

Extended Programming manual

(for software release 4.2.0.0)

M00055EN-05

Edition 10.2022



flexium+

www.num.com



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.

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

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

Summary

1	Foreword	13
1.1	Use of the manual.....	13
1.2	Use of symbols in the manual.....	13
1.3	Document revisions	14
1.4	How to read the manual.....	15
1.5	Syntax conventions.....	16
1.6	Relations with the PLC.....	17
1.7	NUM Agencies	17
2	Structured Programming	21
2.1	General	21
2.2	Commands used in structured sequences	22
2.2.1	General syntax rules	22
2.2.2	Nesting and branches	23
2.3	Structured programming commands	24
2.3.1	Condition graph	24
2.3.2	Conditional Execution : ▶ IF, THEN, ELSE, ENDI	25
2.3.3	REPEAT Loop : ▶ REPEAT, UNTIL	26
2.3.4	WHILE Loop : ▶ WHILE, DO, ENDW	27
2.3.5	Loops with control variable : ▶ FOR, TO/DOWNT, BY, DO, ENDF	28
2.3.6	Selective jump : ▶ SWITCH, CASE, BREAK, ENDS	29
2.3.7	Next iteration : ▶ CONT	31
2.3.8	Exiting the Loop : ▶ EXIT	32
2.3.9	Example of Structured Programming	33
3	Reading the program status access symbols	39
3.1	General	39
3.2	Symbols addressing a current Boolean value	40
3.2.1	Active G Functions	40
3.2.2	Active M Functions.....	40
3.3	Symbols addressing a list of current status bit	43
3.3.1	Programmed addresses	43
3.3.2	I, J and K arguments programmed	43
3.3.3	P, Q and R arguments programmed	44
3.3.4	Axes programmed in the current block.....	44
3.3.5	Axes programmed since the beginning of the program	44
3.3.6	Primary or Secondary axes programmed	45
3.3.7	Mirrored axes.....	45
3.4	Symbols addressing current numerical values	46
3.4.1	Feed rate	46
3.4.2	Spindle speed (for the main spindle of the channel only)	46
3.4.3	Tool number	46
3.4.4	Tool correction number	46
3.4.5	Angle of block	46
3.4.6	Line number	46
3.4.7	Number of the last block encountered	47
3.4.8	Angular offset	47
3.4.9	Spindle orientation (milling)	47
3.4.10	Dwell time programmed (G04 F..)	47
3.4.11	Canned cycle number	47
3.4.12	Spline curve number value	47

3.4.13	Tool axis orientation	48
3.4.14	Nesting level of the current subroutine	48
3.4.15	List of axes programmed in the block	48
3.5	Symbols addressing a list of current numerical values	49
3.5.1	Spindle programmed	49
3.5.2	Spindle Index angle	49
3.5.3	Value of abstract angles	49
3.5.4	Axes dimensions programmed	49
3.5.5	Offsets values programmed.....	50
3.5.6	I, J and K values programmed.....	50
3.5.7	P, Q and R. values programmed.....	50
3.5.8	Subroutine number	51
3.5.9	Programmed angular offsets origin (G59 I.. J.. K..)	51
3.5.10	Spindle speed (per logical address)	51
3.5.11	Logical address of Main to Aux3 spindles	51
3.5.12	Logical address of the measured spindle.....	51
3.5.13	Physical address of Main to Aux3 spindles	52
3.5.14	Physical address of the measured spindle	52
3.6	Symbols accessing the data of the previous block.....	53
3.7	Accessing data via L900 to L959	54
3.7.1	F, S, T, D, H and N arguments	54
3.7.1.1	Particular case of H and N	54
3.7.2	EA to EZ arguments	55
3.7.2.1	Shortcut for L900 to L951	55
4	Creating and managing symbolic variable tables	59
4.1	Defining a table ▶ TAB	59
4.1.1	Table dimensions	60
4.1.2	Initializing variables and tables	61
4.2	Creating tables for storing part profiles	62
4.2.1	Data that can be stored in a table	62
4.3	Storing a profile ▶ BUILD	63
4.3.1	Allocation of additional entries	63
4.3.2	Declaring entries as a list of bits	64
4.4	Storing a profile interpolated in the plane ▶ P.BUILD	65
4.4.1	Entries of the first dimension	66
4.5	Offsetting an open profile and updating ▶ R.OFF	67
4.6	C.OFF : OFFSET A CLOSED CONTOUR	69
4.6.1	C.INIT	70
4.6.2	C.OFF	71
4.6.3	ADDITIONAL PARAMETERS FOR C.OFF	72
4.7	Redefining a profile according to the tool clearance angle ▶ CUT	77
4.8	M Functions and/or axes motion enable/ disable ▶ BSET, BCLR	79
4.9	Searching the stack for symbolic variables ▶ SEARCH	80
4.10	Preserves a list of symbolic variables ▶ SAVE	81
4.11	Copying Blocks from one table to another ▶ MOVE	82
4.11.1	General Syntax	82
4.11.2	Syntax for simple copy of blocks	83
4.11.3	Syntax for partial copy of blocks	83
4.11.4	Syntax for specifying entries to be copied	84
4.11.5	Examples	85
4.12	Indirect addressing of symbolic variables.....	87
4.13	Programming examples	88
4.13.1	Use of the BUILD function for back and forth milling.....	88
4.13.2	Use of P.BUILD for turning.....	89
4.13.3	Getting a symbolic variable or part program address =@	91

4.13.3.1	Syntax for reading a symbolic variable address.....	91
4.13.3.2	Syntax for reading a part program address	91
4.14	Creating / Deleting Binary Files G76+/- H<bin>	92
4.14.1	Syntax for binary file creation	92
4.14.2	Syntax for file deletion	92
4.15	Saving and retrieving symbolic variables	94
4.15.1	Syntax for saving to a memory segment	94
4.15.2	Syntax for clearing a memory segment.....	95
4.15.3	Syntax for reading a memory segment.....	95
4.15.4	Getting access to the address of a segment	95
5	Program stack - Symbolic variables - Strings	99
5.1	Program stack	99
5.1.1	Use of the stack	99
5.1.2	Stack Allocation	99
5.2	Saving and restoring ► L variables	100
5.2.1	Push Function: ► PUSH	100
5.2.2	Pull Function: ► PULL	101
5.3	Symbolic variables: ► VAR, ENDV	103
5.3.1	Symbolic variables declaration: VAR and ENDV.....	103
5.3.2	Using symbolic variables in ISO programs.....	104
5.3.3	Getting the variables in the stack of another channel: ► VAR H.. N.. N..	106
5.3.3.1	Programmed deletion of variables: ► DELETE	107
5.4	Strings	108
5.4.1	String functions	109
5.4.2	String functions returning a value	109
5.4.3	String function without return value	109
5.4.4	String functions for user interaction	110
5.4.5	String functions for exchange with the PLC	110
6	Subroutine call by G Functions	115
6.1	General	115
6.2	List of G functions calling a subroutine	116
6.3	Particularities	116
6.4	Theory.....	117
6.4.1	Preliminary	117
6.4.2	Operation	117
6.5	Subroutine example	118
6.5.1	Cycle syntax and parameters	118
6.5.2	Graphical representation	118
6.5.3	Main machining program	119
6.5.4	Cycle subroutine	119
6.6	Custom cycle example 2.....	121
6.6.1	Cycle syntax and parameters	121
6.6.2	Graphical representation	122
6.6.3	Cycle subroutine	123
6.7	Decrypting a standard canned cycle	125
6.7.1	Syntax of the G83 cycle.....	125
6.7.1.1	Subroutine %10080 common to all canned cycles.....	129
6.7.2	Timed M	132
6.7.2.1	Accuracy.....	132
6.7.2.2	Extreme cases.....	133
6.7.3	Inter samples	133
6.7.3.1	Delay or anticipation.....	133
6.7.3.2	Multiple M in one sample.....	133

7	MLEGB (Multi Level Electronic Gear Box).....	137
7.1	General	137
7.2	G770 : Continuous running of an axis	137
7.3	G771 : EGB Synchro definition / destruction	138
7.3.1	Synchronisation by tables.....	139
7.4	G772: Synchronisation modification	140
7.5	G775 : EGB Activation / Deactivation	142
7.5.1	Syntax for activation	142
7.5.2	Syntax for de-activation	142
7.6	G776 : Waiting for EGB Synchronisation completion before proceeding.....	143
7.7	G779 : Synchro-Homing (... or "Gear alignment")	144
7.7.1	Other Syntax.....	144
7.7.2	Example	145
7.8	Important notes.....	146
7.8.1	Emergency	146
7.8.2	Override	146
7.8.3	Reset.....	146
7.8.4	EGB Improvements.....	147
7.8.4.1	Rotary leader managed as a linear	147
7.8.4.2	Follower axis limits	147
7.8.4.3	EGB Follower filter action	148
7.8.5	Useful parameters.....	149
8	Dynamic operators	153
8.1	General	153
8.1.1	Virtual axes	153
8.1.2	List of operation codes	154
8.1.3	General syntax of an operation	155
8.1.3.1	Example of definition of an operation.....	155
8.1.4	Cancel an operation	155
8.1.4.1	Cancel an operation by operation code = 0	155
8.1.5	Order of execution of the operations	156
8.1.5.1	Execution of consecutive operations	156
8.1.6	Dynamic operations can be used by any channel	156
8.2	Types of operands used.....	157
8.2.1	Type 1 external parameters	158
8.2.2	Type 2 external parameters	159
8.2.3	Type 3 external parameters	165
8.2.4	Immediate values	166
8.2.5	Operands pointing two consecutive memory locations	166
8.2.6	L program variables	166
8.2.7	Symbolic variables	166
8.2.8	Axis Addresses	166
8.2.9	NCK Address	166
8.3	Dynamic operators execution	167
8.4	Code1 ► Add	168
8.5	Code2 ► Subtract	169
8.6	Code3 ► Multiply	170
8.7	Code4 ► Rotate a vector	171
8.8	Code5 ► Load a parameter with or without mask	172
8.9	Code6 ► Load of a peripheral memory with or without mask	173
8.9.1	Load an axis address	173
8.9.2	Load from PLC Data	174
8.9.2.1	Parameters E47000 to E47124.....	174

8.9.2.2	Parameters E4300x.....	174
8.9.2.3	Parameters E4400x	175
8.9.2.4	Parameters E4410x	175
8.9.2.5	Parameters E4600x	175
8.10	Code7 ► Indexed load of a parameter	176
8.11	Code8 ► Indexed storage of a parameter	177
8.12	Code9 ► Compare two parameters	178
8.13	Code10 ► Conditions on the sum of two parameters	180
8.14	Code11 ► Store data in a peripheral memory	182
8.14.1	Storage at an axis address	182
8.14.2	Store in a PLC memory	183
8.14.2.1	Parameters E10000 to E10031	183
8.14.2.2	Parameters E3300x.....	184
8.14.2.3	Parameters E3400x.....	184
8.14.2.4	Parameters E3410x.....	184
8.14.2.5	Parameters E3600x.....	184
8.14.2.6	Parameters E37000 to E37127	185
8.14.2.7	Parameters E3410x	186
8.15	Code12 ► Increment with modulo	187
8.16	Code13 ► Multiply by a program variable	188
8.17	Code14 ► Arc tangent function	189
8.18	Code15 ► Axes manual control	190
8.19	Code16 ► Third-degree polynomial and derivative	191
8.20	Code17 ► Speed on trajectory	192
8.21	Code18 ► Divide	193
8.22	Code19 ► Square root	194
8.23	Code20 ► Axis speed synchronisation.....	195
8.24	Code21 ► Oscillation cycles	197
8.24.1	Time base modulation	198
8.25	Code22 ► Execute a C dynamic operator	199
8.25.1	Development tools	201
8.25.2	Steps for creating a project.....	202
8.25.3	Modules supplied by NUM.....	203
8.25.4	Reading parameters	203
8.25.4.1	E parameters	203
8.25.4.2	L parameters	204
8.25.5	Structure of the C modules	205
8.25.5.1	init (UINT16 gr_no, UINT16 no_arg, ARGUMENT *)	206
8.25.5.2	princ (void)	207
8.25.5.3	quitt (void)	208
8.25.6	Limits and precautions	208
8.25.7	Transferring the code for execution	208
8.25.8	Advanced features	208
8.25.9	Error messages	209
8.26	Code23 ► Electronic cam	214
8.27	Code24 ► Differential shift	215
8.28	Code25 ► Read current block data	217
8.29	Examples of dynamic operators programs.....	219
8.29.1	Example 1.....	219
8.29.2	Example 2.....	220
8.29.3	Example 3.....	221

9	Inclined Plane and RTCP	227
9.1	General	227
9.2	Inclined Plane machining.....	228
9.2.1	Overview.....	228
9.2.2	Conventional Inclined Plane definition.....	229
9.2.3	Programming Syntax	230
9.2.3.1	Specification with respect to MCS	230
9.2.3.2	Specification after positioning the tool in a known point of ICS	231
9.2.3.3	Enabling an ICS from the current tool orientation	233
9.2.3.4	Inclined plane suspension	236
9.2.3.5	Re-Enabling inclined plane.....	237
9.2.4	Notes about the inclined plane function.....	238
9.3	Complements about inclined plane G150.....	239
9.3.1	Background on the coordinate transformation	239
9.3.2	Case studies for an iPLANE implementation	239
9.3.3	Case of a tilting turntable	240
9.3.3.1	Machining the first oblong P1 pocket	240
9.3.3.2	Summary of the implementation procedure	242
9.3.3.3	Machining the second oblong pocket.....	243
9.3.4	Case of a C'-axis turntable combined with an A-axis Head	246
9.3.4.1	Machining the oblong P1 pocket	246
9.4	RTCP (Rotation around Tool Center Point)	248
9.4.1	General	248
9.4.2	Convention of Tool Center Point definition	249
9.4.3	Datums DAT1 and DAT2	249
9.4.3.1	Both DAT1 and DAT2 offsets applied before ICS	249
9.4.3.2	Only DAT1 offset applied before ICS.....	250
9.4.3.3	Only DAT1 offsets applied before the ICS.....	250
9.4.4	Programming Syntax	251
9.4.4.1	RTCP enabling with inclined coordinate system specification (ICS).....	251
9.4.4.2	RTCP enabling with specification of an ICS after positioning.....	253
9.4.4.3	Enabling a tool ICS from the current tool orientation	254
9.4.4.4	Suspension of the RTCP function and its associated ICS	257
9.4.4.5	Re-Enabling the RTCP function without its associated ICS	257
9.4.4.6	Re-Enabling the RTCP function with its associated ICS.....	258
9.4.5	Additional functions.....	259
9.4.5.1	Tool length correction	259
9.4.5.2	Tool wear offset	259
9.4.5.3	Tridimensional tool correction (G29)	259
9.4.5.4	Table off-centering (DAT3).....	259
9.4.5.5	Program origin offset (G59)	260
9.4.6	Notes about the RTCP function and its associated inclined plane.....	262
10	Tool Vector Programming and 3D Workpiece Positioning	267
	Compensation.....	267
10.1	General	267
10.2	Implementation	268
10.3	Tool orientation	269
10.4	Details on G codes	270
10.5	Complement.....	271
10.6	Precautions and limits of usage.....	272

10.7	Early Block Change	275
10.8	Limitations.....	276
10.9	Modality.....	276
10.10	Programming	277
10.11	Mode Q0 – EBC suspended	277
10.12	MODE Q1 – EBC when traversed distance higher than.....	278
10.13	MODE Q2 – EBC when residual distance lower than	278
10.14	MODE Q3 – EBC with PLC control.....	279
10.15	MODE Q4 – EBC immediate	279
10.16	MODE Q5 – Wait for confirmation bit	279
10.17	MODE Q6 – only set output bit xx when distance traversed higher than.....	280
10.18	MODE Q7 – only set output bit when remaining distance lower than	280
10.19	PLC Flags	280
11	Indexed axes names.....	285
11.1	Declaration.....	285
11.2	Programming	285
12	N/M AUTO function	289
12.1	General	289
12.1.1	Non interpolated axes.....	289
12.1.2	Errors in N/M AUTO.....	289
12.2	Using the N/M AUTO function	290
12.2.1	Information for the machine tool builder	290
12.3	Stopping and restarting in N/M AUTO mode	291
12.4	Checks included in N/M AUTO	292
12.4.1	Acceleration checks.....	292
12.4.2	Speed checks	292
12.4.3	Check of travels	292
12.4.4	Miscellaneous checks.....	292
13	Polynomial interpolation.....	297
13.1	General	297
13.1.1	Distinction between segmented and smooth polynomial interpolation	297
13.2	Segmented polynomial interpolation	298
13.2.1	Notes on the axes and coefficients programmed	298
13.2.2	Geometric transformations	299
13.2.3	Radius correction using polynomial interpolation	299
13.2.4	Interpolation Feed rate.....	300
13.3	Smooth polynomial interpolation	301
13.3.1	Restrictions on smooth polynomial interpolation	302
14	B-Spline function.....	307
14.1	General	307
14.2	Restrictions and programming errors	308
14.3	Precision about H parameter with G62.....	309
15	Application specific E parameters and G functions	313
15.1	General	313
15.2	Continuous probing E11019	313
15.3	Enabling reference writing for virtual axes E11020.....	314
15.4	Assigning axes to a channel during Reset E11024.....	314

15.5	Inhibiting program modification and deletion E31002	314
15.6	Inhibit writing into memory segments E31003.....	315
15.7	Circle tolerance E32012.....	315
15.8	Tool disengagement in line to line transition E69004.....	315
15.9	Axis duplication and synchronisationE941..., E96.....	316
15.10	Monitoring drives test points E351..., E352.....	317
15.11	Swapping drive parameters E353.....	318
15.12	Recording continuous axes position G751	319
A	Appendix	323
A.1	Structured programming commands	323
A.2	Operators	324
A.3	String functions	325
A.4	Symbolic variable instructions	326
A.5	Program status access symbols	328
A.5.1	G and M functions.....	328
A.5.2	List of bits	328
A.5.3	Values	329
A.6	Symbols stored in L900 to L951 variables	330
A.6.1	Symbols stored in variables L900 to L925	330
A.6.2	Symbols stored in variables L926 to L951	330
A.7	List of Dynamic Operations	331
A.8	Coordinate conversions.....	333
A.8.1	General	333
A.8.2	Using the coordinate conversion matrix ► G24	333
A.8.2.1	Properties of the function	334
A.8.2.2	Error numbers and messages.....	334
A.9	RTCP function.....	335
A.9.1	General	335
A.9.1.1	Control of rotary axes	335
A.9.1.2	Control of rotary axes	335
A.9.2	Programming the RTCP function ► G26	336
A.9.2.1	General	336
A.9.2.2	Programming errors	336
A.9.2.3	Other Errors	336
A.9.2.4	Notes	337
A.9.3	Structure description	338
A.9.3.1	Twist Heads configuration	338
A.9.3.2	Representation of a twist head configuration.....	338
A.9.3.3	Turntables configuration.....	339
A.9.3.4	Representation of a turntable configuration	339
A.9.4	Other processing related to the RTCP function	340
A.9.4.1	Part on inclined plane	340
A.9.4.2	Table off-centering (DAT3)	341
A.9.4.3	Tool length correction	341
A.9.4.4	Tool wear offset	341
A.9.4.5	Tool correction (G29) : 3 or 5-axis tool correction	341
A.9.5	Restrictions and conditions of use	342

Summary

1	Foreword	13
1.1	Use of the manual.....	13
1.2	Use of symbols in the manual.....	13
1.3	Document revisions	14
1.4	How to read the manual.....	15
1.5	Syntax conventions.....	16
1.6	Relations with the PLC.....	17
1.7	NUM Agencies.....	17

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

A

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 manual carefully and retain for future reference.

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.

1.3 Document revisions

Date	Index	Reason for revision	See chapter
1 - 2018	00	Document creation	-
2 - 2018	01	- Reminder - New section "Assigning axes to a channel during Reset E11024" - New Operators - New section "RTCP function"	13.1 13.4 A2 A8
11 - 2018	02	- New section "Strings". - New section "String functions". - New section "String functions returning a value". - New section "String functions without returning a value". - New section "String functions for user interaction". - New section "String functions for exchange with PLC". - New section "Timed M functions". - New section "Timed M and accuracy". - New section "Inter samples". - New section "Type 1 external parameters". - New section "Type 2 external parameters". - New section "Type 3 external parameters". - New remark on section "Parameters E37000 to E37127". - New section "Recording continuous axes position G751".	5.4 5.4.1 5.4.2 5.4.3 5.4.4 5.4.5 6.8 6.8.1 6.8.2 8.2.1 8.2.2 8.2.3 8.14.2.6 13.12
10 - 2021	03	SW update 4.1.30.00 Update chapter 2 - Structured Programming - General General syntax rules Loops with control variable: ► FOR, TO/DOWNT0, BY, DO, ENDF Selective jump : ► SWITCH CASE BREAK ENDS Next iteration : ► CONT Exiting the Loop : ► EXIT Table update - List of multi spindle M functions Variable update - Angle of block Variable update - Line number Variable update - Spindle programmed Variable update - Spindle Index angle Variable update - Value of abstract angles Remark - I, J and K values programmed Information update - Useful parameters Table update - Type 1 external parameters Table update - Type 2 external parameters Table update - Code25 ► Read current block data New chapter - Early Block Change New chapter - Indexed axes names Table update - Structured programming commands A4 - Program status access symbols: Table update - Values Paragraph removal General information	2.1 2.2.1 2.3.5 2.3.6 2.3.7 2.3.8 3.2.2 3.4.5 3.4.6 3.5.1 3.5.2 3.5.3 3.5.6 7.8.4 8.2.1 8.2.2 8.28 10 11 A.1 A.4.3 A.7.3 - A.7.4 A.8.1
10 - 2022	05	SW update 4.2.0 C.OFF: offset a closed contour - New paragraph G772: Synchronisation modification - New paragraph EGB Improvements Useful parameters update Code14 ► Arc tangent function Symbolic variable instructions - Table update	4.6 7.4 7.8.4 7.8.5 8.17 A.4

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 customize the system for a particular 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 This chapter introduces: 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	Creating and managing symbolic variable tables This chapter introduces: A table creation, dimensions, values, commands and blocks.
5	Program stack - Symbolic variables - Strings This chapter introduces: The use of the stack, stack allocation, all the stack commands and symbols.
6	Subroutine call by G Functions This chapter introduces: - List of G functions calling a subroutine - Subroutine example and graphical representation.
7	MLEGB (Multi Level Electronic Gear Box) This chapter introduces: MLEGB functions and all related functions G770: Continuous running of an axis, G771: Master slave definition, G775: Activation / De-activation of the synchronisation previously defined, G776: Wait for effective synchronisation , G779: Axes alignment.
8	Dynamic operators This chapter introduces: - Overview and description of the syntax specific to each dynamic operators - Examples of programs using the dynamic operators.
9	Inclined Plane and RTCP This chapter introduces: Inclined Plane function and RTCP (Rotation around Tool Center Point) function.
10	Tool Vector Programming and 3D Workpiece Positioning Compensation This chapter introduces: G777 function to simultaneously execute two blocks.
11	Indexed axes names This chapter introduces: Possibility to have similar axes names in a single channel.
12	N/M AUTO Function This chapter introduces: Possibility of controlling the N/M AUTO axes while the other machine axes follow a programmed path.
13	Polynomial interpolation This chapter introduces: - Polynomial interpolation use and syntax. - Segmented polynomial interpolation and smooth polynomial interpolation.
14	B-Spline function This chapter introduces: Possibility to define a path described by several "Poles".
15	Application specific E parameters and G functions This chapter introduces: 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 This chapter introduces: Structured programming commands, Symbolic variable management commands, Program status access symbols, Symbols stored in L900 to L951 variables, List of Dynamic Operations, RTCP function.

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 to which the syntax relates is highlighted by the use of bold 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 M00044.

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. :	<code>W_ALLCNC.NMAuto</code>	syntax in the present manual
	<code>Flexium_NCK.W_ALLCNC.NMAuto</code>	syntax used in PLC programming.

1.7 NUM Agencies

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

Page deliberately empty

Summary

2	Structured Programming	21
2.1	General	21
2.2	Commands used in structured sequences	22
2.2.1	General syntax rules	22
2.2.2	Nesting and branches	23
2.3	Structured programming commands	24
2.3.1	Condition graph	24
2.3.2	Conditional Execution : ▶ IF, THEN, ELSE, ENDI	25
2.3.3	REPEAT Loop : ▶ REPEAT, UNTIL	26
2.3.4	WHILE Loop : ▶ WHILE, DO, ENDW	27
2.3.5	Loops with control variable : ▶ FOR, TO/DOWNT, BY, DO, ENDF	28
2.3.6	Selective jump : ▶ SWITCH, CASE, BREAK, ENDS	29
2.3.7	Next iteration : ▶ CONT	31
2.3.8	Exiting the Loop : ▶ EXIT	32
2.3.9	Example of Structured Programming	33

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**
- **FOR**
- **REPEAT**
- **SWITCH**
- **WHILE**

It ends with:

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



Contrary to previous versions the syntax of the keywords is strictly enforced. The analysis is no more limited to the first 4 characters; it is necessary to use the full keyword.



These instructions must not be used in programs executed in DripFeed mode.

Full list of keywords

"AND", "BCLR", "BREAK", "BSET", "BUILD", "BY", "C.OFF", "C.INIT", "CASE", "CONT", "CUT", "DELETE", "DEFAULT", "DO", "DOWNT", "ENDS", "ELSE", "ENDF", "ENDI", "ENDV", "ENDW", "EXIT", "FOR", "IF", "MOVE", "OR", "P.BUILD", "PULL", "PUSH", "R.OFF", "REPEAT", "SAVE", "SEARCH", "THEN", "TO", "UNTIL", "VAR", "WHILE".

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, DOWNT0, BY, DO, ENDF**
- Exit from a loop: **EXIT**
- Abort iteration: **CONT**

2.2.1 General syntax rules

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

The words **IF, CASE, DO, DOWNT0, ELSE, REPEAT, UNTIL, WHILE** must always be followed by a space.

e.g.:

WHILE L0 <3 is not recognized 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

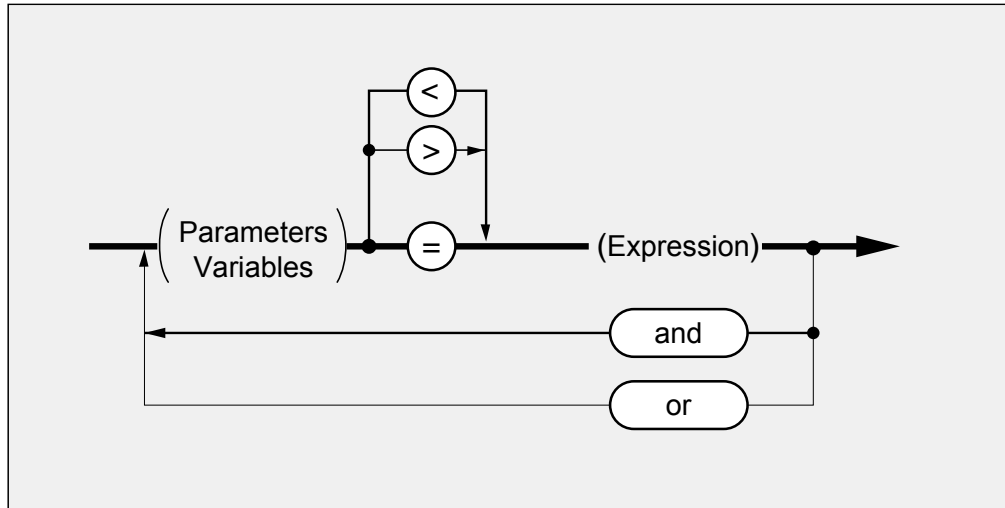
```

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 M00040).
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

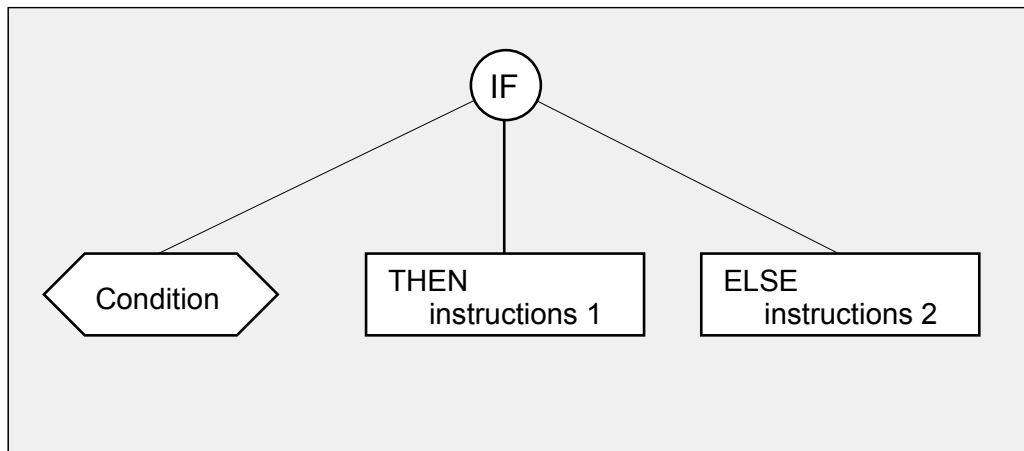
```

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

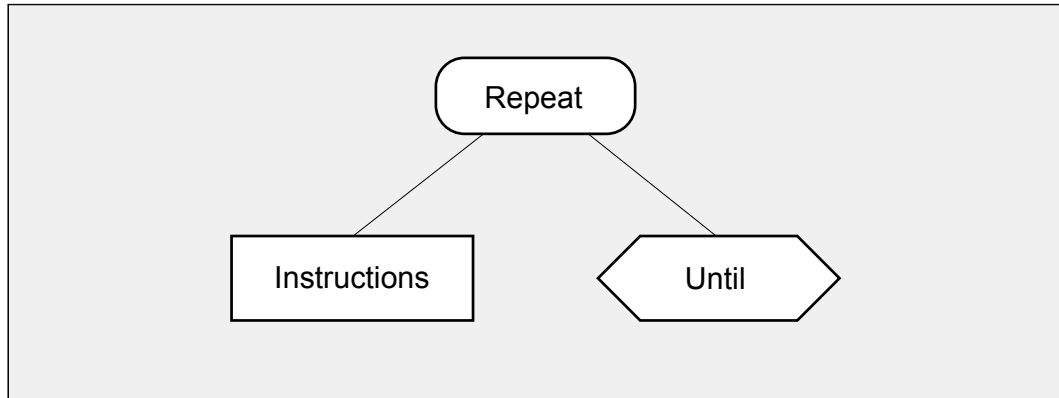
Syntax

```
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

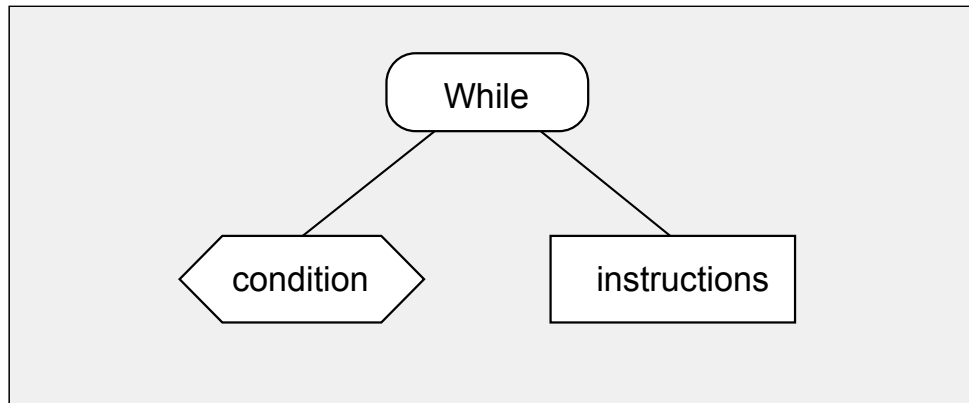
Syntax

```
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
```

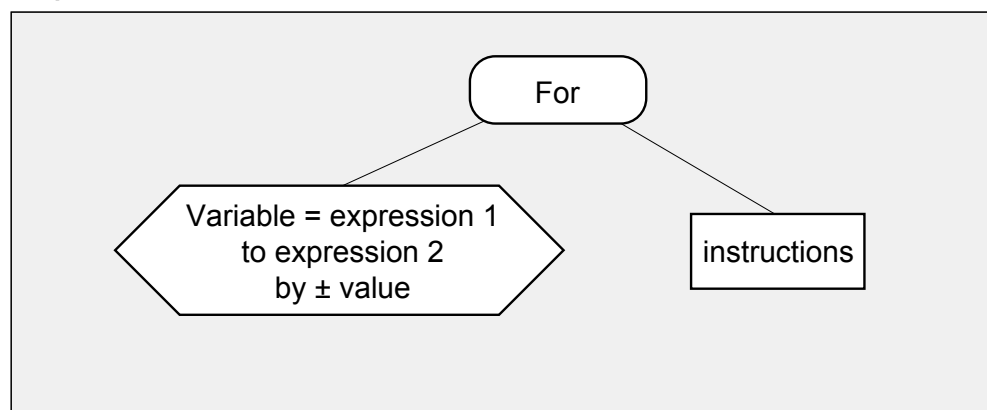
«*Variable*» is first initialized to «*expression 1*».

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 Selective jump : ► SWITCH, CASE, BREAK, ENDS

Syntax

```

SWITCH expression
    CASE constant expression 1
        statements to execute until the next " break" keyword
    BREAK

    CASE constant expression 2
        statements to execute until the next " break" keyword
    BREAK
    .....
    CASE constant expression n
        statements to execute until the next " break" keyword
    BREAK

    DEFAULT
        statements to execute in other cases

ENDS

```

According to the value of "Expression" (SWITCH) execution will continue at the respective CASE statement

If there is no respective CASE execution will be transferred to DEFAULT

The optional keyword BREAK transfers the control to the label ENDS. IF BREAK is omitted in one CASE condition, then execution **will continue in sequence at the next CASE**

They can be up to 65535 CASE statements in a single SWITCH block

The expression used in a switch statement can be an L variable, an E parameter, a symbolic variable or a numerical expression with curly brackets '{,}'. The switch expression is converted to an integer internal variable before being compared with CASE constant-expression.

The constant-expression for a CASE must be an integer constant {1,2,3,..} or an ASCII char { 'A','B',..., 'a', 'b', ..., '1', '2',... '\n', '\r', '\t', ' ', ... }

Example

\$ The color selected is

SWITCH L0

CASE 1

\$+ Blue

BREAK

CASE 2

\$+ Red

DEFAULT

\$+ Unknown

ENDS

As the **BREAK** statement has been forgotten for **CASE** 2 the display will show the following when L0 = 2:

The color selected is Red Unknown

2.3.7 Next iteration : ► **CONT**

Syntax

CONT

The instruction CONT is used to skip to the next iteration in a loop

Example

```
FOR L0 = 1 TO 20 DO  
    Instructions block 1  
    IF L0 = L1 THEN  
        CONT  
    ENDI  
    Instruction block 2  
ENDF
```

Instruction block 2 will not be executed if L0 = L2; however the loop will continue to iterate until L0 = 20

2.3.8 Exiting the Loop : ► **EXIT**

Syntax

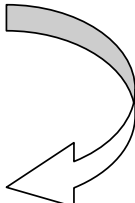
EXIT

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.

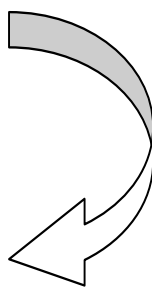
The word EXIT is used only in IF loops.

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
```

