

---

**NUM**  
**1020/1040/1050/1060**

**SUPPLEMENTARY**  
**PROGRAMMING**  
**MANUAL**

**0101938872/A**

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 contractual.

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

© Copyright NUM 2000.

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 2000 software NUM 1000 line.

This software is the property of NUM. Each memorized 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 authorized.

---

# Table of Contents

<b>1 Structured Programming</b>			1 - 1
	1.1	General	1 - 3
	1.2	Structured Programming Commands	1 - 6
	1.3	Example of Structured Programming	1 - 13
<b>2 Reading the Programme Status Access Symbols</b>			2 - 1
	2.1	General	2 - 3
	2.2	Symbols for Accessing the Data of the Current Block	2 - 3
	2.3	Symbols Accessing the Data of the Previous Block	2 - 11
<b>3 Storing Data in Variables L900 to L951</b>			3 - 1
	3.1	General	3 - 3
	3.2	Storing F, S, T, D, H and N in Variables L900 to L925	3 - 3
	3.3	Storing EA to EZ in Variables L926 to L951	3 - 4
	3.4	Symbolic Addressing of Variables L900 to L951	3 - 4
<b>4 Creating and Managing Symbolic Variable Tables</b>			4 - 1
	4.1	Creating Symbolic Variable Tables	4 - 3
	4.2	Symbolic Variable Management Commands	4 - 8
<b>5 Creating Subroutines Called by G Functions</b>			5 - 1
	5.1	Calling Subroutines by G Functions	5 - 3
	5.2	Inhibiting Display of Subroutines Being Executed	5 - 5
	5.3	Programming Examples	5 - 6
<b>6 Polynomial Interpolation</b>			6 - 1
	6.1	General	6 - 3
	6.2	Programming Segmented Polynomial Interpolation	6 - 3
	6.3	Programming Smooth Polynomial Interpolation	6 - 7
<b>7 Coordinate Conversions</b>			7 - 1
	7.1	General	7 - 3
	7.2	Using the Coordinate Conversion Matrix	7 - 3
	7.3	Application of Coordinate Conversion	7 - 5
	7.4	Example of Application Subroutine	7 - 6

<b>8</b>	<b>RTCP Function</b>		8 - 1
	8.1	General	8 - 3
	8.2	Using the RTCP Function	8 - 4
	8.3	Description of Movements	8 - 6
	8.4	Processing Related to the RTCP Function	8 - 9
	8.5	Use in JOG and INTERV Modes	8 - 11
	8.6	Restrictions and Conditions of Use	8 - 11
<b>9</b>	<b>N/M AUTO Function</b>		9 - 1
	9.1	General	9 - 3
	9.2	Using the N/M AUTO Function	9 - 7
	9.3	Procedure After Enabling the N/M AUTO Function	9 - 8
	9.4	Stopping and Restarting in N/M AUTO Mode	9 - 11
	9.5	Checks Included in N/M AUTO	9 - 12
	<b>Appendix A Table of Structured Programming Commands</b>		A - 1
	<b>Appendix B Table of Symbolic Variable Management Commands</b>		B - 1
	<b>Appendix C Table of Programme Status Access Symbols</b>		C - 1
	C.1	Addressing G and M Functions	C - 3
	C.2	Addressing a List of Bits	C - 3
	C.3	Addressing a Value	C - 3
	C.4	Addressing a List of Values	C - 4
	<b>Appendix D Table of Symbols Stored in Variables L900 to L951</b>		D - 1
	D.1	Symbols Stored in Variables L900 to L925	D - 3
	D.2	Symbols Stored in Variables L926 to L951	D - 3

## Record of Revisions

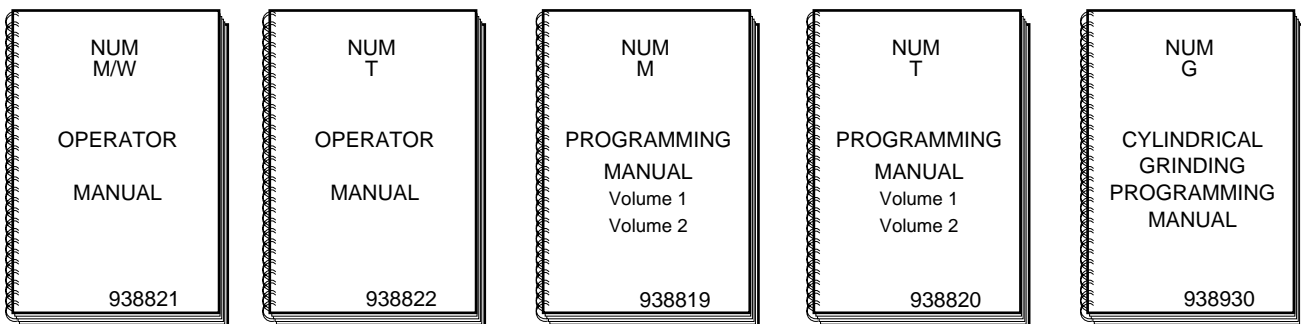
<b>DOCUMENT REVISIONS</b>		
<b>Date</b>	<b>Revision</b>	<b>Reason for revisions</b>
02-93	0	Document creation (conforming to software at index D)
01-95	1	<p>Conforming to software at index G</p> <p>Additions to the manual</p> <ul style="list-style-type: none"> <li>- RTCP function</li> <li>- N/M AUTO function</li> </ul> <p>Inclusion of changes</p> <p>Software at index E:</p> <ul style="list-style-type: none"> <li>- Addressing of the calling function by [.RG80] in a subroutine called by a G function</li> <li>- Addressing by [.IRDI(i)] defining the origin of angular offsets</li> </ul> <p>Software at index F:</p> <ul style="list-style-type: none"> <li>- Coordinate conversions</li> </ul>
06-97	2	<p>Conforming to software at index K</p> <p>Additions to the manual:</p> <ul style="list-style-type: none"> <li>- Smooth polynomial interpolation</li> <li>- For addressing by [.IBX(i)], [.IRX(i)] and [.IBI(i)], [.IRI(i)], added indexes 10, 11, 4 and 5 related to G21 and G22.</li> </ul> <p>Inclusion of changes</p> <p>Software at indexes H and J:</p> <ul style="list-style-type: none"> <li>- Update of N/M AUTO function</li> </ul>
04-00	A	Miscellaneous corrections



## NUM 1020/1040/1060 Documentation Structure

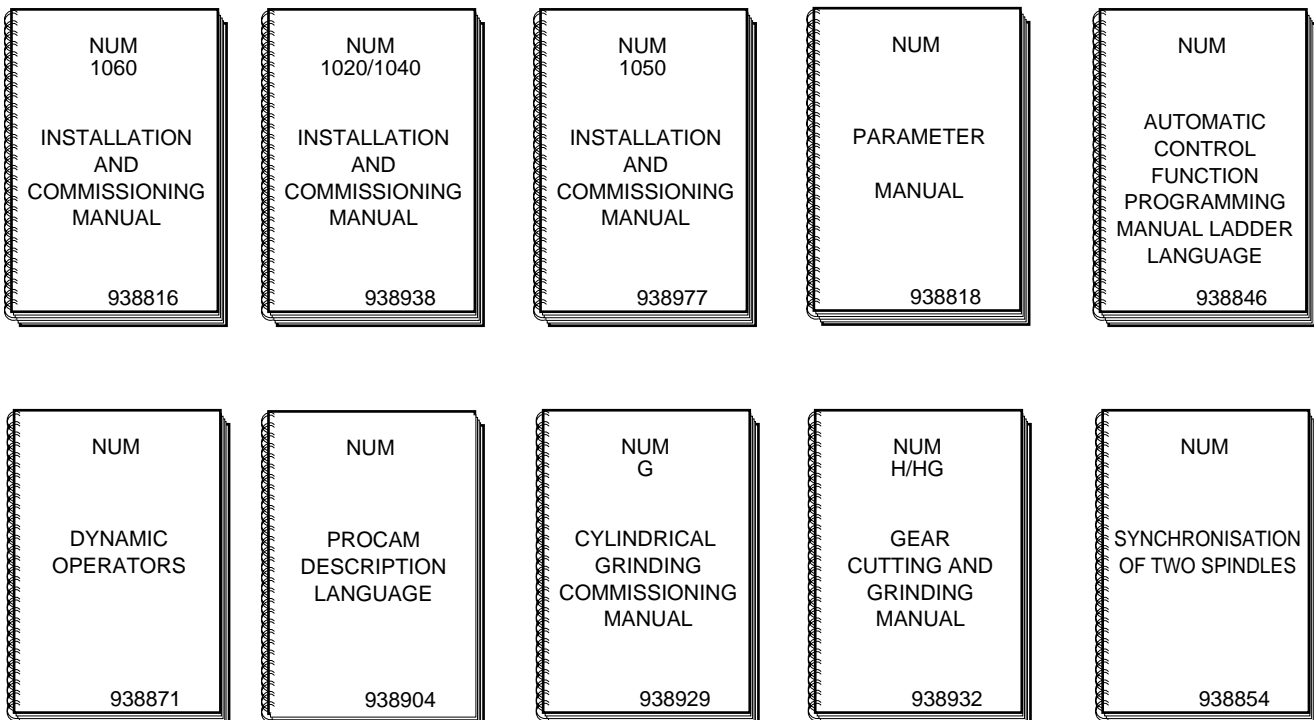
### User Documents

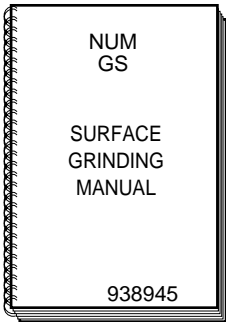
These documents are designed for use of the CNC.



### Integrator Documents

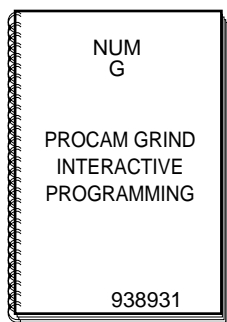
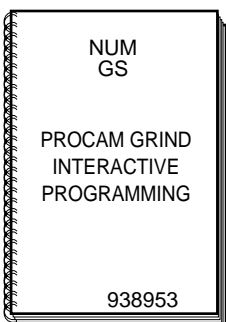
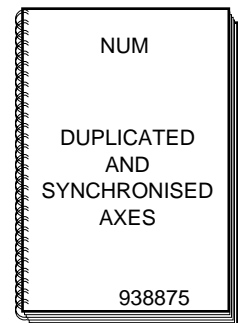
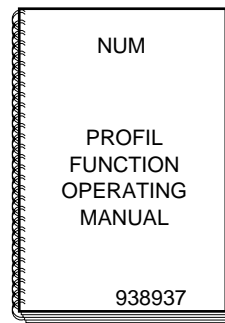
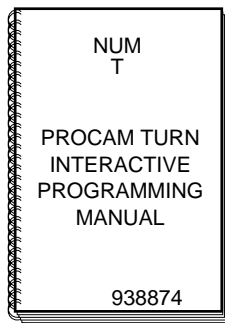
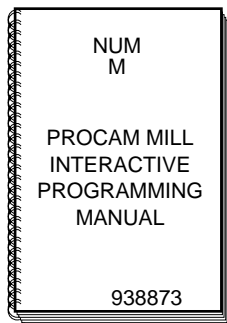
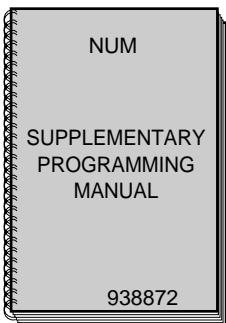
These documents are designed for setting up the CNC on a machine.





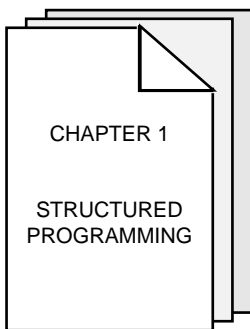
## Special Programming Documents

These documents concern special numerical control programming applications.

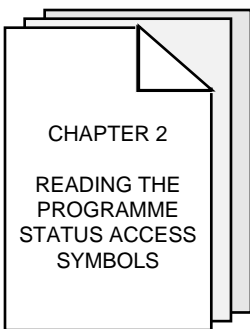


## Supplementary Programming Manual

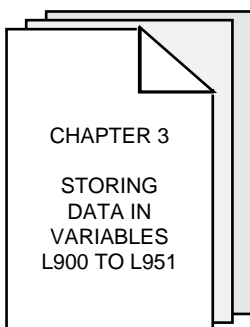
### Manual Contents



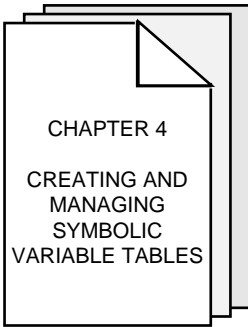
Presentation of the commands used for structured programming of branches and loops.



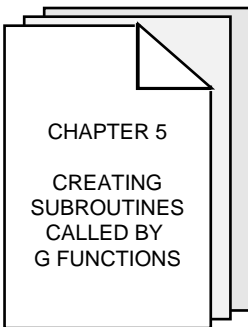
Presentation of the symbols giving visibility into the programmed functions and programme context during call of a cycle by a G function.



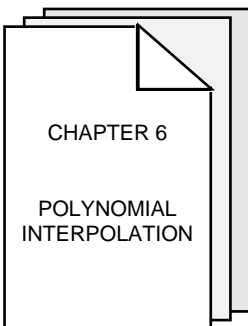
How to store values related to the arguments or functions programmed in variables L900 to L951 when calling a cycle by a G function.



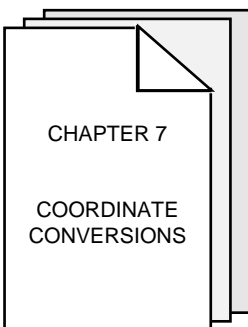
How to create and manage symbolic variable tables for storing functions and cutting paths.



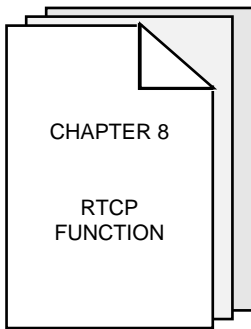
How to create subroutines called by G functions.



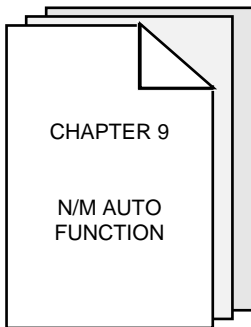
How to specify tool paths by polynomials.



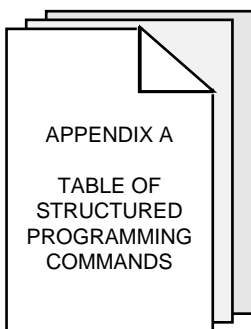
Coordinate conversions using a square matrix.



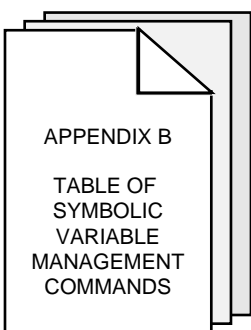
Possibility of controlling the movements of a machine to position the tool with respect to the part and pivot it around its centre.



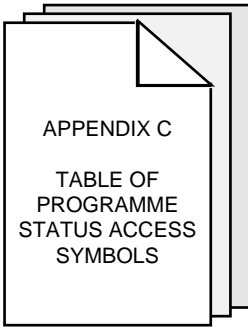
Possibility of controlling the N/M AUTO axes while the other machine axes follow a programmed path.



Presents the structured programming commands in table form.

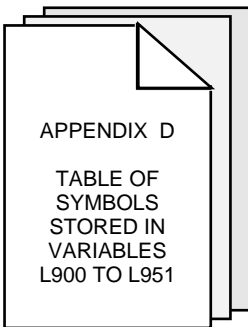


Presents the symbolic variable management commands in table form.



Presents the programme status access symbols in table form:

- G function addressing,
- M function addressing,
- addressing a list of bits,
- addressing a value,
- addressing a list of values.



Presents lists of symbols stores in variables L900 to L951 in table form.

- Symbols stored in variables L900 to L925.
- Symbols stored in variables L926 to L951.

## Using the Supplementary Programming Manual

### Syntax Conventions

The command lines (blocks) used in programming include commands, symbols, variables, functions and/or arguments.

A particular syntax is used for each of the elements described herein. The applicable syntax rules describe how to write the programme blocks.

Certain syntaxes are given on one or more lines. Writing is simplified by use of the following conventions:

- the functionality(ies) to which the syntax relates is (are) highlighted by the use of bold face characters,
- «..» or lower case letters after one or more capital letters, addresses or signs replace a numerical value (e.g. N..),
- the ellipsis «...» replaces a character or address string similar to that preceding it in the block (e.g. [Symb1]/[Symb2]...),
- «xx» after one or more address letters replaces alphanumeric characters (e.g. [.IBxx(i)]),
- «xxx» after an address letter replaces numerical values (e.g. Gxxx).

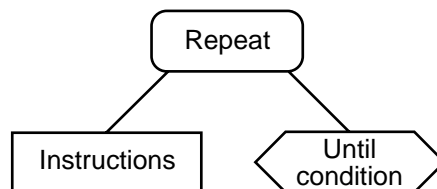
### Examples

Syntax for creating a symbolic variable table

```
P.BUILD [TAB(7,NB)] H.. N.. +n N..+n
```

Syntax of a «repeat until» loop and its graphic representation

```
REPEAT  
  (instructions)  
UNTIL (condition)
```



## **Index**

The index at the end of the volume gives access to information by keywords.

## **Questionnaire**

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

## **Agencies**

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

---

# 1 Structured Programming

---

<b>1.1 General</b>		1 - 3
	1.1.1	Commands Used in Structured Sequences 1 - 3
	1.1.2	General Syntax Rules 1 - 3
	1.1.3	Nesting and Branches 1 - 5
<b>1.2 Structured Programming Commands</b>		1 - 6
	1.2.1	Condition Graph 1 - 6
	1.2.2	Instruction Execution Conditions 1 - 7
	1.2.3	REPEAT UNTIL Loops 1 - 8
	1.2.4	WHILE Loops 1 - 9
	1.2.5	Loops with Control Variable 1 - 10
	1.2.6	Exiting the Loop 1 - 12
<b>1.3 Example of Structured Programming</b>		1 - 13

---



The system provides the possibility of programming structured branches and loops, making the programmes easier to read and simplifying the programming of complex part programmes.

The programming tools described in this chapter are used to create subroutines called by G functions (see Chapter 5).

## 1.1 General

A structured sequence always begins and ends with keywords.

It begins with one of the following keywords:

IF  
REPEAT  
WHILE  
FOR

It ends with:

ENDI for IF  
UNTIL for REPEAT  
ENDW for WHILE  
ENDF for FOR

The word ELSE can be interposed between the words IF and ENDI.

### 1.1.1 Commands Used in Structured Sequences

- Conditional execution of instructions: IF, THEN, ELSE, ENDI
- Repeat until loops: REPEAT, UNTIL
- While loops: WHILE, DO, ENDW
- Loops with control variables: FOR, TO, DOWNT0, BY, DO, ENDF
- Exit from a loop: EXIT

### 1.1.2 General Syntax Rules

The words IF, REPEAT, WHILE, FOR, ENDI, UNTIL, ENDW and ENDF must be the first words in a block (no sequence number).

The words IF, REPEAT, THEN, ELSE, UNTIL, WHILE, DO and DOWNT0 must always be followed by a space, e.g.:

WHILEL0 <3 is not recognised by the system. The required syntax is WHILE L0 <3.

The words DO and THEN must immediately follow the condition. Alternately, if these two words are not located on the same line as the words IF, WHILE and FOR, they must be the first words on the next line.

Blocks with sequence numbers (N..) are allowed in loops.

Blocks beginning with the words ENDI, ENDW, ENDF, EXIT or UNTIL must not include ISO programming functions.

One of words DO, THEN or ELSE can be followed by ISO programming functions in a same block.

Example:

```
WHILE L1 < 3 DO G91 X12
```

or

```
WHILE L1 < 3  
DO G91 X12
```

The following sequence is refused:

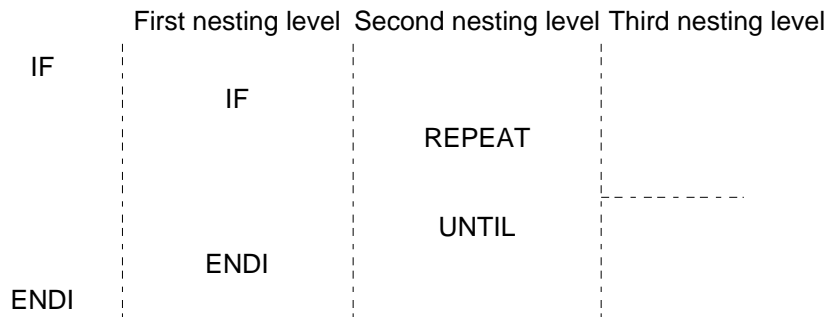
```
WHILE L0 < 3 G91 X10  
DO                  
          Not allowed in a  
          conditional instruction
```

### 1.1.3 Nesting and Branches

#### Nesting

Fifteen structured nesting levels are possible independently of subroutine calls by function G77 ...

Example:



#### Branches

Programming of a conditional or unconditional branch by G79 ... is allowed in a structured sequence, but must branch to the lowest nesting level of the current programme or subroutine.

