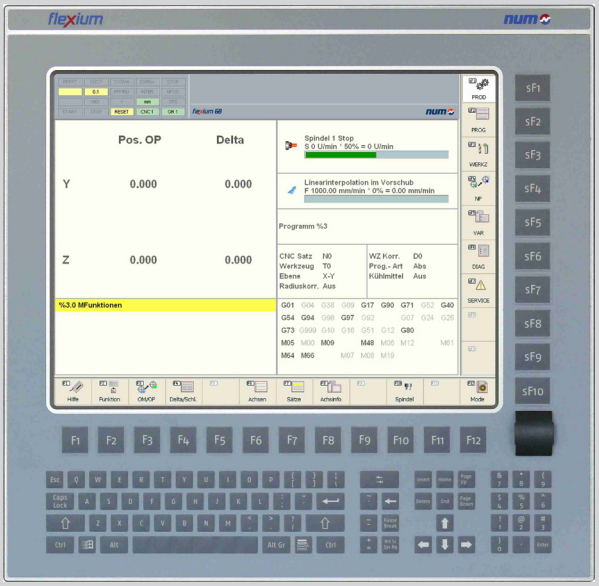
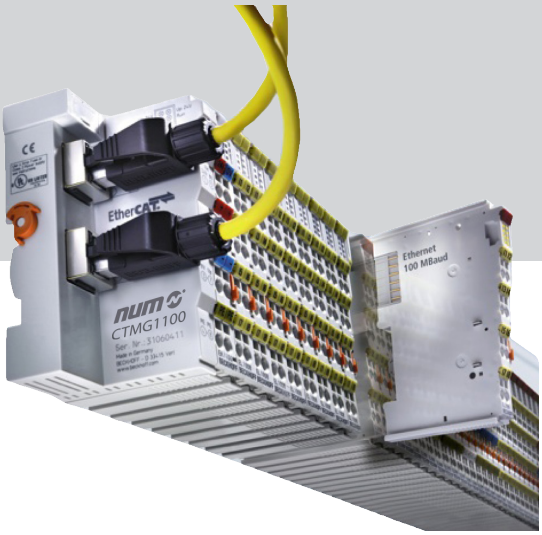
Flexium

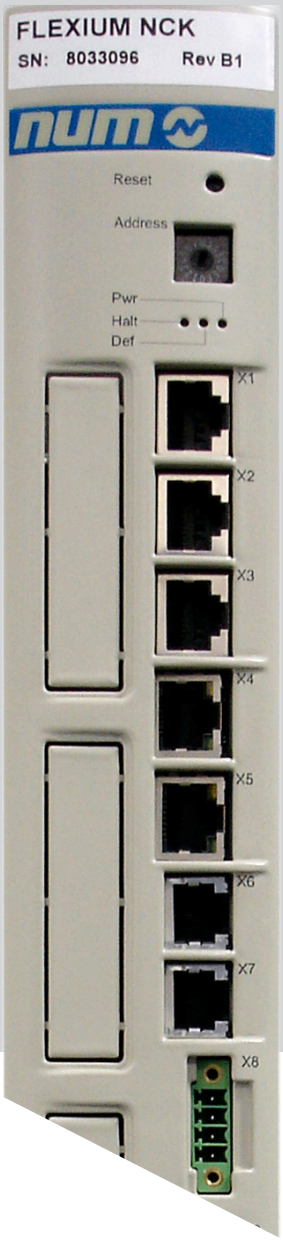
Extended Programming manual

M00020EN-02

Edition 05.2013







www.num.com

num®
CNC HighEnd Applications

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.

Summary

Page deliberately empty

Page deliberately empty

Summary

Page deliberately empty

1 Foreword

1.1 Use of the manual

Even though this manual contains information that can be used for machining a part, it is first aimed at the OEM for customizing the CNC to a dedicated solution.

This manual is for use by qualified personnel only.

To reduce the risk due to an improper use of the product, do not perform any servicing other than that contained in this manual unless you are qualified to do so.

To ensure proper use of these products, please always read this User's manual carefully and retain for future reference. Should the unit require maintenance, please contact a NUM authorized service center only.

1.2 Use of symbols in the manual

Symbol	Meaning of the symbol
	Important notes on product use.
	You may find this symbol whenever a wrong operation may damage the product.
	You may find this symbol whenever a dedicated Milling section or chapter is used.
	You may find this symbol whenever a dedicated Turning section or chapter is used.
	Please note that whenever functions in the manual present this Milling / Turning symbol means that such sections or chapters are used upon both cases.

1.3 Document revisions

Date	Index	Reason for revision	See chapter
04 - 2011	01	Document creation	-
05 - 2013	02	- New E3300x parameters - New E3400x parameters - New E3410x parameters - New E3600x parameters - New E4300x parameters - New E4400x parameters - New E4600x parameters - New E47000 to E47124 parameters	12

1.4 How to read the manual

Flexium Extended Programming manual gives information on how to create or adapt part program macros and optionally set the parameters in order to customise the Flexium system for a particular Milling or Turning machine-tool.

Chapter	Description
1	**Foreword** This chapter introduces: - The use of the Syntax conventions. - The general Relations with the PLC.
2	**Structured Programming** This chapter introduces: - General description of the commands and its use with the machine tool. - The general syntax rules.
3	**Reading the program status access symbols** - Presentation of the symbols giving visibility into the programmed functions and program context during call of a cycle by a G or M function.
4	**Program stack - L variables and symbolic variables** This chapter introduces the use of the stack, stack allocation, all the stack commands and symbols.
5	**Creating and managing symbolic variable tables** This chapter introduces a table creation, dimensions, values, commands and blocks.
6	**Subroutine call by G Functions** This chapter introduces: - List of G functions calling a subroutine - Subroutine example and graphical representation.
7	**Polynominal interpolation** This chapter introduces: - Polynominal interpolation use and syntax. - Segmented polynominal interpolation and smooth polynominal interpolation.
8	**Coordinate conversions** This chapter introduces the use of the Coordinate conversions and its application.
9	**RTCP Function** - Possibility of controlling the movements of a machine to position the tool with respect to the part and pivot it around its centre.
10	**N/M AUTO Function** - Possibility of controlling the N/M AUTO axes while the other machine axes follow a programmed path.
11	**B-Spline function** - Possibility to define a path described by several "Poles".
12	**Dynamic operators** - Overview and description of the syntax specific to each dynamic operators - Examples of programmes using the dynamic operators.
13	**Application specific E parameters** - Specific E parameters application as for: Continuous probing, Enabling reference writing for Virtual axes, Circle tolerance, Axis duplication and synchronisation, Monitoring drives test points ...
A	**Appendix** - Structured programming commands, Symbolic variable management commands, Program status access symbols, Symbols stored in L900 to L951 variables and List of Operations.

1.5 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 program 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 symbol **«...»** 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).
- **N** put as first character in the command line defines the sequence number.

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)
```

1.6 Relations with the PLC

This document will from time to time mention PLC flags. These data are mostly aimed at the OEM for activating the related function; detailed information can be found in the Flexium Commissioning manual M00010.

It should be noted that, for simplification reasons, the PLC flags are here mentioned in short form. For programming the PLC the full name including the NC identification should always be used.

e.g. : `W_ALLCNC.NMAuto` syntax in the present manual
`Flexium_NCK.W_ALLCNC.NMAuto` syntax used in PLC programming.

1.7 NUM Agencies

The list of NUM agencies is given on http://www.num.com

1.8 Index

The index at the end of the manual facilitates access to infomation by keywords.

Page deliberately empty

Page deliberately empty

Summary

Page deliberately empty

2 Structured Programming

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

The programming tools described in this chapter are used among other to create canned cycles (see chapter 6). Adapting these cycles or creating new ones requires their knowledge.

2.1 General

A structured sequence is defined within keywords.

It is introduced with one of the following:

- **IF**
- **REPEAT**
- **WHILE**
- **FOR**

It ends with:

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

2.2 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, DOWNTO, BY, DO, ENDF**
- Exit from a loop: **EXIT**

2.2.1 General syntax rules

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

The words **IF, REPEAT, THEN, ELSE, UNTIL, WHILE, DO** and **DOWNTO** 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 follow the condition **statement** immediately.

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 any 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 X12.5
```

or

```
WHILE L1 < 3
DO G91 X12.5
```

The following sequence is refused:

```
~~WHILE L1 < 3 G91 X12.5~~
```

2.2.2 Nesting and branches

Nesting

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

Example

	First level	Second level	Third level
IF			
	IF		
		REPEAT	
			- - - - - -
		UNTIL	
	ENDI		
ENDI			

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 program or subroutine.

Example

```
%1
IF …
          WHILE…
                G79 N100          Good
                G77 H2
                G79 N50           Wrong
                G79 N30           Wrong
                N30
          ENDW
          N50
ENDI
N100
M02



%2
G79 N100                          Good
          REPEAT
                IF …
                   G79 N100       Good
                   G79 N50        Wrong
                ENDI
                N50
          UNTIL …
N100
M02
```

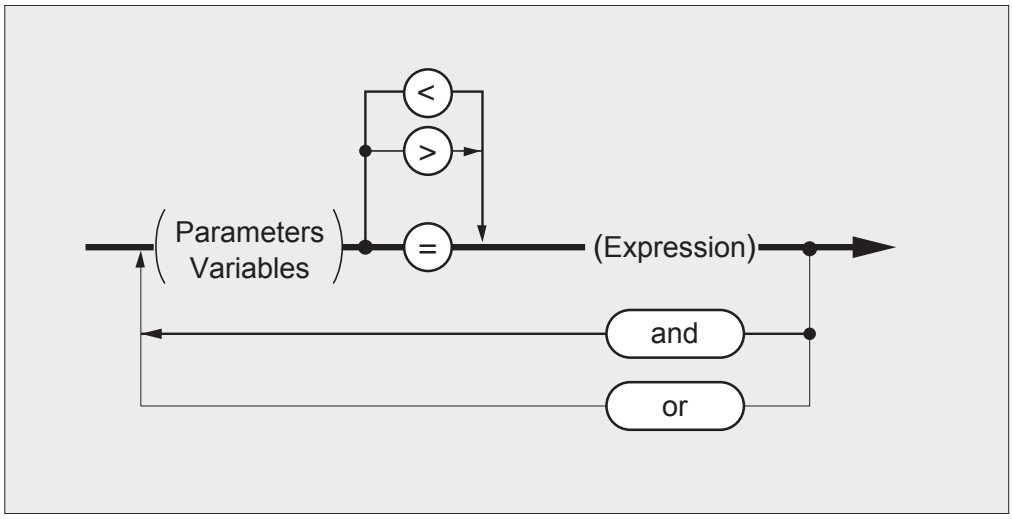
2.3 Structured programming commands

2.3.1 Condition graph

A condition must follow one of words **IF, WHILE** or **UNTIL** and must be located in the same block.

Limits and optional increment must immediately follow the word **FOR**.

Condition Graph



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

Be careful with L1xx and E parameters (*) as they can stop the block preparation. See note here under.

Expression Sequence of parameters and immediate values connected by symbols +, -, *, /, !, & (see Flexium Programming Manual M00018).
The operations are calculated in sequence from left to right.

Note

(*) Reading or writing an E parameter as well as writing an L1xx variable will stop block preparation because it is required to have the exact value at the very moment the block is executed.

A consequence is that it is not allowed to perform an operation on such parameters when in cutter compensation.

To overcome this case function M999 is available.

The goal of M999 is to authorize writing into variables L100 to L199, L900 to L999 and symbolic variables without holding block preparation. However M999 when active will prevent MDI mode and subroutine call by PLC to make sure such variables are kept strictly up to date.

2.3.2 Conditional Execution : ► IF, THEN, ELSE, ENDI

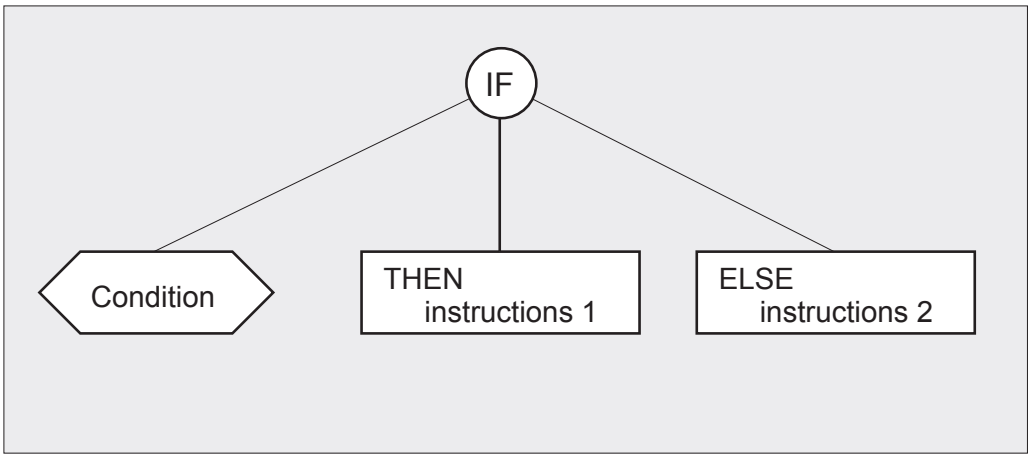
Syntax

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

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

The branch ELSE is optional.

Graph



Example

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

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

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

2.3.3 REPEAT Loop : ► REPEAT, UNTIL

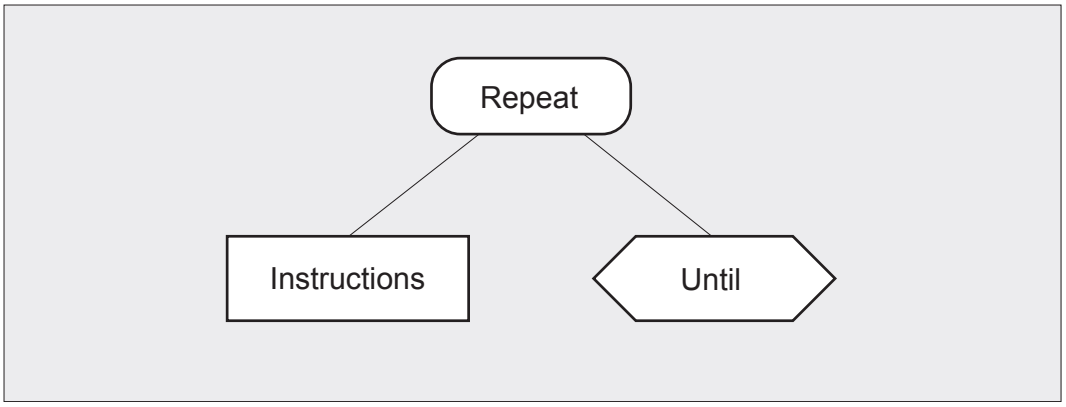
Syntax

REPEAT
(instructions)
UNTIL (condition)

The instructions are executed repetitively until the condition becomes true.

The instructions are executed at least once, even if the condition is false at start.

Graph



Example

Waiting for a operator answer

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

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

2.3.4 WHILE Loop : ► WHILE, DO, ENDW

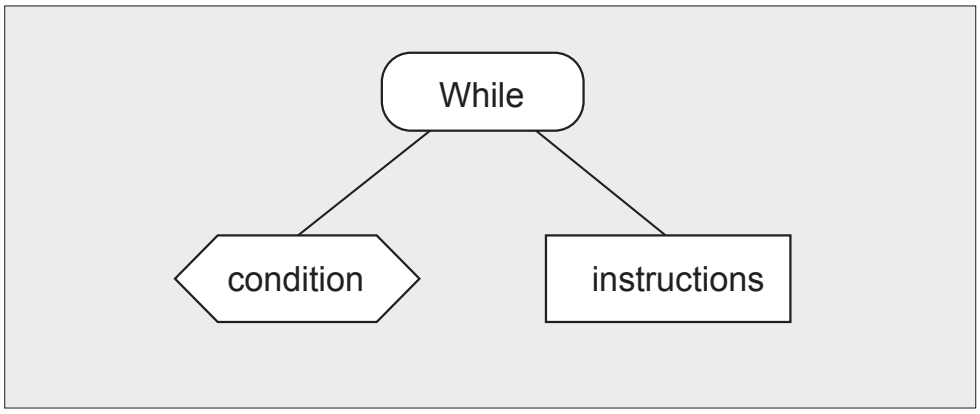
Syntax

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

The instructions are executed as long as the condition remains true.

Unlike REPEAT UNTIL loops, the instructions are not executed if the condition is false at start.

Graph



The mandatory 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
```

2.3.5 Loops with Control Variable : ► FOR, TO/DOWNTO, BY, DO, ENDF

Syntax

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

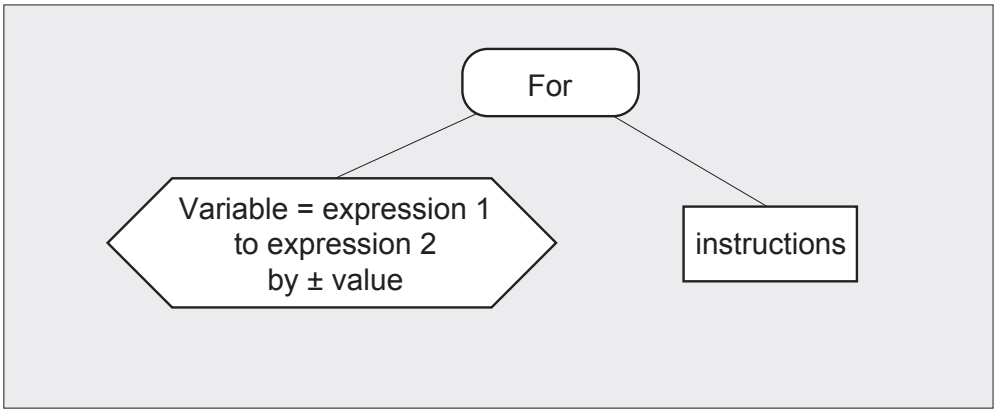
«Variable» is first initialised to *« expression 1»*.

Then the instructions are executed.

«Variable» is incremented (TO) or decremented (DOWNTO) by *«value»*

The process is continued as long as *«variable»* is not equal to *«expression 2»*.

Graph



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

Please note that the loop variable can only be an L parameter, a symbolic variable or an E80xxx parameter. If not Error 191 is triggered

- 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 (BY is optional and the default value of **BY** is equal to 1).
- **BY** must always have a positive value. If the value of BY is negative, functions **TO** and **DOWNTO** are reversed.
- The word **DO** is mandatory.

Example

Resetting the tool data

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

2.3.6 Exiting the Loop : ► EXIT

Syntax

EXIT

The EXIT instruction is used to exit from an iteration loop (**FOR, WHILE** or **REPEAT**) and return to its original nesting level, ignoring any subsequent IF functions.

The word EXIT is used only in IF loops.

Examples

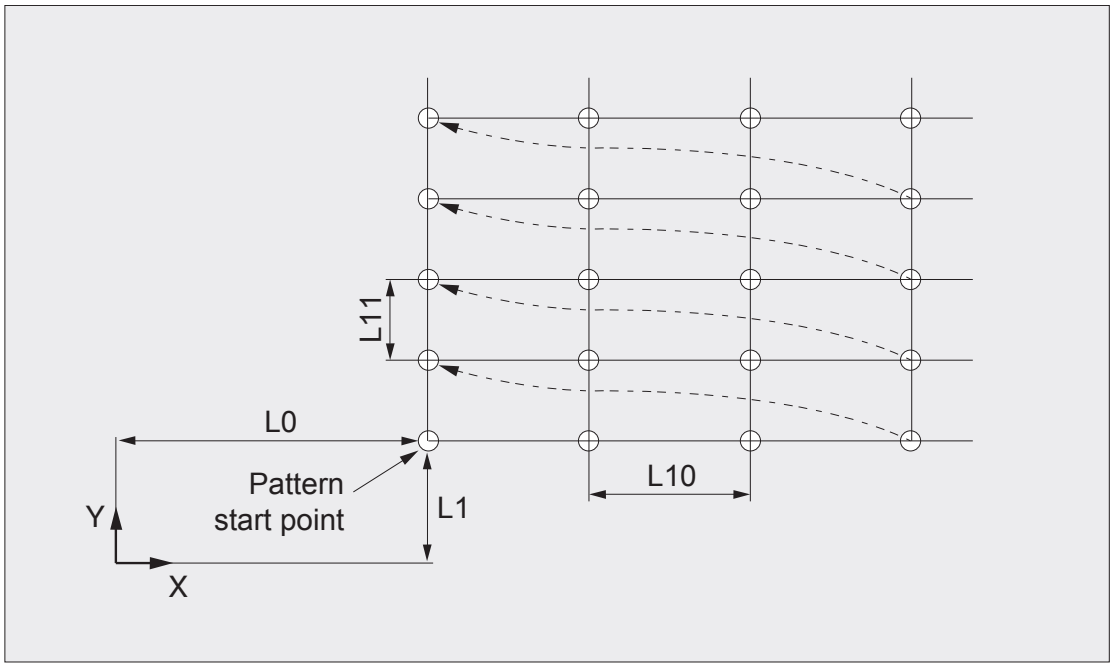
```
REPEAT
        (instructions)
        IF (condition) THEN
                (instructions)
        ELSE
                EXIT
        ENDI
UNTIL (condition)

(continued)

WHILE (condition) DO
        REPEAT
                IF (condition) THEN
                        (instructions)
                        IF (condition) THEN
                                EXIT
                        ENDI
                        (instructions)
                ENDI
        UNTIL (condition)

        (instructions)
ENDW
```

2.3.7 Example of Structured Programming

Hole drilling pattern

L2 = number of holes in X

L3 = number of holes in Y




```
%20
$ X START:                              Input of parameters
L0=$
$ Y START:
L1=$
$ X STEP:
L10=$
$ Y STEP:
L11=$
L4=0

WHILE L4 <> 25 DO                       Test for answer yes: Y
  REPEAT $ NUMBER OF POINTS IN X:
    L2=$
  UNTIL L2>0                            Test for number of columns greater than 0

  REPEAT $ NUMBER OF POINTS IN Y:
    L3=$
  UNTIL L3>0                            Test for number of rows greater than 0

  REPEAT $ PATTERN OF POINTS IN X:
    $=L2
    $+ /Y:
    $=L3
    $+ OK (Y/N) :
    L4=$
  UNTIL L4=14 OR L4=25                  Wait for answer: N (no) or Y (yes)
ENDW
```

```
$ THANK YOU                          GO !
N.. G00 G52 XO Z0
N.. T1 D1 M06                        Execution
N..
FOR L5=1 TO L3 DO                    Loop on rows
  $ ROW POSITION:
  $=L5
  XL0 L6=L5-1*L11+L1 YL6

  G00 Z0

  FOR L4=1 TO L2 DO                  Loop on columns
    $ DRILLING ROW:
    $=L5
    $+COLUMN:
    $=L4
    L6=L4-1*L10+L0 XL6

    G01 Z-10 F50
    G00 Z0

  ENDF
ENDF

N..
M02
```

Page deliberately empty

Summary

Page deliberately empty

3 Reading the program status access symbols

The programming tools described in this chapter are required for creating subroutines called by G functions.

3.1 General

The program 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 program context when the call is made.

These read-only symbols are accessible by parametric programming.

These symbols can:

- Access the data in the current block **[.]**
- 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. These symbols can be:

- **Symbols addressing a Boolean value**

  The symbol **[.Byxx]** addresses boolean values associated with program function y number xx.

  They are used to determine whether the functions are active or not.

  The Boolean values are defined by 0 or 1.

- **Symbols 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.

- **Symbols addressing a numerical value**

  The symbol **[.Rxx]** is used to address a value corresponding to an element (specified by xx) of the current block.

  The values are real numbers with reference to the element being searched.

- **Symbols addressing a list of values**

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

  The values are real numbers with reference to the element searched.

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

General Syntax

Variable = [.symbol(i)]

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

3.2 Symbols addressing a current Boolean value

3.2.1 Active G Functions

[.BGxx]

The symbol [.BGxx] is used to determine whether the G functions specified by xx are active or not:
[.BG86]=0: function G86 not active
[.BG70]=1: function G70 active

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]		

3.2.2 Active M Functions

[.BMxx]

The symbol [.BMxx] is used to determine whether the M functions specified by xx are active or not :
[.BM40]=0: function M40 not active
[.BM12]=1: function M12 active

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                          Tool check subroutine call
N..
N350 M02

%9000
VAR
[GPLANE] [MRANGE] [MSENS] [GINCH] [GABS]
[XRETURN] [YRETURN] [ZRETURN]
ENDV
$ SAVING CONTEXT
N10 [GPLANE]=17*[.BG17]
[GPLANE]=18*[.BG18]+[GPLANE]               Interpolation plane
[GPLANE]=19*[.BG19]+[GPLANE]

N20 [MRANGE]=40*[.BM40]                    Speed range
[MRANGE]=41*[.BM41]+[MRANGE]

N30 [MSENS]=03*[.BM03]
[MSENS]=04*[.BM04]+[MSENS]                 Direction of rotation
[MSENS]=05*[.BM05]+[MSENS]

N40 [GINCH]=70*[.BG70]                     Inches
[GINCH]=71*[.BG71]+[GINCH]

N50 [GABS]=90*[.BG90]                      Absolute or Incremental
[GABS]=91*[.BG91]+[GABS]

[XRETURN]=E70000
[YRETURN]=E71000                           Current machine coordinate
[ZRETURN]=E72000

N60 G90 G70 G00 G52 Z0
N70 G52 X100 Y100 M05                      Move to a predefined position
N80 G52 G10 Z-500
N …                                        Performs different operations


N100 G52 Z [ZRETURN]                       Return to the initial position
N110 G52 X [XRETURN]
N120 G52 Y [YRETURN]

G[GPLANE] M[MRANGE]                        Restore full context
M[MSENS] G[GINCH] G[GABS]
```

3.3 Symbols addressing a list of current status bit

This list can be read by parametric programming if the system is in state G999.

3.3.1 Programmed addresses

[.IBE0(i)]

[.IBE1(i)]

.IBE0 and .IBE1 are used to find which addresses are programmed in the block.

.IBE0(i) refers to D, F, H, N, N, S and T for an index value of 4, 6, 8, 14, 15 ,19 and 20.

The associated value will be respectively found in L903, L905, L907, L913, L914, L918 and L919

.IBE1(i) refers to all addresses from ‘EA (index 1 associated value in L926) up to ‘EZ (index 26 associated value in L951).

For additional information, see section 3.7.2 in this chapter.

3.3.2 I, J and K arguments programmed

[.IBI(i)]

Index i = 1 to 5. (List only modal in state G999)

i = 1	argument I
i = 2	argument J
i = 3	argument K
i = 4 in G21	argument J
i = 4 in G22	argument K
i = 5 in G21	argument I
i = 5 in G22	argument J

3.3.3 P, Q and R arguments programmed

[.IBP(i)]

Index i = 1 to 3. (List only modal in state G999)

i = 1	argument P
i = 2	argument Q
i = 3	argument R

3.3.4 Axes programmed in the current block

[.IBX(i)]

Index i = 1 to 11	
i = 1	X axis
i = 2	Y axis
i = 3	Z axis
i = 4	U axis
i = 5	V axis
i = 6	W axis
i = 7	A axis
i = 8	B axis
i = 9	C axis
i = 10 in G21	Y axis
i = 10 in G22	Z axis
i = 11 in G21	X axis
i = 11 in G22	Y axis

3.3.5 Axes programmed since the beginning of the program

[.IBX1(i)]

Index i = 1 to 9 Same as list of axes programmed in the current block (see [.IBX(i)]).

The axes last programmed with respect to the measurement origin (G52) are removed from this list, but will be included again as soon as programmed with respect to the program origin.

3.3.6 Primary or Secondary axes programmed

[.IBX2(i)]

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)]).

Programming a primary axis cancels its equivalent secondary axis and vice versa.

For instance, programming axis V resets the bit relative to Y and sets the bit for V.

3.3.7 Mirrored axes

[.IBXM(i)]

The bit is set to indicate mirroring and reset otherwise.

Index i = 1 to 9 same as the list of axes programmed in the current block (see [.IBX(i)]).

3.4 Symbols addressing current numerical values

3.4.1 Feed rate

[.RF]

Units as programmed by G93, G94 or G95.

3.4.2 Spindle speed

[.RS]

G97: format according the spindle characteristics declared in machine parameter P6.

3.4.3 Tool number

[.RT]

3.4.4 Tool correction number

[.RD]

3.4.5 Number of the last block encountered

[.RN]

If the current block is not numbered, [.RN] will return the last numbered block encountered.

3.4.6 Angular offset

[.RED]

3.4.7 Spindle orientation (milling)

[.REC]

3.4.8 Dwell time programmed (G04 F..)

[.RG4]

This is a non modal function however the value remain stored and can be read if the system is in state G999 or G998.

3.4.9 Canned cycle number

[.RG80]

In a subroutine called by G function, the G code number is contained in [.RG80].

When in G80 state the value is zero.

3.4.10 Spline curve number value

[.RNC]

3.4.11 Tool axis orientation

[.RDX]

Defined by the following signs and values:

Sign	Value
G16 P+	+1
G16 Q+	+2
G16 R+	+3
G16 P-	-1
G16 Q-	-2
G16 R-	-3

3.4.12 Nesting level of the current subroutine

[.RXH]

Defined by the following signs and values:

Sign	Value
1	main program
2	first nesting level
3	second nesting level, etc. (8 possible nesting levels)

3.4.13 List of axes programmed in the block

[.RLX]

[.RLX] contains the list of axes programmed in the current block **in coded format** i.e. X axis in bit 0, Y axis in bit 1, Z in bit 2 and so on.

