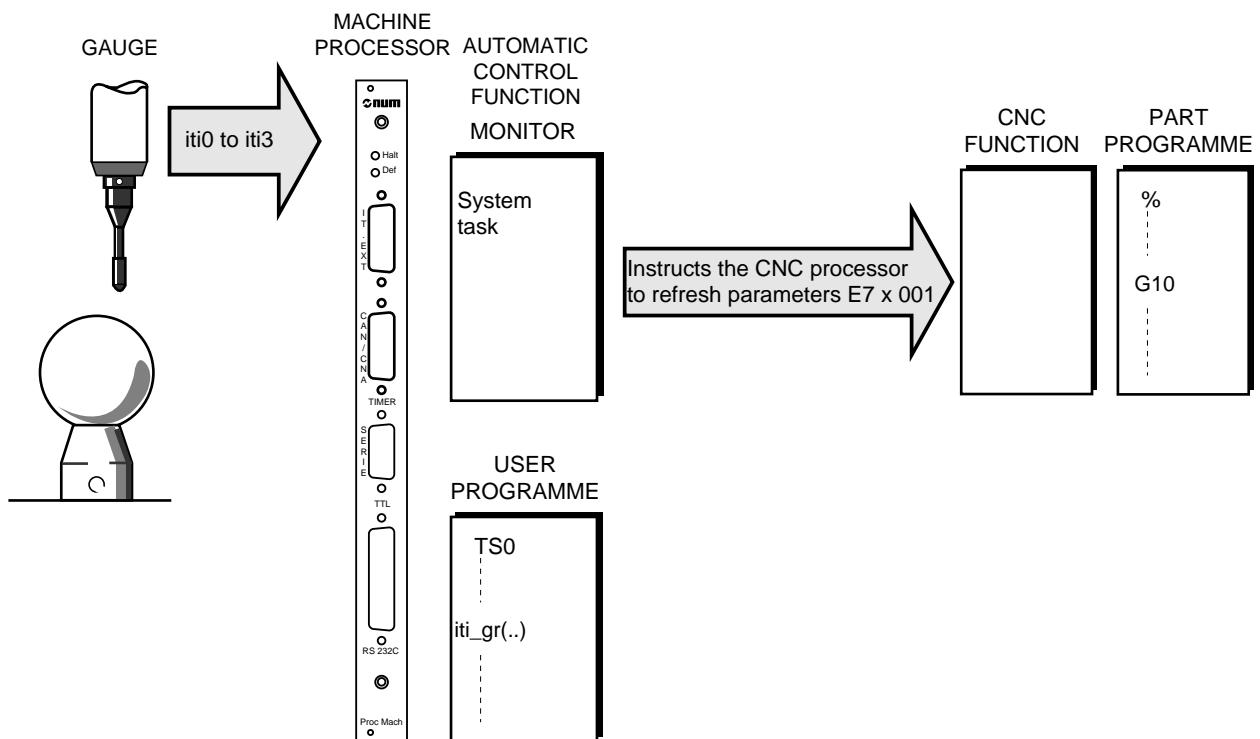


Figure 11.1 - Processing for on-the-fly dimension measurement

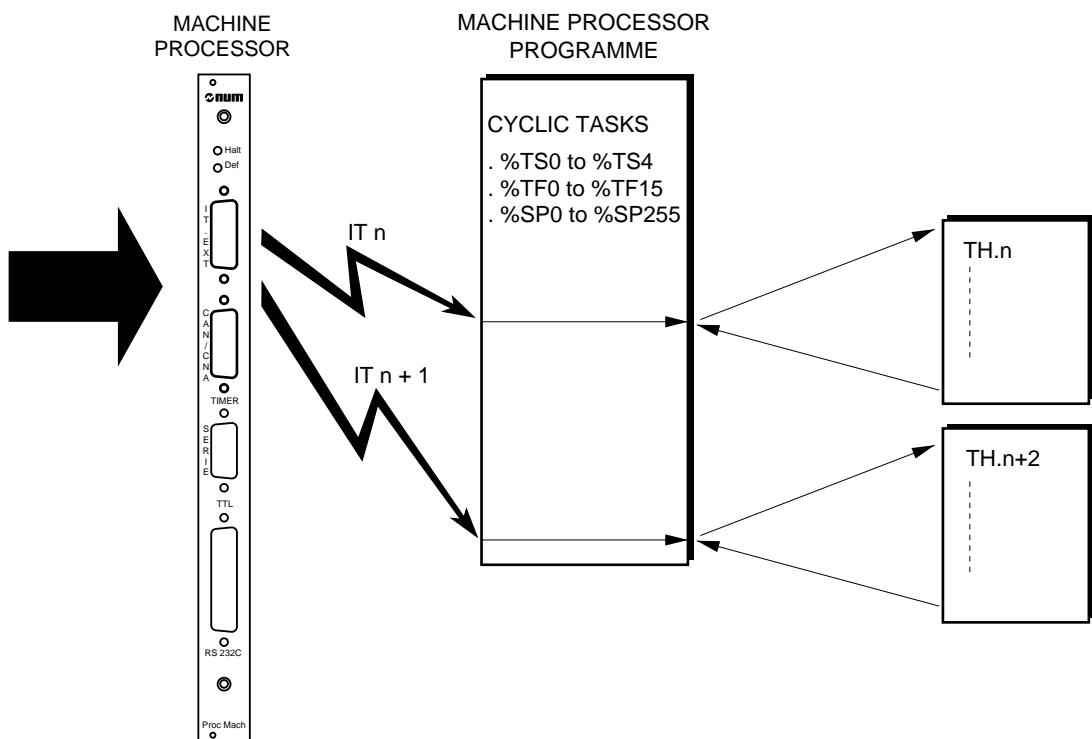


Figure 11.2 - Interrupts assigned to a TH

## 11.2 Principle of Line Assignment

### NUM 1060 Series I and NUM 1060 Series II (Multicard)

The interrupt input line numbers are assigned by increasing order to:

- the machine processor card,
- the first IT/Serial line card encountered in the rack,
- the second IT/Serial line card encountered in the rack.

Pin No.			Interrupt input line No.	Card type
5V	24V	Common		
1	2	9	Line 0	Machine processor card
10	11	3	Line 1	Machine processor card
4	5	12	Line 2	Machine processor card
13	14	6	Line 3	Machine processor card
1	2	5	Line 4	First IT/Serial line card
3	4	5	Line 5	First IT/Serial line card
6	7	5	Line 6	First IT/Serial line card
8	9	5	Line 7	First IT/Serial line card
1	2	5	Line 8	Second IT/Serial line card
3	4	5	Line 9	Second IT/Serial line card
6	7	5	Line A	Second IT/Serial line card
8	9	5	Line B	Second IT/Serial line card

### NUM 1060 Series II (UCSII)

Only one line is available:

Pin No.			Interrupt input line No.	Card type
5V	24V	Common		
6	1	2	Line 0	USCII card

## 11.3 Associate an Interrupt Input with Axis Groups

**iti\_gr**

### Syntax

**iti\_gr(itino, group)**

itino: Interrupt input number (From 0 to 0xB).

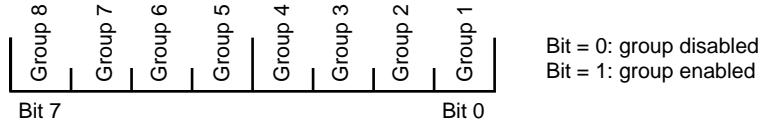
group: List of bits indicating the axis groups involved in the interrupt.

This function is used to associate an IT input with one or more axis groups.

### Operation

An interrupt on the input causes the monitor to read all the axis couplers for the axes in the groups declared. The monitor then indicates an axis read to the CNC processor for each axis group.

### Detail of the «group» argument



### **Return code**

If OK

0

Error

-1: n\_itio not between 0 and 0xB

## **11.4 Configure an Interrupt Input**

**itictl**

### **Syntax**

**itictl(itino, iti\_config)**

itino:                          Interrupt input number (From 0 to 0xB).

iti\_config:                   Component configuration code.

Used to configure an interrupt input.

### **Operation**

The component configuration code on 8 bits is transmitted in parameter iti\_config.

After detection of the change of state, the machine processor waits for the signal to settle before acting on the change of state.

The possibilities for filtering selection are independent of the active edge.

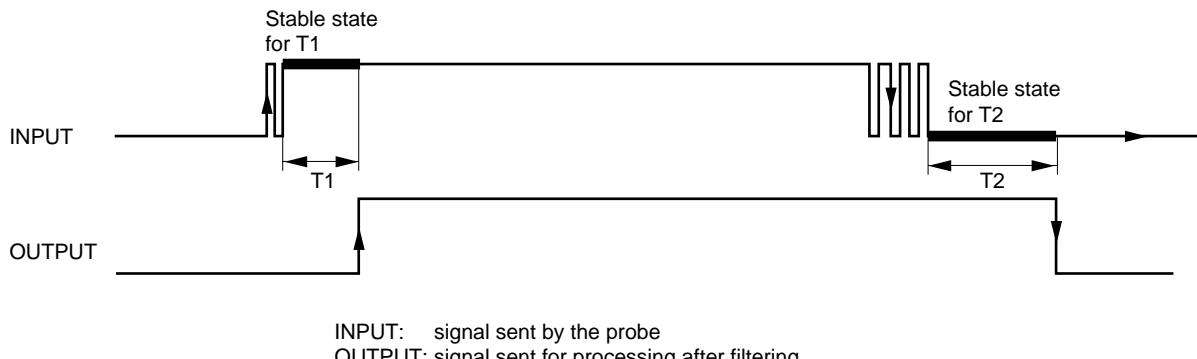
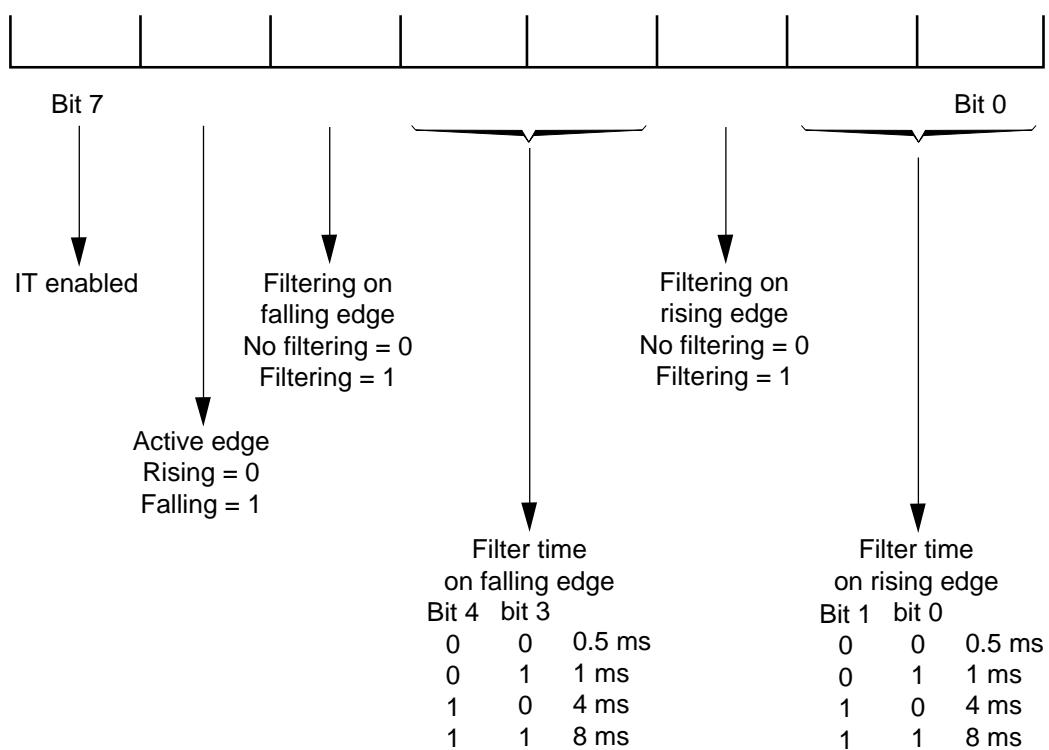
The filtering time corresponds to the settling time before action.

The filtering time can be parameterised with the following values:

- 0.5 ms
- 1 ms
- 4 ms
- 8 ms
- 1 to 3 ms for a fast cycle (without filtering).

**Example**

Filtering on rising edge T1 and falling edge T2

**Detail of Parameter iti\_config****Return code**If OK

0

## 11.5 Read an Interrupt Input

**itiget**

### Syntax

```
itiget(itino)
```

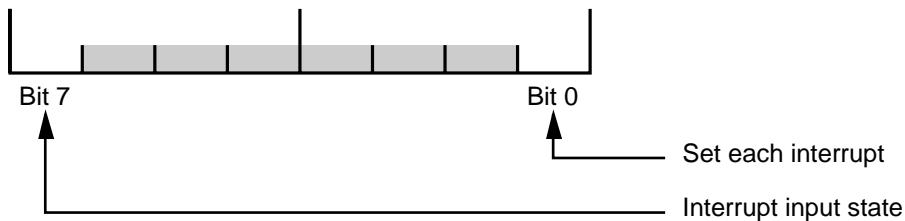
itino:                                      Interrupt input number (From 0 to 0xB).

### Operation

itiget() returns the interrupt input status register contents.

This parameter is updated by the monitor each RTC cycle. The maximum delay with respect to the real state of the line is 20 ms.

### Detail of the register



The call to itiget() resets bit 0 of the register (record of the occurrence of an IT).

### Return code

#### If OK

Input status register.

#### Error

-1: n\_itii not between 0 and 0xB

## 11.6 Associate a %TH Task with an IT Input

**thiti**

### Syntax

```
thiti(thno, itino)
```

thno: %TH task number.

itino: Interrupt input number (From 0 to 0xB).

Associates the %TH task with an interrupt input.

### Operation

When the IT input causes a hardware interrupt, the system calls the associated %TH task.

### Return code

#### If OK

0



---

## 12 Serial Lines

<b>12.1 General</b>		12 - 3
<b>12.2 Select Data Rate and Format</b>	<b>comf</b>	12 - 4
<b>12.3 Send a Buffer</b>	<b>comout</b>	12 - 6
<b>12.4 Reception of a Buffer</b>	<b>comin</b>	12 - 7
<b>12.5 Read the Status of a Serial Line</b>	<b>comreg</b>	12 - 10
<b>12.6 Control the Serial Line Driver</b>	<b>comctl</b>	12 - 11
<b>12.7 Transmission Standards</b>		12 - 12
12.7.1 Before Software Index F		12 - 12
12.7.1.1 No Flow Control		12 - 12
12.7.1.2 RTS/CTS Flow Control		12 - 12
12.7.1.3 Xon/Xoff Flow Control		12 - 12
12.7.2 RS232 Standard		12 - 12
12.7.2.1 No Flow Control		12 - 12
12.7.2.2 RTS/CTS Flow Control		12 - 12
12.7.2.3 Xon/Xoff Flow Control		12 - 13
12.7.3 RS485 Standard		12 - 13
12.7.4 RS422 Standard		12 - 13
12.7.4.1 No Flow Control		12 - 13
12.7.4.2 Xon/Xoff Flow Control		12 - 13



## 12.1 General

The automatic control function controls 12 serial lines distributed as follows:

CNC processor card	Machine processor card	IT/Serial line card (max. 2 cards)	UCSII card
1060 Series I DNC - PERIPH	RS232C - TTL	Line 1 to line 4	
1060 Series II	RS232C - TTL	Line 1 to line 4	
1060 Series II			COMM 1 - COMM 2

The user programme can control data transfers with a peripheral in the context of special applications.

For more details on installation of these lines, refer to the Installation and Commissioning Manual.

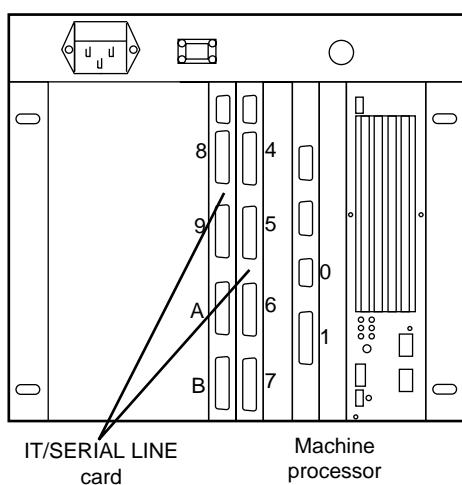
### Line Numbering

#### NUM 1060 Series I and NUM 1060 Series II (Multicard)

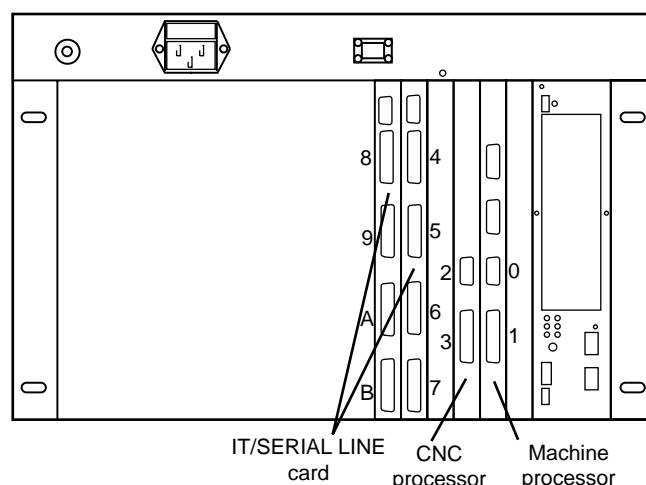
The line numbers are set once and for all on the CNC and machine processor cards:

- 0 for the machine processor TTL line,
- 1 for the machine processor RS232C line,
- 2 for the CNC processor DNC line,
- 3 for the CNC processor PERIPH line.

Numbers 4 to 7 are assigned to the first IT/SERIAL LINE card encountered in the rack and numbers 8 to B are assigned to the second IT/SERIAL LINE card encountered (scanned from right to left).



NUM 1060 Series II



NUM 1060 Series I

#### NUM 1060 Series II (UCSII)

The line number is hard-wired on the UCSII card:

- 0 for the COMM1 line,
- 1 for the COMM2 line.

## 12.2 Select Data Rate and Format

**comf**

### Syntax

```
comf(portno, txrate, rxrate, format)
```

portno: Communication port number (0 to B).

txrate: Transmit data rate.

rxrate: Receive data rate.

format: Data format code.

Used to select the transmit and receive data rate and the format of a serial line.

### Operation

Function comf() allocates the line to the automatic control function and configures the port. Once the line has been initialised, it cannot be allocated to another user (CNC function, etc.).

The function comf(portno, txrate, rxrate, 0) deallocates the line and makes it available for another user.

 **CAUTION**

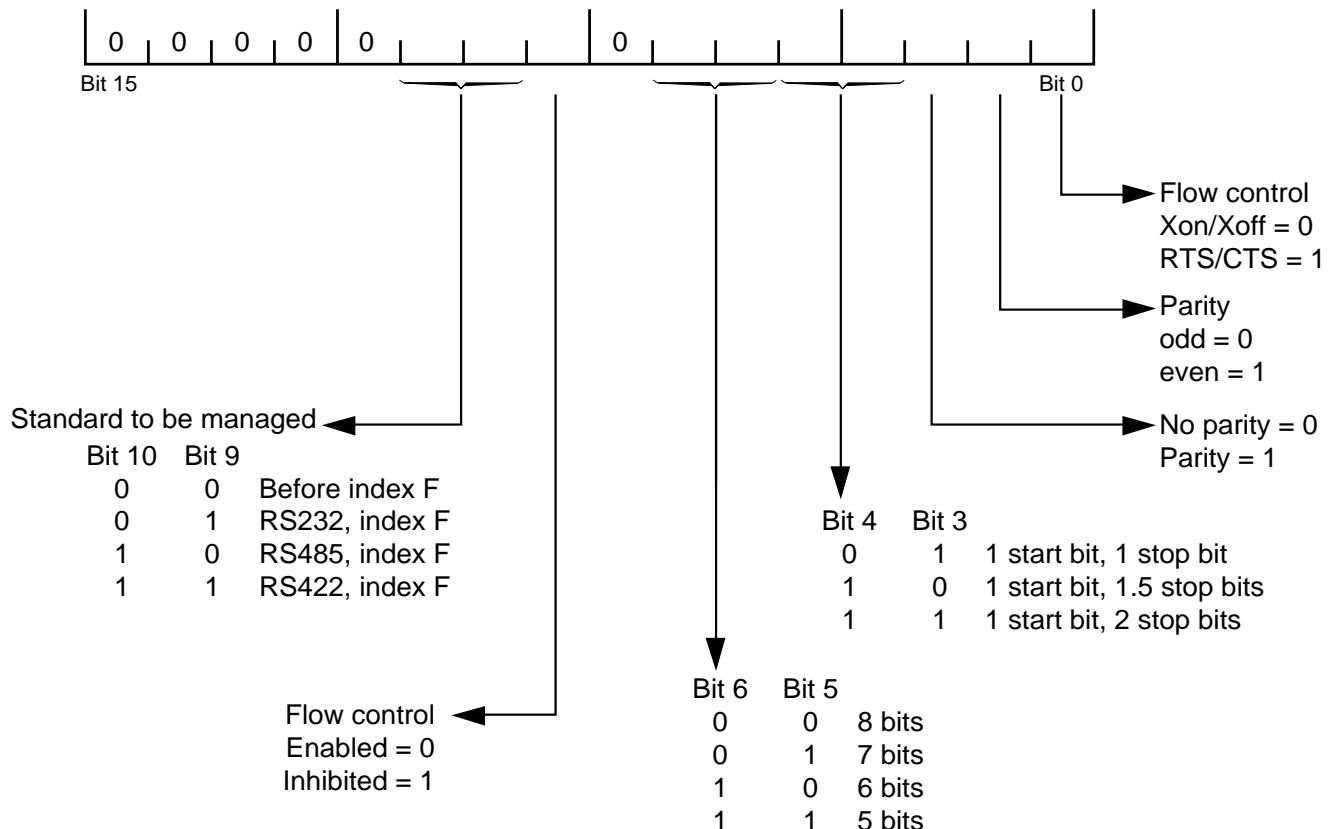
The transmit and receive baud rates must always be the same.

When initialising a line on the CNC processor card (line 2 or 3), it is necessary to call function comf() in a task %TF.

### Value of arguments «txrate» and «rxrate»

The values correspond to baud rates.

Value of txrate and rxrate	Baud rate
300	300
600	600
1200	1200
2400	2400
4800	4800
9600	9600
19200	19200

Detail of the «format» argument

**REMARK** In a future version, it is recommended to reset the nonsignificant bits.

**Return code**

If OK

0

Error

-1:

«format» argument invalid  
Line already allocated to a user other than the automatic control function.

## 12.3 Send a Buffer

**comout**

### Syntax

```
comout(portno, &buffer, nb)
```

portno:	Communication port number (0 to B).
&buffer:	Address of the buffer to be sent.
nb:	Number of bytes to be sent (1 <= nb <= 255). nb is encoded on an unsigned byte.

Sends a buffer over serial communication line portno.

### Operation

When this function is called, the system copies the «&buffer» buffer, starts transmission and returns to the calling function. This function is nonblocking and transmission continues by IT until reaching the end of the buffer. Function comreg() returns the status of the transmission in progress.

The call comout (n\_port, &buffer, 0) aborts a transmission in progress if any.

### Return code

#### If OK

0

#### Error

-1:	n_port invalid Line not initialised Transmission in progress Argument «nb» greater than 255 No full duplex with Xon/Xoff flow control.
-----	--

### Programming error causing a CPU fault

Access to a prohibited address:

- &buffer parameter error,
- &buffer+nb outside authorised area.

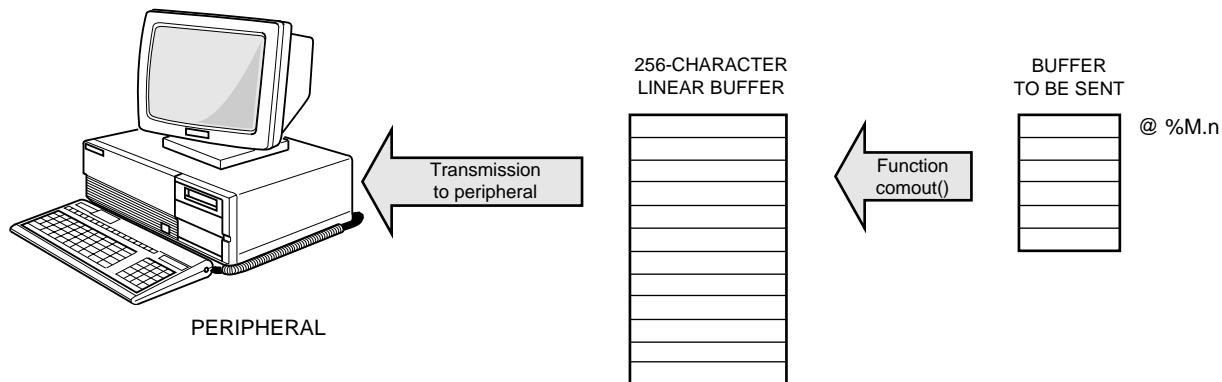


Figure 12.1 - Transmission of a buffer

## 12.4 Reception of a Buffer

**comin**

### Syntax

```
comin(portno, &buffer, nb)
```

- portno: Communication port number (0 to B).  
&buffer: Address of the buffer where the characters received are to be stored.  
nb: Maximum number of characters to be read.

Reads from the reception buffer of serial line portno.

### Operation

The system manages a 236 character circular reception buffer. Function comin() is used to read all or part of the buffer. The number of characters read is equal to the number of characters requested (nb) or the number of characters present in the circular buffer, whichever is lower.

If the line was not already set to reception by function comctl(), the first call to comin() causes an immediate switch to reception.

The call to comin(portno, &buffer, 0) causes a stop and initialises reception.

## Return code

### If OK

$n \geq 0$

Number of characters copied into &buffer.

### Error

-1:

n\_port invalid  
Line not initialised  
No full duplex with Xon/Xoff flow control

## Programming error causing a CPU fault

Access to a prohibited address:

- &buffer parameter error,
- &buffer+nb outside authorised area.