Example:

```

%1
IF
  WHILE
    G79 N100 good
    G77 H2
    G79 N50 bad
    G79 N30 bad
    N30
  ENDW
  N50
ENDI
N100
M02

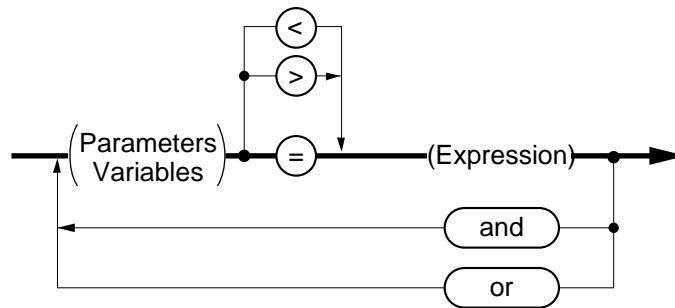
%2
G79 N100 good
  REPEAT
    IF
      G79 N100 good
      G79 N50 bad
    ENDI
  N50
UNTIL
N100
    
```

## 1.2 Structured Programming Commands

### 1.2.1 Condition Graph

A condition must follow one of words IF, WHILE or UNTIL and must be located in the same block. If the block contains limits and a possible increment, they must follow the word FOR.

#### Condition Graph



**Variables:** All the variables used in parametric programming:  
L variables, E parameters and symbolic variables.

**Expression:** Sequence of parameters and immediate values connected by symbols +, -, \*, /, !, & (see Chapter 6 of the Programming Manual). The operations are calculated in sequence from left to right.

## 1.2.2 Instruction Execution Conditions

### Syntax

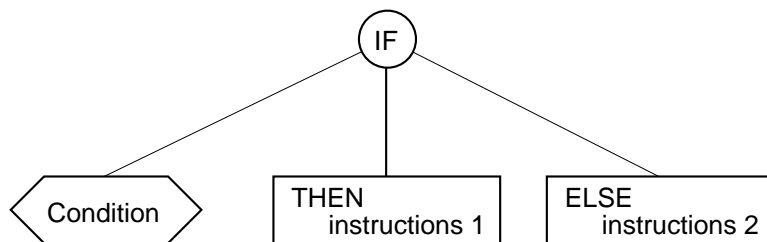
```

IF(condition) THEN
  (instructions 1)
ELSE
  (instructions 2)
ENDI
    
```

If the condition is true, «instructions 1» are executed. Else, «instructions 2» are executed.

The word ELSE is optional.

### Graph



### Example

```

IF E70000 > 100 AND E70000 < 200 THEN
  G77 H100
ELSE
  G77 H500
ENDI
    
```

The word THEN may be programmed at the beginning of the next block and followed by the functions to be executed.

Example:

```

IF L4 < 8
THEN L6 = L2+1 XL6
ENDI
    
```

### 1.2.3 REPEAT UNTIL Loops

#### Syntax

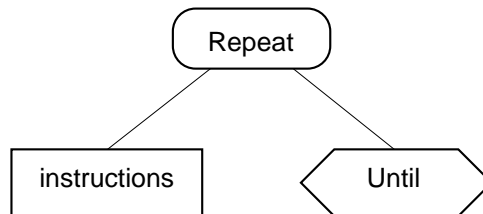
```

REPEAT
    (instructions)
UNTIL (condition)
```

The instructions are executed repetitively until the condition becomes true.

«CHANGE» Even if the condition is false at the beginning, the instructions are executed once.

#### Graph



#### Example

Wait for a correct answer

```

REPEAT
    $ EXIT FROM THE PROGRAMME (Y/N) ?
    [ANSWER]= $
UNTIL [ANSWER] = 14 OR [ANSWER] = 25
```

The answer «Y» returns the value 25 and the answer «N» returns the value 14 (ranks of the letters Y and N in the alphabet)

## 1.2.4 WHILE Loops

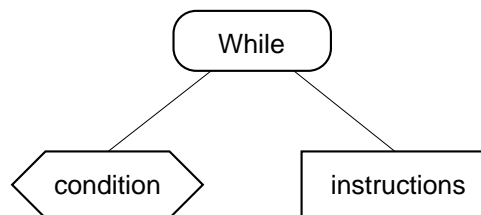
1

### Syntax

```
WHILE (condition) DO
  (instructions)
ENDW
```

The instructions are executed while the condition remains true. Unlike REPEAT UNTIL loops, the instructions are not executed if the condition is false at the beginning.

### Graph



The word DO may be programmed at the beginning of the next block and followed by the functions to be executed.

### Example:

```
WHILE (condition)
DO XL6
ENDW
```

## 1.2.5 Loops with Control Variable

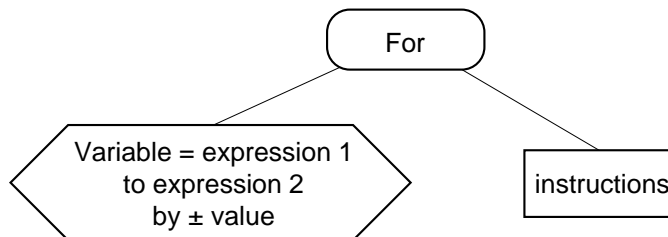
### Syntax

```
FOR (variable) = (expression 1) TO/DOWNTO (expression 2) BY (value) DO
(instructions)
ENDF
```

The instructions are executed for «variable = expression 1». Then «variable» is incremented (TO) or decremented (DOWNTO) by «value» before the «instructions» are executed again, and the process is continued until «variable» is equal to «expression 2».

The word «BY» is optional.

### Graph



The variable can be:

- an L programme variable,
- a symbolic variable,
- a parameter E80000, E81000 or E82000 (be careful about stopping computations for L100 to L199, Exxxx and the symbolic variables).

The expressions are positive or negative integers. The system rounds them off (down to 0.5 and up above 0.5).

With TO, the loop continues to be executed as long the current value of the variable is less than or equal to the final value (incrementing of the variable).

With DOWNTO, the loop continues to be executed as long the current value is higher than or equal to the final value (decrementing of the variable).

The value of BY is incremented or decremented from the variable each cycle (the default value of Y is equal to 1).

BY must always have a positive value. If the value of BY is negative, functions TO and DOWNTO are reversed.

### Example

Resetting the tool data

```
FOR L1=1 TO 20
DO
  L2=56000+L1
  EL2=0
ENDF
```

## 1.2.6 Exiting the Loop

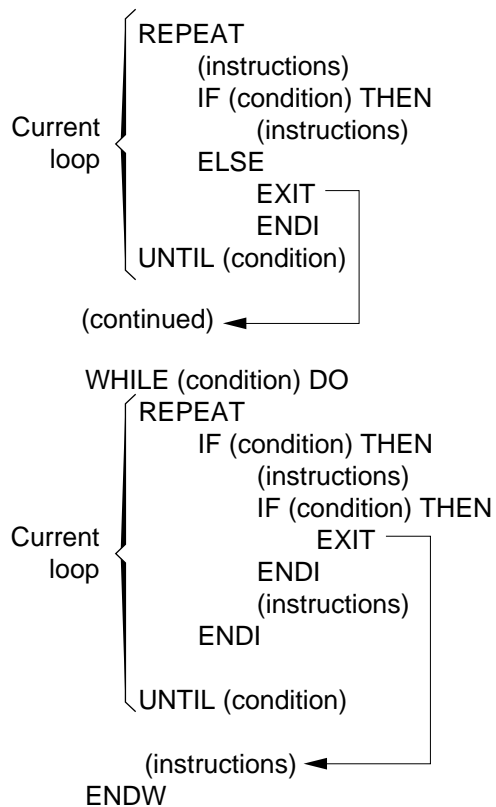
### Syntax

**EXIT**

The EXIT instruction is used to exit from an iteration loop (FOR, WHILE or REPEAT) and go to the next higher nesting level, ignoring any subsequent IF functions.

The word EXIT is used only in IF loops.

### Example

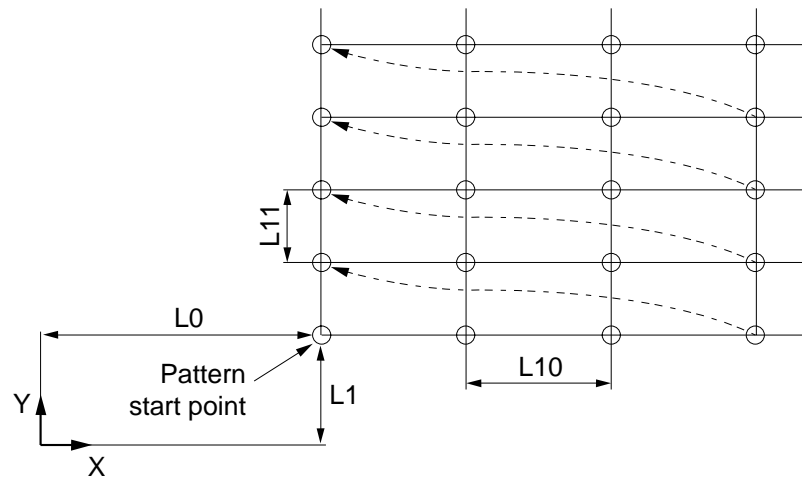


### 1.3 Example of Structured Programming

Hole drilling pattern

L2 = number of holes in X

L3 = number of holes in Y



```

%20
$ X START:
L0=$
$ Y START:
L1=$
$ X STEP:
L10=$
$ Y STEP:
L11=$
L4=0
WHILE L4 <> 25 DO
  REPEAT $ NUMBER OF POINTS IN X:
    L2=$
  UNTIL L2>0

  REPEAT $ NUMBER OF POINTS IN Y:
    L3=$
  UNTIL L3>0
  REPEAT $ PATTERN OF POINTS IN X:
    $=L2
    $+ /Y:
    $=L3
    $+ OK (Y/N) :
    L4=$
  UNTIL L4=14 OR L4=25
ENDW
$
N.. G00 G52 XO Z0
N.. T1 D1 M06
N..
FOR L5=1 TO L3 DO
  $ ROW POSITION:
  $=L5
  XL0 L6=L5-1*L11+L1 YL6
  G00 Z0
  FOR L4=1 TO L2 DO
    $ DRILL THE ROW POINT:
    $=L5
    $+COLUMN:
    $=L4
    L6=L4-1*L10+L0 XL6
    G01 Z-10 F50
    G00 Z0
  ENDF
ENDF
N..
M02

```

Test for answer yes: Y

Test for number of columns greater than 0

Test for number of rows greater than 0

Wait for answer: N (no) or Y (yes)

Loop on rows

Loop on columns

---

## 2 Reading the Programme Status Access Symbols

---

<b>2.1</b>	<b>General</b>	2 - 3
<b>2.2</b>	<b>Symbols for Accessing the Data of the Current Block</b>	2 - 3
	2.2.1	Symbols Addressing Boolean Values 2 - 3
	2.2.1.1	Addressing G Functions 2 - 4
	2.2.1.2	Addressing of M Functions 2 - 4
	2.2.1.3	Addressing a List of Bits 2 - 6
	2.2.2	Symbols Addressing Numerical Values 2 - 8
	2.2.2.1	Addressing a Value 2 - 8
	2.2.2.2	Addressing a List of Values 2 - 9
<b>2.3</b>	<b>Symbols Accessing the Data of the Previous Block</b>	2 - 11

---



The programming tools described in this chapter are required for creating subroutines called by G functions (see Chapter 5).

## 2.1 General

The programme status access symbols give visibility into the functions programmed in a block used to call a machining cycle by a G function. They also give information on the part programme context when the call is made.

These read-only symbols are accessible by parametric programming.

These symbols can be:

- symbols to access the data in the current block,
- symbols to access the data in the previous block.

Each symbol addresses a data item or list of data items with the form of a one-dimensional array or table.

## 2.2 Symbols for Accessing the Data of the Current Block

The symbols can be:

- symbols addressing Boolean values,
- symbols addressing numerical values.

### General Syntax

Variable = [ $\bullet$ symbol(i)]
-----------------------------------

Variable	L programme variable, symbolic variable [symb], E parameter.
$\bullet$ symbol(i)]	Symbol between square brackets, preceded by a decimal point, possibly followed by an index (i).

### 2.2.1 Symbols Addressing Boolean Values

The symbols addressing Boolean values associated with programmed functions are used to determine whether the functions are active or not.

The Boolean values are defined by 0 or 1.

### 2.2.1.1 Addressing G Functions

[•BGxx] G function addressing.

The symbol [•BGxx] is used to determine whether the G functions specified by xx are enabled or inhibited, e.g.:

[•BGxx]=0: function Gxx inhibited

[•BGxx]=1: function Gxx enabled

#### List of G Functions

[•BG00]	[•BG01]	[•BG02]	[•BG03]	[•BG17]	[•BG18]	[•BG19]
[•BG20]	[•BG21]	[•BG22]	[•BG29]	[•BG40]	[•BG41]	[•BG42]
[•BG70]	[•BG71]	[•BG80]	[•BG81]	[•BG82]	[•BG83]	[•BG84]
[•BG85]	[•BG86]	[•BG87]	[•BG88]	[•BG89]	[•BG90]	[•BG91]
[•BG93]	[•BG94]	[•BG95]	[•BG96]	[•BG97]		

### 2.2.1.2 Addressing of M Functions

[•BMxx] M functions addressing.

The symbol [•BMxx] is used to determine whether the M functions specified by xx are enabled or inhibited, e.g.:

[•BMxx]=0: function Mxx inhibited

[•BMxx]=1: function Mxx enabled

#### List of M Functions

[•BM03]	[•BM04]	[•BM05]	[•BM07]	[•BM08]	[•BM09]	[•BM10]	[•BM11]
[•BM19]	[•BM40]	[•BM41]	[•BM42]	[•BM43]	[•BM44]	[•BM45]	[•BM48]
[•BM49]	[•BM62]	[•BM63]	[•BM64]	[•BM65]	[•BM66]	[•BM67]	[•BM68]
[•BM69]	[•BM997]	[•BM998]	[•BM999]				

**Example**

```

%100
N10 G00 G52 Z0 G71
N..
N40 G97 S1000 M03 M41
N50 M60 G77 H9000
N..
N350 M02

%9000
VAR
[GPLAN] [MGAMME] [MSENS] [GINCH] [GABS]
[XRETOUR] [YRETOUR] [ZRETOUR]
ENDV

$ CONTEXT SAVE
N10 [GPLAN]=17* [.BG17]
      [GPLAN]=18* [.BG18]+[GPLAN]
      [GPLAN]=19* [.BG19]+[GPLAN]
N20 [MGAMME]=40* [.BM40]
      [MGAMME]=41* [.BM41]+[MGAMME]
N30 [MSENS]=03* [.BM03]
      [MSENS]=04* [.BM04]+[MSENS]
      [MSENS]=05* [.BM05]+[MSENS]
N40 [GINCH]=70* [.BG70]
      [GINCH]=71* [.BG71]+[GINCH]
N50 [GABS]=90* [.BG90]
      [GABS]=91* [.BG91]+[GABS]
[XRETOUR]=E70000
[YRETOUR]=E71000
[ZRETOUR]=E72000
N60 G90 G70 G00 G52 Z0
N70 G52 X100 Y100 M05
N80 G52 G10 Z-500
N90 G52 Z0
N100 G52 Z [ZRETOUR]
N110 G52 X [XRETOUR]
N120 G52 Y [YRETOUR]
G[GPLAN] M[MGAMME]
M[MSENS] G[GINCH] G[GABS]

```

Tool check subroutine call

} Interpolation plane

Speed range

Direction of rotation

Inches

Absolute

Tool check position

Return to Z position

Restore context

### 2.2.1.3 Addressing a List of Bits

[•IBxx(i)] Addressing a list of bits.

The symbol [•IBxx(i)] addresses a list of bits corresponding to the items specified by xx.

The values are Boolean, i.e. 0 or 1.

The index (i) defines the rank of the element in the list.

- |            |  |
|------------|--|
| [•IBX(i)]  | <p>List of axes programmed in the current block.<br/>Index i = 1 to 11.<br/>This nonmodal list can remain stored and be read by parametric programming if the system is in state G999.</p> <ul style="list-style-type: none"> <li>i = 1: X axis</li> <li>i = 2: Y axis</li> <li>i = 3: Z axis</li> <li>i = 4: U axis</li> <li>i = 5: V axis</li> <li>i = 6: W axis</li> <li>i = 7: A axis</li> <li>i = 8: B axis</li> <li>i = 9: C axis</li> <li>i = 10: in G21: Y axis address index 10<br/>          in G22: Z axis address index 10</li> <li>i = 11: in G21: X axis address index 11<br/>          in G22: Y axis address index 11</li> </ul> |
| [•IBX1(i)] | <p>List of axes programmed from the beginning of the programme to the current block.<br/>Index i = 1 to 9: Same as list of axes programmed in the current block (see [•IBX(i)]).<br/>The axes programmed with respect to the measurement origin (G52) are cancelled in this list, but are included in it again if programming is resumed with respect to the programme origin.</p>   |
| [•IBX2(i)] | <p>List containing the last axes programmed (primary axes X Y Z or secondary axes U V W).<br/>Index i = 1 to 6. Except for axes A, B and C, the list is the same as for the axes programmed in the current block (see [•IBX(i)]).<br/>Programming an axis in a group cancels the equivalent axis in other group. For instance, programming axis V resets the bit relative to X and sets the bit for V.</p>   |

- [•IBXM(i)] Mirroring of the axes. The bit is set to indicate mirroring and is reset to indicate no mirroring.  
Index i = 1 to 9: same as the list of axes programmed in the current block (see [•IBX(i)]).
- [•IBI(i)] List of arguments I, J and K programmed in the current block.  
Index i = 1 to 5.  
List modal only in state G999.  
i = 1: argument I  
i = 2: argument J  
i = 3: argument K  
i = 4: in G21: J component address index 4  
in G22: K component address index 4  
i = 5: in G21: I component address index 5  
in G22: J component address index 5
- [•IBP(i)] List of arguments P, Q and R programmed in the current block.  
Index i = 1 to 3.  
List modal only in state G999.  
i = 1: argument P  
i = 2: argument Q  
i = 3: argument R

## 2.2.2 Symbols Addressing Numerical Values

The symbols addressing numerical values are used to read the modal data of the current block.

### 2.2.2.1 Addressing a Value

[•Rxx] Addressing a value.

The symbol [•Rxx] is used to address a value corresponding to the elements specified by xx.

[•RF]	Feed rate (units as programmed by G93, G94 or G95).
[•RS]	Spindle speed (G97: format according the spindle characteristics declared in machine parameter P7).
[•RT]	Tool number.
[•RD]	Tool correction number.
[•RN]	Number of the last sequence (block) encountered. If the block number is not specified, the last numbered block is analysed.
[•RED]	Angular offset.
[•REC]	Spindle orientation (milling).
[•RG4]	Dwell time programmed (G04 F..). This nonmodal function may however remain stored. Its value can therefore be read if the system is in state G999 or G998.
[•RG80]	Number of the calling function in a subroutine called by G function. In a subroutine called by G function, the number of the calling function is addressed by [•RG80] (in state G80, its value is zero).
[•RNC]	Value of NC (spline curve number).
[•RDX]	Tool axis orientation. Defined by the following signs and values: +1 for G16 P+            +2 for G16 Q+            +3 for G16 R+ -1 for G16 P-            -2 for G16 Q-            -3 for G16 R-
[•RXH]	Nesting level of the current subroutine. 1: main programme 2: first nesting level 3: second nesting level, etc. (8 possible nesting levels)

2.2.2.2 Addressing a List of Values

[•IRxx(i)] Addressing a list of values.

The symbol [•IRxx(i)] is used to address a list of values corresponding to the elements specified by xx.

Index (i) defines the rank of the element in the list.

[•IRX(i)] Values of the dimensions programmed on the axes.  
 Index i = 1 to 11.  
 i = 1: value of X  
 i = 2: value of Y  
 i = 3: value of Z  
 i = 4: value of U  
 i = 5: value of V  
 i = 6: value of W  
 i = 7: value of A  
 i = 8: value of B  
 i = 9: value of C  
 i = 10: in G21: Y axis address index 10  
           in G22: Z axis address index 10  
 i = 11: in G21: X axis address index 11  
           in G22: Y axis address index 11

[•IRTX(i)] Values of the offsets programmed on the axes.  
 Index i = 1 to 9, same as the values of the dimensions programmed on the axes (see [•IRX(i)]).

[•IRI(i)] Values of arguments I, J and K. Index i = 1 to 5.  
 i = 1: value of I  
 i = 2: value of J  
 i = 3: value of K  
 i = 4: in G21: J component address index 4  
           in G22: K component address index 4  
 i = 5: in G21: I component address index 5  
           in G22: J component address index 5

[•IRP(i)] Values of arguments P, Q and R. Index i = 1 to 3.  
 i = 1: value of P  
 i = 2: value of Q  
 i = 3: value of R

[•IRH(i)] Numbers of current programmes or subroutines or lower nesting levels.  
 Index i = 1 to n subroutine.  
 i = 1: addresses the main programme  
 i = 2: addresses the subroutine called by the main programme  
 i = 3: addresses the next subroutine, etc. (8 possible nesting levels)

[•]RDI(i) Values defining the origin of the programmed angular offsets (G59 I.. J.. K..). Index i = 1 to 3.  
i = 1: value of I  
i = 2: value of J  
i = 3: value of K

## 2.3 Symbols Accessing the Data of the Previous Block

The same data can be accessed in the previous block as in the current block (see 2.2). The same symbols are used, but are preceded by two decimal points (instead of one).

The symbols can be:

- symbols addressing Boolean values, or
- symbols addressing numerical values.

The symbols are used to read the modal data of the previous block.