3.5 Symbols addressing a list of current numerical values

3.5.1 Axes dimensions programmed

[.IRX(i)]

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	value of Y
i = 10 in G22	value of Z
i = 11 in G21	value of X
i = 11 in G22	value of Y

3.5.2 Offsets values programmed

[.IRTX(i)]

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

3.5.3 I, J and K values programmed

[.IRI(i)]

Index i = 1 to 5	
i = 1	value of I
i = 2	value of J
i = 3	value of K
i = 4 in G21	value of J
i = 4 in G22	value of K
i = 5 in G21	value of K
i = 5 in G22	value of J

3.5.4 P, Q and R. values programmed

[.IRP(i)]

Index i = 1 to 3	
i = 1	value of P
i = 2	value of Q
i = 3	value of R

3.5.5 Subroutine number

[.IRH(i)]

Index i = 1 to 1 to n subroutine (max 8)	
i = 1	main program number
i = 2	subroutine called by the main program number
i = 3	next subroutine,number etc. (8 possible nesting levels)

3.5.6 Programmed angular offsets origin (G59 I.. J.. K..)

[.IRDI(i)]

Index i = 1 to 3	
i = 1	value of I
i = 2	value of J
i = 3	value of K

3.6 Symbols accessing the data of the previous block

The exact same data can be accessed for the previous block.

The same base symbols are used, but in this case they are preceded by two decimal points instead of one.

[..symbol(i)]

Examples

[..IRTX(i)]

[..RG80] [..IBX(i)]

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 of the function G999.

3.7 Accessing data via L900 to L951

According to the machining cycle programmed, certain arguments or functions may have different meanings.

The system offers the variables L900 to L951. In a general case they are be used as temporary variables like the L1xx however in the case of custom macros, those variables can be used to retrieve the value passed with an argument.

The status symbols [.IBE0(i)] and [.IBE1(i)] will be used to detect if these arguments are programmed in the block calling a custom G cycle. They can be accessed by parametric programming.

3.7.1 F, S, T, D, H and N arguments

For each of these arguments a status symbol will be used to define if they are programmed and an L9xx variable to retrieve the value.

Status symbols

[.IBE0(4)]	set if D is programmed
[.IBE0(6)]	set if F is programmed
[.IBE0(19)]	set if S is programmed
[.IBE0(20)]	set if T is programmed

Values

L903	D
L905	F
L918	S
L919	T

3.7.1.1 Particular case of H and N

H and N are only be stored when they do not relate to G75, G76, G77 or G79.

Status symbols

[.IBE0(8)]	set if H is programmed
[.IBE0(14)]	set if the first N is programmed
[.IBE0(15)]	set if the second N is programmed

Values

L907	H
L913	first N argument (but not the bloc number)
L914	second N argument

3.7.2 EA to EZ arguments

Additional arguments can be defined by the codes EA to EZ.

Status symbol [.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).

Status symbols

[.IBE1(1)]	set if EA is programmed
[.IBE1(2)]	set if EB is programmed
....	and so on up to
[.IBE1(26)]	set if EZ is programmed

Values

L926	for EA
L927	for EB
...	and so on up to
L951	for EZ

3.7.2.1 Shortcut for 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 «‘».

- L900 to L925 can be addressed by symbols ‘A to ‘Z.
- L926 to L951 can be addressed by symbols ‘EA to ‘EZ

Examples

‘C = ‘A + ‘B is equivalent to L902 = L900 + L901

‘A = ‘B - ‘EA / ’EZ is equivalent to L900 = L901 - L926 / L951.

Page deliberately empty

Page deliberately empty

Summary

Page deliberately empty

4 Creating and managing symbolic variable tables

The programming tools described in this chapter are mostly used to create or edit subroutines invoked by G functions (custom G codes).

The rules for writing symbolic variables used when creating tables are the same as those defined for parametric programming. Please refer to Flexium Programming manual M00018.

4.1 Defining a table ► TAB

[TABLn(a,b,c ...)]

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 «,» .

4.1.1 Syntax

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

[TABLn(a,b,c...)] table name in the stack.

(a,b,c ...) : table dimensions

Each table declaration will use on the stack 16 bytes for the header plus n*6 bytes according to the number of elements.

Example of table definitions

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

For table 2 [TABL2] above, the number of items reserved equals: 2 x 5 x 3 = 30 entries.

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 each dimension must be between 1 and 255.

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

Examples

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

Notice that VAR1 is entered without brackets in the table definition.

A dimension cannot be defined using:

- L program variables,
- E external parameters.

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

The table indexes can be symbolic variables and be amended via additions or subtractions.

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
```

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) and separated by commas (as seen in the examples above).

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 and memory structure

Tables are stored in memory according to their first index then the second, the third...

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

`L0= [TABLE(2,3)]` : L0 will be loaded with 2

TABLE(1,1)	3
TABLE(2,1)	6
TABLE(1,2)	10
TABLE(2,2)	1
TABLE(1,3)	8
TABLE(2,3)	2
TABLE(1,4)	6
TABLE(2,4)	6

4.2 Creating tables for storing part profiles

The system offers the possibility of storing a profile written in ISO or PGP in a table. The table is created on the fly 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 can then be accessed by parametric programming.

4.2.1 Data that can be stored in a table

The following ISO programming data can be stored in tables.

4.2.1.1 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 it is one of the modal functions G00, G01, G02 or G03 which is stored.

4.2.1.2 Values of the Programmed Axes

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

The modal values programmed with the axes are stored.

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

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

4.2.1.4 Feed Rate F

The modal value related to the F function is stored.

4.2.1.5 Spindle Speed S

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

4.2.1.6 Tool Call T

The T code is not stored in the table unless it is present in the block, otherwise, a value of zero is stored.

4.3 Storing a profile ► BUILD

The BUILD function creates and fills a two dimension table for storing profile paths.

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

4.3.1 Syntax

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

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 second dimension of the table (number of blocks: maximum 255).
H..	Definition of the limits of the profile.

- N.. N..
- H.. N.. N..
- N..+n N..+n
- H N..+n N..+n

The BUILD function must be the first word in the block.

Example

```
%555                                  %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..
```

The two-dimension table **TAB** is created by the above program:

- First dimension: 5 entries (G,X,Y,I,J)
- Second dimension: 4 entries (4 blocks in %55) ← NB = 4

4.3.2 Allocation of additional entries

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

Example

```
%55
BUILD [PROF1(G/X/Y/Z/0/0,NB)] N110 N220
```

The first table dimension will include 6 entries.

4.3.3 Declaring entries as a list of bits

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 of declaration

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

Declaration of I.Symb

The value of symbolic variable [Symb] is defined by the sum of the indexes ranks for the required symbols.

- 1 for index i = 1
- 2 for index i = 2
- 4 for index i = 3
- 8 for index i = 4
- … etc

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..
```

4.4 Storing a profile interpolated in the plane ► P.BUILD

The P.BUILD function 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.

4.4.1 Syntax

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

TAB	Table name in the stack.
7	Number of entries in the first dimension (can be other value, 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

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.

4.4.2 Entries of the first dimension

Entry 1	Type of interpolation: • Value = 0 for linear interpolation • Value = -1 for clockwise circular interpolation • Value = +1 for counterclockwise 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 on X: • Value on the X axis for circular interpolation, otherwise = 0
Entry 5	Position of the centre on Y: • Value on the Y axis for circular interpolation, otherwise = 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
N10 G17 X5 Y10
P.BUILD [TAB(7,NB)] H80 N110 N130
N..
```

```
%80
N10 ...
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:

NB = 3

TAB

0	20	10	0	0	5	10
+1	20	50	20	30	20	10
0	10	20	0	0	20	50

4.5 Offsetting an open profile and updating ► R.OFF

The R.OFF function is used to normally offset a profile that was initially created by a 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).

4.5.1 Syntax

R.OFF [Pa(7,Nb)] / ±1 / R

Pa	Table.
Name	Number of table entries.
Nb	Name of the variable containing the number of blocks in 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.

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

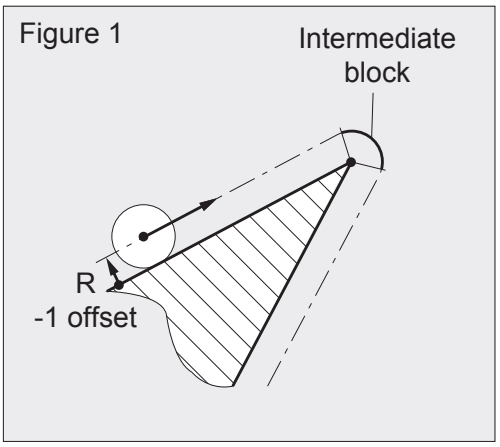
The profile executed with the R.OFF function must be an open profile, i.e. the start point of the profile must be different from the end point (see Fig. 2).

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. (Figure 1)

Figure 1



Open contour

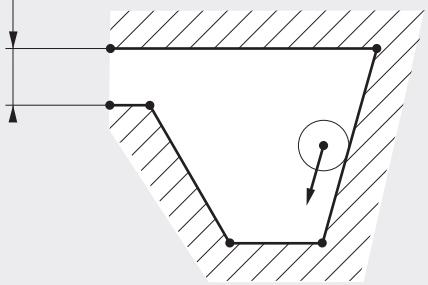
When the profile includes a narrowed section, it must be large enough to allow passage of the tool.

Otherwise, the system considers the profile to be closed.

(Figure 2)

Figure 2

Narrowed section > tool



Example

```
%92
N..
P.BUILD [P(7,NB)] N110 N200          Profile building
...
…
R.OFF [P(7,NB)] /-1 /10              Profile offset to the left by 10
```

4.6 Redefining a profile according to the tool clearance angle ► CUT

The CUT function is used to eliminate the grooves or parts of groove located inside the tool clearance angle.

It applies to grooves located in the path of a plane profile created in a table by the P.BUILD function or a table of 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.

4.6.1 Syntax

CUT * [Pa(7,Nb)] / Angle

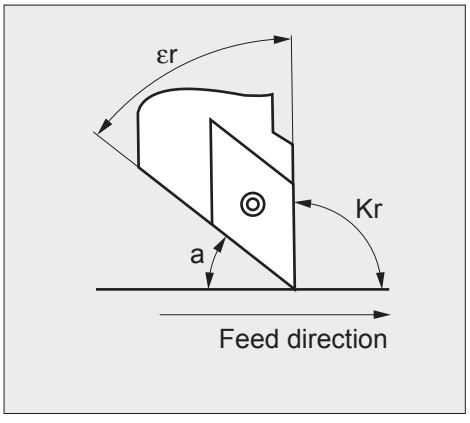
*	When the character * precedes the table name, all the grooves located within the tool relief angle are processed. When the character * is missing in front the table name, only the first groove located within the tool relief angle is processed.
Pa	Table name.
7	Number of table entries.
Nb	Name of the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

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

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

Characteristic angles:

- Kr: Approach angle,
- εr : Tool nose angle,
- a : Clearance angle



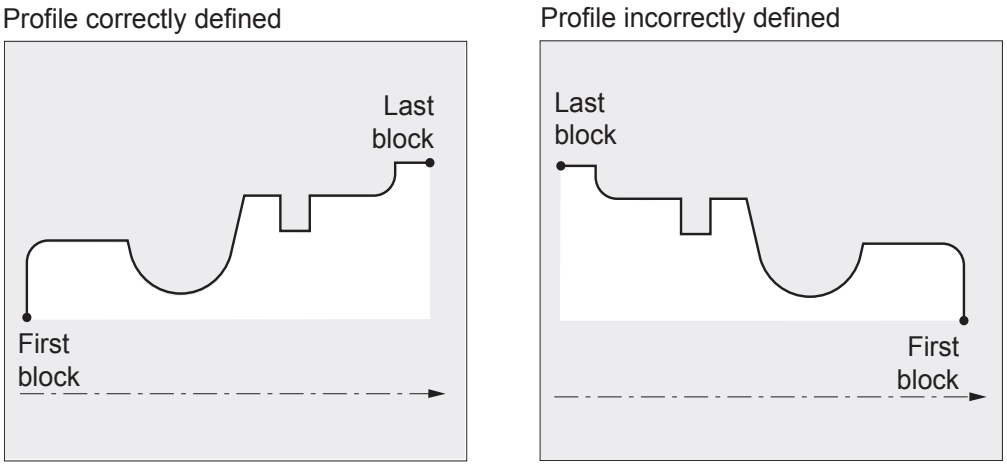
Processing of the table

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 on the X axis the profile start dimension is less than the profile end dimension.

Example



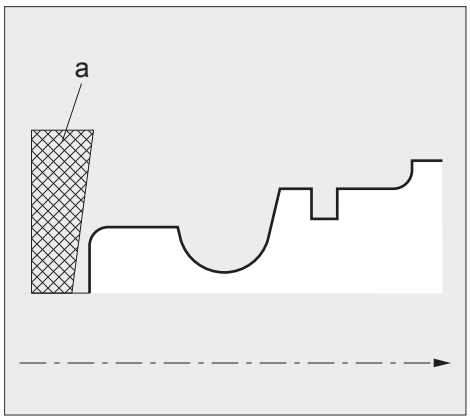
When the clearance angle is negative or zero (between 0 degrees and -180 degrees), the profile areas located below this angle are eliminated.

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

Examples

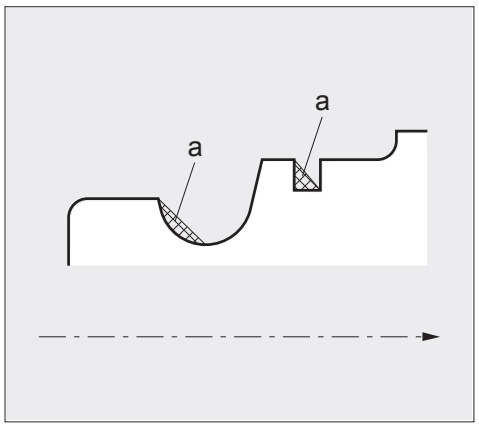
Elimination of the first groove in the profile (no * in front of the variable). «a» shows the areas that are eliminated.

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



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

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



4.7 M Functions and/or axes motion enable/ disable ► BSET, BCLR

- BSET Enables the execution of M functions and/or axes motions, and sets the bits of [.IBE0(i)] and [.IBE1()].
- BCLR Disables the execution of M functions and/or axes motions, plus resets the bits of [.IBE0(i)] and [.IBE1(i)].

4.7.1 Syntax

BSET [.BMxx] / [.IBX(i)] / [.IBE0(i)] / [.IBE1(i)]	
[.BMxx] / [.IBX(i)]	In state G999, 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)]

BCLR [.BMxx] / [.IBX(i)] / [.IBE0(i)] / [.IBE1(i)]	
[.BMxx] / [.IBX(i)]	In state G999, 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

The 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.
- 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 (@2) and B (@8) axes as well as function M05 are inhibited.

```
N..
N..
N.. G999
X10 Y10 Z10 B30 M05
…
BCLR [.IBX(2)] / [.IBX(8)] / [.BM05]
…
N120 G997
```

4.8 Searching the stack for symbolic variables ► SEARCH

4.8.1 Syntax

SEARCH [Symb] N..

[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..
```

4.9 Preserves a list of symbolic variables ► SAVE

SAVE Saves a list of symbolic variables for use by the root program or any of its subroutines.

4.9.1 Syntax

SAVE [Symb1]/[Symb2]...

[Symb1]/[Symb2]... List of symbolic variables.

Notes

The symbolic variables in the list may be declared within subroutines at any nesting levels.

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

No space character is admitted before and after the «/» otherwise the alarm 1 will occur.

Example

After returning from subroutine %30, the program %10, the subroutine %20 (as well as any new subroutines) can use the symbolic variables [V1] and [TB(4)], until a RESET or M02 is encountered.

```
%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..
```

4.10 Copying Blocks from one table to another ► MOVE

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.

The following code (to be completed) will create a profile table (defined in a part program subroutine) and modify it for backwards execution.

```
P.BUILD [T(7,NB)] HL907 NL913 NL914   (Create a table from a part program profile )

VAR [S(7,NB)]                         (Create the work table with the same dimension)
ENDV

MOVE [S(7,NB)],2,[NB] = [T(7,NB)],[NB],2/1=-1/2=6/3=7/6=2/7=3/4=4/5=5
```

(The work table is made from the Profile table with the following changes)

(Blocks inverted : 2,[NB] ➔ [NB,2])

(G2 and G3 inverted ➔ /1=-1)

(Start and End point on X interverted /2=6 and /6=2)

(Start and End point on Y interverted /3=7 and /7=2)

(Circles center on X and Y maintained /4=4 and /5=5)

4.10.1 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.

4.10.2 Syntax for simple copy of Blocks

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

For simple copy both tables must have the same format. i.e. nj=ni and mj=mi.

4.10.3 Syntax for partial blocks copy

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

- 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.

4.10.4 Syntax for specifying entries to be copied

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

- 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 particular entries need to be copied, each of them must be preceded by the character «/», the target entry index followed by the character «=» and the source entry index.
  The value of an entry copied to the target table can be inverted by preceding the source entry index by the «-» sign.

Notes

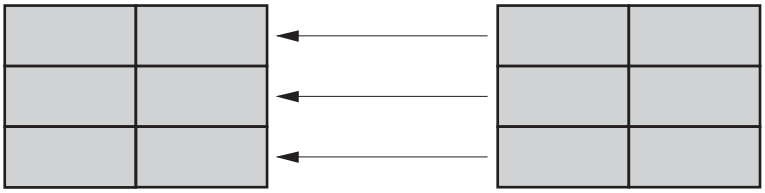
- The MOVE function must be the first word in the block (no sequence number).
- The maximum number of blocks in a finished profile is limited to 95.
- It is possible to reverse the order of a copy by reversing the start and end limit indexes in one of the tables.
- The indexes can be specified in symbolic 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.

4.10.5 Examples

MOVE simple copy of blocks

Copy the content of a table into another table

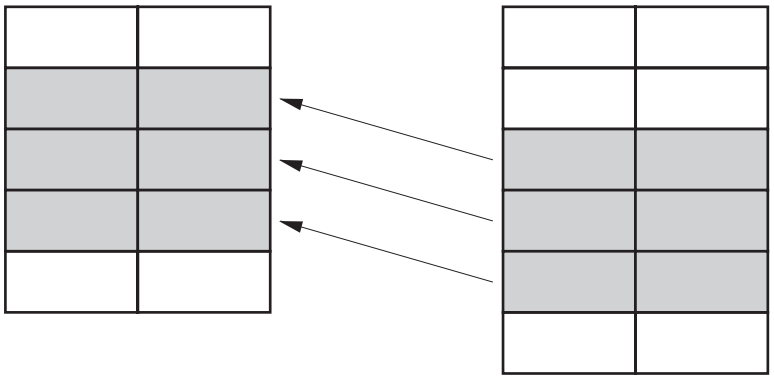
MOVE [PB(2,3)] =[PA(2,3)]



MOVE partial copy of blocks

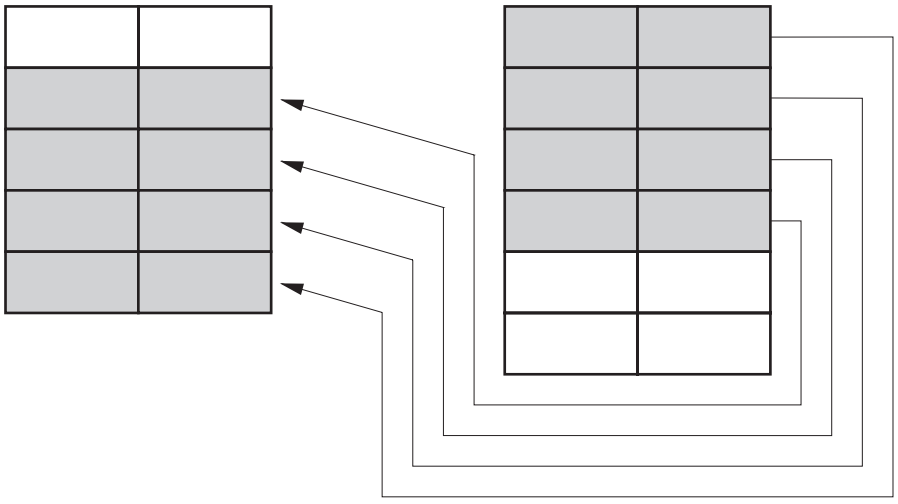
Copy part of a table into part of another table

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



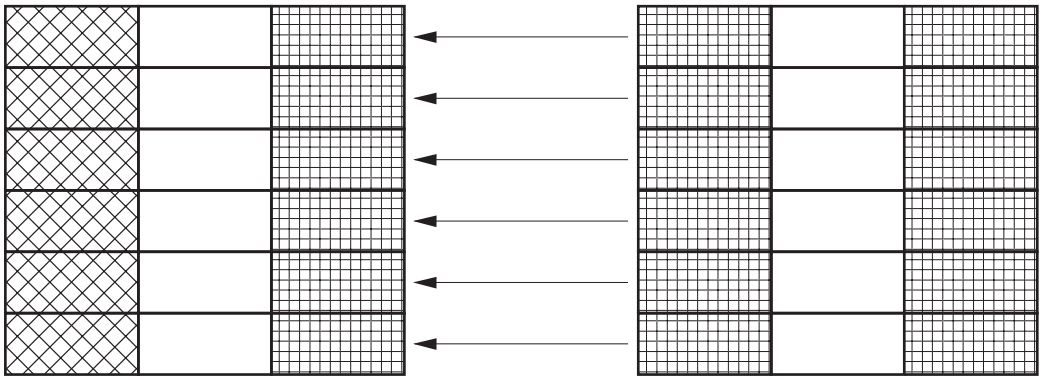
Copy part of a table into part of another table in reverse order

MOVE [PB(2,5)],2,5 = [PA(2,6)],4,1 ← Notice the inversion



Copy part of table blocks into part of another table pied with inversion

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



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

4.11 Indirect addressing of symbolic variables

- A variable or a table of symbolic variables can be referenced by a value.
- This symbolic variable addressing mode is indirect, since it is via another variable
- 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 no with 10 entries is referenced by the value 7

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

4.12 Programming examples

4.12.1 Use of the BUILD function for back and forth milling

Using radius correction, path from 1 to 6 then back from 6 to 1.

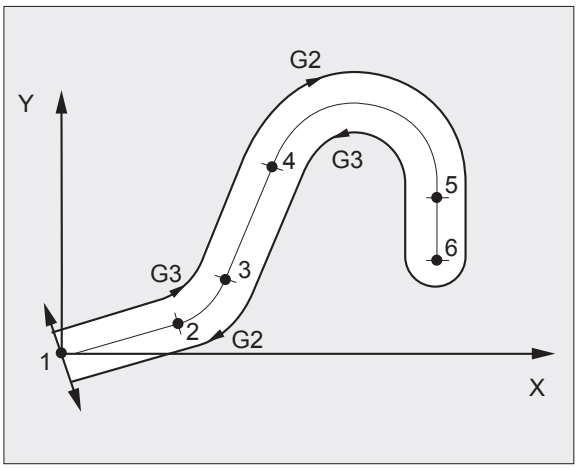


Table built by the program

	G	X	Y	I	J
1	**1**	**0**	**0**	**0**	**0**
2	**1**	**22.267**	**8.104**	**0**	**0**
3	**3**	**28.243**	**14.081**	**18.846**	**17.501**
4	**1**	**35.905**	**35.13**	**0**	**0**
5	**2**	**65**	**30**	**50**	**30**
6	**1**	**65**	**20**	**0**	**0**

```
%350
G79 N100
N10   G1 X Y
      EA20 ES EB10
      EA70
      G2 I50 J30 R15
N40   G1 Y20
N100
```

Definition (points 1 to 6)

```
BUILD [TAB(G/X/Y/I/J,NB)]N10 N40
```

Table creation

```
VAR  [I]
ENDV

G41 D1
FOR [I]=1 TO [NB] DO
G[TAB(1,I)] X[TAB(2,I)] Y[TAB(3,I)] I[TAB(4,I)] J[TAB(5,I)]
ENDF
```

Path from 1 to 6

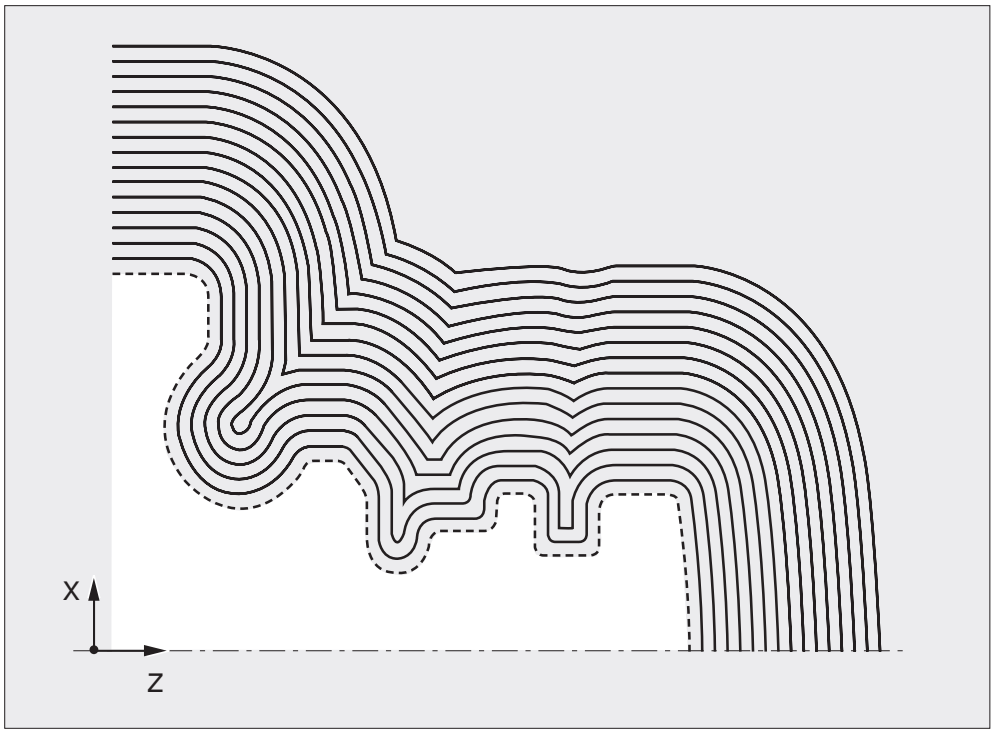
```
FOR [I]=[NB] -1 DOWNTO 1 DO
L0=[TAB(1,I+1)]
IF L0= 2 THEN L0=3
ELSE
IF L0 = 3 THEN L0 = 2
ENDI
ENDI
GL0 X[TAB(2,I)] Y[TAB(3,I)] I[TAB(4,I+1)] J[TAB(5,I+1)]
ENDF
G40 X Y
M2
```

Path back from 6 to 1

Inverting G2 and G3 for the return

4.12.2 Use of P.BUILD for turning

Execution of a profile with offset and use of the MOVE and R.OFF functions




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

FOR [R]= 30 DOWNTO 2 BY 2 DO

      VAR
            [M]=[N]-1 [P(7,M)]
      ENDV

      MOVE [P(7,M)] = [C(7,N)],2,[N]      Copy in another table
      R.OFF [P((7,M)] / + 1 / [R]         Offset profile to the right

      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                                       Profile definition
```

```
Z60
G2 X15 Z50 I15 K55
G1 X25 EB-4
Z40
G2 I30 K30 ES+
G1 EA90 X50 Z25 EB3
N110 Z10
```

4.12.3 Getting a symbolic variable or part program address =@

A symbolic variable address is read by the command **=@** (no spaces between the two characters) and stored into an E parameter, an L variable or another symbolic variable.

Syntax for reading a symbolic variable address

E81000 =@ [TAB(1,1)]

This pointer can then be used to access the symbolic variable via dynamic operators. Be careful to save the symbolic variable if it is declared in a subroutine otherwise it will be lost when exiting the subroutine.

Symbolic variables are stored internally in the NUM floating point structure. Find below an example of conversion in C to obtain an integer value:

```
typedef struct ts_NUMFL /* NUM floating point */
{
SINT32 Mant ;
SINT16 Exp ;
} s_NUMFL ;

SINT32 fl_fix(s_NUMFL *ptV)
{
return((SINT32)((s_NUMFL*)ptV->Mant) >> (31-((SINT16)(s_NUMFL*)ptV->Exp))) ;
}
```

In a similar way, it is possible to retrieve a part program address. This is particularly useful when in conjunction with the Binary file creation feature. It will then be possible to access a large amount of data stored in a file.

Syntax for reading a part program address

E81000 =@ H<prog_number> :

The address of the first block of part program number <prog_number> will be stored in E81000.

For E parameters detailed information please refer to the Flexium Programming manual M00018.

4.12.4 Creating Binary Files G76+ H<bin>

A binary program %<bin> is created from a source program %<source> using the following syntax.

Syntax

G76+ H<bin> = H<source> /X /Y /Z ...