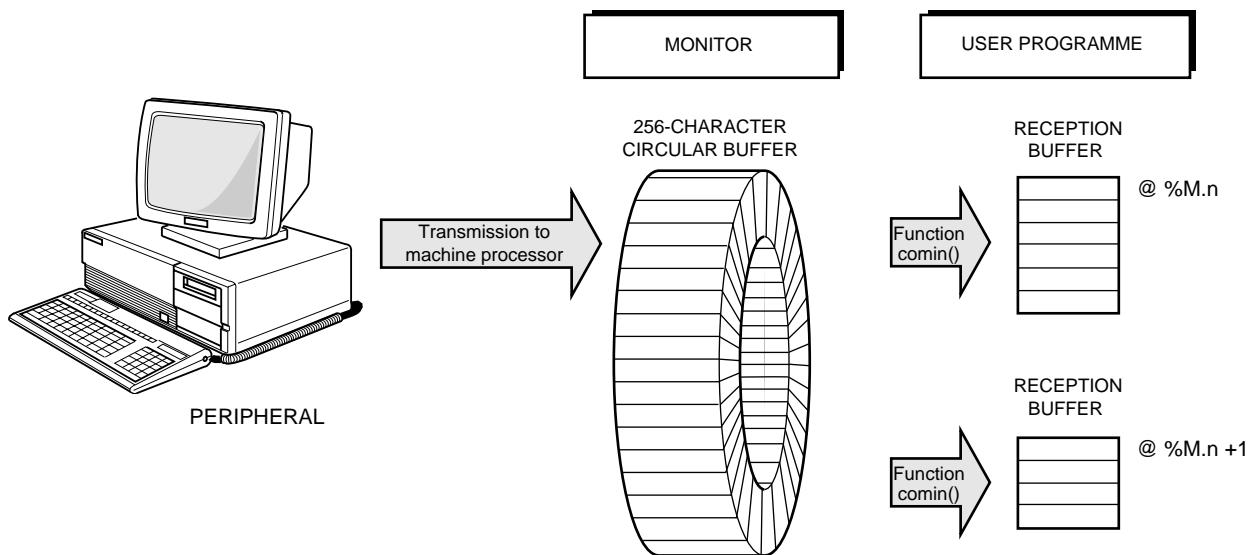


Figure 12.2 - Reception of a buffer

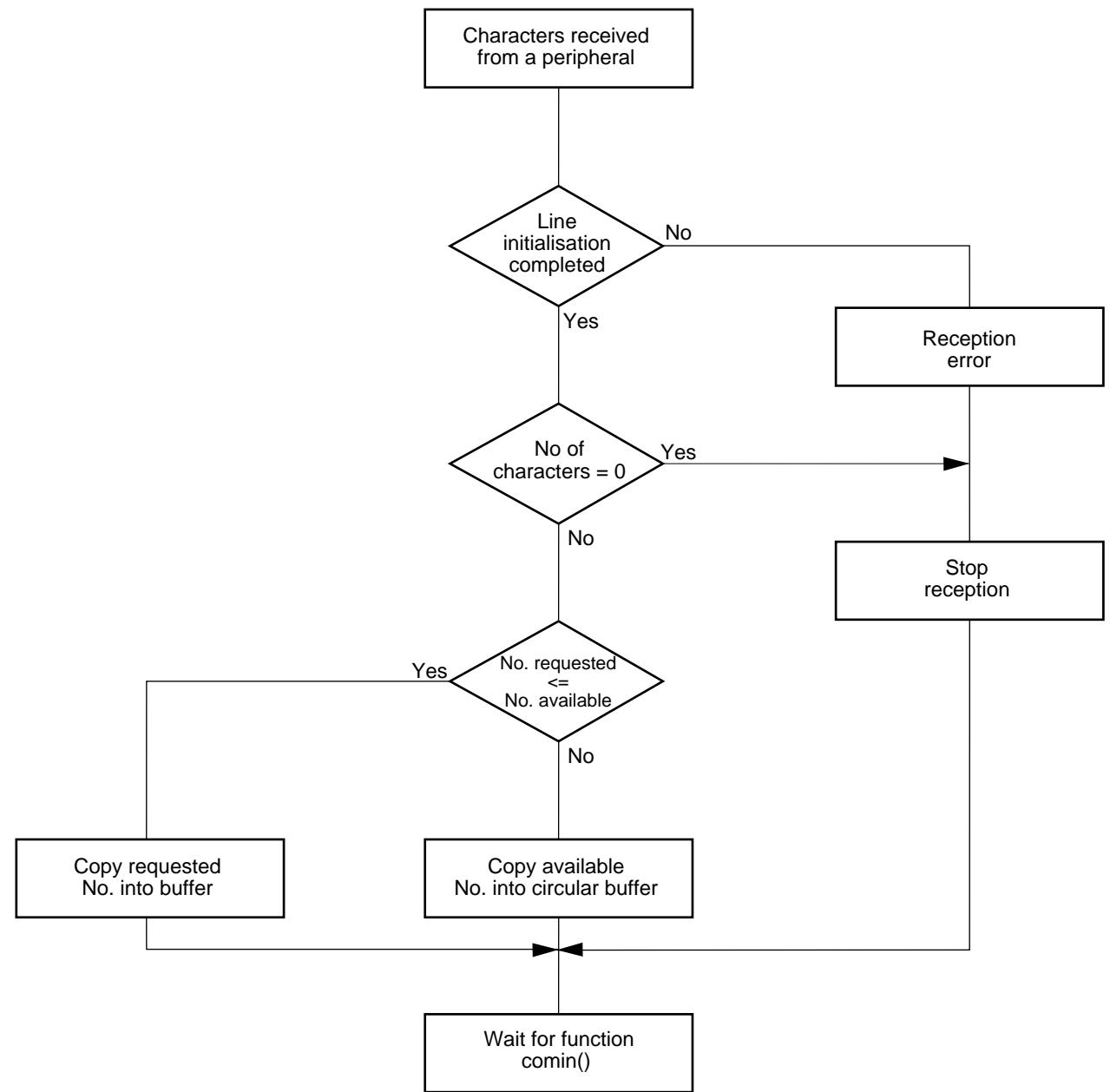


Figure 12.3 - Reception processing by the monitor

## 12.5 Read the Status of a Serial Line

**comreg**

### Syntax

**comreg(portno)**

portno: Communication port number.

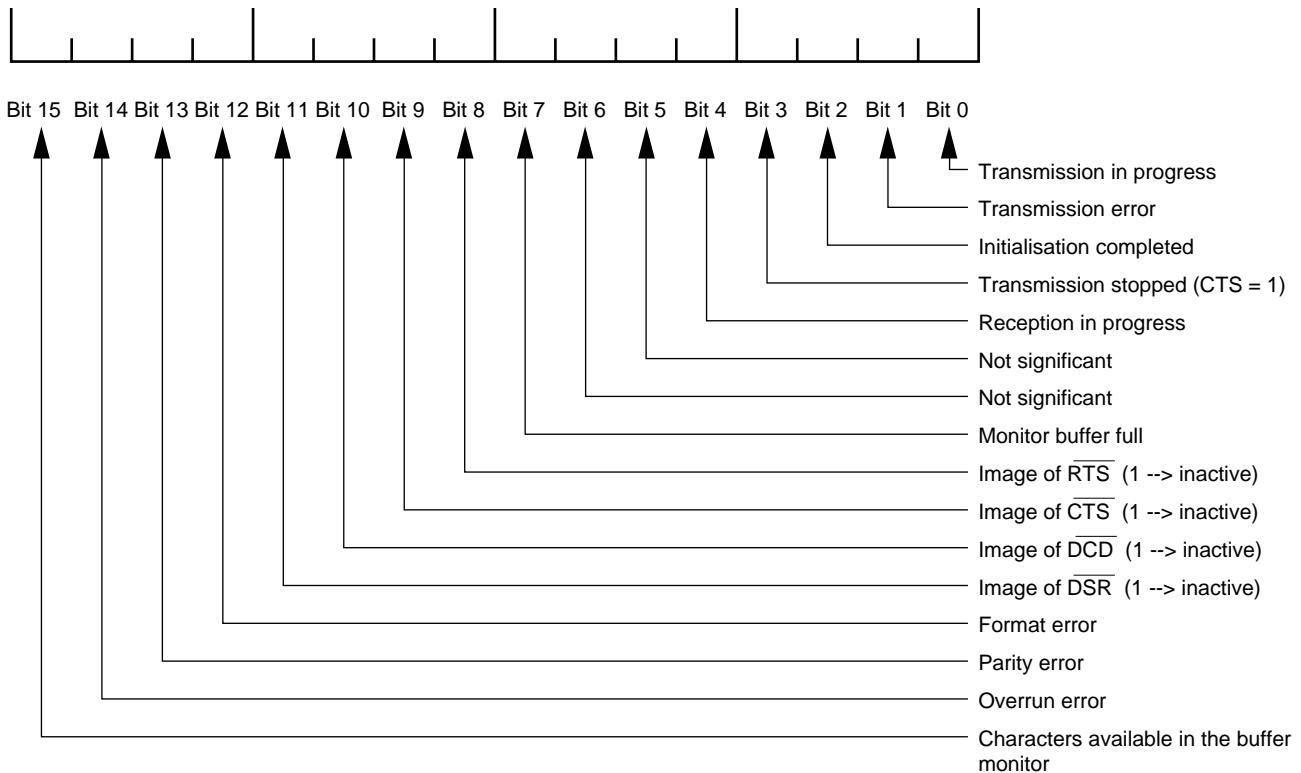
### Operation

Returns the general state of serial line portno.

**! CAUTION**

When initialising a line of the CNC processor card (line 2 or 3), it is necessary to call function comreg() in a task %TF.

### Return code



## 12.6 Control the Serial Line Driver

## comctl

### Syntax

```
comctl(portno, config)
```

portno: Communication port number.

config: Configuration code.

comctl() is used to control the driver of serial line portno.

### Operation

The action depends on the flow control used and the line status at the time of the request.

Value of «config»	No flow control	RTS/CTS flow control	Xon/Xoff flow control
No reception in progress and config == 0	Line switched to reception	Line switched to reception and signal RTS activated	Line switched to reception and Xon character sent
Reception in progress and config == 1	No effect	Signal RTS deactivated	Xoff character sent

### Return code

#### If OK

0

#### Error

-1:

Line not initialised.  
No full duplex with Xon/Xoff flow control.  
Request (config) incompatible with current state.

## 12.7 Transmission Standards

The transmission standard is defined in the «format» argument of function comf() (see Sec. 12.2).

### 12.7.1 Before Software Index F

#### 12.7.1.1 No Flow Control

No hardware or software signals are managed for transmission or reception.

Full duplex operation is possible.

#### 12.7.1.2 RTS/CTS Flow Control

Signal RTS is managed during reception to stop and restart transfers. During transmission, RTS remains active while the buffer is being transmitted.

Full duplex operation is impossible.

**REMARK** *Signal RTS can be wired to CTS to ignore these signals and allow full duplex operation without flow control.*

#### 12.7.1.3 Xon/Xoff Flow Control

During reception, the transfer is monitored by sending control characters on the transmission channel.

As soon as character DC1 (Xon) is sent by the receiver, the sender is authorised to send. When the receiver sends character DC3 (Xoff), the sender has a time equal to transmission of 20 characters to stop transmission.

### 12.7.2 RS232 Standard

For software at index F or above.

#### 12.7.2.1 No Flow Control

No hardware or software signals are managed for transmission or reception.

Full duplex operation is possible.

#### 12.7.2.2 RTS/CTS Flow Control

For reception, signal RTS is controlled to monitor the line. This signal is not set by the sender.

As soon as this signal goes low, the sender must stop sending data. Only one more character can be sent after RTS goes low.

As seen by the sender, transmission must be stopped when signal CTS goes low.

Full duplex operation is possible.

### 12.7.2.3 Xon/Xoff Flow Control

During reception, the transfer is monitored by sending control characters on the transmission channel.

As soon as character DC1 (Xon) is sent by the receiver, the sender is authorised to send. When the receiver sends character DC3 (Xoff), the sender has a time equal to transmission of 20 characters to stop transmission.

Full duplex operation is impossible.

### 12.7.3 RS485 Standard

For software at index F or above.

With standard RS485, flow control is impossible. If standard RS485 is enabled by function comf(), the value of bit 0 of the «format» argument is not significant.

Signal RTS is high during transmission of a buffer and low during reception to set the RS232/RS485 interface adapters to transmission or reception.



#### CAUTION

The use of standard RS485 requires suitable wiring of the RS232/RS485 interface adapters.

### 12.7.4 RS422 Standard

For software at index F or above.

With this standard, signal RTS remains high while the line is being used.

#### 12.7.4.1 No Flow Control

No hardware or software signals are managed for transmission or reception.

Full duplex operation is possible.

#### 12.7.4.2 Xon/Xoff Flow Control

During reception, the transfer is monitored by sending control characters on the transmission channel.

As soon as character DC1 (Xon) is sent by the receiver, the sender is authorised to send. When the receiver sends character DC3 (Xoff), the sender has a time equal to transmission of 20 characters to stop transmission.



# 13 Timer Function

## 13.1 General Description of the Timer Function

The timer function is available only on NUM 1060 series I and NUM 1060 series II multicard systems.

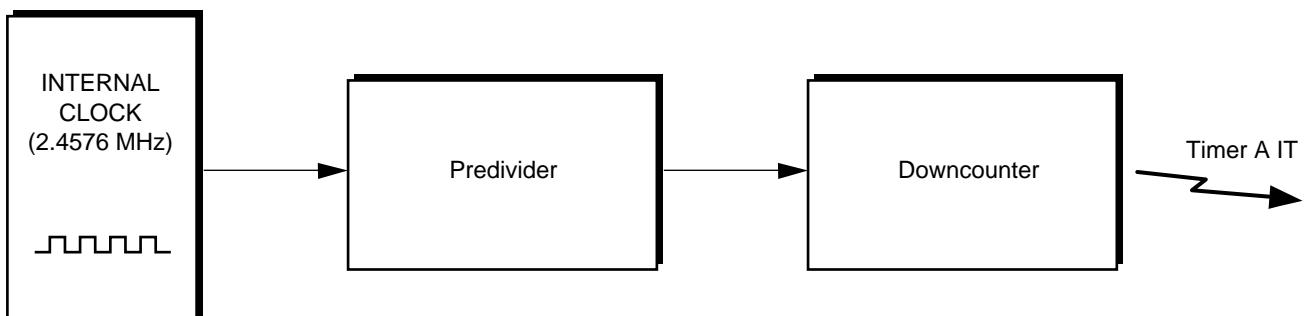
The automatic control function provides the user with two timers, A and B. They each consist of an 8-bit downcounter capable of generating a pulse each time the contents are changed. The downcounter is then immediately reloaded with a programmed value saved in the timer data register.

For further details on the wiring of these inputs, refer to the «Installation and Commissioning Manual».

## 13.2 Use of Timer A

### 13.2.1 Timeout Mode

In timeout mode, the frequency of the internal clock (2.4576 MHz) is divided by the programmable predivider which supplies pulses to the downcounter.



## 13.3 Associate a %TH Task with a Timer

**thtimer**

### Syntax

```
thtimer(thno, timerno, nbmillisec)
```

thno: %TH task number.

timerno: Timer number (0 or 1).

nbmillisec: Time in milliseconds.

Associates a %TH task with an interrupt generated by timer A or B.

### Operation

The call to thtimer() sets the timer whose number is given in timerno with the value specified in nbmillisec. When this time has expired, the timer generates an IT used by the system to call the %TH task number thno.

The time nbmillisec is between 0 and 2,147,483,647 milliseconds.

Function thtimer() is modal, i.e. after the call to thtimer(), the %TH task is called cyclically every nbmillisec.

To cancel a thtimer() function, it is necessary to call thtimer() with parameter nbmillisec equal to ZERO.

The call to thtimer() while the timer is running causes a reset of the timer and a restart from nbmillisec.

**Return code**

If OK

0

Error

-1: nbmillisec < 0 or > 2,147,483,647

# 14 Date-Time Stamp Function

## 14.1 General Description of the Date-Time Stamp Function

The current date is read by function tmget().

The system date is managed by a date-time stamp saved in the global memory.

A CNC page allows the operator to reset the date-time stamp.

## 14.2 Read the Current Date

**tmget**

### Syntax

```
tmget(&date)
```

&date: Address of the memory block (11 bytes) where the date structure is to be stored.

### Operation

Used to read the current date.

### Date Block Structure

Byte No.	Data type	Value
bytes 0 - 1	Year	0 to 65535
byte 2	Month	1 to 12
byte 3	Day	1 to 31
byte 4	Hour	0 to 23
byte 5	Minutes	0 to 59
byte 6	Seconds	0 to 59
bytes 7 - 8	Milliseconds	0 to 999 (Accuracy around 50 ms)

### Return code

If OK

0

Error

-1: The date was not updated.

### Programming error causing a CPU fault

Access to a prohibited address:

- &date parameter error.

## 14.3 Read the Current Date and Day in Week

**dtget**

### Syntax

```
dtget(&date)
```

**&date:** Address of the memory block (11 bytes) where the date structure is to be stored.

### Operation

Used to read the current date and the day in the week

### Date Block Structure

Byte No.	Data type	Value
byte 0	Day in week	0 to 6
byte 1	Date	1 to 31
byte 2	Month	1 to 12
byte 3	Year	0 to 99
byte 4	Hour	0 to 23
byte 5	Minutes	0 to 59
byte 6	Seconds	0 to 59

### Return code

#### If OK

0

#### Error

-1: The data was not updated

### Programming error causing a CPU fault

Access to a prohibited address:

- &date parameter error.

# 15 Exchanges by Protocol

15

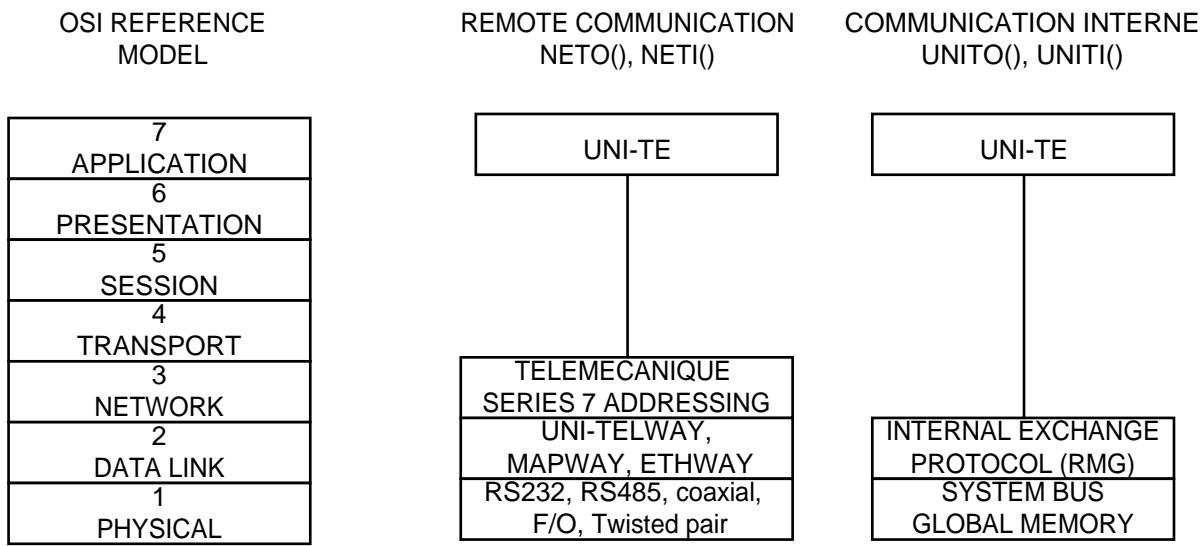
<b>15.1 General Description of Exchanges</b>		15 - 3	
15.1.1	General Description of the DNC1000 Protocol	15 - 4	
15.1.2	DNC1000 Exchange Mechanism	15 - 5	
15.1.2.1	Request Processing Procedure	15 - 5	
15.1.2.2	Concept of Port	15 - 6	
<b>15.2 Objects Accessible by a UNITE Request</b>		15 - 7	
15.2.1	List of Object Type Requests Processed by the CNC Processor	15 - 7	
15.2.2	Object Constituents	15 - 9	
15.2.3	Programme Status Segments	15 - 14	
<b>15.3 UNITE Requests Processed by the CNC Function</b>		15 - 16	
13.3.1	«READ-OBJECT» Request	15 - 16	
15.3.2	«WRITE-OBJECT» Request	15 - 18	
15.3.3	«DELETE-FILE» Request	15 - 19	
15.3.4	«READ-MEMORY-FREE» Request	15 - 20	
15.3.5	«OPEN-DIRECTORY» Request	15 - 21	
15.3.6	«DIRECTORY» Request	15 - 22	
15.3.7	«CLOSE-DIRECTORY» Request	15 - 24	
15.3.8	«READ-BLOCK» Request	15 - 25	
15.3.9	«WRITE-BLOCK» Request	15 - 26	
15.3.10	«RESERVE-MEMORY» Request	15 - 27	
15.3.11	«MESSAGE READ» Request	15 - 28	
<b>15.4 Programming the General Request Function</b>		15 - 29	
15.4.1	Send a Request	<b>unito</b>	15 - 29
15.4.2	Read an Answer	<b>uniti</b>	15 - 30
15.4.3	Programming Rules		15 - 32
<b>15.5 Exchanges With a Remote Station</b>		15 - 34	
15.5.1	Send a Request	<b>neto</b>	15 - 34
15.5.2	Read an Answer	<b>neti</b>	15 - 36
15.5.3	Examples of Series 7 Addressing		15 - 38
15.5.4	Setting up the Common Word Service	<b>setcomw</b>	15 - 39
15.5.5	Answer to the STATU Request	<b>netst_ad</b>	15 - 40



## 15.1 General Description of Exchanges

Exchanges by protocol allow communication between:

- the automatic control and CNC functions of the system (DNC1000 local network),
- the NUM 1060 CNC and remote stations connected to MAPWAY, ETHWAY and UNI-TELWAY networks (remote communication).



**REMARK** Only the DNC1000 local communication and the related requests and functions are covered in this chapter. Refer to the UNITE Protocol Manual for communication with remote stations and to the end of this chapter for the communication functions.

### 15.1.1 General Description of the DNC1000 Protocol

DNC1000 is a local communication procedure between the automatic control function and the other system functions. It allows transmission of data that are inaccessible via the transfer area.

Communication is between a requester (or client) and a server. It uses UNITE requests. Generally, the automatic control function is the requester and the CNC function is the server.

The part programme can also initiate a transfer to the automatic control function (unsolicited data).

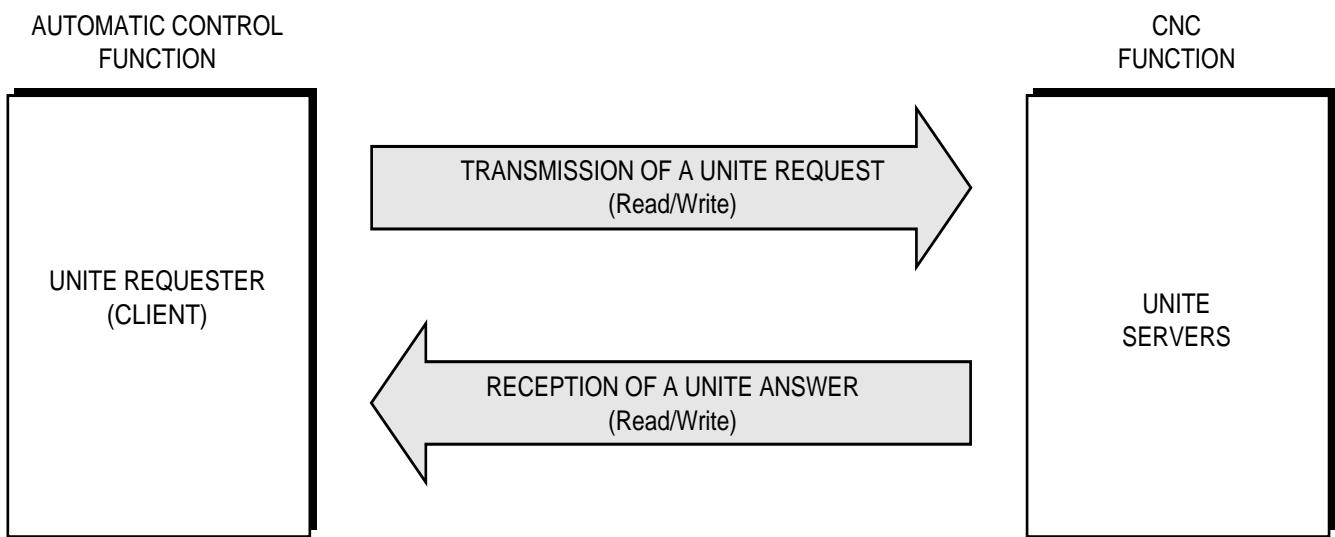


Figure 15.1 - Exchange by protocol

## 15.1.2 DNC1000 Exchange Mechanism

### 15.1.2.1 Request Processing Procedure

The automatic control function (requester) sends a (Read/Write) request to the server. This request, placed in a buffer, is queued. It is then processed by the server which sends an answer code and possibly data to the requester. The answer and data are recovered by the user programme.

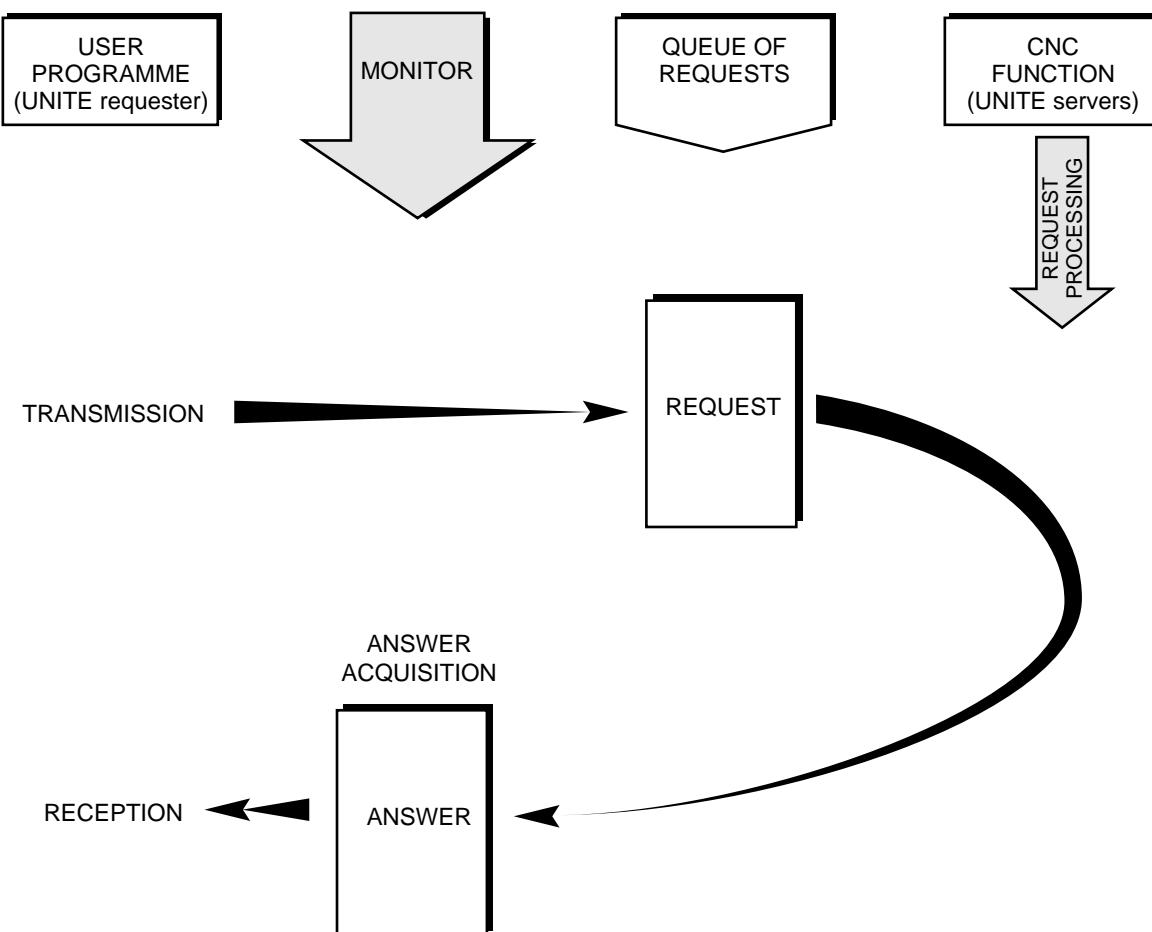


Figure 15.2 - Request processing principle

### 15.1.2.2 Concept of Port

The requester must associate each request with a port.