These data are those of the last executable previous block (or possibly the last block executed).

This addressing is used only when execution of the current block is suspended by programming function G999.

### General Syntax

Variable = [**..**symbol(i)]

Variable	L programme variable, symbolic variable [symb], E parameter.
<b>..</b> symbol(i)	Symbol between square brackets, preceded by two decimal points, possibly followed by an index (i).



---

## 3 Storing Data in Variables L900 to L951

<b>3.1</b>	<b>General</b>	3 - 3
<b>3.2</b>	<b>Storing F, S, T, D, H and N in Variables L900 to L925</b>	3 - 3
<b>3.3</b>	<b>Storing EA to EZ in Variables L926 to L951</b>	3 - 4
<b>3.4</b>	<b>Symbolic Addressing of Variables L900 to L951</b>	3 - 4



The programming tools described in this chapter are necessary for creating subroutines called by G functions (see Chapter 5).

### 3.1 General

Certain arguments or functions may have different meanings in the machining cycles programmed. Status symbols [ $\bullet$ IBE0(i)] and [ $\bullet$ IBE1(i)] are used to detect their presence in blocks including a G function calling a subroutine.

It is up to the subroutine to correctly address these arguments or functions according to their meanings and to store their values in variables L900 to L951.

The bits of [ $\bullet$ IBE0(i)] and [ $\bullet$ IBE1(i)] (each equal to 0 or 1) are accessible for read by parametric programming.

### 3.2 Storing F, S, T, D, H and N in Variables L900 to L925

#### Storing F, S, T and D

Status symbol [ $\bullet$ IBE0(i)] consisting of a list of bits is used to detect programming of F, S, T and D in blocks including a function Gxxx.

Index (i): from 1 to 26 for addresses A to Z.

Addressing the status symbol:

- bit [ $\bullet$ IBE0(4)] is set if D is programmed,
- bit [ $\bullet$ IBE0(6)] is set if F is programmed,
- bit [ $\bullet$ IBE0(19)] is set if S is programmed,
- bit [ $\bullet$ IBE0(20)] is set if T is programmed.

The values of D, F, S and T are stored in the following variables (L900 to L925):

- D in L903,
- F in L905,
- S in L918,
- T in L919.

#### Storing H and N

H and N can only be stored when not preceded by functions G75, G76, G77 or G79 related to them for ISO programming.

A second N (following the first N and defining the last call sequence of a subroutine N.. to N..) is stored in variable L914 (this N can in no case be the block number programmed at the start of the sequence).

Addressing the status symbol:

- bit [ $\bullet$ IBE0(8)] is set if H is programmed,
- bit [ $\bullet$ IBE0(14)] is set if the first N is programmed,
- bit [ $\bullet$ IBE0(15)] is set if the second N is programmed.

The values of H and N are stored in the following variables:

- H in L907,
- first N in L913,
- second N in L914.

### 3.3 Storing EA to EZ in Variables L926 to L951

The values of EA to EZ to be stored in variables L926 to L951 are defined by two alphabetic characters:

- the first is the letter E,
- the second is a letter between A and Z.

Status symbol [ $\bullet$ IBE1(i)] consisting of a list of 26 bits is used to detect the presence of functions EA to EZ in blocks including a function Gxxx (i = alphabetic index of the second letter after E).

Addressing the status symbol:

- bit [ $\bullet$ IBE1(1)] addresses the bit corresponding to EA,
- bit [ $\bullet$ IBE1(2)] addresses the bit corresponding to EB, and so forth down to ...,
- bit [ $\bullet$ IBE1(26)] addresses the bit corresponding to EZ.

The values are stored in the following variables (L926 to L951):

- EA in L926,
- EB in L927, and so forth down to ...,
- EZ in L951.

### 3.4 Symbolic Addressing of Variables L900 to L951

Variables L900 to L925 and L926 to L951 in either the left-hand or right-hand side of an expression can be addressed by alphabetic symbols preceded by the character «'» (apostrophe).

#### Variables L900 to L925

L900 to L925 can be addressed by symbols 'A to 'Z.

Example:

'C = 'A + 'B is equivalent to  $L902 = L900 + L901$

#### Variables L926 to L951

L926 to L951 can be addressed by symbols 'EA to 'EZ respectively.

('EA = L926, 'EB = L927, etc. up to 'EZ = L951).

Example:

'A = 'B - 'EA / 'EZ is equivalent to  $L900 = L901 - L926 / L951$ .

---

# 4 Creating and Managing Symbolic Variable Tables

---

<b>4.1</b>	<b>Creating Symbolic Variable Tables</b>	4 - 3
4.1.1	Defining a Table	4 - 3
4.1.2	Table Dimensions	4 - 3
4.1.3	Initialising Variables and Tables	4 - 5
4.1.4	Creating Tables for Storing Profiles	4 - 6
4.1.5	Data That Can Be Stored in a Table	4 - 7
<b>4.2</b>	<b>Symbolic Variable Management Commands</b>	4 - 8
4.2.1	Storing a Profile	4 - 8
4.2.2	Storing a Profile Interpolated in the Plane	4 - 11
4.2.3	Offsetting an Open Profile and Updating the Table	4 - 13
4.2.4	Redefining a Profile According to the Tool Relief Angle	4 - 15
4.2.5	M Functions and/or Axes Enabled or Inhibited. Setting or Resetting Bits	4 - 18
4.2.6	Searching the Stack for Symbolic Variables	4 - 20
4.2.7	Providing a List of Symbolic Variables	4 - 21
4.2.8	Copying Blocks or Entries from One Table into Another Table	4 - 22
4.2.9	Indirect Addressing of Symbolic Variables	4 - 27
4.2.10	Programming Examples	4 - 28



The programming tools described in this chapter are used to create subroutines called by G functions (see Chapter 5).

## 4.1 Creating Symbolic Variable Tables

The rules for writing symbolic variables used when creating tables are the same as those defined for parametric programming (see Chapter 7 of the Programming Manual).

### 4.1.1 Defining a Table

A table is declared as a symbolic variable between the words VAR and ENDV.

A table is defined by including its dimensions between brackets after the last character of the symbolic variable.

For tables with several dimensions, the dimensions are separated by commas «,» or semicolons «;».

#### Syntax

```

VAR
[TABLn(a,b,c ...)]
ENDV
```

VAR Declaration of symbolic variables.

[**TABL**n(a,b,c...)] **TABL**n: table name in the stack.  
(a,b,c ...) : table dimensions.

ENDV End of symbolic variable declaration.

#### Example of table definitions

```

VAR
[TABL1(10)] [TABL2(2,5,3)]
ENDV
```

For table 2 [**TABL**2] above, the number of entries reserved equals:  
 $2 \times 5 \times 3 = 30$  entries, i.e. 5 groups of 2 then 3 groups of 10.

### 4.1.2 Table Dimensions

Tables can have from one to four dimensions.

For tables with one dimension:

- the value of a dimension must be between 1 and 65535.

For tables with 2, 3 or 4 dimensions:

- the value of a dimension must be between 1 and 255.

The dimensions must be declared as immediate values or declared symbolic variables.

Example:

```
[VAR1] = 10
```

```
[VAR] = [TAB1 (VAR1,5)] is equivalent to: [TAB1 (10,5)]
```

Symbolic variables are real values.

Table indexes are immediate values or symbolic variables.

A dimension cannot be defined using:

- L programme variables,
- E external parameters.

Example:

If symbolic variable [TAB(L0,3)] is programmed, it is not programme variable L0 but symbolic variable [L0] that is searched for.

The table indexes can be additions or subtractions of values or symbolic variables.

Example:

```
VAR [IX] [COSX] [SINX] [NBT] = 4
[TABL(2,NBT)] = 0, 0, 10, 5, 20, 8, 30, -2
ENDV
FOR [IX] = 1 TO [NBT] -1 DO
  [COSX] = [TABL(1,IX+1)] - [TABL(1,IX)]
  [SINX] = [TABL(2,IX+1)] - [TABL(2,IX)]
  L0 = [COSX] * [COSX] L0 = [SINX] * [SINX] + L0
  [COSX] = [COSX] / RL0 [SINX] = [SINX] / RL0
  X [TABL(1,IX+1)] Y [TABL(2,IX+1)]
ENDF
```

#### Structure of tables with several dimensions

The entries for the first dimension are located first in the declaration, then multiplied by the number of entries of the second dimension. The result is then multiplied by the number of entries of the following dimension and so forth.

### 4.1.3 Initialising Variables and Tables

The values are initialised with a default value of zero.

Initialising with other values is made by declaring the character = followed by the initial value(s) separated by commas «,».

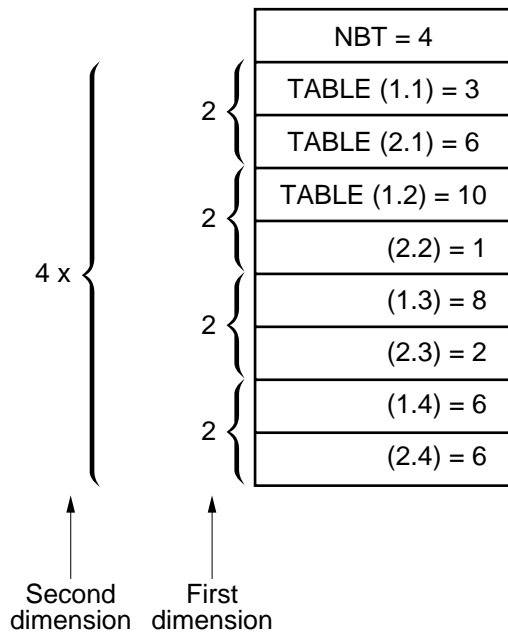
The initial values can be declared in several blocks. In this case, the character = is repeated before the value(s) defined in the next block.

Example:

```
VAR [NTB] = 4 [TABLE(2,NTB)] = 3,6
                                     = 10,1,8,2,6,6
ENDV
```

```
If L0= [TABLE(2,3)]
    L0= 2 (ie. the loth digit)
```

#### Storage in the memory



#### 4.1.4 Creating Tables for Storing Profiles

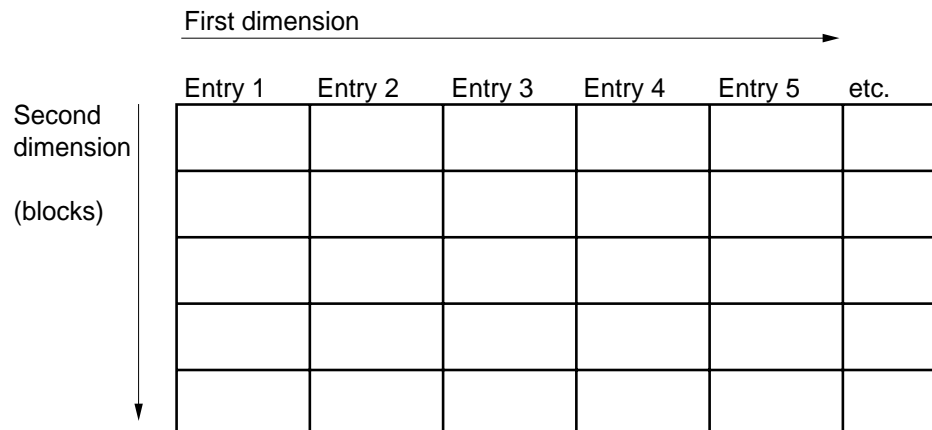
The system offers the possibility of storing a profile written in ISO or PGP in a table of the programme stack. The table is created as the profile blocks are read.

The programmed blocks are stored in a table with two dimensions:

- the first dimension includes all the functions to be saved in a block,
- the second dimension corresponds to the number of blocks in the profile.

The data in the table are then accessed by parametric programming.

Example:



### 4.1.5 Data That Can Be Stored in a Table

The following ISO programming data can be stored in tables.

#### G Functions

Only one G function per block can be stored.

After a change of interpolation plane in a block, the new function is stored (G17, G18 or G19 for milling, G20, G21 or G22 for turning). Otherwise, one of modal functions G00, G01, G02 or G03 is stored.

#### Values of the Programmed Axes

X, Y, Z, U, V, W, A, B, C (depending on the axes declared in machine parameter P0).

The modal values programmed with the axes are stored.

#### Values of I, J, K, P, Q, R

The values of the functions are stored only if the functions are present in the block. Otherwise, the value zero is stored in the corresponding entries.

#### Feed Rate F

The modal value related to function F is stored.

#### Spindle Speed S

The value of S is not stored in the table unless it is present in the block. Otherwise, a value of zero is stored.

#### Tool Call T

Function T is not stored in the table unless it is present in the block. Otherwise, a value of zero is stored.

## 4.2 Symbolic Variable Management Commands

### 4.2.1 Storing a Profile

**BUILD** Creates a table for storing profile paths.

The BUILD function is used to store the profile in a two-dimensional table:

- the first dimension is limited to 16 entries,
- the second dimension is limited to 255 entries.

#### Syntax

**BUILD** [TAB(G / X / Y / I / J,NB)] H.. N..+n N..+n

BUILD	Creation of a table for storing a profile.
TAB	Table name in the stack.
G / X / Y / I / J	Data types whose values are stored in the entries of the first table dimension (16 entries maximum).
NB	Name of the variable containing the number of blocks of the second table dimension (maximum 255 blocks).
H..	Definition of the limits of the profile.
N.. N..	
H.. N.. N..	
N..+n N..+n	
H N..+n N..+n	

#### Note

The BUILD function must be the first word in the block and the table name TAB must be the second. They must be separated by at least one space.

Programming the table name TAB and variable NB automatically creates a two-dimensional table.

Example:

```

%55                    %55
N10 ...                G.. ...   First block of the profile
N..                    G.. ...
N..                    G.. ...
BUILD [TAB(G/X/Y/I/J,NB)] H55   G.. ...   Last block of the profile
N..
    
```

Two-dimensional table created by the above programme:

- first dimension: 5 entries,
- second dimension: 4 entries (blocks).

NB	<div style="display: inline-block; text-align: center;">(blocks)</div> 4																				
TAB	<table border="1" style="width: 100%; height: 100%; border-collapse: collapse; text-align: center;"> <tr><td>..</td><td>..</td><td>...</td><td>..</td><td>..</td></tr> <tr><td>..</td><td>..</td><td>...</td><td>..</td><td>..</td></tr> <tr><td>..</td><td>..</td><td>...</td><td>..</td><td>..</td></tr> <tr><td>..</td><td>..</td><td>...</td><td>..</td><td>..</td></tr> </table>	..	..	...	..	..	..	..	...	..	..	..	..	...	..	..	..	..	...	..	..
..	..	...	..	..																	
..	..	...	..	..																	
..	..	...	..	..																	
..	..	...	..	..																	

### Allocation of Additional Entries in a Field of the BUILD Function

Additional entries can be allocated in the first table dimension if they are not initialised by data in the blocks. In this case, the entries are declared by «0s» separated by the character /.

Example:

```

%55
N10 ...
N..
N110                                First block of the profile
N..
N..
N220                                Last block of the profile
N..
BUILD [PROF1(G/X/Y/Z/O/O,NB)] N110 N220   The first table dimension includes
                                           6 entries
    
```

### Declaring Entries as a List of Bits in a Field of the BUILD Function

In a symbolic variable, some of the following axes and arguments can be declared as a list of bits:

- axes X, Y, Z, etc.,
- arguments I, J, K,
- arguments P, Q, R.

The list of bits is declared by programming one of addresses X, I or P followed by a decimal point and the symbolic variable name in a field of BUILD.

Example:

```
BUILD [TAB1(G/I.Symb/R,NB)] H..          Declaration of I.Symb
```

Symbolic variable [Symb] contains a sum of values. This sum is defined from the indexes of the addressing symbols [**●●**IBX(i)], [**●●**IBI(i)] and [**●●**IBP(i)], i.e.:

- 1 for index i = 1
- 2 for index i = 2
- 4 for index i = 3
- 8 for index i = 4, etc.

Therefore, the value of the variable including I, J and K of [**●●**IBI(i)] is equal to  $2^{I-1} + 2^{J-1} + 2^{K-1}$ .

Example:

```
VAR [LIST] = 6
ENDV
```

```
BUILD [PROF(G/X.LIST/R,NB)] N.. N..      is equivalent to the following block:
BUILD [PROF(G/Y/Z/R,NB)] N.. N..
```

## 4.2.2 Storing a Profile Interpolated in the Plane

P.BUILD Creates a table for storing the dimensions of the profile interpolation plane.

The P.BUILD function is used to store the profile in a two-dimensional table:

- the first dimension is limited to 7 entries,
- the second dimension is limited to 255 entries.

### Syntax

**P.BUILD** [TAB(7,NB)] H.. N..+n N..+n

4

BUILD	Creation of a table for storing a profile.
TAB	Table name in the stack.
7	Number of entries in the first dimension (maximum 7).
NB	Name of the variable containing the number of blocks (maximum 255 blocks).
H..	Definition of the limits of the profile.
N.. N..	
H.. N.. N..	
N..+n N..+n	
H.. N..+n N..+n	

### Note

The P.BUILD function must be the first word in the block and the table name TAB must be the second. They must be separated by at least one space.

Defining the Seven Entries of the First Dimension with the P.BUILD Function

- entry 1: Type of interpolation:  
value = 0 for linear interpolation,  
value = -1 for clockwise circular interpolation,  
value = +1 for anticlockwise circular interpolation.
- entry 2: End point, value of the dimension on the X axis.
- entry 3: End point, value of the dimension on the Y axis.
- entry 4: Position of the centre:  
value on the X axis for circular interpolation, else value = 0.
- entry 5: Position of the centre:  
value on the Y axis for circular interpolation, else value = 0.
- entry 6: Start point, value of the dimension on the X axis.
- entry 7: Start point, value of the dimension on the Y axis.

**Example:**

```

%70                                     %80
N10 G17 X10 Y10                         N10 ...
P.BUILD [TAB(7,NB) H80 N110 N130        N..
N..                                       N..
                                         N110 G01 X20
                                         N120 G03 X20 Y50 I20 J30
                                         N130 G01 X10 Y20
                                         N.. ...
    
```

Table TAB and variable NB create the following table with 7 entries:

(blocks)  
NB 

3
---

TAB	0	20	10	0	0	10	10
	+1	20	50	20	30	20	10
	0	10	20	0	0	20	50

### 4.2.3 Offsetting an Open Profile and Updating the Table

R.OFF Normal offset of an open profile.

The R.OFF function is used for normal offset of a profile initially created in a table by the P.BUILD function or a table with the same format, i.e. [Pa(7,Nb)].

After execution of the R.OFF function, the offset profile is contained in the same table and the variable specifying the number of blocks is updated since intermediate blocks may have been created (see Fig. 1).

#### Syntax

R.OFF [Pa(7,Nb)] / ±1 / R

R.OFF	Normal offset of a profile.
Pa	Table name.
7	Number of table entries.
Nb	Name of the variable containing the number of blocks in table Pa.
±1	Value = +1: right offset of the profile, value = -1: left offset of the profile.
R	Radius expressed in the same units as the dimensions.

#### Note

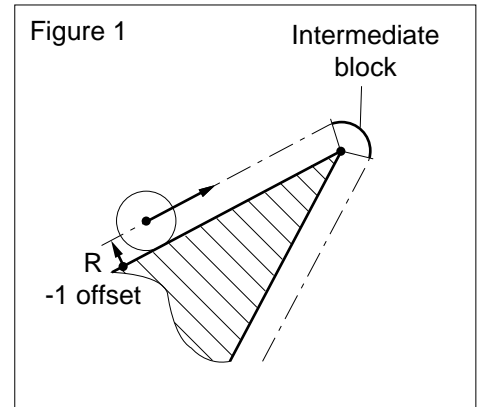
The R.OFF function must be the first word in a block.

The profile executed with the R.OFF function must be an open profile, i.e. the start point of the profile must be different from the end point (see Fig. 2).

The R.OFF function can only operate on profiles contained in P.BUILD that do not include alternating left and right offsets during execution.

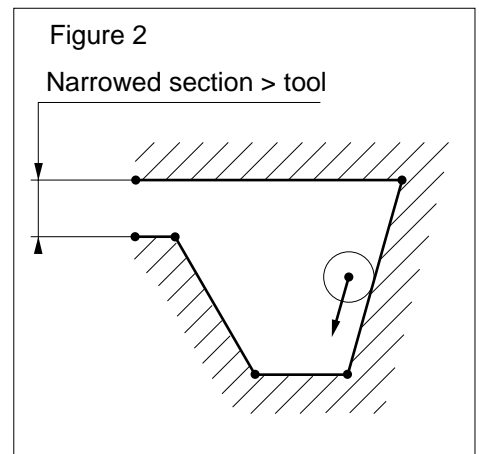
### Creation of an Intermediate Block by the System

When the profile includes particular paths, the system may create a connection block.



### Open contour

When the profile includes a narrowed section, it must be sufficiently large to allow passage of the tool. Otherwise, the system considers the profile to be closed.



### Example

```
%92
N..
P.BUILD [P(7,NB)] N110 N200
...
...
R.OFF [PA(7,NB)] /-1 /10
```

Profile offset by 10 to the left

#### 4.2.4 Redefining a Profile According to the Tool Relief Angle

CUT	Elimination of the grooves or parts of groove located inside the tool relief angle.
-----	---

The CUT function applies to grooves located in the path of a plane profile created in a table by the P.BUILD function or a table with the same format, i.e. [Pa(7,Nb)].

After execution of the CUT function, the new profile is contained in the same table and the variable specifying the number of blocks is updated.