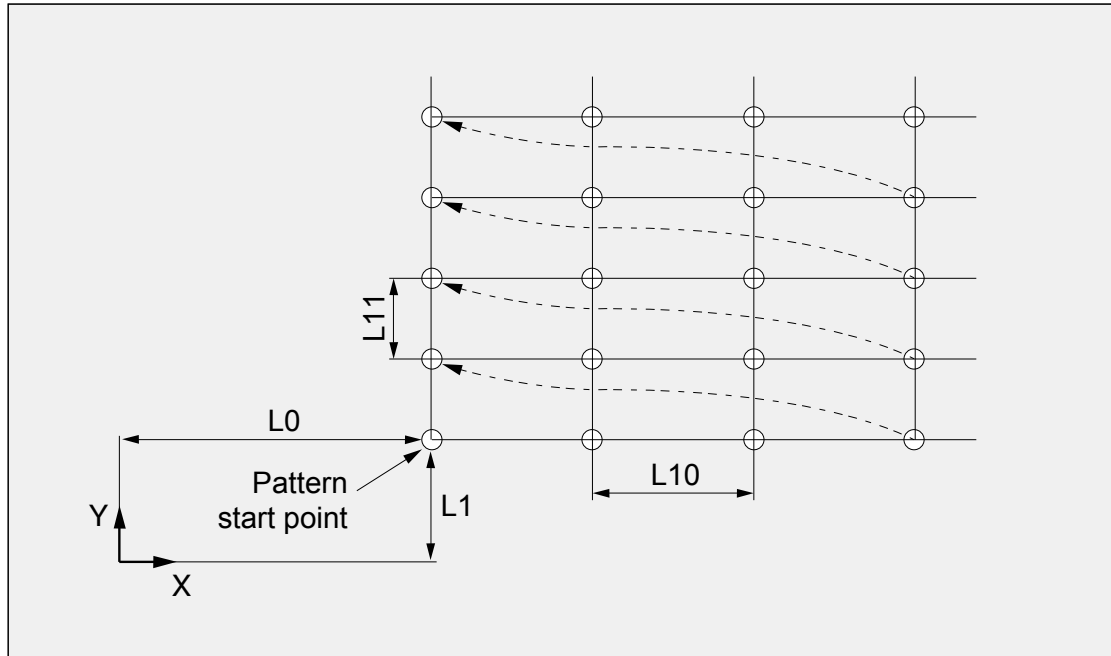


2.3.9 Example of Structured Programming

Hole drilling pattern

L2 = number of holes in X

L3 = number of holes in Y



```
%20
```

```
$ X START:
```

```
L0=$
```

```
$ Y START:
```

```
L1=$
```

```
$ X STEP:
```

```
L10=$
```

```
$ Y STEP:
```

```
L11=$
```

```
L4=0
```

Input of parameters

```
WHILE L4 <> 25 DO
```

```
  REPEAT $ NUMBER OF POINTS IN X:
```

```
    L2=$
```

```
  UNTIL L2>0
```

Test for answer yes: Y

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
ENDW
$ THANK YOU
N.. G00 G52 X0 Z0
N.. T1 D1 M06
N..
FOR L5=1 TO L3 DO
    $ ROW POSITION:
    $=L5
    XL0 L6=L5-1*L11+L1 YL6

    G00 Z0

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

    G01 Z-10 F50
    G00 Z0

ENDF
ENDF

N..
M02

```

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

GO !

Execution

Loop on rows

Loop on columns

Page deliberately empty

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

A

Page deliberately empty

Summary

3	Reading the program status access symbols	39
3.1	General	39
3.2	Symbols addressing a current Boolean value	40
3.2.1	Active G Functions	40
3.2.2	Active M Functions	40
3.3	Symbols addressing a list of current status bit	43
3.3.1	Programmed addresses	43
3.3.2	I, J and K arguments programmed	43
3.3.3	P, Q and R arguments programmed	44
3.3.4	Axes programmed in the current block	44
3.3.5	Axes programmed since the beginning of the program	44
3.3.6	Primary or Secondary axes programmed	45
3.3.7	Mirrored axes	45
3.4	Symbols addressing current numerical values	46
3.4.1	Feed rate	46
3.4.2	Spindle speed (for the main spindle of the channel only)	46
3.4.3	Tool number	46
3.4.4	Tool correction number	46
3.4.5	Angle of block	46
3.4.6	Line number	46
3.4.7	Number of the last block encountered	47
3.4.8	Angular offset	47
3.4.9	Spindle orientation (milling)	47
3.4.10	Dwell time programmed (G04 F..)	47
3.4.11	Canned cycle number	47
3.4.12	Spline curve number value	47
3.4.13	Tool axis orientation	48
3.4.14	Nesting level of the current subroutine	48
3.4.15	List of axes programmed in the block	48
3.5	Symbols addressing a list of current numerical values	49
3.5.1	Spindle programmed	49
3.5.2	Spindle Index angle	49
3.5.3	Value of abstract angles	49
3.5.4	Axes dimensions programmed	49
3.5.5	Offsets values programmed	50
3.5.6	I, J and K values programmed	50
3.5.7	P, Q and R. values programmed	50
3.5.8	Subroutine number	51
3.5.9	Programmed angular offsets origin (G59 I.. J.. K..)	51
3.5.10	Spindle speed (per logical address)	51
3.5.11	Logical address of Main to Aux3 spindles	51
3.5.12	Logical address of the measured spindle	51
3.5.13	Physical address of Main to Aux3 spindles	52
3.5.14	Physical address of the measured spindle	52
3.6	Symbols accessing the data of the previous block	53
3.7	Accessing data via L900 to L959	54
3.7.1	F, S, T, D, H and N arguments	54
3.7.1.1	Particular case of H and N	54
3.7.2	EA to EZ arguments	55
3.7.2.1	Shortcut for L900 to L951	55

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 [symp], 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] / [.IMxx(ab)]

The symbol [.BMxx] is used to determine whether the M functions specified by xx are active or not :

[.BM40]=0: function M40 not active

[.BM11]=1: function M11 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]	[.M997]	[.M998]	[.M999]				

List of multi spindle M functions

[.IM03(1..32)]	[.IM04(1..32)]	[.IM05(1..32)]	[.IM19(1..32)]	[.IM40(1..32)]	[.IM41(1..32)]
[.IM42(1..32)]	[.IM43(1..32)]	[.IM44(1..32)]	[.IM45(1..32)]	[.IM66(1..32)]	

The short spindle relative flags ([.BM03], [.BM04], ... are maintained for compatibility and are only managed when the main spindle is programmed by default (i.e. without specifying its logical address or rank by ':' or '#'). For a status of all spindles it is suggested to use the Flexium+ specific flags [.IMxx(ab)] defined by the list of Multi spindle M functions.

Neither one of the functions M10 and M11 (use of axes clamps) is initialized at power up. Once one has been programmed they become exclusive. As long as they are not initialized the flags [.BM10] and [.BM11] will be equal to -1.

Example using default main spindle programming

```

%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]

```

Extract valid for any spindle (S2 in this case)

```
VAR
    [MRANGE] [MDIRECT] [SPEED]
ENDV

$ Saving context of spindle 2
    [MRANGE]=40*[.IM40(2)]           \\ Speed range
    [MRANGE]=41*[.IM41(2)]+[MRANGE]

    [SPEED] = [.IRS(2)]              \\ Spindle speed

    [MDIRECT]=03*[.IM03(2)]
    [MDIRECT]=04*[.IM04(2)]+[MDIRECT] \\ Direction of rotation
    [MDIRECT]=05*[.IM05(2)]+[MDIRECT]

...

$ Restoring context of spindle 2
M[MDIRECT]:2 M[MRANGE]:2 S[SPEED]:2
M2
```

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 (for the main spindle of the channel only)

[.RS]

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

The programmed speed of each spindle in the system as well as other spindle information (e.g. Logical or Physical addresses) are described in chapters 3.5.6 and following.

3.4.3 Tool number

[.RT]

3.4.4 Tool correction number

[.RD]

3.4.5 Angle of block

[.REA]

[.REA] real values giving (in polar programming) the shift angle programmed EA (related to PGP Programming Geometrical Profile).

3.4.6 Line number

[.RLN]

3.4.7 Number of the last block encountered

[.RN]

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

3.4.8 Angular offset

[.RED]

3.4.9 Spindle orientation (milling)

[.REC]

3.4.10 Dwell time programmed (G04 F..)

[.RG4]

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

3.4.11 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.

3.4.12 Spline curve number value

[.RNC]

3.4.13 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

3.4.14 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)

3.4.15 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 Spindle programmed

[.IBO(i)]

[.IBO(i)] as boolean value indicating the programming of A_0 = ... B_0 = ... C_0 = ...

Programming identical to [.IBX(i)] but with i limited to [1, 2, 3] respectively A_0 = ... , B_0 = ... , C_0 = ...

3.5.2 Spindle Index angle

[.IREC(i)]

3.5.3 Value of abstract angles

[.IRO(i)]

[.IRO(i)] giving the value (real) of the tool vector orientation programmed by its projections on X, Y, Z axes of the working frame (usage of G32).

Reminder: these values are programmed by A_0 = ... B_0 = ... C_0 = ...

3.5.4 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

3.5.5 Offsets values programmed

[.IRTX(i)]

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

3.5.6 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



Remark

When I, J or K are not programmed in a block, the variables [.IRI(i)] are respectively loaded with the position of the main axes.

To properly manage [.IRI(i)] make sure I, J or K are programmed by checking [.IBI(i)]..

3.5.7 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

3.5.8 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 (8 possible nesting levels)

3.5.9 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

3.5.10 Spindle speed (per logical address)

[.IRS(1..32)]

Similar to .RS but for any of the logical address.

3.5.11 Logical address of Main to Aux3 spindles

[.IRSL(1..4)]

3.5.12 Logical address of the measured spindle

[.IRSR(1)]

3.5.13 Physical address of Main to Aux3 spindles

[.IRL(1..32)]

3.5.14 Physical address of the measured spindle

[.IRSM(1)]



For all spindle relative variables above, it is possible to index the spindle number. However this index can only be an E variable, an L9xx or a symbolic variable but never an L0..L99 number due to the look ahead feature.



All these variables are Read Only.

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 L959

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

The system offers the variables L900 to L959. 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 being 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

Summary

4	Creating and managing symbolic variable tables	59
4.1	Defining a table ▶ TAB	59
4.1.1	Table dimensions	60
4.1.2	Initializing variables and tables	61
4.2	Creating tables for storing part profiles	62
4.2.1	Data that can be stored in a table	62
4.3	Storing a profile ▶ BUILD	63
4.3.1	Allocation of additional entries	63
4.3.2	Declaring entries as a list of bits	64
4.4	Storing a profile interpolated in the plane ▶ P.BUILD	65
4.4.1	Entries of the first dimension	66
4.5	Offsetting an open profile and updating ▶ R.OFF	67
4.6	C.OFF : OFFSET A CLOSED CONTOUR	69
4.6.1	C.INIT	70
4.6.2	C.OFF	71
4.6.3	ADDITIONAL PARAMETERS FOR C.OFF	72
4.7	Redefining a profile according to the tool clearance angle ▶ CUT	77
4.8	M Functions and/or axes motion enable/ disable ▶ BSET, BCLR	79
4.9	Searching the stack for symbolic variables ▶ SEARCH	80
4.10	Preserves a list of symbolic variables ▶ SAVE	81
4.11	Copying Blocks from one table to another ▶ MOVE	82
4.11.1	General Syntax	82
4.11.2	Syntax for simple copy of blocks	83
4.11.3	Syntax for partial copy of blocks	83
4.11.4	Syntax for specifying entries to be copied	84
4.11.5	Examples	85
4.12	Indirect addressing of symbolic variables	87
4.13	Programming examples	88
4.13.1	Use of the BUILD function for back and forth milling	88
4.13.2	Use of P.BUILD for turning	89
4.13.3	Getting a symbolic variable or part program address =@	91
4.13.3.1	Syntax for reading a symbolic variable address	91
4.13.3.2	Syntax for reading a part program address	91
4.14	Creating / Deleting Binary Files G76+/- H<bin>	92
4.14.1	Syntax for binary file creation	92
4.14.2	Syntax for file deletion	92
4.15	Saving and retrieving symbolic variables	94
4.15.1	Syntax for saving to a memory segment	94
4.15.2	Syntax for clearing a memory segment	95
4.15.3	Syntax for reading a memory segment	95
4.15.4	Getting access to the address of a segment	95

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 M00040.

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 «,» .

Syntax

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

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

4.1.1 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 initialized 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] / RLO [SINX] = [SINX] / RLO  
    X [TABL(1,IX+1)] Y [TABL(2,IX+1)]  
ENDF
```

4.1.2 Initializing variables and tables

The values are initialized with a default value of zero.

Initializing 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.

- **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.

- **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.

- **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.

- **Feed Rate F**

The modal value related to the F function is stored.

- **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.

- **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.

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

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

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

4.3.1 Allocation of additional entries

Additional entries can be allocated in the first table dimension if they are not initialized 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.
```

4.3.2 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 fixed to 7 entries
- The second dimension is limited to 65536 entries (Depending on the setting of P7 N4 Bit 5).

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.
NB	Name of the variable containing the number of blocks (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 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.

4.4.1 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 center on X: • Value on the X axis for circular interpolation, otherwise = 0
Entry 5	Position of the center 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                                %80
N10 G17 X5 Y10                    N10 ...
P.BUILD [TAB(7,NB)] H80 N110 N130  N..
N..                                N..
                                N110 G01 X20
                                N120 G03 X20 Y50 I20 J30
                                N130 G01 X10 Y20
                                N.. ...
```

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

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).

Syntax

R.OFF [Pa(7,Nb)] / ± 1 / R

P	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.

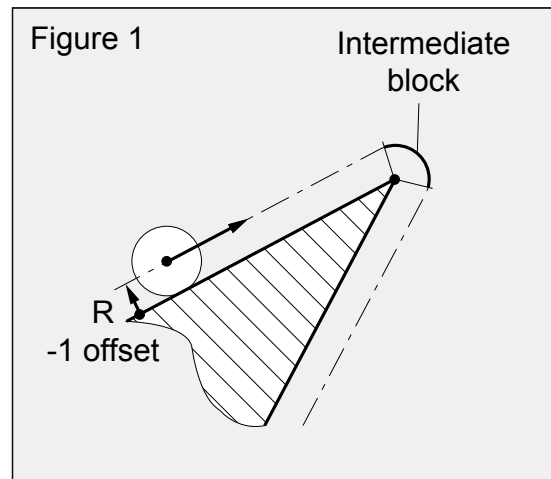
The R.OFF function must be the first word in a block.

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)



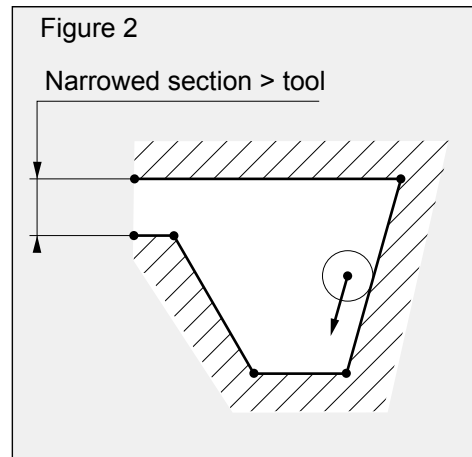
From version 4.2.0 a fourth optional argument is possible. It defines the value of the angle under which connecting arcs will be replaced by an extension of the segments with intersection on the bisectrice.

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)



Example

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

Profile building

Profile offset to the left by 10

4.6 C.OFF : OFFSET A CLOSED CONTOUR

The C.OFF function is intended to offset a given closed (unlike R.OFF) contour. This function is mostly intended for grinding shapes (e.g. cams) on a circular grinder but can also be used for other purposes (e.g. pocket milling).

The operation is divided in several steps:

- Define a contour in ISO
- Using P.BUILD, create a table from this contour
- Initialise the function via the command **C.INIT** and create an output contour table
- Edit this output contour including an offset via **C.OFF**
- Execute this trajectory by reading this output contour table
- If necessary repeat the last two steps with different offset values

Example

```

VAR
  [OFF] = 15
  [I]
  [V]
  [GCode(3)]=2,1,3
ENDV
P.BUILD [TAB(7,N)] N1000 N2000
C.INIT INP=[TAB(7,N)] / OUT=[P] / SIDE = -1
C.OFF OFFSET=[OFF] (Offset contour with static, fixed offset)
FOR [I] = 1 TO [N] DO
  [V] = [P(1,I)]+2
  G[GCode(V)] X[P(2,I)] Y[P(3,I)] I[P(4,I)] J[P(5,I)]
ENDF
M2
N1000
  Profile definition in G01, G02 and G03 and PGP
N2000

```

Syntax

The mandatory parameters are in bold.

The parameters are separated by the "/" character and followed by the sign "=" and a regular expression.

4.6.1

C.INIT

C.INIT INP=[tab(M,N)] / OUT=[out] / SIDE=[side] / POLAR=[polar]											
INP	Declares the name of the symbolic variable containing the input contour and the contour dimensions. TAB is the name of the symbolic variable containing the contour to offset. M is the number of columns of the input contour (P.BUILD default = 7). It is possible to add extra columns in the input table using the BUILD command to create a second table and merging both before performing the C.INIT. These extra columns (e.g. Feedrate) are copied directly in the output contour N is the number of rows of the input contour (max = 65536) Refer to P.BUILD in this manual for additional information about this table										
OUT	Declares the name of the symbolic variable where the contour will be stored. C.INIT creates the variable OUT with the proper dimensions (not known a priori). The number of rows will be output in variable N and the number of columns is M (the same column in input) plus possibly additional columns.										
SIDE	Defines the side where the offset will be applied, the possible value are -1 (offset to left = G41) or 1 (offset to right = G42). <table><tr><th>Input</th><th>Contour definition is CW</th><th>Contour definition is CCW</th></tr><tr><td>From the inside out</td><td>SIDE=1 & G42</td><td>SIDE=-1 & G41</td></tr><tr><td>From outside to inside</td><td>SIDE=-1 & G41</td><td>SIDE=1 & G42</td></tr></table>		Input	Contour definition is CW	Contour definition is CCW	From the inside out	SIDE=1 & G42	SIDE=-1 & G41	From outside to inside	SIDE=-1 & G41	SIDE=1 & G42
Input	Contour definition is CW	Contour definition is CCW									
From the inside out	SIDE=1 & G42	SIDE=-1 & G41									
From outside to inside	SIDE=-1 & G41	SIDE=1 & G42									
POLAR	Optional parameter. This parameter must be assigned to 1 when the input contour is in polar coordinates. The output contour will be stored in polar coordinates too. The input format of a polar contour is a symbolic variable table with two columns. The first column contains the angular position and the second the distance from the centre. Ideally, the table for POLAR CONTOUR should start at 0 degrees and go to 359 degrees. POLAR has a default value 1, so POLAR=1 is equivalent to POLAR If POLAR is not programmed or assigned to 0 the contour is taken in Cartesian coordinates.										

4.6.2 C.OFF

C.OFF presents two different syntaxes (constant offset or variable offset) and some optional parameters

CONSTANT OFFSET

C.OFF OFFSET= [offset]

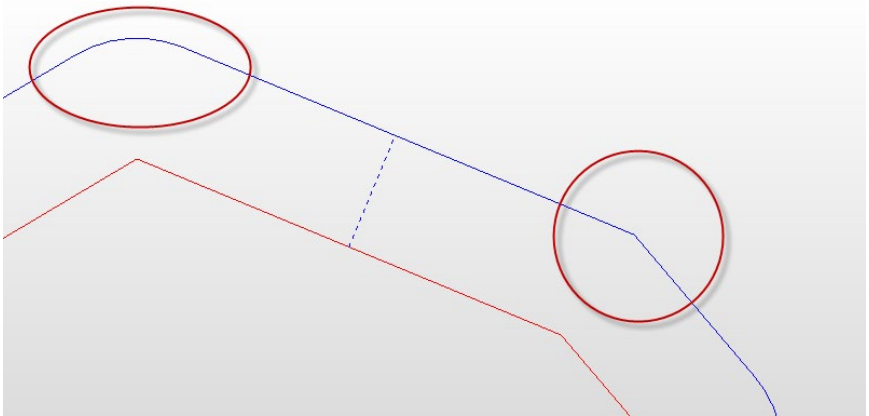
VARIABLE OFFSET

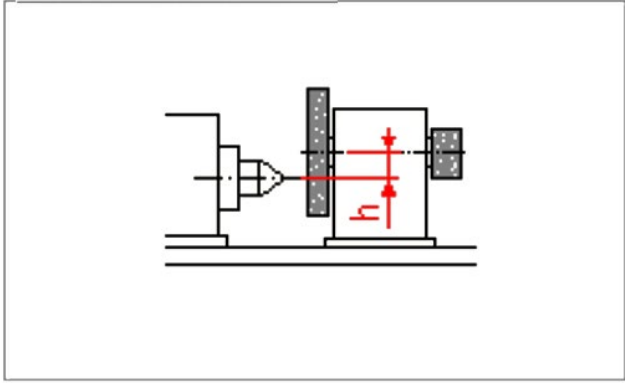
C.OFF OFFS_S= [start_offset] / OFFS_E=[end_offset]

The offset will be gradually applied according to the path length from the first segment to the last segment. The path length is calculated on the input contour. It will not take in account connecting circles. Discarded segments are taken in account.

Connecting circles will always have constant offset. Programmed circle will have variable offset, therefore the resulting trajectory will not be a real arc resulting in error 107 (difference between start radius and final radius higher than the tolerance. To ignore this condition the parameter E32055 has been created and must be set to a proper value to allow it).

4.6.3 ADDITIONAL PARAMETERS FOR C.OFF

ANGLE :	Angle between consecutive segments to replace connecting arcs. Angle value (minimum value: 0°, maximum value 120°) under which connecting arcs between segments are replaced by an extension of the segments with intersection on the bisector (see G92 L in the Flexium+ programming manual). Example C.OFF OFFSET=[OFFS] / ANGLE = 30 
CCPS_F :	Constant Contact Point Speed Filter In case of polar contours, C.OFF works to maintain a constant feedrate at contact point. However, to clean the noise due to the axes dynamics, this feed needs to be filtered which is the purpose of this parameter. The allowed values are 0 – 3 – 5 – 9 – 13 – 17. The higher the parameter the stronger is the filter action. 0 means no filter actions, the value programmed is forced to the nearest allowed value. EG: the actions of values 14,15,16 are identical to 13. This parameter represents the number of points that are averaged. By default, this filter is active and the value is 5.  Left : No filtering, Right maximal filter action CCPS_F = 17 Green : X axis speed Blue : C axis speed Red : Feed rate at contact point

SHFT_X , SHFT_Y :	Shift on X and Y The resulting figure is translated from the programmed values along X and Y
CENT_X :	Centre on Y Parameter to compensate the distance between the centre of rotation and the X axis (for cylindrical grinders). 
COL_OFFS / COL_OFFE :	Columns for storing starting and ending offsets They define in which columns the offsets will be stored. Obviously the table created by C.INIT must contain those additional columns. Example <pre> C.INIT INP=[TAB(7,N)] / OUT=[P] / SIDE = -1 C.OFF OFFS_S=20 / OFFS_E=10 / COL_OFFS = 9 / COL_OFFE = 10 G1 X[P(6,1)] Y[P(7,1)] F[GC(115)] (6,1 / 7,1 go to start point for first point) FOR [I] = 1 TO [N] DO [V] = [P(1,I)]+2 L0 = [P(9,I)] (store start offset of this block) L1 = [P(10,I)] (store end offset of this block) G[GCode(V)] X[P(2,I)] Y[P(3,I)] I[P(4,I)] J[P(5,I)] ENDF </pre>
MONOTO:	Monotonic output This parameter is for polar contour only. It is generally used to create a table suitable for the MLEGB function. With this parameter the rotary axis output will be arranged to be strictly ascending.
PRINT:	Print the output contour on the serial line Reserved for NUM R&D.

Example

Single constant offset to the right

%10.0()

VAR

```
[I][N][V]
[G(3)]=2,1,3
[OFFS]=5
```

ENDV

G77 N1000 N2000

\\Executes profile for reference

P.BUILD [TAB(7,N)] N1000 N2000

C.INIT INP=[TAB(7,N)] / OUT=[P] / SIDE = 1

M6

\\Just to change color in FX3D

C.OFF OFFSET=[OFFS]

G0 X[P(6,1)] Y[P(7,1)]

\\Start point (see P.BUILD)

FOR [I] = 1 TO [N] DO

[V] = [P(1,I)]+2

\\ Refer to P.BUILD for G1/G2/G3

G[G(V)] X[P(2,I)] Y[P(3,I)] I[P(4,I)] J[P(5,I)]

ENDF

M2

N1000

G1 X7 Y-3

G1 X35 Y-15

G1 X43.3 Y-25 EB.2

G2 X-43.3 Y-25 I0 J200

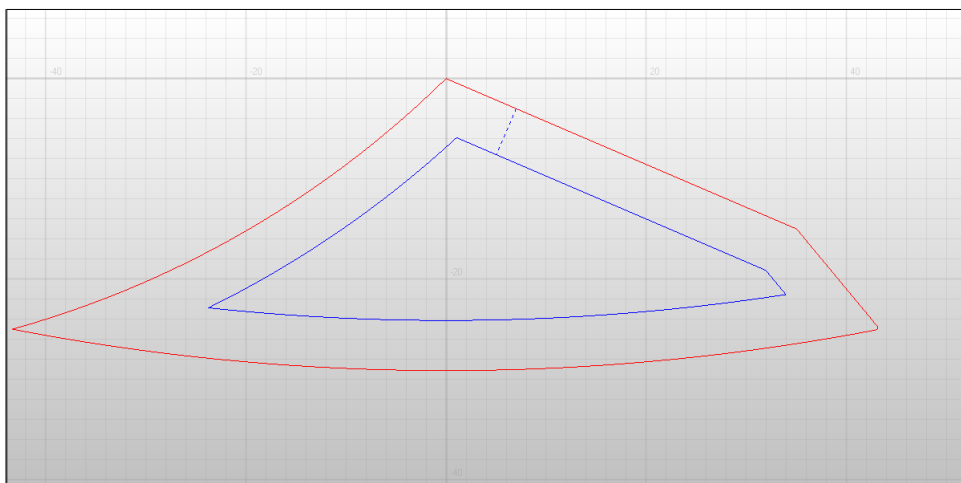
G3 X0 Y0 R100

G1 X7 Y-3

N2000

Red: reference profile

Blue: Profile handled by C.OFF



Example**Multiple variable offsets to the left**

```
%10.0( )
```

```
VAR
```

```
    [I][N][V]
    [G(3)]=2,1,3
    [OFFS]=10
```

```
ENDV
```

```
E32055 = 10000
```

```
\\ Tolerance of radius variation
```

```
G0 X0 Y0
```

```
G77 N1000 N2000
```

```
P.BUILD [TAB(7,N)] N1000 N2000
```

```
C.INIT INP=[TAB(7,N)] / OUT=[P] / SIDE = -1
```

```
REPEAT
```

```
    M6
```

```
    C.OFF OFFS_S=[OFFS] / OFFS_E = {[OFFS]-2.5}
    [OFFS] = {[OFFS]-2.5}
```

```
    IF [OFFS] >= 0 THEN
```

```
        G0 X[P(6,1)] Y[P(7,1)]
```

```
\\ Start point - refer to P.BUILD
```

```
        FOR [I] = 1 TO [N] DO
```

```
            [V] = [P(1,I)]+2 \\ Refer to P.BUILD for G1/G2/G3
```

```
            G[G(V)] X[P(2,I)] Y[P(3,I)] I[P(4,I)] J[P(5,I)]
```

```
        ENDF
```

```
    ENDI
```

```
UNTIL [OFFS] <= 0
```

```
M2
```

```
N1000
```

```
G1 X17.5 Y-7.5
```

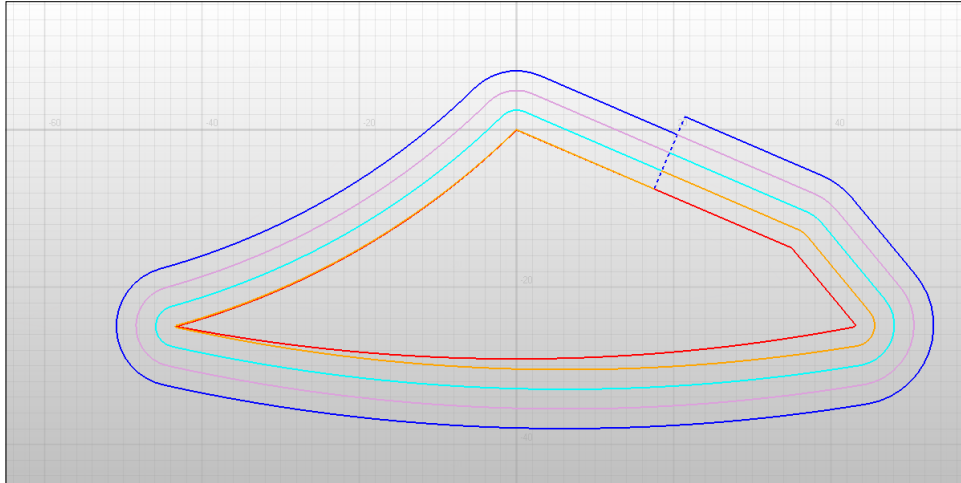
```
G1 X35 Y-15
```

```
G1 X43.3 Y-25 EB.2
```

```
G2 X-43.3 Y-25 IO J200
```

G3 X0 Y0 R100
G1 X17.5 Y-7.5
N2000

Red: reference profile
Blue: Profile handled by C.OFF



4.7 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.

Syntax

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

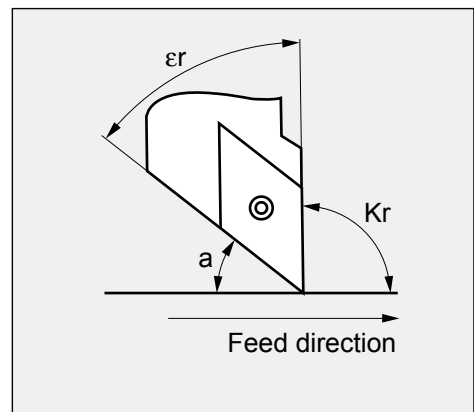
*	When the character * precedes the table name, all the grooves located within the tool relief angle are processed. When the character * is missing in front the table name, only the first groove located within the tool relief angle is processed.
Pa	Table name.
7	Number of table entries.
Nb	Name of the 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 in the ZX Plane

Characteristic angles:

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



Processing of the table

The table analysis begins on the first block and ends:

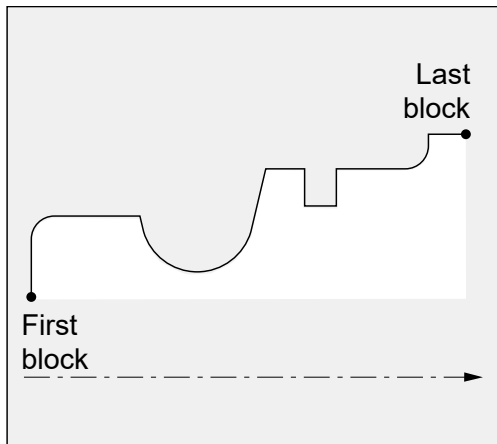
- On the last block with CUT * ...,
- On the first cut with CUT ...



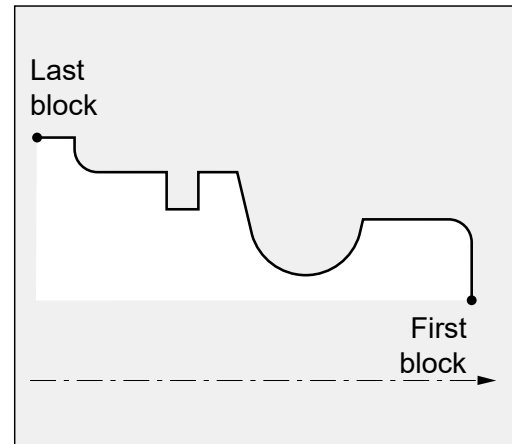
In the table, the profile must always be defined so that on the X axis the profile start dimension is less than the profile end dimension.

Example

Profile correctly defined



Profile incorrectly defined



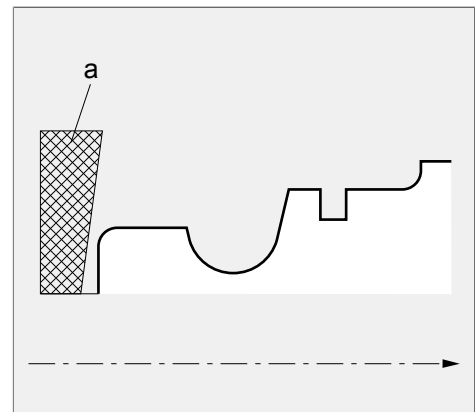
When the clearance angle is negative or zero (between 0 degrees and -180 degrees), the profile areas located below this angle are eliminated.

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

Examples

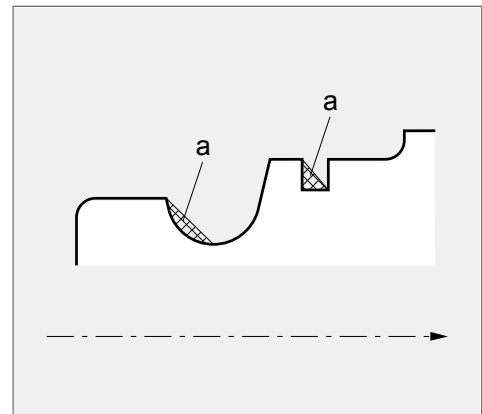
Elimination of the first groove in the profile (no * in front of the variable). «a» shows the areas that are eliminated.