The same port is used for transmission of a request and reception of the answer.

Two types of port can be accessed:

- 16 ports with addresses from 0x30 to 0x3F allow the requester to send several requests in parallel,
- 8 ports with addresses from 0x10 to 0x17, associated with axis groups 1 to 8, are used to receive unsolicited data from the part programme (\$1 in the part programme).

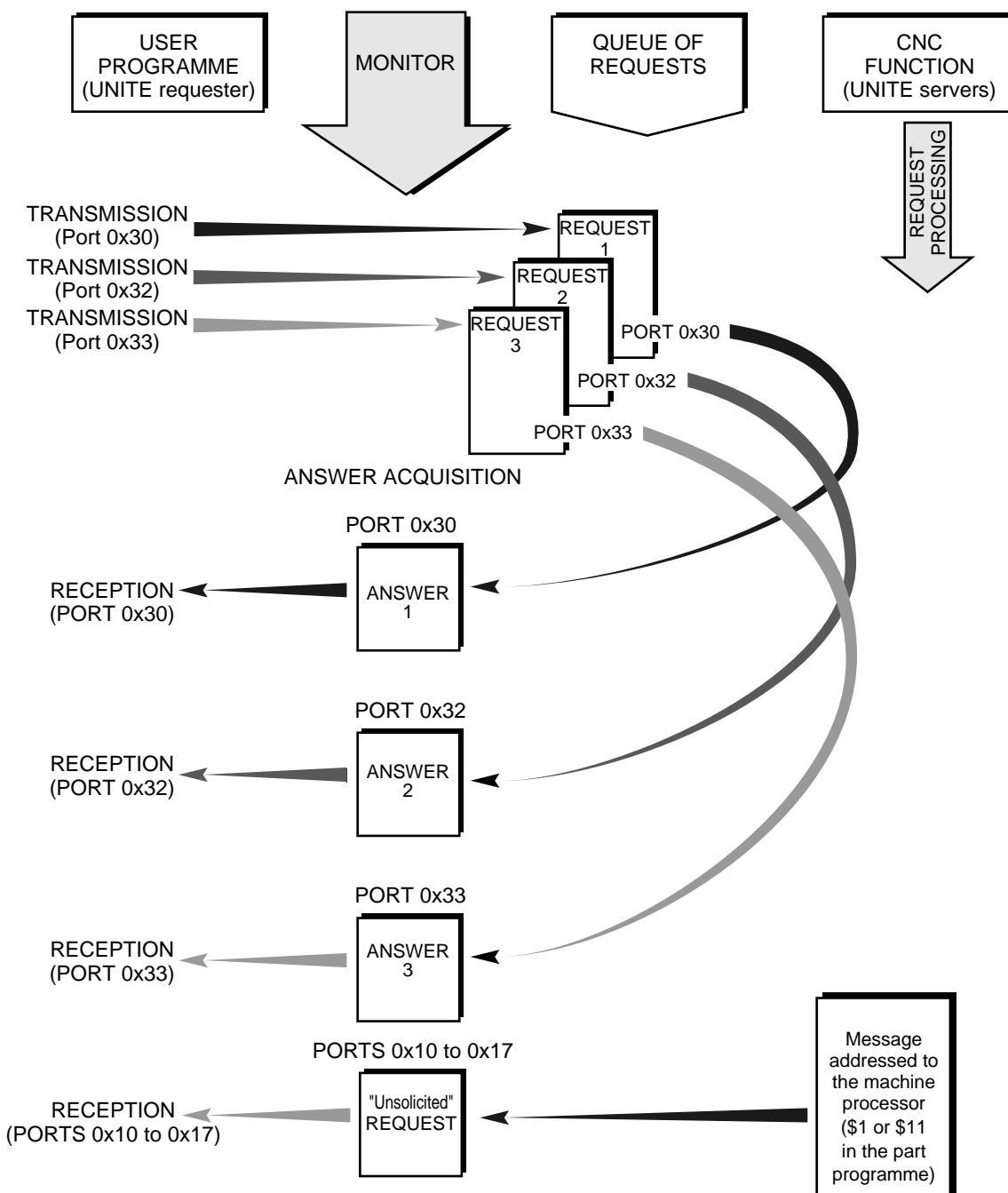


Figure 15.3 - Use of ports

## 15.2 Objects Accessible by a UNITE Request

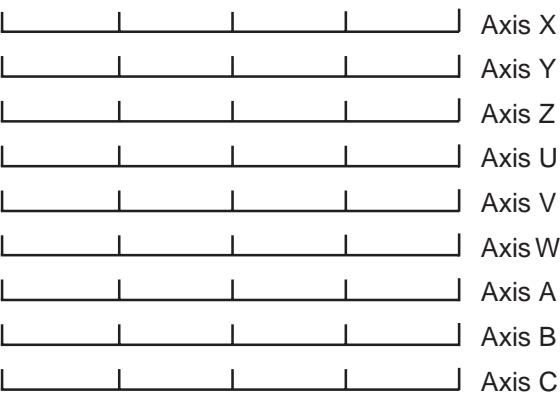
### 15.2.1 List of Object Type Requests Processed by the CNC Processor

Family name	Segment No.	Accessible for		Name	Object definition	
		Read	Write		Size	Maximum number of objects in the family
Axis position reference	128	X		Programme axes	9 longs mots	8 (1 per axis group)
Axis measurement	129	X		Programme axes	9 long words	8 (1 per axis group)
DAT1	130	X	X	Programme axes	9 long words	8 (1 per axis group)
DAT2	131	X	X	Programme axes	9 long words	8 (1 per axis group)
DAT3	132	X	X	Programme axes	9 long words	8 (1 per axis group)
Minimum machining limit	133	X	X	Programme axes	9 long words	8 (1 per axis group)
Maximum machining limit	134	X	X	Programme axes	9 long words	8 (1 per axis group)
Axis inclination	135	X	X		1 long word	32 (1 per axis)
Machine origin	136	X	X	Physical axes	1 long word	32 (1 per axis)
Min. machine travels	137	X	X	Physical axes	1 long word	32 (1 per axis)
Max. machine travels	138	X	X	Physical axes	1 long word	32 (1 per axis)
Axis reference correction	139	X		Physical axes	1 long word	32 (1 per axis)
Axis position reference	140	X		Physical axes	1 long word	32 (1 per axis)
Measured axis position	141	X		Physical axes	1 long word	32 (1 per axis)
Servo controlled axis	143	X		Axes present	1 long word	1 (1 bit per axis)
Measured spindle speed	144	X		Spindles	1 long word	4 (1 per spindle)

Family name	Segment No.	Accessible for		Name	Object definition	
		Read	Write		Size	Maximum number of objects in the family
Measured spindle position	145	X		Spindles	1 long word	4 (1 per spindle)
Tool correction	146	X	X	Tools	7 long words	255
Variable H	147	X	X	Tool use time	1 long word	255
Interpolation status	148	X		Interpolation status	4 long words	8 (1 per axis group)
Axes initialised	149	X	X	Axes present	1 long word	1 (1 bit per axis)
Parameters E80000	150	X	X		1 long word	51
Parameters E81000	151	X	X		1 long word	Number declared in machine parameter P58
Parameters E82000	152	X	X		1 long word	Number declared in machine parameter P58
Programme status	153	X			22 bytes	8 (1 per axis group)
End of block dimensions	157	X			11 long words	8 (1 per axis group)
Mode selection	180	X	X		1 word	1
Current part programme selection	181	X	X		1 word	1
Data sent to the programme being executed	224	X	X		1 long word	8 (1 per axis group)
Message acknowledgement	226	X	X		1 byte	8 (1 per axis group)
IT line configuration	227	X	X		1 byte	8 (1 per line)
Axis synchronisation enable/cancel	235	X	X		1 long word	1 (1 bit per axis)

## 15.2.2 Object Constituents

The IU corresponds to the internal unit of the system defined by a machine parameter.

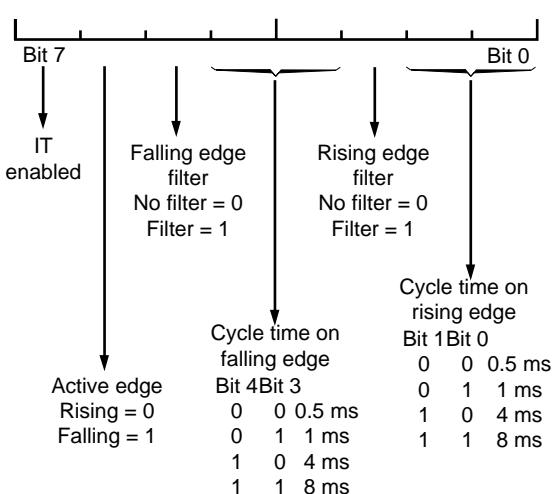
Segment No. (Hex)	Accessible for	Description	Value or unit	Corresponding parameters
128 (0x80)	Read	Axis position reference Object size: 9 x 4 bytes Address of the first object in the family: 0 	IU Or 1/10000°	E70000 to E78000
129 (0x81)	Read	Axis measurement Object size: 9 long words Address of the first object in the family: 0	-99999999 to 99999999 IU	E90000 to E90031
130 (0x82)	Read/Write	DAT1 Object size: 9 long words Address of the first object in the family: 0	-99999999 to 99999999 IU	E60000 to E68000
131 (0x83)	Read/Write	DAT2 Object size: 9 long words Address of the first object in the family: 0	-99999999 to 99999999 IU	E60001 to E68001
132 (0x84)	Read/Write	DAT3 Object size: 9 long words Address of the first object in the family: 0	-99999999 to 99999999 IU	E60004 to E68004
133 (0x85)	Read/Write	Minimum dynamic travel Object size: 9 long words Address of the first object in the family: 0	-99999999 to 99999999 IU	E60002 to E68002
134 (0x86)	Read/Write	Maximum dynamic travel Object size: 9 long words Address of the first object in the family: 0	-99999999 to 99999999 IU	E60003 to E68003

Segment No. (Hex)	Accessible for	Description	Value or unit	Corresponding parameters
135 (0x87)	Read/Write	Value of the angles for inclined axes Object size: 1 long word Address of the first object in the family: 0	1/10000°	E69001
136 (0x88)	Read/Write	Machine origin Object size: 1 long word Address of the first object in the family: 0	IU Or 1/10000°	Parameter P16
137 (0x89)	Read/Write	Minimum static travel Object size: 1 long word Address of the first object in the family: 0	IU	Parameter P17
138 (0x8A)	Read/Write	Maximum static travel Object size: 1 long word Address of the first object in the family: 0	IU	Parameter P17
139 (0x8B)	Read	Current corrections of a slave axis Object size: 1 long word Address of the first object in the family: 0	-99999999 to 99999999 IU	E95000 E95031
140 (0x8C)	Read	Reference position of an axis Object size: 1 long word Address of the first object in the family: 0	IU	E70000 to E78000
141 (0x8D)	Read	Measured position of an axis Object size: 1 long word Address of the first object in the family: 0	IU	E90000 to E90031
143 (0x8F)	Read	Slaved axis Object size: 1 long word Address of the first object in the family: 0	0 or 1	E91000 to E91031
144 (0x90)	Read	Measured spindle speed Object size: 1 long word Address of the first object in the family: 0	IU	
145 (0x91)	Read	Measured spindle position reference Object size: 1 long word Address of the first object in the family: 0	0 to 3599999 °/10000	E90101 to E90104

Segment No. (Hex)	Accessible for	Description	Value or unit	Corresponding parameters
146 (0x92)	Read/Write	<p>Turning tool corrections Object size: 7 long words Address of the first object in the family:</p> <p>Long word 1: Length in X Long word 2: Length in Z Long word 3: Insert radius Long word 4: Wear offset in X Long word 5: Wear offset in Z Long word 6: Tool tip direction Long word 7: Tool type</p>	<p>IU IU IU IU IU From 0 to 8 1 or 2</p>	<p>E50001 to E50255 E51001 to E51255 E52001 to E52255 E53001 to E53255 E54001 to E54255 E55001 to E55255 E57001 to E57255</p>
146 (0x92)	Read/Write	<p>Milling tool corrections Object size: 7 long words Address of the first object in the family: 1</p> <p>Long word 1: Tool length Long word 2: Cutter tip radius Long word 3: Tool radius Long word 4: Length wear offset Long word 5: Radius wear offset Long word 6: Not significant Long word 7: Tool type</p>	<p>IU IU IU IU IU</p>	<p>E50001 to E50255 E51001 to E51255 E52001 to E52255 E53001 to E53255 E54001 to E54255 0 E57001 to E57255</p>
147 (0x93)	Read/Write	<p>Parameters available (H of the table of dynamic corrections) Object size: 1 long word Address of the first object in the family: 1</p>	-99999999 to 99999999	E56001 to E56255

Segment No. (Hex)	Accessible for	Description	Value or unit	Corresponding Parameters
148 (0x94)	Read	<p>Interpolation status Object size: 4 long words Address of the first object in the family: 0</p> <p>Long word 1: Current feed rate Long word 2: Distance remaining to be travelled on the current block (in the path) Long word 3: Programmed feed rate Long word 4: Feed rate override coefficient</p>	$\mu\text{m}/\text{Sample}$ $\mu\text{m}$ $\text{mm}/\text{min}$ $\text{mm}/\text{rev}, \text{V/D}$ $2^{-16}$	
149 (0x95)	Read/Write	<p>HOME not set for an axis Object size: 1 long word Address of the first object in the family: 0</p>	1 or 0	E91100 to E91131
150 (0x96)	Read/Write	<p>Local data parameter Object size: 1 long word Address of the first object in the family: 0</p>	-99999999 to 99999999	E80000 E80050
151 (0x97)	Read/Write	<p>Master axis position reference (interaxis calibration) Object size: 1 long word Address of the first object in the family: 0</p>	-99999999 to 99999999 IU	E81000 E81999
152 (0x98)	Read/Write	<p>Slave axis correction (interaxis calibration) Object size: 1 long word Address of the first object in the family: 0</p>	-99999999 to 99999999 IU	E82000 E82999
153 (0x99)	Read	<p>Programme status (See Sec. 15.2.3) Object size: 22 bytes Address of the first object in the family: 0</p> <p>1 long word: List of G functions present 1 long word: Active programme number 1 long word: Current block number 1 long word: Programme error number 1 long word: Incorrect block number 1 long word: Tool number 1 long word: Tool direction 1 long word: Tool corrector number 1 long word: List of operations remaining to be executed</p>		

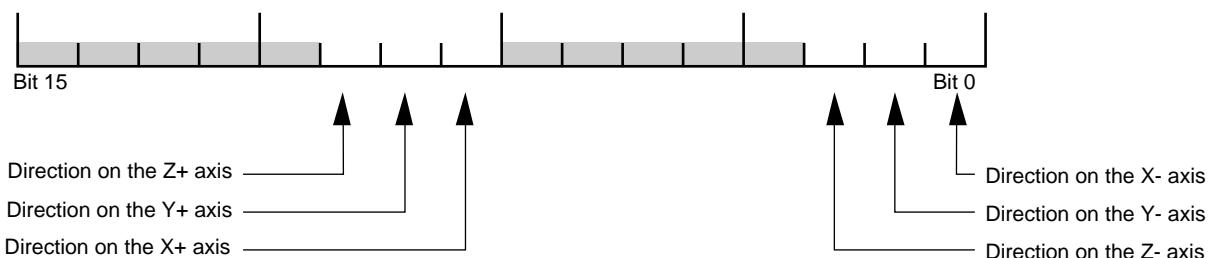
Segment No. (Hex)	Accessible for	Description	Value or unit	Corresponding parameters
157 (0x9D)		End of block dimension Object size: 11 long words Address of the first object in the family: 0 The first 36 bytes indicate the end of block dimensions (4 bytes per axis) for axes X, Y, Z, U, V, W, A, B, C The next long word indicates the x coordinate of the centre in circular interpolation. The last long word indicates the y coordinate of the centre in circular interpolation.		
180 (0xB4)	Read/Write	Mode selection Object size: 1 word Address of the first object in the family: 0  Auto mode Single mode MDI mode Dryrun mode Search mode Edit mode Test mode Manual mode Home mode Shift mode TLSET mode Load mode Unload mode  If the MSB (bit 15) is set, the mode change request is latched.	0x0000 0x0001 0x0002 0x0003 0x0004 0x0005 0x0006 0x0007 0x0008 0x0009 0x000A 0x000D 0x000F	E41000
181 (0xB5)	Read/Write	Current programme selection Object size: 1 word Address of the first object in the family: 0	1 to 99999	
224 (0xE0)	Read/Write	Data transmitted to the part programme being executed (see Sec. 15.3.11) Object size: 1 long word Address of the first object in the family: 0		
226 (0xE2)	Read/write	Acknowledgement of blocking messages sent by the part programme "\$11" (see Sec. 15.3.11) Object size: 1 byte Address of the first object in the family: 0		

Segment No. (Hex)	Accessible for	Description	Value or unit	Corresponding parameters																														
227 (0x E3)	Read/Write	<p>Configuration of the IT lines on the IT/serial line cards</p>  <table border="1"> <thead> <tr> <th>Bit 1</th> <th>Bit 0</th> <th>Cycle time on falling edge</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>0.5 ms</td> </tr> <tr> <td>0</td> <td>1</td> <td>1 ms</td> </tr> <tr> <td>1</td> <td>0</td> <td>4 ms</td> </tr> <tr> <td>1</td> <td>1</td> <td>8 ms</td> </tr> </tbody> </table> <table border="1"> <thead> <tr> <th>Bit 1</th> <th>Bit 0</th> <th>Cycle time on rising edge</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>0.5 ms</td> </tr> <tr> <td>0</td> <td>1</td> <td>1 ms</td> </tr> <tr> <td>1</td> <td>0</td> <td>4 ms</td> </tr> <tr> <td>1</td> <td>1</td> <td>8 ms</td> </tr> </tbody> </table>	Bit 1	Bit 0	Cycle time on falling edge	0	0	0.5 ms	0	1	1 ms	1	0	4 ms	1	1	8 ms	Bit 1	Bit 0	Cycle time on rising edge	0	0	0.5 ms	0	1	1 ms	1	0	4 ms	1	1	8 ms		
Bit 1	Bit 0	Cycle time on falling edge																																
0	0	0.5 ms																																
0	1	1 ms																																
1	0	4 ms																																
1	1	8 ms																																
Bit 1	Bit 0	Cycle time on rising edge																																
0	0	0.5 ms																																
0	1	1 ms																																
1	0	4 ms																																
1	1	8 ms																																
235 (0xEB)	Read/Write	Axis synchronisation enable/inhibit Object size: 1 long word (1 bit per axis)																																

### 15.2.3 Programme Status Segments

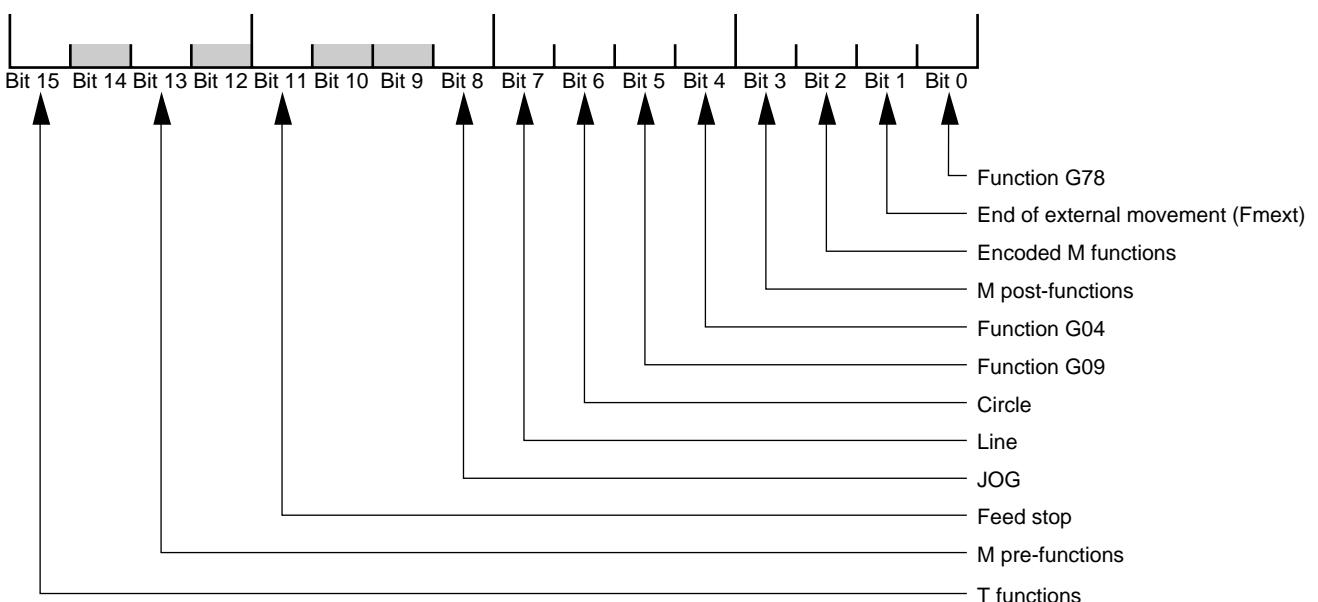
#### Tool Direction

The value of the tool direction is set in the low byte if it is negative or in the high byte if it is positive.



**Detailed List of G Functions Present****List of Operations Remaining to be Performed**

The bit with the highest weight designates the function being executed.



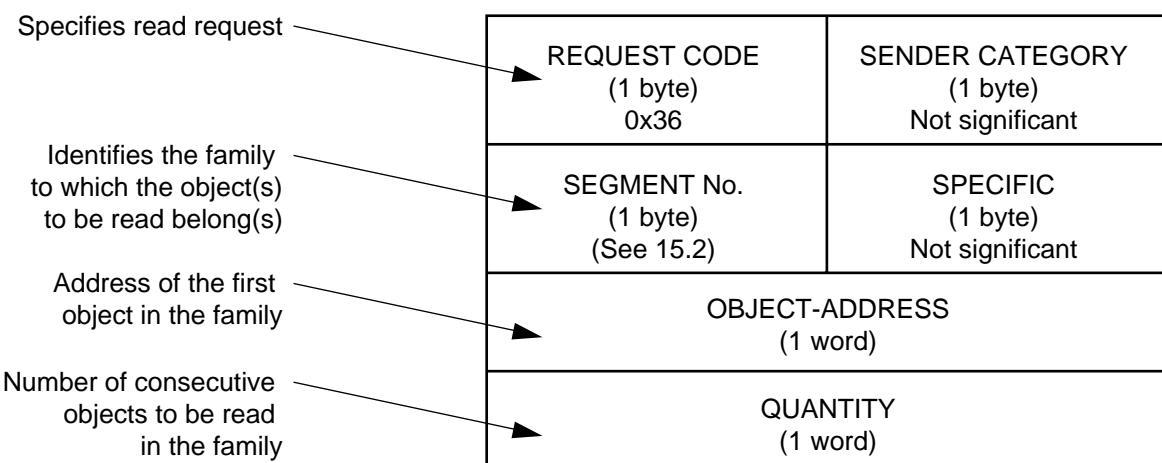
## 15.3 UNITE Requests Processed by the CNC Function

### 15.3.1 «READ-OBJECT» Request

#### Description

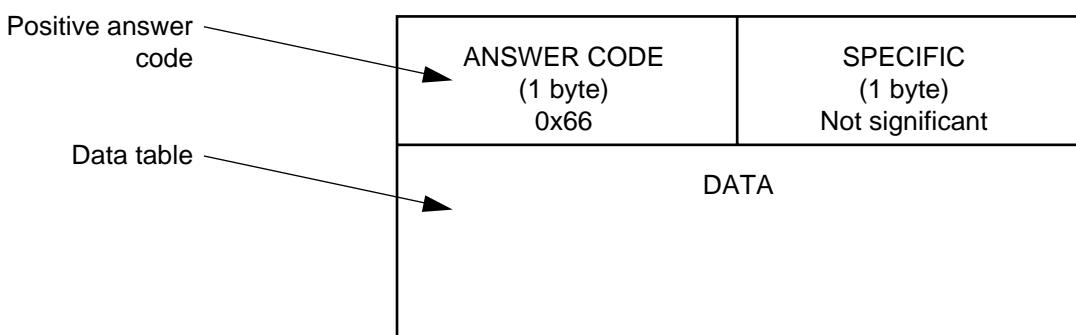
The READ-OBJECT request is used to read those objects accessible for read by the CNC server (See Sec. 15.2.2).

#### Request Format



#### Answer Format

##### Positive answer



##### Negative answer

ANSWER CODE (1 byte) 0xFD	X
---------------------------------	---

**REMARK** If the quantity specified is such that the answer could contain more than 128 bytes, the request is refused (negative acknowledgement).

**Example of reading the current programme number**Request sent

REQUEST CODE 0x36	SENDER CATEGORY 0x00
SEGMENT 0xB5	SPECIFIC 0x00
OBJECT-ADDRESS 0x0000	
QUANTITY 0x0001	

Positive answer with data

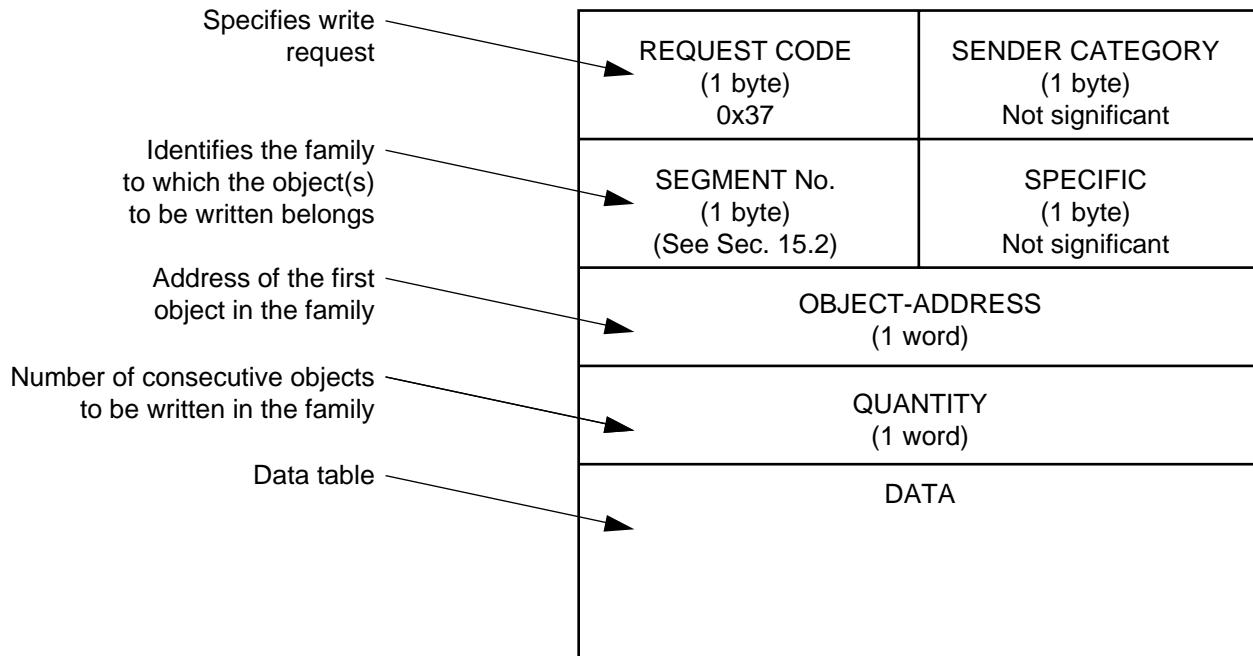
ANSWER CODE 0x66	SPECIFIC 0x00
DATA 0x0053 (Programme %83.)	