##### Syntax

<b>CUT * [Pa(7,Nb)] / Angle</b>
---------------------------------

CUT	Eliminates the grooves or parts of grooves located within the tool relief angle.
*	When the character * precedes the table name, all the grooves located within the tool relief angle are processed. When the character * is missing in front the table name, only the first groove located within the tool relief angle is processed.
Pa	Table name.
7	Number of table entries.
Nb	Name of the variable containing the number of blocks in table Pa.
Angle	Angle Relief angle in degrees.

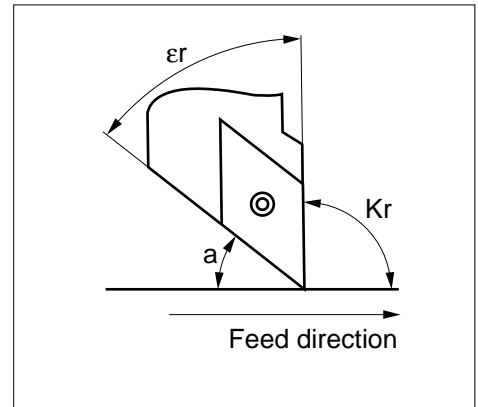
##### Note

The CUT function must be the first word in the block (no sequence number).

### Review of the Angles of a Cutting Tool Defined in Plane ZX

Characteristic angles:

- $K_r$ : approach angle,
- $\epsilon_r$ : tool nose angle,
- $a$ : clearance angle.



### Processing of the table

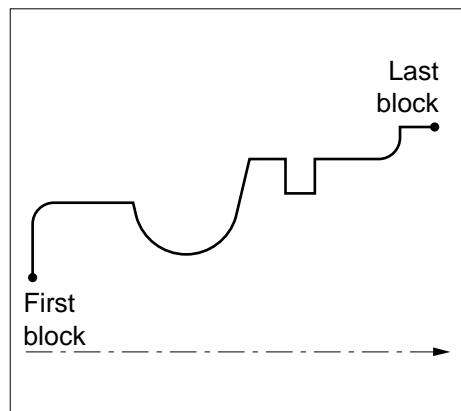
The table analysis begins on the first block and ends:

- on the last block with CUT \* ...,
- on the first cut with CUT ...

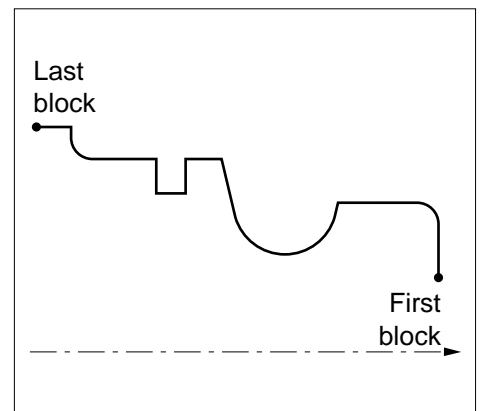
In the table, the profile must always be defined so that the profile start dimension on the X axis (first block) is less than the profile end X dimension (last block).

Example:

Profile correctly defined



Profile incorrectly defined



When the clearance angle is negative or zero (between 0 degrees and -180 degrees), the profile areas located below this angle are eliminated.

When the relief angle is positive (between 0 degrees and +180 degrees), the profile areas above the angle are eliminated.

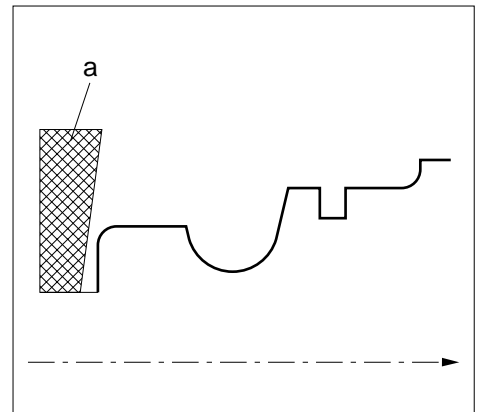
### Examples:

«a» shows the areas that are eliminated.

#### Example 1:

Elimination of the first groove in the profile (no \* in front of the variable).

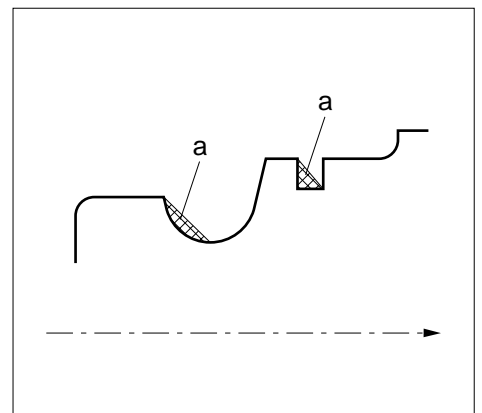
CUT [PA(7,NB)] / -95



#### Example 2:

Elimination of the grooves or parts of grooves located on the profile (\* in front of the variable).

CUT \* [PA(7,NB)] / -50



#### 4.2.5 M Functions and/or Axes Enabled or Inhibited. Setting or Resetting Bits

**BSET** Programming of M functions and/or one or more axes enabled.  
Setting of the bits of [**•**IBE0(i)] and [**•**IBE1(i)].

**BCLR** Programming of M functions and/or one or more axes inhibited.  
Resetting of the bits of [**•**IBE0(i)] and [**•**IBE1(i)].

##### Syntax

**BSET** [**•**BMxx] / [**•**IBX(i)] / [**•**IBE0(i)] / [**•**IBE1(i)]

**BSET** Enabling the programming of M functions and/or one or more axes.

[**•**BMxx] / [**•**IBX(i)] When the system is in state G999, enabling by BSET sets the bits of [**•**BMxx] and/or [**•**IBX(i)].

[**•**IBE0(i)] / [**•**IBE1(i)] When a subroutine is called by function Gxx, BSET also sets the bits of [**•**IBE0(i)] and [**•**IBE1(i)].

##### Syntax

**BCLR** [**•**BMxx] / [**•**IBX(i)] / [**•**IBE0(i)] / [**•**IBE1(i)]

**BCLR** Inhibiting the programming of M functions and/or one or more axes.

[**•**BMxx] / [**•**IBX(i)] When the system is in state G999, inhibiting by BCLR resets the bits of [**•**BMxx] and/or [**•**IBX(i)].

[**•**IBE0(i)] / [**•**IBE1(i)] When a subroutine is called by function Gxx, BCLR also resets the bits of [**•**IBE0(i)] and [**•**IBE1(i)].

### Note

Functions BSET and BCLR:

- must be the first words in the block (no sequence number),
- must be separated from the list of symbols by at least one space. However, there must be no spaces in the list of symbols,
- are followed by the list of symbols to be enabled or inhibited. The symbols are separated by «/».

### Example

In block N120, only movements X10 and Z10 are executed. The movements on the Y and B axes and the "post M" function M05 are inhibited (see Chapter 2 for indexes (2) and (8) corresponding to Y and B respectively).

N..

N..

N.. G999 X10 Y10 Z10 B30 M05

BCLR [.IBX(2)] / [.IBX(8)] / [.BM05]            Disabled

N120 G997

## 4.2.6 Searching the Stack for Symbolic Variables

SEARCH Searches the stack for a symbolic variable.

### Syntax

**SEARCH [Symb] N..**

SEARCH	Searches the stack for a symbolic variable.
[Symb]	Name of the symbolic variable. When the variable searched for is found, analysis of the block is continued.
N..	Block number to which a branch is made when the symbolic variable is not found.

### Example

```
%30
N..
VAR [Symb]
ENDV
N..

%35
N..
N90 ...
SEARCH [Symb] N100
N..
N100
N..
```

## 4.2.7 Providing a List of Symbolic Variables

SAVE	Provides the main programme and subroutines with a list of the symbolic variables declared in any subroutine.
------	---

### Syntax

<b>SAVE</b> [Symb1] / [Symb2] ...
-----------------------------------

SAVE	Provides the main programme and subroutines with a list of the symbolic variables declared in any subroutine.
[Symb1] / [Symb2] ...	List of symbolic variables.

### Notes

The symbolic variables provided may be declared within subroutines at any nesting levels.

The variables declared after SAVE must be separated by the character «/».

### Example

After a return from subroutine %30, programme %10 and subroutine %20 as well as any new subroutines can use symbolic variables [V1] and [TB(4)].

```
%10
N..
N.. G77 H20
N..

%20
N..
N.. G77 H30
N..

%30
N..
VAR [V1] / [TB(4)]
ENDV
N..
SAVE [V1] / [TB(4)]
N..
```

## 4.2.8 Copying Blocks or Entries from One Table into Another Table

**MOVE** Copies all or part of a table into another table.

The MOVE function is used to copy tables with the following formats:

- [P(m)] : m blocks of an entry,
- [P(n,m)] : m blocks of n entries.

### General Syntax

**MOVE** [Pj(nj,mj)],mj1,mj2 = [Pi(ni,mi)],mi1,mi2 / j1=i1 / j2=i2 /jn=in

The MOVE function provides several possibilities for copying:

- simple copying of blocks,
- partial copying of blocks,
- specification of the entries to be copied.

### Syntax for Simple Copying of Blocks

**MOVE** [Pj(nj,mj)] = [Pi(ni,mi)]

MOVE	Copies the contents of one table into another table. During a simple copy, the two tables must have the same format, i.e.: nj = ni and mj = mi.
Pj	Target table name.
nj,mj	Entries and blocks of the target table.
Pi	Source table name.
ni,mi	Entries and blocks of the source table.

### Syntax for Partially Copying Blocks

**MOVE** [Pj(nj,mj)],mj1,mj2 = [Pi(ni,mi)],mi1,mi2

MOVE	Copies the contents of one table into another table.
Pj	Target table name.
nj,mj	Entries and blocks of the target table.
mj1,mj2	Limits of target table Pj between which are copied the blocks indexed mi1 to mi2 of the source table Pi. The other blocks of Pj are not modified.
Pi	Source table name.
ni,mi	Entries and blocks of the source table.
mi1,mi2	Indexed limits of source table Pi. These limits and the blocks between these two limits are copied into table Pj between limits mj1 and mj2.

### Syntax for Specifying Entries to Be Copied

**MOVE** [Pj(nj,mj)] = [Pi(ni,mi)] / j1=i1 / j2=i2 / jn=in

MOVE	Copies the contents of one table into another table.
Pj	Target table name.
nj,mj	Entries and blocks of the target table.
Pi	Source table name.
ni,mi	Entries and blocks of the source table.
/ j1=i1 / j2=i2 / jn=in	When certain entries of a table are not to be copied, each entry to be copied must be specified with, after the character «/», the target entry index followed by the character «=» and the source entry index. The value of an entry copied in the target table can be inverted by preceding the source entry index by the sign «-» (minus).

### Notes

The MOVE function must be the first word in the block (no sequence number).

The maximum number of blocks in a finished profile is limited to 95.

It is possible to reverse the order of a copy by reversing the start and end limit indexes in one of the tables.

The indexes can be specified in symbolic variables.

In case of a programming error, the system returns the following error numbers:  
 ERROR 196 (inconsistency in index declaration),  
 ERROR 199 (syntax error).

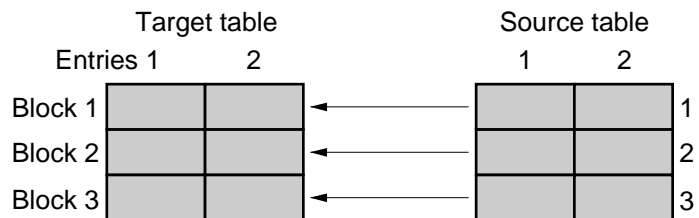
All tables of one or more dimensions can be copied by the MOVE function.

### Examples

#### MOVE simple copy of blocks

Example: Copying the contents of one table into another table.

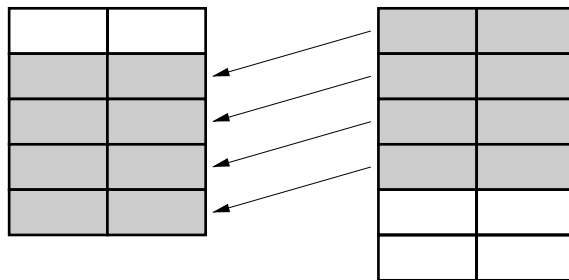
MOVE [PB(2,3)] = [PA(2,3)]



MOVE partial copy of blocks

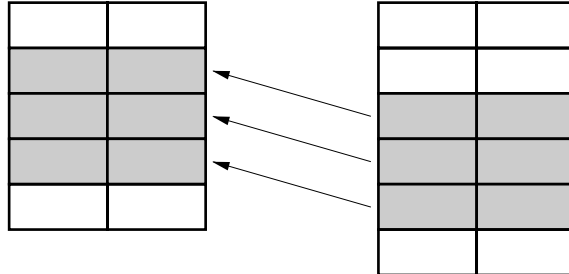
Example 1: Copying part of a table into another table.

MOVE [PB(2,5)],2,5 = [PA(2,6)],1,4



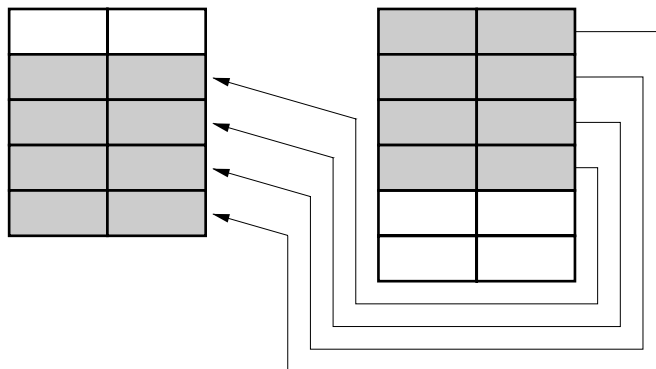
Example 2: Copying part of a table into another table.

MOVE [PB(2,5)],2,4 = [PA(2,6)],3,5



Example 3: Reversal of the limit and block indexes when copying part of a table into another table.

MOVE [PB(2,5)],2,5 = [PA(2,6)],4,1

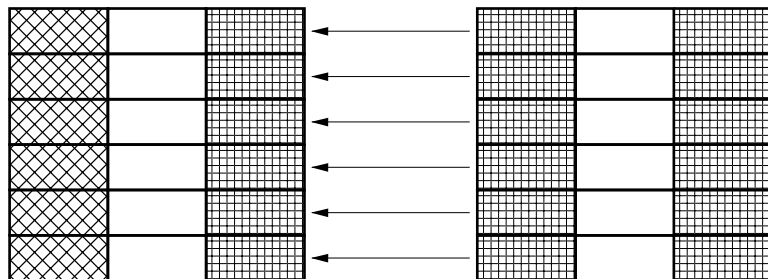


MOVE: Partial copying of blocks and specification of the entries to be copied

Example: Inversion of the values of the entries when copying part of a table into another table.

Only the first and third entries are copied into the target table PB, and the values of the first entries are inverted.

MOVE [PB(3,6)] = [PA(3,6)] / 1=-1 / 3=3



**Reminder**

DELETE Function

The DELETE (or DELE) function can be used to programme deletion of the symbolic variables (see Chapter 7 of the Programming Manual).

### 4.2.9 Indirect Addressing of Symbolic Variables

A variable or a table of symbolic variables can be referenced by a value.

This addressing mode simplifies linking tables by using numbers instead of the names of the symbolic variables.

This symbolic variable addressing mode is indirect, since it is via another address vector variable whose value is the reference of another symbolic variable or table of symbolic variables.

Numerical addressing is symbolised by the character @ followed by the address vector name.

The variables addressed by negative numbers are automatically deleted by function G80.

#### Example

```
VAR [no] = 7  [@no(10)]
    [Symb]
ENDV
[Symb] = 7   L0 = [@Symb(2)]
```

The table with 10 entries is referenced by the value 7

«@Symb» addresses the same table declared as «@no»

## 4.2.10 Programming Examples

### Example 1

Use of BUILD for milling (XY plane). With radius correction, path from 1 to 6 then back from 6 to 1.

Representation of machining

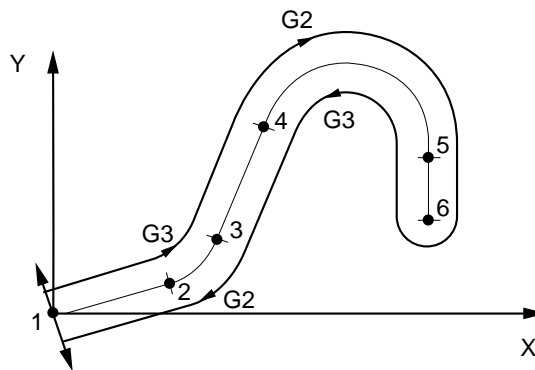


Table built by the programme

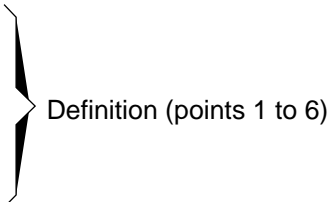
Points		G	X	Y	I	J		
	1	1	0	0	0	0		
	2	2	22.268	8.104	0	0		
	3	3	28.243	14.081	18.846	17.501	Values stored in the table when BUILD is executed	
[NB] -1	4	1	35.905	35.13	0	0		
	5	2	65	30	50	30		
	6	1	65	20	0	0		

$G[TAB(1,I)]$        $X[TAB(2,I)]$

```

%350
Z0
M999 G79 N100
N10 G1 X Y
EA20 ES EB10
EA70
G2 I50 J30 R15
N40 G1 Y20
N100
BUILD [TAB(G/X/Y/I/J,NB)]N10 N40
VAR [I]
ENDV
FOR [I]=1 TO [NB] D0 G41 D1
    G[TAB(1,I)] X[TAB(2,I)] Y[TAB(3,I)] I[TAB(4,I)] J[TAB(5,I)]
ENDF
FOR [I]=[NB] -1 DOWNT0 1 D0
    L0=[TAB(1,I+1)]
    IF L0>1 THEN L0=L0*3+1&3
    GLO X[TAB(2,I)] Y[TAB(3,I)] I[TAB(4,I+1)] J[TAB(5,I+1)]
ENDF
G40 X Y
M2

```



Path from 1 to 6

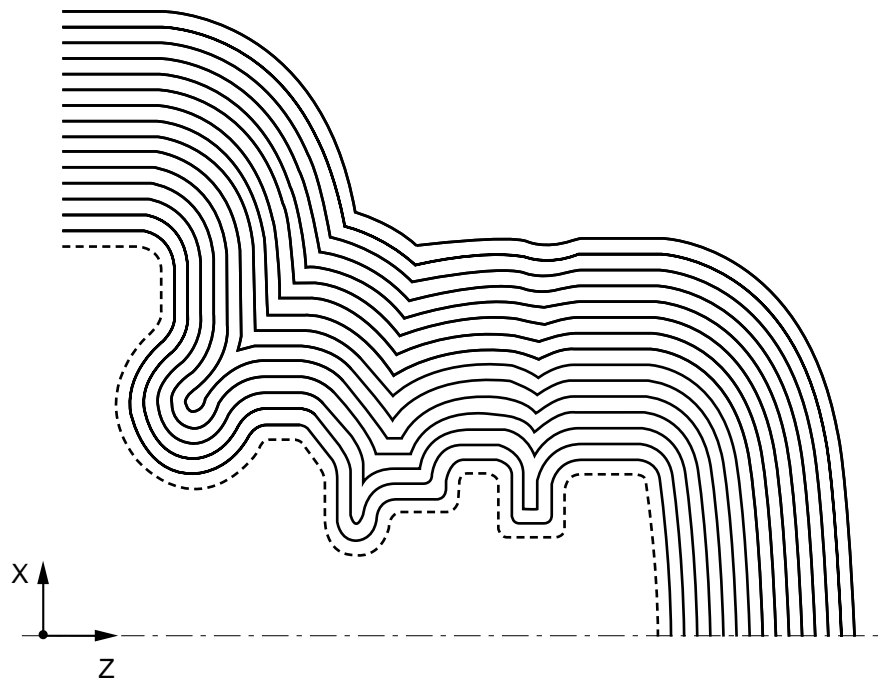
Path back from 6 to 1

Reversal of G2 and G3 for the return

**Example 2**

Use of P.BUILD for turning. Execution of a profile with offset and use of the MOVE and R.OFF functions

Representation of machining



```

%46
P.BUILD [C(7,N)] N100 N110
VAR [R][G(3)]=2,1,3 [I] [V]
ENDV

...
FOR [R]= 30 DOWNT0 2 BY 2 DO
  VAR [M]=[N]-1 [P(7,M)]
  ENDV
  MOVE [P(7,M)] = [C(7,N)],2,[N]
  R.OFF [P((7,M)] / + 1 / [R]
  G X60 Z110
  X[P(7,1)] Z[P(6,1)]
  FOR [I] = 1 TO [M] DO [V] = [P(1,I)]+2
  G[G(V)] X[P(3,I)] Z[P(2,I)] I[P(5,I)] K[P(4,I)]
  ENDF
  DELE [P]/[M]
ENDF
M02
N100 X0 Z100
G3 I0 K20 ES-
G1 EA180 X20 Z85 EB2
X12
Z75
X20
Z70
X15
Z60
G2 X15 Z50 I15 K55
G1 X25 EB-4
Z40
G2 I30 K30 ES+
G1 EA90 X50 Z25 EB3
N110 Z10

```

Creation of a table to store the profile

Copy the table into another table  
Offset the profile to the right

Profile definition



---

# 5 Creating Subroutines Called by G Functions

---

<b>5.1</b>	<b>Calling Subroutines by G Functions</b>	5 - 3
<b>5.2</b>	<b>Inhibiting Display of Subroutines Being Executed</b>	5 - 5
<b>5.3</b>	<b>Programming Examples</b>	5 - 6

---



## 5.1 Calling Subroutines by G Functions

Gxxx Subroutine call by a G function.