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



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

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 sets 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
```

Syntax

SEARCH [Symb] N.. or SEARCH [Symb] [Para2]	
[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.
[Para2]	L E or symbolic variable. With this syntax [Para2] will return 1 (true) if [Symb] exists and 0 (false) otherwise.

Note: SEARCH uses the same rule for G79 about nesting level.

Example

```

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

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

%1
IF ...
  WHILE ...
    SEARCH [Symb1] N100 (GOOD)
    G77 H2
    SEARCH [Symb2] N50 (WRONG)
    SEARCH [Symb3] N30 (WRONG)
    N30
  ENDW
  N50
ENDI
N100
M2

%2
SEARCH [Symb1] N100 (GOOD)
REPEAT
  IF ...
    SEARCH [Symb2] N100 (GOOD)
    SEARCH [Symb3] N50 (WRONG)
  ENDI
  N50
UNTIL ...
N100
M2

```


4.10 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.

Syntax

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



NOTE

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	%20	%30
N..	N..	N..
N.. G77 H20	N.. G77 H30	VAR [V1][TB(4)]
N..	N..	ENDV
		N..
		SAVE [V1]/[TB(4)]
		N..

4.11 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 swapped /2=6 and /6=2)

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

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

4.11.1 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.

4.11.2 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.

4.11.3 Syntax for partial copy of blocks

MOVE [Pj(nj,mj)],mj1,mj2 = [Pi(ni,mi)],mi1,mi2	
Pj	Target table name.
nj, mj	Entries and blocks of the target table.
nj1, 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.

4.11.4 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

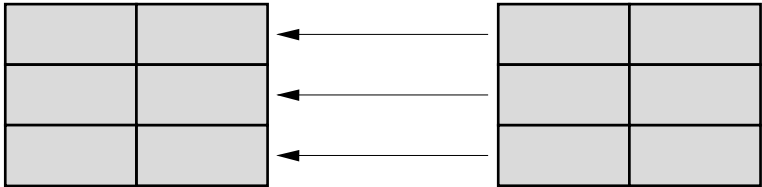
- The MOVE function must be the first word in the block (no sequence number).
- The maximum number of blocks in a finished profile is limited to 95.
- It is possible to reverse the order of a copy by reversing the start and end limit indexes in one of the tables.
- The indexes can be specified in symbolic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

4.11.5 Examples

MOVE simple copy of blocks

Copy the content of a table into another table

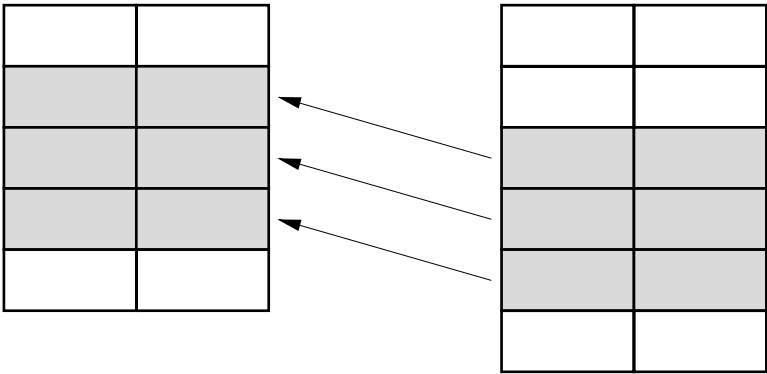
MOVE [PB(2,3)] =[PA(2,3)]



MOVE partial copy of blocks

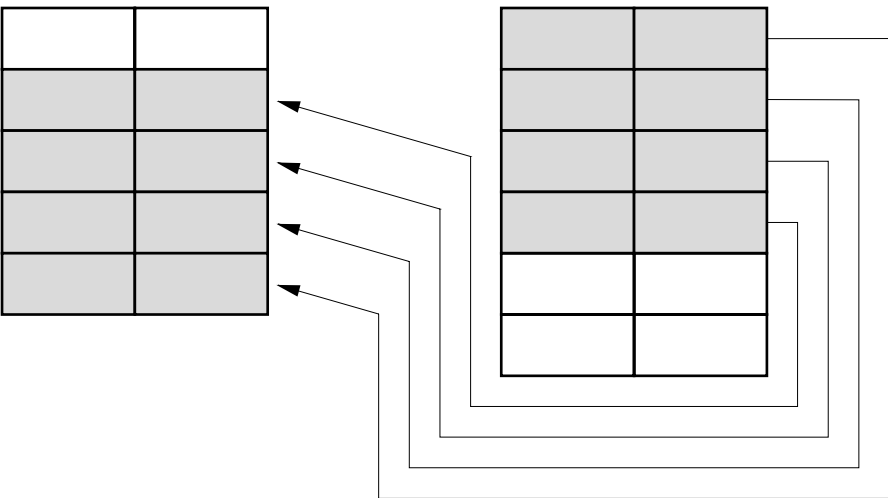
Copy part of a table into part of another table

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



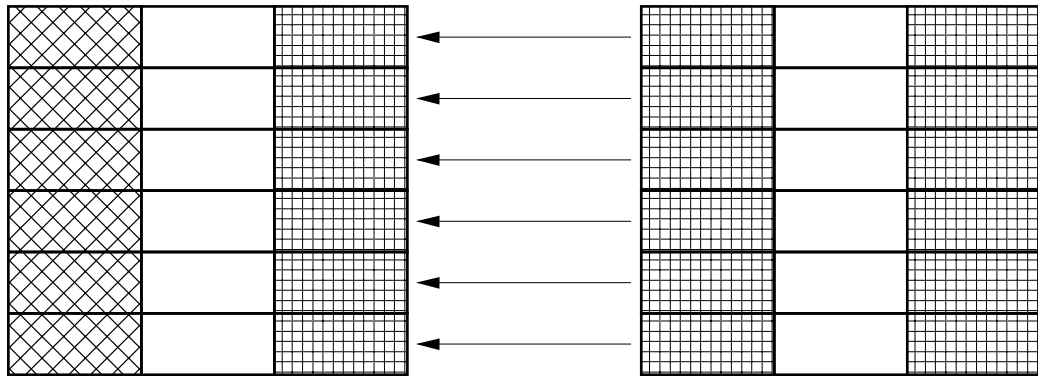
Copy part of a table into part of another table in reverse order

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.

4.12 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 symbolized 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]

The table no with 10 entries is referenced by the value 7

ENDV

[Symb] = 7

L0 = [@Symb(2)]

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

4.13 Programming examples

4.13.1 Use of the BUILD function for back and forth milling

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

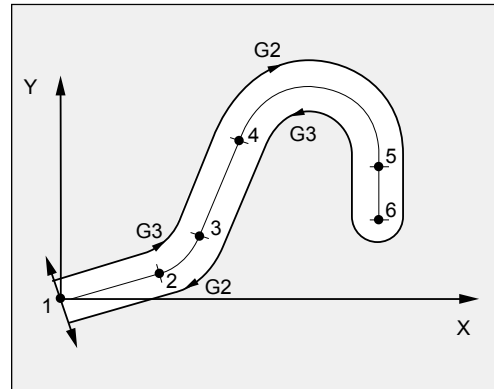


Table built by the program

	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
```

Path from 1 to 6

```
G[TAB(1,I)] X[TAB(2,I)] Y[TAB(3,I)] I[TAB(4,I)] J[TAB(5,I)]
```

```
ENDF
```

```
FOR [I]=[NB] -1 DOWNT0 1 DO
```

Path back from 6 to 1

```
L0=[TAB(1,I+1)]
```

```
IF L0= 2 THEN
```

```
L0=3
```

```
ELSE
```

Inverting G2 and G3 for the return

```
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)]
```

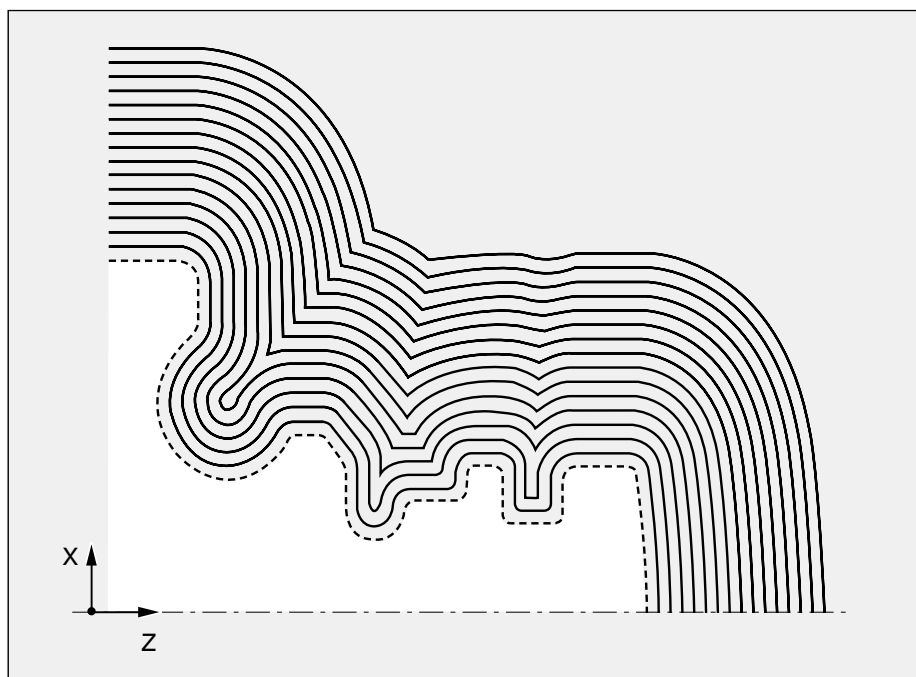
```
ENDF
```

```
G40 X Y
```

```
M2
```


4.13.2 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
VAR
    [R][G(3)]=2,1,3 [I] [V]
ENDV
```

Table creation

```
FOR [R]= 30 DOWNT0 2 BY 2 DO
```

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

```
MOVE [P(7,M)] = [C(7,N)],2,[N]
R.OFF [P((7,M)] / + 1 / [R]
```

Copy in another table
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

Z60

G2 X15 Z50 I15 K55

G1 X25 EB-4

Z40

G2 I30 K30 ES+

G1 EA90 X50 Z25 EB3

N110 Z10



Profile definition

4.13.3 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.

4.13.3.1 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.

4.13.3.2 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 M00040.

4.14 Creating / Deleting Binary Files G76+/- H<bin>

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

4.14.1 Syntax for binary file creation

G76+ H<prog-number> = 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 not present 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.

4.14.2 Syntax for file deletion

G76- H<prog-number>

It is not possible to overwrite an already existing program. The syntax to delete an existing binary file is **G76 H-**. It will not generate an error if the program does not exist in NC memory.



This syntax is valid for the deletion of any type of file and not only for binary files created with G76+.

Example**%1**

G76+ H100 =H20/X/Y

E81000=@H100

← E81000 will point the beginning of the file
(relative offset to the first word of data)

%20

N10 X10

N20 X20 Y50

(empty block)

N30 Y60

N40

N50 X40 Y80

Program created

```

xxxxx      File Header <-- This is the 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)
0x1300 1300 (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 =@ contains the offset to the first block, it is necessary to search for the first word in the data table when opening it via the dynamic operator. This first block follows 0x1300 after the program number; the end of valid data is identified by the sequence 0x13001300.



Even though the E parameters of Flexium+ are real values, the data saved by G76+ remains 32 bit integer.

4.15 Saving and retrieving symbolic variables

In addition to saving variables in a dedicated file, it is now possible to copy part or the total of the symbolic variables contained in the stack in a memory location. This memory location is organized by segments (up to 32) defined by machine parameter P59.

The size of each segment is limited to 65535 bytes the total length of this location being 1024 Kbytes. Please note that these memory locations are not saved after a shutdown. And that this feature can be used by NUM applications (VEComp for example)

This feature is mostly dedicated to be used with dynamic operators in C.

4.15.1 Syntax for saving to a memory segment

G76+ S [symbolic variable 1 name(a,b,c,d)] [symbolic variable 2 name(i,j,k,l)]**

No additional programming is permitted in this block. A possible simplified syntax is also allowed when the list of values is restricted to one symbolic variable:

G76+ S [symbolic variable name(a,b,c,d)]**

The argument S gives the number of the sub-segment (from 1 up to 32).

The first variables specified indicated the first variable whose value(s) will be copied in the memory segment. The second variable gives the last variable whose value(s) will be copied in the memory segment. Between the two ones, the symbolic variables existing in the program stack are copied as well.

The list of variables must be included in a same level of subprogram. The variable 1 has been created before variable 2. The search of the variables is performed from the current level of subprogram up to the level of the main part program.

The firmware controls the name and also the format of the variables.

In the search process, the firmware makes the assumption that only one name searched exists at most in a level of subprogram whatever its format. If a name of variable is found but the format expected is not the right one, the search switches to the next level of subprogram (i.e. search aborted in the current level of subprogram).

The memory segment must be empty (it is not possible to add new values to already existing values).



In order to prevent unwanted changes in the segment, the non channel specific E parameter E31003 is created to prevent writing access to the segments when SET.

4.15.2 Syntax for clearing a memory segment

```
:G76- S**
```

The S argument S defines the number of the memory segment (from 1 up to 32).

No additional programming is permitted in the block.

The firmware clears the full length of the specified memory segment.

4.15.3 Syntax for reading a memory segment

```
G76+ [symbolic variable 1 name(a,b,c,d)] [symbolic variable 2 name(i,j,k,l)] S**
```

This is the reciprocal operation to “*saving to a memory segment*”. It copies values stored at a time in a structure to symbolic variables.

The features and constraints are the same ones than those for “*saving to a memory segment*” (refer to this paragraph).

A possible restricted syntax is also allowed: *G76+ [symbolic variable 1 name(a,b,c,d)] S*** when the list of values is restricted to one symbolic variable.

When all conditions are fulfilled, the values of the segment specified are copied in the symbolic variables.

As for the operation “*saving to a memory segment*”, the size of the structure (reserved not necessarily used) should be equal or greater to the size of the list of symbolic variables. This means that all symbolic variables are written by data coming from the structure (possibly null for the last ones if the size used in the structure is lower to the size of the list of symbolic variables).

Please note that the variables must be declared before restitution.

4.15.4 Getting access to the address of a segment

```
E80000 = @S**
```

The address returned is the address of the descriptor of the specified sub-segment defined as below.

```
// descriptor of a segment
typedef struct s_DescStVarSymb
{
    UINT32 NbElementFull;           // number of double available
    UINT32 NbElementUsed;           // number of double used
    double* Pt_StVarSymb;           // address sub-segment of data
} DescStVarSymb;
```

Page deliberately empty

Summary

5	Program stack - Symbolic variables - Strings	99
5.1	Program stack	99
5.1.1	Use of the stack	99
5.1.2	Stack Allocation	99
5.2	Saving and restoring ► L variables	100
5.2.1	Push Function: ► PUSH	100
5.2.2	Pull Function: ► PULL	101
5.3	Symbolic variables: ► VAR, ENDV	103
5.3.1	Symbolic variables declaration: VAR and ENDV	103
5.3.2	Using symbolic variables in ISO programs	104
5.3.3	Getting the variables in the stack of another channel: ► VAR H.. N.. N..	106
5.3.3.1	Programmed deletion of variables: ► DELETE	107
5.4	Strings	108
5.4.1	String functions	109
5.4.2	String functions returning a value	109
5.4.3	String function without return value	109
5.4.4	String functions for user interaction	110
5.4.5	String functions for exchange with the PLC	110

Page deliberately empty

5 Program stack - Symbolic variables - Strings

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: 0-99, 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

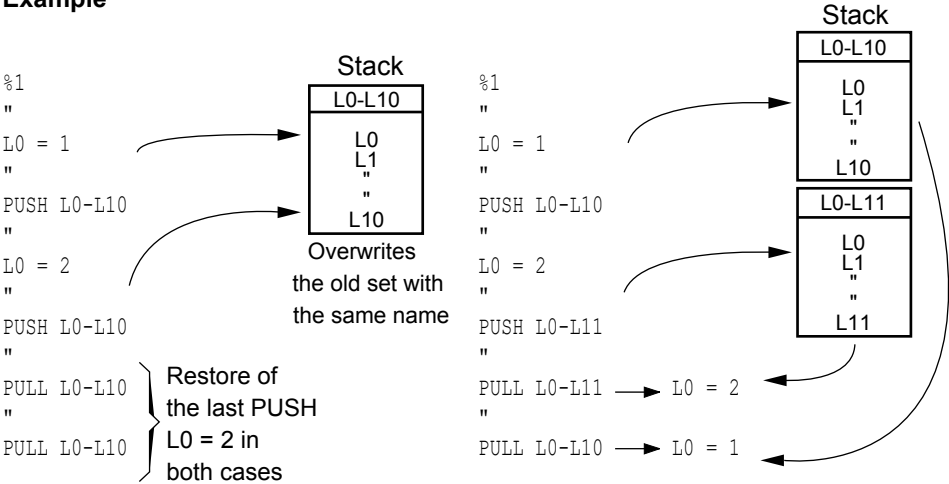
PULL Lm - Ln

Lm-Ln	L variables (m to n inclusive) whose values are to be saved. Number ranges: 0-99, 100-199, 900-959.

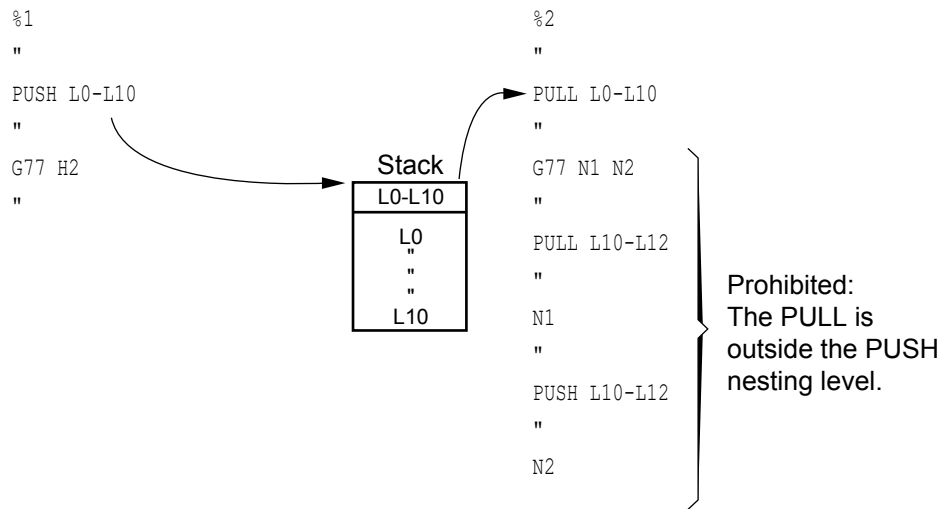
Notes

- 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 32 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.



- 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 authorized 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 up to 32 bytes (according to the name length) bytes for header plus 8 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)
```



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.

As for L or E variables it is possible to perform calculations on symbolic variables, define priorities with curly brackets and assign the result directly to a program address.

Example

```
VAR
[Test] = 1234
ENDV
L0 = L0+1
X{[Test] + L0}
M0
```


Example

Use of symbolic variables for machining an ellipse

```

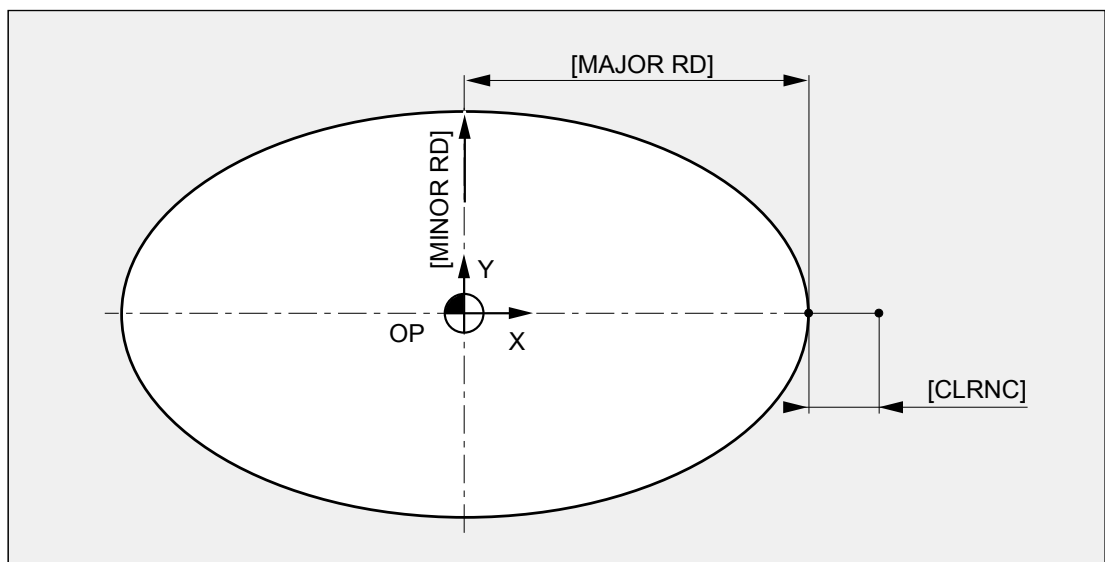
%35
M999

VAR
[SPSPD] = 1000 [DIRN] = 3 [SPDRNG] = 40
[FDRT] = 200 [CLRNC] = 120 [DPHYZ] = 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-[DPHYZ]
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.



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                                %2
N10 ...                          N10
N..                               N..
VAR H2 N50 N60 L0=[I] L1=[T(1,3)] VAR
N..                               N50 [I] = 3 [J] = 4
                                [T(2,3)] = 3, 4, 5, 6, 7 , 8
                                N60
                                ENDV
```

Results

L0 = 3
L1 = 7

5.3.3.1 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 current program or subroutine.
[symb1] / [symb2]	Variables to be deleted.

Deleting symbolic variables from the stack

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

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 reinitializes the stack.

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 /.

Example

DELETE [IX]/[TAB1]/[PROF1]

Deletion of variables

DELETE *[I2]/[TAB3]/[PROF2]

Deletion of variables from the stack

5.4 Strings

Starting with version 4.1.10 Flexium* can handle strings.

Strings can be stored in parameters E83000 to E83999 / E84000 to E84999 as well as in symbolic variables.

Each parameter can store up to 8 characters of a string. If a string is more than 8 characters the next parameter in the series will be used automatically.

The strings are not modified by a reset but not saved at shut down.



The end of a string is identified by the value 0. Therefore the total size in memory is the number of characters + 1.

Let's suppose the string "1234567812345678" stored in parameters E83000 and E83001, the parameters E83002 will be impacted as well to store the final 0.

Strings on symbolic variables

A single symbolic variable can store a string of up to 7 characters (due to the termination 0).

However it is possible to create longer string using table of variables. The total size available becomes $8 * \text{table dimension} - 1$ and the characters will be dispatched over this full size.

Example

```
VAR
    [String1(10)]                Room for 79 characters

ENDV

_SCPY [String1(1)]/"This is a quite long string"    Deletion of variables from the stack
_PUTSTR[String1(1)]                               Shows : This is a quite a long string
_PUTSTR[String1(2)]                               Shows : a quite long string
_PUTSTR[String1(3)]                               Shows : long string
```

5.4.1 String functions

All string functions start with the character underline “_”.

Each parameter can store up to 8 characters

5.4.2 String functions returning a value

_SLEN	Returns the length of the string (number of characters without the final 0) <i>L0 = _SLEN E83000</i>
_SCMP S1 / S2	Compare two strings separated by ' / '. The return value is < 0 if S1 < S2, > 0 if S1 > S2, = 0 if the strings are equals. The comparison is case sensitive and based on the ASCII value of the characters of the string. The second argument can be a parameter E83xxx/E84xxx or a string within double quotes <i>L0 = _SCMP E83000/"MYSTRING"</i>
_SISTR S1 / S2	Returns the index of the first occurrence of string S2 in string S1. If S2 does not exist in S1 the return code is -1 . The second argument can be a parameter E83xxx/E84xxx or a string within double quotes. Suppose E83000 = "Message from PLC" <i>L0 = _SISTR E83000 / "from" -> E81000 = 9</i>
_SSUB S1/S2/ StIndex/EndIndex	Extract from S2 a string between the StIndex and EndIndex characters and returns the result in S1. Suppose S2 = "NUM high end application!" <i>_SSUB S1/S2/1/3 ==> S1 "NUM"</i>

5.4.3 String function without return value

_SCPY S1 / S2	Copy string S2 into S1. The second argument can be a parameter E83xxx/E84xxx or a string within double quotes. This function is currently the mean to define as string using a constant <i>_SCPY E83000/"My own string"</i> .
_SCAT S1 / S2	Add the string S2 at the end of S1. The second argument can be a parameter E83xxx/E84xxx or a string within double quotes. <i>_SCAT E83000/"End"</i>
_SNDEL S1/n/I	Deletes n characters from string S1 starting from index I. Suppose E83000 = "Message from PLC" <i>SNDEL E83000 /5/8 -> E83000 = "Message PLC"</i>
_SNINS S11 / S2 / I	Inserts the string S2 into string S1 starting from index I. The second argument can be a parameter E83xxx/E84xxx or a string within double quotes. Index can be a direct value or a parameter <i>_SNINS E83000 / "to " /9 -> E83000 = "Message to PLC"</i>

5.4.4 String functions for user interaction

_PUTSTR S1	Shows string S1 to the user. Similar to <i>\$ command _PUTSTR E83000</i>
_GETSTR S1	Stores the string written by the user into variable S1. Similar to command = <i>\$ _GETSTR E84000</i> stops the program and wait for the user to write a string. The string will be stored into variable E84000.

5.4.5 String functions for exchange with the PLC

The NCK can exchange strings with the PLC.

Those strings are limited to 35 characters and are available in the PLC via the two variables *Flexium_NCK.RCNC.PLCString* and *Flexium_NCK.WCNC.PLCString* (see Flexium* Commissioning manual M00044)

_SETSPLC S1	Writes the string S1 into <i>RCNC.PLCString</i> <i>_SETSPLC E83000</i> .
_GETSPLC S1	Read a string from <i>WCNC.PLCString</i> and copy it into variable <i>S1_GETSPLC E84000</i> .

Page deliberately empty

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
A

Page deliberately empty

Summary

6	Subroutine call by G Functions	115
6.1	General	115
6.2	List of G functions calling a subroutine	116
6.3	Particularities	116
6.4	Theory	117
6.4.1	Preliminary	117
6.4.2	Operation	117
6.5	Subroutine example	118
6.5.1	Cycle syntax and parameters	118
6.5.2	Graphical representation	118
6.5.3	Main machining program	119
6.5.4	Cycle subroutine	119
6.6	Custom cycle example 2	121
6.6.1	Cycle syntax and parameters	121
6.6.2	Graphical representation	122
6.6.3	Cycle subroutine	123
6.7	Decrypting a standard canned cycle	125
6.7.1	Syntax of the G83 cycle	125
6.7.1.1	Subroutine %10080 common to all canned cycles	129
6.7.2	Timed M	132
6.7.2.1	Accuracy	132
6.7.2.2	Extreme cases	133
6.7.3	Inter samples	133
6.7.3.1	Delay or anticipation	133
6.7.3.2	Multiple M in one sample	133

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).



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+ Commissioning manual M00044.

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 Flexium+ Programming manual M00040)

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.

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 M00040, 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:
N10	N10
N..	N..
N150 G108 ...	N80 G77 H118
N..	N..

During execution of %10108 (invoked by N150 in %10) 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).

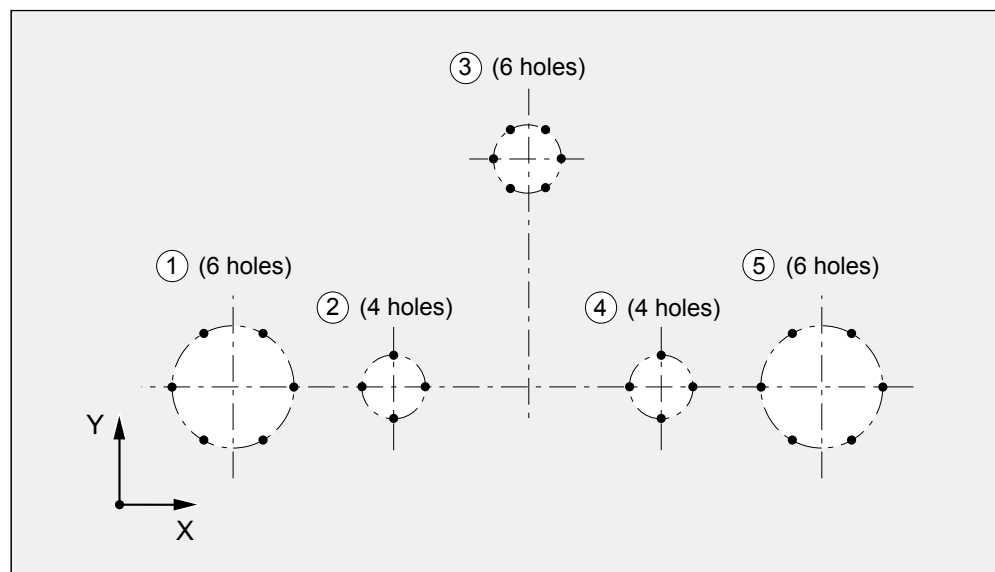
6.5.1 Cycle syntax and parameters

Syntax

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

N.. G182	Cycle for drilling equally spaced holes on a circle.
X.. Y..	Circle center position.
ER..	Approach and clearance position.
Z..	Machining end point.
P..	Number of equally spaced holes.
R..	Radius of the circle of holes.
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 feedrate 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.

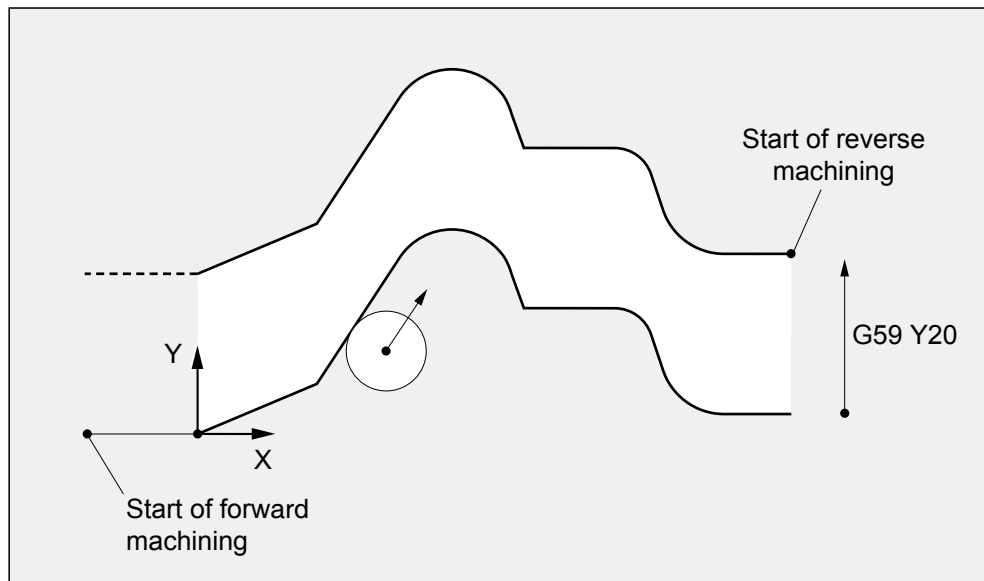
It defines the function G177 which is used to execute a profile by back and forth passes with the possibility of radius correction if required.

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:
	ER40	Machining to tool center.
	ER41	Offset on the left of the profile.
	ER42	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
```

Profile definition

```
N160 G177 N80 N150 ER42
N170 G59 Y20
N180 G177 N150 N80 ER4
N190 G0 G52 Z0
N200 G0 X-100 M5
N210 M02
```

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=	

<pre> 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 </pre>	
<pre> [axis1]= [TAB(3,NB)]* [multi] [axis2] =[TAB(2,NB)] </pre>	Retrieves the end point
<pre> G1 GL943 @Y [axis1] @X [axis2] G998 </pre>	Moves to end point GO !
<pre> FOR [NBLOC]=[NB] DOWNT0 2 DO [G]=[TAB(1,NBLOC)] IF [G]=0 THEN [G]=1 ELSE IF [G]=-1 THEN [G]=3 ELSE [G]=2 ENDI ENDI ENDI </pre>	Interpolation type TAB(1,NBLOC)] = 0 → G1 TAB(1,NBLOC)] = -1 → G3 TAB(1,NBLOC)] = 1 → G2
<pre> [axis1]=[TAB(7,NBLOC)]* [multi] [axis2]=[TAB(6,NBLOC)] [centre1]=[TAB(5,NBLOC)]* [multi] [centre2]=[TAB(4,NBLOC)] </pre>	Calculates coordinates
	Go to position
<pre> G[G] @Y[axis1] @X [axis2] @J[centre1] @I[centre2] ENDF </pre>	Loop for all table
<pre> ELSE G1 GL943 G77 H[H] N[N1] N[N2] ENDI </pre>	Profile defined in direct order Direct execution of the forward profile
<pre> N9900 G80 E11005=[diam] </pre>	Cancels modality 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.



For more detailed descriptions about programming this cycle please refer to the Flexium+ Programming manual M00040.

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 [.IBEL(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 [.IBEL(6)]=1 THEN G4 FL931 ENDI	Optional dwell
[dimnsion]='ER G0 G77 H10080 N[I] N[I] UNTIL 'I = 'L ENDI	Retract to top End of loop
[dimnsion]=L[LZ] G1 F[RF] G77 H10080 N[I] N[I]	Go to bottom of hole
ENDI	
N100 IF [.IBEL(6)]=1 THEN G4 FL931 ENDI	Dwell specified by EF
[dimnsion] = 'ER [I]=[IZ]+10 G0 G77 H10080 N[I] N[I] G997 G9 M[M998] G[G90/1] G[G0/1] F[RF] E11005=[diam]	Retract to ER Go Restore status

6.7.1.1 Subroutine %10080 common to all canned cycles

%10080	Analyse of 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 the axis perpendicular to the tool axis
ENDI	
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
        ENDI
ENDI

[LZ]=[LZ]+26

```

If programming is by diameter, correct 'ER

LZ points to variables L926..L951 on the first block to be initialized
- hole bottom dimension.

```

IF      [..BG80] = 1 THEN L[LZ]=[..IRX(IZ)]
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

```

If programmed, store the new hole
bottom dimension

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

```

Store presence then disable the tool axis

```

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

```

Test spindle rotation if axis is programmed

```

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

G79 N800

```

Error 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)

N11 X[dimension]
N12 Y[dimension]
N13 Z[dimension]
N14 U[dimension]
N15 V[dimension]
N16 W[dimension]

Movements

N800

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

A

Timed M functions

Since release 4.1.10 two new types of M functions are available:

Timed M: End of block M functions partly similar to the M on the fly with two exceptions:

- They can be delayed or advanced with respect to the end of the block by up to +/- 32000 ms. They are processed by the PLC therefore this delay may slightly vary according to scan time and NC<>PLC communication.
- If programmed alone in a block the movement will be interrupted at the end of the previous block.

Inter-samples M: Such functions are also delayed or advanced in respect to the end of the block but they are issued on direct outputs without being processed by the PLC.

Moreover they are issued with an accuracy in the μ Sec range (i.e. this can be between two consecutive real time scans). They are used for example in laser cutting application.

6.7.2 Timed M

There are 32 M codes available from M932 to M963 that can be programmed and timed.

The delay or anticipation is set in the E parameter E3000032 to E3000063; the admitted range is -32000 msec to 32000 msec.

Those functions are all end of block and are sent to the PLC in a similar way as the M functions without acknowledge. Thus the PLC could miss some code if they are sent with a frequency higher than the PLC loop frequency.

Those functions are executed without stopping the axes, **except when they are programmed alone in a block.**

The M code will be executed in advance or delayed (according to the sign of the E parameter) with respect to the reference programmed position issued from the interpolator. The measured position is not considered for issuing the function. Additionally position corrections by dynamic operators are not considered as well.

Only one M at the time can be delayed or anticipated. If a delayed M is executed while another M delayed function is in the pipe waiting the execution then an error will be triggered (Error 83 ext. 1 or 2).

This is also the case for anticipated M, but one anticipated M and 1 delayed could be requested in the same time frame.

6.7.2.1 Accuracy

The execution of such an M function is delayed with an accuracy of one sample, but the transmission delay to the PLC as well as the PLC processing time should also be considered.

If the M code is anticipated the accuracy depends on several factors. Of course the bigger is the anticipation time the higher the error can be. In particular the accuracy is compromised if the feed potentiometer is changed between the moment when the block is processed and the moment when the block is executed.

Another factor is the processing of functions executed between the moment the M is programmed and the instant when it would be actually executed.

The NCK calculates the instant to execute the M taking in account only the axis movement, no dwell time or miscellaneous function are considered.

6.7.2.2 Extreme cases

If an M is delayed and the program ends before the execution moment where it should take place, the end of the program is delayed of the residual time.

If an M is anticipated but there's not enough time in the buffer, a dwell time is introduced in the point where the M is programmed.

6.7.3 Inter samples

Inter-samples M are triggered **exactly in the moment when the axes are in the programmed position in the block within a μSec precision** (this means generally between two consecutive samples).

To maintain this accuracy they are issued directly on direct outputs located on the NCK.

They are up to 32 functions total grouped 2 by two; one setting the output, the other clearing it.

Additional information about the declaration is given in the Flexium+ Commissioning manual M00044.

The M inter samples have fixed code form M900 to M931.

M900/M901	Set/Clear output bit 0
M902/M903	Set/Clear output bit 1

M930/M931	Set/Clear bit 15

6.7.3.1 Delay or anticipation

The M inter-samples can be delayed or anticipated by the parameters E3000000-E3000031 with respect to the functions M900-M931 (value in μsec).

If the value is negative the action (set or reset) is anticipated of that amount of time. For a positive value, the action is delayed.

The range allowed is from -P50 (sampling time) up to 32000 μsec . Thus the M can be anticipated only for a small amount of time and delayed up to 32000 μsec .

6.7.3.2 Multiple M in one sample

Only one single inter samples M can be programmed in each block. If more than one block is to be executed in a single sample only the last one would take effect.

As stated before, the execution of those M doesn't stop the axes motion, except if the M is programmed without a movement then the axes stops, the corresponding bit is Set/Cleared and the movement restarts.

X100	; block ends with axes stop
M900	; M900 is executed with stopped axes
X110	; movement restarts.

Even if the M is executed with stopped axes the delay/anticipation in E3000000-E3000031 is applied.

Page deliberately empty

Summary

7	MLEGB (Multi Level Electronic Gear Box).....	137
7.1	General	137
7.2	G770 : Continuous running of an axis	137
7.3	G771 : EGB Synchro definition / destruction	138
7.3.1	Synchronisation by tables.....	139
7.4	G772: Synchronisation modification	140
7.5	G775 : EGB Activation / Deactivation	142
7.5.1	Syntax for activation	142
7.5.2	Syntax for de-activation.....	142
7.6	G776 : Waiting for EGB Synchronisation completion before proceeding.....	143
7.7	G779 : Synchro-Homing (... or "Gear alignment")	144
7.7.1	Other Syntax.....	144
7.7.2	Example	145
7.8	Important notes	146
7.8.1	Emergency	146
7.8.2	Override	146
7.8.3	Reset.....	146
7.8.4	EGB Improvements.....	147
7.8.4.1	Rotary leader managed as a linear	147
7.8.4.2	Follower axis limits.....	147
7.8.4.3	EGB Follower filter action	148
7.8.5	Useful parameters.....	149

Page deliberately empty

7 MLEGB (Multi Level Electronic Gear Box)

7.1 General

First intended for gear machining, the MLEGB function permits to create a link between axes: a follower and up to five leaders.