### 15.3.2 «WRITE-OBJECT» Request

#### Description

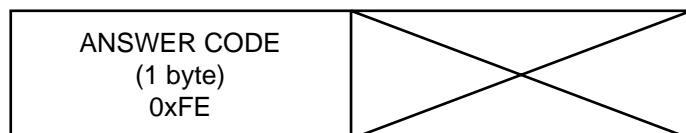
The WRITE-OBJECT request is used to write values to the CNC (See Sec. 15.2.2).

#### Request Format

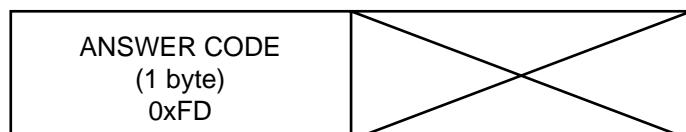


#### Answer Format

##### Positive answer



##### Negative answer



**REMARK** If the quantity specified is such that the request contains more than 128 bytes, the request is rejected (negative acknowledgement).

### 15.3.3 «DELETE-FILE» Request

Used to delete a part programme stored in the CNC RAM.

#### Request format

REQUEST CODE (1 byte) 0xF5	SENDER CATEGORY (1 byte) Not significant
REQUEST CODE COMPLEMENT (1 byte) 0x46	FILENAME (1 long word) byte 1
byte 2	byte 3
byte 4	

#### Detail of the «FILE NAME» field

The «FILE NAME» field indicates the number of the part programme indexed by the axis group (Part programme No. x 10 + axis group No.).

#### Answer format

##### Positive answer

ANSWER CODE (1 byte) 0xF5	ADDITIONAL ANSWER CODE 0x76
STATUS (1 byte) 0x00	

### Negative answer

ANSWER CODE (1 byte) 0xF5	ADDITIONAL ANSWER CODE 0x76
STATUS (1 byte) See table below	

Status code	Definition
0x02	Operation in programme area
0x05	File does not exist
0xA	CNC not in reset status

### **15.3.4 «READ-MEMORY-FREE» Request**

Used to display the number of bytes available in the CNC RAM.

#### **Request format**

REQUEST CODE (1 byte) 0xF5	SENDER CATEGORY (1 byte) Not significant
ADDITIONAL REQUEST CODE (1 byte) 0x47	

#### **Answer format**

##### Positive answer

ANSWER CODE (1 byte) 0xF5	ADDITIONAL ANSWER CODE 0x77
STATUS (1 byte) 0x00	VALUE (1 long word) byte 1
byte 2	byte 3
byte 4	

Negative answer

ANSWER CODE (1 byte) 0xF5	ADDITIONAL ANSWER CODE 0x77
STATUS (1 byte) See table below	X

---

Status code	Definition
0x02	Operation in programme area

---

**15.3.5 «OPEN-DIRECTORY» Request**

Lists all the part programmes present in the CNC RAM.

If the list is too long to be contained in the answer to this request (Status = 0x00), the additional items are given in answer to the «DIRECTORY» request (See Sec. 15.3.6). However, if this request is not made, the «CLOSE DIRECTORY» request (See Sec. 15.3.7) must be sent to end the operation.

If the answer to this request can contain the complete list (Status = 0x0C), the operation ends automatically and the «CLOSE DIRECTORY» request is not required.

**Request format**

REQUEST CODE (1 byte) 0xF5	SENDER CATEGORY (1 byte) Not significant
REQUEST CODE COMPLEMENT (1 byte) 0x48	X
FILE NAME (1 long word) See REMARK	

**REMARK** The «FILE NAME» field indicates the number of the first part programme indexed by the axis group (Part programme No.  $x \cdot 10 +$  axis group No.) to be included in the answer.  
 If this programme is not present in memory, the list is given starting from the next programme.  
 If the value of this field is 0, the list is given starting with the first programme present in memory.

## Answer format

### Positive answer

ANSWER CODE (1 byte) 0xF5	ADDITIONAL ANSWER CODE 0x78
STATUS (1 byte) See table below	
DATA (See REMARK)	

Status code	Definition
0x00	OK - Data still to be sent
0x0C	OK - End of directory (closed automatically)

**REMARK** In the «DATA» field, each part programme is described by two long words:

- the first indicates the part programme number indexed by the axis group (Part programme No.  $x 10 +$  axis group No.)
- the second indicates the number of bytes in the part programme.

### Negative answer

ANSWER CODE (1 byte) 0xF5	ADDITIONAL ANSWER CODE 0x78
STATUS (1 byte) See table below	

Status code	Definition
0x02	Operation in programme area
0x09	Buffer size insufficient for answer

## 15.3.6 «DIRECTORY» Request

Used to display the rest of the list of part programmes present in the CNC RAM following an «OPEN DIRECTORY» request.

If the list is too long to be contained in the answer to this request (Status = 0x00), the additional items can be given in the answer to another «DIRECTORY» request. However, if this request is not made, the «CLOSE DIRECTORY» request (See Sec. 15.3.7) must be sent to end the operation.

If the answer to this request can contain the complete list (Status = 0x0F), the operation ends automatically and the «CLOSE DIRECTORY» request is not required.

**Request format**

REQUEST CODE (1 byte) 0xF5	SENDER CATEGORY (1 byte) Not significant
ADDITIONAL REQUEST CODE (1 byte) 0x49	X

**Answer format**Positive answer

ANSWER CODE (1 byte) 0xF5	ADDITIONAL ANSWER CODE 0x79
STATUS (1 byte) See table below	X
DATA (See REMARK)	

Status code	Definition
0x00	OK - Data still to be sent
0x0C	OK - End of directory (closed automatically)

**REMARK** In the «DATA» field, each part programme is described by two long words:

- the first indicates the part programme number indexed by the axis group (part programme No.  $x$  10 + axis group No.)
- the second indicates the number of bytes in the part programme.

Negative answer

ANSWER CODE (1 byte) 0xF5	ADDITIONAL ANSWER CODE 0x79
STATUS (1 byte) See table below	X

Status code	Definition
0x02	Operation in programme area
0x09	Buffer size insufficient for answer

### 15.3.7 «CLOSE-DIRECTORY» Request

Used to close a directory operation.

#### Request format

REQUEST CODE (1 byte) 0xF5	SENDER CATEGORY (1 byte) Not significant
REQUEST CODE COMPLEMENT (1 byte) 0x4A	X

#### Answer format

##### Positive answer

ANSWER CODE (1 byte) 0xF5	ADDITIONAL ANSWER CODE 0x7A
STATUS (1 byte) 0x00	X

##### Negative answer

ANSWER CODE (1 byte) 0xF5	ADDITIONAL ANSWER CODE 0x7A
STATUS (1 byte) See table below	X

Status code	Definition
0x04	Directory already closed

### 15.3.8 «READ-BLOCK» Request

Used to read a part programme block.

#### Request format

REQUEST CODE (1 byte) 0xF5	ADDITIONAL REQUEST CODE (1 byte) 0x50
PROGRAMME NUMBER (1 long word) (See REMARK)	
BLOCK NUMBER (1 word)	
BLOCK OFFSET (1 word)	

**REMARK** The «PROGRAMME NUMBER» field indicates the number of the part programme indexed by the axis group (Part programme No. x 10 + axis group No.).

#### Answer format

##### Positive answer

ANSWER CODE (1 byte) 0xF5	ADDITIONAL ANSWER CODE 0x80
BLOCK LENGTH (1 word)	
DATA	

**REMARK** The «DATA» field can contain up to 119 bytes and ends with (LF).

##### Negative answer

ANSWER CODE (1 byte) 0xF5	ADDITIONAL ANSWER CODE 0x80
---------------------------------	--------------------------------

For this request, the answer code and additional answer code are not significant. The execution report is given by the code returned in the uniti() function (see Sec. 15.4.2).

### 15.3.9 «WRITE-BLOCK» Request

Used to add, edit or delete a part programme block. By allocating more memory space than is actually required for this programme prior to using this request (RESERVE MEMORY request, See Sec. 15.3.10) a write-block request can be made when the CNC is not in reset status.

**REMARK** *The request can contain up to 132 bytes.*

#### Request format

REQUEST CODE (1 byte) 0xF5	ADDITIONAL REQUEST CODE (1 byte) 0x51
PROGRAMME NUMBER (1 long word) See REMARK 1	
BLOCK NUMBER (1 word)	
BLOCK OFFSET (1 word)	
BLOCK LENGTH (1 word)	
DATA See REMARK 2	

**REMARK 1:** The «PROGRAMME NUMBER» field indicates the number of the part programme indexed by the axis group (Part programme No. x 10 + axis group No.).

**REMARK 2:** The first character must be:

- «+» for an addition after the block given by number and offset,
- «#» for a modification,
- «-» for a deletion.

*The last character must be «LF» and the field must contain no more than 119 characters.*

#### Format of Positive and Negative Answers

ANSWER CODE (1 byte) 0xF5	ADDITIONAL ANSWER CODE 0x81
---------------------------------	-----------------------------------

For this request, the answer code and additional answer code are not significant. The execution report is given by the code returned in function uniti() (see Sec. 15.4.2).

### 15.3.10 «RESERVE-MEMORY» Request

Used to reserve memory space for an existing part programme in order to be able to edit this programme when the CNC is not in reset status.

#### Request format

REQUEST CODE (1 byte) 0xF5	ADDITIONAL REQUEST CODE (1 byte) 0x52
PROGRAMME NUMBER (1 long word) See REMARK 1	
MEMORY SIZE (1 long word) See REMARK 2	

**REMARK 1:** The «PROGRAMME NUMBER» field indicates the number of the part programme indexed by the axis group (Part programme No.  $x 10 +$  axis group No.).

**REMARK 2:** The «MEMORY SIZE» field indicates the size to be reserved for the programme. Setting this field to zero allocates the effective memory size required for the programme.

#### Format of Positive and Negative Answers

ANSWER CODE (1 byte) 0xF5	ADDITIONAL ANSWER CODE 0x82
---------------------------------	-----------------------------------

For this request, the answer code and additional answer code are not significant. The execution report is given by the code returned in function uniti() (see Sec. 15.4.2).

### 15.3.11 «MESSAGE READ» Request

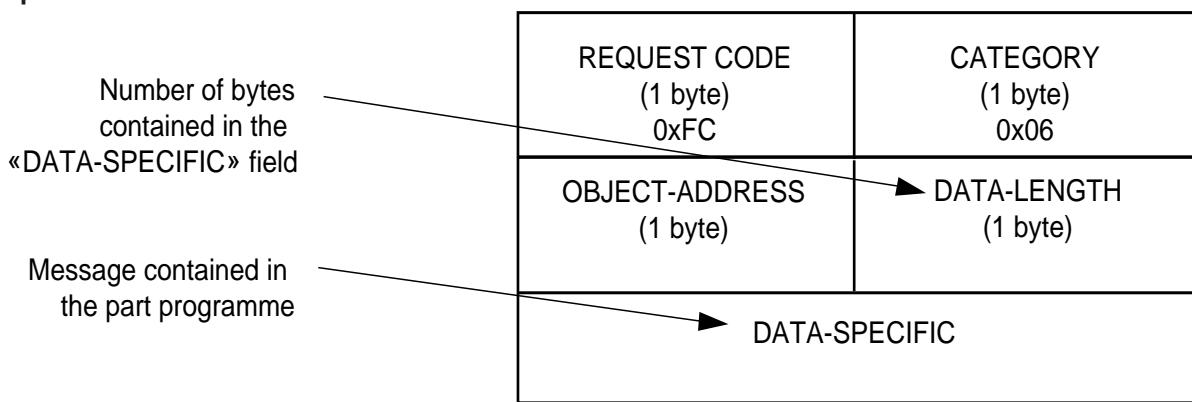
The CNC function sends this request at its own initiative to the automatic control function after instruction «\$1» or «\$11» in the part programme being executed.

Two types of messages are sent by the CNC server:

- nonblocking messages «\$1 «Message» LF»,
- blocking messages «\$11 «Message» Lf».

The automatic control function receives the message sent to it by the monitor on ports 0x10 to 0x17 (axis groups 1 to 8) using the uniti() function.

#### Request format



#### Transmission of a Nonblocking Message

This request does not wait for an answer from the automatic control function. It may however be associated with another request initiated by the automatic control function which is used to acknowledge the unsolicited data. The mechanism used to wait for the acknowledgement is described below.

The instruction «Ln = \$1» programmed in the part programme is used to retrieve an answer from the automatic control function.

#### Transfer mechanism

The message «\$1 «Message» LF» is sent once to the automatic control function. The part programme then continues without waiting for an acknowledgement.

The answer is transmitted by a «WRITE OBJECT» request with segment 224. The value of the object is stored by the CNC function until it is read by the part programme.

The part programme retrieves this value by function \$1 programmed in a parametric expression of the type «Ln = \$1».

If no value is sent or if the last value sent was already acknowledged, the part programme goes on wait for a new write of segment 224.

For read, the CNC sends back the last value sent to it if it is still stored, i.e. if it has not yet been retrieved by the part programme. Otherwise, it sends the one's complement of this value.

**REMARK** *An attempt to transfer a message from a part programme to the automatic control function cancels storage of a previous write of segment 224 on the axis group considered.*

### Transmission of a Blocking Message

After sending a blocking message «\$11», the CNC function goes on wait for an acknowledgement from the automatic control function. The CNC retransmits the message every 10 s and the part programme remains on wait until the write request indicating the acknowledgment is received.

#### Transfer Mecanism

The message «\$11 «Message» LF» is sent to the automatic control function and the part programme is on wait.

The message is then retransmitted every 10 s to the automatic control function until the automatic control function sends a «WRITE OBJECT» request with segment 226 = 1 (acknowledgement without CNC release).

The answer is sent by a «WRITE OBJECT» request with segment 224.

The automatic control function acknowledges the message by sending a «WRITE OBJECT» request with segment 226 = 2 (acknowledgement and release of the CNC).

Release of the CNC allows continuation to the next block and retrieval of the answer by function \$11 programmed in a parametric expression of the type «Ln = \$11».

## 15.4 Programming the General Request Function

### 15.4.1 Send a Request

**unito**

#### Syntax

<b>unito (source_port, &amp;datagram)</b>
---

source\_port: Number of the source port.

&datagram: Address of the buffer to be sent.

#### Description

Sends a request to a server on the 16 source ports with addresses 0x30 to 0x3F.

#### Return code

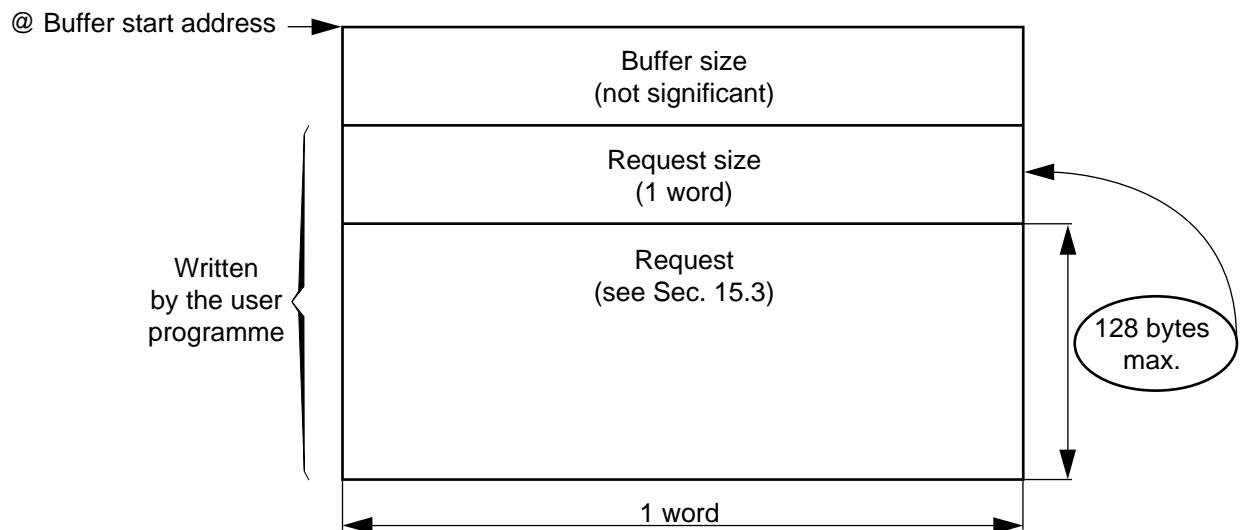
##### If OK

Code	Report message
0x00	Transmission correct

##### Error

Code	Report message
0x01	Buffer too long
0x02	Zero buffer length
0x03	Queue saturated - the 16 buffers are full
0x04	Incorrect port number
0x05	Option not valid for this request
0xFF	Not in a background task

### Structure of the transmission Buffer



### Programming error causing a CPU fault

Access to a prohibited address:

- &datagram parameter error,
- &datagram+size outside authorised area.

#### 15.4.2 Read an Answer



##### Syntax

**uniti (source\_port, &datagram)**

**source\_port:** Number of the source port.

**&datagram:** Address of the memory block which will receive the request.

##### Description

The uniti function works on ports 0x30 to 0x3F and 0x10 to 0x17.

This function:

- Receives the answer to a request previously sent by unito() on the same source port
- Receives an «unsolicited» request sent by the part programme of an axis group. In this case, the source port indicates the axis group from which the message is to be received (0x10 to 0x17).

**REMARKS** If the return code is 0x06, the uniti() function must be called periodically until the message is received.

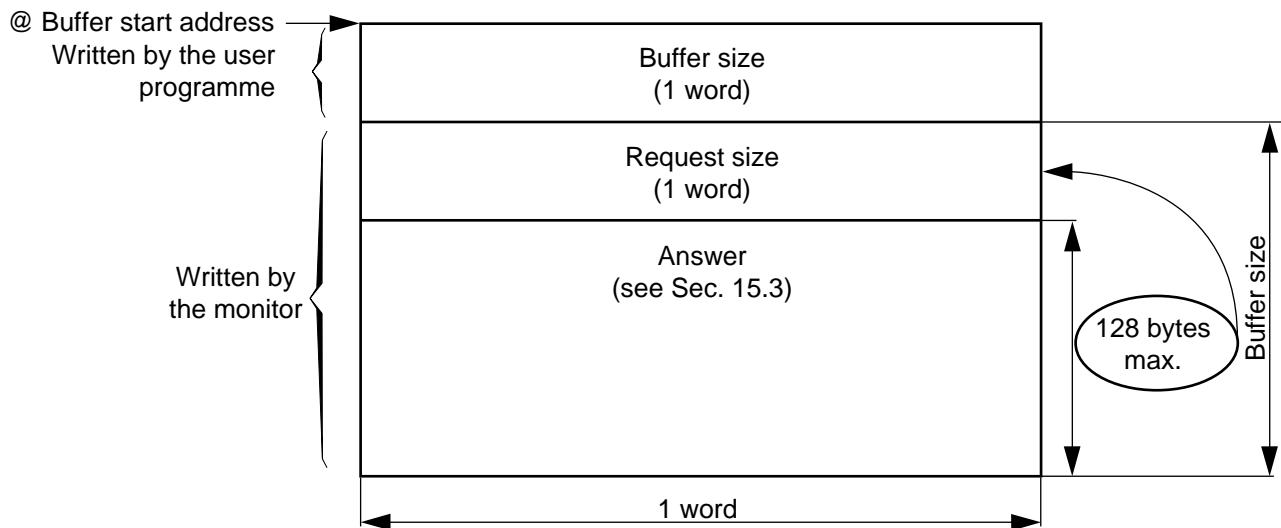
**Return code**If OK

Code	Report message
0x00	Read correct

Error

Code	Report message
0x04	Incorrect port number
0x06	No message at this port number
0x07	Buffer too small to store the answer
0x81	Programme number already exists
0x82	Programme printout in progress
0x83	Part programme area saturated
0x84	File closed
0x85	Programme number does not exist
0x86	File open
0x87	Saturation of buffer PPP
0x88	Header error Segment not recognised Write request prohibited «Quantity» zero or negative «Object-address» negative «Quantity» + «Object address» higher than the maximum number of terminals
0x89	Buffer size insufficient
0x8A	CNC status incompatible with the exchange
0x8B	Data exchanged incoherent
0x8F	Automatic close indicator
0xFF	Not in a background task

### Storage of the reception buffer



### Programming error causing a CPU fault

Access to a prohibited address:

- &datagram parameter error,
- &datagram+size outside authorised area.

#### 15.4.3 Programming Rules

For rational organisation of a programme including exchanges by protocol, follow the recommendations given below.

 **CAUTION**

Functions `unito()`, `uniti()`, `neto()` and `neti()` must be programmed in a background task (%TF0 to %TF15).

The size of a request must not be zero or greater than 128 bytes (except for the READ\_BLOCK and WRITE\_BLOCK requests).

**REMARK** *Sixteen exchange buffers are accessible to the user programme, allowing a maximum of sixteen requests to be sent each RTC cycle.*

Consecutive transmission of several requests on the same port without acknowledgement of the answers causes the loss of one or more answer codes.

**It is important to:**

- programme transmission to the server followed by reception of the answer on a given port before programming any other transmission, or
- use different ports for each transmission/reception command.

For the servers, the concept of queuing may cause an offset in the timing of request processing. This is because the load on the different servers may mean that the requests are not processed in the order in which they were sent.

To ensure correct timing of request processing, make sure the answer is received before sending the next request on a port.

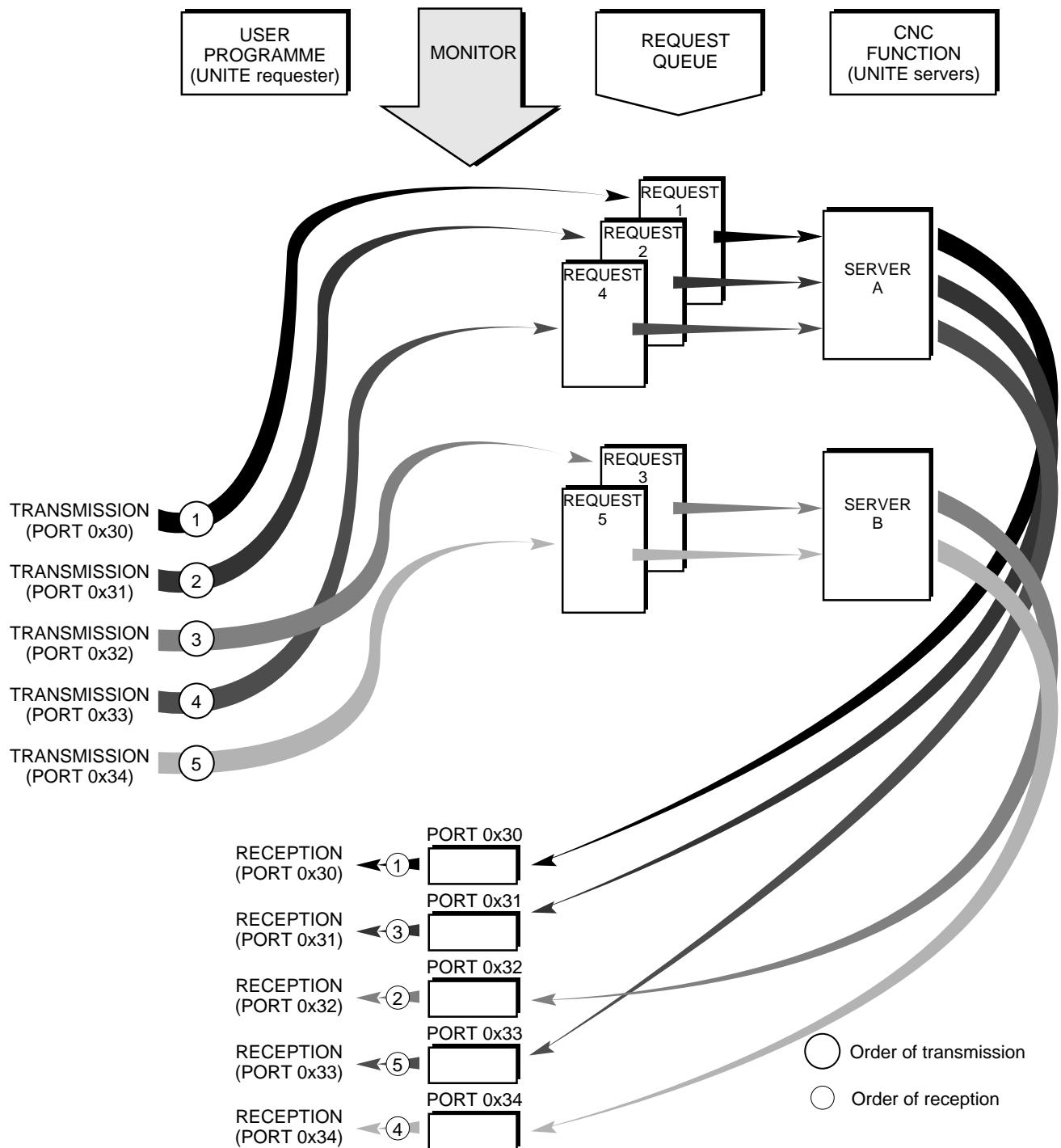


Figure 15.4 - Request processing by the servers

## 15.5 Exchanges With a Remote Station

### 15.5.1 Send a Request

**neto**

#### Syntax

```
neto (source_port, &datagram)
```

source\_port: Number of the source port.

&datagram: Address of the buffer to be sent.

#### Description

Sends a request to a remote station. The request is sent on one of the 16 source ports with addresses 0x50 to 0x5F.

#### Operation

If the source port number is valid (between 128 and 143) when the neto(..) function is called and if series 7 addressing (network, station, port, module, channel) is valid, the monitor:

- sends the request to the destination server and returns to the caller with an OK return code if the transmit channel is free,
- returns to the caller with the SATURATION return code if the transmit channel is saturated.