The arguments preceded by / designate the functions of the source block to be converted to binary. These arguments are axis names (X, Y, Z, U, V, W, A, B, C).
The maximum number of arguments is 8.

The binary values (encoded on 32-bit words) are stored in the order in which the arguments are defined. The binary file is a table in which all the values of a block and the blocks themselves are contiguous.

The order and number of the arguments define the table structure.

Functions not declared as arguments are ignored (without generating an error message).

When at least one function (including the block number) is present in a block, all other functions declared as argument but missing in this block will be considered with the value they had in the previous block (zero for the first block). However, if the block is empty or includes only comments, it is ignored.

The last word in the table is identified by hex code 13001300.

Example

```
%1
G76+ H100 =H20/X/Y
E81000=@H100
```

← E81000 will point the first block value in this file (prog number)

```
%20
N10 X10
N20 X20 Y50
(empty block)
N30 Y60
N40
N50 X40 Y80
```

Program created

	Various headers	
Data addressed by E81000	% 100<CR>	
	0x1300	End of program
	10000	X value block N10
	0	Y value block N10
	20000	X value block N20
	50000	Y value block N20
	20000	X value block N30 coming from N20
	60000	Y value block N30
	20000	X value block N40 coming from N20
	60000	Y value block N40 coming from N30
	40000	X value block N50
	80000	Y value block N50
	0x 1 3 0 0 1 3 0 0	Code : End of data

No binary block is created from the empty source block.

Variant

Allocate workspace in a file by G76+ H<bin> = <space allocated>

The space allocated is a number of 32-bit words in the file. These words are set to zero.

Since the program address read by =@ is that of the first block, it is necessary to search for the first word in the data table in the OPEN of the dynamic operator. This first block follows 0x1300 after the program number.

Page deliberately empty

Summary

Page deliberately empty

5 Program stack - L variables and symbolic variables

5.1 Program stack

Each channel is allocated a stack located at the bottom of the memory.

5.1.1 Use of the stack

The stack is used to:

- Save and restore L program variables,
- Assign symbolic variables,
- Save spline curve coefficients.

5.1.2 Stack Allocation

The stack size is defined by the machine builder (parameter P58).

By default it is 20Kb per channel. It can be up to 64Kb.

5.2 Saving and restoring ► L variables

5.2.1 Push Function: ► PUSH

PUSH saves the values of the L variables in the stack.

Syntax

PUSH Lm - Ln

Lm-Ln L variables (m to n inclusive) whose values are to be saved.

Number ranges: 0-19
100-199
900-959.

Notes

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

Saving variables from different ranges of numbers with one single PUSH is not allowed:

- **PUSH** L1 - L19 **OK**
- **PUSH** L100 - L110 **OK**
- **PUSH** L1 - L110 **Not OK**

5.2.2 Pull Function: ► PULL

PULL restores the values of the L variables previously saved by a PUSH.

Syntax

PULL Lm - Ln

Lm-Ln L variables (m to n inclusive) whose values are to be saved.
Number ranges: 0-19
100-199
900-959.

Notes

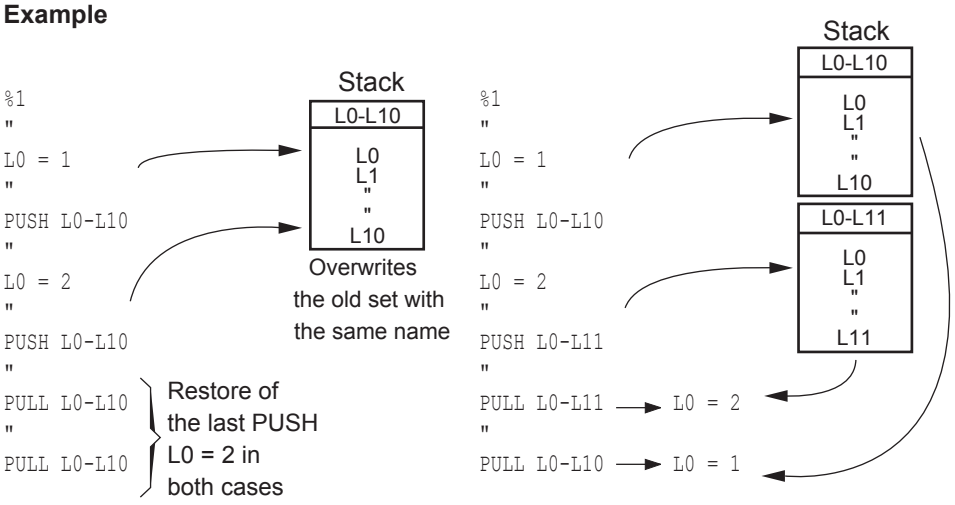
The PULL function must be the first word in a block (no sequence number).

The set of variables defined with PULL must have been previously saved by an identical PUSH.

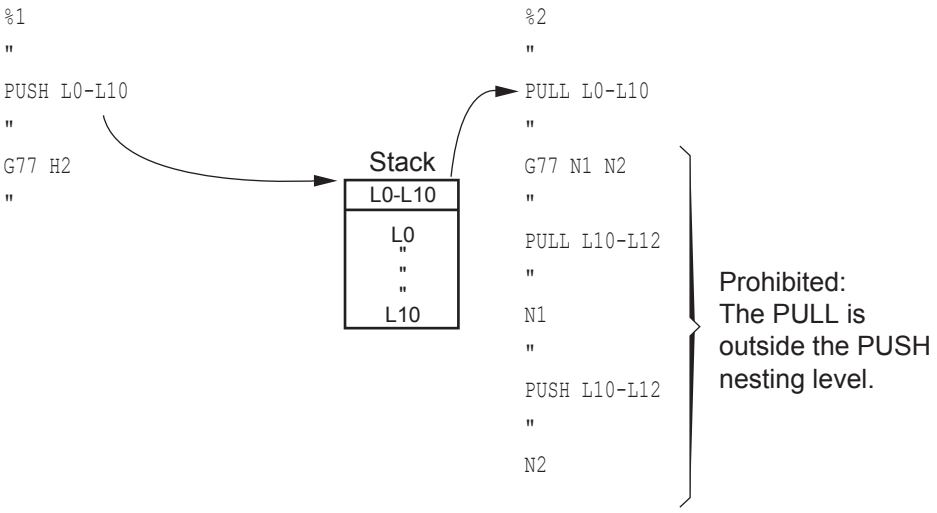
The PULL function:

- Allows for retrieving any set of previously saved variables (not necessarily the last one)
- Does not delete the values from the stack
- Variable sets saved by a PUSH can be restored several times by PULL
- As the PULL does not delete the values from the stack a stack overflow can occur after a series of PUSH-PULL operations (in this case, the system returns error message 195).

Example



A PULL from Lm to Ln restores only the variables saved in the current main program or subroutine or those from lower nesting levels (to be valid, a PULL must always be preceded by a PUSH).



5.3 Symbolic variables: ► VAR, ENDV

The system offers the possibility to create variables with symbolic names. The use of symbolic variables allows the number of variables used in parametric programming to be extended.

Symbolic variables are created on the stack.

Variables can be single variable or multi dimension arrays.

5.3.1 Symbolic variables declaration: VAR and ENDV

Symbolic variables are declared in a section between the functions VAR and ENDV.

Symbolic variables are declared between square brackets and are denoted [symb] in the syntax defined below.

VAR Beginning of the section for symbolic variable declaration

ENDV End of the section for symbolic variable declaration.

Syntax

```
VAR [symb]
[symb1]…[symbn]
[Table(2,4)]
ENDV
```

VAR Symbolic variable declaration.

[symb] Symbolic variables with up to 8 alphanumeric characters, the first of which must be alphabetic. The name is case sensitive.

[Table(i,j)] Array of i*j elements

ENDV End of symbolic variable declaration.

Notes

- The functions **VAR** and **ENDV** must be the first words in the block (no sequence number).
- There should be no sequence number between VAR and ENDV.
- VAR must be separated from the symbolic variable(s) by a space.
- Variable names are case sensitive [VAR] and [Var] are different objects.
- Assuming the first character is a letter, all ASCII characters (excluding "[,], (,)") are authorised in a variable name. However it should be noticed that only variables names made of numbers and letters can be displayed on the HMI (variable page).
- **ENDV** must be the only word in the block.
- Each variable uses on the stack 16 bytes for header plus 6 for the value itself.

5.3.2 Using symbolic variables in ISO programs

A symbolic variable declaration or use suspends preparation of the block to which the variables belong until execution of the previous block is finished (according to the same principle as variables L100 to L199). To be able to use symbolic variables during cutter compensation (G41/G42) programming M999 is required.

Symbolic variables have real values. Symbolic variables can be:

- Assigned to all the ISO programming function
- Used in parametric expressions
- Associated, where applicable, with L program variables and E external parameters.

The program can only access the symbolic variables declared in it or, if a subroutine, those from lower nesting levels.

Defining an L variable or E parameter defined by a symbolic variable

The system provides the possibility of programming addresses by an L variable or an E parameter defined by a symbolic variables.

The number of the L variable or E parameter is assigned to a symbolic variable [symb...].

Example

```
VAR
[symb1]=80000                E Parameter number
[symb2]=0                    L Variable number
ENDV

...
...
...

XE[symb1]                    X axis programming (equivalent to XE80000)
BL[symb2]                    B axis programming (equivalent to BL0)
```

Notes

If X[symb2] or XL0 or XL[symb2] is programmed, the unit for the value given by the symbolic variable or the L variable is the programming unit (mm, inches or degree).

If XE80000 or XE[symb1] is programmed, the unit for the value is the internal system unit

Other Uses of Symbolic Variables

Symbolic variables can be used:

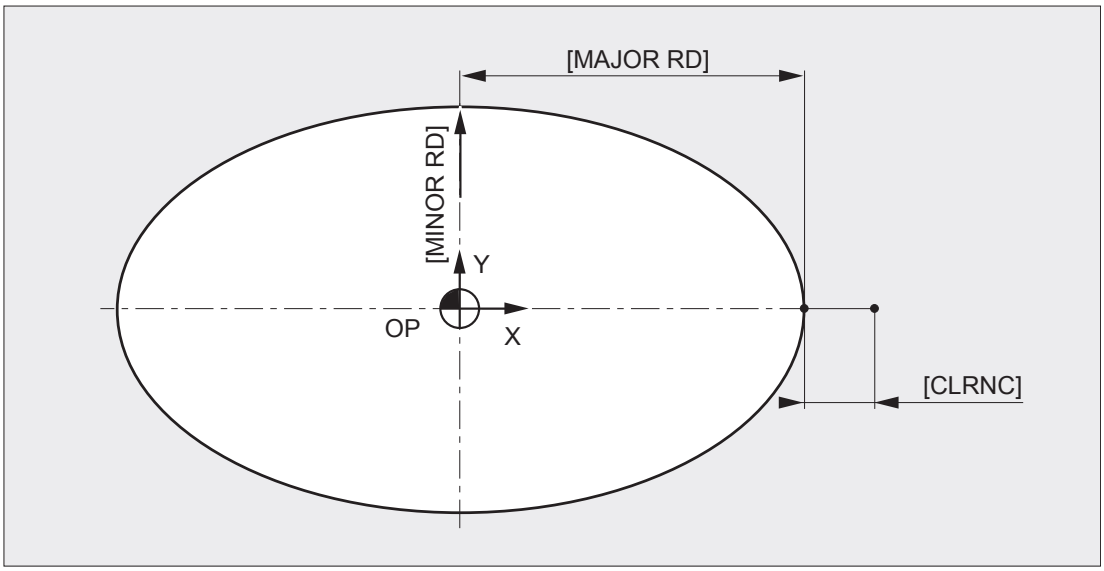
- To create tables or arrays
- With structured branch and loop commands.

Example

Use of symbolic variables for machining an ellipse

```
%35
M999

VAR
[SPSPD] = 1000 [DIRN] = 3 [SPDRNG] = 40
[FDRT] = 200 [CLRNC] = 120 [DPTHZ] = 15
[MAJOR RD] = 100 [MINOR RD] = 50 [INCR] = 10
[STRT ANG] = 0 [END ANG] = 360
[COS] [SIN]
ENDV

$ MACHINING
N10 G0 G52 Z0
N20 T1 D1 M06 (CUTTER R=5)
N30 G97 S[SPSPD] M[DIRN] M[SPDRNG]
N40 G0 X[CLRNC] Y0
N50 Z-[DPTHZ]
N60 G01 X[MAJOR RD] F[FDRT]
N70 $ MACHINING AN ELLIPSE
[COS] = [MAJOR RD] * C[STRT ANG]
[SIN] = [MINOR RD] * S[STRT ANG]
X[COS] Y[SIN]
[STRT ANG] = [STRT ANG] + [INCR]
G79 [STRT ANG] = < [END ANG] N70 + 1
N80 X[CLRNC]
$ END OF MACHINING

N90 G0 G52 Z0 M05
N100 M02
```

View of machining



5.3.3 Getting the variables in the stack of another channel: ► VAR H.. N.. N..

The function VAR H..N..N.. is used to read the symbolic variables declared in a program from an other channel.

Syntax

VAR H..:sN..N..+i+j

VAR	Acquisition of variables declared in a program from another channel.
H..	Number of the program or subroutine where the symbolic variables are to be fetched.
:s	Number «s» of the memory zone where program H.. is searched (s can have any value between 0 and 3 after the character :). This argument :s is optional.
N.. N..	Numbers of the first and last block defining the limit between which the symbolic variables are located. (see details below on N.. N..) .
+i +j	Offset with respect to blocks N.. and N.. +i with respect to the first block, +j with respect to the last block.

Notes

When a variable with the same name as a variable declared by VAR N..N.. has already been declared in the calling program or subroutine, the previous variable is destroyed.

Details on blocks N.. N..

If program number H.. is not specified, the limits from N.. to N.. are searched for in the current program or subroutine.

If the second block number N.. is not specified, only the variables in block N.. are read.

Example

```
%1
N10 ...
N..
VAR H2 N50 N60 L0=[I] L1=[T(1,3)]
N..
```

```
%2
N10
N..
VAR
N50 [I] = 3 [J] = 4
[T(2,3)] = 3, 4, 5, 6, 7 , 8
N60
ENDV
```

Results

L0 = 3
L1 = 7

5.3.4 Deleting symbolic variables from the stack

The variables are not deleted until the block preceding the deletion command has been executed.

5.3.4.1 Automatic deletion of symbolic variables

The return to the upper level program deletes all the variables declared in a subroutine and releases the space they occupied in the stack.

The end of program command (M02) or an operator reset deletes all the variables and reinitialises the stack.

5.3.4.2 Programmed deletion of variables: ► DELETE

Syntax

DELETE *[symb1]/[symb2]…

Deletion of symbolic variables from the program or subroutine.

Deletes the global variables from the stack.

* When the variables names are preceded by the character *, the variables to be deleted are searched for in the entire stack. When the variable names are not preceded by the character *, the search for the variables to be deleted is limited to the variables declared in the main program or subroutine.

[symb1] / [symb2]… Variables to be deleted.

Notes

The DELETE function:

- Must be the first word in the block (no sequence number),
- Must be followed by at least one space but there must be no spaces in the list of variables,
- Is followed by the list of variables and arrays to be deleted; the variables are separated by the character /.

Only the first four letters of the keywords are significant for the system.

For instance, DELE is equivalent to DELETE.

Example

`DELE [IX]/[TAB1]/[PROF1]`	Deletion of variables
`DELE *[IZ]/[TAB3]/[PROF2]`	Deletion of variables from the stack

Page deliberately empty

Summary

Page deliberately empty

6 Subroutine call by G Functions

6.1 General

The system gives the possibility to call a subroutine by a G function.
The machining cycles (G81 and up) are based on this functionality.

G100 to G255 will respectively call subroutines %10100 to %10255 allowing special cycles to be created.

Functions G100 to G198 can be freely used as they will not be managed by NUM.

Syntax

N.. Gxxx specific parameters	
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

Functions called by G code are modal.

It is possible to make a function non modal by including a G80 at the end of the code.

Cancellation

The function if they are modal will be cancelled by G80 (this function does not call a subroutine).

Notes

Since functions G199 to G255 may be used for NUM applications, it is recommended to use only functions G100 to G198.

Due to the part program zones priorities, a subroutine %10xxx defined by the User or the OEM will supersede the NUM defined function having the same number.

G199 is a particular function dedicated to automatic homing. It is defined by NUM but can be partly adapted by the OEM according to the machine configuration. Additional information about automatic homing can be found in the Flexium Comissioning manual M00010.

6.2 List of G functions calling a subroutine

Canned cycles	G06, G31, G33, G38, G45, G46, G48, G49, G63, G64, G65, G66, G81-G89
Reserved for NUM	G199, G200 to G255, G104, G159 to G199 (see M00018EN Flexium Programming manual)

6.3 Particularities

- A Gxxx function programmed without arguments will be ignored by the system.
- A subroutine called by 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.

The arguments are interpreted by the %10xxx subroutine called.

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

6.4 Theory

6.4.1 Preliminary

The following functionalities are used in %10xxx subroutines. Their knowledge is necessary to understand and master the theory.

- Functions G997, G998 and G999
- Program variables L900 to L925 and L926 to L951
- External parameters E
- Symbolic variables.

This manual, as well as the Flexium Programming manual M00018, provides additional information on these items.

6.4.2 Operation

A call to a subroutine by a G function automatically sets the function G999 (execution suspended and blocks concatenation forced).

The subroutine called will interpret the arguments being passed with the G code, it must therefore have visibility into the program context and all the functions programmed in the calling block.

The subroutine will use the arguments to Set/Reset axes motion and/or other function execution. Once the interpretation phase is completed, the part program execution will be resumed by G998 and/or G997.

If G80 is not programmed in the subroutine, the call remains modal and %10xxx will be executed for each consecutive part program block until G80 or M02 or reset.

It is possible to inhibit the display of the subroutine internal blocks during execution by inserting the character «:» just after the subroutine number. In this case, only the subroutine call block is displayed.

Example

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

During execution of %10108 and %118 the display will only show N150 G108 and the attached arguments.

6.5 Subroutine example

The cycle below is given only for guidance.

It defines the function G182 (subroutine %10182) which will trigger the execution of several drilling or punching operations «P» distributed on a circle with radius «R» centred on XY (G17).

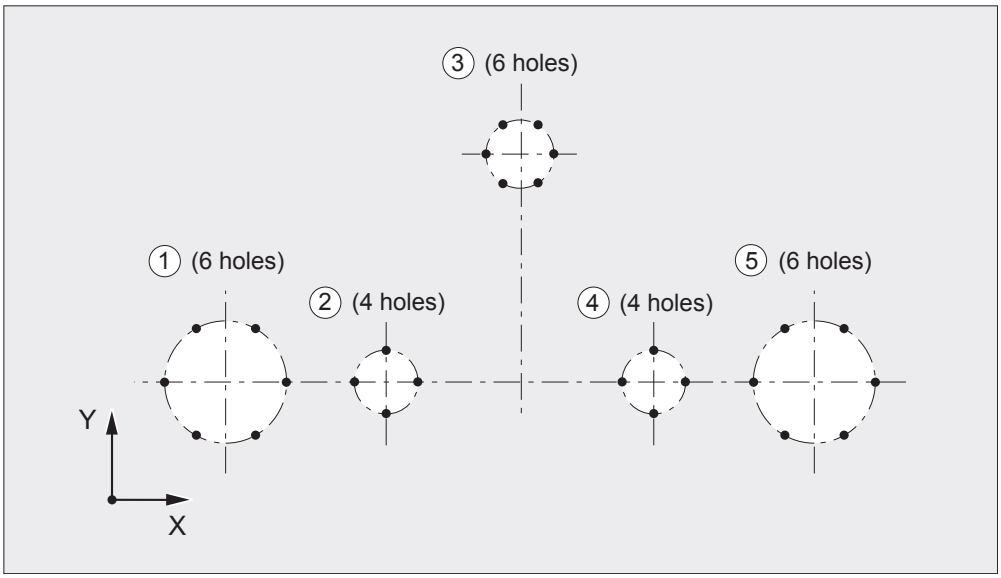
6.5.1 Cycle syntax and parameters

Syntax

N.. G182 X.. Y.. ER.. Z.. P.. R.. F..

N.. G182	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.

6.5.2 Graphical representation



6.5.3 Main machining program

```
%20
N10 G0 G52 Z0
N20 T1 D1 M06 (DRILL)
N30 S1000 M40 M03
N40 X0 Y0 Z5
N50 G182 X50 Y50 ER2 Z-10 P6 R20 F90        Cycle on circle 1
N60 X100 Z-5 P4 R10                         Circle 2
N70 X150 Y150 Z-15 P6 R25                   Circle 3
N80 X200 Y50 Z-5 P4 R10                     Circle 4
N90 X250 Z-10 P6 R20                        Circle 5
N100 G80 G0 G52 M05                         Cycle cancelled
N110 M02
```

6.5.4 Cycle subroutine

```
%10182: (Equally spaced holes on the circle)

VAR
  [G0/1] [RETURN] [FEED] [G94/5]
ENDV

[G0/1]=3 * [..BG03] [G0/1]=2 * [..BG02] + [G0/1]    Save status G0, G1, G2 or G3
[G0/1]=1 * [..BG01] + [G0/1]

[FEED]=[.RF]                                        Save status G94 or G95 and feedrate
[G94/5]= 94 [.BG94]
[G94/5]= 95 [.BG95] + [G94/5]

PUSH L0 - L7                                        Save local variables
```

Analysis: Are P and R programmed in the calling block

```
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
```

Execution:

```
BCLR [.IBX(3)]                          Z axis motion disabled

L0= [.IRX(1)]                           X dimension
L1= [.IRX(2)]                           Y dimension
L2= L0 + [.IRP(3)]                      Cycle start dimension (at 3 o'clock)
XL2                                     X position modified

G997                                    X and Y Movements activated

FOR L4= 1 TO L100
  L5= L4 * 360 / L100                   Current angle
  L6= CL5 * [.IRP(3)] + L0              New X dimension
  L7= SL5 * [.IRP(3)] + L1              New Y dimension

  G0 XL6 YL7                            Positioning for the next hole

  IF [.IBE0(6)]= 1 THEN                 Take programmed federate if any
    G94 FL905
  ENDI

  G1 Z L925                             Go to bottom of hole
  G4 F1                                 Dwell at the bottom
  G0 Z [RETURN]                         Return in Z

ENDF                                    End of cycle

G [G94/5] F [FEED]                      Restore initial conditions
PULL L0 - L7
G79 N9999                               End of cycle

N100 $ P AND R ARE MANDATORY IN G182
E.500                                   Error number 500
N101 $ P MUST BE A POSITIVE INTEGER IN G182
E.501                                   Error number 501

N9999                                   End of module
```

Notice the block in bold : **BCLR [.IBX(3)]**. It is intended to inhibit the move in Z when execution is released via G997. The Z motion should not be simultaneous to XY displacement.

6.6 Custom cycle example 2

The cycle below is given only for guidance.

It defines the function G177 which is used to execute a profile by back and forth passes with the possibility of radius correction if required.

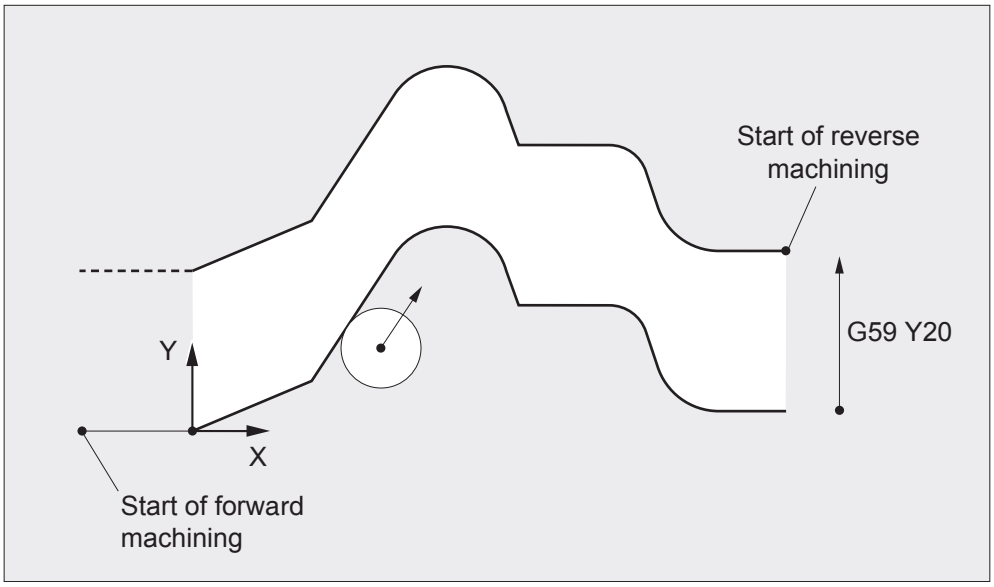
6.6.1 Cycle syntax and parameters

Syntax

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.

6.6.2 Graphical representation



Main machining program

```
%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 ER4
N190 G0 G52 Z0
N200 G0 X-100 M5
N210 M02
```

Profile definition

Forward machining
Trajectory offset
Reverse machining

6.6.3 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                                  Force ER40 if necessary
ENDI

[M998]= [.BM999]-[.BM997]+998 M998          Save status of M997, M998 and M999

[N1]=L913 [N2]=L914

IF L913>L914 THEN
    [N1]=L914 [N2]=L913                     Reverse the direction if necessary
ENDI

[H]=[.RXH]-1 [H]=[.IRH(H)]                  Retrieve the calling program number

IF L913>L914 THEN                           Profile to execute in reverse order
    P.BUILD [TAB(7,NB)] H[H] N[N1] N[N2]    Store the profile in a table

    G997                                    Execution

    [PLANT]=[.BG20]+[.BG21]                 Detects turning

    [diam]=E70007                           Programming by diameter?

    IF [PLANT]<>0 THEN                      Turning?
        IF [.BG20]=1 THEN
            @Y=X @X=Z @J=I @I=K             Profile is defined in XY ➔ ZX in G20
        ELSE
            @Y=Y @X=X @J=J @I=I
        ENDI

        IF [diam]=1 THEN
            [multi]=2                       For turning by diameter (x2)
        ENDI

    ELSE                                    Milling
        E11005=0

        IF [.BG17]=1 THEN                   Addresses equivalence according to plane
            @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]           Retrieves the end point
    [axis2] =[TAB(2,NB)]

    G1 GL943 @Y [axis1] @X [axis2]          Moves to end point
    G998                                    GO !
```

```
    FOR [NBLOC]=[NB] DOWNTO 2 DO
        [G]=[TAB(1,NBLOC)]                  Interpolation type
        IF [G]=0 THEN
            [G]=1                           TAB(1,NBLOC)] = 0 ➔ G1
        ELSE
            IF [G]=-1 THEN                  TAB(1,NBLOC)] = -1 ➔ G3
                [G]=3
            ELSE
                [G]=2                       TAB(1,NBLOC)] = 1 ➔ G2
            ENDI
        ENDI

        [axis1]=[TAB(7,NBLOC)]* [multi]
        [axis2]=[TAB(6,NBLOC)]
        [centre1]=[TAB(5,NBLOC)]* [multi]   Calculates coordinates
        [centre2]=[TAB(4,NBLOC)]

                                            Go to position

        G[G] @Y[axis1] @X [axis2] @J[centre1] @I[centre2]
    ENDF                                    Loop for all table

ELSE                                        Profile defined in direct order
    G1 GL943 G77 H[H] N[N1] N[N2]           Direct execution of the forward profile
ENDI

N9900 G80                                   Cancels modality
E11005=[diam]                               Restore radius or diameter prog status
```

6.7 Decrypting a standard canned cycle

Standard peck drilling cycle called by function G83.
As for all other canned cycles, notice that the subroutine %10083 will call subroutine %10080 to analyse all the cycles created by NUM (see subroutine %10080 at the end of this paragraph).

6.7.1 Syntax of the G83 cycle

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

Cycle description

For more detailed descriptions of this cycle please refer to the Flexium Programming manual M00018.

Cycle subroutine

```
%10083: (peck drilling cycle)
VAR
    [M3/4][M998][G90/1][G0/1][RF][clrance]=1 [diam]
    [IX][IY][IZ][LZ][I][dimnsion][depth][Gplan][E]
ENDV

[diam]=E11005 E11005=0

G77 H10080                                    Call analysis module

IF [..BG80]=1 AND [.IBP(1)]=0 THEN E.889      Check syntax: P present if previous block with G80
 ENDI

IF [.IBP(1)]=1 THEN 'P=[.IRP(1)]
    IF [.IBP(2)]=0 THEN 'Q='P                 Load P and Q if programmed
    ENDI
ENDI

IF [.IBP(2)]=1 THEN 'Q=[.IRP(2)]
ENDI

IF [.BG70]=1 THEN [clrance]=[clrance]/25.4 Convert clearance if in INCHES
 ENDI

IF [.RDX]<0 THEN[clrance]=-[clrance]          Clearance direction according to tool orientation
ENDI

IF [.BG95]=1 THEN G0
ELSE F5000                                    Prepare positioning of the axes
ENDI

IF [I]<>0 THEN
    G9 G998
    [dimnsion] = 'ER [I]=[IZ]+10 G0 G77 H10080 N[I] N[I]

    IF 'P < 0 THEN 'P = -'P                   Assign correct sign to P and Q if programmed
    ENDI
    IF 'Q < 0 THEN 'Q = -'Q
    ENDI

    IF 'P > 0 THEN
        'I=0 'L='ER-L[LZ]

        IF 'L = 0 THEN E.891                  Error if retraction plane = hole bottom
        ENDI

        IF 'L < 0 THEN 'L=-'L
        ENDI

        IF 'Q = 0 OR 'Q > 'P THEN 'Q = 'P
        ENDI

    '    I = 'Q-'P * 'I/'L + 'P + 'I          Calculate depth of first pass
        'J = 'L-'I

        IF 'J <= 0 THEN                       Go to bottom of hole
            [dimnsion]=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

        [dimnsion]=[depth]                          Execute first pass
        G9 G1 F[RF] G77 H10080 N[I] N[I]

        IF [.IBE1(6)]=1 THEN G4 FL931               Optional dwell
        ENDI

        [dimnsion]='ER                              Retract —> ER
        G0 G77 H10080 N[I] N[I]

        REPEAT
            [dimnsion]=[clrance]+[depth]            Calculate approach dimension
            G0 G77 H10080 N[I] N[I]

            'I = 'Q-'P * 'I/'L + 'P + 'I 'J = 'L-'I Calculate and execute next passes
            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

            [dimnsion]=[depth]
            G9 G1 F[RF] G77 H10080 N[I] N[I]        Execute next pass

            IF [.IBE1(6)]=1 THEN G4 FL931           Optional dwell
            ENDI

            [dimnsion]='ER
            G0 G77 H10080 N[I] N[I]                 Retract to top
        UNTIL 'I = 'L                               End of loop
    ENDI

    [dimnsion]=L[LZ]                                Go to bottom of hole
    G1 F[RF] G77 H10080 N[I] N[I]

ENDI

N100                                                Dwell specified by EF
IF [.IBE1(6)]=1 THEN G4 FL931
ENDI

[dimnsion] = 'ER [I]=[IZ]+10                        Retract to ER
G0 G77 H10080 N[I] N[I]                             Go
G997 G9 M[M998]
G[G90/1] G[G0/1] F[RF]                              Restore status
E11005=[diam]
```

6.7.1.1 Subroutine %10080 common to all canned cycles

```
%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)]

[M3/4]=3*[.BM03] [M3/4]=4*[.BM04]+[M3/4]    Read spindle rotation direction and M block sequencing

[M998]=[.BM999]-[.BM997]+998
M997                                            Force M997 status

[IZ] = [.RDX]                                   Read tool axis number
IF [IZ] < 0 THEN [IZ] = -[IZ]
ENDI

[E]=[.BG21]+[.BG22] G79 [E]>0 N85               G21, G22 prohibited during a machining cycle

IF [.BG20]=1 THEN [Gplan]=20                    Plane and tool axis compatible?