Function Gxxx is used to call and execute a subroutine.

Function Gxxx is used for execution of machining cycles. The cycles created can be customised. This functionality is also used to create special functions.

### Syntax

N.. **Gxxx** Parameters specific to each machining cycle.

Gxxx	This function forces a call to subroutine %10xxx whose number corresponds to the machining cycle. Example: - G81 calls subroutine %10081, - G199 calls subroutine %10199.
Parameters	The parameters (or arguments) specific to the machining cycle must be specified after Gxxx.

### Properties of the Functions

Functions Gxxx calling subroutines are modal.

A function Gxxx is nonmodal when the cancellation function G80 is included in the subroutine called by the cycle.

### Cancellation

Modal functions Gxxx are cancelled by function G80 (this function does not call a subroutine).

### Notes



Since functions G200 to G255 may be used for NUM applications, it is recommended to use only functions G100 to G199.

List of G functions forcing a call to a subroutine:

- G06, G31, G33, G38, G45, G46, G48, G49, G63, G64, G65, G66, G81-G89,
- G100 to G255 (reminder: G200 to G255 reserved for NUM).

A subroutine call by a Gxxx function without arguments is ignored by the system. The arguments are interpreted by the %10xxx subroutine called.

The subroutines called by G functions must therefore have visibility into the programme context and all the functions programmed in the calling block.

Execution of a subroutine called by a G function cannot be interrupted by an «immediate» request in the edit mode (EDIT).

A subroutine called by a G function cannot itself call another subroutine by a G function. However, nesting with another type of call is possible (call by M function or machine processor), but two calls of the same type can in no case be nested.

#### **Functionalities Used in %10xxx Subroutines**

- Functions G997, G998 and G999,
- Programme variables L900 to L925 and L926 to L951,
- External parameters E,
- Symbolic variables,
- Programme status access symbols.

A call to a subroutine by a G function implicitly sets function G999 (execution suspended and block concatenation forced). This function has to be cancelled by programming functions G998 and/or G997 set in the subroutine.

During the return to the part programme, state G999 is systematically reset as long as the subroutine calling function (Gxxx) remains present and active (no G80).

For additional information on functions G997, G998 and G999, refer to the programming manuals for:

- Milling (938819),
- Turning (938820).

## 5.2 Inhibiting Display of Subroutines Being Executed

Display on the programme page (PROG) of a subroutine and its internal subroutines during execution can be inhibited.

The character «:» after the subroutine number inhibits display. In this case, only the subroutine call block is displayed.

Example:

Only block N150 including function G108 is displayed during execution of subroutines %10108 and %118.

%10	%10108:	%118
N10	N10	N10
N..	N..	N..
N150 G108 ...	N80 G77 H118	N..
N..	N..	N..

## 5.3 Programming Examples

### Example 1

Creating a particular cycle with function G199 (subroutine %10199).

The cycle below is given only for guidance.

It allows execution of several drilling or punching operations «P» distributed on a circle with radius «R» centred on XY (G17).

Cycle syntax and parameters

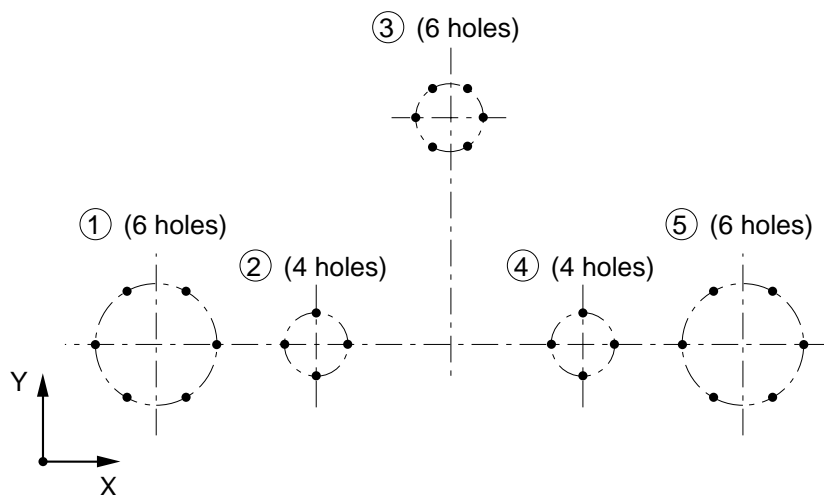
N.. G199 X.. Y.. ER.. Z.. P.. R.. F..

G199	Cycle for drilling equally spaced holes on a circle.
X.. Y..	Circle centre position.
ER..	Approach and clearance position.
Z..	Machining end point.
P..	Number of equally spaced holes.
R..	Radius of the hole circle.
F..	Feed rate.

Main machining programme

%20	
N10 G0 G52 Z0	
N20 T1 D1 M06 (DRILL)	
N30 S1000 M40 M03	
N40 X0 Y0 Z5	
N50 G199 X50 Y50 ER2 Z-10 P6 R20 F90	Circle 1 cycle
N60 X100 Z-5 P4 R10	Circle 2 cycle
N70 X150 Y150 Z-15 P6 R25	Circle 3 cycle
N80 X200 Y50 Z-5 P4 R10	Circle 4 cycle
N90 X250 Z-10 P6 R20	Circle 5 cycle
N100 G80 G0 G52 M05	Cycle cancelled
N110 M02	

Representation of machining



Cycle subroutine

```

%10199: (Equally spaced holes on the circle)
VAR
[G0/1] [RETURN] [FEED] [G94/5]
ENDV

[G0/1]=3 * [..BG03] [G0/1]=2*[..BG02] + [G0/1] Store G0, G1, G2 or G3
[G0/1]=1 * [..BG01] + [G0/1]
[FEED]=[..RF] Store G94 or G95
[G94/5]= 94 * [..BG94]
[G94/5]= 95 * [..BG95] + [G94/5]
PUSH L0 - L7

(Test whether P and R are programmed in the call block)
"CHANGE BG" IF [..BG80]= 1 THEN First block in the cycle?
L0= [.IBP(1)] * [.IBP(3)]
G79 L0= 0 N100 Error if P or R is missing
ENDI
IF [.IBP(1)] = 1 THEN Read next P if any
L100= [.IRP(1)]
ENDI
L100= [.IRP(1)] Store P
G79 L100 < 1 N101 Error if P is not a positive integer
IF [.IBX(3)] = 1 THEN L925 = [.IRX(3)] Hole bottom dimension
ENDI
    
```

[RETURN]= 'ER	Return dimension
BCLR [.IBX(3)]	Z axis disabled
L0= [.IRX(1)]	X dimension
L1= [.IRX(2)]	Y dimension
L2= L0 + [.IRP(3)]	Cycle start dimension
XL2	Hole bottom dimension modified
G997	XY movements enabled
FOR L4= 1 TO L100	
L5= L4 * 360 / L100	Current angle
L6= CL5 * [.IRP(3)] + L0	X dimension
L7= SL5 * [.IRP(3)] + L1	Y dimension
G3 G94 F5000 XL6 YL7 IL0 JL1	Circular positioning
M997	Forced concatenation
IF [.IBEO(6)]= 1 THEN	Feed rate
G94 FL905	
ENDI	
G1 Z L925	Hole drilled
G4 F1	Dwell in bottom of hole
G0 Z [RETURN]	Return in Z
M999	
ENDF	End of cycle
G [G94/5] F [FEED]	Return to initial conditions
PULL L0 - L7	
G79 N9999	End of cycle
N100 E.500	Error number (see %20500)
N101 E.501	Error number (see %20500)
N9999	

#### Error message programme

%20500 (Error messages of cycle G199)  
 N500 \$ P AND R MANDATORY IN G199  
 N501 \$ P MUST BE A POSITIVE INTEGER IN G199

### Example 2

Creating a special cycle with function G177 (subroutine %10177).

The cycle below is given only for guidance.

It is used to execute a profile by back and forth passes with the possibility of radius correction if required.

#### Cycle syntax and parameters

G177 N.. N.. ER..

G177	Machining by back and forth passes.
N.. N..	Numbers of the first and last blocks defining the profile (when the blocks are in reverse order, machining of the profile is reversed).
ER..	Argument forcing or cancelling radius correction: ER 40 : machining to tool centre ER 41 : offset on the left of the profile ER 42 : offset on the right of the profile

5

#### Main machining programme

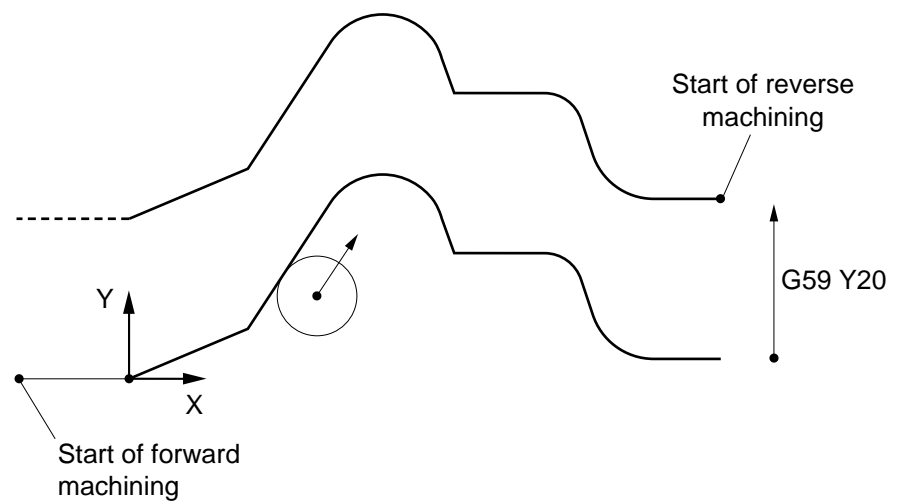
```

%77
N10 G0 G52 Z0
N20 T1 D1 M06 (TOOL R5)
N30 S2000 M40 M03
N40 G0 X-20 Y0
N50 G92 R1
N60 Z-10
N70 G79 N160
N80 G1 X-20 Y0
N90 EA20 ES
N100 EA50
N110 G2 I60 J25 X75 Y25
N120 G1 ET
N130 G2 I95 J10
N140 G3 I120 J15 X120 Y5
N150 G1 G40 X140
N160 G177 N80 N150 ER42
N170 G59 Y20
    
```

```
N180 G177 N150 N80 ER41
N190 G0 G52 Z0
N200 G0 X-100 M5
N210 M02
```

Return cycle

### Representation of machining



### Cycle subroutine

```
%10177: (Profile machined by back and forth passes)
```

```
G998
```

```
VAR [N1] [N2] [PLANF] [G] [H] [diam] [NBLOC] [M998] [multi]=1
  [axis1] [axis2] [PLANT] [centre1] [centre2]
```

```
ENDV
```

```
IF 'ER<40 OR 'ER>42 THEN 'ER=40
```

If ER is not correctly programmed,  
error ER 40

```
ENDI
```

```
[M998]=[.BM999]-[.BM997]+998 M998
```

Storage of M997, M998 and M999

```
[N1]=L913 [N2]=L914
```

Storage of the profile execution direction

```
IF L913>L914 THEN [N1]=L914 [N2]=L913
```

```
ENDI
```

Nesting level

```
[H]=[.RXH]-1 [H]=[.IRH(H)]
```

```
IF L913>L914 THEN
```

```
P.BUILD [TAB(7,NB)] H[H] N[N1] N[N2]
```

Creation of a table to store  
the profile

```
G997
```

```
[PLANT]=[.BG20]+[.BG21]
```

Choice of the plane for turning

```
[diam]=E70007
```

Programming by diameter

```

IF [PLANT]<>0 THEN
  IF [.BG20]=1 THEN @Y=X @X=Z @J=I @I=K
  ELSE @Y=Y @X=X @J=J @I=I
  ENDI
  IF [diam]=1 THEN [multi]=2
  ENDI
ELSE
E11005=0
  IF [.BG17]=1 THEN @Y=Y @X=X @J=J @I=
  ELSE
    IF [.BG18]=1 THEN @Y=X @X=Z @J=I @I=K
    ELSE
      @Y=Z @X=Y @J=K @I=J
    ENDI
  ENDI
ENDI
[axis1]= [TAB(3,NB)]*[multi] [axis2] [TAB(2,NB)]
G1 GL943 @Y [axis1] @X [axis2]
G998
FOR [NBLOC]=[NB] DOWNT0 2 DO
[G]=[TAB(1,NBLOC)]
  IF [G]=0 THEN [G]=1
  ELSE
    IF [G]=-1 THEN [G]=3
    ELSE [G]=2
  ENDI
ENDI
[axis1]=[TAB(7,NBLOC)]*[multi][axis2]=[TAB(6,NBLOC)]
[centre1]=[TAB(5,NBLOC)]*[multi][centre2]=[TAB(4,NBLOC)]
G[G] @Y[axis1] @X [axis2] @J[centre1] @I[centre2]
ENDF
ELSE
G1 GL943 G77 H[H] N[N1] N[N2]
ENDI
N9900 G80
E11005=[diam]

```

For turning by diameter (x2)

Address equivalence for milling

Profile start point equal to end point

Loop until the profile is finished

Execution of forward profile

Programming by diameter or radius as defined at the beginning restored

### Example 3

#### Peck drilling cycle created by NUM and called by function G83

Cycle %10083 calls subroutine %10080 to analyse all the cycles created by NUM (see subroutine %10080 following subroutine %10083).

#### Review of the syntax of cycle G83 for milling

N.. G83 X.. Y.. Z.. ER.. P.. Q.. F..
--------------------------------------

```

%10083:
(peck drilling cycle)
VAR [M3/4][M998][G90/1][G0/1][RF][clearance]=1 [diam]
    [IX][IY][IZ][LZ][I][dimension][depth][Gplan][E]
ENDV
[diam]=E11005 E11005=0
G77 H10080(call analysis module)
(check syntax: P present if previous block with G80)
IF [..BG80]=1 AND [.IBP(1)]=0
    THEN E.889
ENDI
(load P and Q if programmed)
IF [.IBP(1)]=1 THEN 'P=[.IRP(1)]
    IF [.IBP(2)]=0 THEN 'Q='P
    ENDI
ENDI
IF [.IBP(2)]=1 THEN 'Q=[.IRP(2)]
ENDI
(convert clearance if in INCHES)
IF [..BG70]=1 THEN [clearance]=[clearance]/25.4
ENDI
(clearance direction according to tool orientation)
IF [.RDX]<0 THEN [clearance]=-[clearance]
ENDI
(prepare positioning of the axes)
IF [..BG95]=1 THEN G0
ELSE F5000
ENDI
IF [I]>0 THEN G9 G998
    [dimension] = 'ER [I]=[IZ]+10 GO G77 H10080 N[I] N[I]
(assign correct sign to P and Q if programmed)
    IF 'P < 0 THEN 'P = -'P
    ENDI
    IF 'Q < 0 THEN 'Q = -'Q

```

```

EN DI
IF 'P > 0 THEN 'I=0 'L='ER-L[LZ]
(error if retraction plane = hole bottom)
IF 'L = 0 THEN E.891
ENDI
IF 'L < 0 THEN 'L=-'L
ENDI
IF 'Q = 0 OR 'Q > 'P THEN 'Q = 'P
ENDI
(calculate depth of first pass)
'I = 'Q-'P * 'I/'L + 'P + 'I
'J = 'L-'I
IF 'J <= 0 THEN
(go to bottom of hole)
[dimension]=L[LZ] G1 F[RF] G77 H10080 N[I] N[I]
G79N100
ENDI
IF 'J < 'Q/2 THEN 'I = -'Q/2 + 'L
ENDI
IF 'ER > L[LZ] THEN [depth] = -'I + 'ER
ELSE [depth] = 'I + 'ER
ENDI
(execute first pass)
[dimension]=[depth] G9 G1 F[RF] G77 H10080 N[I] N[I]
IF [.IBEL(6)]=1 THEN G4 FL931
ENDI
(retract -> ER)
[dimension]='ER G0 G77 H10080 N[I] N[I]
REPEAT
(calculate approach dimension)
[dimension]=[clearance]+[depth]G0 G77 H10080 N[I] N[I]
(calculate and execute next passes)
'I = 'Q-'P * 'I/'L + 'P + 'I 'J = 'L-'I
IF 'J <= 0 THEN
EXIT
ENDI
IF 'J < 'Q/2 THEN 'I = -'Q/2 + 'L
ENDI
IF 'ER > L[LZ] THEN [depth] = -'I + 'ER
ELSE [depth] = 'I + 'ER
ENDI
[dimension]=[depth] G9 G1 F[RF] G77 H10080 N[I] N[I]
IF [.IBEL(6)]=1 THEN G4 FL931
ENDI
[dimension]='ER G0 G77 H10080 N[I] N[I]
UNTIL 'I = 'L (test for end of loop)

```

```

        ENDI
        (go to bottom of hole)
        [dimension]=L[LZ] G1 F[RF] G77 H10080 N[I] N[I]
    ENDI
    N100
    (dwell specified by EF)
    IF [.IBE1(6)]=1 THEN G4 FL931
    ENDI
    (retract to ER)
    [dimension] = 'ER [I]=[IZ]+10 G0 G77 H10080 N[I] N[I]
    G997 G9 M[M998]
    G[G90/1] G[G0/1] F[RF] E11005=[diam]

```

#### Subroutine %10080 called by cycle %10083

```

%10080
(analyse drilling, tapping cycles, etc.)
IF [.IBE0(6)] = 1 THEN FL905
ENDI
IF [.IBE0(19)] = 1 THEN SL918
ENDI
IF [.IBE0(20)] = 1 THEN TL919
ENDI
BCLR [.IBE0(6)]/[.IBE0(19)]/[.IBE0(20)]
(read spindle rotation direction and M block sequencing)
[M3/4]=3*[.BM03] [M3/4]=4*[.BM04]+[M3/4]
[M998]=[.BM999]-[.BM997]+998 M997
(read tool axis number)
[IZ] = [.RDX]
IF [IZ] < 0 THEN [IZ] = -[IZ]
ENDI
(G21, G22 prohibited during a machining cycle)
[E]=[.BG21]+[.BG22] G79 [E]>0 N85
(plane and tool axis compatible?)
IF [.BG20]=1 THEN [Gplan]=20
ELSE [Gplan]=[.BG19]-[.BG17]+18
[E]=[Gplan]+[IZ] G79 [E]<>20 N83
ENDI
(read axis ranks and station in tool axis L900)
[LZ]=922+[IZ] G79 N[IZ]
N1 [IX]=5 [IY]=6 G79 N3+1
N2 [IX]=4 [IY]=6 G79 N3+1
N3 [IX]=4 [IY]=5
(choose primary or secondary axis)
(on the axis perpendicular to the tool axis)
IF [.IBX2(IX)] = 0 THEN [IX] = [IX]-3
ENDI

```

```

IF [.IBX2(IY)] = 0 THEN [IY] = [IY]-3
ENDI
(and on tool axis)
IF [.IBX2(IZ)] = 0 THEN [IZ] = [IZ]+3 [LZ]=[LZ]-3
  IF [.IBX2(IZ)] = 0 THEN E.880
  ENDI
ENDI
(store the last Z dimension in 'ER=. Return if ER was not programmed)
IF [.IBE1(18)] = 0 THEN 'ER = [..IRX(IZ)]
ELSE [E]=[IZ]-1*1000+70007
(if programming is by diameter, correct 'ER)
IF E[E]=1 THEN 'ER='ER/2
ENDI
ENDI
[LZ]=[LZ]+26 ( LZ points to variables L926..L951)
(on the first block to be initialised - hole bottom dimension)
IF [..BG80] = 1 THEN L[LZ]=[..IRX(IZ)]
ENDI
(if programmed, store the new hole bottom dimension)
IF [.IBX(IZ)] = 1 THEN L[LZ] = [.IRX(IZ)]
ENDI
(test whether the orientation is consistent with the machining direction)
IF 'ER <> L[LZ] THEN
  IF 'ER>L[LZ] AND [.RDX]<0 THEN E.890
  ENDI
  IF 'ER<L[LZ] AND [.RDX]>0 THEN E.890
  ENDI
ENDI
(store the type of positioning)
[G0/1]=3*[.BG03][G0/1]=2*[.BG02]+[.BG01]+[G0/1] [G90/1]=90+[.BG91]
(store feed rate, dwell)
[RF]=[.RF]
(if ED is programmed, read it and if linear interpolation is specified, force
positioning of the axes already programmed)
IF [.IBE1(4)] = 1 THEN EDL929
BCLR [.IBE1(4)]
IF [G0/1] < 2 THEN G91 [dimension]=0
  IF [.IBX1(IX)] = 1 THEN [I]=[IX]+10 G77 H10080 N[I] N[I]
  ENDI
  IF [.IBX1(IY)] = 1 THEN [I]=[IY]+10 G77 H10080 N[I] N[I]
  ENDI
ENDI
ENDI
(store presence then disable the tool axis)
[I]=[.IBX(IZ)]+[.IBX(IX)]+[.IBX(IY)] G90
BCLR [.IBX(IZ)]

```

```
(test spindle rotation if axis is programmed)
IF [I]<>0 THEN [E]=[M3/4]*[.RS] G79 [E]=0 N81
ENDI
G79 N800
(error message)
N81 E.831 (spindle stopped)
N82 E.882 (hole bottom not programmed)
N83 E.890 (tool orientation incompatible)
N84 E.894 (ER prohibited in G20)
N85 E.895 (G21, G22 prohibited during this cycle)
(-movements-)
N11 X[dimension]
N12 Y[dimension]
N13 Z[dimension]
N14 U[dimension]
N15 V[dimension]
N16 W[dimension]
N800
```

---

# 6 Polynomial Interpolation

---

<b>6.1</b>	<b>General</b>		6 - 3
<b>6.2</b>	<b>Programming Segmented Polynomial Interpolation</b>		6 - 3
	6.2.1	Notes on the Axes and Coefficients Programmed	6 - 4
	6.2.2	Geometric Transformations	6 - 4
	6.2.3	Interpolation Feed rate	6 - 5
	6.2.4	Limit on the Number of Coefficients	6 - 6
<b>6.3</b>	<b>Programming Smooth Polynomial Interpolation</b>		6 - 7
	6.3.1	Notes on Smooth Polynomial Interpolation	6 - 7
	6.3.2	Restrictions on Smooth Polynomial Interpolation	6 - 8



## 6.1 General

Polynomial interpolation is a tool for defining paths by polynomials. It is used for spline curve fitting.