If a programming error is detected, a return is made to the caller with the return code indicating the error detected.

As many requests can be exchanged simultaneously as there are source ports available.

#### Return Code

##### If OK

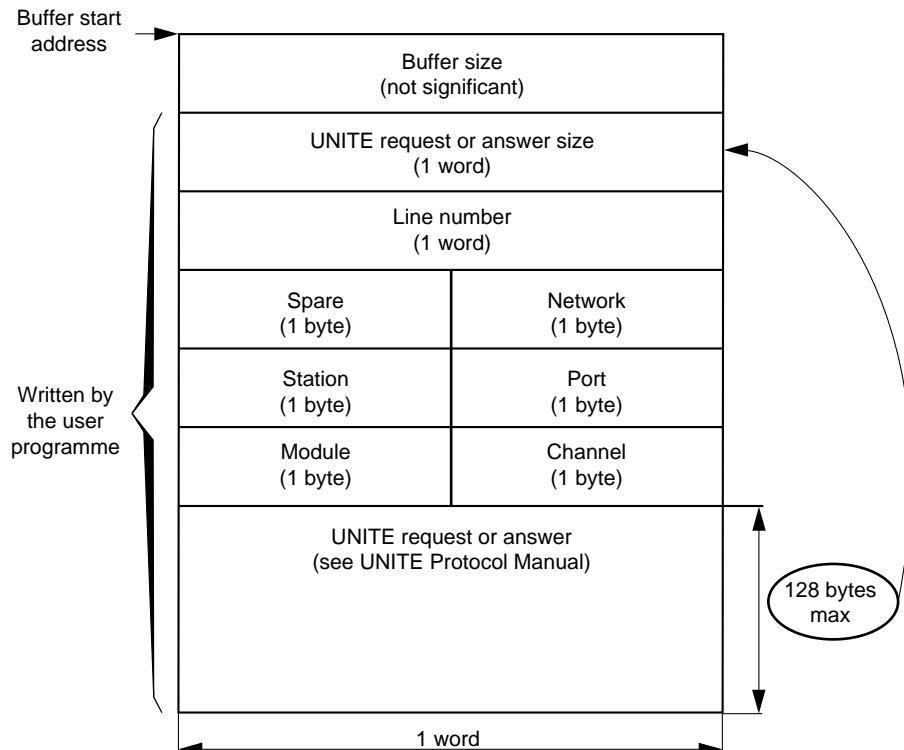
Code	Report message
0x00	Transmission correct

##### Error

Code	Report message
0x01	Buffer too long
0x02	Zero buffer length
0x03	Queue saturated - The 16 buffers are full
0x04	Incorrect port number
0x05	Option not valid for this request
0x08	Line number invalid
0xFF	Not in a background task

**REMARK** The Network, Station, Port, Module and Channel fields correspond to the Telemecanique Series 7 addressing identifying the request recipient. Refer to the manual on the corresponding network.

### Structure of the Transmission Buffer



Line No.	Machine processor card	First IT/serial line card	Second IT/serial line card	Special controller
UNI-TELWAY	0x20 and 0x21	0x24 to 0x27	0x28 to 0x2B	
MAPWAY - ETHWAY				0x30
ETHERNET				0x40

### Programming error causing a CPU fault

Access to a prohibited address:

- &datagram parameter error,
- &datagram+size outside authorised area.

### 15.5.2 Read an Answer

**neti**

#### Syntax

```
neti (source_port, &datagram)
```

**source\_port:** Number of the source port.

**&datagram:** Address of the memory block for receiving the answer.

## Description

The neti function works on ports 0x50 to 0x5F.

This function:

- Receives the answer to a request sent earlier on the network by neto(..)
- Receives an «unsolicited» request sent by a remote station.

## Operation

When the neti(..) function is called, if the queue does not contain any requests with the same source port as in the call to neti(..), the monitor returns to the calling programme with the NO\_REQ return code «0x6».

If the size allocated for reception of the datagram is sufficient, the request is transferred to the address &datagram and a return is made to the caller with the OK return code «0x0».

If the buffer size is insufficient, the return is made to the caller with the SMALL\_BUFF return code «0x7».

It is possible to wait simultaneously for as many requests as there are source ports (16 answers to requests sent by neto(..) or «unsolicited» requests).

## Return Code

### If OK

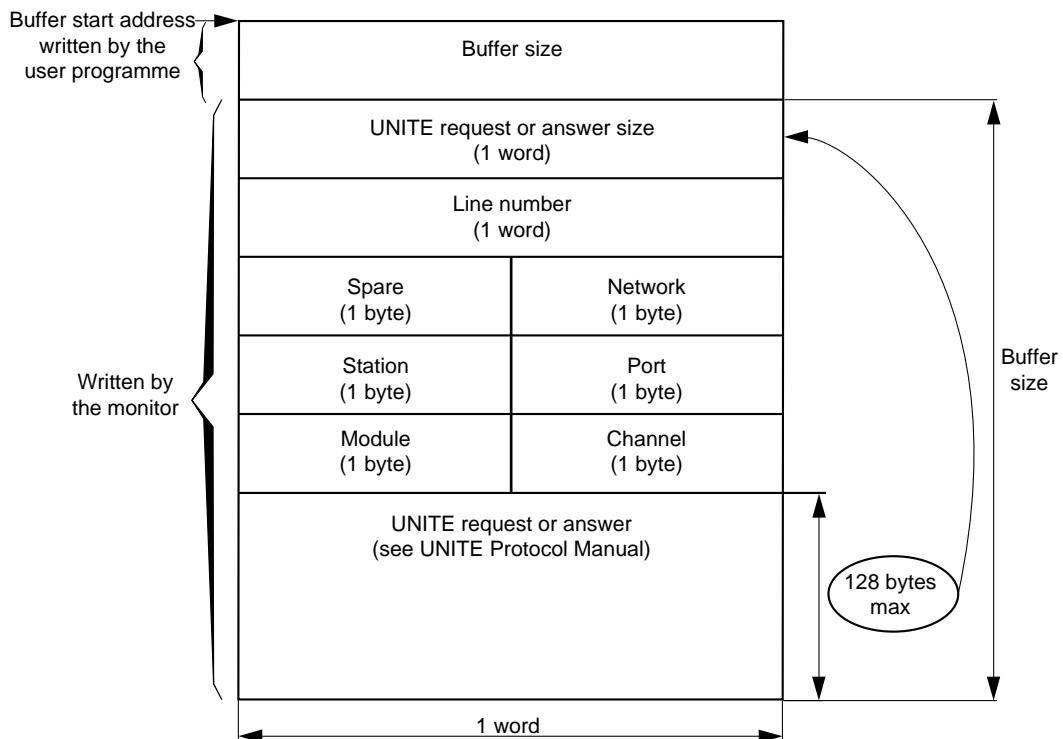
Code	Report message
0x00	OK - Read correct

### Error

Code	Report message
0x04	Incorrect port number
0x06	No request received for this port
0x07	Buffer too small to store the answer
0x08	Invalid line number
0xFF	Not in a background task

**REMARK** *If the return code is 0x06, the neti(..) function must be called cyclically until the request is received.*

### Structure of the Reception Buffer



**REMARK** The Network, Station, Port, Module and Channel fields correspond to the Telemecanique Series 7 addressing identifying the request recipient. Refer to the manual on the corresponding network.

Line No.	Machine processor card	First IT/serial line card	Second IT/serial line card	Special controller
UNI-TELWAY	0x20 and 0x21	0x24 to 0x27	0x28 to 0x2B	
MAPWAY - ETHWAY				0x30
ETHERNET				0x40

**REMARK** With UNI-TELWAY, if the answer is 2 bytes and the return code is 0xFF, the return code can have the following values:

- 0x03: recipient inaccessible
- 0x04: NACK: recipient buffer full
- 0x0A: timeout.

### Programming error causing a CPU fault

Access to a prohibited address:

- &datagram parameter error,
- &datagram+size outside authorised area.

### 15.5.3 Examples of Series 7 Addressing

SOURCE Requester	DESTINATION Server	Line No.	Spare	Network	Station	Port	Module	Channel
UNI-TELWAY Master Line L (16)	UNI-TELWAY Slave S	0x20+L	0	0	0xFE	5	0xFE	0x64+S
UNI-TELWAY Slave Line L (1)	UNI-TELWAY Master	0x20+L	0	0	0xFE	0	0	0
UNI-TELWAY Slave Line L (1)	UNI-TELWAY Master Port P application programme	0x20+L	0	0	0xFE	P	0	0
UNI-TELWAY Slave Line L (1)	UNI-TELWAY Slave S	0x20+L	0	0	0xFE	5	0xFE	0x64+S
MAPWAY-ETHWAY (16)	MAPWAY-ETHWAY Station S of my network	0x30	0	0	S	0	0	0
MAPWAY-ETHWAY (16)	MAPWAY-ETHWAY Station S of network R	0x30	0	R	S	0	0	0
MAPWAY-ETHWAY (1)	MAPWAY-ETHWAY Port P application programme Station S of network R	0x30	0	R	S	P	0	0

(1) 1 source port (0x50 to 0x5F) authorised per destination allows only one exchange with this destination.

(16) 16 source ports (0x50 to 0x5F) authorised per destination allows 16 simultaneous exchanges with the same destination.

*REMARK Only the server function is available for ETHERNET-MMS.*

**setcomw**

15

**15.5.4 Setting up the Common Word Service****Syntax****setcomw(size, activity)**

size: Number of bytes allocated to each station.

activity: Station activity with respect to the common words (0: inactive; 1: read/write; 2: read only).

**Description**

Sets up the common word service.

**Operation**

Function setcomw(..) must be called in task %INI. If this function is not called, the common word service is not active.

The same number of bytes must be allocated to each station in the network. If the value set for a station is incorrect, this station is ignored by the other stations.

Since the stations can share a maximum of 256 words (512 bytes), the size chosen determines the maximum number of stations that can use the common word service.

Size per station	Max No. of stations	Station addresses
8 bytes	64	0 to 0x3F
16 bytes	32	0 to 0x1F
32 bytes	16	0 to 0xF
64 bytes	8	0 to 7
128 bytes	4	0 to 3

An inactive station cannot send common words and cannot read those sent by the other stations.

A station active for read cannot send common words but can read the common words sent by the other stations.

A station active for read/write can send common words and read those sent by the other stations.

The stations declared active for read/write must have the lowest addresses on the network.

The number of common words configured can be less than the maximum authorised per station. This possibility should be used when the volume of information to be sent is small. Processing of the common words by the automatic control function is thereby improved.

**Return Code**If OK

Code	Report message
0x00	Setup OK

## Error

Code	Report message
0x01	Size parameter error
0x02	Activity parameter error
0x03	Size incompatible with the station address (if activity == 1)
0x04	Network processor error
0x05	Network processor in test mode

### 15.5.5 Answer to the STATU Request

### **netst\_ad**

#### Syntax

```
netst_ad(&status_address)
```

&status\_address      Address of the first byte in the user status area.

#### Description

Defines the address of the user area containing the specific values concerning the status of the numerical control accessible by the STATUS request (code 0x31) (see UNITE Protocol Manual).

#### Operation

This function must be called in task %INI.

It defines the start address for storage of the 16 bytes of the USER\_SPECIFIC field.

*REMARK If this function is not called, the USER\_SPECIFIC field is not significant.*

#### Example

```
netst_ad(%M100.&)
```

The STATUS request reads the USER\_SPECIFIC field at address %M100.&.

#### Return Code

##### If OK

Code	Report message
0x00	Setup OK

#### Programming error causing a CPU fault

Access to a prohibited address:

- &status\_address parameter error,
- end of status field outside authorised area.

# 16 Programming in C Language

16

<b>16.1 General</b>	16 - 3
<b>16.2 Call Executable Module</b>	<b>exec</b> 16 - 3
<b>16.3 Identify an Executable Module</b>	<b>exechdl</b> 16 - 4
<b>16.4 Programming in C</b>	16 - 5
16.4.1 Concept of Module	16 - 5
16.4.2 Interface between C Modules	16 - 5
16.4.3 Exchange Area	16 - 7
16.4.4 Accessing the Saved Common Internal Variables	16 - 7
16.4.5 Accessing Unsaved Common Internal Variables	16 - 7
16.4.6 Accessing the Card Inputs	16 - 8
16.4.7 Accessing the Card Outputs	16 - 8
16.4.8 Standard Data Types	16 - 8
16.4.9 Library Functions	16 - 9
16.4.9.1 System Functions	16 - 9
16.4.9.2 Using System Functions	16 - 11
16.4.9.3 Functions for Exchanges by Protocol	16 - 12
16.4.9.4 Managing the Serial Lines	16 - 13
16.4.9.5 Managing the Transparent Mode	16 - 14
16.4.9.6 Programming the Analogue Inputs/Outputs	16 - 20
16.4.9.7 Explicit Read/Write	16 - 21
16.4.9.8 Programming the Interrupt Inputs	16 - 22
16.4.9.9 Managing Background Tasks	16 - 22
16.4.9.10 General Purpose Functions	16 - 23
16.4.9.11 File Management	16 - 26
16.4.9.12 Directory Management	16 - 30



## 16.1 General

Programming the automatic control function in C has the following advantages:

- the programmes are structured (use of explicit variable names, language syntax, data structure, character strings, etc.)
- libraries of functions can be used (management of character strings, arithmetic, etc.)
- the user can write function libraries available for use by several applications.

All the tasks can be written in C.

All the variables of the exchange area are accessible to programmes written in C.

Functions exec(..) and exechdl(..) are used to manage the C executables resulting from the C compiler in a ladder module.

 **CAUTION**

When a pointer is used, it must be correctly initialised. If this check is not made, there is a risk of generating a "prohibited address" fault.

For instance, the return code of the MALLOC function must be tested before using the address returned by the function.

## 16.2 Call Executable Module

**exec**

### Syntax

**exec(whexec, {arg }6 )**

whexec: Logic identifier of the executable module to be called.

arg: Arguments (if any), extended on 32 bits and passed via the stack.

Calls an executable module generated from a C-language compiler.

### Principle

The (signed) arguments arg are extended on 32 bits and stacked in accordance with the C-language convention (the first argument is at the top of the stack). whexec is not stacked.

The logic identifier whexec is provided by function exechdl().

The system analyses whexec and calls the associated executable module.

### Return code

#### If OK

The value returned by the C-language executable module.

## 16.3 Identify an Executable Module

**exechdl**

### Syntax

```
exechdl(&string)
```

&string: Address of a string ending with a ZERO byte.

Reads the logic identifier of a function of an executable module from a C-language compiler.

### Principle

The string addressed by &string contains the name of a C-language function.

To be recognised by the monitor, the function name must be sent from a C-language module by function `export()` (see Sec. 16.4.9).

`exechdl()` must be called in the %INI task.

### Return code

#### If OK

`whexec > 0:` Logic identifier of the executable module (value given on 16 bits). This identifier is used to call the executable module by function `exec()`.

#### Error

`whexec == 0:` The system does not know the name defined by &string.

### Programming error causing a CPU fault

Access to a prohibited address:

- &string parameter error,
- end of string outside authorised area.

## 16.4 Programming in C

### 16.4.1 Concept of Module

A module is a separate executable entity. All the objects defined in a module (data or functions) are internal and can be made available to external entities (C modules, ladder modules or monitor).

To be valid, a module must have a single «main()» function.

A module is the result of compiling and link editing of one or more source files in C. It is visible and accessible under PLCTOOL as a «\*.XCX» type file.

An application can include one or more C modules. It is strongly recommended to split large applications into several modules.

### 16.4.2 Interface between C Modules

Applications written in C can be very large (several hundred Kbytes).

After editing the application, it is necessary to compile, link edit and load the entire module. The time required for these operations can rapidly become very long.

To optimise the processing times, it is necessary to split large modules into several independent smaller modules that can exchange all types of data (functions, tables, structures, variables).

In this way, a module works on pointers to the objects to be transferred and the pointers are initialised when the links are edited.

Below we use the following terms:

- «imported object» to denote the objects used in a module and defined in another,
- «exported object» to denote objects defined in a module and made available to other modules.

An object can be any type of global data:

- Structure,
- Function,
- Global variable,
- Array,
- etc.

Two functions, IMPORT() and EXPORT(), are available for processing imported and exported objects.

## Functions

**IMPORT()** is used to process an external object in a module.

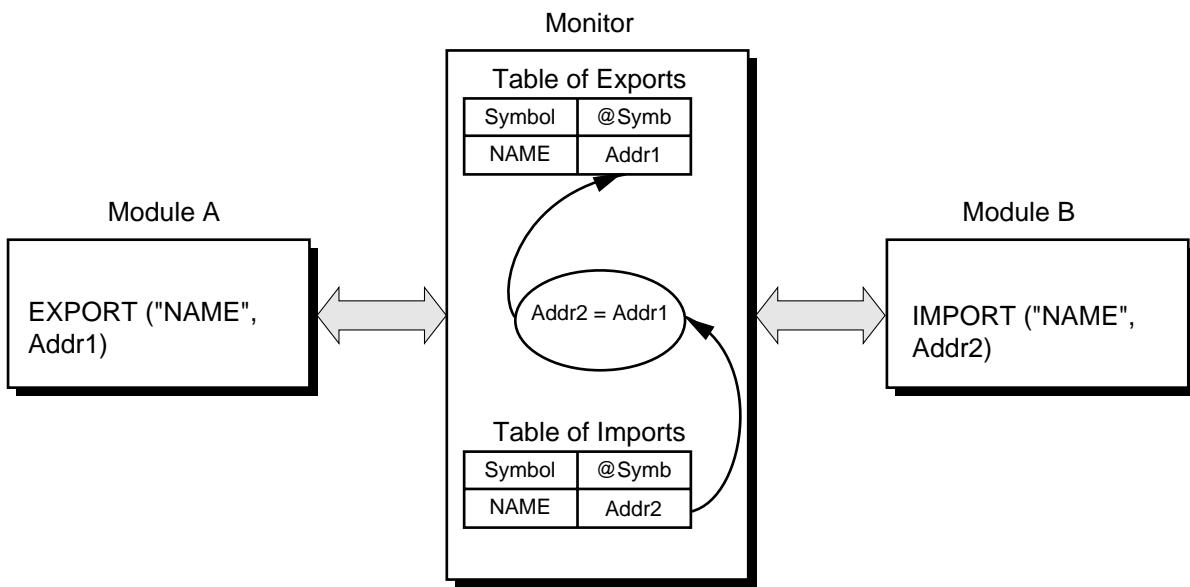
**EXPORT()** is used to make an object available to other modules for use.

**! CAUTION**

An object cannot be imported until it has been exported by its owner module.  
The user defines the imported and exported objects in the main() function of his modules.

When the translator is called, the monitor updates a table of exports and a table of imports. These two tables contain the lists of symbols and their respective addresses. Once all the main() functions of all the modules have been called, the monitor can create the links between exported objects and imported objects.

The symbol name is used by the monitor to link objects. The same name can therefore not be used to export two different objects.



NAME : Symbol (Character string)

This symbol is used during import

Addr1: Address of the object to be exported (defined in module A)

Addr2: Address of the object that will be used in module B

### 16.4.3 Exchange Area

All the variables of the exchange area (see Chapter 3) are accessible for programming in C. The exchange area is defined in header file NUM.H.

It is necessary to include file NUM.H at the beginning of the source files that will be using the exchange area.

File NUM.H is divided into four areas, validated by the following definitions:

- the exchange area is validated by «#define VariablesLAD»,
- the transparent mode function area is validated by «#define NCScreen»,
- the grafset formalism area is validated by «#define Grafset»,
- the file management area is validated by «»#define File\_Management».

#### Example

To use the exchange area

```
#define VariablesLAD
```

```
#include <NUM.H>
```

To use the exchange area and transparent mode functions

```
#define VariablesLAD
```

```
#define NCScreen
```

```
#include <NUM.H>
```

### 16.4.4 Accessing the Saved Common Internal Variables

These byte, word or long word variables are accessible by the following keywords:

Keyword	Value	Definition
_MB(a)	0 < to < 77FF	Signed byte
_MW(a)	0 < to < 77FE	Signed word
_ML(a)	0 < to < 77FD	Signed long word

It is also possible to access data by the address (equivalent to the .& ladder operator).

Keyword	Value	Definition
_pM(a)	0 < to < 77FF	

### 16.4.5 Accessing Unsaved Common Internal Variables

These byte, word or long word variables are accessible by the following keywords:

Keyword	Value	Definition
_VB(a)	0 < to < 77FF	Signed byte
_VW(a)	0 < to < 77FE	Signed word
_VL(a)	0 < to < 77FD	Signed long word

It is also possible to access data by the address (equivalent to the .& ladder operator).

Keyword	Value	Definition
_pV(a)	0 < to < 77FF	

#### 16.4.6 Accessing the Card Inputs

These variables can only be accessed for read by byte, word or long word.

Keyword	Value	Definition
_IB(a,b,c)		Byte
_IW(a,b,c)		Word
_IL(a,b,c)		Long word

For all these keywords:

- a: Rack number ( $0 < a < 6$ ),
- b: Card number in rack ( $0 < b < F$ ),
- c: Logical address in the card ( $0 < c < 3F$ ).

*REMARK To access a particular input, it is necessary to mask the corresponding byte.*

#### 16.4.7 Accessing the Card Outputs

These variables can only be accessed for write by byte, word or long word.

Keyword	Value	Definition
_QB(a,b,c)		Byte
_QW(a,b,c)		Word
_QL(a,b,c)		Long word

For all these keywords:

- a: Rack number ( $0 < a < 6$ ),
- b: Card number in rack ( $0 < b < F$ ),
- c: Logical address in the card ( $0 < c < 3F$ ).

*REMARK To access a particular output, it is necessary to mask the corresponding byte.*

#### 16.4.8 Standard Data Types

The standard data types of C were redefined for greater clarity.

Standard data type	Definition
UINT32	Unsigned variable on 4 bytes
UINT16	Unsigned variable on 2 bytes
UINT8	Unsigned variable on 1 byte
SINT32	Signed variable on 4 bytes
SINT16	Signed variable on 2 bytes
SINT8	Signed variable on 1 byte (character)

## 16.4.9 Library Functions

An application written in C is generated on a separate system. The NUM.OBJ function library gives access to the monitor primitives. This library is used during link editing.

All these functions are prototyped in the NUM.H header file.

### 16.4.9.1 System Functions

#### Export an Object

## EXPORT

#### Syntax

**SINT32 EXPORT(SINT8 \*symbol, void \*symbol\_addr)**

symbol: Character string.

symbol\_addr: Symbol address.

This function makes a C object visible to all the other modules (total visibility) or associates a function with a PLC task.

Report:

- 0 = OK
- -1 = task already defined or too many export symbols.

#### Example 1

This export instruction associates ts01\_in\_C with task TS01.

```
main()
{
    EXPORT(«TS01»,ts01_in_C);
}
void ts01_in_C()
{
    function body
}
```

#### Example 2

This export instruction makes «table» completely visible.

```
SINT16 table[100];
main()
{
    EXPORT(«LABEL»,table);
}
```

## Programming error causing a CPU fault

Access to a prohibited address:

- \*symbol parameter error,
- \*\*symbol\_addr error,
- end of symbol string outside the authorised area.

## Importing an Object

# IMPORT

### Syntax

```
SINT32 IMPORT(SINT8 *symbol, void **symbol_addr)
```

symbol: Character string.

symbol\_addr: Pointer to symbol pointer.

This function allows use of an object defined in another module. To be imported, an object must first be exported by another object.

Report:

- 0 = OK
- -1 = call outside the initialisation task or too many import symbols.

### Example

```
void(*import_function)();  
main()  
{  
    IMPORT(«LABEL», &import_function);  
}  
void test()  
{  
    import_function();  
    function body  
}
```

## Programming error causing a CPU fault

Access to a prohibited address:

- \*symbol parameter error,
- \*\*symbol\_addr error,
- end of symbol string outside the authorised area.

### 16.4.9.2 Using System Functions

**Example: «EXPORT a table»**

```
In file EXP.C
#include <NUM.H>
SINT16 table[100];
main()
{
    EXPORT(«LABEL», table);
}
```

```
In file IMP.C
#include <NUM.H>
SINT16 *ptab;
main()
{
    IMPORT(«LABEL», &ptab);
}
```

**Example: «EXPORT a function»**

In module 1

```
#include <NUM.H>
SINT16 Display(UINT8 what, SINT16 quantity)
{
    SINT16 i;
    for (i = 0; i < quantity; i++);
    {
        EMIV(what);
    }
    return(i);
}
main()
{
    EXPORT(«FUNCT1», Display);
}
```

## In module 2

```
#include <NUM.H>
SINT16 (*ImpFunc)(UINT8, SINT16);
/* ImpFunc: Pointer to a function requiring two parameters that return SINT16*/
void funct2()
{
    PCUR(5, 2);
    ImpFunc(«_», 10);
    /* Execution of the imported function*/
}
main()
{
    IMPORT(«FUNCT1», &ImpFunc);
    /* Initialisation of the pointer to the external function*/
}
```

### **16.4.9.3 Functions for Exchanges by Protocol**

The operation and parameters of these functions are the same as for the corresponding ladder functions (see Chapter 15).

#### **Read the Answer from a Distant Server**

**NETI**

##### Syntax

```
UINT8 NETI(UINT8 port, UINT8 *buffer_addr)
```

#### **Programming error causing a CPU fault**

Access to a prohibited address:

- \*datagram parameter error,
- \*datagram+size outside authorised area.

#### **Send a UNITE Request to a Distant Server**

**NETO**

##### Syntax

```
UINT8 NETO(UINT8 port, UINT8 *buffer_addr)
```

#### **Programming error causing a CPU fault**

Access to a prohibited address:

- \*datagram parameter error,
- \*datagram+size outside authorised area.

**Read an Internal UNITE Answer****UNITI**Syntax**UINT8 UNITI(UINT8 source\_port, UINT8 \*datagram)****Programming error causing a CPU fault**

Access to a prohibited address:

- \*datagram parameter error,
- \*datagram+size outside authorised area.

**Send an Internal UNITE Request****UNITO**Syntax**UINT8 UNITO(UINT8 source\_port, UINT8 \*datagram)****Programming error causing a CPU fault**

Access to a prohibited address:

- \*datagram parameter error,
- \*datagram+size outside authorised area.

**16.4.9.4 Managing the Serial Lines**

The operation and parameters of these functions are the same as for the ladder functions (see Chapter 12).

**Select the Serial Line Driver****COMCTL**Syntax**SINT8 COMCTL(UINT8 portno, UINT8 config)****Select Data Rate and Format****COMF**Syntax**SINT32 COMF(UINT8 portno, UINT16 txrate, UINT16 rxrate, UINT16 format)**

## COMIN

Read the Reception Buffer

Syntax

```
SINT16 COMIN(UINT8 portno, UINT8 *buffer, UINT16 no)
```

Programming error causing a CPU fault

Access to a prohibited address:

- \*buffer parameter error,
- \*buffer+nb outside authorised area

## COMOUT

Send a Buffer

Syntax

```
SINT8 COMOUT(UINT8 portno, UINT8 *buffer, UINT16 no)
```

Programming error causing a CPU fault

Access to a prohibited address:

- \*buffer parameter error,
- \*buffer+nb outside authorised area

## COMREG

Read the Status of a Serial Line

Syntax

```
UINT16 COMREG(UINT8 portno)
```

### 16.4.9.5 Managing the Transparent Mode

The operation and parameters of these functions are the same as for the ladder functions (see Chapter 8).