ELSE [Gplan]=[.BG19]-[.BG17]+18                 G17,18,19 detection
    [E]=[Gplan]+[IZ] G79 [E]<>20 N83
ENDI

[LZ]=922+[IZ] G79 N[IZ]                         Read axis ranks and station in tool axis L900
N1 [IX]=5 [IY]=6 G79 N3+1
N2 [IX]=4 [IY]=6 G79 N3+1
N3 [IX]=4 [IY]=5

IF [.IBX2(IX)] = 0 THEN [IX] = [IX]-3           Choose primary or secondary axis on
ENDI                                            the axis perpendicular to the tool axis

IF [.IBX2(IY)] = 0 THEN [IY] = [IY]-3
ENDI

IF [.IBX2(IZ)] = 0 THEN [IZ] = [IZ]+3 [LZ]=[LZ]-3      and on tool axis
    IF [.IBX2(IZ)] = 0 THEN E.880
    ENDI
ENDI

IF [.IBE1(18)] = 0 THEN 'ER = [..IRX(IZ)]       Store the last Z dimension in 'ER
ELSE [E]=[IZ]-1* 1000+70007                     Return if ER was not programmed

    IF E[E]=1 THEN 'ER='ER/2                    If programming is by diameter, correct 'ER
    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)]          If programmed, store the new hole bottom dimension
ENDI

IF [.IBX(IZ)] = 1 THEN L[LZ] = [.IRX(IZ)]
ENDI
```

```
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
```

Test whether the orientation is consistent with the machining direction

Interpolation type, Dwell, Feedrate

```
[G0/1]=3 * [.BG03] [G0/1]=2*[.BG02]+[.BG01]+[G0/1]
[G90/1]=90+[.BG91] [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
```

```
[I]=[.IBX(IZ)]+[.IBX(IX)]+[.IBX(IY)] G90
BCLR [.IBX(IZ)]
```

Store presence then disable the tool axis

Test spindle rotation if axis is programmed

```
IF [I]<>0 THEN
    [E]=[M3/4]*
    [.RS] G79 [E]=0 N81
ENDI

G79 N800
```

Error messages

```
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
```

Page deliberately empty

Summary

Page deliberately empty

7 Polynomial interpolation

7.1 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 a dimensionless parameter varying from 0 at the beginning of the block to 1 at the end.

There are two types of polynomial interpolation:

- Segmented polynomial interpolation
- Smooth polynomial interpolation.

7.1.1 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 10ms 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.

Note

At least one of the options Smooth Polynomial Interpolation or Spline Curve should be enabled.

In the absence of the Smooth polynomial interpolation option Segmented polynomial interpolation will be carried out regardless of the I argument in the syntax.

Otherwise, programming of argument I.. in the syntax is ignored and segmented interpolation is carried out if Spline interpolation option is enabled.

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

7.2 Segmented polynomial interpolation

7.2.1 Syntax

N.. G01 X../Cx1/../Cxn Y../Cy1/../Cyn Z../Cz1/../Czn ...

G01	Linear and polynomial interpolation function.
X..	X axis end coordinate.
/Cx1 /… /Cxn	First, second, n degree etc. coefficients for X.
Y.. /Cy1 /… /Cyn	Y axis end coordinate and coefficients.
Z.. /Cz1 /… /Czn	Z axis end coordinate and coefficients.
…	Other axes coordinates and coefficients.

7.2.2 Notes on the axes and coefficients programmed

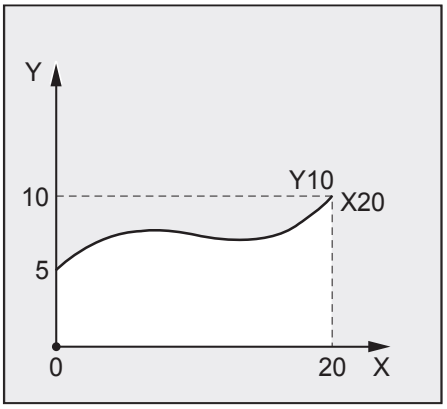
The sum of coefficients with degrees above zero must be equal to the relative movement of the axes in the block (no messages are issued).

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. No error reported for Y coefficients failing to sum.

Limit on the Number of Coefficients

For each block, the coefficients are stored in the program stack with 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.
In case of stack overflow, the system will return error message #6.

Example

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

Axis 0 : 5 Entries : (@,nb) + 4 coefficients
Axis 1 : 4 Entries : (@,nb) + 3 coefficients
Axis 2 : 6 Entries : (@,nb) + 5 coefficients

Total 15 Entries

7.2.3 Geometric transformations

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

- Programmed offset (G59),
- Mirroring (G51 ...),
- Scaling factor (G74),
- Angular offset (ED..).

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
```

7.2.4 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 can result in undesirable material removal.

7.2.5 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 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.

7.3 Smooth polynomial interpolation

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

It is the presence of argument I.. in the syntax that differentiates smooth polynomial interpolation from segmented polynomial interpolation.

7.3.1 Syntax

N.. G01 X../Cx1/../Cxn Y../Cy1/../Cyn Z../Cz1/../Czn ... I..

G01	Linear and polynomial interpolation function.
X..	X axis end coordinate.
/Cx1 /… /Cxn	First, second, n degree etc. coefficients for X.
Y.. /Cy1 /… /Cyn	Y axis end coordinate and coefficients.
Z.. /Cz1 /… /Czn	Z axis end coordinate and coefficients.
…	Other axes coordinates and coefficients.
I…	**Curve length.**

7.3.2 Notes

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 value of parameter I.. can also be used to apply the programmed feed rate to a single axis instead of to the path. (See I.. values below that contain X increments as the curve length).

Application of the feed rate to the X axis

```
...
G01 X10 Y10 F1000
X20 Y25/30/ I10           Interpolation linear on X and 2nd degree Y
X35 Y50/25/ I15           Linear on X and Y (second degree on Y = 0)
```

7.3.3 Restrictions on smooth polynomial interpolation

The following restrictions apply to smooth polynomial interpolation:

- Tool compensation (G41, G42, G29, etc.) is prohibited
- Backoff on the path is impossible
- Modulo rotary axes are not concerned
- The maximum degree of the polynomial is 5.

Page deliberately empty

Summary

Page deliberately empty

8 Coordinate conversions

8.1 General

Coordinate conversions are made performed a square matrix.

This function is used in particular for machining in an inclined plane.

	X	Y	Z
X’	K_{xx}	K_{yx}	K_{zx}
Y’	K_{xy}	K_{yy}	K_{zy}
Z’	K_{xz}	K_{yz}	K_{zz}

The movement conversion X' = K * X is made downstream of the interpolators by the K matrix illustrated above.

However, the travels are checked and the speed and acceleration are limited upstream, during block preparation.

8.2 Using the coordinate conversion matrix ► G24

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 normalised to 1).

8.2.1 Syntax

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

G24+	Coordinate conversion matrix enable.
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.

8.2.2 Cancellation

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

- **G24-** no argument required
- System restart.

8.2.3 Properties of the function

The conversion arguments are modal. After cancellation by G24-, it is possible to reprogram G24+ alone if the arguments are unchanged.
The conversion defined by function G24+ is applied immediately and **remains valid after a reset**.

8.2.4 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 the axes (for instance to retract a broken tool mounted on a tilted head).

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

Even though the function G24 main usage is ‘Machining in an inclined plane’, it can be used for other applications for example non orthogonal axes compensation).

In this case, the values of the conversion matrix coefficients must be between -8 and +8 (if not, the system generates error message 24).

External parameter E11017 (read-only) indicates whether the conversion matrix function is enabled or inhibited:

- 1: function enabled (G24+),
- 0: function inhibited (G24-).

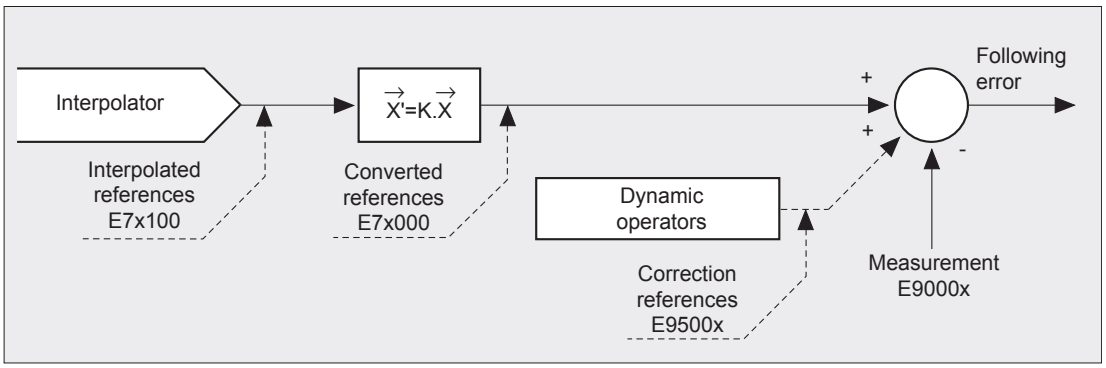
8.2.5 Error numbers and messages

- Error 2: No + or - sign after function G24.
- Error 14: Option not enabled.
- Error 24: Error in the declaration:
  - 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 in G52. prohibited.

8.3 Application of coordinate conversion

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

8.3.1 Schematic representation



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

8.3.2 Restrictions and conditions of use

The transfer (with or without conversion) of the interpolated references (E7x100) into the axes references, E7x000, is carried out only for the axes declared in machine parameters P2 or P3. (possibly modified by parameter E9100x)
(Information to take this into account in dynamic operators applications with dummy axes)

The dynamic operators that calculate reference corrections (E9500x) 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.
- Re-enable the conversion matrix (G24+ ...).

The other declarations that require the conversion matrix disabled are:

- Activation and deactivation of inter-axis 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.

For E parameters detailed informations please refer to the Flexium Programming manual M00018.

8.4 Example of application

This example of inclined plane activation 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

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

[X]=[.IRX(1)]   [V]=[..IRX(1)]   X[V]   Read the inclined plane pivot point and reset
[Y]=[.IRX(2)]   [V]=[..IRX(2)]   Y[V]   the dimensions of the previous block in X, Y and Z
[Z]=[.IRX(3)]   [V]=[..IRX(3)]   Z[V]   Modify as required [X] [Y] [Z] to compensate for head rotation

BCLR [.IBX(1)]/[.IBX(2)]/[.IBX(3)]/[.IBX(7)]/[.IBX(8)]/[.IBX(9)] Make sure there is no movement

[K11]=C[C]*C[B]                        Calculate the matrix coefficients
[K12]=S[C]*C[A] [K12]=C[C]*S[B]*S[A]-[K12]
[K13]=S[C]*S[A] [K13]=C[C]*S[B]*C[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]
                                       Disable the previous matrix if any and enable the new
G24-
G24+ X[X] Y[Y] Z[Z] P[K11]/[K12]/[K13] Q[K21]/[K22]/[K23] R[K31]/[K32]/[K33]
G997 G80                               GO !

Call to the Inclined Plane Enable Function

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

Page deliberately empty

Page deliberately empty

Summary

Page deliberately empty

9 RTCP function

9.1 General

The RTCP function (Rotating around Tool Centre Point) is used to continuously control the machine movements with respect to the part regardless of the tool orientation.

The machining program contains the Cartesian coordinates of the tool tip or point of contact of the part reference system.

Before using this function, it is necessary for the OEM to adapt it to the machine geometry. Please refer to the machine manual.

9.1.1 Control of rotary axes

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

- The rotary coordinates are programmed and associated with the Cartesian coordinates in the machining program, in which case the mode is «programmed RTCP» mode.
- The rotary coordinates are not programmed and the rotary axes are controlled manually by hand-wheels or jogs. This mode is called «N/M AUTO» N out of M axes are controlled manually. (See the N/M AUTO function in this manual)

9.1.2 Operation on the axes

The processing performed on the axes is to:

- Continuously (depending on the travel of the rotary axes) correct the references of the Cartesian axes (X, Y, Z) in order to maintain the tool contact point on the programmed path. This correction is carried out in both programmed RTCP and N/M AUTO modes.
- Make sure the dimensions of the Cartesian axes (X, Y, Z) corrected by the RTCP function remain within the machine travel limits.
  This check is made only in programmed RTCP mode.
- Analyse over-speed and over-acceleration caused on the Cartesian axes (X, Y, Z) by the rotary axes movements compensation in order 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.

9.2 Programming the RTCP function ► G26

9.2.1 General

The RTCP function in a part program complies with the programming syntax given below. This syntax contains the RTCP function enabling code followed by the arguments defining the rotary axes (twist head and/or turntable axes) causing movement on the Cartesian axes and describing the machine movements.

9.2.2 Syntax

G26+ Arguments specifying the rotary axes

G26+ RTCP function enabled.

The arguments specifying the rotary axes are specific to the machine (see below description of movements).

9.2.3 Cancellation

The RTCP function (G26+) is cancelled by:

- G26- (without arguments)
- A reset.

9.2.4 Programming errors

An error during the RTCP activation results in error message 16 with the following reasons:

- 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 channel 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.
- RTCP option not present.

9.2.4.1 Other Errors

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

- Change of tool correction with the RTCP mode enabled (error 16).
- When the RTCP mode is enabled, homing from the keyboard is inhibited and homing by the axis jogs generates error message 24.
- Rotary axes not homed, the system generates error message 159.

9.2.5 Notes

The RTCP function can be enabled with Cartesian and rotary axes in any position. This automatically redefines the new position of the Cartesian axes in the program reference system.

External parameter E11018 (read-only) indicates whether the RTCP function is enabled or inhibited:

- 1 : function enabled (G26+)
- 0 : function inhibited (G26-).

RTCP function can be activated on any channel however as the parameters are different each channel will need a different calling function.

9.3 Structure description

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

- Twist: the movements are described from the tool centre outward toward the machine bed.
- Turntable: 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.

This description is given as a reference. As a general rule the programmer does not have to deal with the following parameters. They will be defined by the machine builder and are entered in a specific cycle via the set up tools provided. The programmer just needs to call the specific cycle with the machining parameters. The calling syntax is given by the machine builder.

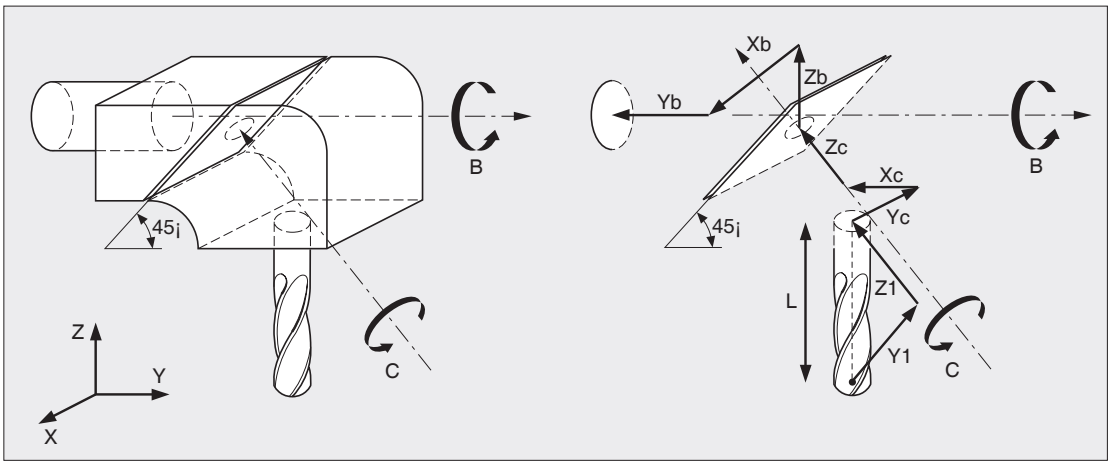
9.3.1 Twist Heads configuration

Each articulation is described by an axis of rotation and a translation vector.
The translation vector is defined from the zero position of the axis of rotation.
The axis of rotation is declared by TA, TB or TC followed the three associated translations each 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 unity 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+).

9.3.1.1 Representation of a twist head configuration



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
```

9.3.2 Turntables configuration

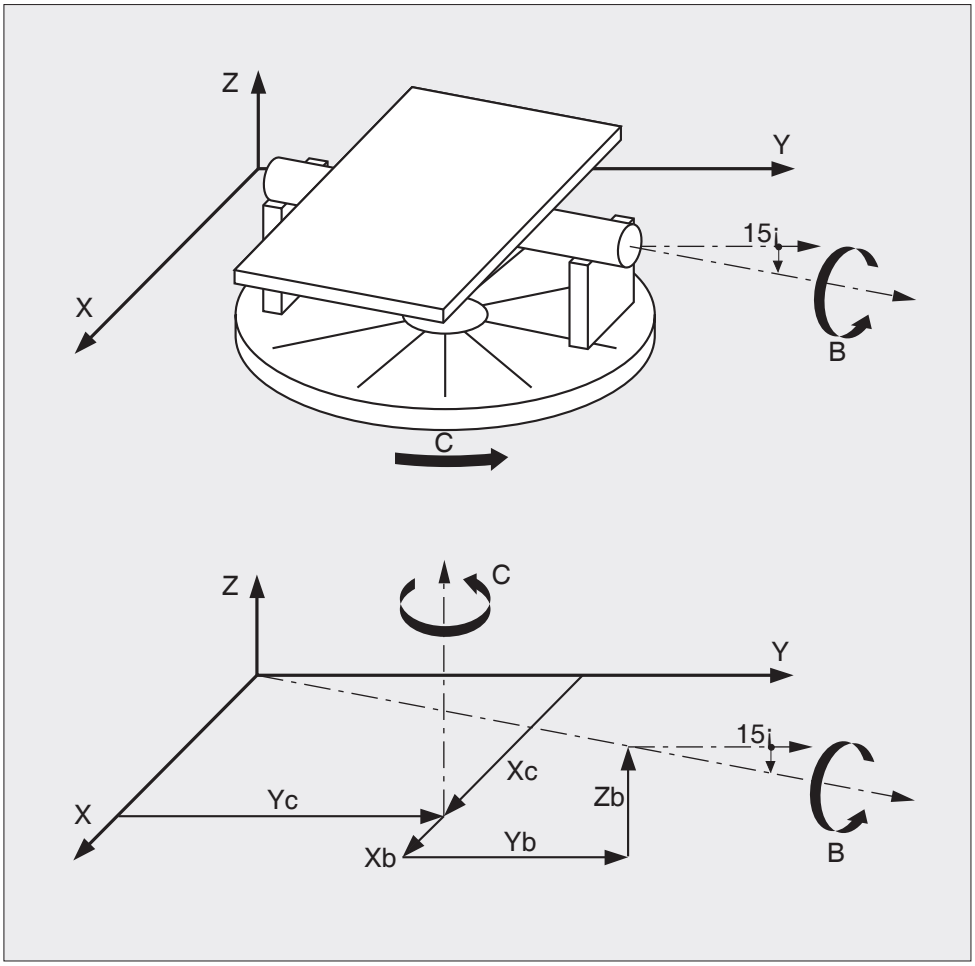
Each articulation is described by an axis of rotation and a translation vector.

The first translation vector defines the centre of the first table (table resting on the machine bed) in absolute coordinates. The following table(s) are defined in reference to the zero position of their carrier.

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

If an articulation rests on an inclined plane, this plane is declared 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.

9.3.2.1 Representation of a turntable configuration



Programming

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

9.4 Other processing related to the RTCP function

Other geometric processing that can be associated with the RTCP function:
- Part on inclined plane
- Table off-centering
- Tool length correction
- Tool wear offset
- Tool correction in space.

9.4.1 Part on inclined plane

When the part to be machined rests on an inclined plane, this plane is specified by:

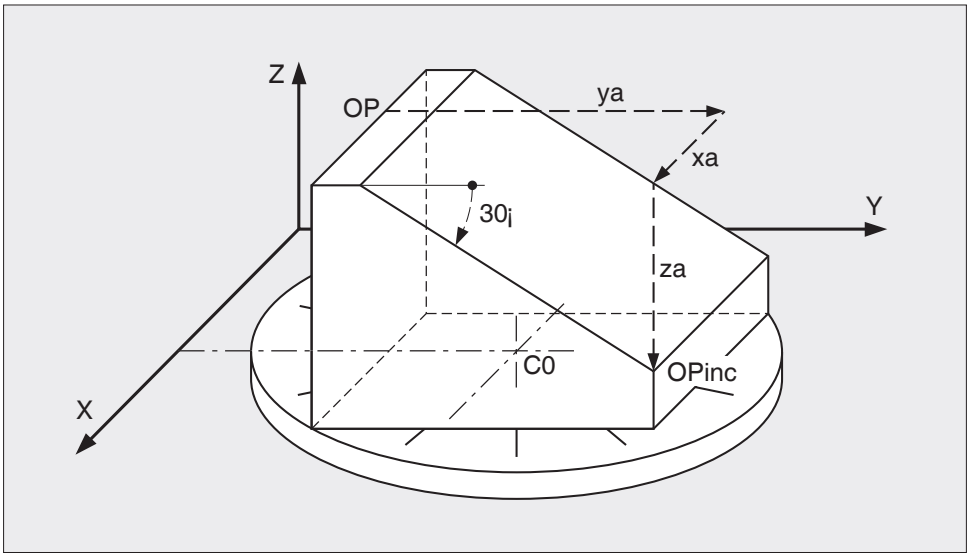
- A pivot point declared in the programming syntax by PD followed by the three components of a translation vector.

- 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 program origin and the pivot point becomes the new program origin.

Example

Structure representation on an inclined plan



OP: Old program origin
OPinc: New program origin

Programming