The position on each of the axes is defined by a polynomial based on an independent (dimensionless) parameter which varies from 0 to 1 from the beginning to the end of the path.

There are two types of polynomial interpolation:

- segmented polynomial interpolation,
- smooth polynomial interpolation.

### Distinction Between Segmented and Smooth Polynomial Interpolation

With segmented polynomial interpolation, the segment size depends on the programmed feed rate. Each segment is calculated so as to be executed in 10 ms and is interpolated linearly for each sample.

With smooth polynomial interpolation, interpolation is carried out in real time, i.e. a point on the curve is calculated for each sample.

### Optional Functionalities

To be able to use smooth polynomial interpolation, it is necessary to enable option 52 (smooth polynomial interpolation). Otherwise, programming of argument I.. in the syntax is ignored and segmented interpolation is carried out if option 51 (spline curve) is enabled.

Segment polynomial interpolation (absence of I.. in the syntax) is accepted if either of options 51 and 52 is enabled.

## 6.2 Programming Segmented Polynomial Interpolation

In the block syntax, each polynomial is characterised by the end position followed by the coefficients of increasing degrees separated by «/».

### Syntax

```
N.. G01 X../ Coefficients / Coeff nth deg Y../ Coefficients Z../ Coefficients ...
```

G01	Linear and polynomial interpolation function.
X..	Interpolation end point X coordinate.
/ Coeff n <sup>th</sup> deg	Polynomial first, second degree, etc. coefficients.
Y.. Z..	Interpolation end point on the Y, Z and other axes.

## 6.2.1 Notes on the Axes and Coefficients Programmed

The sum of coefficients with degrees above zero must be equal to the relative movement of the axis in the block.

**REMARK** *It should be noted that no messages are sent if an error occurs in the sum of coefficients.*

The coefficients are expressed in:

- mm (or inches if G70 is active) for linear axes,
- degrees for rotary axes.

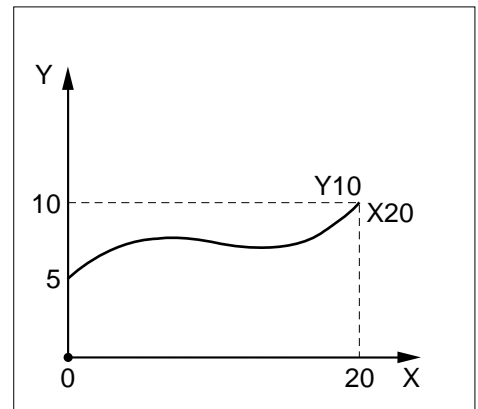
In a given polynomial interpolation block:

- certain axes can be programmed with different degrees,
- linear interpolation can be programmed.

Example:

```
N..
N80 G01 X0 Y5 Z-5
N90 X20/9.5/22/-11.5 Y10/18/-33/20 Z10
N..
```

The Z axis is interpolated linearly.



## 6.2.2 Geometric Transformations

All the following geometric transformations can be applied to the curve:

- programmed offset (G59),
- mirroring (G51 ...),
- scaling factor (G74),
- angular offset (ED..).

**REMARK** *To make an angular offset, both axes of the interpolation plane must be programmed and must have the same number of coefficients. The polynomial for one axis may have to be padded out with coefficients whose value is zero so that it has the same number of coefficients as the other axis of the plane. If this is not the case, the system returns error message 133.*

Example:

Without angular offset

```
G1 X20 Y0 /50/-180/240/-110
```

With angular offset ED..

```
G1 X20 /20/0/0/0 Y0 /50/-180/240/-110
```

### Radius correction using polynomial interpolation

With radius correction (G41 or G42), the normal tool offset is not carried out unless both axes of the interpolation plane are programmed.

Throughout interpolation (from the start point to the end point), the tool is held normal to the curve in the interpolation plane.

Consecutive polynomial curves must be tangent. If a curve not tangent with another polynomial curve (line or circle) is sequenced, the connection is made by a connection circle that positions the tool normal to the new curve at its start point.

When two curves are sequenced and the tool diameter is too large to be positioned on the normal to one of the programmed paths (connection radius smaller than the tool radius or path inaccessible), the system nevertheless applies the path continuity rules, which results in undesirable material removal.

6

### 6.2.3 Interpolation Feed rate

The curve segmenting step is directly related to the feed rate programmed:

- in G93 (V/L) and G94 (mm/min), the segmenting step is computed so as to be executed in 10 milliseconds (ms) if the sampling period is less than 5 ms,
- in G95 (mm/revolution), the segmenting step is equal to the programmed feed per revolution.

## 6.2.4 Limit on the Number of Coefficients

For each block, the coefficients are stored in the programme stack which has a maximum of 32 entries. Each coefficient is stored in an entry. In addition, for each axis, one entry is occupied by the physical address of the axis followed by the number of coefficients.

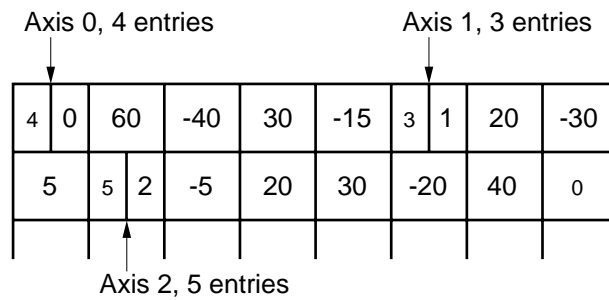
If the stack becomes full, the system returns the error message 6.

The coefficients have the same format and same limit values as dimensions.

Example:

Storage of the coefficients of a block in the stack.

N20 X100/60/-40/30/-15 Y70/20/-30/5 Z80/-5/20/30/-20/40



## 6.3 Programming Smooth Polynomial Interpolation

In the block syntax, each polynomial is characterised by the end position following by the coefficients of increasing degrees separated by «/».

Argument I.. in the syntax differentiates smooth polynomial interpolation from segmented polynomial interpolation (see Sec. 6.1).

### Syntax

N.. G01 X../Coefficients/Coeff nth deg Y../Coefficients Z../Coefficients ... I..

G01	Linear and polynomial interpolation function.
X..	Interpolation end point X coordinate.
/ Coeff nth deg	Polynomial first, second, etc. degree coefficients.
Y.. Z..	Interpolation end point on the Y, Z and other axes.
I..	Curve length (path on which the programmed feed rate applies).

### 6.3.1 Notes on Smooth Polynomial Interpolation

When declaring polynomials, the coefficient of the highest degree does not have to be specified. Since sum of the coefficients of an axis is equal to the relative movement, the system can determine the highest degree automatically.

Example:

```
...
G01 X0 Y..
X20/10/-5/
...
```

The third degree coefficient is equal to  $(20-0)-(10-5) = 15$

#### Special Application

With smooth polynomial interpolation, the check of parameter I.. can also be used to apply the programmed feed rate to a single axis instead of to the path.

Application of the feed rate to the X axis

```
...
G01 X10 Y10 F1000
X20 Y25/30/ I10
X35 Y50/25/ I15
...
```

Linear interpolation on X and 2nd degree interpolation  $(25-10)-30 = -15$  on Y X35  
 Linear interpolation on X and Y  
 (second degree coefficient = 0)

### **6.3.2 Restrictions on Smooth Polynomial Interpolation**

The following restrictions apply to smooth polynomial interpolation:

- Tool offsets (G41, G42, G29, etc.) are prohibited
- Backoff on the path is impossible
- Modulo rotary axes are not concerned
- The maximum degree of the polynomial is 5
- Traces and graphic simulations are limited to those generated by segmented interpolation.

---

## 7 Coordinate Conversions

<b>7.1</b>	<b>General</b>		7 - 3
<b>7.2</b>	<b>Using the Coordinate Conversion Matrix</b>		7 - 3
<b>7.3</b>	<b>Application of Coordinate Conversion</b>		7 - 5
		7.3.1	Restrictions and Conditions of Use 7 - 5
		7.3.2	Processing Time 7 - 5
<b>7.4</b>	<b>Example of Application Subroutine</b>		7 - 6



## 7.1 General

Coordinate conversions are made using a square matrix. This tool is used by the NUM application for machining in an inclined plane.

The conversion of movements  $X' = \vec{K} \cdot \vec{X}$  is made downstream of the interpolators by the matrix illustrated below.

### Coordinate conversion matrix

	X	Y	Z
X'	Kxx	Kxy	Kzx
Y'	Kxy	Kyy	Kzy
Z'	Kxz	Kyz	Kzz

*REMARK* The travels are checked and the speed and acceleration are limited upstream, when the interpolations are prepared.

## 7.2 Using the Coordinate Conversion Matrix

The coordinate conversion matrix uses the programming syntax given below. This syntax includes the matrix enabling function followed by coefficients P, Q and R separated by «/» (these coefficients are normed to 1).

### Syntax

N.. **G24+** X.. Y.. Z.. P(Kxx)/(Kyx)/(Kzx) Q(Kxy)/(Kyy)/(Kzy) R(Kxz)/(Kyz)/(Kzz)

G24+	Coordinate conversion matrix enabling function.
X.. Y.. Z..	Conversion matrix reference system origin.
P(Kxx)/(Kyx)/(Kzx)	Matrix X coefficients.
Q(Kxy)/(Kyy)/(Kzy)	Matrix Y coefficients.
R(Kxz)/(Kyz)/(Kzz)	Matrix Z coefficients.

### **Cancellation**

When enabled by (G24+ ...), conversion is cancelled by:

- G24- with no argument,
- turning the machine off and back on.

### **Properties of the Function**

The conversion arguments are modal. After cancellation by G24-, it is possible to reprogramme G24+ followed all the arguments or G24+ alone in the block.

The conversion defined by function G24+ is applied immediately and remains valid after a reset.

### **Notes**

The X, Y and Z origins define the origin of the matrix reference system with respect to the basic reference system declared by DAT1 + DAT2 (and possibly DAT3). These origins must mandatorily:

- follow function G24+,
- precede coefficients P, Q and R.

It is possible to enable the conversion matrix before homing on the axes (for instance to retract the tool according to the head position after applying power).

The conversion matrix must not be enabled when homing is requested on an axis. Otherwise, the system generates error message 24 in homing mode.

Function G24 can be used for other applications than rotation of the programming plane in space (for instance for correction of axes that are not orthogonal). In this case, the values of the conversion matrix coefficients must be between -8 and +8 (if not, the system generates error message 24). The same is true for the values of the inverse matrix calculated by the system. It should be noted that it is not necessary to comply with the angle crossing speeds and maximum accelerations.

### **Error Numbers and Messages**

Error 2: No + or - sign after function G24.

Error 14: Inclined plane option not enabled.

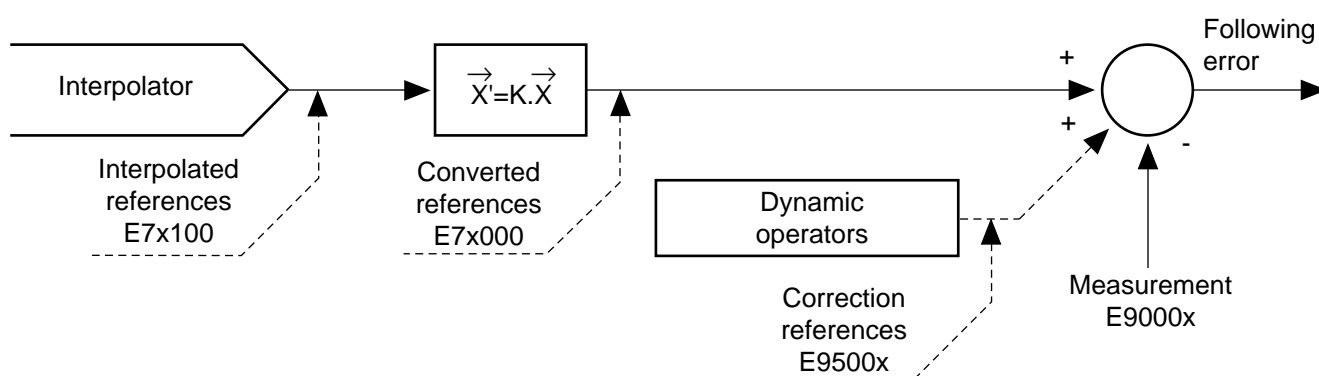
Error 24: Error in declaration of an inclined plane:

- Activation of the function when it is already enabled.
- Incomplete declaration of the arguments of the function.
- The axis of the pivot point does not exist or is not servo-controlled.
- Incoherent value of one of the matrix terms.
- Programming of an inclined plane axis prohibited under G52.

## 7.3 Application of Coordinate Conversion

Coordinate conversion must be applied after the interpolators but before processing of any dynamic operators and interaxis calibration.

### Schematic Representation



Interaxis calibration uses the converted references to calculate the corrections. Parameter E7x100 (read-only) applies to the position references output by the interpolators.

### 7.3.1 Restrictions and Conditions of Use

Copying (with or without conversion) the E7x100 interpolated references into the references used in the servo-control, E7x000, is carried out only for the axes declared in machine parameters P2 or P3 (possibly modified by parameter E9100x). It is necessary to take this into account in applications with dynamic operators using dummy axes.

The dynamic operators that calculate reference corrections (E9500x: destination operand) should not be declared when the conversion matrix is enabled. It is necessary to proceed as follows:

- cancel the conversion matrix (G24-),
- declare the dynamic operators,
- reenable the conversion matrix (G24+ ...).

The other declarations that can be made with the conversion matrix disabled are as follows:

- activation and deactivation of interaxis calibration by E9400x,
- declaration of an axis as servo-controlled or not by E9100x,
- activation and deactivation of dynamic operators 15, 20 and 21,
- homing.

### 7.3.2 Processing Time

Conversion processing time: 80 microseconds.

Interpolation preparation time: 500 microseconds more.

## 7.4 Example of Application Subroutine

### Application Subroutine %10150

This example is given only for guidance.

```

%10150
VAR [A]=[.RX(7)] [B]=[.RX(8)] [C]=[.RX(9)]
    [K11] [K12] [K13]
    [K21] [K22] [K23]
    [K31] [K32] [K33]
    [V] [X] [Y] [Z]
ENDV

(Reset the dimensions of the previous block in A, B and C)
IF [.IBX(7)] = 1 THEN
    [V]=[..IRX(7)] A[V]
ENDI
IF [.IBX(8)] = 1 THEN
    [V]=[..IRX(8)] B[V]
ENDI
IF [.IBX(9)] = 1 THEN
    [V]=[..IRX(9)] C[V]
ENDI

(Read the inclined plane pivot point)
(and reset the dimensions of the previous block in X, Y and Z)
[X]=[..IRX(1)] [V]=[..IRX(1)] X[V]
[Y]=[..IRX(2)] [V]=[..IRX(2)] Y[V]
[Z]=[..IRX(3)] [V]=[..IRX(3)] Z[V]

(Modify as required [X] [Y] [Z] of the spindle nose offset due to head rotation)
(Make sure there is no movement)
BCLR [.IBX(1)]/[.IBX(2)]/[.IBX(3)]/[.IBX(7)]/[.IBX(8)]/[.IBX(9)]

(Calculate the matrix coefficients)
[K11]=C[C]*C[B]
[K12]=S[C]*C[A] [K12]=C[C]*S[B]*S[A]-[K12]
[K13]=S[C]*S[A] [K13]=C[C]*S[B]*S[A]+[K13]
[K21]=S[C]*C[B]
[K22]=C[C]*C[A] [K22]=S[C]*S[B]*S[A]+[K22]
[K23]=C[C]*S[A] [K23]=S[C]*S[B]*C[A]-[K23]
[K31]=-S[B]
[K32]=C[B]*S[A]
[K33]=C[B]*C[A]

G24- (Disable the previous matrix)
(Enable the new matrix)
G24+ X[X] Y[Y] Z[Z] P[K11]/[K12]/[K13] Q[K21]/[K22]/[K23] R[K31]/[K32]/[K33]
G997 G80
  
```

**Call to the Inclined Plane Enable Function**

```
%...  
N..  
N.. G150 X.. Y.. Z.. A.. B..  
N..
```



---

## 8 RTCP Function

<b>8.1</b>	<b>General</b>		8 - 3
	8.1.1	Control of Rotary Axes	8 - 3
	8.1.2	Processing Performed on the Axes	8 - 3
<b>8.2</b>	<b>Using the RTCP Function</b>		8 - 4
<b>8.3</b>	<b>Description of Movements</b>		8 - 6
	8.3.1	Description of Twist Heads	8 - 6
	8.3.2	Description of Turntables	8 - 8
<b>8.4</b>	<b>Processing Related to the RTCP Function</b>		8 - 9
	8.4.1	Part on Inclined Plane	8 - 9
	8.4.2	Turntable Off-Centering (DAT3)	8 - 10
	8.4.3	Tool Length Correction	8 - 10
	8.4.4	Tool Wear Offset	8 - 10
	8.4.5	3D Tool Correction (G29)	8 - 10
<b>8.5</b>	<b>Use in JOG and INTERV Modes</b>		8 - 11
<b>8.6</b>	<b>Restrictions and Conditions of Use</b>		8 - 11



## 8.1 General

The RTCP function (Rotating Tool Centre Point) is used to continuously control the machine movements to orient the tool with respect to the part by pivoting it around its centre.

The machining programme contains the cartesian coordinates of the tool tip or point of contact of the part reference system.

NUM has created an application able to process most cases by the RTCP function.

### 8.1.1 Control of Rotary Axes

Two cases can occur, depending on the control mode of the rotary axes:

- the rotary coordinates are programmed and associated with the cartesian coordinates in the machining programme, in which case the mode is «programmed RTCP» mode,
- the rotary coordinates are not programmed and the rotary axes are controlled manually by handwheels or jogs. This mode is called «3/5 AUTO» (see the N/M AUTO function in Chapter 9 below).

### 8.1.2 Processing Performed on the Axes

The processing performed on the axes is used to:

- continuously (depending on the travel of the rotary axes) correct the references of the cartesian axes (X, Y, Z) to maintain the tool contact point on the programmed path. This correction is carried out in both programmed RTCP mode and in 3/5 AUTO mode,
- make sure the dimensions of the cartesian axes (X, Y, Z) corrected by the RTCP function remain within the machine travel limits. The check is made only in programmed RTCP mode,
- analyse the overspeeds and overaccelerations caused on the cartesian axes (X, Y, Z) by movement of the rotary axes to determine the optimum acceleration in the path and possibly limit the programmed speed so that no axis is driven beyond its maximum speed. This analysis is made only in programmed RTCP mode.



### Programming Errors

If a programming error is detected when the RTCP function is enabled, the system returns error message 16 which concerns the following errors:

- programming of G26+ when the RTCP function or inclined plane are already enabled. G24+ and G26+ are incompatible,
- vector IJK (or one of its components) not programmed (or double declaration of a component) before declaration of a twist axis or programmed when no twist axis is declared,
- twist axis declared after a turntable. With a twist axis, the machine movements must be described starting from the tool centre outward to the machine bed whereas in turntable mode, the movements must be described from the machine bed to the part,
- declaration of a rotary axis that does not belong to the group in which the RTCP is enabled,
- no arguments or too many arguments separated by the character «/». The rotary axes and pivot points must be programmed with three arguments and the inclinations with two,
- more than seven articulations and plane inclinations,
- no RTCP function number on the Options page of the CNC.

### Other Errors

The following errors can also cause generation of an error message:

- change of tool correction with the RTCP mode enabled, also causing generation of error message 16,
- when the RTCP mode is enabled, homing from the keyboard is inhibited and homing by the axis jogs generates error message 24,
- when the inclined plane is enabled, if the rotary axes are not homed, the system generates error message 159.

## 8.3 Description of Movements

The machine movements are described differently in twist and turntable modes:

- twist mode: the movements are described from the tool centre outward toward the machine bed,
- turntable mode: the movements are described from the machine bed toward the part.

The twist and turntable machine movements described herein are defined using a configuration example specific to each case. Depending on the case, the RTCP function is enabled with the specific arguments designating the rotary axes.

Different cases of configuration are processed by a NUM application.

### 8.3.1 Description of Twist Heads

In the case of twist heads, each articulation is described by a rotary axis and a translation vector. The translation vector is defined for the zero position of the rotary axis.

The rotary axis is declared by TA, TB or TC followed the three associated translations separated by the character «/».