## EMIV

Send a Character to the Display

Syntax

```
void EMIV(UINT8 char)
```

char:           Character or character code (see Sec. 8.3.4.1)

Initialise the Graphic Display Characters

**INIG**

Syntax

```
void INIG()
```

Select the Main Window

**MAIN\_WINDOW**

Syntax

```
void MAIN_WINDOW()
```

Select the User Window

**STATUS\_WINDOW**

Syntax

```
void STATUS_WINDOW()
```

Select the Key Window

**KEY\_WINDOW**

Syntax

```
void KEY_WINDOW()
```

Position the Cursor

**PCUR**

Syntax

```
void PCUR(UINT8 line, UINT8 col)
```

Draw a Line

**PICO**

Syntax

```
void PICO(UINT8 Line_Type, UINT16 X, UINT16Y)
```

Line\_Type: Type of line to be drawn

Line\_Type Line type

0	Solid
1	Dotted
2	Dashed
3	Combined
4	No line

X, Y: End point (in pixels)

### Draw an Arrow

## FLEC

#### Syntax

```
void FLEC(UINT8 Code, UINT16 Length, UINT16 Width)
```

Code: Arrow pointing

Code Arrow type

1	Pointing right
2	Pointing left
3	Pointing up
4	Pointing down

Length: Arrowhead length (in pixels)

Width: Arrowhead width (in pixels)

### Draw a Rectangle

## RECT

#### Syntax

```
void RECT(UINT16 Width, UINT16 Length)
```

Width: Rectangle width (in pixels)

Length: Rectangle length (in pixels)

**Draw a Circle****CERC**Syntax**void CERC(UINT16 Radius)**

Radius: Circle radius (in pixels)

**Draw a Diamond****LOSA**Syntax**void LOSA(UINT16 Width, UINT16 HalfHeight, UINT16 TotalHeight)****Draw a Clockwise Arc****ARCA**Syntax**void ARCA(UINT8 Line\_Type, UINT16 Endx, UINT16 Endy, UINT16 Centrex, UINT16 Centrey)**

LineType: Type of line to be drawn

Endx, Endy: Coordinates of the end point (in pixels)

Centrex, Centrey: Coordinates of the centre (in pixels)

**Draw a Counterclockwise Circle****ARCT**Syntax**void ARCT(UINT8 Line\_Type, UINT16 Endx, UINT16 Endy, UINT16 Centrex, UINT16 Centrey)**

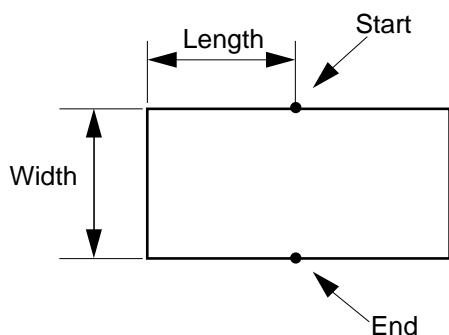
LineType: Type of line to be drawn

Endx, Endy: Coordinates of the end point (in pixels)

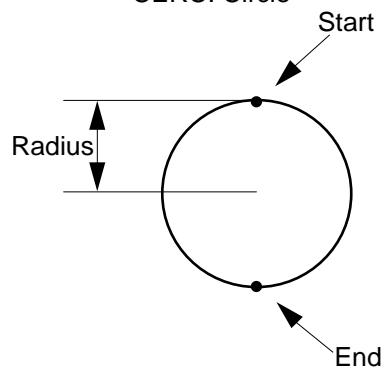
Centrex, Centrey: Coordinates of the centre (in pixels)

**Draw a Test Icon****TEST**Syntax**void TEST(UINT16 HalfBase, UINT16 HalfWidth, UINT16 GHalfHeight, UINT16 Height)**

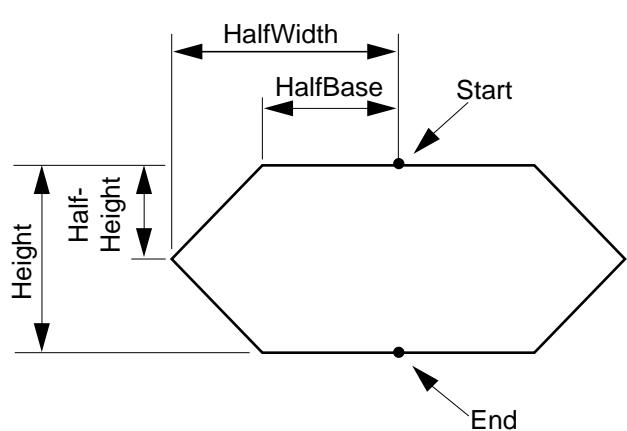
**RECT:** Rectangle



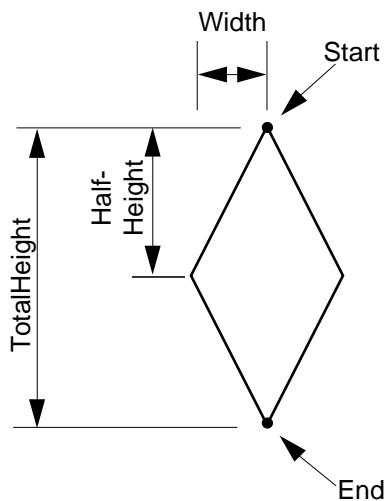
**CERC:** Circle



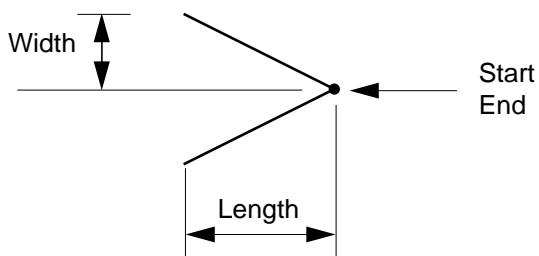
**TEST:** Test



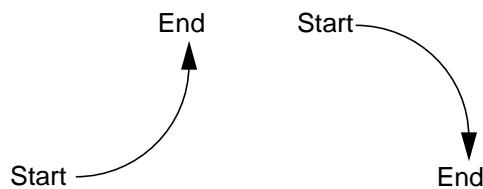
**LOSA:** Diamond/Triangle



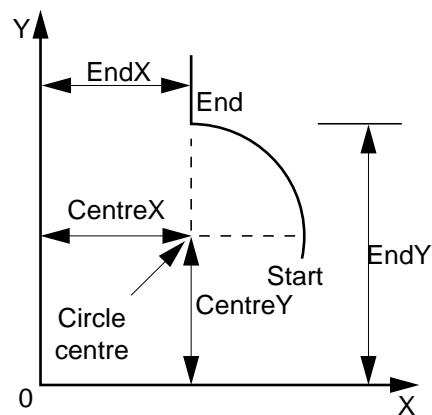
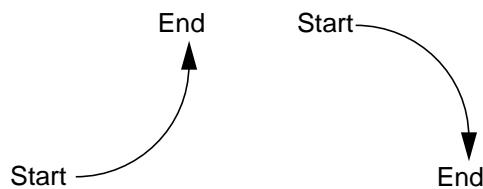
**FLEC:** Arrow



**ARCT**



**ARCA**



**Colour and Stop on Contour****COLOR**Syntax**void COLOR(UINT8 Colour, UINT16 X, UINT16 Y)**

Colour: Contour colour code from 0 to 15 (see Sec. 8.3.3.2).

X, Y: Colouring start point (in pixels)

**Select a Colour****SELCOL**Syntax**void SELCOL(UINT8 Colour)****Simulate Operator Panel Keyboard****PUTKEY**Syntax**SINT32 PUTKEY(UINT8 Code\_touche)****Open a Keyboard Acquisition****SCANO**Syntax**SINT32 SCANO(UINT8 \*Question, UINT16 Width)****Programming error causing a CPU fault**

Access to a prohibited address:

- \*question parameter error,
- end of string outside authorised area.

**Open a Numerical Keypad Acquisition****SCANU**Syntax**SINT32 SCANU(UINT8 \*Question, UINT16 Width)****Programming error causing a CPU fault**

Access to a prohibited address:

- \*question parameter error,
- end of string outside authorised area.

## SCANS

Read a String

Syntax

```
SINT32 SCANS(UINT8 *Dest)
```

**Programming error causing a CPU fault**

Access to a prohibited address:

- \*dest parameter error,
- end of acquisition field outside authorised area.

Read and Convert a Decimal Number

## SCAND

Syntax

```
SINT32 SCAND(UINT32 *variableL)
```

**Programming error causing a CPU fault**

Access to a prohibited address:

- \*lvariable parameter error.

Read and Convert a Hexadecimal Number

## SCANX

Syntax

```
SINT32 SCANX(UINT32 *variableL)
```

**Programming error causing a CPU fault**

Access to a prohibited address:

- \*lvariable parameter error.

Close a Keyboard Acquisition

## SCANC

Syntax

```
SINT32 SCANC()
```

### 16.4.9.6 Programming the Analogue Inputs/Outputs

The operation and parameters of these functions are the same as for the ladder functions (see Chapter 9).

Reassign an Analogue Card

## ANAA

Syntax

```
SINT32 ANAA(UINT8 Initial_cch, UINT8 Final_cch)
```

**Read an Analogue Input****ANAI**Syntax

```
SINT32 ANAI(UINT8 cch, SINT16 *winput)
```

**Programming error causing a CPU fault**

Access to a prohibited address:

- \*winput parameter error.

**Write an Analogue Output****ANAO**Syntax

```
SINT32 ANAO(UINT8 cch, SINT16 woutput)
```

**Configure an Analogue I/O Card****ANAS**Syntax

```
SINT32 ANAS(UINT8 ccn, SINT16 wconfig)
```

**16.4.9.7      Explicit Read/Write**

The operation and parameters of these functions are the same as for the ladder functions (see Chapter 10).

**Explicit Read of an Input Card****READ\_I**Syntax

```
SINT8 READ_I(UINT16 rcmch, UINT8 n)
```

**Explicit Write of an Output Card****WRITE\_Q**Syntax

```
SINT8 WRITE_Q(UINT16 rcmch, UINT8 n)
```

#### 16.4.9.8 Programming the Interrupt Inputs

The operation and parameters of these functions are the same as for the ladder functions (see Chapter 11).

##### Configure an Interrupt Input

**ITICTL**

###### Syntax

```
SINT32 ITICTL(UINT32 itino, UINT8 iti_config)
```

##### Read an Interrupt Input

**ITIGET**

###### Syntax

```
UINT8 ITIGET(UINT8 itino)
```

##### Associate an Interrupt Input with an Axis Group

**ITI\_GR**

###### Syntax

```
SINT32 ITI_GR(UINT32 itino, UINT32 group)
```

##### Associate a Hardware Task with an Interrupt Input

**THITI**

###### Syntax

```
SINT32 THITI(UINT32 thno, UINT32 itino)
```

#### 16.4.9.9 Managing Background Tasks

The operation and parameters of these functions are the same as for the ladder functions (see Chapter 7).

##### Start a Critical Section

**CSBEGIN**

###### Syntax

```
void CSBEGIN(void)
```

**End a Critical Section****CSEND**Syntax

```
void CSEND(void)
```

**Start a %TF Task****TFSTART**Syntax

```
SINT32 TFSTART(UINT16 tf_number)
```

**Stop a %TF Task****TFSTOP**Syntax

```
STIN32 TFSTOP(UINT16 tf_number)
```

**Suspend a %TF Task for n PLC Cycles****WHTR**Syntax

```
void WHTR(UINT16 n)
```

**16.4.9.10 General Purpose Functions**

The operation and parameters of these functions are the same as for the corresponding Ladder functions (see Chapter 6).

**Update Analogue Output 0****CNA0**Syntax

```
void CNA0(SINT16 value)
```

**Update Analogue Output 1****CNA1**Syntax

```
void CNA1(SINT16 value)
```

Optimise Circular Search

## QCKTOOL

Syntax

```
SINT32 QCKTOOL(SINT32 origin, SINT32 destination, SINT32 n)
```

Wear Offset

## TOOLDYN

Syntax

```
SINT32 TOOLDYN(SINT16 offset, UINT8 axis, UINT8 n_toolno)
```

On Type Timeout

## TEMPO\_ENCLENCHEMENT

Syntax

```
SINT32 TEMPO_ENCLENCHEMENT(SINT8 Instance, UINT8 Input, SINT32 Threshold)
```

Off Type Timeout

## TEMPO\_DECLENCHEMENT

Syntax

```
SINT32 TEMPO_DECLENCHEMENT(SINT8 Instance, UINT8 Input, UINT32 Threshold)
```

Pulse Type Timeout

## TEMPO\_IMPULSION

Syntax

```
SINT32 TEMPO_IMPULSION(SINT8 Instance, UINT8 Input, UINT32 Threshold)
```

Call a Subroutine

## SP

Syntax

```
void SP(n_moduleno)
```

**Semaphore****SEMA**Syntax**SINT8 SEMA(SINT8\*semaphore)****Programming error causing a CPU fault**

Access to a prohibited address:

- \*flag parameter error.

**Read n E42000 Variables****R\_E42000**Syntax**SINT32 R\_E42000(SINT8 \*dest, UINT32 number, UINT32 n)****Programming error causing a CPU fault**

Access to a prohibited address:

- \*dest parameter error,
- \*dest+n outside authorised area

**Write n E42000 Variables****W\_E42000**Syntax**SINT32 W\_E42000(SINT8 \*source, UINT32 number, UINT32 n)****Programming error causing a CPU fault**

Access to a prohibited address:

- \*source parameter error
- \*source+n outside authorised area

**Read Current Date****TMGET**Syntax**TMGET(\*date)****Programming error causing a CPU fault**

Access to a prohibited address:

- \*dest parameter error.

## Read Current Date and Day in Week

## DTGET

### Syntax

```
DTGET(*date)
```

### Programming error causing a CPU fault

Access to a prohibited address:

- \*date parameter error.

### 16.4.9.11 File Management

The global memory is managed like a disk drive. The data are stored in it as files in directories.

There are three directories for the different file types:

- ladder application files with .XLA extension,
- C application files with .XCX extension,
- user files.

Programming in C allows you to create your own files. A number of primitives are available for managing these files.

The error codes returned by the primitives (file and directory) are:

Error code	Description
UF_SYSFAIL	System error
UF_DSKFULL	Disk full
UF_ERRNAME	Error in file name
UF_NEXIST	No such file
UF_OPEN	File open
UF_NOPEN	File not open

### Create a File in the User Directory

## USER\_CREATE\_F

### Syntax

```
SINT32 USER_CREATE_F(SINT8 *filename)
```

### Parameters

Inputs:      filename: The path must not be specified because the user files are necessarily stored in a particular directory.

Output:      None

**REMARKS** If the filename already exists, the function returns error code «UF\_ERRNAME». An existing file must first be deleted by function «USER\_REMOVE\_F» before the name can be reused for another file.

After execution of function «USER\_CREATE\_F», the new file is empty. Function «USER\_CREATE\_F» is only used for data files. Function «USER\_CREATE\_F» does not automatically open the file. To open it, use function «USER\_OPEN\_F».

#### Programming error causing a CPU fault

Access to a prohibited address:

- \*filename parameter error.

#### Delete a File

## USER\_DELETE\_F

#### Syntax

`SINT32 USER_DELETE_F(SINT8 *filename)`

#### Parameters

Inputs: filename: The path must not be specified because the user files are necessarily stored in a particular directory.

Output: None

**REMARK** If a file is always open, it is not deleted.

#### Programming error causing a CPU fault

Access to a prohibited address:

- \*filename parameter error.

#### Open a File

## USER\_OPEN\_F

#### Syntax

`SINT32 USER_OPEN_F(UINT32 *pF_id, SINT8 *filename)`

#### Description

Opens the file identified by filename. This file is accessible for read and write. If the operation is executed normally, the file manager returns an identifier pF\_id used by functions «USER\_CLOSE\_F», «USER\_READ\_F», «USER\_WRITE\_F» and «USER\_SEEK\_F».

#### Parameters

Inputs: filename: The path must not be specified because the user files are necessarily stored in a particular directory.

Output: pF\_id: File identifier if the operation was correctly completed.

**REMARKS** Function «USER\_OPEN\_F» does not check the file type.  
Function «USER\_OPEN\_F» sets the file pointer to the first byte in the file.

#### Programming error causing a CPU fault

Access to a prohibited address:

- \*pF\_id parameter error,
- \*filename parameter error.

## USER\_CLOSE\_F

#### Syntax

```
SINT32 USER_CLOSE_F(UINT32 F_Id)
```

#### Description

Closes a file opened by function «USER\_OPEN\_F».

#### Parameters

Inputs: pF\_id: File identifier

Outputs: None

**REMARK** Since only a limited number of files can be opened simultaneously, function «USER\_CLOSE\_F» must be used as soon as it is no longer necessary to have the file open.

## USER\_READ\_F

#### Read the Data in a File

#### Syntax

```
SINT32 USER_READ_F(UINT32 pF_Id, UINT8 *pBuf, UINT32 Requestno, UINT32 No_read)
```

#### Parameters

Inputs: pF\_id: File identifier returned by USER\_OPEN\_F.

pBuf: Received data buffer.

Requestno: Number of bytes to be read.

Outputs: No\_read: Number of bytes effectively read. If this number is less than the number requested, the end of the file was reached.

**REMARKS** The data are read from the current position of the file pointer. The file pointer is automatically repositioned after the read.

#### Programming error causing a CPU fault

Access to a prohibited address:

- \*pBuf parameter error,
- \*pBuf+Requestno outside authorised area,
- \*No\_read parameter error.

**Write a File****USER\_WRITE\_F**Syntax

```
SINT32 USER_WRITE_F(UINT32 pF_Id, UINT8 *pBuf, UINT32 Buf_Size)
```

Description

Writes data in the file specified by pF\_id. The write always begins on the file pointer. After the write, the pointer is positioned on the last byte in the file.

Parameters

Inputs:      pF\_id: File identifier returned by function «USER\_OPEN\_F».  
               pBuf: Buffer containing the data to be written.  
               Buf\_Size: Buffer size.

Outputs:     None

**REMARKS** *The data are written starting from the current position of the file pointer. The file pointer is automatically repositioned after the write.*

**Programming error causing a CPU fault**

Access to a prohibited address:

- \*pBuf parameter error,
- \*pBuf+Requestno outside authorised area.

**Reposition the Pointer****USER\_SEEK\_F**Syntax

```
SINT32 USER_SEEK_F(UINT32 pF_Id, UINT32 mode, SINT32 offset, UINT32 *Old_Ptr)
```

Description

Repositions the pointer for read or write in the file specified by pF\_id.

Parameters

Inputs:      pF\_id: File identifier returned by function «USER\_OPEN\_F».  
               mode: Seek start position. 0 seek from beginning of file.  
               1 seek from current position.  
               2 seek from end of file.  
               offset: Signed offset relative to the mode selected.

Outputs:     Old\_Ptr: Initial pointer value.

**REMARKS** *The pointer is different for each file.*

*The pointer is an unsigned variable. Movement outside the file limits generates an error.*

## Programming error causing a CPU fault

Access to a prohibited address:  
- \*pOld\_Ptr parameter error.

### 16.4.9.12 Directory Management

Only the user directory is accessible. Three primitives are available to read the contents of this directory.

#### Open User Directory

## USER\_OPEN\_DIR

#### Syntax

```
SINT32 USER_OPEN_DIR()
```

#### Parameters

Inputs: None

Outputs: None

*REMARK A directory is open for read only.*

#### Close User Directory

## USER\_CLOSE\_DIR

#### Syntax

```
SINT32 USER_CLOSE_DIR()
```

#### Description

Closes the user directory opened by the primitive «USER\_OPEN\_DIR».

#### Parameters:

Inputs: None

Outputs: None

#### Read User Directory

## USER\_READ\_DIR

#### Syntax

```
SINT32 USER_READ_DIR(UINT8 *pBuf, UINT32 No_Requested, UINT32*No_Read)
```

#### Parameters

Inputs: pBuf: Data receive buffer.

No\_Requested: Number of bytes to be read.

Outputs: No\_Read: Number of bytes effectively read. If this number is less than the number requested, the end of the directory was reached.

Description of a Directory File

A directory file is described by 32 bytes organised as follows:

Number of bytes	Description
8 bytes	File name If filename[0] has one of the following values, then: 0x00 End of directory 0x2E System file 0xE5 File deleted 0x05 The name begins with 0x05
3 bytes	File extension
1 byte	File attribute Bit 0 = 1: Read only Bit 1 = 1: Hidden file Bit 2 = 1: System file Bit 3 = 1: Volume name (ROOT) Bit 4 = 1: Directory file Bit 5 = 1: Archive bit Bit 6 = 1: Spare Bit 7 = 1: Spare
10 bytes	Spare
2 bytes	Time in INTEL format
2 bytes	Date in INTEL format
2 bytes	Start cluster in INTEL format
4 bytes	File size in INTEL format

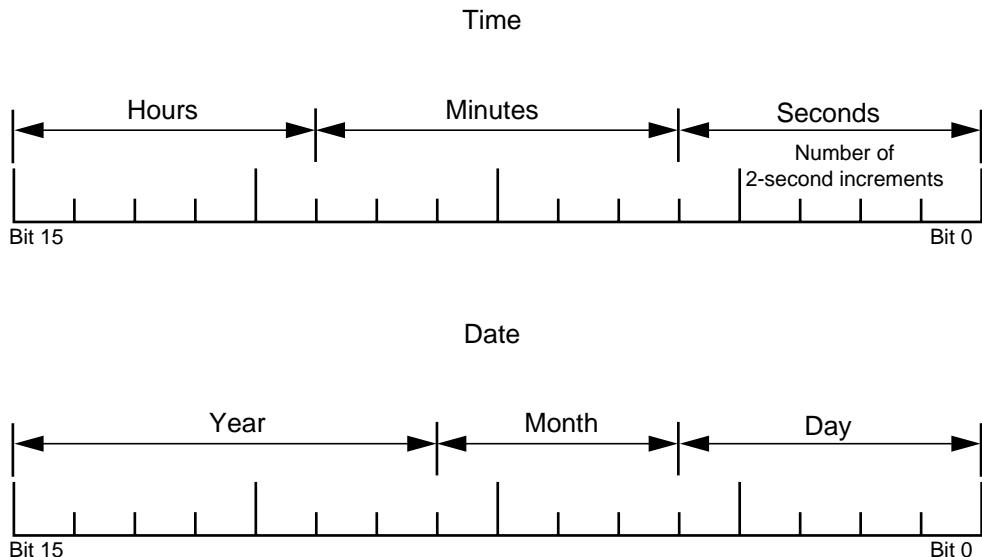
**Programming error causing a CPU fault**

Access to a prohibited address:

- \*pBuf parameter error,
- \*pBuf+Requestno outside authorised area,
- \*No\_read parameter error.

The time, date and size are in INTEL format, i.e. the MSBs and LSBs are reversed with respect to the MOTOROLA format.

In the MOTOROLA format, these bits are as follows:



To facilitate read of these data, the following structures (defined in the NUM.H header file) can be used:

```

struct S_TIME_FILE
{
    UINT8 Sec      :5; /* Second */
    UINT8 Min      :6; /* Minutes */
    UINT8 Hour     :5; /* Hours */
};

struct S_DATE_FILE
{
    UINT8 Day      :5; /* Day */
    UINT8 Month    :4; /* Month */
    UINT8 Year     :7; /* Year */
};

struct S_DIRECTORY_ELEMENT
{
    UINT8 filename[8];
    UINT8 Extension[3];
    UINT8 Attribute;
    UINT8 Spare[10];
    UINT16 Intel_Time; /*Word in INTEL format*/
    UINT16 Intel_Date /*Word in INTEL format*/
    UINT16 Start_Cluster;
    UINT32 Intel_File_Size; /*Long word in INTEL format*/
};

```

## 17.1 General

The PLC axis programming function allows the automatic control function to control these axes.

These PLC axes are controlled by the CNC function.

Depending on the 1060 system, the CNC function can control a maximum of:

	No. of axes	No. of groups
NUM 1060 Series I	32	8 (9 axes per group)
NUM 1060 Series II	8	3 (9 axes per group)

The PLC axes are contained in PLC axis groups within which they can be interpolated.

The number of axis groups (CNC and PLC) in the system is defined by machine parameter P97 (see Parameter Manual).

The part programming functions performed by the PLC axis groups are the same as those performed by the CNC axis groups except for functions M01, M12 and G75 (see Part Programming Manual).

The main operating modes of the PLC axis groups are the AUTO and SINGLE modes (see Operator Manual). A PLC axis group operates in one of these modes independently of the modes of the other groups (CNC or PLC).

Operation in JOG mode:

- When a PLC group is enabled by %Wg00.2 followed by a reset (pulse %Wg01.0), it can be controlled in JOG mode. Feed rate override is then control by potentiometer setting variable %Wg02.b.

## 17.2 Programming Principle

Commands and reports are exchanged between the automatic control function and the CNC function via the exchange area. The exchanges must be processed in the user programme.

### 17.2.1 Data Transfers from the Automatic Control Function to the CNC Function

The data concerning the PLC axis groups sent by the automatic control function to the CNC function include:

Function	Variable	Mnemonic
Emergency retraction request	%Wg01.4	C_DGURG1 to C_DGURG8
Cycle start request	%Wg01.2	C_CYCLE1 to C_CYCLE8
Axis group reset request	%Wg01.0	C_RAZ1 to C_RAZ8
Mode selection (AUTO or SINGLE)	%Wg00.7	C_MODE1 to C_MODE8
High speed command maintained on group	%Wg00.6	C_FAST1 to C_FAST8
M function reports for group	%Wg00.5	CRM1 to CRM8
Subroutine call by the machine processor	%Wg00.4	APPSS1 to APPSS8
Stop on switch signal	%Wg00.3	AR BUT1 to AR BUT8
Axis group enable	%Wg00.2	VALID1 to VALID8
End of external movement signal	%Wg00.1	C_FMEXT1 to C_FMEXT8
Axis group feed authorisation	%Wg00.0	C_AUTAV1 to C_AUTAV8

## 17.2.2 Data Transfers from the CNC Function to the Automatic Control Function

The data concerning the PLC axis groups sent by the CNC function to the automatic control function include:

Function	Variable	Mnemonic
Axis group fault	%Rg01.6	E_DEF1 to E_DEF8
Axis on wait for position	%Rg01.5	NO_POS1 to NO_POS8
Emergency retraction in progress	%Rg01.4	E_DGURG1 to E_DGURG8
Cycle in progress	%Rg01.2	E_CYCL1 to E_CYCL8
Reset in progress	%Rg01.0	E_RAZ1 to E_RAZ8

---

# 18 Programme Debugging

<b>18.1 Programme Debugging with the PLCTOOL Software Workshop</b>	18-3
<b>18.2 Debugging on the CNC</b>	18-3
18.2.1 Utility Access Procedure	18-3
18.2.2 Monitoring CPU Operation	18-5
18.2.2.1 PLC Status	18-6
18.2.2.2 Background Task Activity	18-8
18.2.2.3 Hardware Task Activity	18-8
18.2.2.4 CPU Command	18-8
18.2.2.5 Resetting Saved Variables	18-10
18.2.3 Monitor and %TS Task Times	18-11
18.2.4 File Management	18-13
18.2.4.1 Format the Volume	18-14
18.2.4.2 Application Directory	18-15
18.2.4.3 File Deletion	18-16
18.2.4.4 Enabling/Inhibiting the PLCTOOL Link	18-16
18.2.5 Input/Output Configuration	18-17
18.2.6 Software Backup	18-20
18.2.6.1 Software Backup	18-21
18.2.6.2 Check of Downloading	18-21
18.2.6.3 Loading the programme	18-21
18.2.7 Software Load/Download to/from PLCTOOL	18-22
18.2.8 Ladder Animation	18-22



## 18.1 Programme Debugging with the PLCTOOL Software Workshop

Refer to the document PLCTOOL - Ladder Language Programming Tool.