This link can be with a fix ratio or according to a table with two types of interpolation between consecutive points.

A full application relies on the following functions:

- G770 : Continuous running of an axis.
- G771 : Master slave definition.
- G772: Synchronisation modification
- G775 : Activation / De-activation of the synchronisation previously defined.
- G776: Wait for effective synchronisation.
- G779: Axes alignment.

7.2 G770 : Continuous running of an axis

This function is intended to start/stop a continuous rotation of a rotary modulo axis, almost like a spindle but with the difference that unlike a spindle this axis is controlled in position.

Syntax

G770 A|B|C M3|4|5 S...

Examples

G770 C M3 S2500	Run C axis forever counter-clockwise @ 2500 rpm.
G770 C M5	Stop C axis rotation.



Unlike an axis a Cycle Stop command will not stop the rotation. This function is based on the N/M auto feature which must be enabled by parameter E912xx = 1 (xx = axis address) as well as in the PLC. (Refer to M00044 Flexium+ Commissioning manual for details)

This function is intended to define/ un-define a synchronisation between two axes: a leading and a following.

The definition is incremental, this means it is possible to add leader axes to an already defined synchronisation; so creating a multi-leader axes to one follower axis synchronisation.

It is possible to define several synchronisations.

The type of synchronisation is defined as well by this function and can be:

- A simple ratio specified by two numbers (numerator and denominator) addresses P and Q
- An interpolated bi-dimensional table with two kinds of interpolation:
 - Linear interpolation (table points are jointed by linear segments)
 - Cubic spline interpolation (table points are jointed by 3rd degree polynomial arcs) code S3.

Syntax

Synchronisation with a ratio

G771+ <Follower> <Leader> P... Q...

P and Q could be real value with max 7 digit for integer part and 8 for decimal part

P1234567.12345678

Synchronisation defined by a table with linear interpolation

G771+ <Follower> <Leader> [Table]

Synchronisation defined by a table with polynomial interpolation

G771+ <Follower> <Leader> [Table] S3]

Cancel a synchronisation

G771- <Follower>

Cancel all synchronisations of the channel

G771-

Synchronisation is incremental this means that several consecutive G771 functions for the same follower will be programmed to define all its leading axes. Each synchronisation can be of a different type.

Cancelling synchronisation is NOT incremental *G771- <Follower>* kills the link between the follower and all its leaders.

G771 defines the synchronisation but do not activate it. An other code is required.

G771- can be called in the %11000 (reset routine).



Pay attention to the order. First call G775- and only then G771-.

7.3.1 Synchronisation by tables

The tables are defined before the synchronisation. They are defined as bi-dimensional as they involve only one leader and one follower.

Of course it is possible to use the same table for different synchronisations:

- The first column is the leader axis reference.
- The second column is the follower axis reference.
- If the leader axis is a rotary modulo axis, then the table must be intended as periodical table.



From version 4.2 the table size is not limited to 255 anymore but defined by P58 N2. On the other hand each leader cannot accept more than one follower.

Example according to the rules below.

Follower	1 st leader	2 nd leader	3 rd leader	4 th leader	5 th leader
X	-	-	-	-	-
Y	X via "Table"	-	-	-	-
Z	-	-	-	-	-
U	-	-	-	-	-
V	-	-	-	-	-
W	-	-	-	-	-
A	-	-	-	-	-
B	-	-	-	-	-
C	Y via P, Q	A via P, Q	-	-	-

VAR

```
[Table(2,12)] = 0, 0.0
               = 30, 22.5
               = 60, 32.5
               = 90, 33.0
               = 120, 17.5
               = 150, 16.5
               = 180, 2.5
               = 210, -10.0
               = 240, -12.5
               = 270, -10.0
               = 300, 0.0
               = 330, 0.7
```

ENDV

G771+ Y X[Table] S3	Link between X and Y (leader) via a table with polynomial interpolation.
G771+ C A P37 Q41	Link between C and A (leader) via a ratio.
G771+ C Y P1.414 Q3.14	Link between C and Y (second leader) via a ratio as well.

7.4 G772: Synchronisation modification

The G772 function (available from version 4.2.0) allows synchronisation and desynchronisation of an EGB link previously synchronised with G771/G775 as well as changing of the synchronisation ratio on the fly.

Syntax

G772-

Desynchronisation on the fly (with no stop of look-ahead or axis movement).

G772+

Synchronisation on the fly (with no stop of look-ahead or axis movement).

G772=

Changing the P /Q ratio on the fly.

Examples

1)

G772- Z C desynchronise the link Z-C

G772+ Z C synchronise the link Z-C

G772= Z C P-10 Q3 change the ratio of the link Z-C at the value -10/3



All those functions executed on the fly may cause shocks on the follower axis. To avoid them it is necessary to activate the filter on the follower setting via parameter E7x212.

Examples**2)**

```

// Step 1
E72213 = 0           (Z negative limit 0)
E72214 = 200/U1      (Z positive limit 200 mm in internal units)
E72215 = 0           (Z values are OP origin)
// Step 2
G01 Z0               (Move to starting position along Z)
// Step 3 – Init MLEGB
G771+ Z C P10 Q360   (Ratio 10:360 → Means Z axis will do +10mm per positive 360°
                      turn of C axis)
G775+ Z C             (Activation of Synchronisation of Z to C axis)
G776 Z (Waiting for MLEGB Synchronisation completion before proceeding)
// Step 4 – Move to upper Z Limit
FOR L0 = 0 TO 20 THEN
    C360 (Z axis will move by +10mm for each completed rotation of C axis)
ENDF
// Step 5 – Change ratio on the fly
G772= Z C P-10 Q360  (Ratio -10:360 → Means Z axis will do -10mm per positive 360°
                      turn of C axis)
// Step 6 – Move to lower Z Limit
FOR L0 = 0 TO 20 THEN
    C360 (Z axis will move by -10mm for each completed rotation of C axis)
ENDF

```

7.5 G775 : EGB Activation / Deactivation

Once all synchronisations have been defined, and synchro homing performed, then it is possible to activate the synchronisation.

EGB activation is NOT incremental; the ISO code syntax needs to specify only the follower axis.

7.5.1 Syntax for activation

G775+ <Follower>

7.5.2 Syntax for de-activation

G775- <Follower>



If there is no follower axis specified with G775 - then all synchronisation previously activated will be deactivated G775+ without specification of a follower axis is not authorized.

G775- can be called in the %11000 (reset routine).



Pay attention to the order. First call G775- and only then G771-.

7.6 G776 : Waiting for EGB Synchronisation completion before proceeding

After EGB activation it is probably necessary to wait for the follower axis to be perfectly synchronised to all of its leaders before proceeding with the execution of the part-program.

But, sometimes it is possible to save part-program execution time by starting to move immediately other axis, not involved in EGB machining, while waiting for EGB synchro completion.

The function G776 is intended for this.

Syntax

G776 <Follower> F...

F	The F value represents a Time-out in seconds.

The function works as below:

- Hold part program execution until the synchronisation of <Follower> is complete and then resume immediately.
- If synchronisation is not complete before the time out then generate an error and stops execution.



EGB calculation is always performed in reference to a specific position of all involved axes: the follower and all its leaders.
Such a specific position reference, called the "*synchro-homing*" reference is the position of perfect coupling of all involved axis.

This function is intended to define the perfect coupling before activating the synchronisation.

The user has to move the axes involved in the EGB synchronisation, to the "perfect coupling position", before launching the ISO instruction G779.

This action will define the start position for synchronisation.

Syntax

G779 <Follower>

G779

Only specifies the follower axis.

All leader axes involved in the same EGB synchronisation must be in their own "*Synchro-Homing*" position !

7.7.1

Other Syntax

Two other types of "synchro homing" are possible:

G779+ <Follower> <Leader1>... <leader n>...

The "*Synchro Homing*" position of several leader axes will be incremented by the value given in argument like an offset.

G779= <Follower> <Leader1>... <leader n>

The "*Synchro Homing*" position of several leader axes will be set to the value given in argument.



- The first syntax (G779) allows "**probing**" of the correct Synchro Homing position.
- The second syntax (G779+) allows "**offsetting**" of Synchro Homing.
- The third syntax (G779=) allows assignment of Synchro Homing to a previously known synchronized position.

7.7.2 Example

```

.....
G771+ C A P37 Q41      C-A EGB definition.
.....
G01 A45 C135           Move independently C and A axis to their synchro-homing.
G779 C                 Freeze synchro-homing position.
.....
G770 A M3 S5000        Start rotating A axis @ 5000 rpm counter clock-wise.
.....
G01 X... Y...          Prepare any other movement; note that C is still at standstill.
.....
G775+ C                Start synchro "on the fly" of C to A; C starts to accelerate, to reach A.
.....
G01 X... Y...          Auxiliary movements, while C is synchronizing its reference to A.
.....
G776 C F4              Wait here until synchro completion for 4 seconds maximum.
.....
G01 X...Y...           Continue with PP... as soon as synchronisation is complete.

```



When the time out programmed is expired without reaching the synchronisation condition, an emergency message is given and then execution stops.

7.8 Important notes

7.8.1 Emergency



- Out of synchronisation could lead to serious damage on the machine.
In order to prevent these damages an emergency retract procedure should be prepared (G75).
- When the emergency retract procedure is executed due to an out of sync error then the value 3 will be written in the E11022 parameter.
Parameter E11023 gives the physical address of the follower axis which went in emergency.
- If there is no emergency retract procedure programmed then error 162 extension 2 is issued.

7.8.2 Override



M49 does not affect the axes launched by G770.
They have their own override (refer to the OEM operation manual).

7.8.3 Reset



- The Reset function doesn't cancel the synchronisation except in case of synchronisation error. This is done to allow the operator to move the axis in manual or MDI after an emergency, maintaining if it is possible the synchronisation.
- If an axis is rotating in NM-AUTO with G770, Reset will stop it, waits that it is fully stopped and then terminates.

7.8.4 EGB Improvements

EGB primarily use is for gear cutting. However the advanced features could be used for other functions for example non-circular grinding.

For this several improvements have been developed (available from version 4.1.10).

7.8.4.1 Rotary leader managed as a linear

If a rotary axis could be managed as linear, it becomes possible to define in the table all cylindrical grinding cycle (see table below).

To enable treatment of rotary leader axis as linear the E parameter E7x210 must be set, where x stands for the axis name in the channel.

E7x210 is only read during the G771 function and the functionality will be maintained until the link is cancelled with G771- regardless of the actual value of the parameter.

This allows a different behaviour of the axis in different EGB link. Indeed, an axis might be simultaneously treated as linear and rotary in multiple EGB links.

Example of application

C Axis	OP Z Position	Comment
C360*0	Z0	Z start at 0 position when C is at 0
C360*2	Z0	First two turns of C and Z doesn't move. Infeed phase with X axis
C360*12	Z100	10 turns of C and Z makes a 100 mm movement to the end position
C360*14	Z100	Two turns and Z doesn't move. Infeed phase with X axis
C360*24	Z0	10 turns of C and Z makes a 100 mm movement to the start position
C360*26	Z0	Two turns and Z doesn't move. Infeed phase with X Axis
C360*36	Z100	10 turns of C and Z makes a 100 mm movement to the end position
...	...	Complete the cycle until the final depth

With this table the Z position is not related to the absolute position on C but it keep track of the actual revolution done.

7.8.4.2 Follower axis limits

It is possible to limit the follower axis target position using parameters E7x213 and E7x214 respectively for the negative value and the positive value.

The value is interpreted in machine origin or work piece origin according to E7x215. If E7x215 is set the values are interpreted as machine origin, otherwise they are in work piece origin.

The follower target position is limited according to the nominal acceleration and velocity.

Approaching the limit positions programmed:

Although approaching and when out of the limits the axes do not maintain the programmed synchronisation, the synchronisation alarm is not raised and axes remain considered as effectively synchronised.

Synchronisation on the fly:

As the target position returns within the limits the follower axis resumes following the leader, therefore it is necessary that follower maximum speed and acceleration are higher than leader's.

7.8.4.3 EGB Follower filter action

The follower position is strictly linked to the leader position. This means that no control on velocity or acceleration of the follower axis occurs, it simply follows the leader whether or not it had to overcome the dynamic limitations.

With parameter E7x212 set it is possible to force the follower to respect its own velocity, acceleration and/or the position characteristics. This inevitably means that the leader-follower link might be not respected as the leader is by definition not influenced by the follower which can be limited in his dynamic response.

7.8.5 Useful parameters

E7x200 - 7x205	Contains the EGB synchro homing programmed with G779. E7x200 to 7x204 contains the first to fifth leader homing value, 7x205 contains the follower homing value.
E7x206	Contains the EGB synchro tolerance.
E7x207	Contains the current synchro error.
E7x208 - E7x209	Contains the minimum and maximum value of the EGB synchro error (E7x207).
E7x210	0 = Set axis to linear 1 = Set axis to rotary
E7x211	0 = MLEGB works in the OP workframe
E7x212	0 = No position, velocity or acceleration control on this follower axis. 1 = The axis will respect the position, velocity and acceleration limits.
E7x213	Lower Travel Limit of the axis
E7x214	Upper Travel Limit of the axis
E7x215	0 = The values of E7x213 and E7x214 are interpreted in the OP workframe 1 = The values of E7x213 and E7x214 are interpreted in the OM workframe

The range of value for these parameters is similar to the range of axes position (double).

E7x200 to E7x206 are RW

E7x207 to E7x209 are read only

Page deliberately empty

Summary

8	Dynamic operators	153
8.1	General	153
8.1.1	Virtual axes	153
8.1.2	List of operation codes	154
8.1.3	General syntax of an operation	155
8.1.3.1	Example of definition of an operation	155
8.1.4	Cancel an operation	155
8.1.4.1	Cancel an operation by operation code = 0	155
8.1.5	Order of execution of the operations	156
8.1.5.1	Execution of consecutive operations	156
8.1.6	Dynamic operations can be used by any channel	156
8.2	Types of operands used.....	157
8.2.1	Type 1 external parameters	158
8.2.2	Type 2 external parameters	159
8.2.3	Type 3 external parameters	165
8.2.4	Immediate values	166
8.2.5	Operands pointing two consecutive memory locations	166
8.2.6	L program variables	166
8.2.7	Symbolic variables	166
8.2.8	Axis Addresses	166
8.2.9	NCK Address	166
8.3	Dynamic operators execution	167
8.4	Code1 ▶ Add	168
8.5	Code2 ▶ Subtract	169
8.6	Code3 ▶ Multiply	170
8.7	Code4 ▶ Rotate a vector	171
8.8	Code5 ▶ Load a parameter with or without mask	172
8.9	Code6 ▶ Load of a peripheral memory with or without mask	173
8.9.1	Load an axis address	173
8.9.2	Load from PLC Data	174
8.9.2.1	Parameters E47000 to E47124	174
8.9.2.2	Parameters E4300x	174
8.9.2.3	Parameters E4400x	175
8.9.2.4	Parameters E4410x	175
8.9.2.5	Parameters E4600x	175
8.10	Code7 ▶ Indexed load of a parameter	176
8.11	Code8 ▶ Indexed storage of a parameter	177
8.12	Code9 ▶ Compare two parameters	178
8.13	Code10 ▶ Conditions on the sum of two parameters	180
8.14	Code11 ▶ Store data in a peripheral memory	182
8.14.1	Storage at an axis address	182
8.14.2	Store in a PLC memory	183
8.14.2.1	Parameters E10000 to E10031	183
8.14.2.2	Parameters E3300x	184
8.14.2.3	Parameters E3400x	184
8.14.2.4	Parameters E3410x	184
8.14.2.5	Parameters E3600x	184
8.14.2.6	Parameters E37000 to E37127	185

8.14.2.7	Parameters E3410x	186
8.15	Code12 ▶ Increment with modulo	187
8.16	Code13 ▶ Multiply by a program variable	188
8.17	Code14 ▶ Arc tangent function	189
8.18	Code15 ▶ Axes manual control	190
8.19	Code16 ▶ Third-degree polynomial and derivative	191
8.20	Code17 ▶ Speed on trajectory	192
8.21	Code18 ▶ Divide	193
8.22	Code19 ▶ Square root	194
8.23	Code20 ▶ Axis speed synchronisation	195
8.24	Code21 ▶ Oscillation cycles	197
8.24.1	Time base modulation	198
8.25	Code22 ▶ Execute a C dynamic operator	199
8.25.1	Development tools	201
8.25.2	Steps for creating a project	202
8.25.3	Modules supplied by NUM	203
8.25.4	Reading parameters	203
8.25.4.1	E parameters	203
8.25.4.2	L parameters	204
8.25.5	Structure of the C modules	205
8.25.5.1	init (UINT16 gr_no, UINT16 no_arg, ARGUMENT *)	206
8.25.5.2	princ (void)	207
8.25.5.3	quitt (void)	208
8.25.6	Limits and precautions	208
8.25.7	Transferring the code for execution	208
8.25.8	Advanced features	208
8.25.9	Error messages	209
8.26	Code23 ▶ Electronic cam	214
8.27	Code24 ▶ Differential shift	215
8.28	Code25 ▶ Read current block data	217
8.29	Examples of dynamic operators programs	219
8.29.1	Example 1	219
8.29.2	Example 2	220
8.29.3	Example 3	221

8 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.

8.1 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 X81000/X81020/X80055/3600000`.

8.1.1 Virtual axes

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

A virtual axis (or spindle) is declared in words N4 to N7 of P120 (without measure and without set value). 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 M00041.

8.1.2 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
25	Read a field in current block

8.1.3 General syntax of an operation

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

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

Operation number	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 operation 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.

8.1.3.1 Example of definition of an operation

O35 = 2 E81000 / E70000 / E71000 /2

8.1.4 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.

8.1.4.1 Cancel an operation by operation code = 0

Syntax

On = 0

This syntax does not include any operands.

8.1.5 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 ...
```

8.1.5.1 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

8.1.6 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.
A CNC or PLC reset cancels all the operations for all the CNC channels.

8.2 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 inter-axes 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 M00040.

8.2.1 Type 1 external parameters

E7x000	Position reference of an axis in the channel
E7x001	Sampling the axes positions for on-the-fly probing
E/x008	Axes with a programmed movement outside E6x0008 - E6x0009 boundary
E7x100	Interpolator position references
E79000	Reference position of the current measured spindle
E79001	Speed demand for the current main spindle
E79002	Value of the feed rate potentiometer assigned-the channel
E79003	Distance remaining-be travelled in the block being executed
E79013	Feed rate override writeable with the Dyn Operator
E81000- E81999	Master axis position reference
E82000- E82999	Slave axis corrections with respect-the master axes
E83000- E83999	General purpose DOUBLE R/W parameters
E84000- E84999	General purpose DOUBLE R/W parameters
E88000- E89999	General purpose DOUBLE R/W parameters reserved for NUM.
E90000- E90031	Physical axis measure value
E90500- E90531	Spindle x speed setting
E91100- E91131	Homing status on an axis
E92000- E92999	Axis origin switch enable state (analogue axes only)
E95000- E95031	Axis reference correction
E95300- E95399	Differential offset value

8.2.2 Type 2 external parameters

E10000 to E10031	Boolean I/O parameters for exchange with the PLC application
E11000	Angular offset (function ED..) enabled
E11001	Program origin offset (function G59) enabled
E11002	Not used
E11003	Processing of mirror function (function G51) enabled
E11004	Not used
E11005	Programming by diameter enabled (P4)
E11006	Tool centre programming (Turning Machine)
E11007	Spindle speed override enabled
E11008	Drawing of a Complete Circle
E11009	Complete revolution in G1 on rotary axes
E11010	Shortest direction selection on rotary axes
E11011	Indexed programming axes enabled
E11012	Zero following error operation (P19, E982xx, E983xx, E981xx)
E11013	Gradual acceleration
E11014	Deceleration function over several blocks
E11015	Angle crossing management enabled
E11016	Selects front or rear turret (Turning Machine)
E11017	Inclined plane function enabled (G24+)
E11018	RTCP function enabled (G26+)
E11019	Storing axes position on hardware interrupt
E11020	Allows writing the reference position for a phantom axis
E11021	Activation of the speed averaging filter outside HSC
E11022	G75 activated by CNC error 249(1) or by SamX (2) or by EGB(3)
E32014	Trajectory radius for G2,G3,G23,G01, spline and polynomial
E32016	Current line number of the block being executed
E32017	Current block number of the block being executed (N function)
E32018	NCK software version in hexadecimal
E32019	NCK FPGA version in hexadecimal
E35100 to E35131	First 32 bits test point value for drive 00 to 31
E35132 to E35163	Second 32 bits test point value for drive 00 (E35132) to 31 (E35163)
E41010	Interrupt channel triggered
E50001 to E50255	Tool length L of correction xxx or X dimension for Turning Machines
E51000	Current tool axis orientation
E51001 to E51255	Cutter tip radius @ of correction xxx or Z dimension for Turning Machines
E52001 to E52255	Tool radius R of correction xxx
E53001 to E53255	Dynamic length correction DL of correction xxx
E54001 to E54255	Dynamic radius correction DR of correction xxx

Type 2 external parameters	
E55001 to E55255	Tool tip orientation
E56001 to E56255	Available parameters (H of the table of dynamic corrections)
E57001 to E57255	Parameter defining the tool type of correction xxx:
E58001 to E58032	Loaded Tool from Tool Manager for each Spindle
E58000	Loaded Tool from PLC
E59000	Tool radius Overstock
E6x000	DAT1 on axis x
E6x001	DAT2 on axis x
E6x002	Minimum dynamic machine travel on the x axis
E6x003	Maximum dynamic machine travel on the x axis
E6x004	Eccentricity offsets on axis x (DAT3) (x=axis address from 0 to 5)
E6x005	Programmed offset by G59 on X axis
E6x006	WPC_DEC: Shift and Euler angles for WPC
E6x007	WPC_ANG: Angular offset applied on physical axes
E6x008	Same as 6x002 but do not stop execution
E6x009	Same as 6x003 but do not stop execution
E6x012	Angular offsets on RTCP rotary axes
E6xnn0	DAT1 ORIGIN for axis x, origin nn
E6xnn1	DAT2 ORIGIN for axis x, origin nn
E69000	Scaling factor
E69001	Inclined axis angle on grinders
E69002	Inclined wheel angle on grinders
E69003	Axis re-assignment
E69004	Tool disengagement in G41/G42 for an angle bigger that 120°
E69005	Tilting angle for jet cutting
E69006	Height of the part for jet cutting
E69007	Radius smoothing for jet cutting
E69008	Type of head for jet cutting
E7x000	Position reference of an axis in the channel
E7x001	Sampling the axes positions for on-the-fly probing
E7x002	Minimum static machine travel
E7x003	Maximum static machine travel
E7x004	Direction of movement of the axes being interpolated
E7x005	Axis address assignment
E7x006	Axis coupling
E7x007	Axes programmed by diameter
E7x008	Axes with a programmed movement outside E6x0008 - E6x0009 boundary
E7x009	Position axes got on-the-fly expressed in program frame (MCS or ICS)
E7x010	Sampling the axes positions on-the-fly probing WITHOUT reference correction
E7x100	Interpolator position references
E7x101	Interpolation speed limiting.
E7x200	EGB Synchro homing on first leader axis

Type 2 external parameters	
E7x201	EGB Synchro homing on second leader axis
E7x202	EGB Synchro homing on third leader axis
E7x203	EGB Synchro homing on fourth leader axis
E7x204	EGB Synchro homing on fifth leader axis
E7x205	EGB Synchro homing on follower axis
E7x206	Current EGB tolerance
E7x207	Current EGB Synchro error
E7x208	Maximum value reached in Ex7x207
E7x209	Minimum value reached in Ex7x207
E79000	Position actual value of the currently measured spindle
E79001	Speed set value for the currently controlled spindle
E79002	Value of the feed rate override assigned to the channel
E79003	Distance remaining to be travelled in the block currently executed
E79004	Current feed rate on the path programmed in the block
E79005	Minimum value for feed override
E79006	Maximum value for feed override
E79007	Minimum value for spindle override
E79008	Maximal value for spindle override
E79009	Distance to go within the block
E79010	Distance remaining in the current programmed block
E79011	Expansion factor
E79012	Current feed rate on the path programmed in the block NOT filtered
E79013	Feed rate override writeable with the Dyn Operator
E79014	(reserved) Feed maximum in feed mode
E79020 to E79022	Tool vector orientation in Machine frame in RTCP
E79023 to E79025	Tool vector orientation in Turntable frame in RTCP
E79026 to E79028	Tool vector orientation in Program frame in RTCP
E80000 to E80199	Local parameters
E81000 to E81999	General purpose or Master axis position reference
E82000 to E82999	General purpose or Slave axis corrections with respect to the master axes
E83000 to E83999	General purpose parameters
E84000 to E84999	General purpose parameters
E88000 to E89699	General purpose parameters reserved for NUM Macros
E89700 to E89999	General purpose parameters reserved for NUM Macros(NVRAM)
E90000 to E90031	Axis measurement
E90032 to E90063	Lag error

Type 2 external parameters	
E90064 to E90095	Motor Brake Test - check the axes position to start the test
E90400 to E90431	Spindle angular position
E90432 to E90463	Spindle x modulo
E90464 to E90495	C axis / Spindle measurement conversion used for Spindle/ axis commutation
E90500 to E90531	Spindle x speed setting
E90532 to E90563	Spindle position set value
E90564 to E90595	Spindle offset on speed set value
E90600 to E90631	Maximum spindle x indexing speed
E90632 to E90663	Spindle x in-position window
E90664 to E90695	Spindle x indexing gain
E90700 to E90731	Spindle x indexing acceleration
E90732 to E90763	Speed threshold below which spindle x is considered stopped
E90764 to E90795	Tolerance interval coefficient
E90800 to E90831	Enable synchronisation on master spindle
E90832 to E90863	Spindle rotation speed for maximum torque
E90864 to E90895	Spindle rotation speed for maximum power
E90900 to E90931	position actual value gap master slave for synchronised spindle
E90932 to E90963	Spindle integrative constant
E90964 to E90995	Spindle JOG speed
E91000	Servo or non-servo status of the axes
E91032	Servo or non-servo status of the spindle
E91100	Homing status on an axis
E91132	Homing status on a spindle
E91164 to E91195	Number of the channel where the spindle is located
E91200 to E91231	N/M AUTO axes
E91300	Clampable axis enable state
E91400	Homing forced by user on an axis
E91432	Homing forced by user on an spindle
E91500	E parameter copy of RCNC.BlockChanged[0..31]

Type 2 external parameters	
E91600	E parameter copy of WCNC.BlockChangeCtrl[0..31]
E92000	Axis origin switch enable state (Only for analogue axis)
E92100	Axis Error masking for delayed power down
E93700	Indication of axis in error or not
E93800	Flag handwheel in default
E95000 to E95031	Axis (or spindle) reference correction
E95100 to E95131	Position of the origin switch with respect to the machine origin
E95200 to E95231	Axis (or spindle) measurement correction
E95300 to E95331	Differential offset
E95400 to E95431	Measure correction for "volumetric error compensation" (Vecomp)
E95500 to E95531	Measure correction from PLC (ENA).
E95600 to E95631	Measure correction from interaxes calibration
E95700 to E95731	Measure correction from axes calibration
E96000	Duplicated axis in automatic modes
E96100	Duplicated axis in manual mode (JOG)
E96200	Synchronised axis
E96300	Symmetrically driven axis
E96400	Synchronised axis with offset = 0
E2000001 to E200xxyy	Tool length L of correction xxx or X dimension for Turning Machines
E2010001 to E201xxyy	Cutter tip radius @ of correction xxx or Z dimension for Turning Machines
E2020001 to E202xxyy	Tool radius R of correction xxx
E2030001 to E203xxyy	Dynamic length correction DL of correction xxx
E2040001 to E204xxyy	Dynamic radius correction DR of correction xxx
E2050001 to E205xxyy	Tool tip orientation
E2060001 to E206xxyy	Available parameters (H of the table of dynamic corrections)
E2070001 to E207xxyy	Parameter defining the tool type of correction xxx:
E2080001 to E208xxyy	Corrector Life Value
E2090001 to E209xxyy	Corrector Life Warning
E2100001 to E210xxyy	Corrector Life Error
E2110001 to E211xxyy	Corrector Life Type

Type 2 external parameters	
E2120001 to E212xxyy	Corrector Max Speed
E2130001 to E213xxyy	Corrector Max Feed
E3000000 to E3000031	M intersample time delay
E3000032 to E3000063	M timed delay



Contrary to part programming access, access to a specific parameter in the series E921xx, E924xx, E937xx or E938xx is performed by accessing respectively E92100, E92400, E93700 or E93800 and checking the respective bit. Access by dynamic operator of those particular parameters actually returns an array of 32 bits.

8.2.3 Type 3 external parameters

Writing of E3400x has an 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.LockAnalogueOutput := FALSE*
(Please refer to the Flexium* Commissioning Manual M00044).

Table of type 3 parameters for operator 6 (load)	
E10000 (-E10031)	Boolean Read / Write parameters for exchange with the PLC application
E20000 (-E20031)	Boolean Read parameters for exchange with the PLC application
E34010- E34011	UINT8 parameters-read the NCK embedded digital outputs value
E34110- E34111	UINT16 bits parameters-read the NCK embedded analogue outputs value
E43000- E43009	SINT8 Addressing PLC Inputs
E44000- E44001	UINT8 parameters-read the NCK embedded digital inputs
E44100- E44103	UINT16 parameters-read the NCK embedded analogue inputs
E46000- 46002	UINT16 Read parameters-access the PLC memory
E47000- E47127	UINT8 Read Read parameters-access the PLC memory

Table of type 3 parameters for operator 11 (store)	
E10000 (-E10031)	Boolean Read/ Write parameters for exchange with the PLC application
E33000- E33009	SINT8 Addressing of the PLC Outputs
E34000- E34001	UINT8 parameters-write the NCK embedded digital outputs
E34100- E34101	UINT16 bits parameters-write the NCK embedded analogue outputs
E36000- E36002	UINT16 parameters-write the PLC memory
E37000- E37127	UINT8 Read parameters-access the PLC memory

Please refer to Flexium* Programming manual M00040 for details about E parameters.



Loading a particular E100xx or E200xx parameters is performed by loading E10000 or E20000 and checking the corresponding bit.

8.2.4 Immediate values

The immediate operands are positive or negative integers.

In arithmetic operations, the immediate value is a number of shifts (>0 for left or <0 for 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 signalled and could lead to unpredictable results.

8.2.5 Operands pointing two consecutive memory locations

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

Example

E61000 points two memories E61000 and E61001.

8.2.6 L program variables

These operands are symbolized by the letter «L». They refer to local variables: L0 to L99.

8.2.7 Symbolic variables

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

8.2.8 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 M00040.

8.2.9 NCK Address

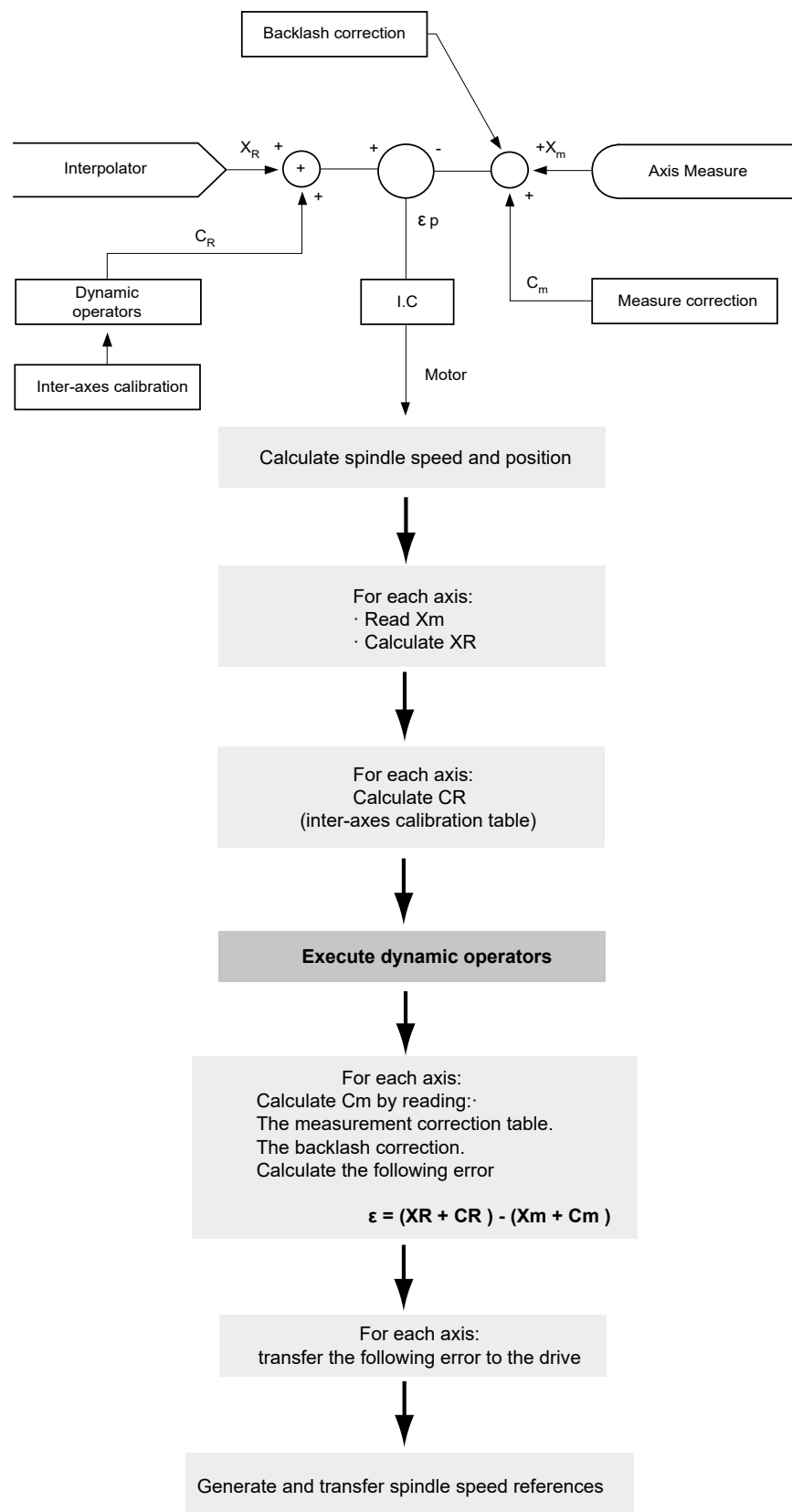
It is as well possible to read data at a particular NCK address. This must be exercised with caution as a wrong address can lead to an exception.

This possibility should be reserved for debug only and under control of NUM technical team:

\$	32 bits integer
#	64 bits real

8.3 Dynamic operators execution

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



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

Syntax

$$O_n = 1 \text{ Ea} / \text{Eb} / \text{Ec} \{I\}$$

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

Definition

Ea integer part of $(Eb + Ec) \{ \times 2^I \}$

Example

$$O_n = 1 \text{ E81000/E81001/E81002/-7}$$

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

For E parameters detailed information please refer to the Flexium* Programming manual M00040.

8.5 Code2 ► Subtract

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

Syntax

$O_n = 2 E_a / E_b / E_c \{I\}$	
Ea	Type 1 external parameter
Eb	Type 2 external parameter
Ec	Type 2 external parameter
I	Immediate value

Definition

Ea integer part of $(E_b - E_c) \{ \times 2^I \}$

Example

$O_n = 2 E_{81000} / E_{81001} / E_{81002} / 1$

Equivalent to: E81000 equals the integer part of $(E_{81001} - E_{81002}) \times 2$

For E parameters detailed information please refer to the Flexium+ Programming manual M00040.

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

Syntax
$$On = 3 \text{ Ea} / \text{Eb} / \text{Ec} \{ / I \}$$

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 $\text{Eb} \times \text{Ec} \{ \times 2^I \}$

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 unpredictable results. It is important to make sure that the different operands are correctly scaled.

Example

$On = 3 \text{ E81000} / \text{E81001} / \text{E81002} / -16$

Equivalent to: E81000 equals the integer part modulo 2^{32} of $(\text{E81001} \times \text{E81002}) / 65536$

For E parameters detailed information please refer to the Flexium⁺ Programming manual M00040.

8.7 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.

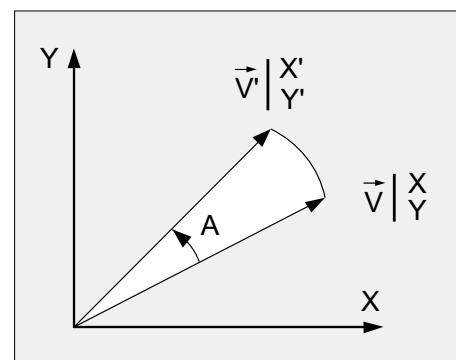
`On = 4 E81000/E81020/E80055/3600000`

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



8.8

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).

Eb can as well define an NCK address.

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	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	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	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	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 M00040.

8.9 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.

8.9.1 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 M00040.

Example with handwheels

01 = 6 E81000/W2

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

8.9.2 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 M00040.

Example

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

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

8.9.2.1 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.

8.9.2.2 Parameters E4300x

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

Example

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

8.9.2.3 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

8.9.2.4 Parameters E4410x

Parameters E4410x address the analogue input of the NCK.

E4410x: x = 0 to 3

E44100 = analogue in0 connector X9

E44101 = analogue in1 connector X9

E44102 = analogue in2 connector X9

E44103 = analogue in3 connector X9.

Example

On = 6 E81000/E44100

(E81000)=4096 = 10V

8.9.2.5 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 M00040.

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

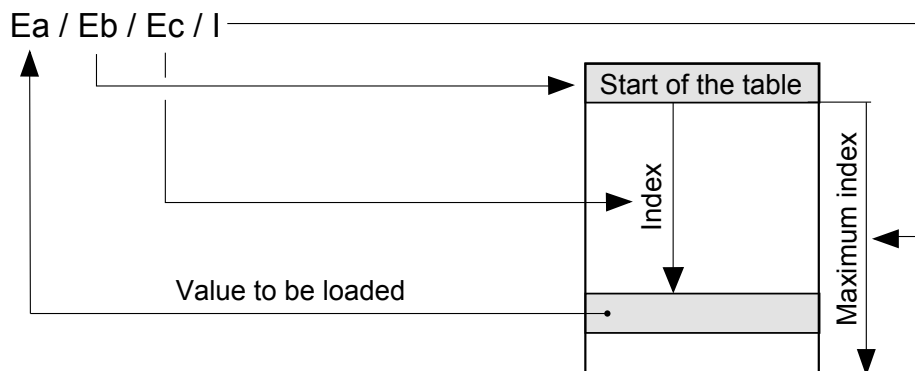
Description

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 M00040.

8.11 Code8 ► Indexed storage of a parameter

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

Syntax

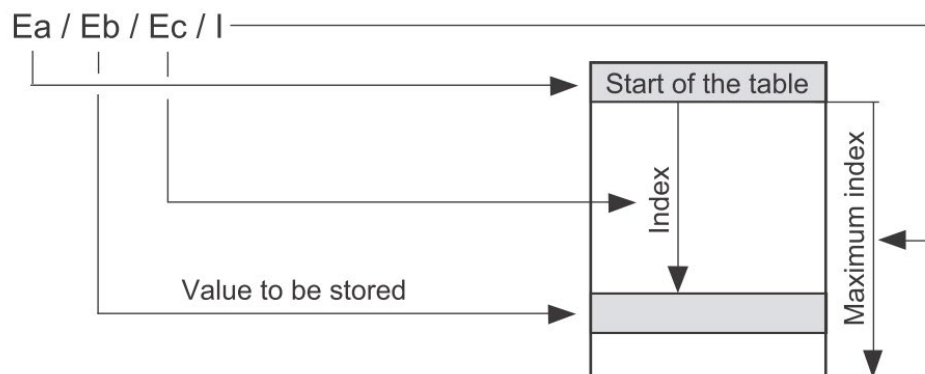
On = 8 Ea / Eb / Ec / I	
Ea	Type 1 external parameter
Eb	Type 2 external parameter
Ec	Type 2 external parameter
I	Immediate value

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).

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

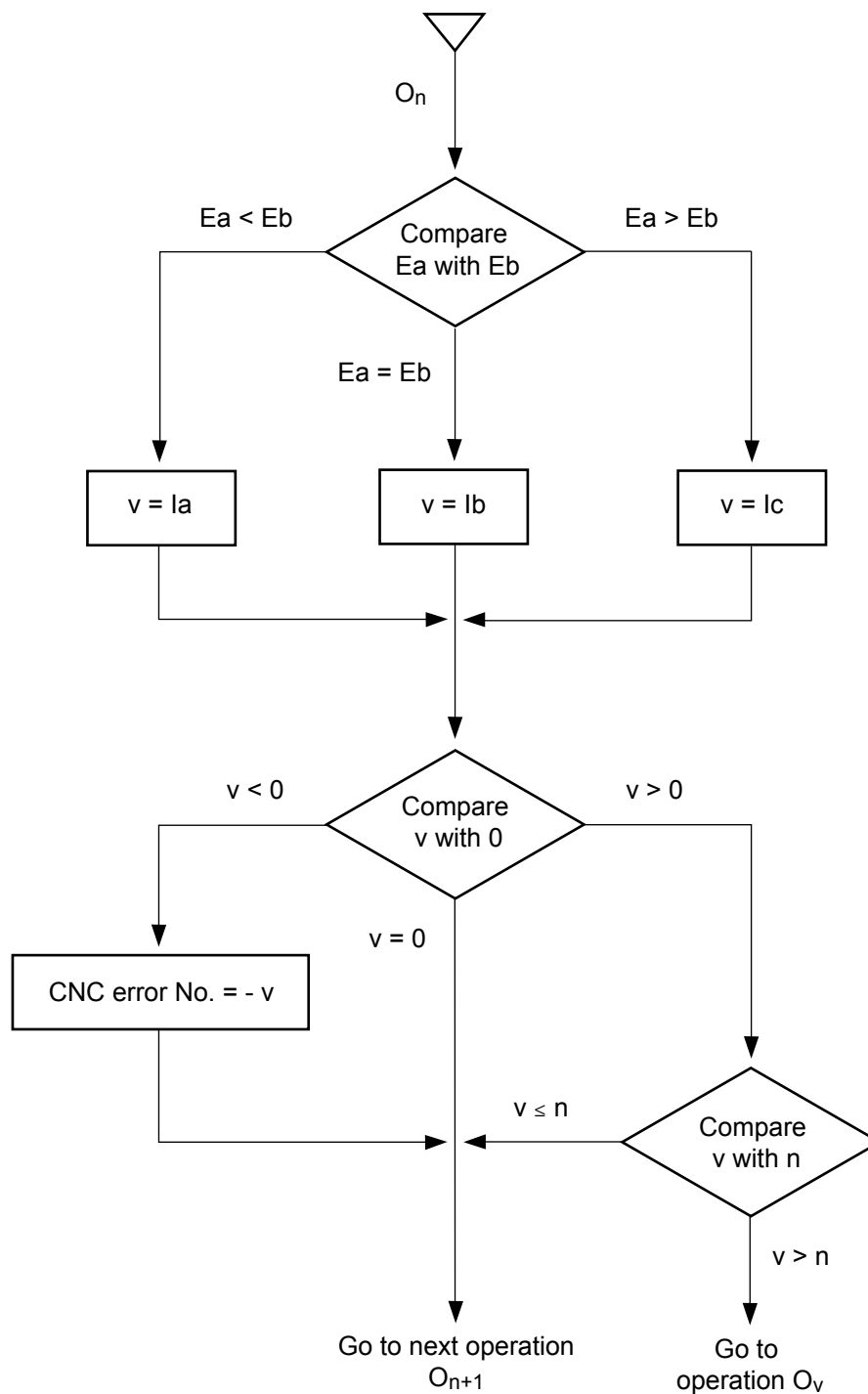
Example

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.

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

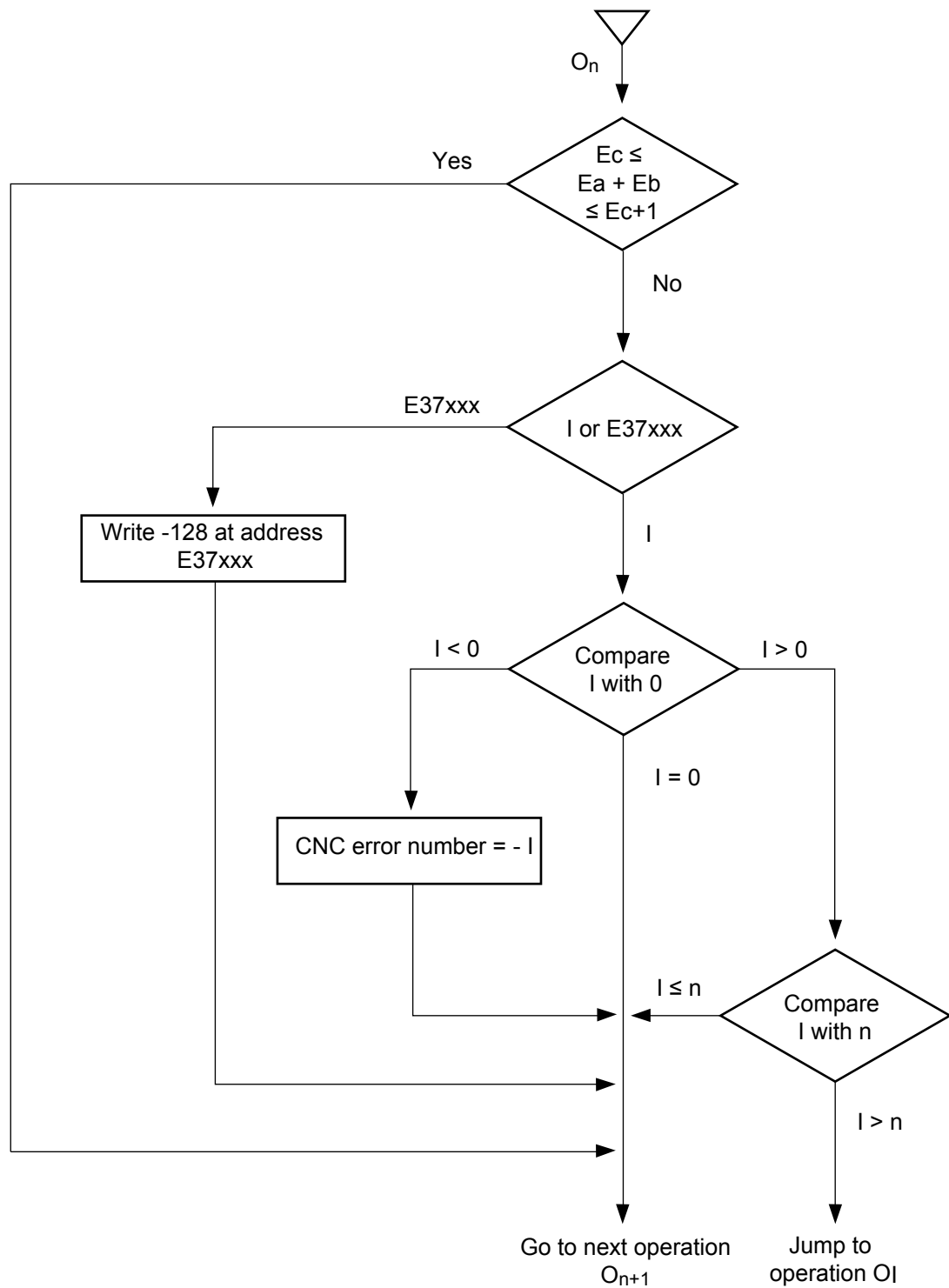
On = 10 E81000/E81001/E82000/E37xxx

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 M00040.

Flowchart



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.

8.14.1 **Storage at an axis address**

Syntax

On = 11 @.. / E..	
@..	Axis address (destination)
E..	Type 2 external parameter

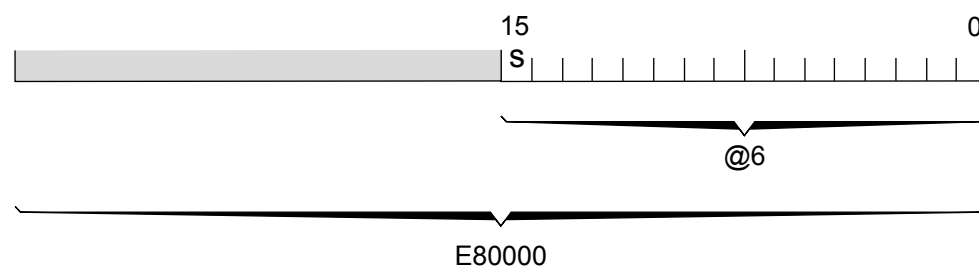
Description

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



8.14.2 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 M00040.

8.14.2.1 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).

8.14.2.2 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)

8.14.2.3 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.

8.14.2.4 Parameters E3410x

Parameters E3410x address the analogue output of the NCK.

E3410x: $x = 0$ or 1

E34100 = analogue out0 connector X9

E34101 = analogue 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 analogue outputs can be managed via Parameters E3410x if the PLC variable *Flexium_NCK.WCNC.IO.LockAnalogueOutput[x]* := FALSE.

8.14.2.5 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)

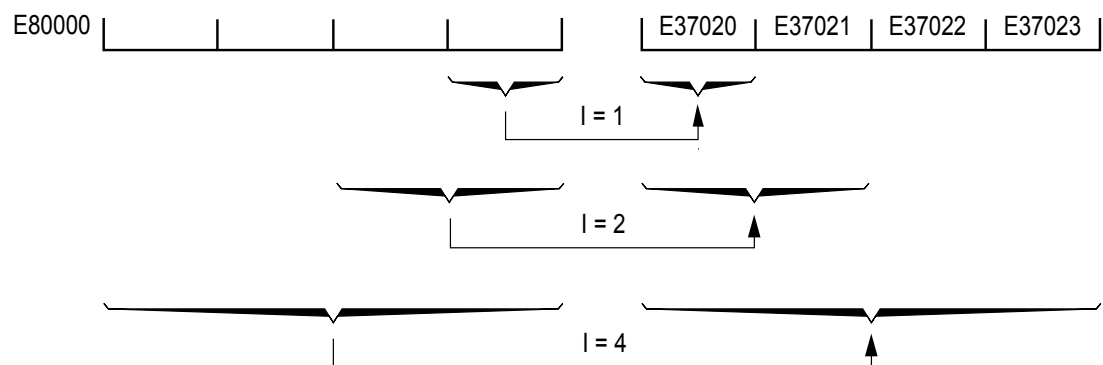
8.14.2.6 Parameters E37000 to E37127

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).



REMARK

E parameters permit to pass real number (floating point) to the PLC. This requires some programming on the PLC side as the format of data is different. Further explication can be found in the Flexium⁺ Commissioning manual M00044

A real number being coded on 8 bytes 8 parameters will be required. Please find an example below:

E83102 = -5022.99

Oxx = 11 E37120/E83102/8

The value will be written into parameters E37020 to E37027 (8 bytes) in floating point format.

To retrieve the values in the PLC:

```
pt_E37_real : POINTER TO LREAL := ADR(Flexium_NCK.RCNC.E37000[20]);
```

```
tmpReal : LREAL;
```

```
myReal : LREAL;
```

```
tmpReal := pt_E37_real ^;
```

```
myReal := SwapLReal(tmpReal );
```

The value will be retrieved in myReal

For E parameters detailed information please refer to the Flexium⁺ Programming manual M00040.

8.14.2.7 Parameters E3410x

Parameters E3410x address the analogue output of the NCK.

E3410x: x = 0 or 1

E34100 = analogue out0 connector X9

E34101 = analogue 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 analogue outputs can be managed via Parameters E3410x if the PLC variable *Flexium_NCK.WCNC.IO.LockAnalogueOutput[x]* := *FALSE*.

8.15 Code12 ► Increment with modulo

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

Syntax

$$On = 12 \text{ Ea / Eb / I}$$

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

Description

$$EaRTC_i = (EaRTC_{i-1} + Eb) \text{ modulo } I$$

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 M00040.

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 M00040.

8.17 Code14 ► Arc tangent function

This operator calculates an angle by the arc tangent function.

Syntax

On = 14 Ea / Eb / Ec / J	
Ea	Type 1 external parameter
Eb	Type 2 external parameter
Ec	Type 2 external parameter
J	Immediate value 0,1,2 (default 0)

Description

Ea returns the integer part of Arctangent of 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 side)
- 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 M00040.

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.



From version 4.2.0 a new parameter (J) has been added for handling the case where both Eb and Ec are near 0 according to the following table.

```
atan2(+0, +0) = +0;
atan2(+0, -0) = +π;
atan2(-0, +0) = -0;
atan2(-0, -0) = -π.
```

The possible values of J are in this case :

- J = 0 result always positive [0, 90°]
- J = 1 Computation is not done, Ea unchanged
- J = 2 System error message 94.66

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.

8.19 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

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

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

- 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 M00040.

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).

8.21 Code18 ► Divide

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

Syntax

$$On = 18 \text{ Ea} / \text{Eb} / \text{Ec} \{ / I \}$$

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

$On = 18 \text{ E81000} / \text{E80000} / \text{E81001} / 3$

$\text{E81000} \leftarrow \text{integer part modulo } 2^{32} \text{ of } (\text{E80000} / \text{E81001}) * 8$

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 $\sqrt{Eb \{x 2^I\}}$.

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 $\sqrt{E80000 * 8}$

8.23 Code20 ► Axis speed synchronisation

This operator synchronizes 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

- E20xx 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 desynchronized 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:

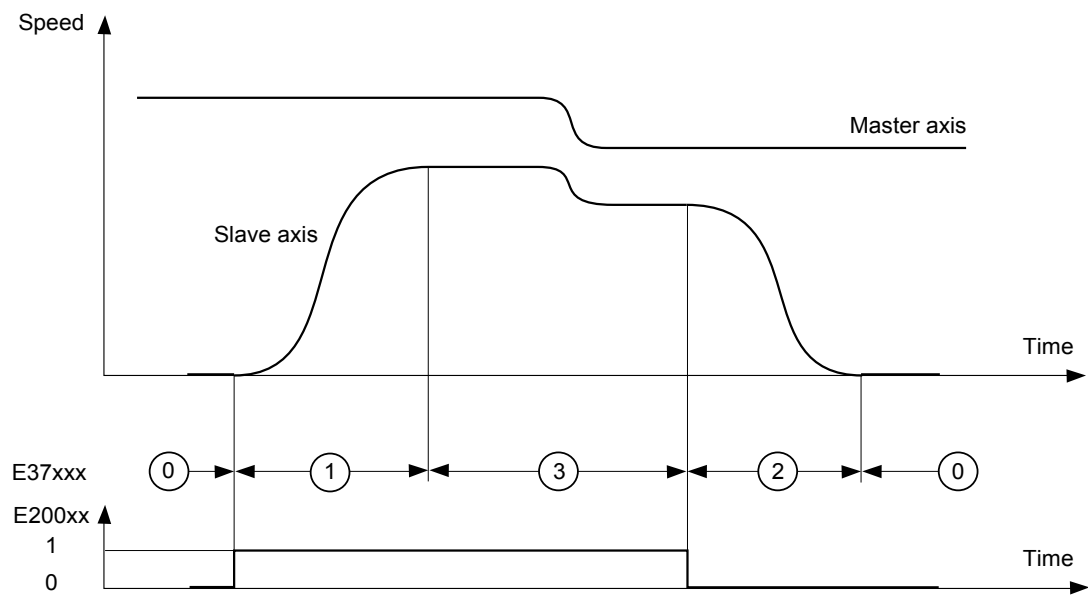
- 0: synchronisation inactive
- 1: slave axis accelerating before synchronisation
- 2: slave axis decelerating after de-synchronisation
- 3: axis synchronized (steady state).

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 M00040.

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 status.

The slave axis speed is 0.75 times the master axis speed (192/256).

8.24 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)] {/E47xxx}

@x, @y, @z	Physical addresses of the X, Y and Z axes
[TX(5,2)], [TY(5,2)], [TZ(5,2)]	Symbolic variables array
E47xxx	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	Rr

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 Rr have the same meanings but are used for the return phase.
- The sum of return movements must equal exactly the sum of forward movements:
 $Xa \cdot (Ra+1) + Xr \cdot Rr+1 = 0$.
- Tr0 is substituted for Ta0 at the beginning of the following cycles.
- The maximum timeout is 60000ms. (From Version 4.2.0.0 the maximum timeout time is 2^{32} ms).
- 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 non servo, the operator controls its reference (E7x000).

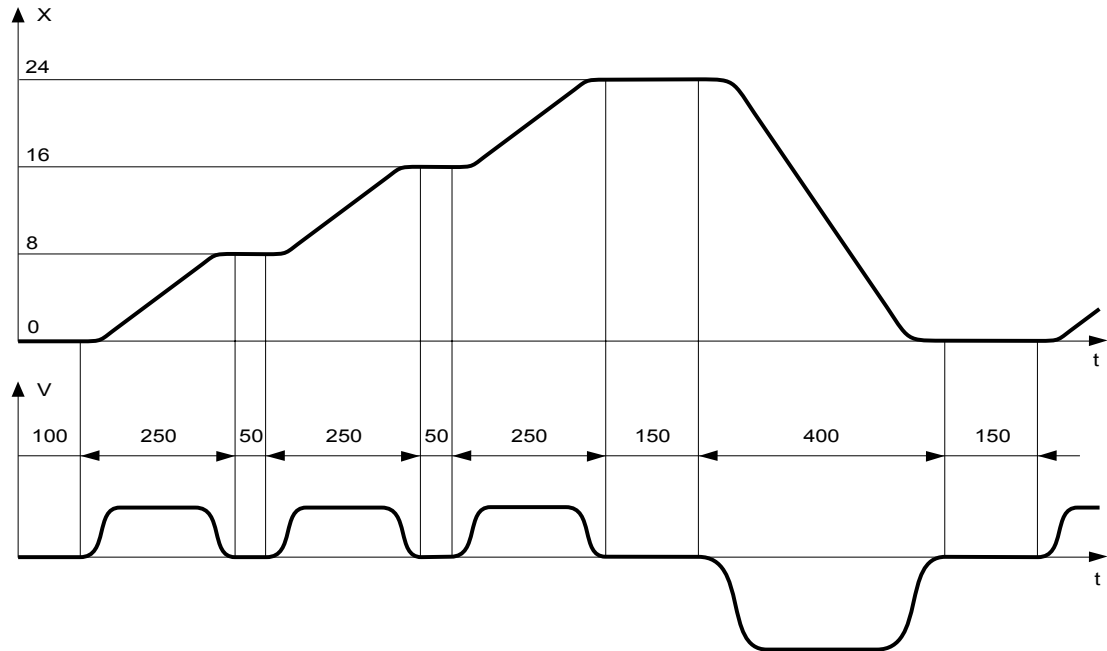
When operator 21 is cancelled by writing On = 0, each axis stops its cycle during the timeout and operator 21 is cancelled once all the axes have stopped.

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 @0 = [V1(5,2)]

[V1(5,2)] =	8	100	250	50	2
	-24	150	400	0	0

To load the table, the following block is to program:
[V1(5,2)]= 8, 100, 250, 50, 2, -24, 150, 400, 0, 0 .

8.24.1 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 E47000 to E47127 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 M00040.

8.25 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. • L100 to L199 parameter.



REMARK

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 Exxxx 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. If 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.

It is also possible to use an L parameter as input. However, this parameter must be in the range L100 to L199.

Example

Calling of a dynamic operator in C named REGUPOUR (source code given in example):

```
O1=22 REGUPOUR /E41005/E72000/E90002/E95002/E80002
```


8.25.1 Development tools

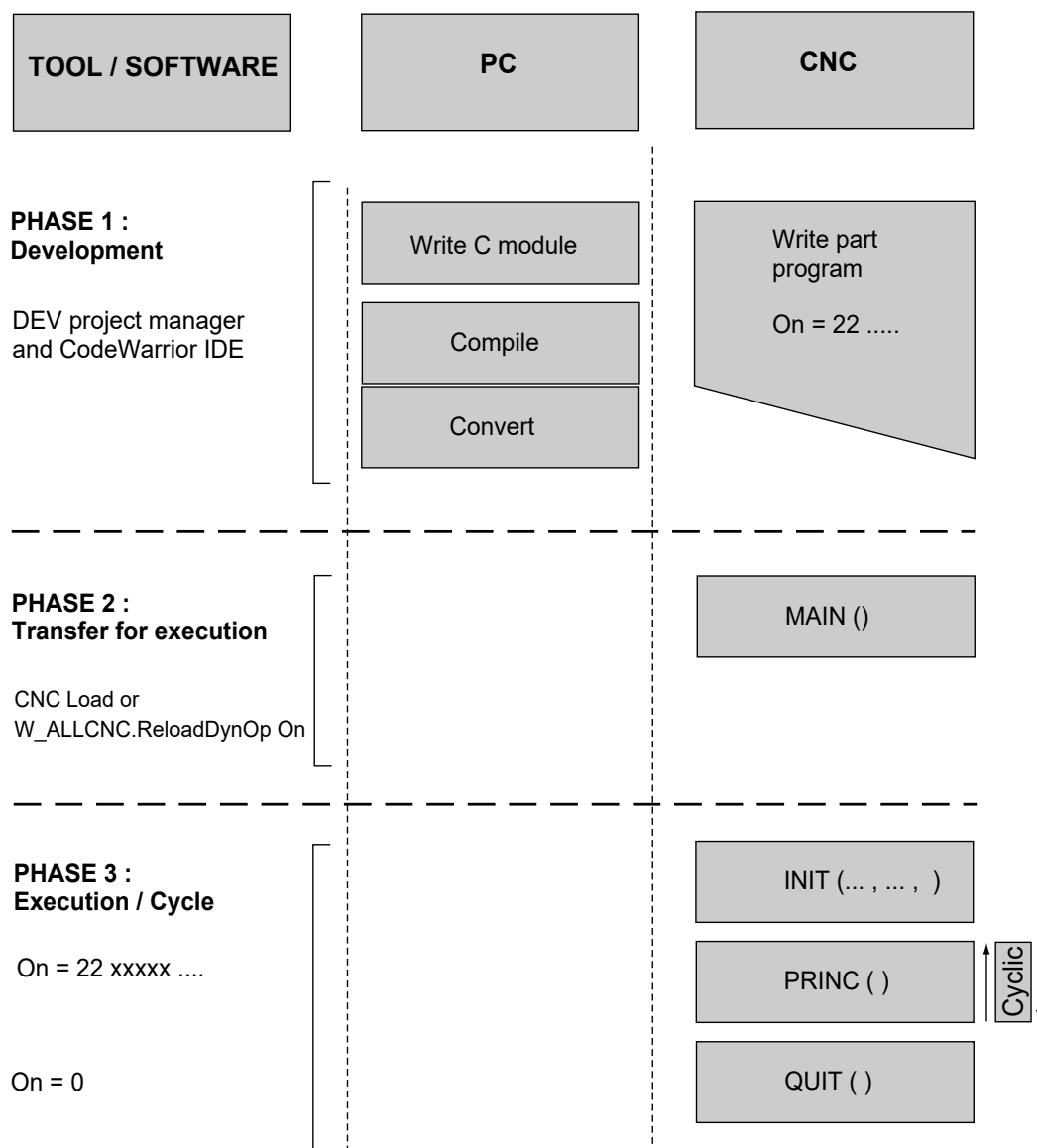
The development system is CodeWarrior Development Studio version 7.2 from Freescale.



This version only runs on 32 bits OS.

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.

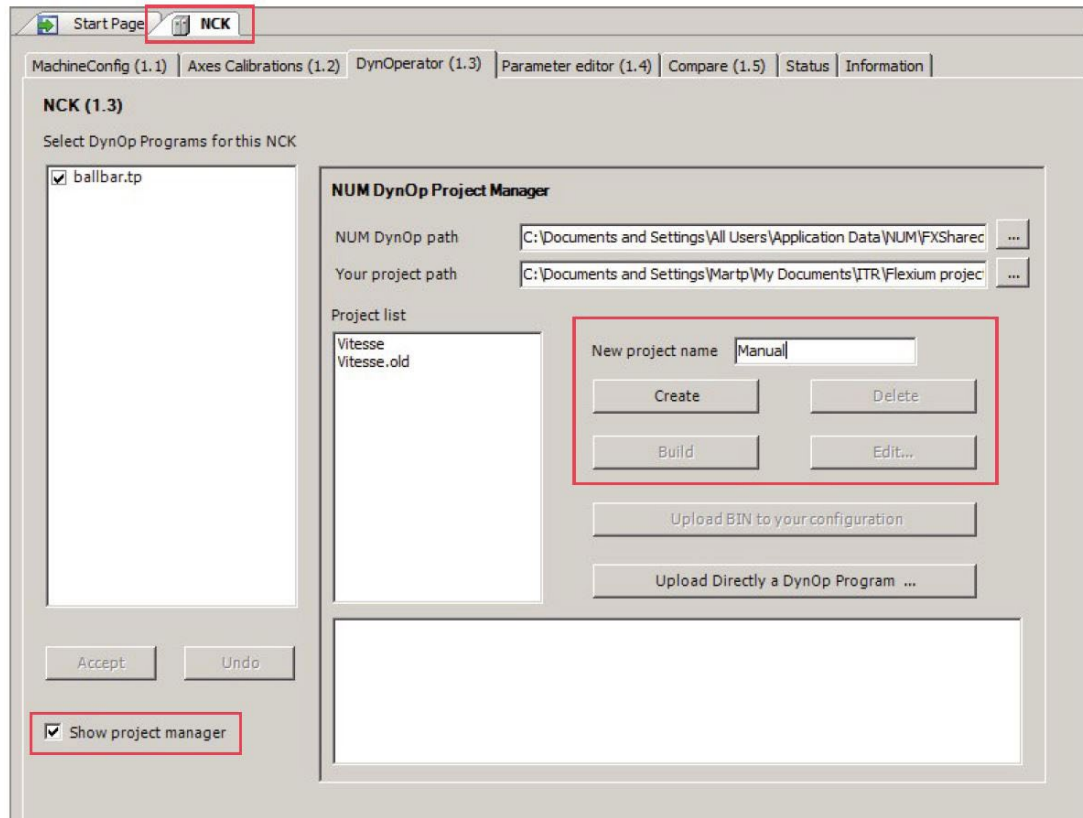
The “*NUM DynOp DEV Project Manager*” provides an interface between your project *CoDeSys* and the *CodeWarrior IDE* tools.



8.25.2 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.

8.25.3 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...)

8.25.4 Reading parameters

In Flexium all E parameters are 32 bits long, so it is possible to read and write them all using " long * " pointers.

In Flexium⁺ It is still possible to use pointers but if it is not the right type it becomes possible to read wrong values or even damage the NC memory by a write operation.

To avoid this problem two new functions were introduced to read and write parameters without using pointers

8.25.4.1 E parameters

Double ReadPar	(UINT32 adrPar, UINT32 typeParam, SINT32 index) returns the value of an E parameter.
void WritePar	(DOUBLE value, UINT32 adrPar, UINT32 typeParam, SINT32 index) writes a value in an E parameter.

A new element has been created : **type_param**

```
typedef struct s_PARA_EXT
{
    UINT32 parametre;
    SINT32 adr_param;
    UINT32 type_param;
} ARGUMENT;
```

For example E80000 being the input of a C dynamic operator So, when you have an input parameter

In init()

```
adrE80000 = pt[0].adr_param;
typE80000 = pt[0].type_param;
```

in Main()

```
valE80000 = ReadPar(adrE80000, typE80000, 0);
valE80020 = ReadPar(adrE80000, typE80000, 20);
```

To write

```
WritePar( valE80000 +1, adrE80000, typE80000, 0); E80000 = E80000 + 1;
WritePar( valE80020 +1, adrE80000, typE80000, 20); E80020 = E80020 + 1;
```

Please note that this function needs more computing time than just using pointer.

8.25.4.2 L parameters

The equivalent functions are available:

*BOOLEAN ReadLocalVar(DOUBLE *value, UINT32 index, UINT32 nGroup)* to return the value of a local variable L.

Input parameters	DOUBLE *value pointer to the returned value of the requested variable UINT32 index index of the variable (L0 - L99, L100 - L199, L900 - L959).
Output parameters	BOOLEAN result TRUE if the operation was successful FALSE if the index of variable or channel was wrong.

Note that L variables are always related to a channel, so it is mandatory to write the index of the channel (range 1 - 8).

It is possible to get the index of the channel where the operator is running as input parameters of the functions INIT(UINT16 nGroup,...) or PRINC(UINT16 nGroup,...).

With these functions it is also possible to read or write L variables related to different channels.

8.25.5 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()

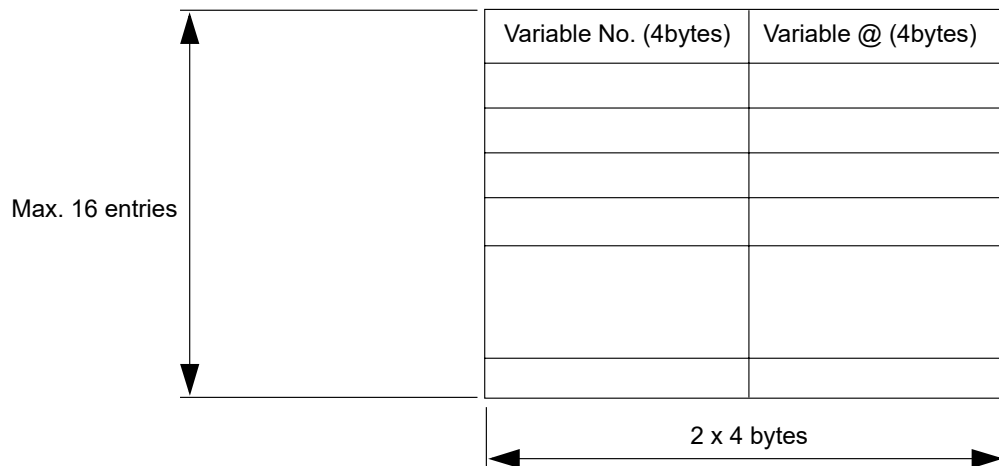
8.25.5.1 init (UINT16 gr_no, UINT16 no_arg, ARGUMENT *)

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 ($1 < \text{gr\_no} < 8$)
- The number of arguments programmed ($0 < \text{no\_arg} < 16$)
- 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 initializations 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.

This routine also permits to return the type of parameter as shown in the example on following page.