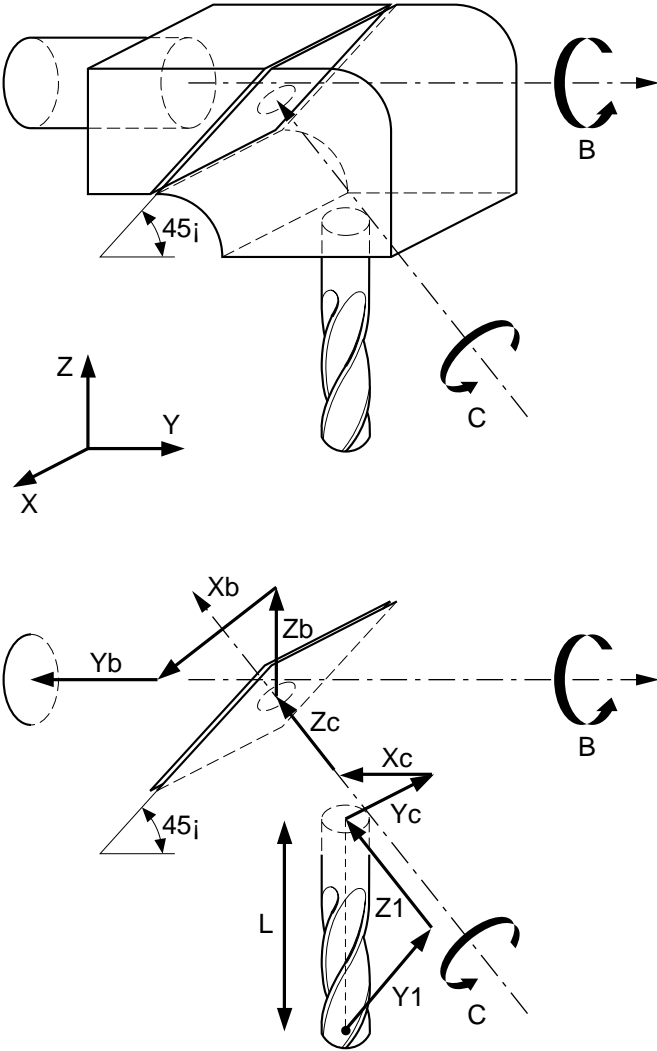
If an articulation rests on an inclined plane, the plane is declared by TI, TJ or TK, depending on whether it is parallel to X, Y or Z and is followed by the cosine and the sine of the angle of inclination of the plane.

The tool direction is defined by the components of a unit vector I, J, K pointing in the plane of rotation of the tool. This value is used to include the tool dimension and wear offsets that may be introduced during machining in the movements.

Vector I, J, K is the first argument following the RTCP function (G26+).

**Example**

Representation of twist head movements



Programming

```
[x1]=X1/L [y1]=Y1/L [z1]=Z1/L  
G26+ I[x1] J[y1] K[z1] TCXc/Yc/Zc TIca/sa TBXb/Yb/Zb
```

### 8.3.2 Description of Turntables

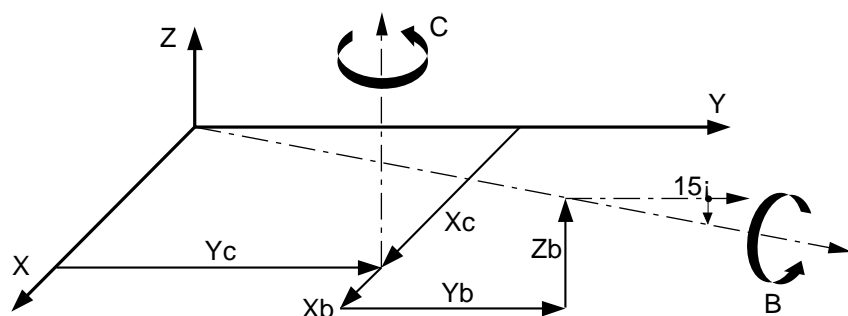
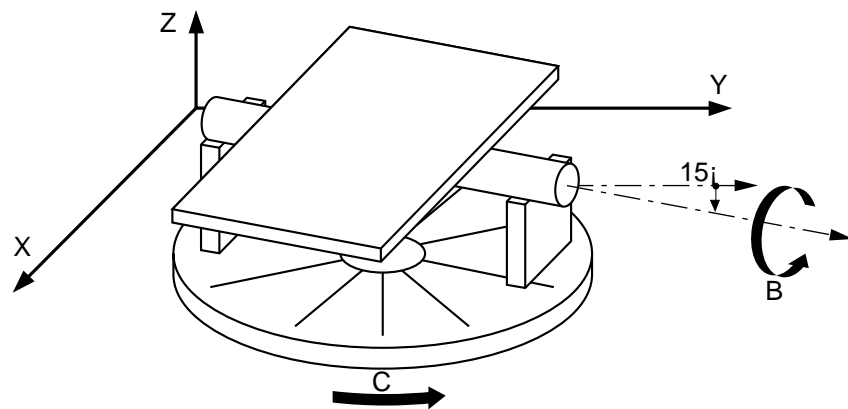
In the case of turntables, each articulation is described by a rotary axis and a translation vector. The first translation vector positions the centre of the first table (table resting on the machine bed) in absolute coordinates and the following table(s) incrementally for table zero positions.

The rotation axis is declared by PA, PB or PC followed by the three associated translations separated by the character «/».

If an articulation rests on an inclined plane, the plane is declared next by PI, PJ or PK depending on whether it is parallel to X, Y or Z, followed by the cosine and the sine of the angle of inclination of the plane.

#### Example

##### Representation of turntable movements



##### Programming

[ca]=cos 15° [sa]=sin 15°  
 G26+ PCxc/yc/0 PBxb/yb/zb PI[ca]/[sa]

## 8.4 Processing Related to the RTCP Function

The other geometric processing that can be associated with the RTCP function is as follows:

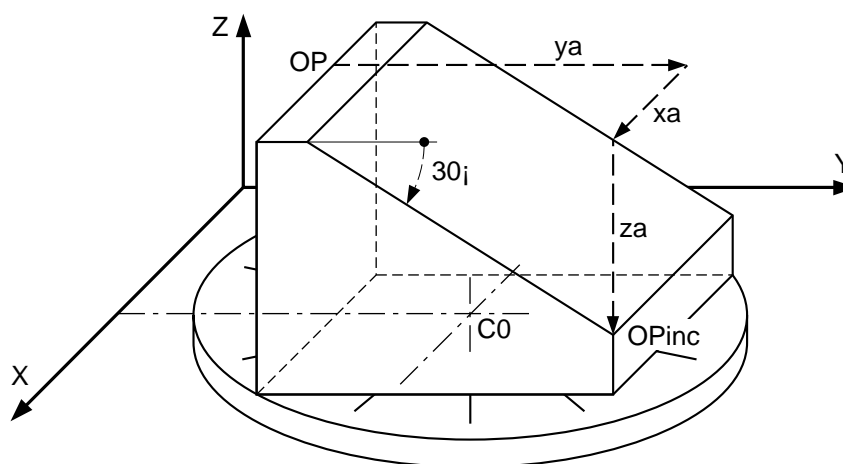
- part on inclined plane,
- table off-centering,
- tool length correction,
- tool wear offset,
- tool correction in space.

### 8.4.1 Part on Inclined Plane

When the part to be machined rests on an inclined plane, this plane is specified by a pivot point declared in the programming syntax by PD followed by the three components of a translation vector and by one or two inclinations defined by PI, PJ or PK followed by the cosine and sine of the angles. This translation vector positions the pivot point with respect to the programme origin and the pivot point becomes the new programme origin.

#### Example

Representation of movements on an inclined plane



OP = Old programme origin  
OPinc = New programme origin

#### Programmation

$[ca]=\cos-30^\circ$   $[sa]=\sin-30^\circ$   
G26+ PCxc0/yc0/zc0 PDxa/ya/za PI[ca]/[sa]

## 8.4.2 Turntable Off-Centering (DAT3)

Offsets related to off-centering of the turntable (DAT3) are no longer processed as such when the RTCP function is enabled. Since the position of the turntable rotary axis is known, this correction is automatically included in the RTCP function.

## 8.4.3 Tool Length Correction

When the tool is carried by a twist axis, the length correction is no longer processed as an offset on the tool axis, but is included in the twist movements with the orientation declared in the IJK vector programmed (see «Description of Twist Heads»).

Declaration and dimension assigned to the tool correction

The tool correction must be declared before enabling the RTCP function. The value assigned to the correction is stored when RTCP is enabled and saved as long as RTCP remains enabled. Any changes in the correction are ignored until RTCP is inhibited (G26-) then reenabled.

## 8.4.4 Tool Wear Offset

Tool length wear offsets less than  $100\ \mu$  (0.1 mm) are immediately taken into account and included (like the length correction) in the twist movements with the orientation declared in the vector IJK programmed (see «Description of Twist Heads»).

## 8.4.5 3D Tool Correction (G29)

### 3- or 5-axis tool correction in G29

#### 3-axis tool correction

When the RTCP function is used with 3-axis tool correction (no tool orientation vector IJK in the G29 blocks), the system processes only the case of the spherical tool.

#### 5-axis tool correction

When the RTCP function is used with 5-axis tool correction (presence of tool orientation vector IJK in the G29 blocks), the system is able to process the case of the toroid tool.

Display of the tool centre in G29.

During interpolation, the tool centre dimensions can be displayed on the «AXIS» current position coordinate page with respect to the programme origin (OP).

**Note on Tool Correction in G29 with Turntables (without Twist Axis)**

When the moving assembly of a machine includes only turntables (no twist axis), it is necessary to declare the tool direction vector IJK followed by argument TD after function G26+ before describing the turntable movements, in order for the 3D tool correction (G29) to be taken into account.

Example:

```
G26+ IJK1 TD0/0/0 PC../../. . . . .
...

```

**8.5 Use in JOG and INTERV Modes**

In manual JOG and intervention INTERV modes (after machining is stopped by CYHLD) when the RTCP function is enabled, movement of the axis by the jog is limited to a value such that no axis goes beyond the limit switches. Similarly, the acceleration and speed of movement on the axis can be reduced so that the other driven axes do not go beyond their limits.

In JOG mode controlled by handwheel, only the limit switches are controlled.

**8.6 Restrictions and Conditions of Use**

In applications with dynamic operators, certain restrictions and conditions of use of the RTCP function must be complied with.

The dynamic operators that calculate reference corrections (E9500x: destination operand) should not be declared when the RTCP function is enabled. It is necessary proceed as follows:

- cancel the RTCP function (G26-),
- declare the dynamic operators,
- reenale the RTCP function (G26+ ...).

The other declarations that can be made with RTCP disabled are as follows:

- activation and deactivation of interaxis calibration by E9400x,
- declaration of an axis as servo-controlled or not by E9100x,
- activation and deactivation of dynamic operators 15, 20 and 21,
- homing.



---

## 9 N/M AUTO Function

<b>9.1</b>	<b>General</b>		9 - 3
	9.1.1	General Requirements for N/M AUTO	9 - 3
	9.1.2	Uninterpolated Axes and NMA Axis	9 - 3
	9.1.3	Errors in N/M AUTO	9 - 4
	9.1.4	Processing Example	9 - 4
	9.1.5	Examples of Programmes and Use	9 - 5
<b>9.2</b>	<b>Using the N/M AUTO Function</b>		9 - 7
<b>9.3</b>	<b>Procedure After Enabling the N/M AUTO Function</b>		9 - 8
	9.3.1	Using the Jogs	9 - 8
	9.3.1.1	Case of Axis Clamping While E912xx Is Set	9 - 8
	9.3.2	Using the Handwheel	9 - 9
	9.3.2.1	Axis Assignment	9 - 9
	9.3.2.2	Example of Use with Handwheel	9 - 9
	9.3.2.3	Movements with the Handwheel	9 - 9
	9.3.2.4	Case of Axis Clamping While E912xx Is Set	9 - 10
<b>9.4</b>	<b>Stopping and Restarting in N/M AUTO Mode</b>		9 - 11
<b>9.5</b>	<b>Checks Included in N/M AUTO</b>		9 - 12
	9.5.1	Acceleration Checks	9 - 12
	9.5.2	Speed Checks	9 - 12
	9.5.3	Check of Travels	9 - 12
	9.5.4	Miscellaneous Checks	9 - 12



## 9.1 General

N/M AUTO means that N axes out of the M axes of a machine are controlled by the part programme and that the other axes are controlled manually. For instance:

- 2/3 AUTO for two axes interpolated and the third axis moved manually
- 3/5 AUTO for three axes interpolated and the other two moved manually.

The CNC axes to be moved manually must first be declared as “uninterpolated” (these axes are called “NMA axes”).

An NMA axis can be moved manually by:

- the axis jogs, or
- the associated handwheel.

The N/M AUTO function is active during machine cycles:

- simultaneously with control of the other interpolated axes in automatic (AUTO) or single step (SINGLE) mode at both the traverse rate and the feed rate
- provided the CNC is not in manual mode (JOG), nor stopped by programme (M12).

*REMARK The RTCP function (see Chapter 8 above) can also take effect downstream.*

### 9.1.1 General Requirements for N/M AUTO

For N/M AUTO operation, the following requirements must be satisfied:

- The automatic control function must be programmed in Ladder language with PLCTool
- Function 16 must be present on the Options page of the CNC
- Version G or above of the software must be installed. If the RTCP function is to be used, version J of the software is required (in addition, version J monitors the travels, speeds and accelerations on the driven axes).

### 9.1.2 Uninterpolated Axes and NMA Axis

At most five axes can be declared uninterpolated at any given time. An axis is declared uninterpolated by setting external parameter E912xx (xx = physical address of the axis from 00 to 31).

The axes declared uninterpolated are not effectively uninterpolated until the authorisation- enable signal has been received from the automatic control function by setting of variable %W2.1 (C\_NMAUTO); see use of the N/M AUTO function in Sec. 9.2.

*REMARK By definition, an uninterpolated axis is not moved under control of the part programme.*

It should be noted that among the five uninterpolated axes (at a given time):

- only one axis can be moved by NMA using the jogs or handwheel
- the other four (maximum) uninterpolated axes are not moved (at least not directly). However, if the RTCP function is enabled, an uninterpolated axis may be moved indirectly
- all the other axes are normally interpolated.

To be declared as uninterpolated, a CNC axis must:

- be servoed
- not belong to another group.

Uninterpolated axes can belong to any one of the eight axis groups, but the RTCP function is restricted to group 0.

*REMARK In no case can the N/M AUTO function be used to combine two movements on the same axis.*

### 9.1.3 Errors in N/M AUTO

Error message 99 is generated when:

- more than five external parameters E912xx are set
- parameter E912xx is set but associated axis xx is not servoed
- parameter E912xx is set in a part programme but the associated axis xx belongs to another group.

### 9.1.4 Processing Example

For example, a rotary C axis is controlled as an NMA axis using the handwheel. Via the RTCP function, this axis controls the X, Y and Z axes (which are programmed normally).

At all times, the CNC monitors the travel, speed and acceleration on the X, Y, Z and C axes.

For this example, it should be noted that:

- an axis declared as uninterpolated can still be programmed
- in this case, interpolation continues, but is inoperative (i.e. it does not directly cause any movement). However, as soon as the axis stops being uninterpolated, it is possible to resume the programmed path without even waiting for the end of the current block.

### 9.1.5 Examples of Programmes and Use

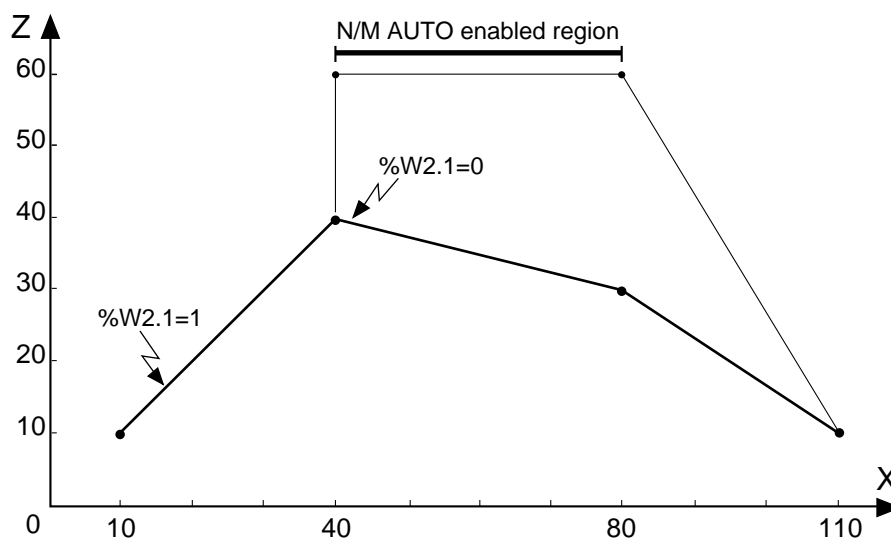
The programmes below define several possible cases of enabling and inhibiting of the N/M AUTO function.

#### Enable/Inhibit without CYHLD

##### Original programme

```
%10
E91202=1
X10 Z10
X40 Z40
X80 Z30
X110 Z10
M02
```

##### Schematic representation



##### Inhibiting taken into account after a cycle stop by function M00

```
%10
E91202=1
X10 Z10
X40 Z40
M00
X80 Z30
M00
X110 Z10
M02
```

Programme stop  
Z axis moved manually to 60  
Programme stop  
Z axis interpolated from 60 to 10

Inhibiting taken into account after resetting of parameter E91202

```

%10
E91202=1
X10 Z0
X40 Z40
M00
X80 Z30
E91202=0
X110 Z10
M02
    
```

Programme stop  
 Z axis moved manually to 60  
 Resetting of parameter E91202  
 Z axis interpolated from 60 to 10

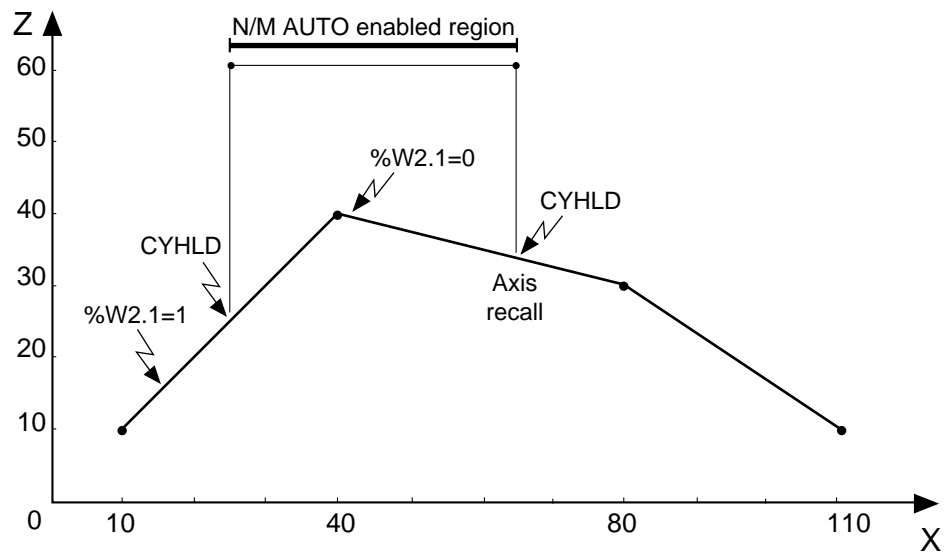
**Enable/Inhibit with CYHLD**

Original programme

```

%10
E91202=1
X10 Z10
X40 Z40
X80 Z30
X110 Z10
M02
    
```

Schematic representation



**REMARK** Both signals can appear simultaneously.

## 9.2 Using the N/M AUTO Function

When using the N/M AUTO function, a physical axis xx can be declared as uninterpolated:

- on the fly
- with the machine stopped (i.e. CYHLD LED lit or INCYC LED unlit).

The PLC programme must enable and/or confirm the N/M AUTO state by setting %W2.1=1. Physical axis xx is then an uninterpolated axis and can be moved by the axis jogs or handwheel.

%R2.1 (E\_NMAUTO) is the report of the CNC. This bit is accessible for read only and is set as soon as the N/M AUTO function is enabled, i.e. when at least one axis can be moved in NMA mode.

%W802.B (group 8 potentiometer) is used to control the feed rate on the NMA axis moved by jogs.

%W4.1 (VITMAN1) and %W4.2 (VITMAN2) are taken into account as in manual mode. These bits define the speed of movement when the handwheel or axis jogs are used.

### Example of Use

E91202=1 is programmed.

Two cases are possible depending on the state of %W2.1 (C\_NMAUTO) when E91202 goes high.

#### Case 1: Axis declared as uninterpolated on the fly

If the PLC has already enabled the N/M AUTO function:

When parameter E91202 goes high, variable %W2.1 (C\_NMAUTO) is already set. Physical axis 2 is immediately set to uninterpolated state. A report is available for the PLC and %R2.1 (E\_NMAUTO) goes high.

#### Case 2: Axis declared as uninterpolated with machine stopped

If the PLC has not yet enabled the N/M AUTO function:

When parameter E91202 goes high, variable %W2.1 (C\_NMAUTO) is low. The state of physical axis 2 remains unchanged. A report is available for the PLC and %R2.1 remains low.

If the PLC subsequently enables the N/M AUTO function:

Variable %W2.1 (C\_NMAUTO) goes high, but the state of physical axis 2 remains unchanged until the next cycle stop.

As soon as the machine stops (CYHLD LED lit, INCYC LED unlit), physical axis 2 is set to uninterpolated state and %R2.1 (E\_NMAUTO) goes high.