## 18.2 Debugging on the CNC

Resident utility 7 is used to manage the PLC application and the link with the PLCTOOL software workshop for file load/unload.

### 18.2.1 Utility Access Procedure

#### Requirements

Utility 7 does not have any particular access requirements.

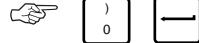
#### Actions

Select the utilities menu.



The “CNC UTILITIES” menu is displayed.

Select the “UTILITIES PRESENT” menu.



A menu listing the utilities present in the CNC memory is displayed.

When the “UTILITIES PRESENT” menu is displayed, it is possible to choose the language in which the utilities will be displayed.

Type “A” for English.



Or

Type “F” for French.



The menu is displayed in the language selected.

Select utility 7.



Display of the main menu, «PLC APPLICATION MANAGEMENT».

PLC APPLICATION MANAGEMENT								
PLC operation								
Monitor and %TS time profile								
File management								
I/O configuration								
Software backup								
Animation								
-- Enter a command (Exit X OFF) --								
.../...								EXIT

#### Exit from the procedure

Enter the command.



Return to the «AXES» page.

## 18.2.2 Monitoring CPU Operation

This function is used to display data concerning operation of the CPU.

### Requirements

«PLC APPLICATION MANAGEMENT» menu displayed.

### Actions

Type «P» for PLC operation.



Display of the «PLC OPERATION» Menu

18

PLC OPERATION								
PLC status: RUNNING (No error)								
TF Activity:								
(0-7) 0 0 0 0 0 0 0 0								
(8-15) 0 0 0 0 0 0 0 0								
TH Activity:								
(0-7) 0 0 0 0 0 0 0 0								
(8-15) 0 0 0 0 0 0 0 0								
Reset saved variables								
.../...								EXIT

### Exit from the Procedure

Press «F11».



Return to the «PLC APPLICATION MANAGEMENT» menu

### 18.2.2.1 PLC Status

The «PLC STATUS» field gives information on the CPU status.

The comments that can be displayed in this area are detailed in the table below:

#### PLC «RUNNING» Status

Messages	Comment
No error	PLC function operating correctly
Bad I/O bus implementation	<b>Cause:</b> Unidentified card present Card missing Discrepancy between programmed configuration and real configuration Watchdog programming error <b>Corrective action</b> Check the input/output configuration (see Sec. 18.2.5) Check the state of %R97F.B (see Sec. 3.8.5) Check %INI in the user programme (card and watchdog configuration variables)
Erroneous I/O bus behaviour	<b>Cause:</b> Link error on the bus <b>Corrective action</b> Check the state of bus status signal %Rrc39.B (see Sec. 3.7.3) Check the continuity of the fibre-optic ring Contact NUM CUSTOMER SERVICE

#### PLC «HALTED» Status

Messages	Comment
Internal monitor fault	<b>Cause:</b> Serious internal error <b>Corrective action</b> Contact NUM CUSTOMER SERVICE
Application time limit overrun	<b>Cause:</b> RTC overrun (endless loop in a programme) <b>Corrective action</b> Correct the user programme
Unknown PLC error	<b>Cause:</b> Serious internal error <b>Corrective action</b> Contact NUM CUSTOMER SERVICE

Messages	Commentaire
Bad I/O bus configuration	<p><b>Cause:</b> Unidentified card resent Card missing Discrepancy between programmed configuration and real configuration Watchdog programming error</p> <p><b>Corrective action:</b> Check the input/output configuration (see Sec. 18.2.5) Check the state of %R97F.B (see Sec. 3.8.5) Check %INI in the user programme (card and watchdog configuration variables)</p>
Erroneous I/O bus behaviour	<p><b>Cause:</b> Link error on the bus</p> <p><b>Corrective action:</b> Check the state of bus status signal %Rrc39.B (see Sec. 3.7.3) Check the continuity of the fibre-optic ring Contact NUM CUSTOMER SERVICE</p>
Too many boards in I/O bus	<p><b>Cause:</b> Too many inputs and outputs for the system</p> <p><b>Corrective action:</b> Decrease the number of inputs and outputs cards within the authorised limits</p>

When installing the customer code

Messages	Comment
Can't read the application	<p><b>Cause:</b> Serious internal error</p> <p><b>Corrective action:</b> Contact NUM CUSTOMER SERVICE</p>
Errors in module TSi / TFi / THi / SPi / C code / ???	<p><b>Cause:</b> Problems when loading (module too large, not enough space in the local memory for loading the module)</p> <p><b>Corrective action:</b> Decrease the module size Increase the local memory size If «??», contact NUM CUSTOMER SERVICE</p>
C module: duplicate export of a symbol	<p><b>Cause:</b> Double export of symbols in a C module</p> <p><b>Corrective action:</b> Check and edit the C module</p>
Duplicate definition of module TSi / TFi / THi / SPi / C code / ???	<p><b>Cause:</b> A module with the same name is present twice in the application</p> <p><b>Corrective action:</b> The user application modules must all have different names. Correct the user application If «??», contact NUM CUSTOMER SERVICE</p>

#### During execution of the user programme

A message is displayed on three lines.

- Line 1: Message text  
Line 2: Indicates the task involved: TSi/THi/TFi/INI  
Line 3: Indicates the module involved: «Module: \*.\*[@ with respect to the module start address]» or «Module: ???[@ with respect to the start of the PLC monitor mapping]».

**REMARK** *Relative addresses with respect to the module start address can only be used for C modules (\*.XCX). The \*.MAP file of the C application gives these addresses.*

Messages	Comment
User code: divide overflow	<b>Cause</b> Overflow on a division <b>Corrective action</b> Check and edit the module involved If «???», contact NUM CUSTOMER SERVICE
User code : @ with respect to the monitor = Illegal address	<b>Cause</b> Operation on a prohibited address <b>Corrective action</b> Check and edit the module involved If «???», contact NUM CUSTOMER SERVICE
Messages	Comment
Inconsistent user code	<b>Cause:</b> Use of incoherent functions or symbols in the user programme User programme incoherent <b>Corrective action</b> Check and edit the module involved If «???», contact NUM CUSTOMER SERVICE

#### **18.2.2.2 Background Task Activity**

The activity of the background tasks is displayed by 16 counters associated with tasks %TF0 to %TF15.

Whenever a background task has been totally or partially processed during an RTC cycle, the task counter is incremented by one. This function is used to display tasks that are dormant, executing, the number of RTC cycles required for execution of a task, etc.

#### **18.2.2.3 Hardware Task Activity**

The activity of the hardware tasks is displayed by 16 counters associated with tasks %TH0 to %TH15. Whenever a hardware task has been processed, the counter is incremented by one.

#### **18.2.2.4 CPU Command**

The commands «GO», «STOP», and «INIT» are used to control the machine processor during user programme debugging.

## Requirements

«PLC OPERATION» menu displayed.

## Actions

Enter the selected command. (see table below)		 
Operation	Command	Comment
Start the machine processor	Type «G» for Go	Watchdog set. User programme runs.
Stop the machine processor	Type «S» for Stop	Watchdog reset. User programme stops. The state «HALTED» is displayed on the screen page.
System initialisation	Type «I» for Init	Requires stopping the machine processor. - Clears all errors, - Initialises the inputs/outputs.

When disassembling user code (check of function calls)

Messages	Comment
Unknown module name	<b>Cause:</b> A pointer that was not correctly initialised in a C module has destroyed a code area. <b>Corrective action</b> Identify the C module and make the necessary corrections.
Errors in moduleINI, TSi, TFi, THi, C code	<b>Cause:</b> A pointer that was not correctly initialised in a C module has destroyed a code area. <b>Corrective action</b> Identify the C module and make the necessary corrections.

When activating a function call check (PLCTOOL)

Messages	Comment
Illegal access address	<b>Cause:</b> The address parameter of a Ladder or C function point to an area other than a data area. <b>Corrective action</b> Edit the module involved and correct the function.
No more 512 authoritatives zones address	<b>Cause:</b> The application loaded contains more than 512 non-contiguous data areas. <b>Corrective action</b> Group the strings and constants so that they follow one another contiguously.

### 18.2.2.5 Resetting Saved Variables

This function is used to reset saved variables (%M).

#### Requirements

«PLC OPERATION» menu displayed.

#### Actions

Enter the command «S»  

PLC OPERATION							
PLC status: RUNNING (No error)							
TF Activity: (0-7) 0 0 0 0 0 0 0 0 (8-15) 0 0 0 0 0 0 0 0							
TH Activity: (0-7) 0 0 0 0 0 0 0 0 (8-15) 0 0 0 0 0 0 0 0							
Commands: GO / STOP / INIT							
Reset saved variables							
.../...							EXIT

Enter the command «R».  

The message «Are you sure? (Y/N)» is displayed.

Confirm reset by entering «Y».  

or

Cancel reset by entering «N».  

Restart the PLC by the command «Go».  

#### Exit from the Procedure

Press «F11».  

Return to the «PLC APPLICATION MANAGEMENT» menu.

### 18.2.3 Monitor and %TS Task Times

This function is used to display the percentage of time occupied by the monitor and %TS tasks each PLC cycle.

It shows:

- the average time occupied by the monitor each cycle,
- the maximum time occupied by the monitor,
- the average time occupied by each %TS task,
- the maximum time occupied by each %TS task,
- the application time overruns each cycle.

#### Requirements

«PLC APPLICATION MANAGEMENT» menu displayed.

#### Actions

Type «T» for «Monitor and %TS time profile».



The «MONITOR AND %TS TIME PROFILE» menu is displayed.

MONITOR AND %TS TIME PROFILE		
Monitor Average: 0% Max: 0%	Application time limit overrun: 0	
TS0: Average: 0% Max: 0%		
TS1: Average: 0% Max: 0%		
Monitor Average: 0% Max: 0%	Application time limit overrun: 0	
TS0: Average: 0% Max: 0%		
TS2: Average: 0% Max: 0%		
Monitor Average: 0% Max: 0%	Application time limit overrun: 0	
TS0: Average: 0% Max: 0%		
TS3: Average: 0% Max: 0%		
Monitor Average: 0% Max: 0%	Application time limit overrun: 0	
TS0: Average: 0% Max: 0%		
TS4: Average: 0% Max: 0%		
Monitor Average: 0% Max: 0%	Application time limit overrun: 0	
TS0: Average: 0% Max: 0%		
TS5: Average: 0% Max: 0%		
Commands: Valid / Reset Max		
.../...		EXIT

The percentage is calculated with reference to the time allocated to the monitor and user programme, i.e.:

- 18 ms for Series I and Series II biprocessor, even though the RTC cycle is 20 ms, since 2 ms are reserved by the system (see Sec. 2.1)
- Value in ms of P99 in Series II UCSII.

## Measurement Acquisition

Type «V» to validate (enable) measurement acquisition.

The monitor and %TS task times are updated.

## Stop Measurements

Type «I» to inhibit measurement acquisition.

The monitor and %TS task times are reset.

## Reset Maxima

Type «R» to reset the maxima.

The maximum monitor and %TS task times are reset.

## Exit from the Procedure

Press «F11».



EXIT

Return to the «PLC APPLICATION MANAGEMENT» menu.

## 18.2.4 File Management

This function is used to manage the data concerning the files loaded in the machine processor.

### Requirements

«PLC APPLICATION MANAGEMENT» menu displayed.

### Actions

Type «F» for «File Management».



Display of the «PLC FILES MANAGEMENT» menu.

18

PLC FILES MANAGEMENT								
Inhibition of the PLCTOOL link Application directory File delete Format the volume  PLC: 12040 used/17884 free -- Enter a command (Exit F11) --								
.../...								EXIT

The numbers after «PLC» indicate the occupied and free memory space (in bytes).

### Exit from the Procedure

Press «F11».



Return to the «PLC APPLICATION MANAGEMENT» menu.

#### 18.2.4.1 Format the Volume

This command is used to initialise the PLC memory and delete all the files in the memory.

##### Actions

Enter «F» for Format the volume.



The message «System will restart (applic. is lost) please confirm: (Y)» is displayed.



Confirm deletion of all the files in the memory.

USE IMMEDIATELY THE CHANGE								
WARNING! LOADING REQUIRES TO STOP MACHINE CONTROL OK ? (Y/N)								
.../...								EXIT

Restart the system by acknowledging the messages displayed.

##### Exit from the Procedure

Press «F11».



Return to the «PLC APPLICATION MANAGEMENT» menu.

### 18.2.4.2 Application Directory

This page displays all the files loaded in the CPU sectors.

#### Actions

Enter «A» for Application Directory.



Display of the «LADDER FILES DIRECTORY» menu.

PLC sector name	PLCTOOL file type	File size	Load date	Load time
LADDER FILES DIRECTORY				
TS0	.XLA	1538 Octets	24/9/1992	16/51/50
Files (1538 bytes)				
...				EXIT
Number of files present in the memory		Total memory space occupied		

#### Exit from the Procedure

Strike any key.

Return to the «PLC APPLICATION MANAGEMENT» menu.

or

Press «F11».



Return to the «PLC APPLICATION MANAGEMENT» menu.

#### 18.2.4.3 File Deletion

This function is used to delete files from the PLC memory.

##### Actions

Press «D» for File Deletion.



The message «File name?» is displayed.

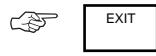
Enter the name of the file to be deleted [Sector name]. [File type].  
(Example: TS0.XLA).



The message «File deleted» is displayed.

##### Exit from the Procedure

Press «F11».



Return to the «PLC APPLICATION MANAGEMENT» menu.

#### 18.2.4.4 Enabling/Inhibiting the PLCTOOL Link

This function is used to enable or inhibit the serial link with the PLCTOOL software for file up/download and for the ON LINE DEBUG function.

##### Enabling the Link

The menu line «Validation of the PLCTOOL link» is displayed.

Press «V» to enable the line.



The menu line becomes «Inhibition of the PLCTOOL link».

##### Inhibiting the Link

The menu line «Inhibition of the PLCTOOL link» is displayed.

Press «I» to inhibit the link.



The menu line becomes «Validation of the PLCTOOL link».

### 18.2.5 Input/Output Configuration

The Input/Output configuration menu displays:

- the type of racks present in the system,
- the type of cards present in each rack.

#### Requirements

«PLC APPLICATION MANAGEMENT» menu displayed.

#### Actions

Type «C» for «I/O Configuration».



Display of the «I/O CONFIGURATION» menu.

18

I/O CONFIGURATION	
Racks	Slots
0 (P8)	C B x 9 x 7 6 5 x x x 1 0
3 (E12)	C B A 9 x x x x 3 2 1 0
4 (E12)	x x x x x x x x 3 2 1 0
-- Commands <R#> or <C##> or <F11> --	
.../...	EXIT

**REMARK** The character «x» indicates the absence of a card.

#### Rack Identification

Type «R» [rack No.]» (number from 0 to 7).



The following messages are displayed on the dialogue line:

Rack No.	dentification	Message
0	P8 (Main rack, 8 slots) P4 (Main rack, 4 slots)	«R0: Main rack, 8 slots» «R0: Main rack, 4 slots» If the rack does not exist: «No rack here!»
1 to 6	E12 (Extension rack, 12 slots) M2 (Module, 2 slots)	«Rx: Extension rack, 12 slots» «Mx: Module rack, 2 slots» If the rack does not exist: «No rack here!» (where «x» equals rack No.)

### Card Identification

Enter the command «C [rack No.][card No.]».



The following messages are displayed on the dialogue line:

Rack type	Card No.	Messages
P8, P4 and E12	0	130 W power supply with optic fiber 130 W power supply without optic fiber 60 W power supply with optic fiber 60 W power supply without optic fiber
P8 and P4	1 to 4	Standard control panel Standard control panel with extension
P8	5 to 0xC	32 relayed output cards
P4	5 to 8	32 DC input card
E12	1 to 0xC	32 input/24 output cards
M2	1 and 2	64 input/48 output cards 32 input/24 output card
P8	1 to 0xC	Card absent
P4	1 to 8	
E12	1 to 0xC	
M2	1 and 2	

### **Exit from the Procedure**

Press «F11».



Return to the «PLC APPLICATION MANAGEMENT» menu.

**Example**

Identification of rack 0 and cards 5, 8 and B equipping it  
 «PLC APPLICATION MANAGEMENT» menu displayed.

Press «C» for «I/O Configuration».



Display of the «I/O CONFIGURATION» menu.

I/O CONFIGURATION	
Racks	Slots
0 (P8)	C B x 9 x 7 6 5 x x x 1 0
3 (E12)	C B A 9 x x x x x 3 2 1 0
4 (E12)	x x x x x x x x 3 2 1 0
-- Commands <R#> or <C##> or <F11> --	
.../...	EXIT

Enter the command «R0».



The message «R0: Main rack, 8 slots» is displayed on the dialogue line.

Enter the command «C05».



The message «32 relayed outputs unit» is displayed on the dialogue line.

Enter the command «C08».



The message «Empty slot!» is displayed on the dialogue line.

Enter the command «C0B».



The message «32 DC inputs unit» is displayed on the dialogue line.

## 18.2.6 Software Backup

Via a CPU serial interface, this module is used to:

- back up the user programme on a peripheral (diskette drive or tape reader/punch),
- check the backed up programme against the source programme,
- restore the backed up programme.

### Requirements

Transmission data rate correct and communication parameters correct on the peripheral.

CNC connected to the peripheral (on a CPU serial interface)

«PLC APPLICATION MANAGEMENT» menu displayed.

### Actions

Type «A» for software backup.



The «APPLICATION BACKUP» menu is displayed.

APPLICATION BACKUP							
<ul style="list-style-type: none"><li>- Download the software</li><li>- Load the software</li><li>- Check-up of the downloading</li></ul>							
<p>-- Enter a command (Exit F11) --</p>							
.../..							EXIT

### Exit from the Procedure

Press «F11».



Return to the «PLC APPLICATION MANAGEMENT» menu.

### 18.2.6.1 Software Backup

#### Actions

Set the peripheral to download mode.



Type «D» to download the software.

The message «Downloading in progress» is displayed.

The message is cleared when downloading is completed.

#### Exit from the Procedure

Press «F11».



EXIT

### 18.2.6.2 Check of Downloading

#### Actions

Type «C» to check downloading.



The message «Waiting...» is displayed.

Set the peripheral to the mode suited to checking the backup.

The message «Check-up in progress» is displayed.

The message is cleared at the end of the check.

#### Exit from the Procedure

Press «F11».



EXIT

### 18.2.6.3 Loading the programme

#### Actions

Type «L» to load the software.



The message «Waiting...» is displayed.

Set the peripheral to load mode.

The message «Loading in progress» is displayed.

The message is cleared when loading is completed.

#### Exit from the procedure

Press «F11».



EXIT

### 18.2.7 Software Load/Download to/from PLCTOOL

For further details, refer to the PLCTOOL - Ladder Language Programming Tool Manual.

Check the state of parameter P112 (see Parameter Manual).

Check that the PLCTOOL link is enabled (see Sec. 18.2.4.4).

Check that the data rates on the CNC and PC are the same.

Install a connecting cable between the parameterised serial line and the PC.

Start the load or unload procedure from PLCTOOL.

*REMARK The current loading procedure does not require any action on the CNC.*

### 18.2.8 Ladder Animation

Ladder animation is used to dynamically display the Ladder contacts of a PLC programme. This gives a graphic display of changes in a Ladder network.

#### Requirements

«PLC APPLICATION MANAGEMENT» menu displayed.

#### Actions

Type «A» for Animation.



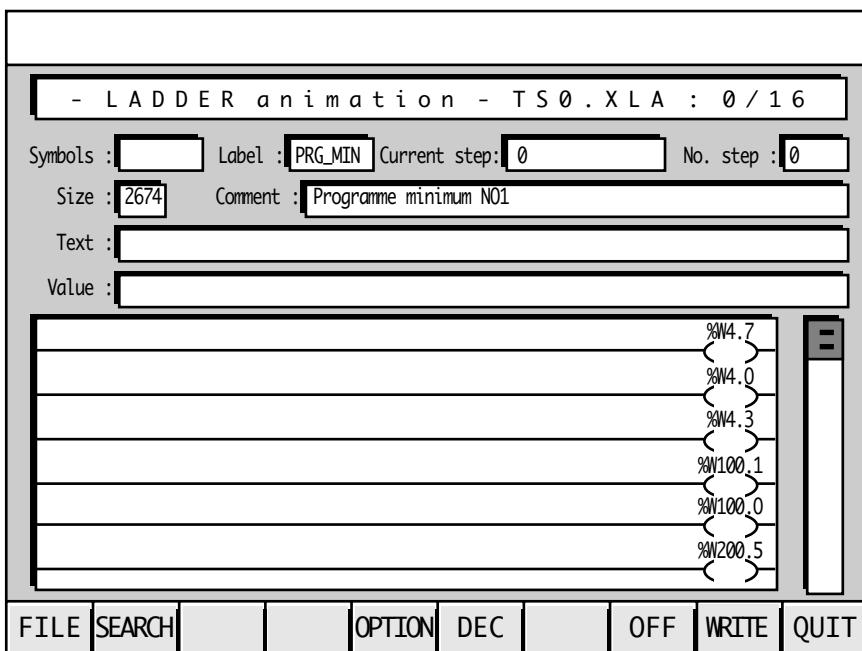
The LADDER Directory is displayed.

LADDER directory				
Name	Size	Date	Time	
TS0 .XLA	2674	08/07/94	13:28:13	
1 File(s), 2674 Byte(s)				
				QUIT



Select the module to be animated using the arrow keys then confirm.

The Ladder grid to be animated is displayed.



When it opens, the component displayed is animated.

In animation mode, the «Step Var.» field becomes «Current Step» and displays the current value of the step variable.

If this value is equal to the step No. (drip-feed component) or if no step variable was defined, the field background takes on the active colour.

#### Interpretation of the Colours

State	Colour monitor	Monochrome monitor
Active	Red	White
Inactive	Black	Black
Indeterminate	Flashing	Flashing

## Browsing Through the Application

Operation	Command
Focus on the next object	 or   or   or  
Move around in the Ladder grid	Focus on the Ladder grid then  or  or  or 
Display the next Ladder component	Focus on the vertical scroll bar then 
Display the previous Ladder component	Focus on the vertical scroll bar then 
Display the last Ladder component	Focus on the vertical scroll bar then 
Display the first Ladder component	Focus on the vertical scroll bar then 

## Stop Animation

Press «F9»  

Ladder animation is stopped. A new key appears in the key bar: «CLEAR».

## Initialise the Ladder Grid

Press «F8».  

All the Ladder grid components are forced to inactive state.

**Load a new Ladder module**

Press «F2».



FILE

The «LADDER Directory» is displayed.

Name	Size	Date	Time
TS0 .XLA	2674	08/07/94	13:28:13

1 File(s), 2674 Byte(s)

QUIT

Select the module to be animated using the arrow keys  
then confirm.



*REMARK To return to the previous Ladder module, press «QUIT».*

The Ladder grid to be animated is displayed.

**Animate the Module**

Press «F9».



ON

The Ladder module is animated.

**Define Options**Decimal/hexadecimal display

Press «F7».



DEC

Or

Press «F7».



HEX

The numerical values are displayed in decimal or hexadecimal format.

### Gridlines

Press «F6».



OPTION

A new menu bar is displayed.

Press «F2».



GRID  
ON

Or

Press «F2».



GRID  
OFF

**Gridlines with the size of the cells are displayed or cleared.**

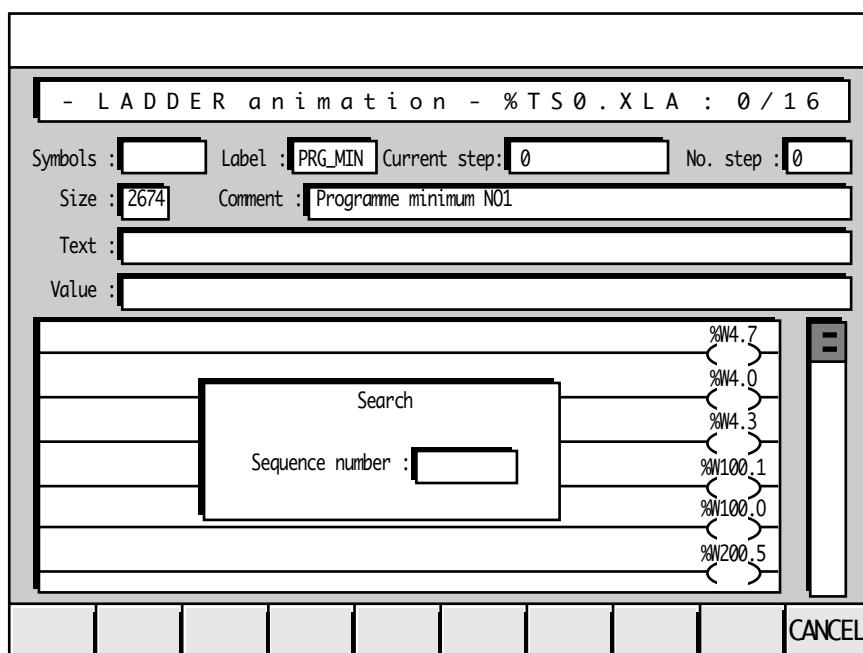
### **Search For a Component**

Press «F3».



SEARCH

The «Search» dialogue box opens.



Enter the number of the component to be searched for.



The specified component is displayed.

Cancel the procedure

Press «F11».