```
[ca]=cos-30° [sa]=sin-30°
G26+ PCxc0/yc0/zc0 PDxa/ya/za PI[ca]/[sa]
```

9.4.2 Table 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.

9.4.3 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»).

9.4.3.1 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-).

9.4.4 Tool wear offset

Tool length wear offsets less than 100 µ (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»).

9.4.5 Tool correction (G29)

9.4.5.1 3 or 5-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 can only process spherical tools.

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 tonic tool.

G29 Tool correction with Turntables only
In order for the 3D tool correction (G29) to be taken into account when the machine includes only turntables (no twist axis), it is necessary to declare the tool direction vector IJK followed by the argument TD just after the G26+ function and before describing the turntable structure.

Example

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

9.5 Restrictions and conditions of use

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

- The dynamic operators that calculate reference corrections (E9500x) should not be declared when the RTCP function is enabled. One should proceed as follows:
    - Cancel the RTCP function (G26-)
    - Declare the dynamic operators
    - Re-enable the RTCP function (G26+ ...).

Other declarations that require RTCP disabled are:

- 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.

For E parameters detailed informations please refer to the Flexium Programming manual M00018.

Page deliberately empty

Page deliberately empty

Summary

Page deliberately empty

10 N/M AUTO function

10.1 General

N/M AUTO means that N out of the M axes of a machine are controlled by the part program, the other axes are controlled manually (Jog controls or Hand-wheel).

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 non interpolated. They are named as NMA later in this chapter.

The N/M AUTO function is active during machining operation.

10.2 Non interpolated axes

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

The axes declared as non interpolated become effectively so with a PLC information `(W_ALLCNC.NMAuto)`

It should be noted that at any given time only one non interpolated axis can be moved with the jogs controls or hand-wheel.

10.3 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 servo controlled.
- Parameter E912xx is set in a part program but the associated axis xx belongs to another channel.

For E parameters detailed information please refer to the Flexium Programming manual M00018.

10.4 Using the N/M AUTO function

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

- On the fly
- With the machine stopped.

The validation of the non interpolated status of the axes is made by the PLC program.

Please refer to the machine operation manual to know how this is activated.

10.4.1 Information for the machine tool builder

- The PLC program must confirm the N/M AUTO state by setting the PLC flag `W_ALLCNC.NMAuto`.

- The flag `RCNC.General.NMAuto` is the report of the CNC. This bit is set by the CNC as soon as the N/M AUTO function is enabled.

- `WCNC.Channel[8].FeedOveride` (channel 8 potentiometer) is used to control the feed rate on the non interpolated axes.

- The jog speed is defined by the variable `W_ALLCNC.ManualSpeed` which is taken into account as in manual mode.

- Two cases are possible depending on the state of `W_ALLCNC.NMAuto` when E912xx goes high:
    - Activation on the fly : If the N/M AUTO function was already set (via `W_ALLCNC.NMAuto`), then the NMA axes (via E912xx) are set to non interpolated immediately and the return variable `RCNC.General.NMAuto` goes high.

    - Activation on machine stop: If the N/M AUTO function was not already set then nothing changes. If this variable is later set then as above, nothing changes. It is only at the next machine stop (Cycle Start OFF and/or Cycle Stop ON) that the NMA status will be set to non interpolated and the `RCNC.General.NMAuto` flag set.

For E parameters detailed information please refer to the Flexium Programming manual M00018.

10.5 Stopping and restarting in N/M AUTO mode

There are three ways of re-enabling inhibiting an axis that has been declared as non interpolated.

- Programming E912xx=0. The N/M AUTO function remains enabled, but the corresponding NMA becomes interpolated immediately again. If this axis was moving, program execution will resume once it has stopped.

- The PLC resets `W_ALL_CNC.NMAuto` and CYCLE START OFF, then the N/M AUTO function is inhibited immediately.
  Error 107 is generated if the next block defines a circle.
  When the PLC sets `W_ALL_CNC.NMAuto` again, the N/M AUTO function is re-enabled immediately with the axis status defined by E912xx (if Cycle Start OFF and/or Cycle Stop ON).

- The PLC resets `W_ALL_CNC.NMAuto` when the CYCLE STOP ON, then an axis recall is requested to resume part program execution.
  The conditions to resume N/M AUTO are as above.

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

For E parameters detailed information please refer to the Flexium Programming manual M00018.

10.6 Checks included in N/M AUTO

The following checks are part of the N/M AUTO function:

- Acceleration check
- Speed check
- Check of travels
- Miscellaneous checks.

10.6.1 Acceleration checks

When the N/M AUTO function is active, the maximum permissible acceleration on the NMA is as defined by the machine parameters. However, if the RTCP function is active, a second limit is applied so as not to exceed the maximum acceleration on the driven axes.

10.6.2 Speed checks

The speed on the NMA is limited by machine parameter. If the RTCP function is active, a second limit is applied so as not to exceed the maximum speeds authorised on the driven axes.

10.6.3 Check of travels

The overtravel test is carried out according to machine parameters. 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, then a cycle hold is automatically generated to stop the interpolator.
The NMA axis can be retracted from the limit switches using the JOG control or hand-wheel.
The user can restart the program by pressing START CYCLE once the overtravel condition is OFF.

10.6.4 Miscellaneous checks

No test is made relative to overspeed (G12).
Homing must be completed on the NMA.

Page deliberately empty

Page deliberately empty

Summary

Page deliberately empty

11 B-Spline function

11.1 General

The B-Spline function is used to define a path described by several ‘Poles’.
A B-Spline trajectory segment is defined in several CNC Blocks. The number of blocks being equal to the number of Poles.

Any axis address (Cartesian or rotary - for modulo axes, the shortest path between poles is used) can be used to describe a pole.

The first pole is assumed to be reached by a prior movement. This movement can be in G00, G01 or a previous B-Spline command.

The second pole is programmed in the following block with one of preparatory functions G61 or G62. This block must include all axes addresses that are interpolated in the rest of the B-Spline.
The argument ND can also be included in the block. It specifies the degree of the polynomials. The default degree is equal to 3.

Two different preparatory functions can be used to define the second pole.
- Preparatory function G61 is used if the parameter used for creation of the B-splines corresponds to the curvilinear abscissa on the Cartesian path.
- Preparatory function G62 is used if the parameter used for creation of the B-splines is without physical meaning.

The following poles are programmed next one block for each pole. All the axes addresses included in these blocks must have been programmed in the block associated with the second pole (block G61/G62).

The last pole is associated with function G60.

Segmentation of the parametric domain (nodal vector) is programmed incrementally. The values programmed are the parametric distances between two consecutive nodes. The parametric length of an interval is associated with address H.

The block containing function G61 or G62 (second pole declaration) also declares the parametric length of the first real interval. The following blocks declare the parametric lengths of the following intervals.

11.2 Syntax

N.. G61/G62 [ND3] X... Y... Z... H ...

G61/G62	Second pole definition.
ND	Polynom degree.
X<x1> Y<y1>...	Pole coordinates
G60	Last pole definition.

11.3 Restrictions and programming errors

The B-Spline function is an optional feature. If the option is not activated, then error 4 is generated. This option also requires smooth polynomial interpolation option.

The parametric length of an interval associated with the address H. has the same format as the axes (5.3 in metric, 4.4 in inches).

Functions allowed in pole definition blocks:

- N: Block numbering.
- G61 or G62: B-Spline declaration, optionally associated with ND.
- G60: End of B-Spline declaration.
- X,Y… H Pole definition and interval H.
  The axes can be linear, rotary modulo or not.
- L0-L19: Variables (L0=.. and/or XL0)
  Empty blocks.(comment, block number and/or variable declaration L0 to L19.

Functions prohibited in pole definition blocks, generating error 22:

- S,T,M: Auxiliary and miscellaneous functions. Feedrate F.. is allowed in the declaration block (G61/G62) but not in the subsequent blocks
- I, J, K and P, Q, R Can not be are programmed in the current B.Spline version
- ND above 5: Polynomial degree ND above 5
- More than 6 axes: Number of axes addresses included in a pole above 6.

The table containing the polynomial coefficients is currently limited to 31 entries. Each axis of a pole uses a number of entries equal to the polynomial degree + 1.

With G62, five additional entries are used to characterize the function used as interpolation parameter to project the tool head position on the curvilinear axis. Saturation of this table generates error 6.

	Polynomial degree	Maximum number of axes
G61	3	6
	4	6
	5	5
G62	3	6
	4	5
	5	4

11.4 Precision of parameter H with G62

This section is specifically addressed to post-processor developers. The post-processor converts a CAD/CAM CL-FILE to an ISO file that can be read by the CNC.

The choice of the B-Spline parameter is left to the initiative of the smoothing algorithm developer. The only requirement is that this parameter be continuous and increasing.

The smoothing algorithm manages this parameter as a floating point parameter with the appropriate relative precision. The parameter is therefore included in the CL-FILE with the same precision (as many decimal digits as necessary).

For H, only three or four decimal digits are available to the post-processor developer, depending on the choice of inch or metric. Depending on the choice made for the smoothing parameter, this precision may be insufficient. However, a scaling factor can be applied between the value of the parameter of the CL-FILE and its value in the CNC ISO file.

Applying a scaling factor to the parameter does not modify the geometry of the interpolated path. (The scaling factor must remain constant through a given B-Spline function)

This section is therefore intended to help the post-processor developer choose the scaling factor to be used.
The general rule to be followed for the precision of the programmed parametric interval (H) can be inferred from the following considerations.

Consider any parametric interval. When the B-Spline parameter travels over this interval, the interpolated point follows a given path segment. The relative precision of the expression for the parametric interval must be better than or equal to the absolute geometric precision required in the path.

As an example, one can consider:

- If, for a given B-Spline, the CL-FILE parameter varies from 0 to 10 and the distance travelled over the path is 50mm, the scaling factor "k" to be applied is around 50mm/10 = 5.
- If, for another CAD/CAM, the CL-FILE parameter always varies from 0 to 1 and the B-Spline paths may be up to 1m long, the scaling factor to be used for "k" = 1000mm/1 = 1000.

Page deliberately empty

Summary

12 Dynamic operators

The «dynamic operators» feature is intended to cyclically execute parametric calculations at the rate of the system Real Time Clock (RTC). These calculations are, for example, used to execute rapid transfers of data or axes compensation.

12.1 General

The operations can be included in a part program or a subroutine.
The execution of each programmed operation begins when it is read and is repeated until the operation is cancelled.

The systems allows for up to 128 operations. Each operation is defined by its rank, an operation code and the associated parameters e.g. `O7 = 4 E81000/E81020/E80055/3600000.`

12.1.1 Virtual axes

Dynamic operators as other particular NUM functions may use what is called 'Virtual axes'.

A Virtual axis is declared in machine parameter P3 (linked to an interpolator), Parameter P9 (name and channel) but not in parameter P2 (not measured).
It can be displayed or not (P0) according to the particular need.

Such an axis can be programmed and interpolated or controlled by dynamic operators but will not directly generate a physical axis move.

Typical use could be for coordinate transformation, axes swapping and many other cases of applications.

For Px parameters detailed information please refer to the Flexium Parameters manual M00021.

12.1.2 List of operation codes

Code	Description
0	Cancel a particular operator
1	Add
2	Subtract
3	Multiply
4	Rotate a Vector
5	Load a Parameter With or Without Mask
6	Load a Peripheral Memory With or Without Mask
7	Indexed Load of a Parameter
8	Indexed Storage of a Parameter
9	Compare of Two Parameters
10	Conditions on the Sum of Two Parameters
11	Store Data in a Peripheral Memory (axis or PLC)
12	Increment with Modulo
13	Multiply by a Program Variable
14	Arc Tangent Function
15	Manual Control of an Axis (Axis jog)
16	Third-Degree Polynomial and its Derivative
17	Speed on Path (Feed rate on path)
18	Divide
19	Square Root
20	Axis Speed Synchronisation
21	Oscillation Cycles
22	Dynamic Operators in C Function
23	Electronic Cam
24	Differential Datum Shift

12.1.3 General syntax of an operation

On = Code Operand /Operand /Operand /...

The character «=» separates the operation number **On** from the operator **Code** .

Operation **n**umber:	Each operation to be performed is identified by the letter «O» followed by a decimal number n from 1 to 128. The number assigned to the operations determines the order in which they are executed.
Code:	Type of Operation. The operation is defined by the decimal number of the operator «m» from 0 to 24 followed by operands as indicated on previous page.
Operands:	Operands can be: • External parameters • Immediate values (positive or negative integers) • Axis addresses or parameters accessible by the PLC • Program variables or symbolic variables.

The first operand is always the destination, the character «/» is used to separate consecutive operands.

When the result of an operation assigned to an E parameter is not an integer, the system automatically truncates the decimal digits.

The operands are symbolized by «E», «I», «@», «L» for the most commonly used or by symbols specific to certain operations.

12.1.3.1 Example of definition of an operation

```
O35 = 2 E81000 / E70000 / E71000 /2
```

12.1.4 Cancel an operation

The execution of an operation ends with :

- Its cancellation by the operator «0»,
- Its replacement by an operation with the same execution order number,
- The end of program (M02) or reset.

12.1.4.1 Cancel an operation by operation code = 0

Syntax

On = 0

This syntax does not include any operands.

12.1.5 Order of execution of the operations

The system performs the operations in ascending order of their numbers, which do not have to be consecutive.

Example

```
O1 = 3 ...
O7 = 18 ...
O56 = 2 ...
```

12.1.5.1 Execution of consecutive operations

Several numbers of operations of different types can be written on the same line of a program (or subroutine).

Be careful with the order of declaration which can be different than the order of execution.

Example

```
O1 = 2 ... O9 = 19 ... O5 = 3 ...        Operations
X100 Next line
```

To start, the system will decode O1 stores and execute it
On the next RTC cycle, the system executes O1 again (already stored), decodes, stores and executes O9.

On the third cycle, the system performs O1 and O9 then decodes stores and executes O5.
On the following cycles, the system will execute O1, O5 and O9 in that order.

RTC Cycle	Decode	Execute	
1	O1	O1	
2	O9	O1, O9	
3	O5	O1, **O9, O5**	Declaration order
4	-	**O1, O5, O9**	In ascending order
Following cycles	-	**O1, O5, O9**	In ascending order

12.1.6 Dynamic operations can be used by any channel.

An operation declared in a channel can only be cancelled or modified in the same channel.
Any attempt to cancel or redefine an operation declared in another channel (and still active) returns «error 98».

Reset of a channel cancels all the operations declared for that channel.
An CNC or PLC reset cancels all the operations for all the CNC channels.

12.2 Types of operands used

The different operands used are:

- Type 1 external parameters
- Type 2 external parameters
- Type 3 external parameters
- Immediate values
- Operands pointing to two consecutive memory locations
- L program variables
- Symbolic variables
- Axis addresses.

Parameters E81xxx and E82xxx can be used as general purpose parameters if they are not already used for interaxes calibration by the machine builder.

Axes related parameters either refer to a program axis (index 0 for X up to 8 for C) and are then channel specific, or they refer to a physical axis (index 0 to 31 according to the physical address) in such case they are general and not channel specific anymore.

For E parameters detailed information please refer to the Flexium Programming manual M00018.

12.2.1 Type 1 external parameters

E7x000	Position reference of a program axis in the channel (x = 0 to 8)
E7x001	Axis reference memory for on-the-fly measurement (x = 0 to 8)
E7x100	Position reference from the interpolators (read/write if no feedback)
E79001	Speed setting of the spindle controlled by the channel
E79002	Setting of the feed rate potentiometer assigned to the channel
E79003	Distance remaining to be travelled on the path in the current block
E81xxx	General purpose parameters
E82xxx	General purpose parameters
E900xx	Axis measurement (xx = 0 to 31)
E9020x	Spindle speed setting (x = 0 to 3)
E950xx	Reference corrections of a physical axis (xx = 0 to 31)

12.2.2 Type 2 external parameters

E351xx	Test point 16 bits value of measure points selected by V130 for drive xx
E351(32+xx)	Test point 16 bits value of measure points selected by V131 for drive xx
E351(64+xx)	Test point 32 bits value of measure points selected by V130 or V131 for drive xx
E50xxx	Tool length «L» (milling)or X dimension (turning)
E51xxx	Tool tip radius «@» (milling) or Z dimension (turning)
E52xxx	Tool radius «R»
E53xxx	Dynamic length correction (milling) or X dimension (turning)
E54xxx	Dynamic radius correction (milling) or Z dimension (turning)
E55xxx	Tool nose direction (turning)
E56xxx	H of the table of dynamic corrections (milling, turning)
E57xxx	Tool type
E6x000	DAT1 (program axis related)
E6x001	DAT2 (program axis related)
E6x002	Minimum dynamic machine travel
E6x003	Maximum dynamic machine travel
E6x004	Off-centring on the axis (milling) (x = 0 to 5)
E6x005	Programmed shift on the axis
E7x001	Axis reference memory measured on the fly
E7x002	Minimum static machine travel
E7x003	Maximum static machine travel
E7x004	Movement increment
E7x005	Axis assignment
E7x006	Axis coupling
E7x007	Axes programmed by diameter (turning) (x = 0 or 3)
E79000	Position reference of the spindle for the channel
E79001	Speed setting of the spindle controlled by the channel
E79002	Feed rate override value (channel related)
E79003	Remaining distance in the current block
E79004	Current path speed programmed in the block
E79005	Minimum feed override setting (channel related)
E79006	Maximum feed override setting (channel related)
E79007	Minimum spindle override setting (channel related)
E79008	Maximum spindle override setting (channel related)
E80xxx	General purpose parameters (xxx = 0 to 199)
E81xxx (1)	General purpose parameters (xxx=0 to 999) or interaxis calibration table
E82xxx (1)	General purpose parameters (xxx=0 to 999) or interaxis calibration table
E83xxx	General purpose parameters (xxx = 0 to 999)
E84xxx	General purpose parameters (xxx = 0 to 999)
E900xx	Axis measurement (xx = 0 to 31) (write allowed if axis not servo)
E9010x	Spindle position reference (x = 0 to 3)
E950xx	Machine axis reference corrections (xx = 0 to 31)
E951xx	Offset of the machine reference (P16)
E952xx	Axis (or spindle) measure correction offset

(1) The x range is limited by P58N0.

12.2.3 Type 3 external parameters

Writing of E3400x have effect on NCK embedded digital output if PLC variable Flexium_NCK.WCNC.IO.LockDigitalOutput := FALSE

Writing of E3410x have effect on NCK embedded analogue output if PLC variable Flexium_NCK.WCNC.IO.LockAnalogOutput := FALSE
(Please refer to the Flexium Commissioning Manual M00010).

E100xx	PLC bit access parameter read from PLC (xx 0 to 31)
E200xx	PLC bit access parameter written by PLC (xx 0 to 31)
E3300x	PLC memory bytes write access (x 0 to 9)
E3400x	NCK embedded digital output : Bytes (x= 0 to 1)
E3410x	NCK embedded analogue output : Word (x = 0 to 1)
E3600x	PLC memory words write access (x= 0 to 2)
E37xxx	PLC memory byte write access (xxx = 0 to 127)
E4300x	PLC bytes read access (x=0 to 9)
E4400x	NCK embedded digital inputs : Bytes (x= 0 to 1)
E4410x	NCK embedded analogue inputs : Words (x= 0 to 2)
E4600x	PLC Memory words read access (x=0 to 3)
E77xxx	PLC memory byte read access (xxx = 0 to 127)

Please refer to Flexium Programming manual M00018 for other details.

12.2.4 Immediate values

The immediate operands are positive or negative integers.
In arithmetic operations, the immediate value is a number of shifts (>0 ➔ left or <0 ➔ right) this results in multiplying or dividing the result before storing at the destination operand.
The purpose of this shift is to maintain the highest possible precision in consecutive as the calculations are made with integer values.

In operations used for jumps (operators 9 and 10), the immediate value is the operation number to which the jump is to be made. Use of shifts should be done with great care in order to prevent overflow. An overflow will not be signaled and could lead to impredictable results.

12.2.5 Operands pointing two consecutive memory locations

These operands are external parameters symbolised by the letter «E..» (type 1 or 2) pointing two consecutive memory locations.

Example

E61000 points two memories E61000 and E61001.

12.2.6 L program variables

These operands are symbolised by the letter «L». They refer to local variables: L0 to L19.

12.2.7 Symbolic variables

Symbolic variables of type [symb] between square brackets can be used in certain operations.

12.2.8 Axis Addresses

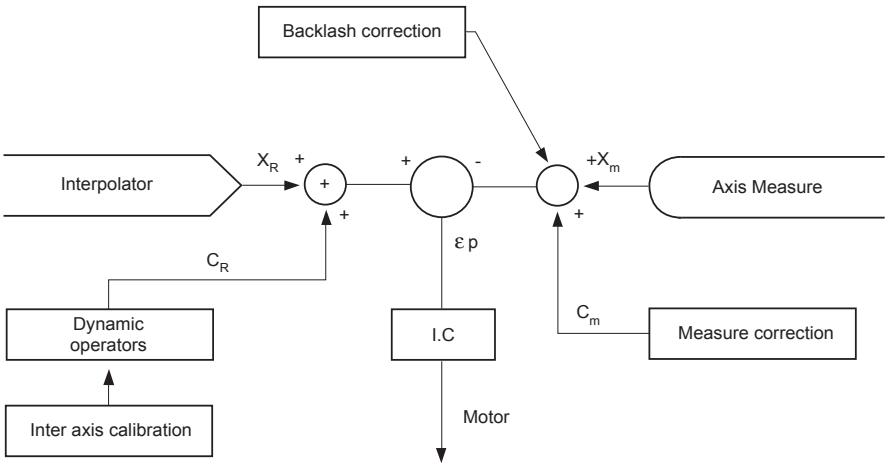
These operands specify the addresses of an axis.
Depending on the operation, the axes are defined by:

@...	Physical address of any machine axis
@x or @y or @z	Physical addresses of the X, Y and Z axes
@m or @e	Physical address of a master axis or a slave axis respectively
W	Handwheel address

For E parameters detailed information please refer to the Flexium Programming manual M00018.

12.3 Dynamic operators execution

The flowchart below defines the location of dynamic operators in execution of the Measurement Servo-Drive CNC Task.



Calculate spindle speed and position

For each axis:
· Read Xm
· Calculate XR

For each axis:
Calculate CR
(interaxes calibration table)

Execute dynamic operators

For each axis:
Calculate Cm by reading:·
The measurement correction table.
The backlash correction.
Calculate the following error

ε = (XR + CR) - (Xm + Cm)

For each axis:
transfer the following error to the drive

Generate and transfer spindle speed references

12.4 Code1 ► Add

This operation performs an addition and multiplies the result by a power of 2.

Syntax

On = 1 Ea / Eb / Ec {/I}

Ea	Type 1 external parameter
Eb	Type 2 external parameter
Ec	Type 2 external parameter
I	Immediate value

Definition

Ea integer part of (Eb + Ec) { x 2^I }

Example

```
On = 1 E81000/E81001/E81002/-7
```

Equivalent to: E81000 equals the integer part of (E81001 + E81002) / 128

For E parameters detailed information please refer to the Flexium Programming manual M00018.

12.5 Code2 ► Subtract

This operator performs a subtraction and multiplies the result by a power of 2.

Syntax

On = 2 Ea / Eb / Ec {/I}

Ea	Type 1 external parameter
Eb	Type 2 external parameter
Ec	Type 2 external parameter
I	Immediate value

Definition

Ea integer part of (Eb - Ec) { x 2^I }

Example

```
On = 2 E81000/E81001/E81002/1
```

Equivalent to: E81000 equals the integer part of (E81001 - E81002) x 2

For E parameters detailed information please refer to the Flexium Programming manual M00018.

12.6 Code3 ► Multiply

This operator performs a multiplication and multiplies the result by a power of 2.

Syntax

On = 3 Ea / Eb / Ec {/I}

Ea	Type 1 external parameter
Eb	Type 2 external parameter
Ec	Type 2 external parameter
I	Immediate value

Definition

Ea integer part modulo 2^32 of Eb x Ec { x 2^I }
In this operation, Ea, Eb and Ec are taken as 32 bit integers.

Precaution to prevent overflow should be taken in general and more importantly in case of multiplication. An overflow (value exceeds 2^32) could arise and will result in inpredictable results. It is importnat to make sure that the different operand are correctly scaled.

Example

```
On = 3 E81000/E81001/E81002/-16
```

Equivalent to: E81000 equals the integer part modulo 2^32 of (E81001 x E81002) / 65536

For E parameters detailed information please refer to the Flexium Programming manual M00018.

12.7 Code4 ► Rotate a vector

This operation rotates a vector in the interpolation plane.

Syntax

On = 4 Ea / Eb / Ec / I

Ea	Type 1 external parameter pointing to two consecutive memory locations
Eb	Type 2 external parameter pointing to two consecutive memory locations
Ec	Type 2 external parameter
I	Immediate value

Definition

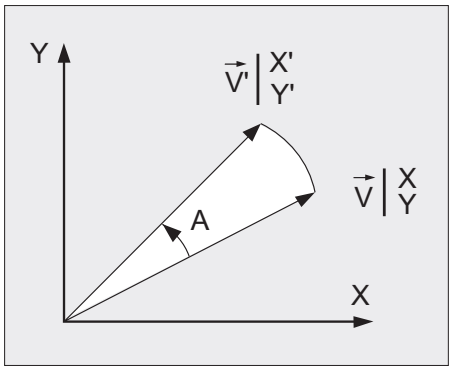
- Ea points to two consecutive integers corresponding to the coordinates of the vector resulting of the rotation.
- Eb points to two consecutive integers corresponding to the coordinates of the vector to be rotated .
- Ec is the angle of rotation measured in the unit defined by I.
- I defines the unit angle as a fraction of a complete revolution, e.g. i = 3,600,000 for one ten-thousandth of a degree. The value for I should always be positive, otherwise error 93 will be generated.
- The vector is rotated counter clockwise.

Vector rotation uses auxiliary tables (one per axis processed) similar to those used with operators:
- 14 Arc tangent
- 21 Oscillation cycles
- 22 Dynamic operators in C
- 23 Electronic cam.

The total number of tables is limited to sixteen. If this number is exceeded, the system returns «error 93».

Description of the operations

- Ea will be loaded with X' and Ea+1 is loaded with Y'
- Eb contains X and Eb+1 contains Y
- Ec contains angle A



Example

The following operation can be used to calculate the cosine and sine of an angle A.

```
On = 4 E81000/E81020/E80055/3600000
```

Where:
E81020 = 1000 and E81021 = 0 (vector with norm 1000 parallel to the X axis)
E80055 = A (angle in ten-thousandths of a degree)

Then: E81000 = 1000 x cos A and E81001 = 1000 x sin A

12.8 Code5 ► Load a parameter with or without mask

This operation is used to load an external parameter with or without a mask.

Syntax

On = 5 Ea / Eb {/Ec} {/I}

Ea	Type 1 external parameter
Eb	Type 2 external parameter
Ec	Type 2 external parameter
I	Immediate value

Description

Ea will be loaded with the 32 LSBs of the result of Eb {AND Ec} shifted I bits left (I positive) or right (I negative).

Examples

- Store the value of the milling spindle reference

```
On = 5 E79001/E80000
```

Where:
E80000 Spindle reference value
E79001 Spindle references loaded with the contents of E80000.

- Load of the second byte of a parameter to set the feed rate potentiometer

```
On = 5 E79002/E80021/E80001/-8
```

X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	s	t	u	v	w	x	y	z	X	X	X	X	X	X	X	X	E80021
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	E80001
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	s	t	u	v	w	x	y	z	0	0	0	0	0	0	0	0	MASKING
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	s	t	u	v	w	x	y	z	RESULT

Application of the mask, shift by -8 bits and stores stuvwxyz E79002

For E parameters detailed information please refer to the Flexium Programming manual M00018.

12.9 Code6 ► Load of a peripheral memory with or without mask

This operator is used to load an axis or PLC memory with or without a mask.
Two syntaxes are used for the operation, depending upon the source:
- Axis address or
- PLC address.

12.9.1 Load an axis address

Syntax

On = 6 Ea / @.. {/Ec} {/I} {W.}

Ea	Type 1 external parameter
@..	Axis address
Ec	Type 2 external parameter
I	Immediate value
W	Handwheel address (W0 to W3)

Description

Ea will be loaded with the integer part of @.. {AND Ec} { x 2^I }
In the operations, @.. is the address of the axis , i.e. the value read in the axis measurement register (equivalent to E900xx).

For E parameters detailed information please refer to the Flexium Programming manual M00018.

Example with handwheels:

```
O1 = 6 E81000/W2
```

Load parameter E81000 with the counter value of the third handwheel regardless of its type. (NCK, CAN or EtherCAT)

12.9.2 Load from PLC Data

Syntax

On = 6 Ea / Eb {/Ec} {/I}

Ea	Type 1 external parameter
Eb	Type 3 external parameter
Ec	Type 2 external parameter
I	Immediate value

Description

Ea will be loaded with the integer part of Eb {AND Ec} {x 2^I}
Eb is the source operand which can be:

- E10000 (no other E10xxx allowed)
- E20000 (no other E20xxx allowed)
- E47000 to E47124
- E43xyz and E44xxy
- E46000 to E46002.

Parameters E10000 and E20000

The variables E10000 and E20000 respectively recover the content of parameters E10000 to E10031 and E20000 to E20031 (list of 32 bits).
- E10000 (or E20000) on bit 0
- E10001 (or E20001) on bit 1 and so forth up to E10031 (or E20031) on bit 31.

In this case, operand Ec is the mask and I is the shift to select the bit(s) concerned.
For E parameters detailed information please refer to the Flexium Programming manual M00018.

Example

Read of E20020 and assignment to bit 0 (mask on bit 20):

```
E81000 = 1048576
On = 6 E81001/E20000/E81000/-20
```

12.9.2.1 Parameters E47000 to E47124

Parameters E47000 to E47124 address four consecutive bytes starting on the address specified.
For instance, E47100 addresses bytes E47100, E47101, E47102, E47103.
In this case, operand Ec is the mask and I is the shift to select the bit(s) concerned.

12.9.2.2 Parameters E4300x

Parameters E4300x address the PLC memory Flexium_NCK.WCNC.E43000[x] x=0 to 9.

Example

```
On = 6 E81000/E43000
```

12.9.2.3 Parameters E4400x

Parameters E4400x address the digital input of the NCK.
E4400x: x = 0 or 1
E44000 = in7..in0 connector X10
E44001 = in15..in8 connector X10.

Example

```
On = 6 E81000/E44000
```

12.9.2.4 Parameters E4410x

Parameters E4410x address the Analog input of the NCK.
E4410x: x = 0 to 3
E44100 = Analog in0 connector X9
E44101 = Analog in1 connector X9
E44102 = Analog in2 connector X9
E44103 = Analog in3 connector X9.

Example

```
On =  6 E81000/E44100
```

(E81000)=4096 = 10V

12.9.2.5 Parameters E4600x

Parameters E4600x address the PLC memory Flexium_NCK.WCNC.E46000[x] x=0 to 2.

Example

```
On =   6 E81000/E46000
```

For E parameters detailed information please refer to the Flexium Programming manual M00018.

12.10 Code7 ► Indexed load of a parameter

This operator accesses an indexed value contained in a table.

Syntax

On = 7 Ea / Eb / Ec / I

Ea	Type 1 external parameter
Eb	Type 2 external parameter
Ec	Type 2 external parameter
I	Immediate value

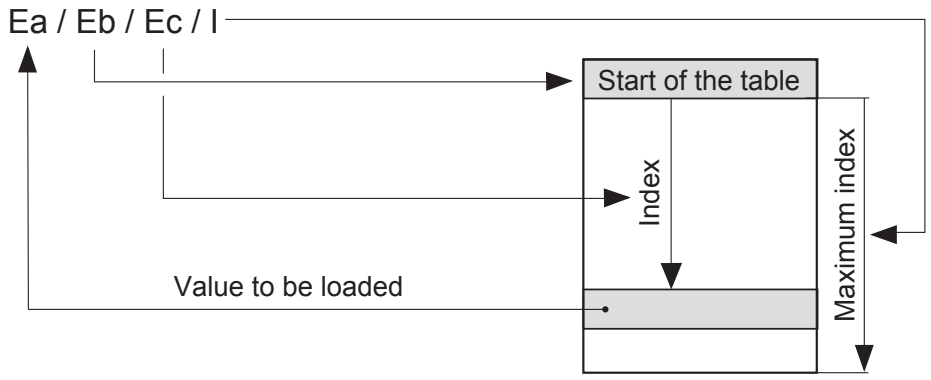
Description

Ea will be loaded with the contents of E(b + contents of Ec)

- Eb addresses the start of the table (index 0)
- Ec contains the index
- I is a positive value giving the maximum index value.

The operation cannot be performed if the value of Ec is negative or higher than I-1.
(No error is reported).

Schematic representation



Example

```
On = 7 E82020/E81000/E82011/90
```

If E82011 = 35, then E82020 will be loaded with the contents of E81035 (E81000 + 35).
If E82011 = 111, the operation will not be performed (since 111 is higher than the maximum index).

For E parameters detailed information please refer to the Flexium Programming manual M00018.

12.11 Code8 ► Indexed storage of a parameter

This operation is used to store a value in an indexed table.

Syntax

On = 8 Ea / Eb / Ec / I

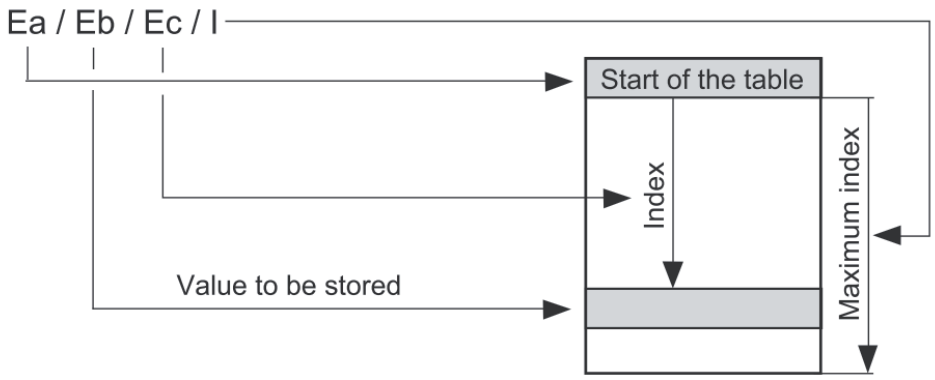
Ea	Type 1 external parameter
Eb	Type 2 external parameter
Ec	Type 2 external parameter
I	Immediate value

Description

- Ea addresses the start of the table (index 0)
- Eb will be stored (written) in E (a + contents of Ec)
- Ec contains the index
- I is a positive value giving the maximum index value.

The operation cannot be performed if the value of Ec is negative or higher than I-1. (No error is reported).

Schematic representation



Example

Store the contents of parameter E82000 in a table