```

SINT32 init(UINT16 no_gr, UINT16 nb_arg, ARGUMENT *pt)
{
    UINT16 no_gr_local;
    no_gr_local = no_gr;

    adrE70000 = pt[0].adr_param;
    adrE90000 = pt[1].adr_param;
    adrE95000 = pt[2].adr_param;
    adrE80000 = pt[3].adr_param;
    typeE70000 = pt[0].type_param;
    typeE90000 = pt[1].type_param;
    typeE95000 = pt[2].type_param;
    typeE80000 = pt[3].type_param;

    OREFX = ReadPar(adrE70000,typeE70000,0);
    ER_MESX = MVITFIL = VITFIL = 0;
    return(ERR_AUCUNE);
}

/* End init() */

```

The available types are

```

typedef enum ee_tpPar
{
    tpUnknown = -1, // -1: unknown type
    tpByte = 0, // 0: signed byte - SINT8
    tpUByte = 1, // 1: unsigned byte - UINT8
    tpShort = 2, // 2: short integer - SINT16
    tpUShort = 3, // 3: unsigned short integer - UINT16
    tpLong = 4, // 4: long integer - SINT32
    tpULong = 5, // 5: unsigned long integer - UINT32
    tpDouble = 6, // 6: DOUBLE floating point
    tpByteMap = 7, // 7: byte bitmap - read as UINT8, shown as BOOLEAN
    tpWordMap = 8, // 8: word bitmap - read as UINT16, shown as BOOLEAN
    tpLongMap = 9, // 9: long bitmap - read as UINT32, shown as BOOLEAN
    tpLLong = 10, // 10: 64 bits integer long
    tpByteLongMap = 11 // 11: long bitmap - read as UINT32, show as UINT8
} tpPar;

```

8.25.5.2 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 optimized 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 opplib.a) and 2000 stack bytes for auto class variables.

8.25.5.3 **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 initialized pointer.

8.25.6 **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.

8.25.7 **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.

8.25.8 **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.

For details, please refer to G76+ H in this manual.

8.25.9 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 C dynamic operator
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

According to the value given in E80000, the following example will:

- Read 10 L parameters from the one given in argument and store them in E83000 to E83010.
- Store the values of L0-L99, L100-L199 and L900-L959 parameters into E83xxx.
- Read E83xxx parameters and store them in L0-L99, L100-L199 and L900-L959.

Sample Program in «C»

```
%400(TEST OPDYN READ WRITE L VAR )()

VAR [ix] [jx] [kx] [E83]
ENDV

N100
(Clear E parameter area)
FOR [ix]=0 TO 999
  DO [E83]=83000+[ix]   E[E83]=0
ENDF

E80000=0

G4F1
N200
(set values fro L0-L99 and for L100-L199)
FOR [ix]=0 TO 99 DO
  L[ix]=[ix]
  [jx]=100+[ix]
  L[jx]=[jx]
ENDF

(set values for L900-L959)
FOR [ix]=0 TO 59 DO
  [kx]=900+[ix]
  L[kx]=[kx]
ENDF

N300
E80000=3
O1 = 22 APPLVAR/E80000/E83000/L100
G4F1
E80000=0

M0 $ TEST R/W L VARIABLES BY SETTING E80000=1 OR 2 OR 3
M2
```

```

/*D*****
/*- Module name           : AppLvar                      -*/
/*-                               -*/
/*-                               on = 22 APPLVAR/E80000/E83000/L100 -*/
/*- Function:                               -*/
/*-                               -*/
/*-                               -*/
/*- Create date:           19/12/2016                      -*/
/*-----*/
/*- Evolution(s)           :                               -*/
/*-                               -*/
/*- Date :                               Author :           -*/
/*- Changes :           -                               */
/*-                               -*/
/*-                               .                               -*/
/*-----*/

#include "numopdyn.h"

/*-----*/
/* Prototypes of local functions DO NOT EDIT           */
/*-----*/
SINT32 main();
SINT32 init (UINT16 no_gr, UINT16 nb_arg, ARGUMENT *pt); /* Entry for the first
run */
SINT32 princ (); /* Entry for the cur-
rent run */
SINT32 quitt (); /* Entry for the last
run */

/*-----*/
/* Global variables APPLICATION SPECIFIC           */
/*-----*/

double *pE83xxx;
double *pE80xxx;
double *ptLvar ;
UINT32 nChannel=0 ;

/*-----*/
/* Invoked during installation in memory           */
/*-----*/
SINT32 main()
{
    SINT32 ret;
    ENTRY_PT pt_entree;

    pt_entree.INIT = init; /* Entry for the first run */
    pt_entree.MAIN = princ; /* Entry for the current run */
    pt_entree.EXIT = quitt; /* Entry for the last run */

    /* Application identifier */
    /*-----*/
    ret = EXPORT("APPLVAR","OPDYN2", &pt_entree);

    return (ret);
} /* end main() */

/*-----*/
/* Entry for the first run                               */
/*-----*/
SINT32 init(UINT16 no_gr, UINT16 nb_arg, ARGUMENT *pt)
{
    int i ;

    nChannel = (UINT32)no_gr ;

```

```

/* Checking the number of parameter passed */
if (nb_arg == 3) /* Number of parameters OK? */
{
    pE80xxx = (double *) pt[0].adr_param;
    pE83xxx = (double *) pt[1].adr_param;
    ptLvar = (double *) pt[2].adr_param;

    if(ptLvar == NULL) return(ERR_UsrInit) ;

    return(ERR_AUCUNE);
}
else /* Number of parameters not OK : Error 410*/
{
    return(ERR_PARAM);
}
}

/*-----*/
/*-----*/
SINT32 princ()
{
    int i;
    static double value ;
    BOOLEAN bool_answer ;

    if(pE80xxx[0]==1) // test of reading L serie value from L argument
    {
        for(i=0; i<10; i++)
        {
            pE83xxx[i]=ptLvar[i];
        }
    }
    else
    {
        if(pE80xxx[0]==2 && nChannel != 0) // use the primitive of reading of local variables
        {
            for(i=0; i<100; i++)
            {
                bool_answer = ReadLocalVar(&value,i,nChannel); // read L0-L99 values
                if(bool_answer == TRUE) pE83xxx[i] = value ;
                else return (430);
            }
            for(i=100; i<200; i++)
            {
                bool_answer = ReadLocalVar(&value,i,nChannel); // read L100-L199 values
                if(bool_answer == TRUE) pE83xxx[i] = value ;
                else return (431);
            }
            for(i=900; i<960; i++)
            {
                bool_answer = ReadLocalVar(&value,i,nChannel); // read L900-L959 values
                if(bool_answer == TRUE) pE83xxx[i] = value ;
                else return (432);
            }
        }
    }
    else
    {
        if(pE80xxx[0]==3 && nChannel != 0) // use the primitive of of local variables
        {
            for(i=0; i<100; i++)
            {
                value = pE83xxx[i] ;
                bool_answer = WriteLocalVar(value,i,nChannel);
                if(bool_answer != TRUE) return (433);
            }
        }
    }
}

```

```
    for(i=100; i<200; i++)
    {
        value = pE83xxx[i] ;
        bool_answer = WriteLocalVar(value,i,nChannel);
        if(bool_answer != TRUE) return (434);
    }

    for(i=900; i<960; i++)
    {
        value = pE83xxx[i] ;
        bool_answer = WriteLocalVar(value,i,nChannel);
        if(bool_answer != TRUE) return (435);
    }
}
}

return (ERR_AUCUNE);

}/* end princ() */

/*-----*/
/*-----*/
SINT32 quitt()
{
    return (ERR_AUCUNE);
}
```

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

A

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 M00040.

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

8.27 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 possible to move the axes with a hand-wheel if authorized 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 inter-axes 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 Synchronized Axes

In order to apply the datum shift to the slave, duplicated or synchronized 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 hand-wheel.

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.

8.28 Code25 ► Read current block data

The operator number 25 (Read Field BE) provides the value of a particular data field of the block in progress and its address (for later use by C dynamic operator).

Additionally (as for operator 5 for instance) it provides the capability to mask and shift the value read from this field.

Syntax

Oi=25 Ea/Eb/#Symbol value/#Index value/Ec/I	
Ea	External parameter type 1 to store the field value
Eb	External parameter type 1 to store the address of the field
Symbol	Symbol of the field to get access
Index	Index value when the field is a vector
Ec	External parameter type 2 to mask the value read
I	Shift of the value (masked or not)



NOTE

- Ea, Eb may be the same external parameter. In this case Ea and Eb both contain the address; the value of the symbol is irrelevant.
- Ec and I arguments are optional.

Currently available fields

Symbol	Programming syntax related	Meaning	Index
RN	N...	Block number	0
RL	D...	Tool corrector number	0
DIROUT	G16 P-/Q-/R-	Tool axis orientation negative	0
	G16 P+/Q+/R+	Tool axis orientation positive	1
ETAG0	G Codes	Status of G codes set 1	0
ETAG1	G Codes	Status of G codes set 2	0
ETAG2	G Codes	Status of G codes set 3	0

Example

```

\\ extraction of bit G71 status on bit 0
\\ G71 is in ETAG1 bit 0x3E-0x20=0x1E => 30 in decimal => mask = 2^30 and right shift = -30)
E83030=32768*32768 (2^15*2^15)
O2=25 E84002/E84003/#ETAG1/#0/E83030/-30

```

The table below describes the bits associated with the different G codes.
Rigid tapping in particular regroups different possibilities ,each associated with a different bit.

Additional information about those bits can be found in the ENA manual.

First set of G codes	
G00	(0X0)
G01	(0X1)
G02	(0X2)
G03	(0X3)
G04	(0X4)
G59	(0X5)
G38	(0X6)
G09	(0X7)
G17	(0X8)
G18	(0X9)
G19	(0XA)
G90	(0XB)
G91	(0XC)
G710	(0XD)
G52	(0XE)
G22	(0XF)
G40	(0X10)
G41	(0X11)
G42	(0X12)
G53	(0X13)
G54	(0X14)
G29	(0X15)
G43	(0X16)
G93	(0X17)
G94	(0X18)
G95	(0X19)
G164	(0X1A)
G96	(0X1B)
G97	(0X1C)
G30	(0X1D)
G20	(0X1E)
G21	(0X1F)

Second set of G codes	
G80	(0X0)
G61	(0X1)
G62	(0X2)
G777	(0x3)
G64	(0X4)
G65	(0X5)
G32	(0X6)
G07	(0X7)
G84	(0X8)
G84	(0x9)
G43-	(0xA)
G43+	(0xB)
G94 EF	(0xC)
G94 EF	(0xD)
G84	(0xE)
G84	(0xF)
G84	(0x10)
G73	(0X11)
G74	(0X12)
G999	(0X13)
G84	(0x14)
G84	(0x15)
G84	(0x16)
G76P	(0x17)
G10	(0X18)
G16	(0X19)
G51	(0X1A)
G34	(0X1B)
G12	(0X1C)
G70	(0X1D)
G71	(0X1E)
G34	(0X1F)

Third set of G codes	
G748	(0X0)
EG	(0X1)
G772	(0x2)
G46	(0X3)

8.29 Examples of dynamic operators programs

8.29.1 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 inter-axes calibration).

The part program is written for the U and Y axes declared in the following machine parameters:

- P0 (axes displayed),
- P9 (axis assignment to a channel),
- P3 (servo 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 M00040.

Dynamic Operator Used

- Operator 13: multiplication by a program variable.

Main Program

```
%1
R = 50
G77 H100
...
...
...
...
...
M02
```

Cylinder radius
Subroutine call

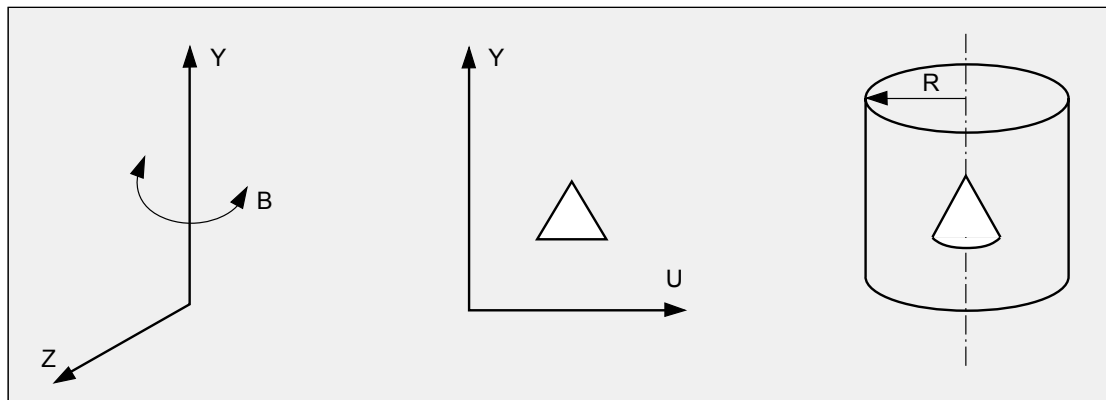
} Description of the profile with Y and U

} 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



8.29.2 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  C X
(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..

```

Shift 30 bits left because
the value is equal to 1 at most

Delta Z
Delta X

$$CZ = f \left(\Delta \frac{x}{Z} \right)$$

$$CX = f \left(\Delta \frac{\mathbf{x}}{Z} \right)$$

Program or subroutine

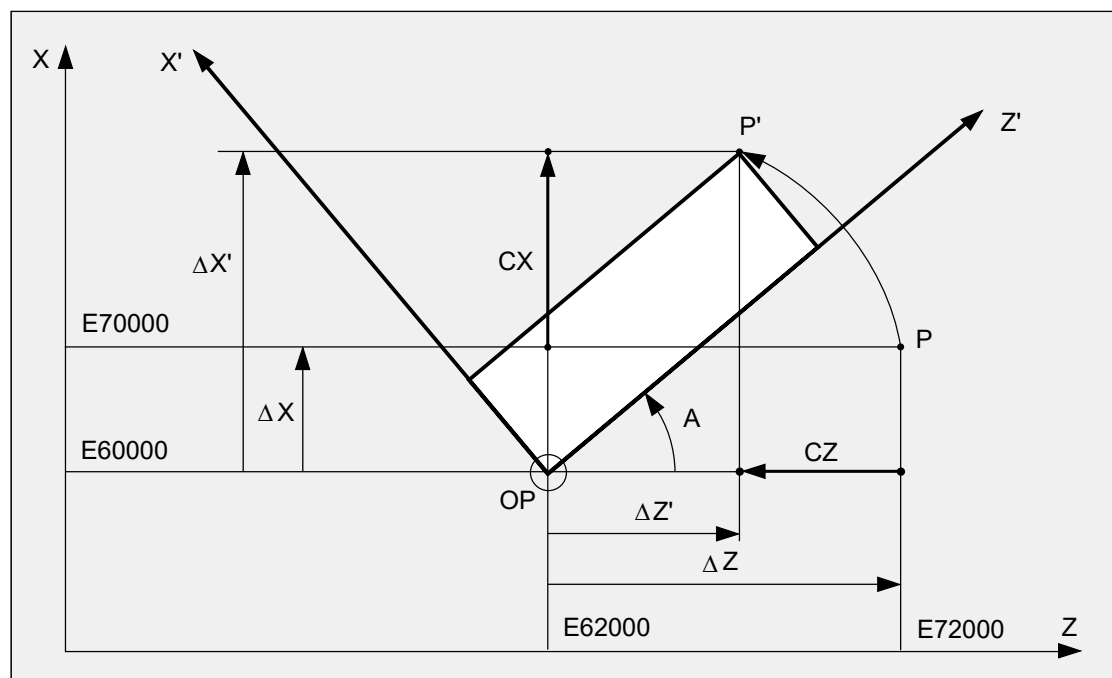
$$CZ = \Delta Z' (1 - \cos A) - \Delta X' \sin A$$

$$CX = \Delta X' (1 - \cos A) - \Delta Z \sin A$$

$$CZ = \Delta Z (\cos A - 1) + \Delta X \sin A$$

$$CX = \Delta X (\cos A - 1) - \Delta Z \sin A$$

Description of the Inclination on the X and Z Axes



8.29.3 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]

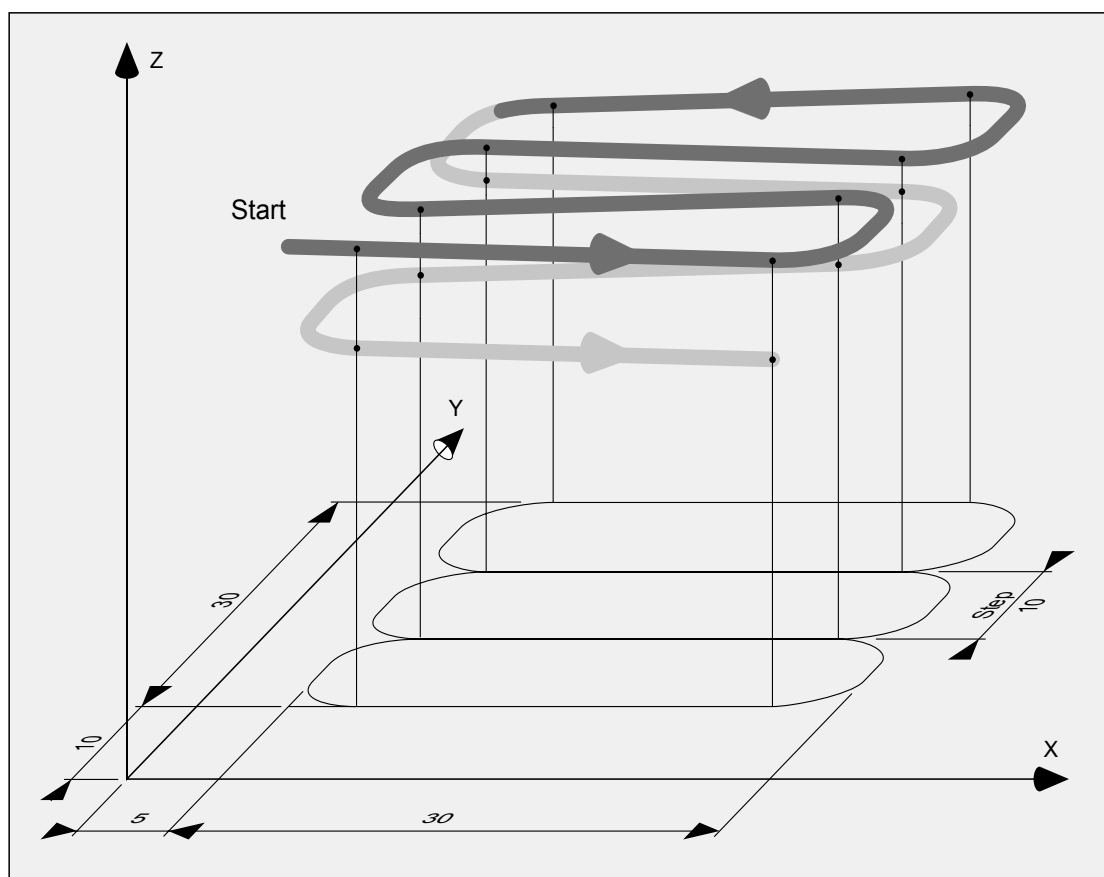
(Calculate the scan times on IY)
(The timeout separating two scans is 10 ms)
[V]=[vx(iy)]/[Fp]*60000 [ty(3,1)=T[V] [ty(3,2)=T[V] [ty(2,2)]=100
(and on IX)
[V]=[tx(1,1)]/[Fp]*60000/2 [V]=T[V] [tx(3,1)]=[V]*2 [tx(3,2)]=[V]*2
[tx(2,1)]=[ty(3,1)]+50-[V] [V]=[ty(3,2)]+100-[V]-[V]
[tx(4,1)]=[V] [tx(2,2)]=[V] [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

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

A

Page deliberately empty

Summary

9	Inclined Plane and RTCP	227
9.1	General	227
9.2	Inclined Plane machining	228
9.2.1	Overview	228
9.2.2	Conventional Inclined Plane definition	229
9.2.3	Programming Syntax	230
9.2.3.1	Specification with respect to MCS	230
9.2.3.2	Specification after positioning the tool in a known point of ICS	231
9.2.3.3	Enabling an ICS from the current tool orientation	233
9.2.3.4	Inclined plane suspension	236
9.2.3.5	Re-Enabling inclined plane	237
9.2.4	Notes about the inclined plane function	238
9.3	Complements about inclined plane G150	239
9.3.1	Background on the coordinate transformation	239
9.3.2	Case studies for an iPLANE implementation	239
9.3.3	Case of a tilting turntable	240
9.3.3.1	Machining the first oblong P1 pocket	240
9.3.3.2	Summary of the implementation procedure	242
9.3.3.3	Machining the second oblong pocket	243
9.3.4	Case of a C'-axis turntable combined with an A-axis Head	246
9.3.4.1	Machining the oblong P1 pocket	246
9.4	RTCP (Rotation around Tool Center Point)	248
9.4.1	General	248
9.4.2	Convention of Tool Center Point definition	249
9.4.3	Datums DAT1 and DAT2	249
9.4.3.1	Both DAT1 and DAT2 offsets applied before ICS	249
9.4.3.2	Only DAT1 offset applied before ICS	250
9.4.3.3	Only DAT1 offsets applied before the ICS	250
9.4.4	Programming Syntax	251
9.4.4.1	RTCP enabling with inclined coordinate system specification (ICS)	251
9.4.4.2	RTCP enabling with specification of an ICS after positioning	253
9.4.4.3	Enabling a tool ICS from the current tool orientation	254
9.4.4.4	Suspension of the RTCP function and its associated ICS	257
9.4.4.5	Re-Enabling the RTCP function without its associated ICS	257
9.4.4.6	Re-Enabling the RTCP function with its associated ICS	258
9.4.5	Additional functions	259
9.4.5.1	Tool length correction	259
9.4.5.2	Tool wear offset	259
9.4.5.3	Tridimensional tool correction (G29)	259
9.4.5.4	Table off-centering (DAT3)	259
9.4.5.5	Program origin offset (G59)	260
9.4.6	Notes about the RTCP function and its associated inclined plane	262

Page deliberately empty

9 Inclined Plane and RTCP

9.1 General

NUM provides the functions to handle inclined plane programming as well as RTCP. Details of those functions are explained later in this chapter.

The machine builder will adapt the cycles provided to a particular machine configuration; therefore programming your machine may slightly differ from the syntax given below in particular by the cycle number.

This manual refers to the function G150 for inclined plane and G151 for RTCP which is the NUM standard but the machine builder may have selected other G codes in particular for multi channels machine but in any case, the other arguments of the syntax remain unchanged.

List of abbreviations used in this chapter

MCS	Machine Coordinate System
ICS	Inclined Coordinate System
WCS	Workpiece Coordinate System
TCP	Tool Center Point
RTCP	Rotation Tool Center Point

E Parameters in use

For internal operations and data backup, inclined plane and RTCP are using the following parameters as standard. However, it is possible that the machine builder has selected other parameters.

Refer to the machine operation manual.

The inclined plane function use the 4 following parameters as standard:

E80196	Used for saving the kinematic configuration Tn.
E80197	Used to define the coordinate system of the inclined plane function.
E80198	Containing the position measurement of the rotary axis and according to the type of the R rotary axis measurement.
E80199	Containing the position measurement of the swivel axis and according to the type of the S swivel axis measurement.

The RTCP function uses a series of 25 E Parameters for saving the current context of RTCP internal calculated variables:

E80170 to E80194	Reserved by the RTCP function in case of a usual customization.
E80194	Reserved by the RTCP function for saving the kinematic configuration Tn



Note to the OEM

The PLC flags *axis_moving* reflect the state of the programmed axes but not necessarily of the axes that might be moved by RTCP or Inclined plane.

It is important to take this in account in particular in the case of clampable axes.

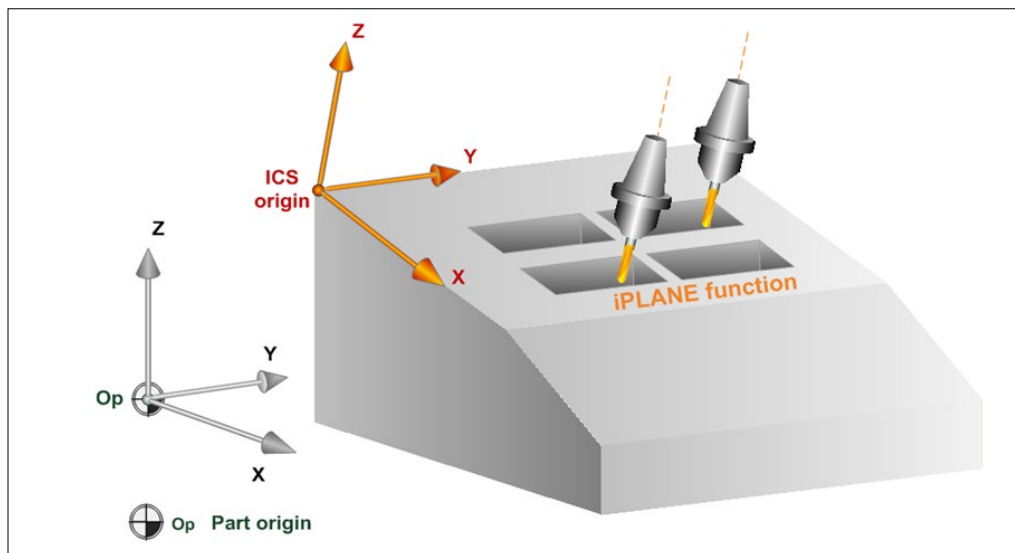
9.2 Inclined Plane machining

9.2.1 Overview

The Inclined Plane machining function is designed for allowing machining in a coordinate system that is not aligned with the physical axes of the machine while keeping programming simple.

The six following functionalities are supported:

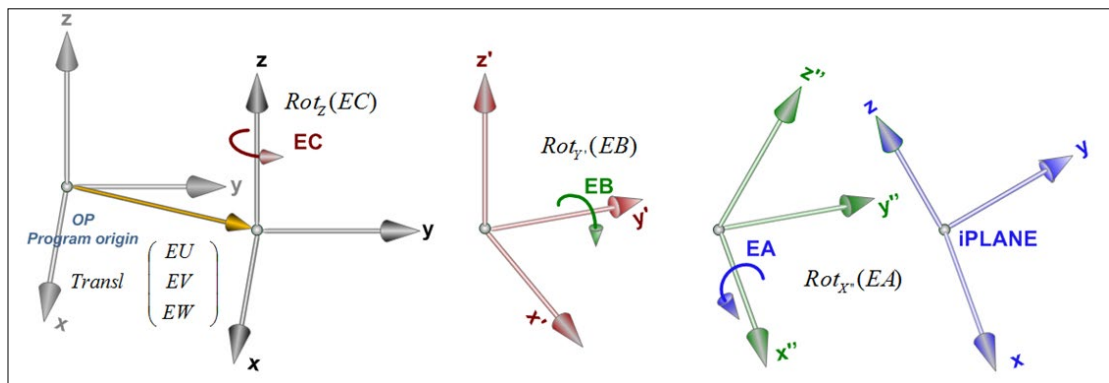
- Selection of one out of a maximum of eight kinematic configurations.
- Selection of the primary axes (X,Y or Z) or secondary axis (U,V or W) used in the inclined plane.
- Enabling the function after positioning the tool in a known position with respect to the new referential origin.
- Enabling by programming its origin.
- Temporarily suspension (for instance for a tool change).
- Reactivation of the last programmed inclined plane.



9.2.2 Conventional Inclined Plane definition

The specification of an inclined coordinate system in space is described by the following four successive transformations:

- **1st step:** Translation of the inclined plane origin of (EU , EV , and EW), relative to the reference coordinate system.
 EU , EV and EW are respectively the 3 translation components along X , Y and Z axis.
- **2nd step:** Rotation of the previous shifted referential, obtained in step 1, by an angle EC around axis Z .
- **3rd step:** Rotation of the *current referential*, obtained in step 2, by the angle EB around axis y' .
- **4th step:** Rotation of the *current referential*, obtained in step 3, by the angle EA around axis x'' .



It is important to notice that the inclined plane function will only operates correctly when complying with this definition of the successive transformations (translation of origin, rotation around Z axis, then around Y axis, then around X axis).

9.2.3 Programming Syntax

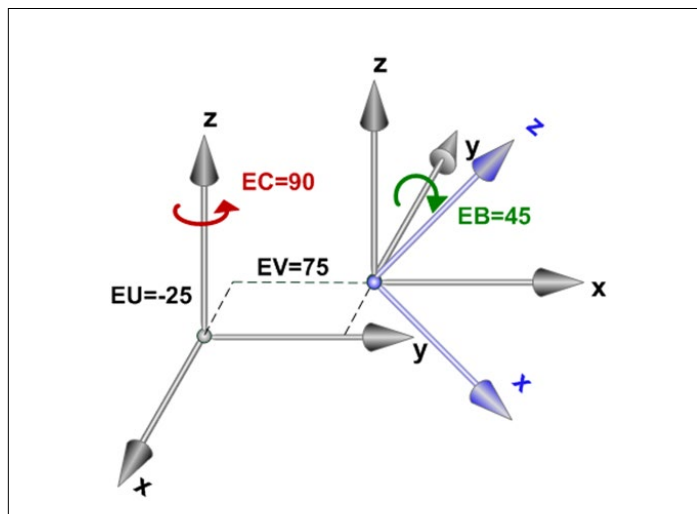
9.2.3.1 Specification with respect to MCS

G150 EA...EB...EC... EU...EV...EW...[T]	
G150	G function of inclined plane.
EA EB EC	Specify the three orientation angles of the new coordinate system. The presence of one of these three angles is mandatory. If an angle is not specified, a zero value is assigned to it. The angles values are given in degrees.
EU EV EW	Position of the inclined coordinate system origin with respect to the part program origin. EU, EV and EW are respectively the offsets of origin along the X, Y and Z and are expressed in mm. At least one of these three arguments must be specified.
T (optional)	Code number of the kinematic configuration among a max of eight configurations. Different configurations may be required for machine with removable milling heads. The correspondence between the head and the T code is defined by the machine builder. If the T argument is omitted, the function chooses the previously programmed configuration (the first by default).

Example: G150 EC90 EB45 EU-25 EV75 T2

This statement enables an inclined plane, set with the second kinematic configuration (T2) and defines an inclined coordinate system whose origin is located at -25 mm along X and at 75 mm along Y from the initial workpiece origin.

The new referential (ICS) orientation is given by two successive rotations: the first rotation of 90 degrees around the Z axis followed by a rotation of 45 degrees around Y' axis.



9.2.3.2 Specification after positioning the tool in a known point of ICS

G150 X...Y...Z...EA...EB...EC...D...[T]	
G150	G function of inclined plane.
X Y Z	Specify the position of the tool tip position with respect to the inclined coordinate system (ICS) If one of coordinates is not programmed, a zero value is assigned to it. The OEM has defined the default coordinate system that will be used (XYZ, UYZ, XVZ,...). The axes can be changed by programming. Examples: if the XVZ Cartesian coordinate system is chosen as default during the inclined plane integration. G150 EA90 EU10 Defines an inclined plane on the default coordinate system, i.e. on X, V and Z. G150 U..Y..EA90 EU10 Defines the same inclined plane on axes U, Y and the third axis of the default coordinates, i.e. Z.
EA EB EC	Specify the three orientation angles of the new coordinate system. The presence of one of these three angles is mandatory. If an angle is not specified, a zero value is assigned to it. The angles values are given in degrees.
D	Tool offset assigned to the current tool. This argument must be specified.
T (optional)	Code number of the kinematic configuration among a max of eight configurations. If T argument is omitted, the function chooses the previously programmed configuration (the first one by default).

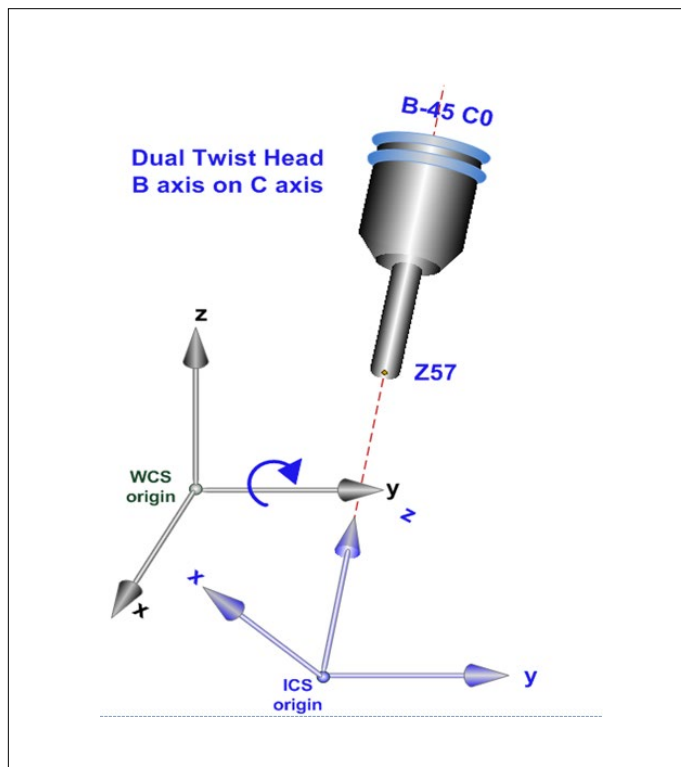
Example G150 Z57 EB-45 D17 T2

Consider a dual twist of which the swivelling axis is positioned at -45 degrees as shown in opposite figure.

This statement specifies an inclined coordinate system in which the tool tip is located at 0mm in X, 0mm in Y and 57mm in Z with respect to the ICS.

The origin position of inclined plane is computed knowing the current tool and its tool offset given by D17. The new ICS orientation is given by a rotation of -45 degrees around the Y axis.

The kinematic setup considered by the inclined plane function is programmed by T2.



9.2.3.3 Enabling an ICS from the current tool orientation

G150 R1 D...	
G150	G function of inclined plane.
R1	From the current tool orientation, this function computes an inclined coordinate system (ICS) whose origin coincides with the tool tip, Z axis is directed along the tool axis direction. This function allows programming an axial retraction or a lateral motion of the tool according to the referential TOOL ICS, in JOG mode or in MDI programming. Its use is restricted to 5-axis kinematics having a single-twist head or a dual-twist head. Otherwise, the following error message is set (Error 515 extension 2): <i>"Tool referential computation doesn't support this 5 axis kinematics."</i>
D...	Tool offset assigned to the current tool. This argument must be specified.

Example G150 R1 D4

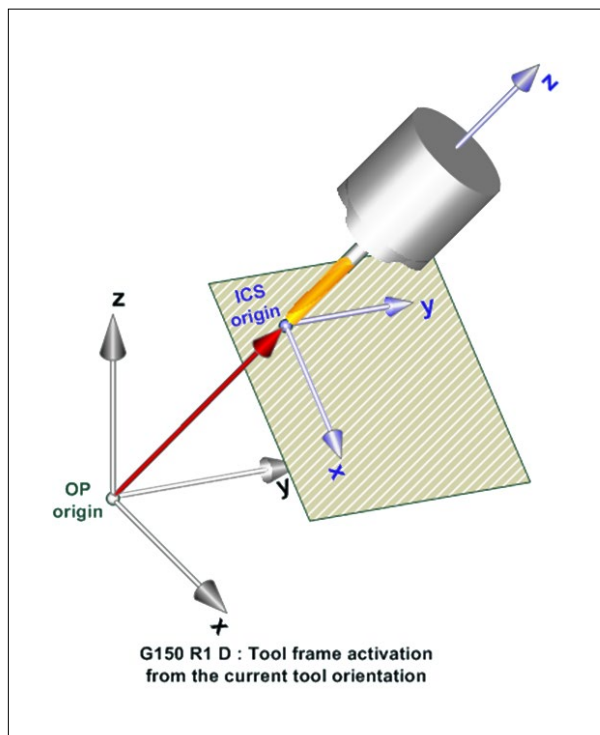
From the current orientation of the tool, the 5-axis function **G150 R1 D...**, calculates a set of three Euler angles defining the referential TOOL ICS.

EA angle around the X axis,
EB angle around the Y axis,
EC angle around the Z axis.

Then activates the ICS whose origin coincides with the tool tip.

Note.

The activation of the tool referential by **G150 R1 D** saves the current context. Therefore, a reactivation of the previous inclined plane function by **G150 S1** restores the retraction frame.



Programming example

```
G90 G1 F G16R+
D..
X... Y... Z...
```

```
A30 C0
G150 EA30 EU10.5 T1
```

```
A... C... Z...X... Y...
A... C... Z...X... Y...
```

Machining with respect to the ICS.

```
.....
G150 R1 D...
Z10 F200
```

Computes referential TOOL ICS and activates the RTCP.

Retraction along the Z axis direction of the referential TOOL ICS.

```
.....
```

G91 programming is optional because the motion is expressed in ICS.

```
.....
G150 S0
X... Y... Z...
A... C...
```

Reset inclined plane.

```
G150 R1 D...
X10 F200
```

Changes the tool orientation.

Computes and enables a new referential TOOL ICS.

Lateral motion.

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

A

9.2.3.4 Inclined plane suspension

G150 S0	
G150	G function of inclined plane.
S0	This function cancels the inclined plane function. Subsequently the programmed movements of axes in the machine coordinate system (MCS). The inclined parameters are saved for further reactivation.

9.2.3.5 Re-Enabling inclined plane

G150 S1	
G150	G function of inclined plane.
S0	This function reactivates the last active inclined plane (previously specified by following syntaxes G150 EA EB EC EU EV EW T or G150 EA EB EC D T) and suspended by G150 S0, or a reset.