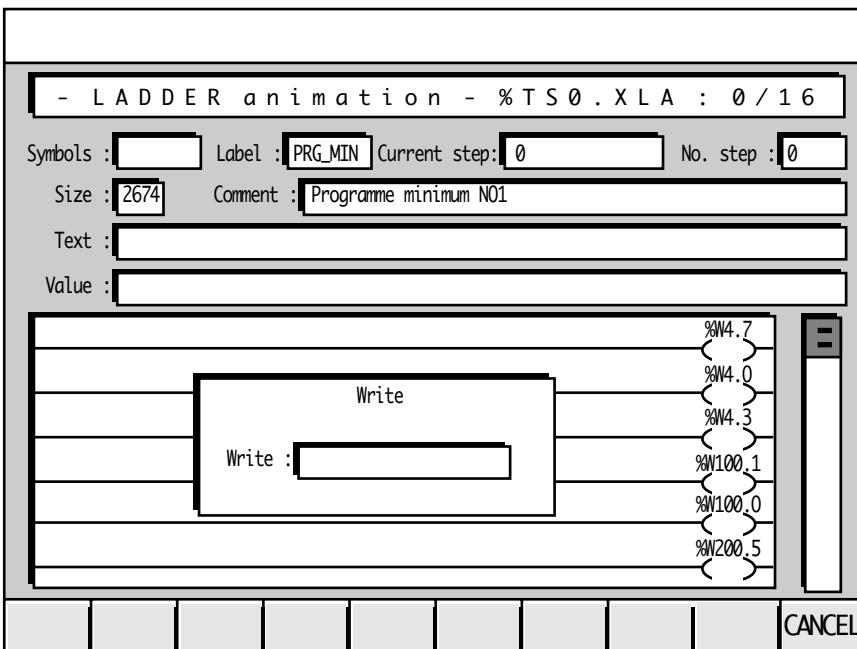
**Write a Variable**

Press «F10».



WRITE

The «Write» dialogue box is displayed.



Enter the name of the variable to written and its value.

*REMARK All the variables except %lxx.x input variables are accessible for write.*Cancel the procedure

Press «F11».



CANCEL

Cancel the Procedure

Press «F11».



QUIT

Return to the «PLC APPLICATION MANAGEMENT» menu.



---

# 19 Errors and Diagnostic

## 19.1 List of Hardware Errors

ERR\_BUS\_SBCE: Serial bus error.

## 19.2 List of Configuration Errors

ERR\_CONFIG\_SBCE: I/O card incorrectly inserted or configuration error.

## 19.3 List of Programming Errors

ERR\_HTR: Real time clock overrun.

ERR\_ACCESS\_VIOLATION: Attempt to read or write in a prohibited area.



---

## A Lists of Functions

<b>A.1 List by Themes</b>	<b>A - 3</b>
A.1.1 General Purpose Functions	A - 3
A.1.2 Task Management	A - 4
A.1.3 Transparent Mode	A - 4
A.1.4 Analogue Input/Output Management	A - 4
A.1.5 Explicit Read/Write of a Card	A - 4
A.1.6 Interrupt Input Management	A - 5
A.1.7 Serial Line Management	A - 5
A.1.8 Timer Control	A - 5
A.1.9 Date-Time Stamp Management	A - 5
A.1.10 Exchanges by Protocol	A - 5
A.1.11 Call Modules in C	A - 5
<b>A.2 Alphanumeric List</b>	<b>A - 6</b>

A



## A.1 List by Themes

### A.1.1 General Purpose Functions

Function	Description	Page
atoi()	Convert an ASCII string to a signed integer of 32 bits	6-3
atoj()	Convert an ASCII string to a signed integer of 32 bits	6-4
bcd_bin()	BCD —> binary code conversion	6-5
bin_bcd()	Binary —> BCD code conversion	6-6
bit()	Separate bits into bytes	6-7
call()	Jump to a module label with return	6-19
cpyarg()	Read the parameters stored on the stack	6-8
cpyb()	Copy one or more bytes	6-9
cpyl()	Copy one or more long words	6-11
cpyw()	Copy one or more words	6-10
diagiq()	Set the self-test period	6-11
goto()	Jump to a module label without return	6-19
iota()	Convert a signed integer to an ASCII string	6-12
itostr()	Convert an unsigned integer to an ASCII string	6-12
oct()	Concatenate a byte into bits	6-13
putkey()	Simulate operator panel keyboard	6-15
qcktool()	Shortest Path Calculation	6-15
R_E42000	Read n variables E42000	6-31
rchb()	Search for the value of a byte	6-16
rchl()	Search for the value of a long word	6-17
rchw()	Search for a value of a word	6-16
return()	Return to the calling module or network	6-18
sema()	Flag	6-20
setb()	Write one or more bytes	6-20
setl()	Write one or more long words	6-22
setx()	Write one or more words	6-21
sp()	Call an %SP module	6-22
sprintf()	Format a character string	6-24
spy()	Call an %SP module with %Y local variables	6-23
sqrt()	Integer square root	6-25
sscanf()	Analyse an ASCII string	6-25
strcmp()	Compare two character strings	6-26
strcpy()	Copy a character string	6-27
strlen()	Calculate the length of a string	6-27
swapl()	Swap the four bytes of a long word	6-29
swapw()	Swap the even and odd bytes of a word	6-28
tooldyn()	Change a tool wear offset	6-30

Function	Description	Page
W_E42000	Write n variables E42000	6-32
y_init()	Initialisation of the base associated with the %Y variables	6-33

## A.1.2 Task Management

Function	Description	Page
csbegin()	Start a critical section	7-3
csend()	End a critical section	7-3
tfstart()	Start a %TF task	7-4
tfstop()	Stop a %TF task	7-4
whtr()	Suspend a %TF task	7-3

## A.1.3 Transparent Mode

Function	Description	Page
inig()	Graphic init	8-17
pcur()	Position the cursor	8-7
print()	Display a buffer	8-8
printf()	Display a string with format	8-9
putchar()	Display a character	8-7
putimage()	Position and display an image	8-16
puts()	Display a string without formatting	8-8
scanc()	Close a keyboard acquisition	8-16
scand()	Read and convert a decimal number	8-14
scano()	Open a keyboard acquisition	8-12
scans()	Read a string	8-13
scanu()	Open a numerical keypad acquisition	8-13
scanx()	Read and convert a hexadecimal number	8-15

## A.1.4 Analogue Input/Output Management

Function	Description	Page
anaa()	Reassign an analogue card	9-7
anai()	Read an analogue input	9-6
anao()	Write an analogue output	9-5
anas()	Configure an analogue I/O card	9-3

## A.1.5 Explicit Read/Write of a Card

Function	Description	Page
read_i()	Explicit read of an input card	10-3
write_q()	Explicit write of an output card	10-4

### A.1.6 Interrupt Input Management

Function	Description	Page
iti_gr()	Associate an interrupt input with an axis group	11-5
itictl()	Configure an interrupt input	11-6
itiget()	Read an interrupt input	11-8
thiti()	Associate a %TH task with an IT input	11-9

### A.1.7 Serial Line Management

Function	Description	Page
comctl()	Control the serial line driver	12-11
comf()	Select data rates and formats	12-4
comin()	Read the reception buffer	12-7
comout()	Send a buffer	12-6
comreg()	Read the status of a serial line	12-10

### A.1.8 Timer Control

Function	Description	Page
thtimer()	Associate a %TH task with a timer	13-1

### A.1.9 Date-Time Stamp Management

Function	Description	Page
tmget()	Read the current date	14-1
dtget()	Read the current date and day in week	14-2

### A.1.10 Exchanges by Protocol

Function	Description	Page
neti()	Read a request from a distant server	15-36
neto()	Send a request to a distant server	15-34
netst_ad	Answer a STATUS request	15-40
uniti()	Read a response	15-30
unito()	Send a request	15-29
setcomw	Set up the common service words	15-39

### A.1.11 Call Modules in C

Function	Description	Page
exec()	Call an executable module	16-3
exechdl()	Identify an executable module	16-4

## A.2 Alphanumerical List

Function	Description	Page
anaa()	Reassign an analogue card	9-7
anai()	Read an analogue input	9-6
anao()	Write an analogue output	9-5
anas()	Configure an analogue I/O card	9-3
atoi()	Convert an ASCII string to a signed integer of 32 bits	6-3
atoj()	Convert an ASCII string to a signed integer of 32 bits	6-4
bcd_bin()	BCD → binary code conversion	6-5
bin_bcd()	Binary → BCD code conversion	6-6
bit()	Separate bits into bytes	6-7
call()	Jump to a module label with return	6-19
comctl()	Control the serial line driver	12-11
comf()	Select data rates and formats	12-4
comin()	Read the reception buffer	12-7
comout()	Send a buffer	12-6
comreg()	Read the status of a serial line	12-10
cpyarg()	Read the parameters stored on the stack	6-8
cpyb()	Copy one or more bytes	6-9
cpyl()	Copy one or more long words	6-11
cpyw()	Copy one or more words	6-10
csbegin()	Start a critical section	7-3
csend()	End a critical section	7-3
diagiq()	Set the self-test period	6-11
dtget()	Read the current date and day in week	14-2
exec()	Call an executable module	16-3
execndl()	Identify an executable module	16-4
goto()	Jump to a module label without return	6-19
inig()	Graphic init	8-17
itictl()	Configure an interrupt input	11-6
itiget()	Read an interrupt input	11-8
iti_gr()	Associate an interrupt input with an axis group	11-5
itoa()	Convert a signed integer to an ASCII string	6-12
itostr()	Convert an unsigned integer to an ASCII string	6-12
neti()	Read a request from a distant server	15-36
neto()	Send a request to a distant server	15-34
netst_ad	Answer a STATUS request	15-40
oct()	Concatenate a byte into bits	6-13
pcur()	Position the cursor	8-7
print()	Display a buffer	8-8
printf()	Display a string with format	8-9
putchar()	Display a character	8-7

Function	Description	Page
putimage()	Position and display an image	8-16
putkey()	Simulate operator panel keyboard	6-15
puts()	Display a string without formatting	8-8
qcktool()	Shortest path calculation	6-15
rchb()	Search for the value of a byte	6-16
rchl()	Search for the value of a long word	6-17
rchw()	Search for a value of a word	6-16
read_i()	Explicit read of an input card	10-3
return()	Return to the calling module or network	6-18
R_E42000	Read n variables E42000	6-31
scanc()	Close a keyboard acquisition	8-16
scand()	Read and convert a decimal number	8-14
scano()	Open a keyboard acquisition	8-12
scans()	Read a string	8-13
scanu()	Open a numerical keypad acquisition	8-13
scanx()	Read and convert a hexadecimal number	8-15
sema()	Flag	6-20
setb()	Write one or more bytes	6-20
setcomw	Set up the common service words	15-39
setl()	Write one or more long words	6-22
setx()	Write one or more words	6-21
sp()	Call an %SP module	6-22
sprintf()	Format a character string	6-24
spy()	Call an %SP module with %Y local variables	6-23
sqrt()	Integer square root	6-25
sscanf()	Analyse an ASCII string	6-25
strcmp()	Compare two character strings	6-26
strcpy()	Copy a character string	6-27
strlen()	Calculate the length of a string	6-27
swapl()	Swap the four bytes of a long word	6-29
swapw()	Swap the even and odd bytes of a word	6-28
tfstart()	Start a %TF task	7-4
tfstop()	Stop a %TF task	7-4
thiti()	Associate a %TH task with an IT input	11-9
thtimer()	Associate a %TH task with a timer	13-1
tmget()	Read the current date	14-1
tooldyn()	Tool wear offset	6-30
uniti()	Read a response	15-30
unito()	Send a request	15-29
whtr()	Suspend a %TF task	7-3

Function	Description	Page
write_q()	Explicit write of an output card	10-4
W_E42000	Write n variables E42000	6-32
y_init()	Initialise the base associated with the %Y variables	6-33

# Index

## Symbols

%I		%W18.W	3-43
	Organisation	%W1A.B	3-44
%INI		%W1E.B	3-44
		%W2.W	3-38
%Irc39.B		%W21.B	3-44
%Irc3A.W		%W22.W	3-45
%Irc3C.W	3-11, 3-20, 3-25	%W24.W	3-45
%Irc3E.W	3-10, 3-18, 3-19, 3-22	%W2A.W	3-45
	3-24, 3-25	%W2C.W	3-48
%I variable structure	3-10	%W30.L	3-49
%Q		%W34.L	3-50
	Organisation	%W38.0	3-50
%Qrc3B.0		%W3A.L	3-51
%Qrc3B.1		%W4.W	3-39
%Qrc3C.B		%W6.L	3-40
%Qrc3D.B		%W900.0	3-66
%Qrc3E.W		%WA.L	3-41
%Q variable structure	3-10	%WE.L	3-41
%R0.W	3-29, 8-4	%WE0.B to WE1F.B	3-51
%R12.W	3-33	%Wg00.W	3-61
%R14.0	3-35	%Wg02.B	3-65
%R15.B	3-34	%Wg03.B	3-62
%R16.B	3-34	%Y	3-70
%R17.B	3-35	%Y variable initialisation	6-33
%R18.B	3-35	*Serial line driver monitoring	12-13
%R19.B	3-35	32 discrete input card	3-18
%R1A.W	3-35	32 discrete output card	3-18
%R1C.W	3-36	32-24 discrete I/O card	3-19
%R2.W	3-29	32-24 I/O card	3-19
%R22.W	3-36	64-48 I/O card	3-22
%R24.L	3-36		
%R4.W	3-30		
%R6.L	3-31		
%R97C.W	3-65		
%R97F.0	3-65	Accessible objects	15-7
%R97F.1	3-65	Action area	4-4, 5-15
%R97F.2	3-65	Activation of Grafset steps	5-4
%RA.L	3-32	Alphanumeric character	8-25
%RE.L	3-32	Alphanumeric space	8-18
%Rg00.W	3-53	anaa	9-7
%Rg02.B	3-54	anai	9-6
%Rg03.B	3-54	Analogue card reassignment	9-7
%Rg04.W	3-55	Analogue I/O configuration	9-3
%Rg1E.W	3-55	Analogue input	9-3
%Rg20.L	3-56	Analogue inputs/outputs	16-20
%Rg24.W	3-58	Analogue output	9-3
%Rg7C.L	3-59	Animate module	18-25
%S	3-68	Animation	8-34
	Organisation	Answer to STATUS request	15-40
%TF	2-6	Application directory	18-15
%TF processing	2-8	Application structure	2-13
%TH	2-9	ASCII string analysis	6-25
%TH operation	2-9	Assignment operator	4-6
%TH priority	2-9	Assignment_operator	4-5
%TH-IT association	11-9	atoi	6-3
%TH/timer association	13-1	atoj	6-4
%TS	2-5	Axes in motion	3-31
%TS processing	2-8	Axis clamp	3-59
%W13.B	3-42	Axis group	3-53, 3-78
%W14.B	3-42	inputs	3-61, 3-81
%W15.B	3-42	outputs	3-43
%W16.B	3-42	selection	3-59
%W17.B	3-43	Axis unclamp	

## B

Background task	2-6
activity	18-8
management	16-22
operation	2-7
priority	2-7
states	2-6
Backward movement on path	3-51
Battery status	3-35
bcd_bin	6-5
bin_bcd	6-6
Binary_operator	4-5
bit	6-7
Bit variable	5-7
Bit_variable	4-3, 4-4, 5-15
Block diagram	
of machine processor card	1-7
of UCSII card	1-8
Blocking message	15-29
Branches	5-14
Browse through the application	18-24
Building a network	5-18
Bus status	3-11
Byte value search	6-16
Byte variable	4-7
C	
C language	16-3
Call	6-19
%SP module	6-22, 6-23
executable module	16-3
Canned cycle in progress	3-45
Card	
configuration	3-10
configuration variable	3-12
diagnostic	3-10
diagnostic variable	3-10
I/O update	2-5
identifier	3-10, 3-12, 3-16
input	16-8
output	16-8
priority	3-13
status	3-11
Change of sign	4-9
Character string	2-16
Check of unloading	18-21
Close a keyboard acquisition	8-16
CLOSE_DIRECTORY request	15-24
CNC access enable	3-14
CNC I/O update	2-3
CNC inputs	3-29, 3-72
CNC output	3-38, 3-75
CNC status	3-30
Code conversion	
BCD -> binary	6-5
Binary -> BCD	6-6
Coded character	8-5
Coercion field	3-8
Colour selection	8-24, 8-27
Combined operator	4-6
comctl	12-11
comf	12-4

comin	12-7	DIRECTORY request	15-22	<b>G</b>
Comment	4-3	Display		
Common elements of sequences	5-3	a buffer	8-8	G function status
Common word configuration	3-69	a character	8-7	3-54
Common word service	15-38	a string	8-8, 8-9s	General organisation of multicard CPU
Common word variables	3-68	an image	8-16	1-4
comout	12-6	management	8-3	single card CPU
Compact panel	3-27	DNC1000	15-3	1-5
Comparison	4-3	DNC1000 protocol	15-4	goto
Comparison operator	4-6	Downcounters	5-12	6-19
Comparison_operator	4-4	Draw a key bar	8-40	Grafset step
comreg	12-10	dtget	14-2	2-15, 5-3
Concatenate bytes into bits	6-11			Grafset step deactivation
Conditional action	5-9			5-4
Configuration errors	19-1			Graphic init
Contact	5-7	E30xxx	3-66	8-17
Convert		E33xxx	3-14	Graphic instructions
a decimal number	8-14	E40xxx	3-67	8-29
a hexadecimal number	8-15	E42000	6-31, 6-32	Graphic space
a signed integer	6-12	E42xxx	3-67	8-18
an ASCII string	6-3, 6-4	E43xxx	3-14	Group commands
an unsigned integer	6-12	Encoded M function		3-61
Copy		with report	3-55	Group status
a byte	6-9	without report	3-55	3-53
a character string	6-27	Endless loop in a programme	2-10, 2-11	<b>H</b>
a long word	6-11	Exchange		
a word	6-10	area	3-5, 3-72, 16-7	Handwheel assignment
Counters	5-12	mechanism	15-5	3-44
CPU control	18-9	protocol	15-3	Hardware errors
CPU monitoring	18-5	variable	8-4	19-1
cpyarg	6-8	with a remote station	15-34	Hardware task activity
cpyb	6-9	Exchanges	3-5	18-8
cpyl	6-11	by protocol functions	16-12	Header
cpyw	6-10	exec	16-3	5-3
Creating objects	15-9	exechdl	16-4	Hex_digit
Critical section		Executable C module	2-13	4-4
end	7-3	Executable module identification	16-4	Hexadecimal number acquisition
start	7-3	Explicit input card read	10-3	8-15
csbegin	7-3	Explicit output card write	10-4	Highlighted character
csend	7-3	Explicit read/write accesses	16-20	8-27
CTD_n	5-12	EXPORT	16-9	<b>I</b>
CTU_n	5-12	Export an object	16-9	
Current mode	3-34			Icons
Current reduction	3-51	<b>F</b>		8-36
Cursor		F_T cell	5-8	Immediate integers
movement	8-28	Falling trig	5-8	4-7
not visible	8-28	Feed rate potentiometer setting	3-62	Immediate_integer
steady	8-28	Feed stop per axis	3-51	4-4
Cyclic task	2-5	File *.XCX	2-13	IMPORT
<b>D</b>		File *.XLA	2-13	16-10
Date-time stamp	14-1	File load	18-22	Import an object
Date-time stamp function	14-1	File management	16-26	16-10
Debugging on CNC	18-3	File unload	18-22	Index
Decimal number acquisition	8-14	Flag	6-19	3-7
Decoded M function	3-56, 3-58	Flow control	12-14	Index field
Delete	8-29	Font format	8-26	3-7
Delete a file	18-16	Format a character string	6-24	Indirect addressing
DELETE_FILE request	15-19	Format volume	18-14	3-70
diagiq	6-11	Forward movement on path	3-51	Inhibited JOG increments
Dialogue error counter	3-11	Function	4-4	3-48
Digit	4-4	Function library	16-9	Inhibited modes
Directory management	16-30	Function_call	4-3, 4-4, 5-9, 5-15	3-49
		Function_name	4-4	iniq
				Initialisation
				2-3
				Initialise Ladder grid
				18-24
				Initialised axes
				3-32
				Initialising the table of constants
				2-15
				Input image
				3-10
				Input/output configuration
				18-17
				Input/output update
				1-6
				Internal computation format
				4-7
				Interpretation of colours
				18-23
				Interrupt input configuration
				11-6
				Interrupt inputs
				16-22
				IT/axis group association
				11-5
				iti_gr
				11-5
				itiCtl
				11-6
				itiGet
				11-8
				itoa
				6-12
				itostr
				6-10
				<b>J</b>
				JOG increment
				3-34, 3-42
				Jump with return
				6-19
				Jump without return
				6-19
				<b>K</b>
				Keyboard characters
				3-29

<b>L</b>		Organisation of %I and %Q variables %R and %W variables %S variables	3-15 3-67 3-69	Real-time clock Real-time tasks Reassign an analogue card	1-6 2-9 9-7
Label	4-3			Receive a buffer	12-7
Ladder module structure	2-15			Remote communication	15-3
Ladder subroutine	2-13	Output image	3-10	Requested mode	3-42
Ladder task	2-13	Overflow	4-9	Requested programme number	3-43
Latching commands	3-39	Overrun	2-10	Requester	15-4
LECTURE DE MESSAGES request	15-28			RESERVE_MEMORY request	15-27
Line assignment	11-5			Reserved variable	3-67
Line numbering	12-3			return	6-18
Literal elements	4-3			Return to calling program	6-18
Literal entities	4-3, 4-4	E10000 to E10031 E20000 to E20031 E30xxx, E40xxx and E42xxx	3-32 3-41 3-66	Rising trig	5-8
Local communication	15-3			RTS/CTS	12-12
Logical number field	3-6	pcur	8-7		
Logical/geographical address	3-12	PLC axes	17-1	<b>S</b>	
Long word value search	6-17	PLCTOOL enable	18-16	Saved variable	3-8, 16-7
Long word variable	4-7	PLCTOOL inhibit	18-16	Scan a network	5-16
		Pointers	3-70	scanc	8-16
<b>M</b>		Port	15-6	scand	8-14
main()	2-13	Position cursor	8-7	scano	8-12
Machine panel	3-24	Position image	8-16	scans	8-13
Machine panel extension card	3-25	Positioning of spaces	8-20, 8-21	scanu	8-13
Machine status	3-29	Positive JOG commands	3-40	scannx	8-15
MCC68K compiler	1-6	Principle of exchanges	3-5	Screen area filling	8-40
Message display	3-42	print	8-8	Screen drawing	8-34
Mnemonic	3-6, 3-8	printf	8-8	Screen enabled in PCNC configuration	3-35
Modal functions	3-56	Priority interrupts	11-3	Screen font	8-26
Module	2-13	Priority of operators	4-5	Screen reference system	8-38
Monitor and %TS task time	18-9	Programming errors	19-1	Self-test period	6-11
		Programming in C	16-5	sema	6-20
		Propagation of variables	4-7	Send a buffer	12-6
<b>N</b>		Pulse commands	3-38	Send a request	15-29, 15-34
Negative JOG commands	3-41	putchar	8-7	Send characters to the screen	8-4
neti	15-35	putimage	8-16	Separate bits into bytes	6-7
neto	15-34	putkey	6-15	Sequence header	2-15
netst_ad	15-39	putkey()	8-4	Serial line initialisation	12-4
Network sequence	2-16, 4-3, 5-7	puts	8-8	Serial line management	16-13
No animation	8-34			Serial line status	12-10
Non-underlined character	8-27			Serial lines	12-3
Nonblocking message	15-28	<b>Q</b>		Server	15-4
Normal character	8-27	qcktool	6-15	setb	6-20
NUM.H	16-9	QVN axis pair	3-50	setcomw	15-39
NUM.OBJ	16-9	QVN axis reference	3-50	setl	6-22
Numerical_assignment	4-3, 4-4, 5-9, 5-16			setw	6-21
Numerical_comparison	5-8	<b>R</b>		Shift screen origin	8-35
Numerical_expression	4-4	R_E42000	6-31	Signed_number	4-4
Numerical_variable	4-4	R_T cell	5-8	Size field	3-7
		Rack identifier	3-17	Software backup	18-20
<b>O</b>		rchb	6-15	Software initialisation	8-23
Object type requests	15-7	rchl	6-16	Software load	18-22
oct	6-13	rchw	6-16	Software unload	18-22
On-the-fly measurement	11-3	Read		sp	6-22
OPEN_DIRECTORY request	15-21			Spindle controls	3-45
Operational errors	2-10	analogue input	9-6	Spindle potentiometer	3-44
Operator		answer	15-30, 15-35	Spindle speed	3-30
<<	4-6	date	14-1	Spindle speed setting	3-45
=	4-6	E42000	6-31	Spindle status	3-58
>>	4-6	interrupt input	11-8	Spindles in position	3-33
Operator panel keyboard simulation	6-15	parameters in stack	6-8	sprintf	6-24
Operator panel screen use	8-18	Read current date and day in week	14-2	spy	6-23
Optimum circular search	6-15	READ_BLOCK request	15-25	sqrt	6-25
Order of expressions	4-7	read_i	10-3	Square root	6-25
		READ_MEMORY_FREE request	15-20		
		READ_OBJECT request	15-16		

sscanf	6-25	uniti	15-30
Standard		unito	15-29
RS232	12-12	Unsaved variable	3-8, 16-7
RS422	12-13	Unsigned_number	4-4
RS485	12-13	Unsolicited data	15-6
Standard data	16-8	Use of string	2-16
Start a %TF task	7-4	Use of table of constants	2-15
Step	4-3	User area filling	8-38
Stop a %TF task	7-5	User drawing	8-32
Stop animation	18-24	User reference system	8-29, 8-31, 8-38
strcmp	6-26	User task	2-5
strcpy	6-27	Utility	18-3
String			
acquisition	8-13	<b>V</b>	
comparison	6-26	Variable	
initialisation	2-16	%I	3-9
length calculation	6-27	%Q	3-9
strlen	6-27	%R	3-29
Suspend a %TF task	7-4	%R, other	3-35
Swap the bytes of a long word	6-29	%W	3-38
Swap the bytes of a word	6-28	%Y	3-70, 6-23
swapl	6-29	Variable representation	3-6
swapw	6-28		
Symbol field	3-6	<b>W</b>	
System error management	3-66	W_E42000	6-32
System errors	3-65	W1D.B	3-44
System task	2-3	Watchdog	3-14
<b>T</b>		whtr	7-4
Table of constants	2-15	whtr(..)	2-7
Task		Window	
%INI	2-5	definition	8-18
%TF	2-6	dimensions	8-19
%TH	2-9, 11-3	selection	8-24
%TS	2-5	Word value search	6-17
Test area	4-3, 5-7, 5-9	Word variable	4-7
tfstart	7-4	Write	
tfstart(..)	2-6	analogue output	9-5
tfstop	7-5	byte	6-20
tfstop(..)	2-6	E42000	6-32
thiti	11-9	long word	6-22
thtimer	13-1	output cards	3-66
Timeouts	5-10	word	6-21
Timer	13-1	WRITE_BLOCK request	15-26
Timer function	13-1	WRITE_OBJECT request	15-18
tmget	14-1	write_q	10-4
TOF_n	5-10	wstrcmp	6-25
TON_n	5-10		
Tool correction	6-30	<b>X</b>	
Tool number	3-56	Xon/Xoff	12-14
tooldyn	6-30		
TP_n	5-10	<b>Y</b>	
Transfer current point	8-35	y_init	6-33
Transmission standards	12-12		
Transparent mode	8-3		
instructions	8-22		
management	16-14		
<b>U</b>			
Unary_operator	4-4		
Underlined character	8-27		
UNITE requests	2-5		
UNITE server	2-5		