```
On = 8 E81000/E82000/E82011/70
```

If E82011 = 25, then E82000 will be stored in E81025 (E81000 + 25)
If E82011 = 96, the operation will not be performed (96 is higher than the maximum value of the index) .

12.12 Code9 ► Compare two parameters

This operator makes a comparison whose result can be:
- A conditional jump to a specific operation
- An error detection with a halt.

Syntax

On = 9 Ea / Eb / Ia {/ Ib {/ Ic}}

Ea	Type 1 external parameter
Eb	Type 2 external parameter
Ia	Immediate value
Ib	Immediate value
Ic	Immediate value

Description

See the flow chart on the following page.
The comparison of Ea and Eb determines the value of the intermediate result :«v»:

if Ea < Eb	v = Ia
if Ea = Eb	v = Ib (default = 0)
if Ea > Eb	v = Ic (default = 0)

Then the system compares v with n (number of the current dynamic operator)

v > n:	jump to operation Ov
0 <= v <= n	go to the next operation On + 1
if v < 0	display a machine error with a halt (machine error -v) and go to the next operation

In certain cases, operands Ia, Ib and Ic can be replaced by:

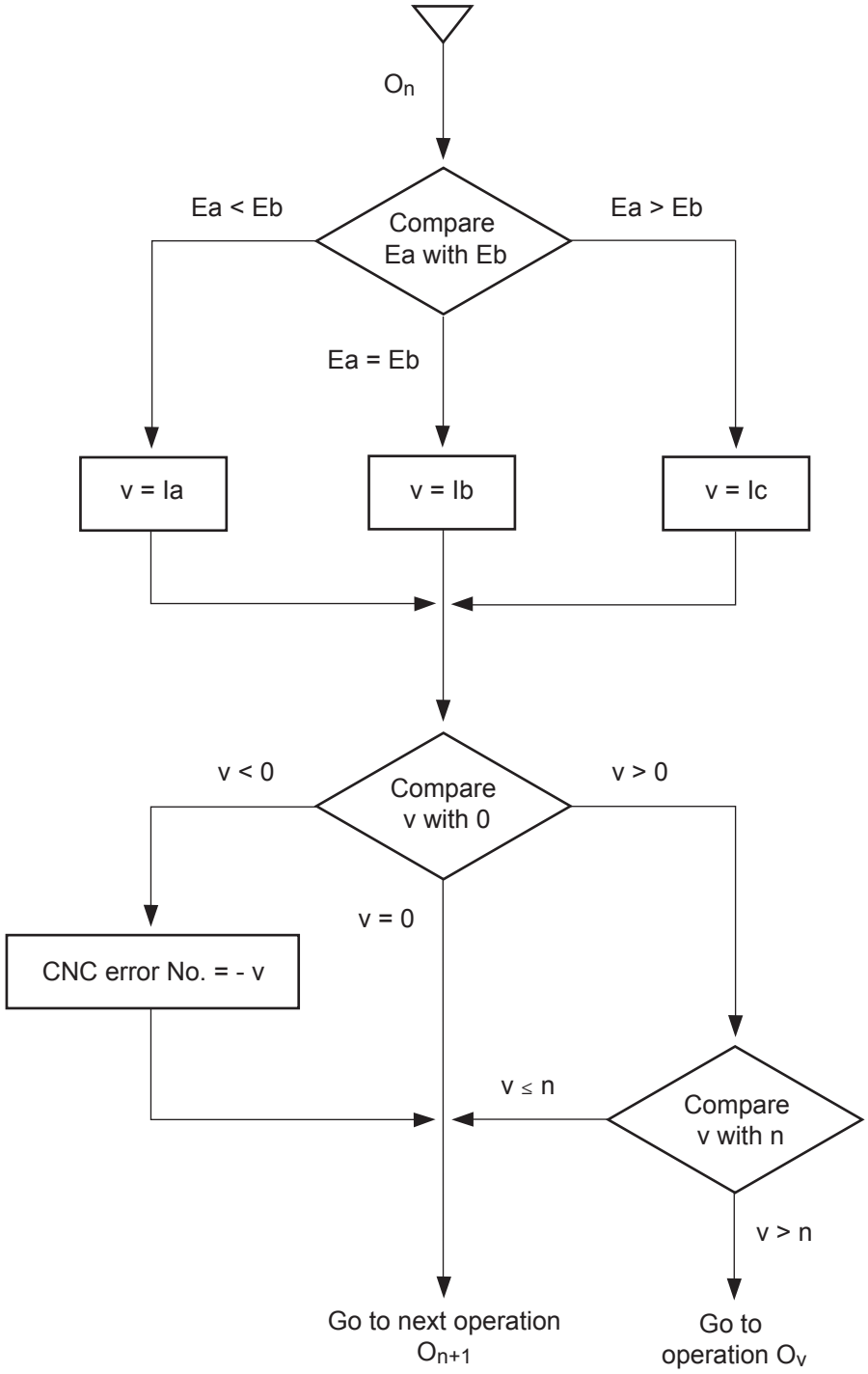
L	program variables
[symb]	symbolic variables

Example

```
O7 = 9 E81000/E81005/15/0/-5
```

In this example:
```
If E81000 < E81005: jump to operation O15
```
(if operation 6 were specified instead of 15, go to next operation O8)
```
If E81000 = E81005: go to next operation O8
If E81000 > E81005: display CNC error number 5.
```

Flowchart



- If Ib and / or Ic are not defined, their value default to 0.
- A jump is always forward, for safety purposes there is no possibility to create a loop or re-execution of the same operation.

12.13 Code10 ► Conditions on the sum of two parameters

This operator is used to check whether the sum of two parameters is between a minimum and a maximum.

The result of the operation can cause:
- A conditional jump to a specific operation
- The display of a machine error followed by a halt.

Syntax

On = 10 Ea / Eb / Ec / I or E37xxx

Ea	Type 1 external parameter
Eb	Type 2 external parameter
Ec	Type 2 external parameter pointing to two consecutive memory locations
I or E37xxx	Immediate value or Type 3 external parameter

Description

Refer to the flowchart on the following page.

- Ea and Eb are two parameters whose sum must be within two limits (minimum and maximum)
- Ec defines the limits (minimum limit followed by maximum limit)
- I or E37xxx is the result of the comparison which can be defined by:
    - A positive value causing a jump or a negative value corresponding to machine error (-I)
    - A parameter E37xxx which addresses a byte of the PLC field by E37000 to E37127 (loaded with the value -128).

This operation is mainly used for checking axis travels, i.e.:
`OT- < reference + corrections < OT+ (OT: Overtravel)`

Example

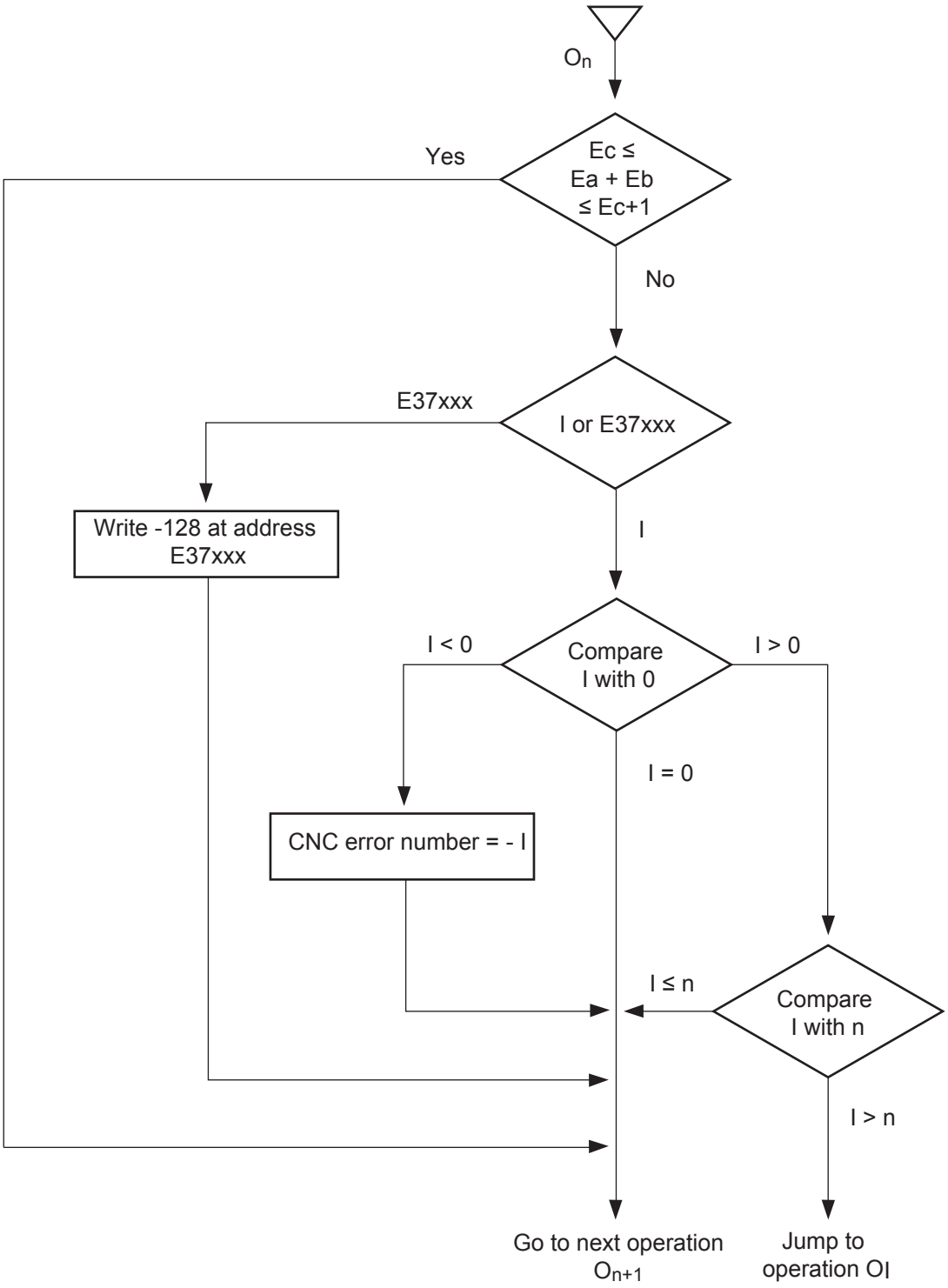
```
On = 10 E81000/E81001/E82000/E37xxx
```

Checks the sum of E81000 and E81001

- E82000 and E82001 contain the minimum and maximum limits of the sum E81000 + E81001
- E37xxx is the address of the byte to be loaded with the value -128, if the sum is not within the limits.

For E parameters detailed information please refer to the Flexium Programming manual M00018.

Flowchart


On
Ec ≤
Ea + Eb
≤ Ec+1
Yes
No
E37xxx
I or E37xxx
Write -128 at address
E37xxx
I
I < 0
Compare
I with 0
I > 0
I = 0
CNC error number = - I
I ≤ n
Compare
I with n
I > n
Go to next operation
On+1
Jump to
operation OI

12.14 Code11 ► Store data in a peripheral memory

This operator is used to store data in an analogue axis or a PLC memory.
The syntax differs according to whether the data is to be stored at:

- An analogue axis address, or
- A PLC address.

12.14.1 Storage at an axis address

Syntax

On = 11 @.. / E..

@..	Axis address (destination)
E..	Type 2 external parameter

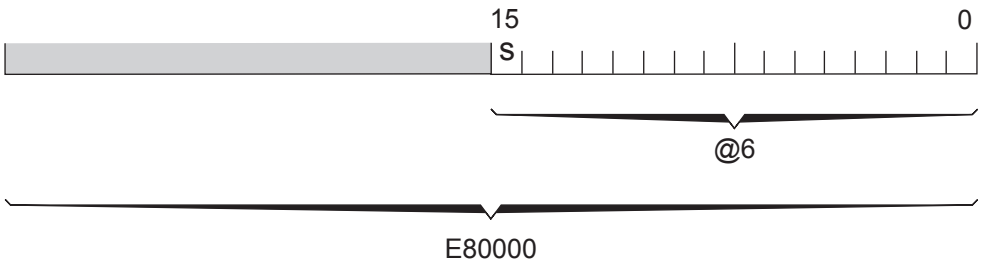
Description

The operation writes to the axis DAC. It is only valid for the NCK embedded analogue axes ports.
@.. is the address of the axis encoder (as defined in P70) where the content will be stored.
The data are absolute signed values encoded on 16 bits, value between $8000 (-10 V) and $7FFF (+10 V).

Example

Store the contents of E80000 (value encoded on 32 bits) at address @6.

```
On = 11 @6/E80000
```



12.14.2 Store in a PLC memory

Syntax

On = 11 Ea / Eb / I

Ea	Type 3 external parameter
Eb	Type 2 external parameter
I	Immediate value

Description
In the operation:
- The value at address Eb is stored in Ea
- I is the number of bits or bytes to be transmitted.

(I must always be specified for storage in the PLC memory).

Ea defines the following PLC parameters:
- E10000 to E10031
- E33000 to E33009
- E34000 and E34001
- E34100 and E34101
- E36000 to E36002
- E37000 to E37127

For E parameters detailed information please refer to the Flexium Programming manual M00018.

12.14.2.1 Parameters E10000 to E10031

The operation is used to write one or more consecutive bits in the list.

Example

```
On = 11 E10012/E81022/I
```

In this case, the I least significant bits (LSBs) of E81022 will be stored in parameters E10012 (bit I-1) to E10012+(I-1) (bit 0).

12.14.2.2 Parameters E3300x

Parameters E3300x address the PLC memory Flexium_NCK.RCNC.E33000[x] x=0 to 9 .

Example

```
On = 11 E33000/E81000/1 (access to one byte at a time)
```

12.14.2.3 Parameters E3400x

Parameters E3400x address the digital output of the NCK.
E3400x: x = 0 or 1
E34000 = out7..out0 connector X10
E34001 = out15..out8 connector X10.

Example

```
On = 11 E34000/E81000/1 (access to one byte at a time).
```

Reminder

The CNC digital outputs can be managed via Parameters E3400x if the PLC variable Flexium_NCK.WCNC.IO.LockDigitalOutput[x]:= FALSE.

12.14.2.4 Parameters E3410x

Parameters E3410x address the Analog output of the NCK.
E3410x: x = 0 or 1
E34100 = Analog out0 connector X9
E34101 = Analog out1 connector X9
.

Example

```
On = 11 E34100/E81000/2 (access to two bytes at a time).
```

E81000=32767 (corresponds to 10 V).

Reminder

The CNC analog outputs can be managed via Parameters E3410x if the PLC variable Flexium_NCK.WCNC.IO.LockAnalogOutput[x]:= FALSE.

12.14.2.5 Parameters E3600x

Parameters E3600x address the PLC memory Flexium_NCK.RCNC.E36000[x] x=0 to 2.

Example

```
On = 11 E36000/E81000/2 (access to two bytes at a time)
```

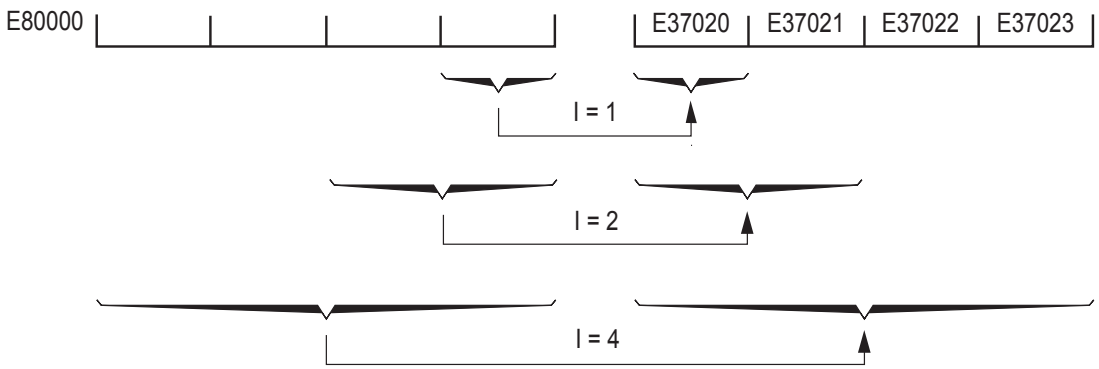
12.14.2.6 Parameters E37000 to E37127

Parameters E37xxx are general purpose Bytes parameters used for exchange via dynamic operators.

Example

```
On = 11 E37020/E80000/I
```

- If I is equal to 1: the least significant byte E80000 is stored in E37020.
- If I is equal to 2: the two least significant bytes of E80000 are stored in E37020 and E37021 (least significant byte).
- If I is equal to 4: All 32 bits of E80000 are stored in E37020 (most significant byte) to E37023 (least significant byte).



For E parameters detailed information please refer to the Flexium Programming manual M00018.

12.14.2.7 Parameters E3410x

Parameters E3410x address the Analog output of the NCK.
E3410x: x = 0 or 1
E34100 = Analog out0 connector X9
E34101 = Analog out1 connector X9.

Example

```
On = 11 E34100/E81000/2 (access to two bytes at a time)
```

E81000=32767 (corresponds to 10 V).

Reminder

The CNC analog outputs can be managed via Parameters E3410x if the PLC variable Flexium_NCK.WCNC.IO.LockAnalogOutput[x]:= FALSE.

12.14.2.8 Parameters E3600x

Parameters E3600x address the PLC memory Flexium_NCK.RCNC.E36000[x] x=0 to 2.

Example

```
On = 11 E36000/E81000/2 (access to two bytes at a time)
```

12.15 Code12 ► Increment with modulo

This operator is used to increment a variable by a defined amount and according to a modulo.

Syntax

On = 12 Ea / Eb / I

Ea	Type 1 external parameter
Eb	Type 2 external parameter
I	Immediate value always positive

Description

EaRTCi = (EaRTCi-1 + Eb) modulo I

Examples

```
E80000=3
On = 12 E81000/E80000/11
```

Increment by 3 modulo 11

```
E80000=1
On = 12 E81000/E80000/3600000
```

Each RTC, the value of E81000 is incremented by one up to 3600000. E.g. rotation angle.

For E parameters detailed information please refer to the Flexium Programming manual M00018.

12.16 Code13 ► Multiply by a program variable

This operator multiplies the contents of a parameter by a program variable.

Syntax

On = 13 Ea {/ Eb} / L..

Ea	Type 1 external parameter
Eb	Type 2 external parameter
L..	Program variable

Description

The integer part of Eb x L... will be stored in Ea.
L.. is a variable from L0 to L19.

Examples

```
On = 13 E81000/E80001/L19
```

E81000 ← integer part of E80001 x L19

If E80001 = 10 and L19 = 1.55, then:
E81000 = 15 (truncated value of the result, 15.5).

If Eb is missing then it is considered equal to 1.

```
On = 13 E81000/L10
```

E81000 ← integer part of L10

For E parameters detailed information please refer to the Flexium Programming manual M00018.

12.17 Code14 ► Arc tangent function

This operator calculates an angle by the arc tangent function.

Syntax

On = 14 Ea / Eb / Ec / I

Ea	Type 1 external parameter
Eb	Type 2 external parameter
Ec	Type 2 external parameter
I	Immediate value

Description

Ea ← Integer part of arc tangent Ec/Eb (Ea is always positive and less than the modulo)

- Ea is loaded with the angle in the unit defined by I
- Eb is the cosine of the angle (or adjacent side)
- Ec is the sine of the angle (or opposite site)
- I is the unit in which the modulo of the angle is expressed, i.e. I = 3600000 for ten-thousandths of a degree.

The arc tangent function uses auxiliary tables (one per axis processed) similar to those used with operators 4 (vector rotation) and 21 (oscillation cycles), 22 (Dynamic operators in C) and 23 (Electronic cam).

The total number of tables is limited to sixteen.
If this number is exceeded, the system returns «error 93».

For E parameters detailed information please refer to the Flexium Programming manual M00018.

Examples

```
On = 14 E81000/E71004/E70004/3600000
```

E81000 will show the current angle of the trajectory (in X-Y) for example to make sure the tool is always parallel to the tangent of the curve being machined.

12.18 Code15 ► Axes manual control

This operator allows control of an axis by the jogs when executing a part program.
The operator complies with the maximum speed and acceleration on the axis.

Syntax

On = 15 @.. / E47xxx / Eb / I

@..	Machine axis address
E47xxx	Type 3 external parameter
Eb	Type 2 external parameter pointing to two consecutive memory locations
I	Immediate value

Description

- @.. is the address (0 to 31) of the machine axis to be controlled.
- E47xxx is a parameter from E47000 to E47127 containing the speed override coefficient (a value of 255 corresponds to 100%).
- Eb points to two consecutive integers. The first contains the minimum travel and the second the maximum travel.
- I defines the speed in the unit in which the movement is expressed.

The operation cannot be executed unless the axis is declared as non-interpolated by programming using parameter E910xx = 0 (xx = physical axis address).
After resetting E910xx = 1, the operator remains valid.

Examples

```
On = 15 @3/E47xxx/E81000/120
```

- @3 is the axis to be manually controlled
- E47xxx contains the override coefficient (e.g. E47xxx = 192, speed = 75%)
- E81000 contains the minimum travel, maximum travel is in E81001
- 120 is the speed of movement in mm/min.

12.19 Code16 ► Third-degree polynomial and derivative

This operator computes a third-degree polynomial and its derivative from the three coefficients and the constant contained in program variables.

Syntax

On = 16 Ea / Eb / Lxx {/*}

Ea	Type 1 external parameter pointing to two consecutive memory locations
Eb	Type 2 external parameter
L_{xx}	First program variable
{/*}	Request to calculate the derivative of the function

Description

E_a = integer part of $L_{xx} * E_b^3 + L_{xx+1} * E_b^2 + L_{xx+2} * E_b + L_{xx+3}$

If /* is present: E_{a+1} = Integer part of $3 * L_{xx} * E_b^2 + 2 * L_{xx+1} * E_b + L_{xx+2}$

- Lxx is the first of four consecutive program variables (xx = 00 to 16). The three coefficients of the function and the constant are set in these four variables.
- The optional symbol * in {/*} is the request to calculate the derivative of the function. This derivative is then stored in memory Ea + 1.

Examples

```
On = 16 E81000/E80000/L5/*
```
(derivative calculation requested)

E81000 ← $L5 * E80000^3 + L6 * E80000^2 + L7 * E80000^1 + L8$
E81001 ← $3*L5 * E80000^2 + 2 * L6 * E80000 + L7$.

For E parameters detailed information please refer to the Flexium Programming manual M00018.

12.20 Code17 ► Speed on trajectory

This operator measures speed on the path defined by one to three axes.

Syntax

On = 17 Ea / @.. {/@..} {/@..}

Ea	Type 1 external parameter
@..	Address of first axis
@..	Address of second axis
@..	Address of third axis

Description

Ea = resultant speed on @.. + {@..} + {@..}
In the operation, @.. (0 to 31) is the address of each of the machine axes of a channel.

The result of the operation is expressed in internal system unit/sample).

Examples

```
On = 17 E81000/@0/@2/@4
```

E81000 ← resultant speed on X,Z and U (X@0, Z@2 and U@4).

12.21 Code18 ► Divide

This operator performs a division and multiplies the result by a power of two.

Syntax

On = 18 Ea / Eb / Ec {/I}

Ea	Type 1 external parameter
Eb	Type 2 external parameter
Ec	Type 2 external parameter
I	Immediate value

Description

Ea integer part modulo 2^{32} of (Eb / Ec) {* 2^I} .

In the operation, Ea, Eb and Ec are values on 32 bits.
It should be noted that no error is reported for a division by 0. The result is then the maximum value with the same sign as Eb.

Examples

```
On = 18 E81000/E80000/E81001/3
```

E81000 ← integer part modulo 2^{32} of (E80000 / E81001) * 8

12.22 Code19 ► Square root

This operator is used to calculate a square root and multiply the result by a power of two.

Syntax

On = 19 Ea / Eb / I

Ea	Type 1 external parameter
Eb	Type 2 external parameter
I	Immediate value

Description

Ea integer part modulo 2^{32} of √Eb {x 2^I} .

In the operation, Ea and Eb are values on 32 bits.
It should be noted that the result of the operation has the same sign as Eb.

Examples

```
On = 19 E81000/E80000/3
```

E81000 ← integer part modulo 2^{32} of √E80000 * 8

12.23 Code20 ► Axis speed synchronisation

This operator synchronises the speed of a slave axis with a master axis.

Syntax

On = 20 @e / @m / E20xxx / E37xxx / I

@e	Slave axis address
@m	Master axis address
E20xxx	Type 3 external parameter
E37xxx	Type 3 external parameter
I	Immediate value

Description

- E200xx is a parameter E20000 to E20031
- E37xxx is a parameter E37000 to E37127
- I (expressed in 1/256) set the value of the coupling coefficient which may be different from 1 (positive or negative integer).

Synchronisation can be performed on the move, i.e. when synchronisation is activated while the master axis is moving, the slave axis is accelerated up to the master axis reference speed by a $\sin^2$ curve.

Synchronisation can also be deactivated on the move. In this case, the slave axis is desynchronised from the master axis and decelerates according to the same $\sin^2$ curve.

Synchronisation is activated by the PLC via parameter E200xx (= 1: synchronisation active).

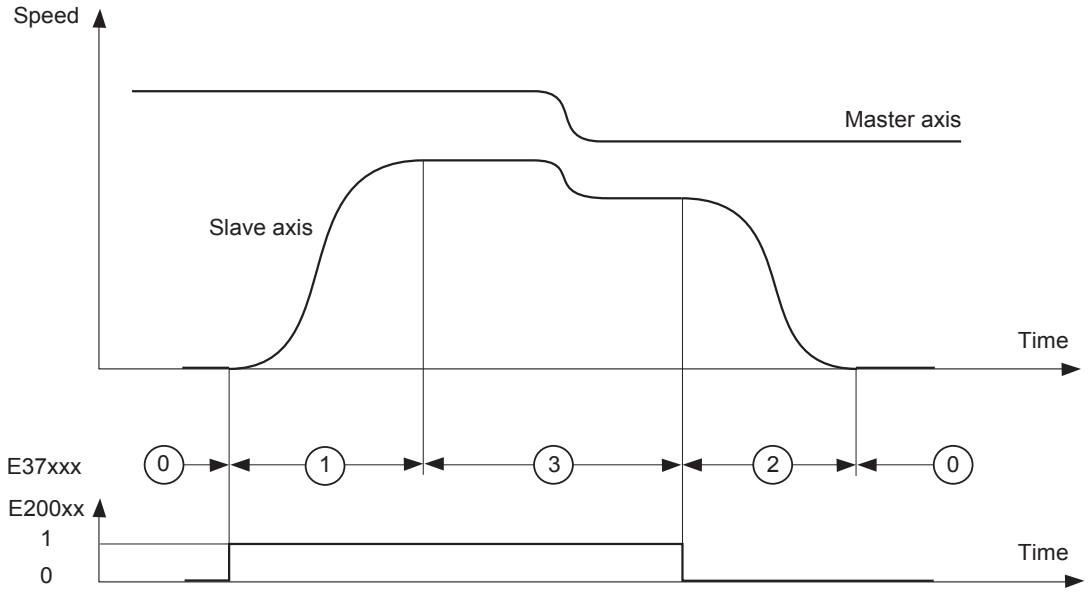
The current synchronisation state is transmitted in parameter E37xxx, as follows:
- 0: synchronisation inactive
- 1: slave axis accelerating before synchronisation
- 2: slave axis decelerating after de-synchronisation
- 3: axis synchronised (steady state).

The slave axis can be declared servo or not, but its state must not change while this operator is programmed. If it is declared servo, the operator controls its reference correction (E950xx). If not, the operator controls its reference (E7x000).

The acceleration used by the operator for starting and stopping is the second value of the pair assigned to this axis in machine parameter P32.

For E parameters detailed information please refer to the Flexium Programming manual M00018.

Description of the Operations



Examples