9.2.4 Notes about the inclined plane function

- When inclined plane is enabled, the Z axis of the new ICS has to be parallel to the tool axis direction.
- Change of tool orientation with inclined plane enabled is not permitted and could cause machining geometry errors or machine crash.
- An active inclined plane remains enabled even after a reset or a program end M02. It can only be cancelled by programming G150 S0 or reinitializing the CNC.
- When power is restored after a power failure, it is possible to activate an inclined plane or to re-enable the last activated before homing is completed on the axes (for instance, to retract a tool in the same orientation after a power failure).
- When inclined plane is enabled, homing is prohibited.
- Any change of the DAT1 or DAT2 datum shifts, either directly or by programming of the corresponding external parameters (E60xxx and E61xxx), is prohibited when the inclined plane is enabled.
- Programming of movements in G52, i.e. in absolute coordinate referenced to the measurement origin, is prohibited when inclined plane is enabled, except if authorized by the machine builder (parameter P7N1.6)
- Programming of a movement with DAT1 or DAT2 inhibited (G53) is prohibited when inclined plane is enabled.

9.3 Complements about inclined plane G150

9.3.1 Background on the coordinate transformation

The iPLANE function (G150) uses the coordinate transformation, G24+ function, applied downstream of the CN interpolator.

The coordinate transformation is characterized by a square matrix (3x3), expressing the orientation of the three vectors of the inclined frame with respect to the program frame OP (declared by the sum of datum shifts DAT1 + DAT2), and by a 1x3 vector of origin shifts of ICS frame relatively to the OP frame.

9.3.2 Case studies for an iPLANE implementation

In these examples, we will study how to correctly program the iPLANE function for two types of 5-axis kinematics.

The first kinematics is a dual rotary table, the primary rotary table (cradle axis) carries the secondary rotary table, which in turn carries the fixture and the part. The second is a rotary table which carries the work piece, combined with a spindle head that articulates the tool with tilting motions

Before calling G150, the rotary axes are positioned so that the tool is normal to the face to be machined.

In the particular case of kinematics having a rotary table or a tilting turntable, the three linear offsets of the inclined frame origin O_{Pi} are only defined when the 5-axis kinematics is in the reference pose.

The function G150 recalculates the offsets taking into account the current position of the rotating axes.



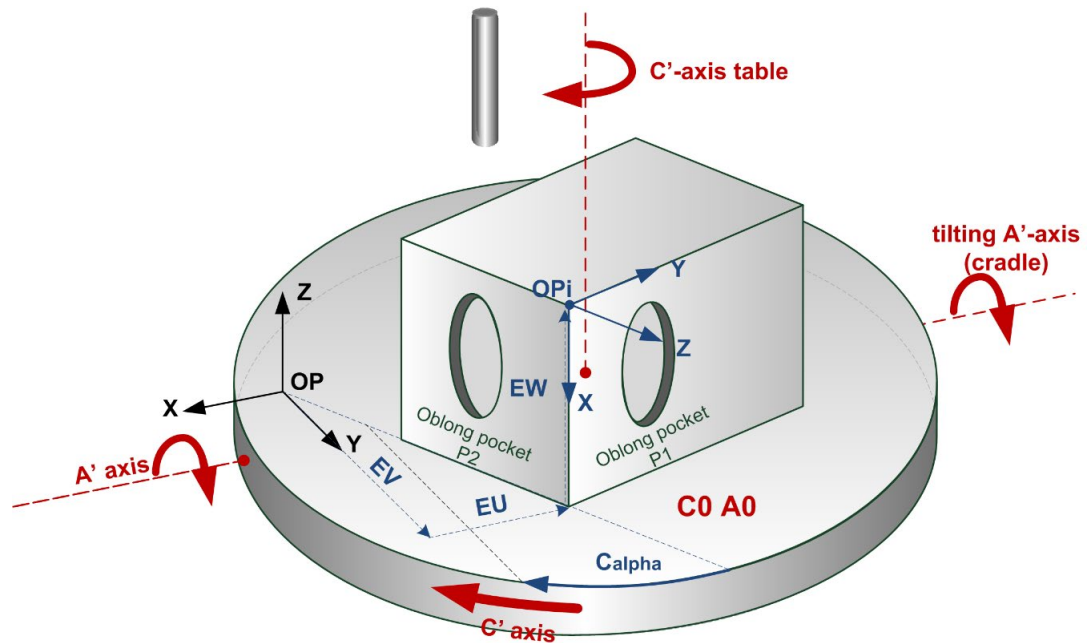
Unlike the RTCP function, the iPLANE function does not move the OP program frame when rotating or tilting a turntable.

9.3.3 Case of a tilting turntable

Considering a rotary table C'-axis carried on a tilting table A'-axis

9.3.3.1 Machining the first oblong P1 pocket

We want to machine a first oblong pocket, named P1, on one side of a blank rectangular parallelepiped fixed on the table. This part is positioned with a slight misalignment (orientation) with respect to the main axes of the OP reference frame when the 5-axis kinematics is in reference position A0C0.



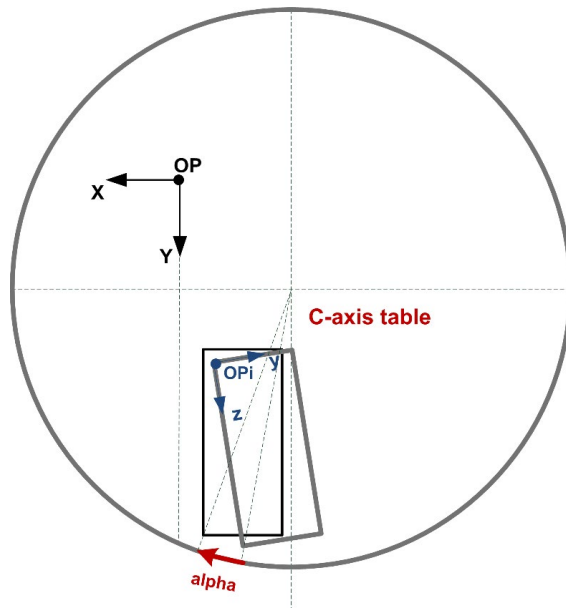
Tilting turntable (C-axis on A-axis) in the reference position A0C0
Preparing for the machining of the oblong pocket P1

The first step consists in placing the tilted reference frame on the face of the part, thus defining the interpolation plane, that is to say placing the origin O_{Pi} and its 3 main axes of the referential.

The second step is to define the three longitudinal offsets EU, EV and EW, respectively along the X, Y and Z-axes, in order to position the origin O_{Pi} at a corner of our part.

These three offsets are expressed with respect to the program frame OP and are those programmed as arguments of the function G150.

The method consists of rotating the table by α around C-axis so that the Z-X plane of the ICS frame lies in the Y-Z plane of the OP reference frame. Then, this resulting frame is tilted about X so that the X-Y interpolation plane is normal to the tool vector.



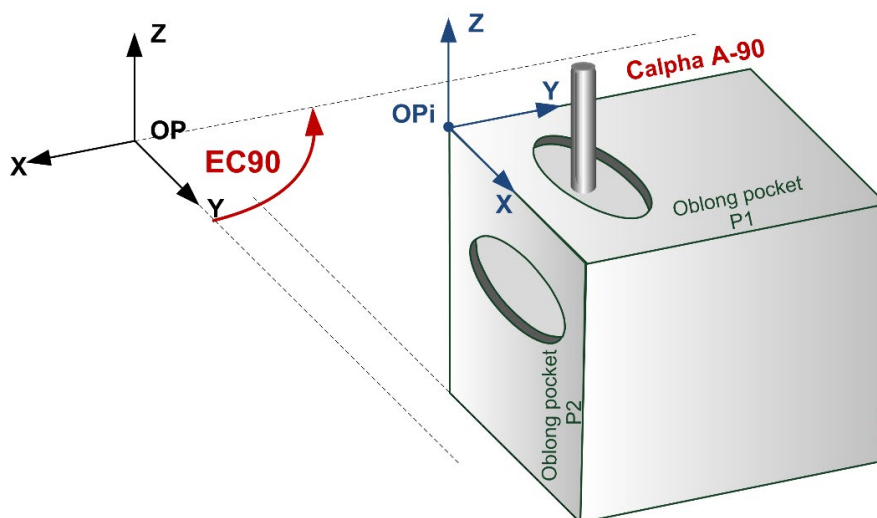
Alignment of the edge of the part on the Y-axis of the OP frame by rotating the C-axis table by an angle α

The figure below highlights the correct orientation of the pocket after tilting the cradle axis to the -90-degree position. Then from this position, we specify the Euler angles associated with the interpolation plane.

The change of coordinate system consists of programming a rotation of 90 degrees around the Z-axis of the coordinate system OP, using the EC argument to define the inclined coordinate system.



Only the Euler angles used in the transformation are programmed in the call of function G150.



Orientation of the part after the rotations A-90 and Calpha
The interpolation plane programmed by iPLANE and with origin Opi
G150 EU.. EV.. EW.. EC90

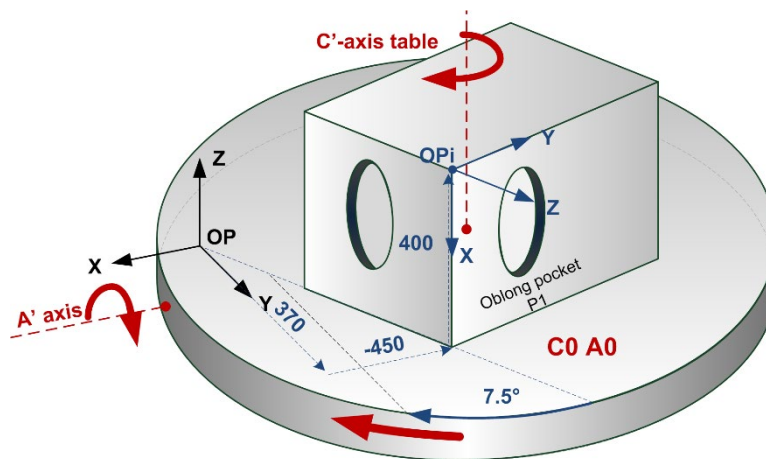
9.3.3.2 Summary of the implementation procedure

The three offsets EU, EV, EW are defined when the rotary axes A and C are in their zero-reference position.

The machining face is correctly oriented vertically to the direction of the tool, i.e., the interpolation plane is correctly oriented with respect to the tool.

Finally, one specifies the three angles of Euler EA, EB, and EC with respect to the OP frame.

Example of programming the machining of the oblong P1 pocket



Positioning of the ICS frame for machining of the pocket P1

\\ machining the first oblong pocket using the inclined plane.

G150 S0

\\ Cancel the inclined frame.

D1 G54 G510

G0 XYZ A0C0

G0 X-600 Y240 Z800

\\ approach position.

A-90 C7.5

\\ Prior orientation of the P1 pocket in the plane normal to the tool.

G150 EU-450 EV370 EW400 EC90

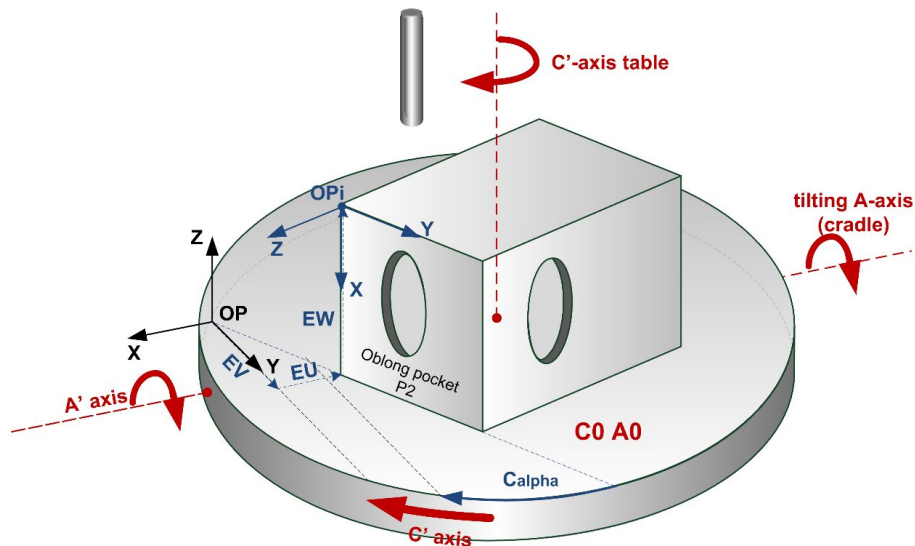
\\ definition of the interpolation plane.

9.3.3.3 Machining the second oblong pocket

Then we proceed to the machining of a second oblong pocket, named P2, located on the adjacent side of this same part.

The first step is to place the inclined frame on the side of the part, defining the interpolation plane, i.e. its origin O_{Pi} and its three main axes.

Then we define the three translational offsets EU , EV and EW , respectively along the X , Y and Z -axes, in order to fix the origin O_{Pi} at a corner of our part.

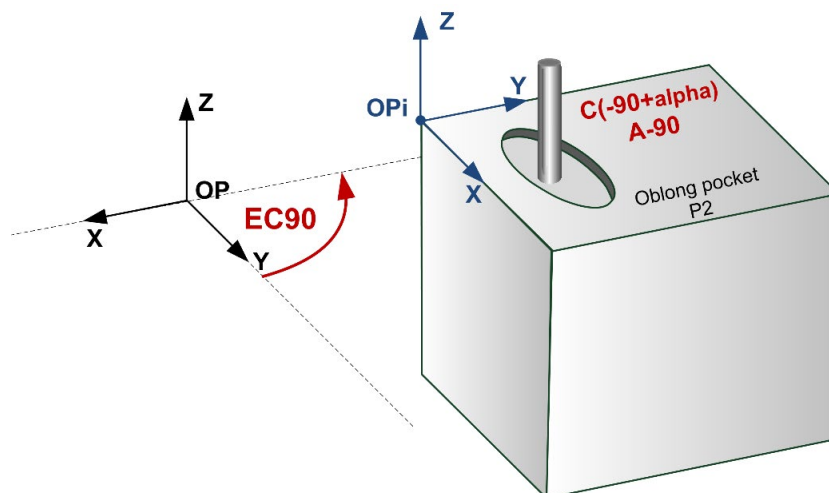


Tilting turntable (C-axis on A-axis) in the reference position A0C0
Preparing for the machining of the oblong pocket P2

The method consists in rotating the turntable (C-axis) by an angle equal to $(-90+\alpha)$ degrees (the plane $Z-X$ of ICS frame lies in the plane $Y-Z$ of OP frame) and in tilting the A-axis to the position -90 degrees.

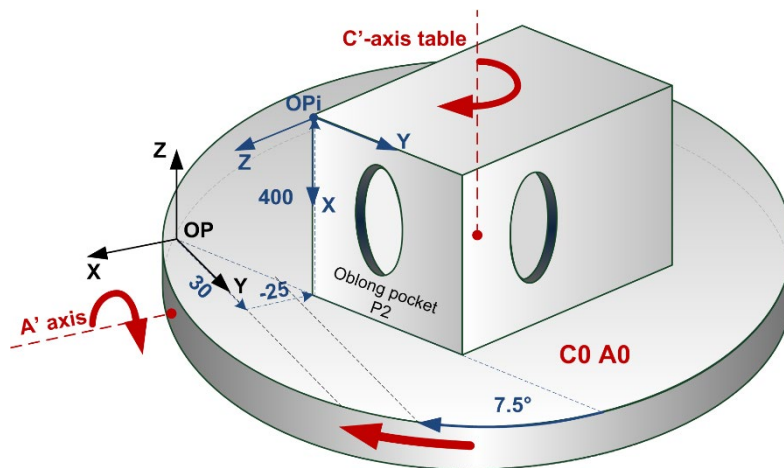
The figure below highlights the correct orientation of the pocket. Then from this position, we specify the Euler angles associated with the interpolation plane.

The change of coordinate system consists of programming a rotation of 90 degrees around the Z -axis of the program frame OP , using the EC argument to define the inclined coordinate system.



Preparation for the machining of the oblong pocket P2
Orientation of the part after rotating A-90 and C(-90+alpha)
The interpolation plane programmed by iPLANE function as G150 EUEVEW EC90

Example of programming to the machining of the oblong P2 pocket



Positioning of the ICS frame for machining of the pocket P2

\\ machining the second oblong pocket using the inclined plane.

G150 S0

\\ Cancel the inclined frame.

D1 G54 G510

G0 XYZ A0C0

G0 X-80 Y100 Z800

\\ Approach position.

L1=7.5-90 A-90 CL1

\\ Prior orientation of the P2 pocket in the plane normal to the tool.

G150 EU-450 EV370 EW400 EC90

\\ Definition and activation of the interpolation plane.

Third example for machining an oblong pocket located in an inclined face

Let us consider an oblong pocket located on the inclined face of a workpiece whose angle of inclination is 32.5° with respect to the adjacent upper face.

```
\\ Program for an oblong pocket located on a inclined side of the part
G150 S0                                \\ Cancel the inclined frame
D1 G55 G510
G0 XYZ A0C0
G0 X-600 Y350 Z850                    \\ Approach position.
C21.5 A-32.5                          \\ orientation of the pocket normal to the tool direction.
G150 EU-450 EV627 EW517 EC90          \\ Definition and activation of the interpolation plane
\\with respect to OP frame
```

9.3.4 Case of a C'-axis turntable combined with an A-axis Head

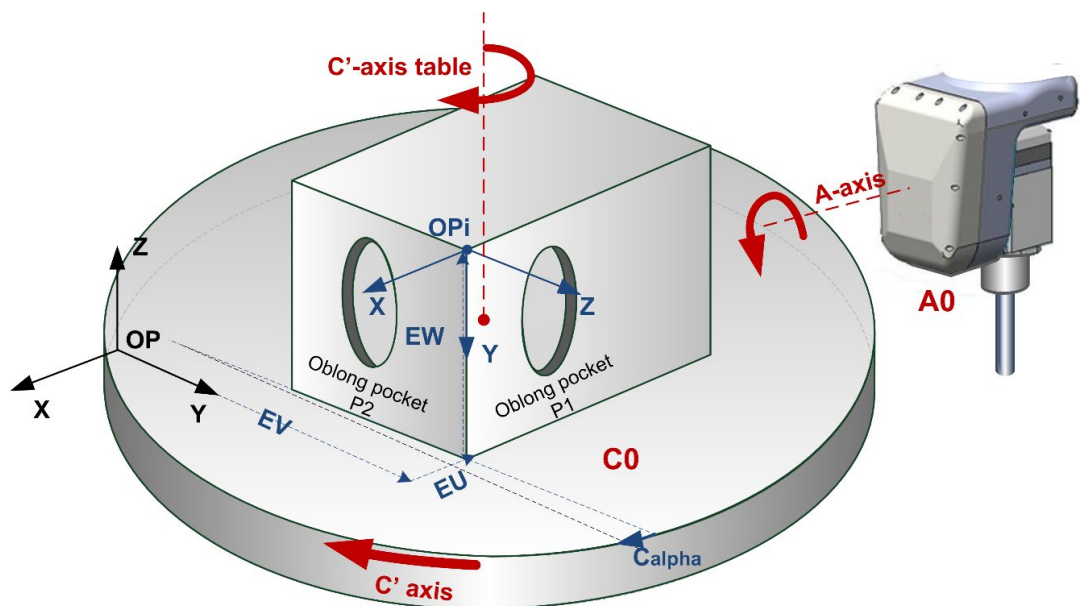
Consider a kinematics composed of a rotary table around Z-axis (C'-axis) combined with a simple A-axis twist head.

9.3.4.1 Machining the oblong P1 pocket

We want to machine the oblong pocket, noted P1, on one side of the part. This pocket is positioned with a slight orientation with respect to the main axes of the OP program frame when the rotary axes are at zero (i.e., A0 C0).

We place the inclined frame on the surface of the part, defining the interpolation plane, i.e. its origin O_{Pi} and its three main axes

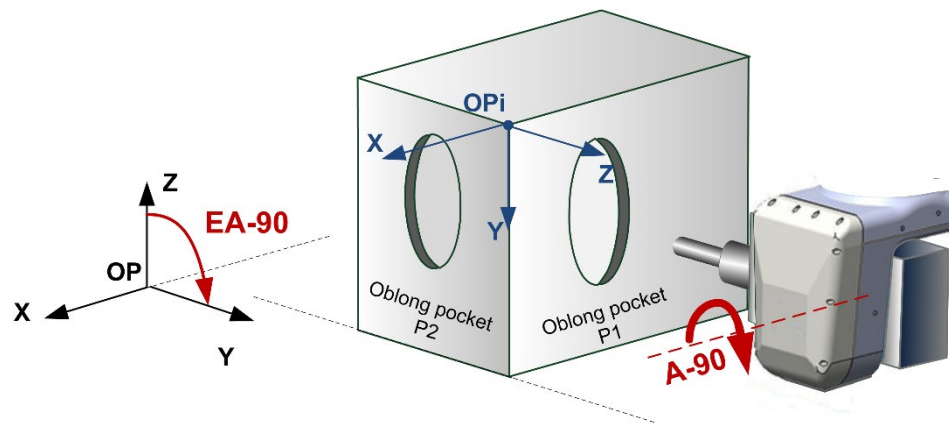
The three offsets EU, EV, EW are defined when the rotary axes A and C are in their zero reference position.



Turntable C-axis & A-axis twist head in the reference position C0A0
Preparing for the machining of the oblong pocket P1

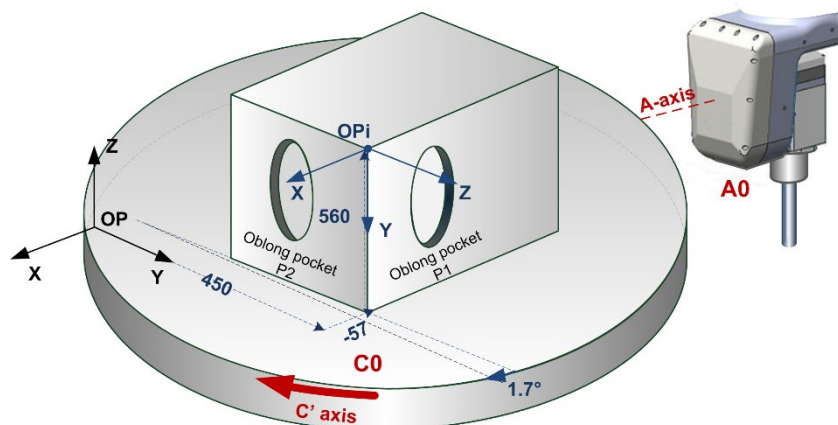
We orient the rotary table by an angle α so as to align the part with the main axes of the OP frame and we tilt the tool by a rotation of the A axis of -90 degrees. The tool is then normal to the interpolation plane.

Then, we use the Euler angle EA-90 by a rotation around the X axis with respect to OP frame to finalize the transformation from the OP frame to the inclined Opi frame..



Orientation of the part after rotating Calpha and A-90
Interpolation plane programmed by iPLANE function as G150 EU..EV..EW.. EA-90

Example of programming to the machining of the oblong P1 pocket



Positioning of ICS frame for machining the pocket P1

\\ machining the oblong pocket P1 using the inclined plane.

G150 S0

\\ Cancel the inclined frame.

D1 G58 G510

G0 XYZ A0C0

G0 X-150 Y870 Z300

\\ Approach position.

L1=1.7 A-90 CL1

\\ Prior orientation of the P1 pocket in the plane normal to the tool.

G150 EU-57 EV450 EW560 EA-90

\\ Definition and activation of the interpolation plane.

9.4 RTCP (Rotation around Tool Center Point)

9.4.1 General

The RTCP function allows keeping the tool center point (TCP) on the programmed trajectory when the tool orientation changes (in case of an end ball cutting tool).

It is possible to associate an inclined plane to the RTCP function when programming the tool center path. The RTCP function takes into account the geometric offsets of the 5 axes kinematics but it also considers the characteristics of the tool and its dynamic tool offsets.

The speed and acceleration of each linear axis but also their travel limits are controlled in real time.

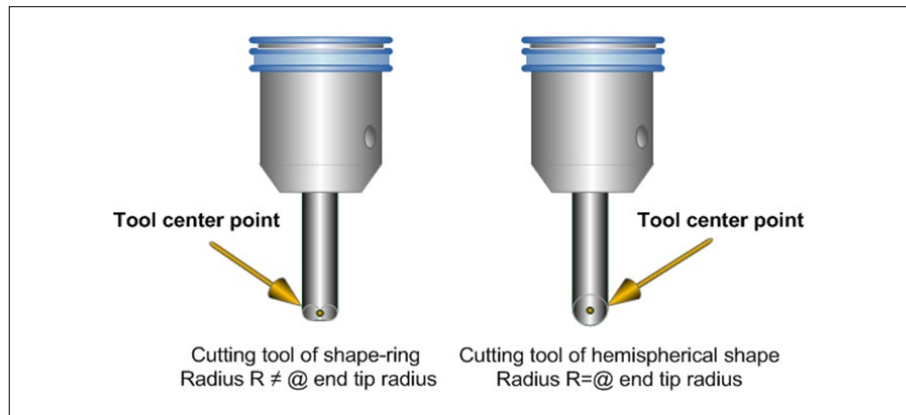
Six functions are provided:

1. Selection of one kinematic configuration among eight configurations.
2. RTCP function enabling with specification of a inclined coordinate system with respect to the MCS.
3. RTCP function enabling with specification of an inclined coordinate system after a tool positioning in a known point of this inclined referential (enabling in teaching mode).
4. Cancelling RTCP function and its inclined coordinate system.
5. Re-enabling RTCP function and its associated inclined coordinate system.
6. Re-enabling RTCP function without inclined coordinate system.

9.4.2 Convention of Tool Center Point definition

The RTCP function is used to change the tool orientation without moving the contact point between the tool and the piece.

This operation is effective only for an end ball tool (i.e. whose radius R is equal to the tool tip radius $@$). For a toroidal ($R \neq @$) the value of the tool tip radius is used to calculate the pivot point compensation.



It is important to comply with the rules defining the inclined planes described in chapter 9.2.2

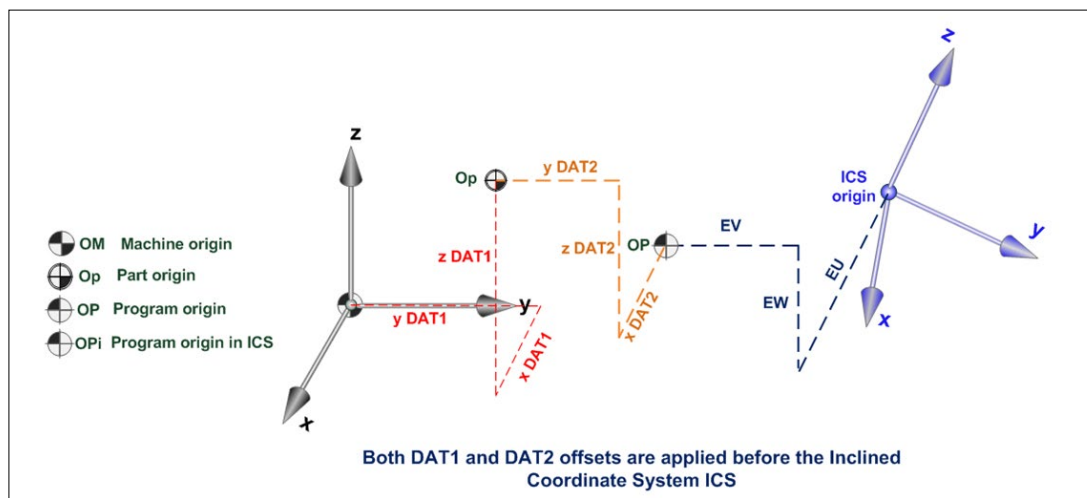
9.4.3 Datums DAT1 and DAT2

They are two possibilities to define when DAT1 and DAT2 are applied.

This is defined once and for all during the commissioning of the machine.

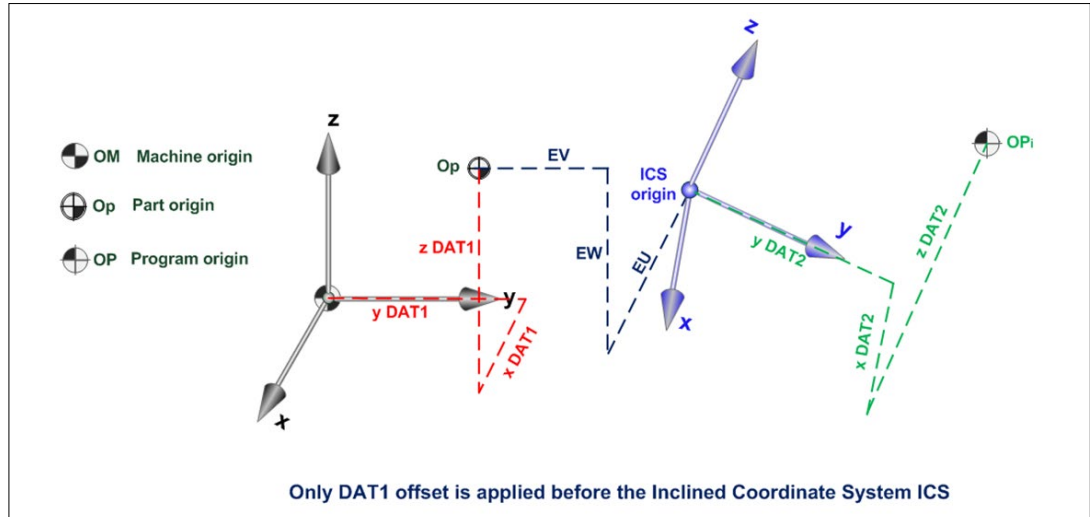
9.4.3.1 Both DAT1 and DAT2 offsets applied before ICS

The figure below shows the different transformations to move from a machine coordinate system (OM origin) to the inclined coordinate system (ICS).



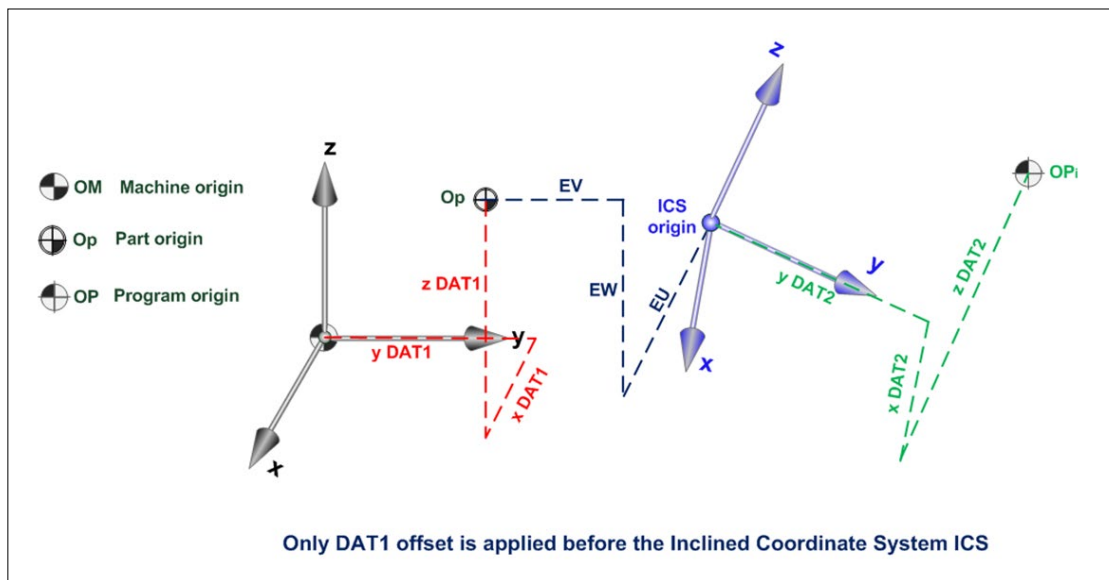
9.4.3.2 Only DAT1 offset applied before ICS

The figure below shows the different transformations to move from a machine coordinate system (machine origin) to the inclined referential (program origin).



9.4.3.3 Only DAT1 offsets applied before the ICS

The figure below shows the different transformations to move from a machine coordinate system (OM origin) to the inclined frame (OPi origin).



9.4.4 Programming Syntax**9.4.4.1 RTCP enabling with inclined coordinate system specification (ICS)**

G151 [EA...EB...EC...] EU...EV...EW...[T]

G151	G function of RTCP.
EA EB EC (optional)	Specify the three orientation angles of the new coordinate system (EA angle around X axis, EB angle around Y axis and EC angle around Z axis). If an angle is not specified, a zero value is assigned to it. The angles are expressed in degrees.
EU EV EW	Specify the position of inclined coordinate system origin with respect to the part program origin. EU, EV and EW are respectively the offsets of origin along the X, Y and Z. At least one of these three arguments must be specified. These parameters are expressed in mm.
T (optional)	Code number of the kinematic configuration among a max of eight configurations. This is required in case of multiple configurations for example removable milling heads. The correspondence between the machining heads and the T code is defined by the machine builder. When the T argument is omitted, the function chooses the previously programmed configuration (the first one by default).

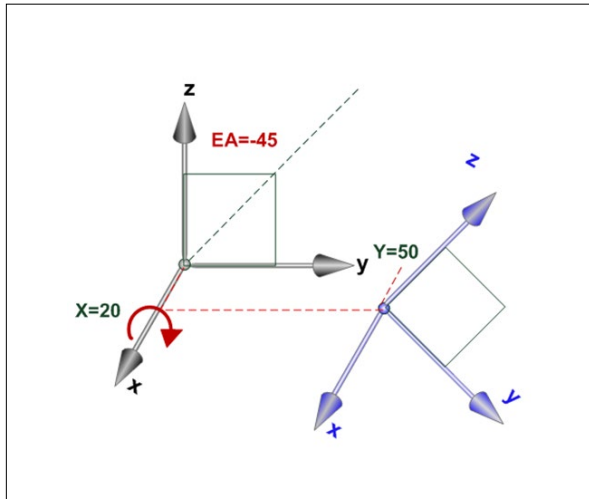


The RTCP function computes the 5 axis transformation with the current tool offset (L length, R radius and @ tool tip radius). This implies that a tool offset corrector has been programmed before activating the RTCP function.

Example: G151 EA-45 EU20 EV50 T1

This statement enables the RTCP function, set with the first kinematic configuration (T1) and defines an inclined coordinate system whose origin is located at 20mm along X and at 50 mm along Y from the initial workpiece origin.

The new referential (ICS) orientation is given by a rotation of -45 degrees around the X axis.



9.4.4.2 RTCP enabling with specification of an ICS after positioning

G151 X...Y...Z...[EA...EB...EC...] D...[T]	
G151	G function of RTCP.
X Y Z	Specify the position of the TCP with respect to the inclined coordinate system (ICS). If one of coordinates is not programmed, a zero value is assigned to it.
EA EB EC	Specify the three orientation angles of the new coordinate system. The presence of one of these three angles is mandatory. If an angle is not specified, a zero value is assigned to it. The angles are expressed in degrees.
D	Tool offset assigned to the current tool. This argument must be specified.
T (optional)	Code number of the kinematic configuration among eight max configurations. This is required in case of multiple configurations for example removable milling heads. The correspondence between the machining heads and the T code is defined by the machine builder. When the T argument is omitted, the function chooses the previously programmed configuration (the first one by default).

Example: G151 Z25 EA-45 D7

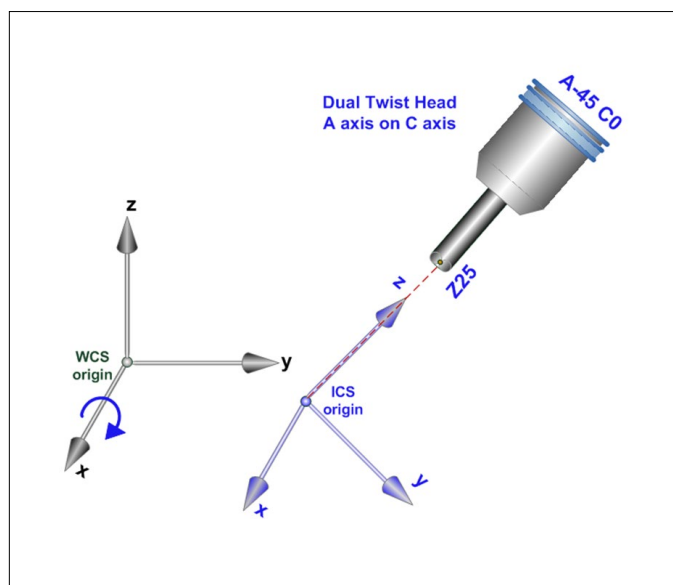
Consider a dual twist of which the swivelling axis is positioned at -45 degrees as shown in opposite figure.