## 9.3 Procedure After Enabling the N/M AUTO Function

The jog type specifies whether the NMA axis is to be moved:

- by the axis jogs (in continuous mode only), or
- by the handwheel.

### Notes Regarding the Modes

The N/M AUTO function remains enabled in all the modes, but can only be used in the following modes:

- AUTOMATIC mode
- DRYRUN mode
- MDI mode
- INTERV mode
- SINGLE step mode
- Sequence number SEARCH mode (see remark below).

**REMARKS** *In SEARCH mode, only the axes defined as interpolatable are concerned by axis recall.  
It is possible to switch from one mode to another, but the mode change must always be carried out with the NMA axis stopped.*

### 9.3.1 Using the Jogs

Reminder: Only continuous mode is authorised for the axis jogs.

The NMA axis is moved in the positive or negative direction while the jog is set.

The speed of movement is defined by the setting of VITMAN1 and VITMAN2 (jog speed selection in manual mode) modulated by the potentiometer (%W802.B).

#### 9.3.1.1 Case of Axis Clamping While E912xx Is Set

An axis may be clamped for the following reasons (even if E912xx = 1):

- group 8 potentiometer is set to 0 (%W802.B)
- the feed authorisation is set to 0
- the NMA axis or a driven axis (if RTCP is active) is against a limit switch
- parameter E912xx was set while %W2.1 (C\_NMAUTO) was low. It is therefore necessary to generate a CYHLD or to programme function M00 for setting of parameter E912xx to be taken into account.

## 9.3.2 Using the Handwheel

### 9.3.2.1 Axis Assignment

In order for an NMA axis to be driven by the handwheel, the PLC must assign this uninterpolated axis to the handwheel:

[%W1x.B := uninterpolated axis name (x = A, B, C or D)]

enables axis feed commands in the desired direction(s):

[%W6.0 to %WD.7 := 1]

If the machine has several handwheels, they must be declared in machine parameter P14 and be present in the system. Handwheels that are present but not used must be set to -1.

### 9.3.2.2 Example of Use with Handwheel

Three handwheels are present in the system:

- Axis 0 is assigned to handwheel 0 in the positive and negative directions
- Axis 4 is assigned to handwheel 1 in the positive and negative directions
- No axis is assigned to handwheel 2.

Axis assignment to handwheels:

- %W1A.B=0
- %W1B.B=4
- %W1C.B=-1

Enabling of positive and/or negative directions:

- %W9.0=1                    %WD.0=1
- %W9.4=1                    %WD.4=1

### 9.3.2.3 Movements with the Handwheel

The NMA axis is driven by turning the corresponding handwheel. Any increments are applied to the diameter.

The axis speed is determined directly from the handwheel speed based on machine parameter P13 and variables VITMAN1 and VITMAN2.

The handwheel and NMA axis positions are the same at the end of movement except in the following cases:

- if the speed was limited to the maximum value for the NMA axis
- if the increment requests drove the NMA axis or a driven axis if RTCP is enabled beyond the travel limits
- if one handwheel is active, movement of the other handwheels is ignored.

**REMARK** *It is possible to use several handwheels, assigning an uninterpolated axis to each handwheel, but they cannot be used simultaneously. The system waits for the end of the movement generated by a handwheel before taking another handwheel into account. However, if a handwheel is active, movement of the other handwheels is ignored and lost.*

#### 9.3.2.4 Case of Axis Clamping While E912xx Is Set

An axis may be clamped for the following reasons (even if E912xx = 1):

- variable %W2.1 (C\_NMAUTO) is low
- the handwheel is not declared, is not recognised or certain values assigned to the machine parameters are incoherent (see P12, P13, P14)
- the axis was not assigned to the handwheel used (see %W1x.B)
- the axis feed and direction commands are set to 0 (see %W6.0 to %WD.7)
- the NMA axis or a driven axis (if RTCP is active) is against a limit switch
- parameter E912xx was set while %W2.1 (C\_NMAUTO) was low. It is therefore necessary to generate a CYHLD or to programme function M00 for setting of parameter E912xx to be taken into account
- an axis is already being driven by another handwheel.

## 9.4 Stopping and Restarting in N/M AUTO Mode

There are three ways of inhibiting an axis that has been declared as uninterpolated.

### Case 1

The part programme contains parameter E912xx=0. The N/M AUTO function remains enabled, but axis xx becomes interpolated.

%R2.1 (E\_NMAUTO) remains high.

If the NMA axis is being moved (by handwheel or jogs), programme execution resumes once the axis has stopped.

### Case 2

The PLC programme resets %W2.1 (C\_NMAUTO) and the INCYC LED is unlit. This completely inhibits the N/M AUTO function.

%R2.1 (E\_NMAUTO) goes low.

Error 107 is generated if the next block defines a circle. However, parameters E912xx remain unchanged but are ignored.

To reenble the N/M AUTO function, it is not necessary to set parameters E912xx explicitly to 0 then to 1.

When the PLC programme sets %W2.1 (C\_NMAUTO) from 0 to 1, and as soon as the INCYC LED goes out and/or the CYHLD LED comes on, the N/M AUTO function is reenbled immediately and:

- parameters E912xx that are set become operative
- %R2.1 (E\_NMAUTO) goes high.

### Case 3

The PLC programme resets %W2.1 (C\_NMAUTO) when the CYHLD LED is lit. The AXRCL LED comes on and the NMA axes must be recalled before continuing the part programme.

The N/M AUTO function is completely inhibited.

%R2.1 (E\_NMAUTO) goes low.

However, parameters E912xx remain unchanged but are ignored.

To reenble the N/M AUTO function, it is not necessary to set parameters E912xx explicitly to 0 then to 1.

When the PLC programme sets %W2.1 (C\_NMAUTO) from 0 to 1, and as soon as the INCYC LED goes out and/or the CYHLD LED comes on, the N/M AUTO function is reenbled immediately and:

- parameters E912xx that are set become operative
- %R2.1 (E\_NMAUTO) goes high.

**REMARK** *A reset returns all the axes to interpolated state (E912xx=0).*

## 9.5 Checks Included in N/M AUTO

The following checks are included in N/M AUTO:

- acceleration check
- speed check
- check of travels
- miscellaneous checks.

### 9.5.1 Acceleration Checks

When the N/M AUTO function is active, the maximum permissible acceleration is the maximum acceleration on the corresponding NMA axis (see machine parameter P32 (even word): F for work rate). If this acceleration is exceeded, the speed on the NMA axis is limited.

If the RTCP function is active, a second limit is applied so as not to exceed the maximum accelerations applicable to the work rates on the driven axes (see P32).

Reminder: If a driving axis of RTCP is declared as uninterpolated, the variation of the interpolation rates is prepared using an acceleration which is equal to half the maximum permissible acceleration.

### 9.5.2 Speed Checks

The speed on the NMA axis driven by handwheel is the speed defined in machine parameter P31. VITMAN1 and VITMAN2 are applied in manual mode.

If the RTCP function is active, a second limit is applied so as not to exceed the maximum speeds authorised on the driven axes (see P30).

Reminder: The interpolation speeds can be modulated by the values assigned to parameters E7x101.

### 9.5.3 Check of Travels

The overtravel test is carried out according to the definition of machine parameter P17. The NMA axis is stopped when it approaches the limit switches.

If the RTCP function is active and a driven axis approaches the limit switches, a cycle hold is generated automatically to stop the interpolator.

The NMA axis can be retracted from the limit switches using the axis jogs or handwheel controlling it.

The user can restart the programme by pressing START CYCLE.

### 9.5.4 Miscellaneous Checks

No test is made to check for conflict with G12 (overspeed). Homing must be completed on the NMA axes.

---

# Appendix A Table of Structured Programming Commands

**EXIT: Exit from a loop** (see 1.2.6)

**Syntax:**

**EXIT**

**FOR: Loops with control variable** (see 1.2.5)

**Syntax:**

**FOR** (variable) = (expression 1) TO/DOWNTO (expression 2) BY (value) DO  
(instructions)

**ENDF**

**IF: Conditional execution of instructions** (see 1.2.2)

**Syntax:**

**IF** (condition) THEN  
(instructions 1)

**ELSE**  
(instructions 2)

**ENDI**

**REPEAT: REPEAT UNTIL loops** (see 1.2.3)

**Syntax:**

**REPEAT**  
(instructions)

**UNTIL** (condition)

**WHILE: WHILE loops** (see 1.2.4)

**Syntax:**

**WHILE** (condition) DO  
(instructions)

**ENDW**

A



# Appendix B Table of Symbolic Variable Management Commands

<p><b>BUILD: Creation of a table for storing the paths of a profile</b> (see 4.2.1)</p> <p><b>General syntax:</b></p> <p><b>BUILD</b> [TAB(G / X / Y / I / J,NB)] H.. N.. +n N..+n</p>
<p><b>BCLR: Inhibiting of programming of M functions and/or one or more axes. Resets the bits of [•IBE0(i)] and [•IBE1(i)]</b> (see 4.2.5)</p> <p><b>Syntax:</b></p> <p><b>BCLR</b> [•BMxx] / [•IBX(i)] / [•IBE0(i)] / [•IBE1(i)]</p>
<p><b>BSET: Enabling of programming of M functions and/or one or more axes. Sets the bits of [•IBE0(i)] and [•IBE1(i)]</b> (see 4.2.5)</p> <p><b>Syntax:</b></p> <p><b>BSET</b> [•BMxx] / [•IBX(i)] / [•IBE0(i)] / [•IBE1(i)]</p>
<p><b>CUT: Elimination of grooves or parts of grooves located within the tool relief angle</b> (see 4.2.4)</p> <p><b>Syntax:</b></p> <p><b>CUT</b> * [Pa(7,Nb)] / Angle</p>
<p><b>MOVE: Copy of all of part of a table into another table</b> (see 4.2.8)</p> <p><b>General syntax:</b></p> <p><b>MOVE</b> [Pj(nj,mj)],mj1,mj2 = [Pi(ni,mi)],mi1,mi2 / j1=i1 / j2=i2 /jn=in</p>
<p><b>P.BUILD: Creation of a table for storing the dimensions of the interpolation plane of a profile</b> (see 4.2.2)</p> <p><b>General syntax:</b></p> <p><b>P.BUILD</b> [TAB(7,NB)] H.. N.. +n N..+n</p>

**R.OFF:** Normal offset of an open profile (see 4.2.3)

**Syntax:**

**R.OFF** [Pa(7,Nb)] /  $\pm 1$  / [R]

**SAVE:** Provision of a list of symbolic variables declared in any subroutine to the main programme and subroutines (see 4.2.7)

**Syntax:**

**SAVE** [Symb1] / [Symb2] ...

**SEARCH:** Search for a symbolic variable in the stack (see 4.2.6)

**Syntax:**

**SEARCH** [Symb] N..

---

# Appendix C Table of Programme Status Access Symbols

<b>C.1 Addressing G and M Functions</b>	<b>C - 3</b>
<b>C.2 Addressing a List of Bits</b>	<b>C - 3</b>
<b>C.3 Addressing a Value</b>	<b>C - 3</b>
<b>C.4 Addressing a List of Values</b>	<b>C - 4</b>



**Access Symbols to the Data of the Current Block or Previous Block**

Only the access symbols to the data of the current block are listed in the table. The access symbols to the data of the previous block, although not given, are exactly the same as those for the current block except that they are preceded by two decimal points instead of only one, e.g.: current block [**•**BGxx], previous block [**••**BGxx], etc.

**C.1 Addressing G and M Functions**

Symbol	See	Description
[ <b>•</b> BGxx]	2.2.1.1	G function addressing
[ <b>•</b> BMxx]	2.2.1.2	M function addressing

**C.2 Addressing a List of Bits**

Symbol	See	Description
[ <b>•</b> BI(i)]	2.2.1.3	List of arguments I, J and K programmed in the current block
[ <b>•</b> BP(i)]	2.2.1.3	List of arguments P, Q and R programmed in the current block
[ <b>•</b> IX(i)]	2.2.1.3	List of axes programmed in the current block
[ <b>•</b> IX1(i)]	2.2.1.3	List of axes programmed from the beginning of the programme to the current block
[ <b>•</b> IX2(i)]	2.2.1.3	List storing the last axes programmed XYZ or UVW
[ <b>•</b> IXM(i)]	2.2.1.3	Mirroring or not of the axes

**C.3 Addressing a Value**

Symbol	See	Description
[ <b>•</b> RD]	2.2.2.1	Tool correction number
[ <b>•</b> RDX]	2.2.2.1	Tool axis orientation (G16)
[ <b>•</b> REC]	2.2.2.1	Spindle orientation
[ <b>•</b> RED]	2.2.2.1	Angular offset
[ <b>•</b> RF]	2.2.2.1	Feed rate
[ <b>•</b> RG4]	2.2.2.1	Programmed dwell (G04 F..)
[ <b>•</b> RG80]	2.2.2.1	Number of the calling function in the subroutine call by G function

C

Symbol	See	Description
[•RN]	2.2.2.1	Number of the last sequence (block) found
[•RNC]	2.2.2.1	Value of NC function
[•RS]	2.2.2.1	Spindle speed
[•RT]	2.2.2.1	Tool number
[•RXH]	2.2.2.1	Current subroutine nesting level

#### C.4 Addressing a List of Values

Symbol	See	Description
[•IRD(i)]	2.2.2.2	Value of programmed angular offsets (G59 I.. J.. K..)
[•IRH(i)]	2.2.2.2	Current programme or subroutine numbers
[•IRI(i)]	2.2.2.2	Values of arguments I, J and K
[•IRP(i)]	2.2.2.2	Values of arguments P, Q and R
[•IRTX(i)]	2.2.2.2	Offsets programmed on the axes
[•IRX(i)]	2.2.2.2	Dimensions programmed on the axes

---

## Appendix D Table of Symbols Stored in Variables L900 to L951

---

D.1 Symbols Stored in Variables L900 to L925	D - 3
D.2 Symbols Stored in Variables L926 to L951	D - 3

---



## D.1 Symbols Stored in Variables L900 to L925

Symbol	See	L variables	Addresses
[•IBE0(6)]	3.2	L905	F
[•IBE0(8)]	3.2	L907	H
[•IBE0(14)]	3.2	L913	N
[•IBE0(15)]	3.2	L914	N
[•IBE0(19)]	3.2	L918	S
[•IBE0(20)]	3.2	L919	T

## D.2 Symbols Stored in Variables L926 to L951

Certain symbols in the table are not used.

Symbol	See	L variables	Addresses
[•IBE1(1)]	3.3	L926	EA
[•IBE1(2)]	3.3	L927	EB
[•IBE1(3)]	3.3	L928	EC
[•IBE1(4)]	3.3	L929	ED
[•IBE1(5)]	3.3	L930	EE
[•IBE1(6)]	3.3	L931	EF
[•IBE1(7)]	3.3	L932	EG
[•IBE1(8)]	3.3	L933	EH
[•IBE1(9)]	3.3	L934	EI
[•IBE1(10)]	3.3	L935	EJ
[•IBE1(11)]	3.3	L936	EK
[•IBE1(12)]	3.3	L937	EL
[•IBE1(13)]	3.3	L938	EM
[•IBE1(14)]	3.3	L939	EN
[•IBE1(15)]	3.3	L940	EO

Symbol	See	L variables	Addresses
[•IBE1(16)]	3.3	L941	EP
[•IBE1(17)]	3.3	L942	EQ
[•IBE1(18)]	3.3	L943	ER
[•IBE1(19)]	3.3	L944	ES
[•IBE1(20)]	3.3	L945	ET
[•IBE1(21)]	3.3	L946	EU
[•IBE1(22)]	3.3	L947	EV
[•IBE1(23)]	3.3	L948	EW
[•IBE1(24)]	3.3	L949	EX
[•IBE1(25)]	3.3	L950	EY
[•IBE1(26)]	3.3	L951	EZ

## Symbols

[..BGxx]	2.10
[..BMxx]	2.10
[..IBxx(i)]	2.10
[..IRxx(i)]	2.10
[..Rxx]	2.10
[.BGxx]	2.4
[.BMxx]	2.4
[.IBxx(i)]	2.6
[.IRxx(i)]	2.9
[.Rxx]	2.8

## A

Access symbols	
Boolean values	2.3-2.7
Data of current block	2.3-2.9
Numerical values	2.8
Read. <i>See:</i> Read, Access symbols	
Table. <i>See:</i> Table, Access symbols	
Addressing	
G functions	2.4
List of bits	2.6-2.7
List of values	2.9-2.10
M functions	2.4-2.5
Value	2.8
Axes	
Enabling. <i>See:</i> Enabling, Axes	
Inhibiting. <i>See:</i> Inhibiting, Axes	

## B

BCLR	4.18
Branches	1.5
BSET	4.18
BUILD	4.8
BY	1.10

## C

Cancelling	
RTCP function	8.4
Square matrix	7.4
Command	
Structured programming	1.3, 1.6-1.12
Symbolic variable management	4.8-4.31
Condition graph	1.6
Conditions	1.7
Coordinate conversion	7.3
Copying	
Blocks of a table	4.22-4.26
Blocks, partial	4.23
Blocks, simple	4.22
Table entries	4.22-4.26
Creating tables	
for storing profiles	4.6
for symbolic variables	4.3-4.7
CUT	4.15

## D

Data of current block	
Access symbols. <i>See:</i> Access symbols, Data of current block,	
Data of previous block	
Access symbols. <i>See:</i> Access symbols, Data of previous block	
Data stored in a table	4.7
Defining a table	4.3
DELETE	4.26
DO	1.9, 1.10
DOWNT0	1.10

## E

ELSE	1.7
Enabling	
Axes	4.18-4.19
M functions	4.18-4.19
RTCP function	8.4
Square matrix	7.3
ENDF	1.10
ENDI	1.7
ENDV	4.3
ENDW	1.9
EXIT	1.12
Exit from loop	1.12-1.14

## F

FOR	1.10
-----	------

## G

G functions	
Addressing. <i>See:</i> Addressing, G functions	

## I

IF	1.7
Indirect addressing of symbolic variables	4.27
Inhibiting	
Axes	4.18-4.19
M functions	4.18-4.19
Inhibiting display of subroutines	5.5
Initialisation	
Tables	4.6
Variables	4.5
Interpolation. <i>See:</i> Polynomial interpolation	6.3

## L

List of bits	
Addressing. <i>See:</i> Addressing, List of bits	
Addressing. <i>See:</i> Addressing, List of values	
Loop	
Exit. <i>See:</i> Exit from loop	
Repeat Until	1.8
While	1.9
with control variable	1.10-1.11

<b>M</b>			
M functions			
Addressing. <i>See:</i> Addressing, M functions			
Enabling. <i>See:</i> Enabling, M functions			
Inhibiting. <i>See:</i> Inhibiting, M functions			
Machining cycles	5.3		
Modal data of current block	2.3		
MOVE	4.22		
<b>N</b>			
Nesting	1.5		
Numerical values			
Access symbols. <i>See:</i> Access symbols, Numerical values			
<b>O</b>			
Offsetting an open profile	4.13-4.14		
<b>P</b>			
P.BUILD	4.11		
Peck drilling cycle	5.12		
Polynomial interpolation			
segmented	6.3		
smooth	6.3		
Profile storage			
Table creation. <i>See:</i> Creating tables, for storing profiles			
Programme status			
Access. <i>See:</i> Read, Access symbols			
Programme status access. <i>See:</i> Read, Access symbols			
Programming			
examples	4.28-4.31, 5.6-5.16		
segmented polynomial interpolation	6.3		
smooth polynomial interpolation	6.7		
Provision of symbolic variables	4.21		
<b>R</b>			
R.OFF	4.13		
Redefining a profile according to relief angle	4.15-4.17		
REPEAT	1.8		
RTCP	8.3		
RTCP function	8.3		
<b>S</b>			
SAVE	4.21		
SEARCH	4.20		
Search for symbolic variables	4.20		
Specifying entries to be copied	4.23		
Square matrix	7.3		
Storage			
Arguments EA to EZ	3.4		
Arguments F, S, T, H and N	3.3-3.4		
Profile interpolated in plane	4.11-4.12		
Stored symbols, table. <i>See:</i> Table, Stored symbols			
Structured programming		1.3-1.14	
Example. <i>See:</i> Structured programming examples			
Table. <i>See:</i> Table, Structured programming commands			
Structured programming examples		1.13-1.14	
Subrouting called by G function		5.2-5.17	
Table. <i>See:</i> Table, Symbolic variable management commands			
Symbolic variable management. <i>See:</i> Commands, Symbolic variable management			
Symbolic variables			
Creating tables. <i>See:</i> Creating tables, for symbolic variables			
Indirect addressing. <i>See:</i> Indirect addressing of symbolic variables			
Management. <i>See:</i> Commands, Symbolic variable management			
Provision. <i>See:</i> Provision of symbolic variables			
Search. <i>See:</i> Search for symbolic variables			
Tables. <i>See:</i> Tables, symbolic variables			
Symbols			
Syntax rules		1.3-1.4	
<b>T</b>			
Table dimensions		4.3-4.4	
Tables			
Initialisation. <i>See:</i> Initialisation, Tables			
Symbolic variables		4.1	
THEN		1.7	
TO		1.10	
<b>U</b>			
UNTIL		1.8	
Updating open profile table		4.13-4.14	
Using			
Coordinate matrix		7.3	
RTCP function		8.4	
<b>V</b>			
Value			
Addressing. <i>See:</i> Addressing, Value			
VAR		4.3	
Variables			
Initialisation. <i>See:</i> Initialisation, Variables			
<b>W</b>			
WHILE		1.9	