```
On = 20 @5/@2/E20025/E37100/192
```

- Slave axis address is 5
- Master axis address is 2
- E20025 defines the synchronisation activation command
- E37100 defines the synchronisation state.

The slave axis speed is 0.75 times the master axis speed (192/256).

12.24 Code21 ► Oscillation cycles

This operator is used to perform oscillation cycles on up to three axes.

Syntax

On = 21 @x = [TX(5,2)] / @y = [TY(5,2)] / @z = [TZ(5,2)] {/E37xxx}

@x, @y, @z	Physical addresses of the X, Y and Z axes
[TX(5,2)], [TY(5,2)], [TZ(5,2)]	Symbolic variables array
E37xxx	Type 3 external parameter

Description

For each axis, the cycle description is declared in a symbolic variable array with 2 x 5 elements.

Ti[(5,2)] =	Xa	Ta0	Ta1	Ta2	Ra
	Xr	Tr0	Tr1	Tr2	Tr

The first five elements define the forward phases of the cycle,
The next five define the return phases of the cycle.

- Cycle forward phase:
  - Xa: displacement (in mm)
  - Ta0: initial timeout used before the start of the first cycle (in ms)
  - Ta1: time of Xa movement (in ms)
  - Ta2: if Ra is different from zero, timeout between two consecutive Xa movements (in ms)
  - Ra: number of additional time the Xa movement is repeated.

- Cycle return phase: elements Xr, Tr0, Tr1, Tr2 and Tr have the same meanings but are used for the return phase.

- The sum of return movements must equal exactly the sum of forward movements.

- Tr0 is substituted for Ta0 at the beginning of the following cycles.

- The maximum timeout is 60000ms.

- The acceleration and positioning phases are carried out according to a $\sin^2$ curve. The maximum acceleration is the one defined in the second word of parameter P32 for the axis considered.

- The axis can be declared servo or not, but its state must not change while this operator is programmed:
  - if the axis is declared servo, the operator controls the reference correction (E950xx),
  - if it is declared nonservo, the operator controls its reference (E7x000).

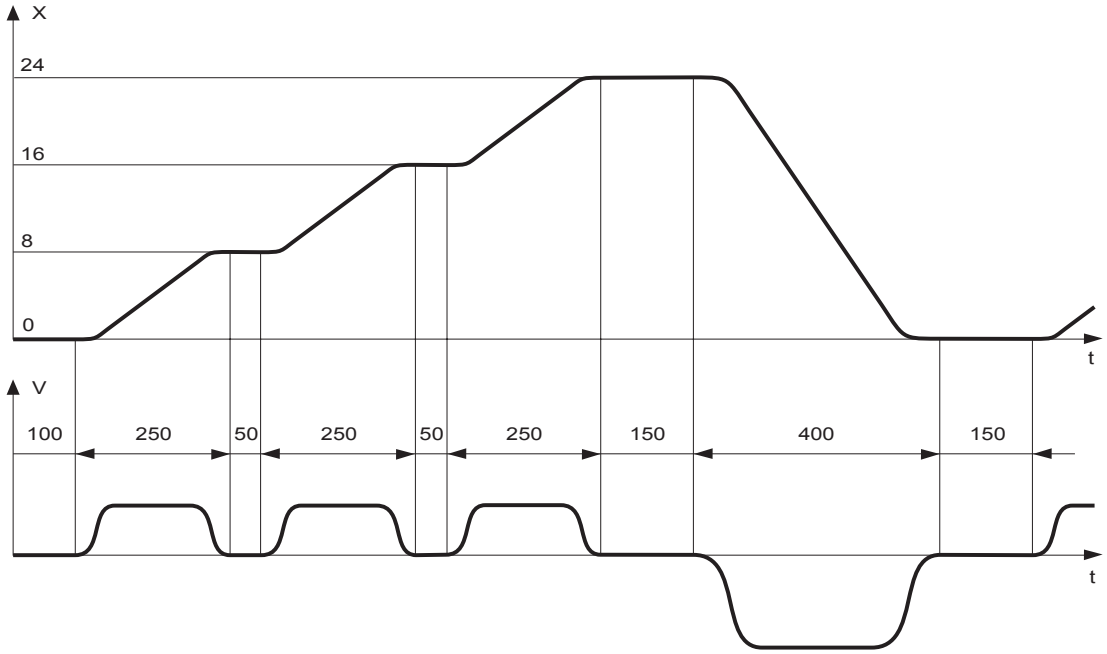
When operator 21 is cancelled by writing On = 0, each axis stops its cycle during the timeout and operator 21 is cancelled once all the axes have stopped.

Operator 21 uses auxiliary tables (one per axis) similar to those used with operators 4 (vector rotation) and 14 (arc tangent), 22 (Dynamic operators in C) and 23 (Electronic cam).

The total number of tables is limited to sixteen. If this number is exceeded, the system returns «error 93».

Example

Oscillation cycle on the X axis




```
On = 21 @x = [V1(5,2)]
```

[V1(5,2)] =	8	100	250	50	2
	-24	150	400	0	0

12.24.1 Time base modulation

The time base can be modulated by positioning an additional operand in the syntax of the operation.

This operand preceded by the character «/» (slash) is a parameter E37000 to E37127 that points to a byte whose value (between 0 and 255) is the speed override coefficient of the time base (a value of 255 corresponds to 100%).

For E parameters detailed information please refer to the Flexium Programming manual M00018.

12.25 Code22 ► Execute a C dynamic operator

Operator 22 is used to activate and run at the rate of the RTC a compiled piece of executable code. This feature requires the additional option ‘dynamic operators in C’.
This chapter does not cover C programming. It is intended for programmers already familiar with this language to explain how to create and run a C dynamic operator.

Capabilities

The dynamic operators in C function, adds new capabilities to the existing dynamic operators:

- Use of a powerful universal language
- Floating point operations
- Extended function libraries
- Loop processing
- Increase in the number of operations performed by each operator (limit of 128 for “conventional” operators)
- Capability to execute real-time computations of varying complexity.

Syntax

On = 22 COperatorName { / P1 /P2 / P3 / … / Pn }

COperatorName	Operator identifier (alphanumeric string of 1 to 13 characters)
/ P1 /P2 / P3 / … / Pn	Parameters or axis addresses separated by / (maximum 16, optional) : • Parameters Exxxxx of types 1, 2 or 3. • Axis addresses symbolized by @x

Notes

The number of parameters being passed can be checked. Refer to the section init (…,…,…)
An editable ISO block (line) is limited to 120 characters. The number of associated parameters may be limited by the length of the editable line.

Only parameters Exxxxx already accessible by conventional dynamic operators can be accessed. In particular:
- E100xx and E200xx are accessed by long words
- E300xx and E400xx are not accessible.

Like operators 4, 14, 21 and 23, operator 4 uses an auxiliary table for each call. The total number of auxiliary tables is limited to sixteen. Il this limit is exceeded error 93 will be triggered.

To add more even flexibility to Dynamic operators in C it might be required to access a large amount of data. This is possible via creating binary files as well as retrieving data in these files by their physical address in memory. Those function are not directly part of Dynamic operators in C, they are however explained at the end of this section.

Example

Calling of a dynamic operator in C named REGUPOUR (source code given in example):

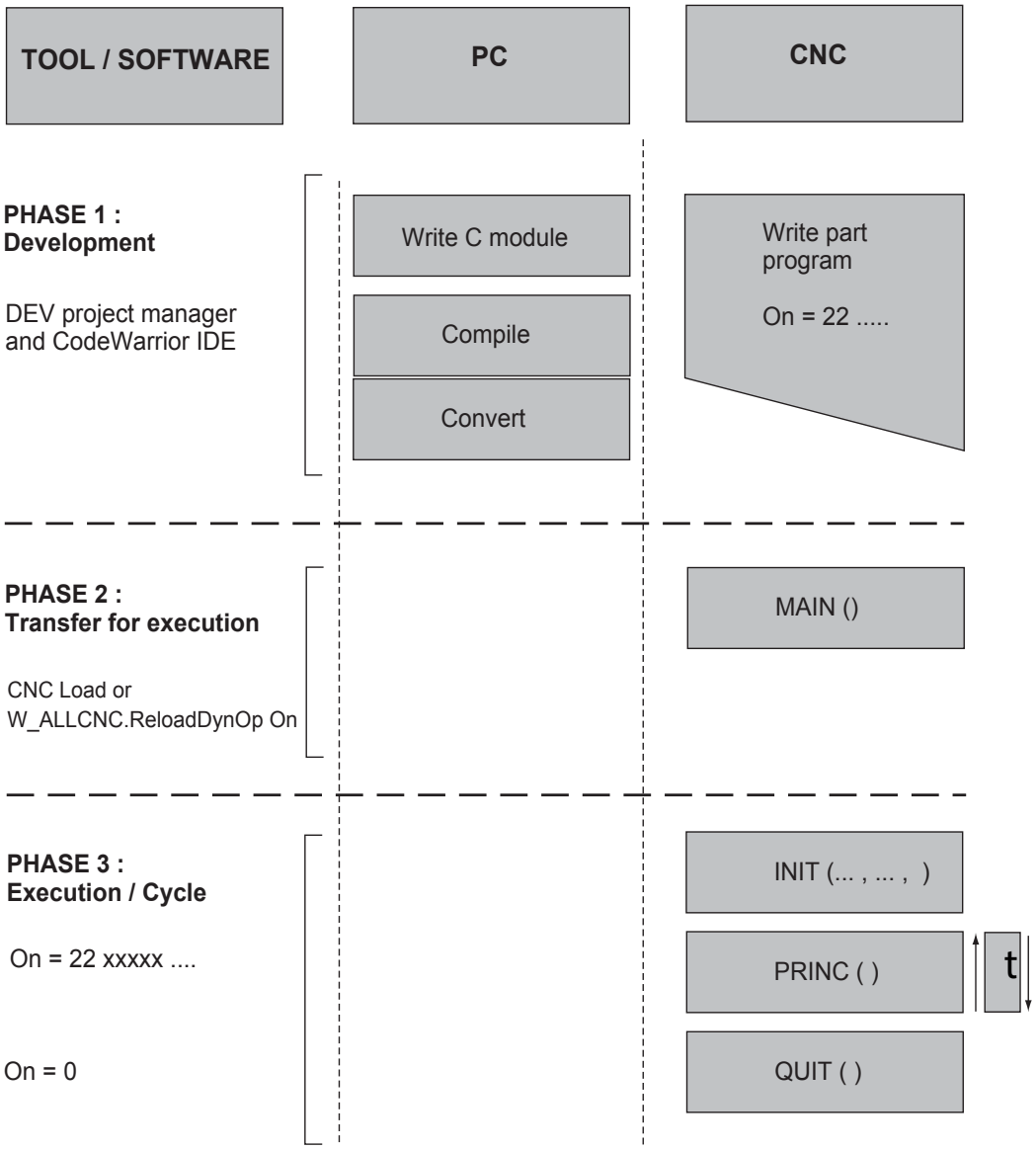
```
O1=22 REGUPOUR /E41005/E72000/E90002/E95002/E80002
```

12.25.1 Development tools

The development system is CodeWarrior Development Studio version 6.3 from Freescale.

A free version exists with some code size restrictions. If the size of your application exceeds the capability of this version, contact your authorized distributor.

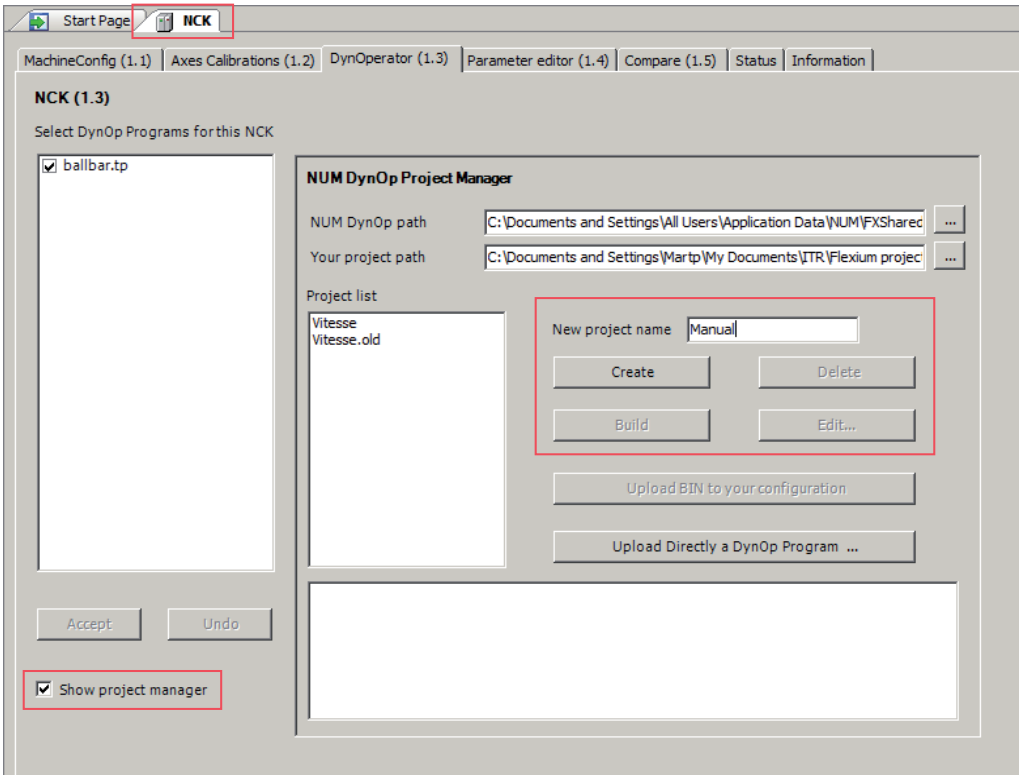
The *"NUM DynOp DEV Project Manager"* provides an interface between your project *CoDeSys* and the *CodeWarrior IDE* tools.



12.25.2 Steps for creating a project

Step 1 : Access “Show DEV Project manager “

- Within FlexiumTools, check the “Show DEV Project manager “ box in the DynOperator tab of Flexium_NCK, to show the « NUM DynOp DEV Project manager » window.



Step 2 : Create project

- Validate the path for the modules supplied by NUM (the default path is OK).
- Enter your project’s path.
- Enter your project’s name (4 characters minimum).
- Click the “Create” button.

Step 3 : Edit project

- Click the “Edit” button. For details, see “Structure of the C Modules”.

Step 4 : Edit project

- Click the “Build” button.
- Warnings and errors may be displayed in the message area.
- If no error are detected, the message “Build OK!” Is displayed.

Step 5 : Upload Bin

- Click the “Upload BIN to your configuration” button to convert the binary generated by the build, make it compatible with FLEXIUM and add it to the list of programs for this CNC.

12.25.3 Modules supplied by NUM

There are three “NUM files” (that should not be changed) and one “User file”.

“NUM Files”:

- Numopdyn.h: Contains the prototypes and user error codes.
- opdlib.a: Library which contains the interfaces and user functions.
- pic_5485.lcf: Contains compile and link information.

“USER Files”:

- opdyn_user.c:
  Example of a C file with all the data to make one dynamic operator call.
  (Includes, prototype, main, init...)

12.25.4 Structure of the C modules

Each of the following subsections are illustrated in the section Sample Program in C below.

main (void) :

The module entry point is main (). It is called during installation in the CNC memory at power on and may also be called after a reset.
Main () must declare each of the operators contained in the module to the CNC executive using the export (…,…,…) primitive supplied: export (COPERATORNAME, OPDYN, &entry_pt).

Export (…,…,…) contains three arguments:

- The name of the dynamic operator (1- to 13-character string)
- Dynamic operator version used, OPDYN for this version
- A pointer to a structure giving the three entry points of each dynamic operator (see details below in the section User Entry Points).
  The prototypes are included in the NUMOPDYN.H include module.

Main () returns zero if installation is OK.

System errors are possible and give rise to CNC error messages (for a detailed list of error messages 400 to 409, refer to the section Error Messages).

User Entry Points

Three steps are considered in execution of a dynamic operator. With each step is associated a subroutine whose entry point must be defined:

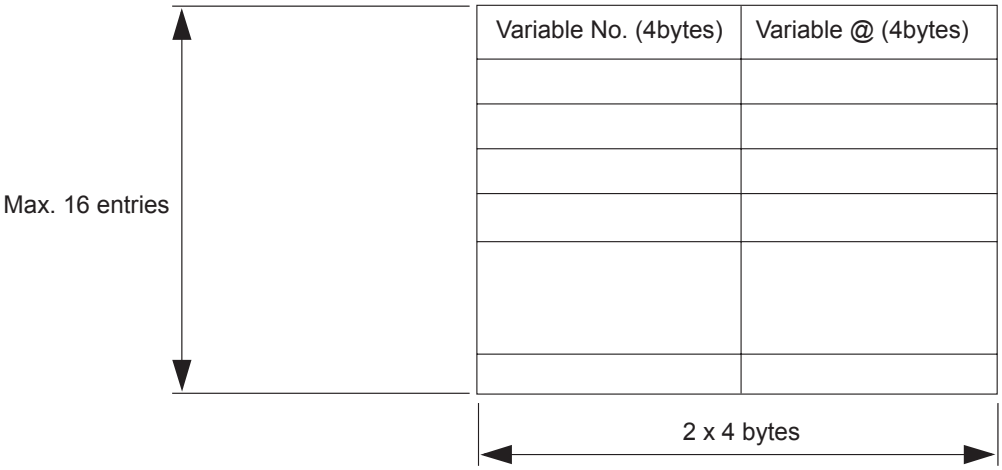
- Call and first execution: Entry point: init()
- Subsequent executions: Entry point: princ()
- Cancellation: Entry point: quitt()

12.25.4.1 init (UINT16 gr_no, UINT16 no_arg, ARGUMENT *):

This subroutine is executed once under control of the low priority RSV task during analysis of the ISO block.

The input parameters of init are:
- The channel number + 1 in which it was programmed (1 < gr_no < 8)
- The number of arguments programmed (0 < no_arg < 16)
- A pointer to the parameter table programmed in the dynamic operator in C.

The parameter table has the following format:



As it is only executed once, this subroutine is not critical from the standpoint of computation time. It can include various initialisations requiring a long computation time.

It returns an execution report:
- Zero if the part program can continue
- Negative if an error is observed.

In case of error, the part program stops and a program error is displayed.

The values which can be returned are:
- ERR_RESERV1 (-1) is reserved by the system to indicate that the dynamic operator is unrecognised
- ERR-PARAM (-3) generates error 410: Dyn. ops in C: Number of parameters passed is wrong
- ERR_UsrInit (*) (-2) generates error 411: Dyn. ops in C: USER ERROR from init.

These values are defined in the numopdyn.h include file.

12.25.4.2 princ (void)

This subroutine is executed each RTC under control of the AXE task. It is therefore critical from the standpoint of computation time. The code must be optimised and long computations must be avoided. No execution report is returned.

Local Memories

The program includes 32 Kbytes of static variables (-32 bytes used in opdlib.a) and 2000 stack bytes for auto class variables.

12.25.4.3 quitt (void)

This subroutine is executed once during cancellation of the operator by Oi = 0 or M02 or during reset of the channel where it was programmed.

It returns an execution report:

- Zero if the program can continue
- Negative if an error is observe.

In which case the part program stops and a program error is displayed.
The values which can be returned are:

- ERR_RESERV2 (-5) is reserved by the system to indicate that the dynamic operator number is unrecognised
- ERR_CLOSE (-6) generates error 420: Dyn. ops in C: USER ERROR from the quitt function
- ERR_UsrQuitt (*) (-7) generates error 421: Dyn. ops in C: USER ERROR from the quitt function

Functions init (), princ () and quitt () are executed by the CNC processor in supervisor mode. This means that no check is made for an uninitialized or incorrectly initialised pointer.

12.25.5 Limits and precautions

The code is executed under the developer's responsibility. No checks are made.

The number of files managed must be < 100.

Unless the developer takes special precautions, the code is non-re-entrant. The same operator in C should not be called under two different numbers. Use of the malloc () function is not allowed.

12.25.6 Transferring the code for execution

The modules identified as dynamic operators in C must be included in the machine project via FXTools. They are transferred into the NCK memory at power up.

12.25.7 Advanced features

To add more even flexibility to Dynamic operators in C it might be required to access a large amount of data. This is possible via the possibility of creating binary files as well as retrieving these data by their physical address in memory.

Those two functions are explained in the chapter 5.

12.25.8 Error messages

400	The size of user code is too big
401	Format error
402	Checksum error
403	The system has insufficient memory for dyn. ops in C
404	Open error
405	Read error
406	Close error
407	The directory is empty
410	Number of parameters passed doesn’t match
411	USER ERROR from Init function (negative return)
413	Unrecognised dyn. ops in C
414	NO MAIN routine
420	ERROR from the quitt function
421	USER ERROR from the quitt function: negative return
423	Too many C files (0...100)

400,401, 402,404, 405,406, 407	Errors can occur during a CNC reset or at power on
403	Error can occurs at power on
410, 414	Errors can occur when the dynamic operator is programmed
420, 423	Errors can occur when the dynamic operator is cancelled

Examples

Sample Part Program

```
%11000.2
```

(This program is called on RESET, if P7 N1 bit 3 =1)

```
Ø01 = 22 REGUPOUR/E41005/E70000/E90004/E95004/E80000
$0 Regulation of the X axis following error, in channel 2 (@4)
```

Sample Program in «C»

```
/*********************************************************************/
/*- Module name : Regupour                                          -*/
/*-                                                                 -*/
/*-                                                                 -*/
/*- Functions : DYNAMIC OPERATORS in C                              -*/
/*- Example : Lag regulation                                        -*/
/*-                                                                 -*/
/*- Created on : 05/12/2008                                         -*/
/*-------------------------------------------------------------------*/
/*- Evolutions                                                      -*/
/*-                                                                 -*/
/*- Edit date: By :                                                 -*/
/*- Nature :                                                        -*/
/*-                                                                 -*/
/*F*********************************************************************/
#include "numopdyn.h"
/*-------------------------------------------------------------------*/
/*    Prototypes of local functions: DO NOT EDIT                    -*/
/*-------------------------------------------------------------------*/

SINT32 main();
SINT32 init (UINT16 no_gr,UINT16 nb_arg,ARGUMENT *pt);  /* First call  */
void princ ();                                          /* Current call*/
SINT32 quitt ();                                        /* Last call   */


/*-------------------------------------------------------------------*/
/*    Global variables: Application specific                        -*/
/*-------------------------------------------------------------------*/

SINT32     ERPOURX, OREFX, ER_MESX, DIFREFX, P56P50, VITFIL, MVITFIL;
SINT32     *pE70000,*pE90000,*pE95000,*pE80000;

/*-------------------------------------------------------------------*/
/*    Called on installation in memory                              -*/
/*-------------------------------------------------------------------*/

SINT32 main()
{
      SINT32 ret;
      ENTRY_PT pt_entree;

      pt_entree.INIT = init;     /* Entry point for the first call      */
      pt_entree.MAIN = princ;    /* Entry point for the current call    */
      pt_entree.EXIT = quitt;    /* Entry point for the last call       */


      /*-------------------------------------------------------------*/
      /*     REGUPOUR identifier for the part program                -*/
      /*-------------------------------------------------------------*/

      ret = EXPORT("REGUPOUR","OPDYN",&pt_entree);
      return (ret);
}

/* End main() */

/*-------------------------------------------------------------------*/
/*    First call                                                    -*/
/*-------------------------------------------------------------------*/

SINT32 init(UINT16 no_gr, UINT16 nb_arg, ARGUMENT *pt)
```

```
{
     SINT32 P50;
     UINT16 no_gr_local;
     no_gr_local = no_gr;


     /* Checking the number of parameters passed */

     if (nb_arg == 5)                    /* OK => initialisation */

     {
          P50 = *(pt[0].adr_param);  /* Param 0 = Scan time in µs */
          pE70000 = pt[1].adr_param;      /* Points on axis reference */
          pE90000 = pt[2].adr_param;      /* Points on axis measure */
          pE95000 = pt[3].adr_param;      /* Points on reference correction */
          pE80000 = pt[4].adr_param;      /* Points on a generic parameter */

          OREFX =*pE70000;
          P56P50 = 60000 / P50;
          ER_MESX = MVITFIL = VITFIL = 0;

          return(ERR_AUCUNE);
     }

     else                           /* NOK => Error 410 */
     {
          return(ERR_PARAM);
     }
}

/* End init() */



/*-------------------------------------------------------------------------*/
/*    Current execution                                                    -*/
/*-------------------------------------------------------------------------*/

void princ()

{
     ERPOURX    =*pE70000 -*pE90000;
     DIFREFX    =*pE70000 -OREFX;
     OREFX      =*pE70000;
     DIFREFX    *= P56P50;
     MVITFIL    +=(DIFREFX -VITFIL);
     VITFIL     = MVITFIL / P56P50;

     ER_MESX    +=(ERPOURX -VITFIL);
     *pE95000   = ER_MESX/10;

}

/* End princ() */

/*-------------------------------------------------------------------------*/
/* Last call                                                           -*/
/*-------------------------------------------------------------------------*/

SINT32 quitt()

{
     *pE95000 =0;
     return (ERR_AUCUNE);
}

/* End quitt() */
```

12.26 Code23 ► Electronic cam

This operator informs the PLC that an axis has reached or overrun a position.

Syntax

On = 23 @<axis> =<dim>/<dim>/<dim> @<axis> =<dim>/<dim>/.... @<axis> =<dim>/<dim> E37xxx

@<axis>	Physical axis address
<dim>	Dimension in internal units
E37xxx	External parameter of type 3

Description

- @<axis> is the address (0 to 31) of a machine axis.
- The axis number can be specified as an immediate value, a variable L or a parameter E.
- The number of physical axes is limited to 3.

- The number of dimensions required for each axis ranges from 2 (minimum) to 32 (maximum).
- The value of <dim> can be specified as an immediate value, a variable L or a parameter E.
- The dimensions <dim> must be specified in ascending order.

- E37xxx denotes a parameter from E37000 to E37127 reflecting the axis position.
- The value sent to the PLC is contained in E37xxx for the first axis, E37xxx+1 for the second axis and E37xxx+2 for the third axis.

The electronic cam function uses auxiliary tables (one per axis processed) similar to those used for operators 4, 14 and 21, 22 and 23 (vector rotation, arc tangent, oscillation cycles, dynamic operators in C and electronic cam respectively). The total number of tables is limited to sixteen. If this number is exceeded, the system returns «error 93».

If the current position of the axis is strictly below the lowest dimension, a value of zero is passed to the PLC.

For E parameters detailed information please refer to the Flexium Programming manual M00018.

Example

```
On = 23 @0=150000/155000/200000/300000 @1=500000/600000/620000 /E37005
```

- E37005 can contain the values 0, 1, 2, 3, 4 according to the position of axis 0
- E37006 can contain the values 0, 1, 2, 3 according to the position of axis 1

12.27 Code24 ► Differential shift

Dynamic operator 24 is used to set differential datum shifts on up to 9 axes by writing parameters E953xx and specifying the hand-wheel used.

Whether or not the machine is in a cycle, stopped or moving, it is be possible to move the axes with a hand-wheel if authorised by a PLC flag (`Flexium_NCK.W_ALLCNC.DifferentialAxisOffset`)

The datum shifts obtained must be preserved and added to the programmed or manual movements (they are cancelled only by homing) without affecting the machining parameters.

Processing

The differential datum shift is processed as a measurement offset added to the existing offsets, axis and interaxis calibration by machine parameter, backlash error compensation and offset by the PLC.

The differential datum shift is stored in the external parameter E953xx (xx = physical axis address). This parameter can only be written by the dynamic operator. Its value is added to the other measurement offsets in E952xx.

Syntax

On = 24 @a / @a / @ a / ... / Wm {/ * }

@ai	Physical address of the axis to be controlled (1 to 9 axes)
Wm	Hand-wheel number (0 to 3)
*	Datum shift of slave, duplicated or synchronised axes (see Notes)

Notes

- Error 93 indicates absence of the selected hand-wheel or a measured axis or presence of one of the handwheel coefficients set to zero.
- When variable `Flexium_NCK.W_ALLCNC.DifferentialAxisOffset = 1`, dynamic operator 24 takes the handwheel variations into account with the conversion coefficients of P13.
- The hand-wheel variations are then assigned to the E953xx of the axis assigned (by the PLC) to this handwheel. This assignation is only made for axes specified in the operator
- These increments are assigned immediately (without management of the acceleration).
- In Jog mode using the hand-wheel, it is recommended to reset `Flexium_NCK.W_ALLCNC.DifferentialAxisOffset = 1` so as not to add movements and differential shifts
- Monitoring of the machine travels does not take the differential datum shifts into account.
- The value of a differential datum shift is limited to +/- 32000 measurement increments.

Special Case of Slave, Duplicated or Synchronised Axes

In order to apply the datum shift to the slave, duplicated or synchronised axes at the same time as to the master axis, a variant of dynamic operator 24 consists of adding an argument to its syntax.

The star character.
Then datum shifts addressed to the master by the PLC are also assigned to the slave axes specified in the list of axis arguments in forward or reverse mode.

For instance, if axes 2 and 3 are slaves of axis 0:

```
On = 24 @a0 / @a1 / @ a2 /@ a3/ Wm /*
```

When the PLC addresses axis 0 to assign it to hand wheel Wm, it also assigns axes 2 and 3 to the same handwheel. However, if it addresses axis 2 or 3, only the slave axis is assigned to the handwheel.

When a master axis controls slaves, the check of the limit values is made only on the master, but not on the slaves, which may in this case have datum shifts which exceed the limit of +/- 32000.

12.28 Examples of use

12.28.1 Example 1

This example is defined for a machine with axes Y, Z and B. The machining to be performed on the part is an interpolation between the B and Y axes (this example can also be processed using interaxis calibration).
The part program is written for the U and Y axes.

The U axis is a dummy axis declared in the following machine parameters:
- P0 (axes displayed),
- P9 (axis assignment to a channel),
- P3 (servoed and interpolated axes).

The U axis is not declared in parameter P2 (measured axis).

Variables and Parameters Used in the Program
- L10: program variable
- L917: program variable defining the radius of the circle
- E95007: B axis correction (physical address 7)
- E73000: position reference of an axis in the channel (logical address 3 for the U axis).

For E parameters detailed information please refer to the Flexium Programming manual M00018.

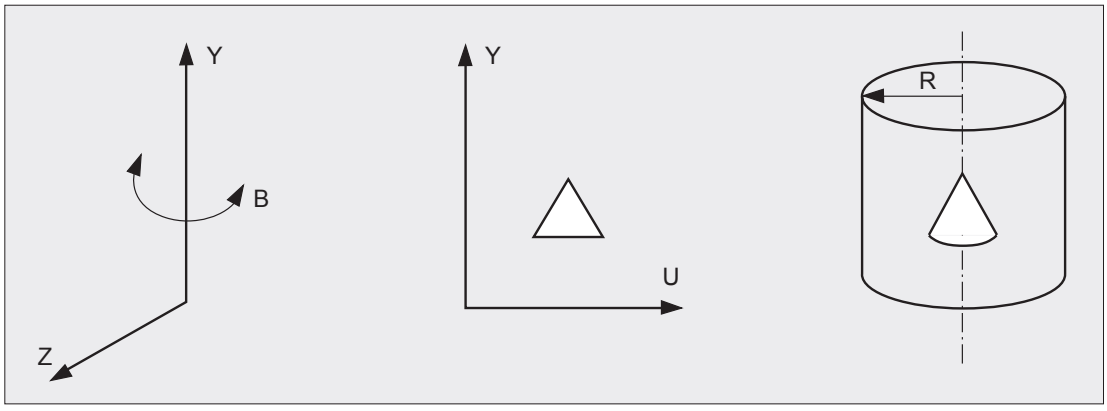
Dynamic Operator Used
- Operator 13: multiplication by a program variable.

Main Program

```
%1
R = 50                                  Cylinder radius
G77 H100                                Subroutine call
...
...
...                                     Description of the profile with Y and U
...
...
M02                                     End of program
```

Subroutine

```
%100
L10 = 180/L917/3.1415926
O1 = 13 E95007/E73000/L10               In the operation, E95007 = B axis correction
                                        and E73000 = interpolated position of the U axis
```



12.28.2 Example 2

The example below describes a program using dynamic operators to define the inclination of the X and Z axes with respect to the program zero reference.

```
%50
N10  (ENABLED)
E81002=E72000-E62000  E81000=E70000-E60000
E82002=1-CL900*E81002  E82002=-E81000*SL900+E82002
E82000=1-CL900*E81000  E82000=E81002*SL900+E82000
(OFFSET INITIALISED)
O5= 5 E95002/E82002  O8= 5 E95000/E82000
(ROTATION MATRIX COEFFICIENT COMPUTED)
E81100= CL900-1*65536*16384  E81101=SL900*65536*16384

(ESTABLISHMENT OF THE ROTATION)
O1= 2 E82002/E72000/E62000/2
O2= 2 E81000/E70000/E60000/2
O3= 3 E81111/E81002/E81100/-32
O4= 3 E82222/E81000/E81101/-32
O5= 2 E95002/E81111/E82222
O6= 3 E81111/E81000/E81100/-32
O7= 3 E82222/E81002/E81101/-32
O8= 1 E95000/E81111/E22222

N20 (MACHINING)
N.. (DISABLED)
E82000=E95000  E82002=E82000
O5= 5 E95002/E82002  O8= 5 E95000/E82000

O1=0 O2=0 O3=0 O4=0 O5=0 O6=0 O7=0 O8=0
N..
```

C X = f (Delta X ')
Z Z'

Shift 30 bits left because
the value is equal to 1 at most

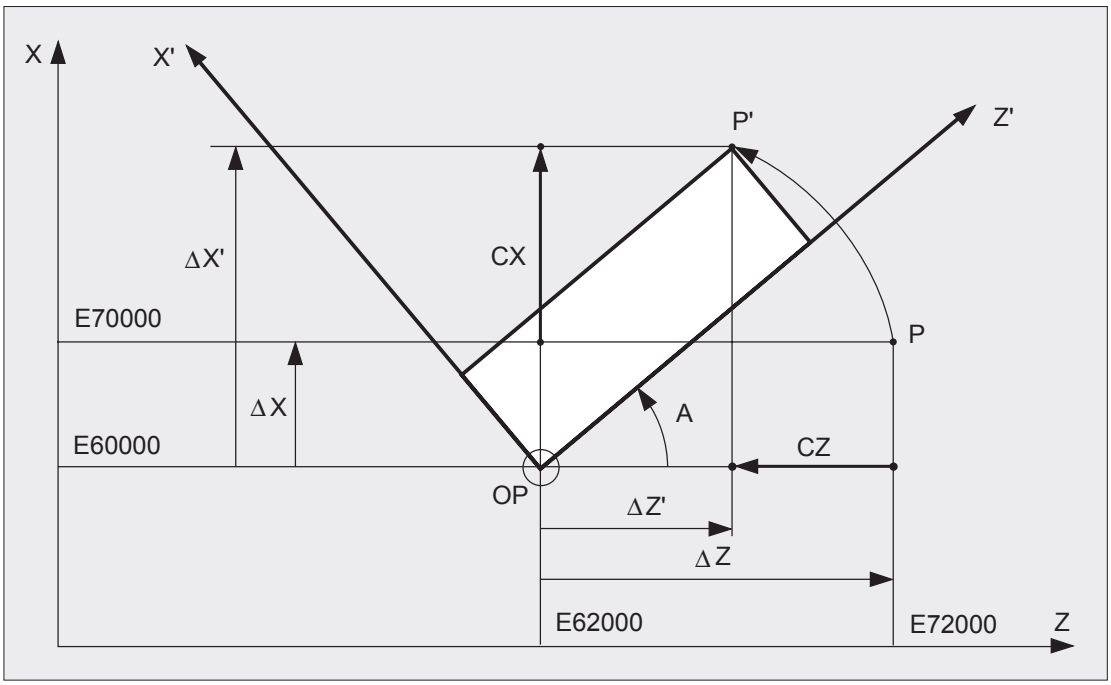
Delta Z
Delta X

$CZ = f\left(\text{Delta } \frac{X}{Z}\right)$

$CX = f\left(\text{Delta } \frac{X}{Z}\right)$

Program or subroutine
CZ= Delta Z' (1-Cos A)-Delta X' Sin A
CX= Delta X' (1-Cos A)-Delta Z Sin A
CZ= Delta Z (Cos A-1)+Delta X Sin A
CX= Delta X (Cos A-1)-Delta Z Sin A

Description of the Inclination on the X and Z Axes



12.28.3 Example 3

The example below describes use of operator 21 in a scan cycle preparation subroutine %10136 called from the main subroutine (for instance by function G136 calling the subroutine by a G function). The cycle is performed in the XY plane.

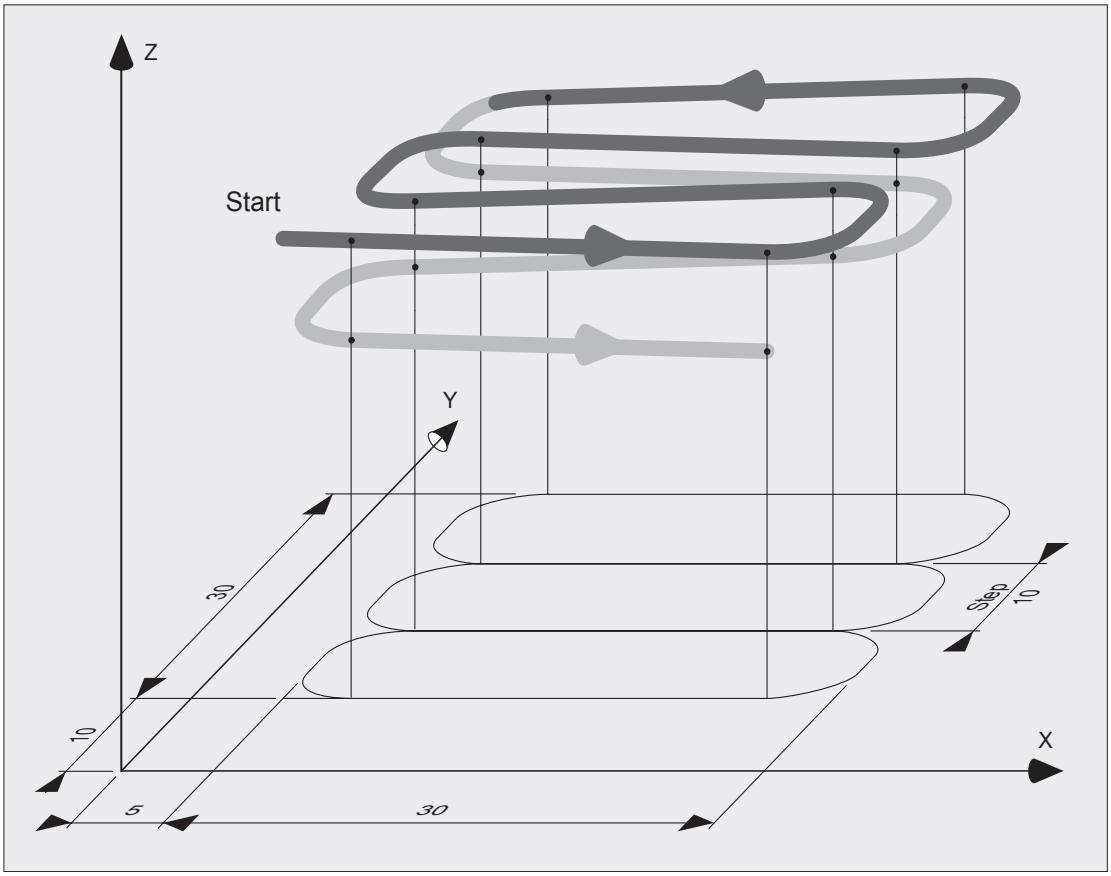
Call of Cycle G136 in a Part Program

```
%100
...
G1 X5 Y10 Z10                    Scan cycle start point
G136 X30 Y30 Q10 F1500           Cycle, position of the end point (XY) and step (Q)
Z5 F10...                        Depth of cut
```

Subroutine

```
%10136: (Scan cycle)
VAR [ix]=1 [iy]=2 [tx(5,2)] [ty(5,2)] [vx(2)] [V] [Fp]
ENDV
[vx(1)] = [.IRX(1)] [V]=[..IRX(1)] X[V]
[vx(2)] = [.IRX(2)] [V]=[..IRX(2)] Y[V]
IF [.IBP(1)] = 0 THEN [ix]=2 [iy]=1 G79 [.IBP(2)]=0 N91
ENDI
IF [.IBE0(6)] = 1 THEN [Fp]=L905 (F.. programmed in cycle call block)
ELSE [Fp]=[.RF]
ENDI

(Position the dimensions and number of measurements in TX and TY)
[ty(1,1)]=[vx(iy)] [ty(1,2)]=-[vx(iy)]
[V]=[vx(ix)]/[.IRP(ix)]+.5 [V]=T[V] G79 [V]=0 N91
[tx(5,1)]=[V]-1 [tx(5,2)]=[V]-1
[V]=[vx(ix)]/[V] [tx(1,1)]=[V] [tx(1,2)]=-[V]

(Calculate the scan times on IY)
(The timeout separating two scans is 10 ms)
[V]=[vx(iy)]/[Fp]*60000 [ty(3,1)=T[V] [ty(3,2)=T[V] [ty(2,2)]=100
(and on IX)
[V]=[tx(1,1)]/[Fp]*60000/2 [V]=T[V] [tx(3,1)=[V]*2 [tx(3,2)]=[V]*2
[tx(2,1)]=[ty(3,1)]+50-[V] [V]=[ty(3,2)]+100-[V]-[V]
[tx(4,1)]=[V] [tx(2,2)]=[V] [tx(4,2)]=[V]

(Get the axis addresses of IX and IY)
[V]=[ix]-1*1000+70005 [ix]=E[V] [V]=[iy]-1*1000+70005 [iy]=E[V]

(Get an available operation number)
[V]=49000
REPEAT [V]=[V]+1
UNTIL E[V] = 0

(Create the oscillation operator)
[V]=[V]-49000 O[V]= 21 @[ix]=[tx(5,2)] / @[iy]=[ty(5,2)] G79 N99
N91 E.636
N99 G80 G997
```

Description of the Scan Cycle



Page deliberately empty

Summary

Page deliberately empty

13 Application specific E parameters

13.1 General

This chapter is intended to provide explanations on specific functions triggered by E parameters that are not necessarily part of the machine primary definition and can be triggered during the life of the machine.

Other functions closely related to the machine structure (e.g. spindle/ axes swapping, spindle synchronisation) are described in detail in the Flexium Commissioning manual M00010 together with more details about the following.

13.2 Continuous probing E11019

External parameter E11019 is used for continuous probing. When this parameter is set, each interrupt generated on the NCK will trigger the recording of axes position in E7x001 in a similar way as the G10 function.

The difference with the G10 function is that the interrupt has no effect on block execution (the movement continues) and that the function remains active as long as E11019 is set.

Taking into effect the recording of the axes position should be done via Dynamic operators by watching the change in the values of E7x001 parameters.

For E parameters detailed information please refer to the Flexium Programming manual M00018.

13.3 Enabling reference writing for Virtual axes E11020

This parameter can be used for very specific operations.

Generally presetting the reference for a standard axis is made by temporary declaring this axis as non servo which is obviously not possible for Virtual axes.

As for a regular axis, writing the reference requires the axis to be non interpolable E910xx = 0.
Inclined plane and RTCP should be suspended.

Writing the reference for virtaul axes is done in steps:

```
E910xx  = 0
E11020  = 1
E900xx  = ....      Presetting the measure will in turn preset the reference
E11020 = 0
E910xx = 1          Back to normal
```

E11020 is cleared (=0) at power up but unaffected by CNC Reset.
It is valid for all axes of all groups.

For E parameters detailed information please refer to the Flexium Programming manual M00018.

13.4 Inhibiting program modification and deletion 31002

Parameter E31002 is intended to prevent program modification in CNC memory when set.

This parameter is automatically reset at Power up and after an CNC reset.

The main purpose of this function is to prevent changing the address of a binary file that can be used by a dynamic operator application.

This does not prevent to edit the program via Flexium HMI, however, program transfer into the CNC memory will not be possible until Reset.

For E parameters detailed information please refer to the Flexium Programming manual M00018.

13.5 Circle tolerance E32012

In circular interpolation, the system allows a certain difference between the radius defined by start point and centre and the radius defined by end point and centre.

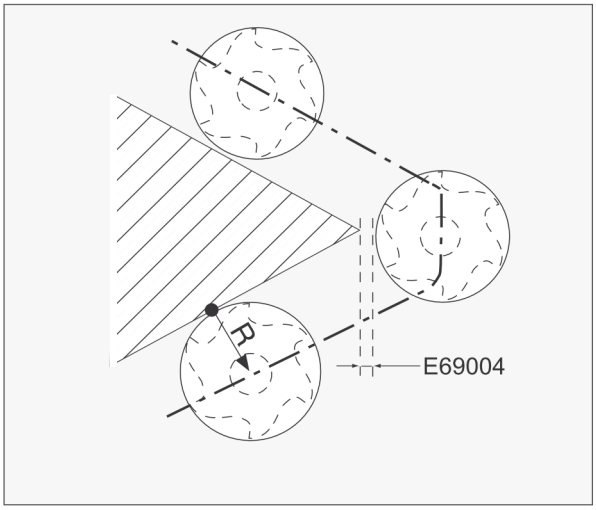
By default the difference is limited to 20 increments. It is defined by a machine parameter but can be edited via E32012. The value for this parameter should be between 20 and 255 increments.
A reset restores the value defined in the machine parameter.

For E parameters detailed information please refer to the Flexium Programming manual M00018.

13.6 Tool disengagement in line to line transition E69004

When in cutter compensation mode (G41/G42, see Flexium Programming manual M00018) in a line to line transition, for an angle bigger than 120°, the tool will turn around the angle while staying in contact with the workpiece, which may cause damage in some cases.

To overcome this situation, the parameter E69004 can be used to define a disengagement distance. E69004 is valid for the channel in which it is programmed. It must be programmed in G40 status, and its value is automatically limited to 0,4 R (R: Tool radius).



13.7 Axis duplication and synchronisation E941.., E96...

Axes duplication and synchronisation is possible via machine parameters (permanent) or E parameters to get additional flexibility. Complete information for axes synchronisation is defined in the Flexium Commissioning manual M00010. This chapter just gives a brief overview.

Coupling definition

This is defined via parameter E941.. which precise for each axis what is the master axis if any.

ex : E94102 = 7 : Axis number 2 is slave of axis number 7

E94105 = FF : Axis number 5 is not linkable to any other axis.

ex : E96102 = 1 Coupling of axis 2 is active in manual operation

Coupling validation in programmed operation (Aut, Seq, Mdi)

This is defined via parameter E960.. (32 parameters)

ex : E96002 = 1 Coupling of axis 2 is active in programmed operation

Coupling validation in manual operation (Home, Jog, Intervention)

This is defined via parameter E961.. (32 parameters)

Synchronisation definition

Defined via parameters E962.. (32 parameters)

ex : E96202 = 1 Axis number 2 is synchronised with its master.
This means the servo mechanism will compensate for each discrepancy between the position of slave and master. If Synchronisation is not active then axes are just duplicated, they follow their master.

Symetry definition

Defined via parameters E963.. (32 parameters)

ex : E96302 = 1 Axis number 2 will move in symetry relative to its master.

An axis can not be simultaneously synchronised and symmetrical.

For E parameters detailed information please refer to the Flexium Programming manual M00018.

13.8 Monitoring drives test points E351.., E352..

It is possible to read in real time (CNC RTC time, P50) the value of two measure points for each drive. The measure points can return values like current, motor speed etc…
The list of the available measure points could be found into the NUMDrive C parameter manual AMOMAN007 and into the Flexium Extended NCK Access Manual M00026.

The value stored into the E parameters are based on drive internal units, so to convert them into physical units (Amps, rpm, etc.) a conversion coefficient must be applied. Rules and conversion coefficients are given into the the Extended NCK Access Manual M00026.

The measure points to read are defined by the drive parameters V130 and V131 and can be accessed via external parameters. It is possible to return two 16 bits values or one 32 bits value.

In case of selection of two 32 bit measure points or one 16 bit and one 32 bit, priority is given to the first 32 bit measure points. Reading can be done via dynamic operators or parametric programming.

This function relies on the following set of E parameters:

E352(64+xx)	0/1 Disable / Enable reading of measure points for drive xx
E352xx	A : Measure point number A stored into V130 for drive xx
E352(32+xx)	B : Measure point number B stored into V131 for drive xx
E351xx	16 bits value of measure points selected by V130 for drive xx
E351(32+xx)	16 bits value of measure points selected by V131 for drive yy
E351(64+xx)	32 bits value of measure points selected by V130 or V131

Depending of the format of the returned values, we have the following situation:

Value requested by V130 - E352xx	Value requested by V131 - E352(32+xx)	E351xx	E351(32+xx)	E351(64+xx)
A (32 bit)	**B** (32 bit)			**A**
A (32 bit)	**B** (16 bit)			**A**
A (16 bit)	**B** (32 bit)			**B**
A (16 bit)	**B** (16 bit)	**A**	**B**	
A (16bit)	**A** (16 bit)	$\mathbf{A}_{(Tn)}$	$\mathbf{A}_{(Tn-1 + RTC/2)}$	
Legenda. A_{Tn} = Measure point value stored on time Tn. $A_{(Tn-1 + RTC/2)}$ = Measure point value stored on time in between Tn and Tn-1.				

For E parameters detailed information please refer to the Flexium Programming manual M00018.

13.9 Swapping drive parameters E353...

For applications where the kinematic response might change according to the situation (Swappable axes, Extra load…) it is possible to select the drive parameters between two sets of predefined values. As this operation is possible on the fly obviously all precautions should be taken to prevent mechanical or human risks.

Programming

Programming is as follows:

E353xx = yz

- xx : Drive number 0 to 31
- y : Position loop parameter set (0/1) equivalent to V261 in the drive
- z : Speed loop parameter set (0/1) equivalent to V260 in the drive.

Position loop

The position loop parameter set 0 is made of:

- V301, V305, V306, V307, V310, V311, V312, V313 and V353.

The set 1 contains the same variable list prefixed by 1 (V1301, V1305 …).

Speed loop

The speed loop parameter set 0 is made of:

- V45, V46, V50, V51, V52, V53, V60, V91, V93, V94,V95, V96, V97, V98, V99.

The set 1 contains the same variable list prefixed by 10 (V1046, V1097 …).

Please refer to the NUM Drive C Parameters manual AMOMAN007 for more detailed informations.

Page deliberately empty

Summary

Page deliberately empty

A Appendix

A.1 Structured programming commands

EXIT	Exit from a loop
Syntax:	**EXIT**

FOR	Loops with control variable
Syntax:	**FOR** (variable) = (expression 1) TO/DOWNTO (expression 2) BY (value) DO (instructions) ENDF

IF	Conditional execution of instructions

Syntax:	**IF** (condition) THEN
	(instructions 1)
	ELSE
	(instructions 2)
	ENDI

REPEAT	REPEAT UNTIL loops
Syntax:	**REPEAT** (instructions) UNTIL (condition)

WHILE	WHILE loops
Syntax:	**WHILE** (condition) DO (instructions) ENDW

A.2 Symbolic variable management commands

BUILD	Table creation to store a profile
Syntax:	**BUILD** [TAB(G/X/Y/I/J,NB)] H… N… +nN… +n

BCLR:	Inhibits activation of an M code or axes movements
Syntax:	`BCLR [.BMxx] / [.IBX(i)] / [.IBE0(i)] / [.IBE1(i)]`

BSET	Activates an M code or axes movements
Syntax:	**BSET** [.BMxx] / [.IBX(i)] / [.IBE0(i)] / [.IBE1(i)]

CUT	Eliminates grooves with the tool relief angle
Syntax:	**CUT *** [Pa(7,NB)] / Angle

MOVE	Copy all or part of a table into an other
Syntax:	**MOVE** [Pj(nj,mj)],mj1,mj2 = [Pi(ni,mi)],mi1,mi / j1=i1 /j2=i2 / jn=in

PBUILD	Create a table for storing the dimensions of a profile
Syntax:	**PBUILD** [TAB(7,NB)] H… N… +nN… +n

R.OFF	Normal offset to an open profile
Syntax:	**R.OFF** [Pa(7,Nb)] / ±1 / R

SAVE	Saves of a list of symbolic variables declared in any subroutine
Syntax:	**SAVE** [Symb1] / [Symb2] ...

SEARCH	Search for a symbolic variable in the stack
Syntax:	SEARCH [Symb] N..

A.3 Program status access symbols

Access Symbols to the Data of the Current Block or Previous Block.
Only the symbols to access the data of the current block are listed in the table.

The symbols to access 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.

A.3.1 G and M functions

[.BGxx]	G function addressing
[.BMxx]	M function addressing

A.3.2 List of bits

[.IBI(i)]	I, J and K arguments programmed in the current block
[.IBP(i)]	P, Q and R arguments programmed in the current block
[.IBX(i)]	Axes programmed in the current block
[.IBX1(i)]	Axes programmed from the beginning of the program to the current block
[.IBX2(i)]	Last axes programmed among XYZ or UVW
[.IBXM(i)]	Mirroring or not of the axes

A.3.3 Values

[.RD]	Tool correction number
[.RDX]	Tool axis orientation (G16)
[.REC]	Spindle orientation
[.RED]	Angular offset
[.RF]	Feed rate
[.RG4]	Programmed dwell (G04 F..)
[.RG80]	G code value calling the subroutine
[.RN]	Number of the last sequence (block) found
[.RNC]	Value of CNC function
[.RS]	Spindle speed
[.RT]	Tool number
[.RXH]	Current subroutine nesting level
[.IRDI(i)]	Value of programmed angular offsets (G59 I.. J.. K..)
[.IRH(i)]	Current program or subroutine number
[.IRI(i)]	I, J and K arguments values
[.IRP(i)]	P, Q and R arguments values
[.IRTX(i)]	Offsets programmed on the axes
[.IRX(i)]	Dimensions programmed on the axes

A.4 Symbols stored in L900 to L951 variables

A.4.1 Symbols stored in variables L900 to L925

[.IBE0(4)]	L903	D
[.IBE0(6)]	L905	F
[.IBE0(8)]	L907	H
[.IBE0(14)]	L913	N
[.IBE0(15)]	L914	N
[.IBE0(19)]	L918	S
[.IBE0(20)]	L919	T

A.4.2 Symbols stored in variables L926 to L951

[.IBE1(1)]	L926	EA
[.IBE1(2)]	L927	EB
[.IBE1(3)]	L928	EC
[.IBE1(4)]	L929	ED
[.IBE1(5)]	L930	EE
[.IBE1(6)]	L931	EF
[.IBE1(7)]	L932	EG
[.IBE1(8)]	L933	EH
[.IBE1(9)]	L934	EI
[.IBE1(10)]	L935	EJ
[.IBE1(11)]	L936	EK
[.IBE1(12)]	L937	EL
[.IBE1(13)]	L938	EM
[.IBE1(14)]	L939	EN
[.IBE1(15)]	L940	EO
[.IBE1(16)]	L941	EP
[.IBE1(17)]	L942	EQ
[.IBE1(18)]	L943	ER
[.IBE1(19)]	L944	ES
[.IBE1(20)]	L945	ET
[.IBE1(21)]	L946	EU
[.IBE1(22)]	L947	EV
[.IBE1(23)]	L948	EW
[.IBE1(24)]	L949	EX
[.IBE1(25)]	L950	EY
[.IBE1(26)]	L951	EZ

A.5 List of Dynamic Operations

Cancel an operation (see chapter 12.1.4)	
Syntax:	On = **0**

Add (see chapter 12.4)	
Syntax:	On = **1** Ea / Eb / Ec {/I}

Subtract (see chapter 12.5)	
Syntax:	On = **2** Ea / Eb / Ec {/I}I

Multiply (see chapter 12.6)	
Syntax:	On = **3** Ea / Eb / Ec {/I}

Rotate a vector (see chapter 12.7)	
Syntax:	On = **4** Ea / Eb / Ec / I

Load a parameter with or without mask (see chapter 12.8)	
Syntax:	On = **5** Ea / Eb {/Ec} {/I}

Load a peripheral memory with or without mask (see chapter 12.9) **Load from a machine axis number** (see chapter 12.9.1)	
Syntax:	On = **6** Ea / @.. {/Eb} {/I}

Load from a PLC data (see chapter 12.9.2)	
Syntax:	On = **6** Ea / Eb {/Ec} {/I}

Indexed load of a parameter (see chapter 12.10)	
Syntax:	On = **7** Ea / Eb / Ec / I

Indexed storage of a parameter (see chapter 12.11)	
Syntax:	On = **8** Ea / Eb / Ec / I

Compare two parameters (see chapter 12.12)	
Syntax:	On = **9** Ea / Eb / Ia {/Ib {/Ic}}

Conditions on the sum of two parameters (see chapter 12.13)	
Syntax:	On = **10** Ea / Eb / Ec / I On = **10** Ea / Eb / Ec / E37xxx

Store data in a peripheral memory (see chapter 12.14) **Store to a machine axis number** (see chapter 12.14.1)	
Syntax:	On = **11** @.. / E

Store to a PLC data memory (see chapter 12.14.2)	
Syntax:	On = **11** Ea / Eb / I

Increment with modulo (see chapter 12.15)	
Syntax:	On = **12** Ea / Eb / I

Multiply by a program variable (see chapter 12.16)	
Syntax:	**13** Ea {/Eb} / L..

Arc tangent function (see chapter 12.17)	
Syntax:	On = **14** Ea / Eb / Ec / I

Axes manual control (see chapter 12.18)	
Syntax:	**15** @.. / E37xxx / Eb / I

Third-degree polynomial and its derivative (see chapter 12.19)	
Syntax:	**16** Ea / Eb / Lxx {/*}

Speed on trajectory (see chapter 12.20)	
Syntax:	**17** Ea / @.. {/@..} {/@..}

Divide (see chapter 12.21)	
Syntax:	On = **18** Ea / Eb / Ec {/I}

Square root (see chapter 12.22)	
Syntax:	On = **19** Ea / Eb / I}

Axis speed synchronisation (see chapter 12.23)	
Syntax:	On = **20** @e / @m / E200xx / E37xxx / I

Oscillation cycles (see chapter 12.24)	
Syntax:	On = **21** @x = [TX(5,2)] / @y = [TY(5,2)] / @z = [TZ(5,2)] {/E37xxx}

Execute a C dynamic operator (see chapter 12.25)	
Syntax:	On = **22** COperateurName { / P_1 / P_2 / P_3 / ... / P_n }

Electronic cam (see chapter 12.26)	
Syntax:	On = **23** @<axis> =<dim>/<dim>/<dim> @<axe> =<dim>/<dim>/.... @<axis> =<dim>/<dim> E37xxx

Differential shift (see chapter 12.27)	
Syntax:	On = **24** @a_0 / @a_1 / @ a_i / .../ Wm {/*}

Page deliberately empty

Symbol

A

B

C

D

E

O

P

Q

R

S

T

U

V

W

X

Y

Z

Published by NUM Spa

Via F. Somma 62
20012 Cuggiono (MI) (Italy)
Tel. : +39 02 979 691
website: www.num.com

Printed May 2013

CNC HighEnd Applications