This block specifies an inclined coordinate system in which the TCP is located at 0mm in X 0mm in Y and 25mm with respect to the ICS.

The origin position of ICS is computed knowing the current tool and its tool offset given by D7.

The new ICS orientation is given by a rotation of -45 degrees around the X axis.

The kinematic setup considered by the RTCP function will be the recently programmed



9.4.4.3 Enabling a tool ICS from the current tool orientation

G151 R1 D...	
G151	G function of RTCP.
R1	From the current tool orientation, this function computes an inclined coordinate system (ICS) whose origin coincides with the tool tip. The Z axis is directed along the tool axis direction. In general, the tool is oriented along Z+ when the 5 axis kinematics is in reference position. This 5-axis function allows programming an axial retraction or a lateral motion of the tool according to the referential TOOL ICS, in JOG mode or in MDI programming. Its use is restricted to 5-axis kinematics having a single-twist head or a dual-twist head. Otherwise, the following error message is triggered (Error 515 extension 2): Tool referential computation doesn't support this 5 axis kinematics.
D...	Tool offset assigned to the current tool. This argument must be specified.

Example G151 R1 D10

From the current orientation of the tool, the 5-axis function.

G151 R1 D..., calculates a set of three orientation angles defining the referential TOOL ICS:

EA angle around the X axis,

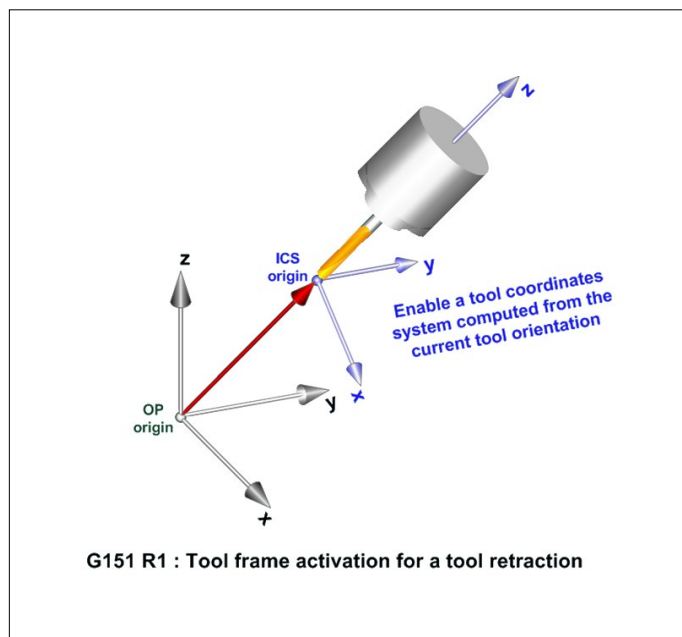
EB angle around the Y axis and

EC angle around the Z axis,

Then activates this referential Tool ICS whose origin coincides with the tool tip.

It is possible to activate the tool referential, associated with a particular kinematic configuration, by adding the additional argument T as following:

G151 R1 D10 T2



A frequent use would be a retraction after a tool breakage by moving the tool along the Z axis direction or would be to carry out an axial clearance movement.

For other uses, it is possible to program a tool movement in both lateral directions in TOOL ICS referential, but also to change the tool orientation around its TCP.

To know exactly the orientation of lateral axes of TOOL ICS referential, we can imagine a tool referential, parallel to the MCS referential whose origin is positioned on the TCP when the twist head axes are in zero position. The orientation of the ICS referential axes is deduced from the head axis positions.

Important note: Executing this function does not change the data of previous programming of RTCP function.

Then G151 S2 re-enables the RTCP function and re-activates the previous referential ICS.

Programming example

G90 G1 F	
D...	Select Tool corrector.
X... Y... Z...	
G151 EC17.5 EU0 T1	RTCP and inclined coordinate system enabled configuration 1.
A... C... Z... X... Y...	Machining with respect to the ICS
...	
G151 R1 D...	Compute a referential TOOL ICS and activate the RTCP.
G91 Z10 F200	Retraction along the Z axis of the ICS. G91 programming is optional because the axis motion is expressed in the referential TOOL ICS.
...	
...	
G90 F5000	
A... C...	
G151 R1 D...	Compute and enable a new referential TOOL ICS
X10 F200	Lateral motion

9.4.4.4 Suspension of the RTCP function and its associated ICS

G151 S0	
G151	G function of RTCP.
S0	This function cancels the RTCP function and its associated inclined coordinate system.



The RTCP function and its associated inclined plane are also canceled after a RESET or an end of program M02.

9.4.4.5 Re-Enabling the RTCP function without its associated ICS

G151 S1	
G151	G function of RTCP.
S1	Use this function to reactivate the RTCP function and to revoke its previous inclined coordinate system. The programmed movement of TCP is in the workpiece coordinate system.

9.4.4.6 Re-Enabling the RTCP function with its associated ICS

G151 S2	
G151	G function of RTCP.
S2	Use this function to re-enable the RTCP function and to restore the last context of the active inclined coordinate system (ICS). The last ICS was specified by the syntaxes: G151 [EA...EB...EC...] EU...EV...EW...[T] G151 X...Y...Z...[EA...EB...EC...] D...[T] and cancelled by G151 S0 or after a NC reset or a CNC initialization.

Example

Assume a RTCP function with its inclined plane is enabled and a tool change is requested.

The changer requires positioning and movements on axes at defined location fixed positions with respect to the measurement origin.

...	
D...	
X... Z...	
A... C... Z...X... Y...	
G151 EA..EU T2	RTCP and inclined coordinate system enabled, with configuration n°2.
Z...	
X... Y...	Machining with respect to the ICS.
...	
G151S0	Cancelling RTCP and its inclined plane.
G G52 Z17	Positioning for a tool change.
T... D... M6	
G151S2	Reactivating RTCP and its inclined plane with configuration n°2.

9.4.5 Additional functions

9.4.5.1 Tool length correction

In case of a twist head, the length correction is included in the kinematic features of the head with its orientation given by IJK vector.

Therefore, the declaration and the dimension must be assigned to the tool correction before activating RTCP.

9.4.5.2 Tool wear offset

The tool length wear offsets less than 100 μ (0.1 mm), immediately taken into account and included in the twist movements with the orientation given by IJK vector at the reference pose.

The total length of tool used in the RTCP transformation is:

$$L_{tool} + L_{tool\_wear} - @_{tip\_radius}$$

9.4.5.3 Tridimensional tool correction (G29)

When the RTCP function is used with **3-axis tool correction** (i.e. no tool orientation vector IJK in the G29 blocks), the CNC can only process spherical tools.

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

9.4.5.4 Table off-centering (DAT3)

The offsets related to an off-centred 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.

Example n°2

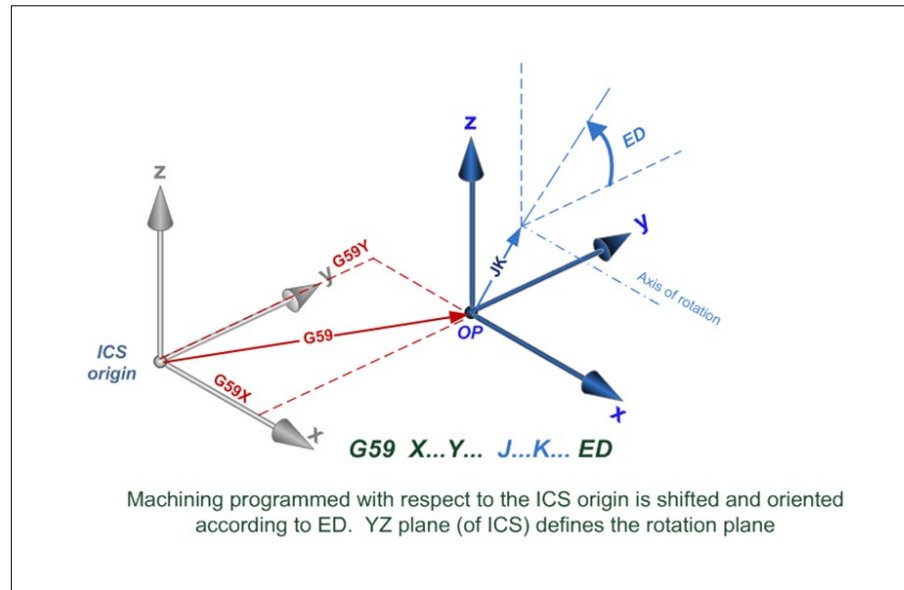
Angular offset programmed in a G19 plane with respect to an ICS origin.

RTCP enabling with its previously programmed ICS:
G151 S2

Programming in absolute dimensions, G19 plane:
G90 G19YZ

Program origin offset with definition of a center of rotation of an angular shift ED in the G19 plane (rotation around X axis with respect to the inclined referential)

G59 X..Y..Z..J..K..ED30

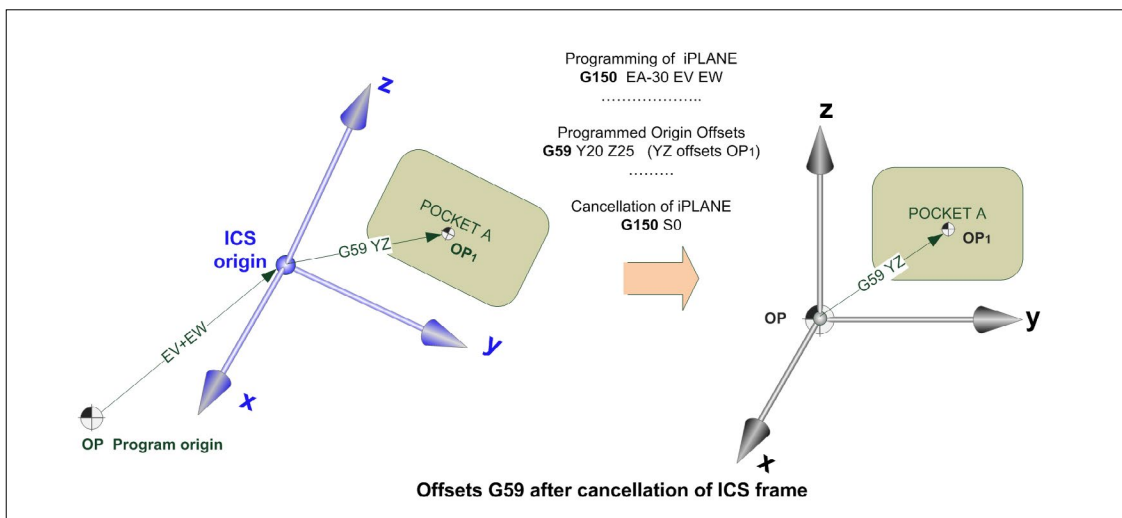


The validation or suspension of an inclined plane does not affect the G59 offsets. However, after suspending an ICS, those programmed offsets become relative to the WCS referential

On the other hand, after validation of an inclined plane, the origin offsets G59 become relative to the ICS referential

9.4.6 Notes about the RTCP function and its associated inclined plane

- The RTCP function and its associated inclined plane are canceled after a RESET or an end of program M02.
- A tool offset change when the RTCP function is active causes an **error message 16**.
- When the RTCP function is enabled, homing is prohibited and generates **error message 24**.
- Any change of the DAT1 or DAT2 datum shifts, either directly or by programming of the corresponding external parameters (E60xxx and E61xxx), is prohibited when RTCP is enabled.
- Programming of movements in G52, i.e. in absolute coordinate referenced to the measurement origin, when an inclined plane is active is prohibited and returns **error message 24**, except if the machine builder has authorized it via the dedicated machine parameter.
- Programming of a movement with DAT1 or DAT2 inhibited is prohibited when RTCP is enabled.
- The rotary axes of the RTCP function have to be homed. Otherwise **error message 159** is generated.
- The validation or suspension of an inclined plane does not affect the G59 offsets. However, after suspending an ICS, those programmed offsets become relative to the WCS referential. On the other hand, after validation of an inclined plane, the origin offsets G59 become relative to the ICS referential.
The same applies to the other transformations: scaling factor (function G74), angular offset (function ED), the mirror function (G51), programming in Inches/ Metric (G70) and the axis address re-assignment.



From version 4.1 the backup of RTCP context is saved in memory segments. These segments are reserved for NUM. So in case of power failure, the restore of data will be guaranteed and it will be possible to re-enable the RTCP function with its last inclined frame assuming homing made on the rotary axes.

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

A

Page deliberately empty

Summary

10	Tool Vector Programming and 3D Workpiece Positioning Compensation.....	267
10.1	General	267
10.2	Implementation	268
10.3	Tool orientation	269
10.4	Details on G codes	270
10.5	Complement.....	271
10.6	Precautions and limits of usage.....	272
10.7	Early Block Change	275
10.8	Limitations.....	276
10.9	Modality.....	276
10.10	Programming	277
10.11	Mode Q0 – EBC suspended	277
10.12	MODE Q1 – EBC when traversed distance higher than.....	278
10.13	MODE Q2 – EBC when residual distance lower than	278
10.14	MODE Q3 – EBC with PLC control.....	279
10.15	MODE Q4 – EBC immediate	279
10.16	MODE Q5 – Wait for confirmation bit	279
10.17	MODE Q6 – only set output bit xx when distance traversed higher than.....	280
10.18	MODE Q7 – only set output bit when remaining distance lower than	280
10.19	PLC Flags	280

Page deliberately empty

10 Tool Vector Programming and 3D Workpiece Positioning Compensation

10.1 General

When a workpiece is installed on a machine, it can be very difficult to align it perfectly due to weight, structure, previous machining operation or any other reason. Any tilt quickly becomes a problem by generating a change in the incidence of the tool. A common solution is, after identifying offsets and tilt angles, to rework the part program to take into account misalignments, but this obviously takes some time.

To handle this issue, NUM provides the 3D WPC function (3D Work Piece Positioning Compensation).

The goal of this function is to compensate the position and orientation of a part to machine with respect to its theoretical values without having to reprocess the program by defining a Balanced Coordinate System (BCS) of the part with respect to an ideally positioned one.

This function currently only applies when RTCP is enabled.

In addition, the tool orientation defined in the part program should be constantly adapted during machining to maintain the desired tool orientation with respect to the ideally positioned part.

For this the function "Tool vector programming" later explained in this chapter should be used.

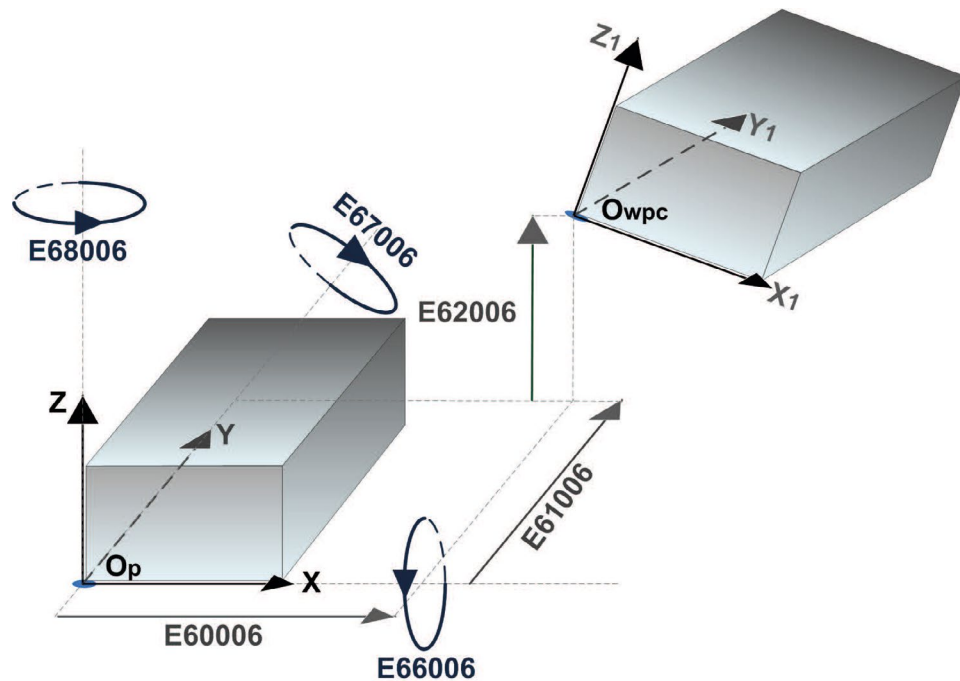
10.2 Implementation

The 3D WPC function only relies on six E parameters to define the part malposition

Shifts theoretical part referential : E60006 – E60001 – E62002

Angles (Pitch, Roll and Yaw) : E66006 – E67006 – E68006

These parameters are read write and can be entered by programming or on the HMI. They define the position and orientation of the BCS (Balanced Coordinate System) with respect to the theoretical Part Coordinate System.



The figure below represents the link between the different referential systems.

The referential $O_p(XYZ)$ represents the theoretical part referential along which the 3DWPC parameters are applied.

10.3 Tool orientation

We have seen that to correctly machine an off-positioned part it is necessary to adapt the tool orientation to maintain its theoretical orientation with respect to the part. To do this the CNC must be aware of the tool orientation either explicitly or by recalculation as well as the mechanical structure of the machine. The following mutually exclusive G codes have been created for this purpose:

G code	Function	Condition	Status after start-up or reset	Specific additional arguments?
G30	The rotary axes are programmed by their physical values. This is the classic current mode; it is created in order to cancel the other G32, G34, G34- when necessary.	No one	Active by default	No
G32	The tool orientation is programmed directly via an extended syntax to define the tool vector. The CNC will compute the physical position of the Tool head axes taking into account this vector and the kinematic head structure. The other rotary axes remain programmed by their dimensions.	RTCP enabled with kinematic number argument	Inactive by default	No. The expression of the tool vector uses a new programming syntax.
G34-	The rotary axes are programmed by their physical dimensions. Considering the Tool orientation axes, the NCK will constantly calculate the tool orientation from these coordinates and compensate it according to the part balancing parameters before moving these axes to their new position.	RTCP enabled with kinematic number argument	Inactive by default	No
G34	Programming the rotary axes by their physical dimensions. Considering the Tool orientation axes, the NCK will constantly calculate the tool orientation from these coordinates and compensate it according to the part balancing parameters and an inclined plane before moving these axes to their new position.	RTCP enabled with kinematic number argument	Inactive by default	No

We have seen that in order to compute the rotary axes dimensions (from the tool vector in G32 or recalculated in G34 / G34-) the NCK needs to be aware of the machine kinematic.

Therefore, a new parameter (N) defining this kinematic has been added to the G26 code. Like always, G26 is handled by the %10151 macro which must be recompiled using Flexium Tools 4.1.2 or above

Currently the above functions (excluding G30) are available for the two following kinematics:

Type 2: Dual twist A on C

Type 4: Dual twist B on C

10.4 Details on G codes

G30

G30 is intended to program the RTCP involved rotary axes by their **physical dimensions**.

It is the standard programming mode. This G code has been created for compatibility and to leave the possibility of cancelling the new G32 / G34 codes.

G32

G32 is intended to define the **tool vector directly** (abstract programming). From the expression of the tool vector, the NCK firmware will compute the dimensions of the RTCP rotary axes.

Expression of the Tool vector

The tool vector is programmed by its projections along the X, Y, Z axes of the balanced coordinate system or inclined coordinate system.

The programming addresses for the tool projections are respectively:

A_0 = ...,

B_0 = ...,

C_0 = ...

The values following the sign "=" can be expressed as usual by a numerical value, an L variable, an E parameter, a symbolic variable or an expression between {...}.

The three projections are not necessary: any missing projection is considered as null; but at least one projection must be programmed in order to get a tool vector orientation.

There is no need to norm the vector at a specific value, the NCK firmware norming it internally to 1.

Example

G30 A0 C0

G32

G1 X.. Y.. Z.. L0 =1/_SQRT3 A_0=-L0 B_0=-L0 C_0=L0

G34 -

G34 – is intended for programming the tool vector by the **physical dimensions** of the RTCP involved axes (like G30) but in this mode the NCK will adapt these values by computing in real time the tool orientation and adjust it taking into account the part balancing parameters.

G34

G34 is intended for programming the tool vector by the **physical dimensions** of the RTCP involved axes (like G30) but in this mode the NCK will adapt these values by computing in real time the tool orientation and adjust it taking into account the part balancing parameters and an inclined plane.

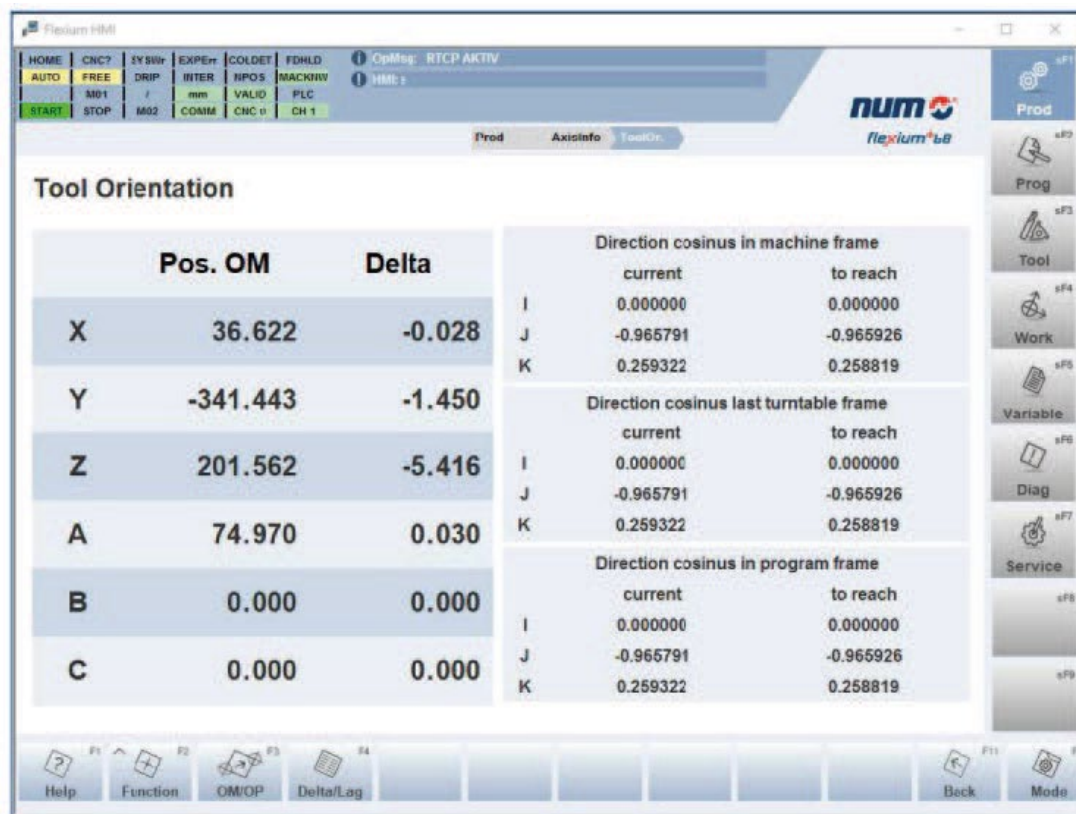
10.5 Complement

The current tool vector data is available as E parameters (read only):

- E790020, E79021, 79022 (in Machine coordinate system respectively X, Y and Z projections).
- E790023, E79024, 79025 (in Part coordinate system respectively X, Y and Z projections).
- E790026, E79027, 79028 (in Balanced coordinate system respectively X, Y and Z projections).

The display of current tool vector is available even outside of Tool vector programming – tool direction compensation. It is available as soon as a twist is present in the RTCP kinematic description. On the other hand, the display of the tool vector to reach at end of the block requires the programming of G32, G34-, or G34.

The tool orientation display is available on the HMI.



	Pos. OM	Delta
X	36.622	-0.028
Y	-341.443	-1.450
Z	201.562	-5.416
A	74.970	0.030
B	0.000	0.000
C	0.000	0.000

Direction cosinus in machine frame		
	current	to reach
I	0.000000	0.000000
J	-0.965791	-0.965926
K	0.259322	0.258819

Direction cosinus last turntable frame		
	current	to reach
I	0.000000	0.000000
J	-0.965791	-0.965926
K	0.259322	0.258819

Direction cosinus in program frame		
	current	to reach
I	0.000000	0.000000
J	-0.965791	-0.965926
K	0.259322	0.258819

10.6 Precautions and limits of usage

3D Workpiece compensation and Tool Vector Programming is available as an option on the CNC

In the case of Tool vector programming, the tool direction compensation supposes small and gradual changes of the orientation programmed. Of course large changes are allowed for example to move to a start position with a particular orientation. In this case it should be handled by the CAM system at a safe distance from the part to avoid collision.

The strategy is to work by choosing the minimal relative path to travel for the RTCP rotary axes. In case of RTCP modulo rotary axes, the NCK forces the minimal relative path.

➔ Important: If there is RTCP modulo rotary axis, when switching from {G32, G34-, G34} to G30 it is strongly recommended to set E11010 to 0 at the beginning of the part program in order force the minimal relative path continuously in G30.

Whatever the type of RTCP rotary axes, it is necessary to program at least once the RTCP rotary axes in absolute programming G90 before using G32, G34- or G34. The reason is that the previous dimensions programmed of these rotary axes should be available to calculate the possible relative paths.

Polynomial programming is allowed but not for RTCP rotary axes.

Examples

Using 3D WPC is done in five simple steps:

- Determine the part misalignment shifts and rotation by probing or any other mean
- Enter those value in the part program in G30 mode (standard mode)
- Make sure the rotary axes are programmed at least once in G90 G30 mode.
- Activates RTCP
- Activate G32 : Programming rotary axes via tool orientation
or G34 / G34- in case the rotary axes are programmed by their physical coordinates
(G34 allows for defining an additional inclined plane with ICS referential)

Programming in G32

%100(ROT_X45_Y45_Z45 NUM G34)()

(Define 3DWPC data)

E60006=0/U1 E61006=0/U1 E62006=0/U1 E63006=0/U1 E64006=0/U1 E65006=0/U1 E66006=0/
M1 E67006=0/M1 E68006=0/M1

G0 G30 M6 T1 D1

M3 S3000

G0 XYZAC

(Activates RTCP)

G151 T1 EU0 EV0 EW0 EA45 EB45 EC45

(Activates G32 - programming tool orientation)

G32

G00 X4.954 Y26.029 Z300.000 C_0=1

Y30.742 B_0=.00099483 C_0=.99999950

Y31.269 Z179.958 B_0=.01539319 C_0=.99988151

Y32.314 Z179.898 B_0=.04722848 C_0=.99888411

...

X1.954 Y37.355 Z178.725 B_0=.22692328 C_0=.97391263

Y37.832 Z178.531 B_0=.24655926 C_0=.96912771

...

G151 S0 G30

M2

Programming in G34

%100(ROT_X45_Y45_Z45 G34)()

(Defines 3DWPC data)

E60006=0/U1 E61006=0/U1 E62006=0/U1 E63006=0/U1 E64006=0/U1 E65006=0/U1 E66006=0/
M1 E67006=0/M1 E68006=0/M1

G0 G30 M6 T1 D1

M3 S3000

G0 XYZAC

(Activates RTCP)

G151 T1 EU0 EV0 EW0 EA45 EB45 EC45

(Activates G34 – programming via rotary axes coordinates)

G34

(Programming physical axes coordinates)

G00 X4.954 Y26.029 Z300.000 A0.000 C0.000

G01 Z179.966

X4.660 Y26.044

Y30.742 A-0.057

...

X1.953 Y32.834 Z179.845 A-3.651

...

G151 S0 G30

M2

10.7 Early Block Change

General

The function Early Block Change (EBC in short), gives the user the ability to begin execution of the next block of part program while the current block is still executing. This allows the machine to be performing two blocks at one time leading to speeding up production cycles in many different industries. In addition, the EBC function also provides a very fast signal exchange between PLC and part program, allowing the PLC to control machine motion and get information about program execution using 64 dedicated flags (32 I, 32 O).

Furthermore, the NCK communicates to the PLC when certain conditions in the program execution have been reached, allowing the PLC to start auxiliary operations without the need to program M functions – and thus without stopping any axis motion.

The different modes of operation (programmed by address Q) are:

- Q1: Begin next block when at a certain distance from the start of the current block.
- Q2: Begin next block when at a certain distance from the end of the current block.
- Q3: Begin next block on a signal from the PLC.
- Q4: Immediate EBC, next block starts immediately after the current block.
- Q5: No EBC, wait for a confirmation bit from the PLC to start next block.
- Q6: No EBC, sets a PLC bit after the block has travelled a certain distance.
- Q7: No EBC, sets a PLC bit when the remaining distance is below a certain value.

Syntax

The function is activated by the G777 code with the following syntax

G777[+/-] [Q..] [D..][I..] [O..] [R..][P..]	
	- Q selects the operation mode of G777. - D set the distance along the path (travelled or to be travelled) for the earlier block change. - I specifies the input bit which control the block change - O Specifies the output bit to set when a block change occurred - R Specifies the output bit to clear when a block change occurred - P Specifies the output bit to set when the traversed/residual distance has reached the value D
G777+	Activates the modal EBC feature, which remains active until G777- or M02 or RAZ.
G777-	Deactivates modal EBC
G777	Allow EBC or bit control on a single block.

All the functions can be assigned using the NUM syntax.
i.e. through E or L parameter, symbolic variables and so on.

10.8 Limitations

- Only one block anticipation at the time is possible. This means that only two interpolations at the time are possible. If three blocks were required at the same time the third will wait until the first or the second ends.
- The same axes cannot be programmed on two consecutive blocks, if they were the earlier block change simply will not occurs.
- Only linear and circular interpolations are allowed, obviously not possible to execute EBC on two consecutive circumferences.
- Auxiliary functions (M S T etc.) prevent earlier block change. F and G0 are allowed. In some cases M are allowed, for example M initial on a block does not prevent EBC as well as M final on the next.

IE:

```
N100 M3 S100 X100      possible to do EBC
N110 Z100 M5
```

- The block where the earlier block change may occur is always ended at null speed.
- Blocks change can be anticipated only in continuous mode execution

If the conditions for an early block change are not fulfilled the second block will be executed when the previous is completed (as normal cases). No messages will be issued.

10.9 Modality

The syntax G777+ makes the function modal.

Using G777+ the D and Q values become modal too.

When EBC is needed on several blocks, this modality simplifies programming, letting the CNC to start every block before the previous was arrived.

The conditions to start the block earlier are described above. If the conditions are not satisfied then EBC is not applied and the machining proceeds normally.

Examples:

```
N10 G777+ Q2 D10
N100 X100 Y100      N110 starts 10 mm before the end of N100
N110 Z100           N120 starts 10 mm before the end of N110
N120 X50 A90        N130 could not start earlier because X axis is programmed in both
N130 X70            N150 starts 10 mm before the end of N130
N140 G777 D5        following blocks are anticipated of 5mm
N150 Y0Z0A0         N160 starts 5 mm before the end of N150
N160 ...
...
G00 X0 Y0 Z0
G1 F1000
N10 G777+ Q1 D40
N100 X100           when X is at X40 N101 starts
N101 Y10Z10         N101 will end before N100, thus N102 won't start early
N102 X0             N102 will be started when N100 ends
...
```

10.10 Programming

Distance in mode Q1 - Q2

The distance travelled or remaining in mode Q1 - Q2 – Q6 –Q7 is programmed with function D.

In case of more axes interpolated this distance is considered on the interpolated path and only on those axes which actually contribute in the feed computation.

Practically these are the axes of the basic referential or the axes selected with G92F.

```
N10 G92 FXY
N11 G777+ Q2 D10
N100 X100 Y100 Z100
N101 A90 C-90
```

N101 is executed when the remaining path on X and Y (Z is not considered) becomes less than 10 mm. For rotary axes the distance is taken into account using the internal unit value.

This could generate a little confusion, because in case of rotary axes the value programmed with X would be 10 times lower (resolution of rotary axes is supposed to be 10 time bigger than linear).

10.11 Mode Q0 – EBC suspended

Mode Q0 simply suspends EBC when it is in modal manner.

It's actually equivalent of not programming G777+ or programming G777- but it let the possibility to keep G777 active.

```
N10 G777+ Q2 D5
N100 X100 Y100
N101 Z100
N102 G777 Q0          EBC temporarily suspended
N103 X50 Y150
G777 Q2
```

The block N101 is started 5 mm before N100 ends, but N103 is started when N101 ends.

10.12 **MODE Q1 – EBC when traversed distance higher than**

In mode Q1 (G777+ Q1) the blocks are executed after the specified distance has been traversed.

```
N10 G777+ Q1 D15  
N100 X100 Y1000  
N101 Z100
```

In this example N101 will be started when 15mm are traversed on the N100 block.

If mode Q1 is active and lxx bit is programmed the two conditions are put in AND thus the block change will occurs only when the distance has been travelled and the bit is set.

```
N100 X1000 Y400 G777 Q1 D10 I15
```

The example below is more complex but very interesting.

After 10 mm are travelled then bit 24 will be set, then block change will actually occurs only when the will be 25 is set.

Bit 24 could control an actuator and bit 25 would give the OK to early block change.

```
N100 Z100 G777 Q1 D10 P24 I25
```

10.13 **MODE Q2 – EBC when residual distance lower than**

In mode Q2 (G777+ Q2) the blocks are started when the remaining distance on the previous block is lower than the programmed value:

```
N10 G777+ Q2 D15  
N100 X100 Y1000  
N101 Z100
```

N101 is executed when the remaining distance on the trajectory is lower than 15 mm.

If mode Q2 is active and lxx bit is programmed the two conditions are put in AND thus the block change will occurs only when the remaining distance is lower than the programmed value and the bit is set.

```
N100 G2 I0 J0 X100 Y0 G777 Q2 D10 I15
```

The example below is more complex but very interesting.

When the remaining distance is lower than 10 mm then bit 24 will be set and block change will occurs only when bit 25 will be set.

Bit 24 could control an actuator bit 25 would give the OK to the early block change.

```
N100 Z100 G777 Q2 D10 P24 I25
```

10.14 MODE Q3 – EBC with PLC control

For modes Q3, 32 dedicated PLC bit, WCNC.BlockChangeCtrl[0..31] are available to allow the PLC to control the block change in the part program execution.

In mode Q3 the EBC occurs when the dedicated PLC bit (specified with parameter I) is set. Although modes Q3 could be modal a function (if programmed with G777+) the control input bit Ixx must be programmed on each block where the PLC controls on block change is needed.

G777+ Q3	selects block change in PLC controlled mode
N100 X100 G777 I15	block change occurs when bit 15 set
N101 Y100 G777 I16	block change occurs when bit 16 set
N102 Z100	block change occurs normally
N103 X0	

In the example above, N103 starts when N102 completely ends. This behaviour is because N102 has Q3 active (PLC control on the block change) but the input bit is not specified and thus the block change will occur only when the block normally ends. This is to avoid unwanted block change and to increase the program readability.

10.15 MODE Q4 – EBC immediate

In mode Q4 the block change occurs immediately.

```
N10 G777+ Q4
N100 X100 Y1000
N101 Z100
```

In this case, block N101 will be executed immediately after N100 has been started.

10.16 MODE Q5 – Wait for confirmation bit

In mode Q5 the early block change never occurs. That means the first block always ends before executing next block AND a further confirmation bit is required to start the second.

For example, after a positioning a proximity signal must be set into the PLC to allow the block change.

N100 G777+ Q5	
N110 X100 I10	block change occurs when X100 I reached and I10 is set
N120 Z-100	

Or, equivalent syntax

X100 G777 I10	block change occurs when X100 I reached and I10 is set

10.17 **MODE Q6 – only set output bit xx when distance traversed higher than**

Mode Q6 is similar to mode Q1 except that EBC will not occur; in fact the P output flag will be set when the distance travelled is greater than the programmed value.
It could be useful to communicate to the PLC that a certain position has been reached to command a certain PLC function.

```
X100
X0 G777 D30 Q6 P12      when X < 70 the bit 12 is set
```

10.18 **MODE Q7 – only set output bit when remaining distance lower than**

Mode Q7 is similar to mode Q2 except that EBC won't occur, in fact it the P output flag will be set when the remaining distance in the block is lower than the programmed value.

It could be useful to communicate to the PLC that a certain position has been reached to command a certain PLC functions.

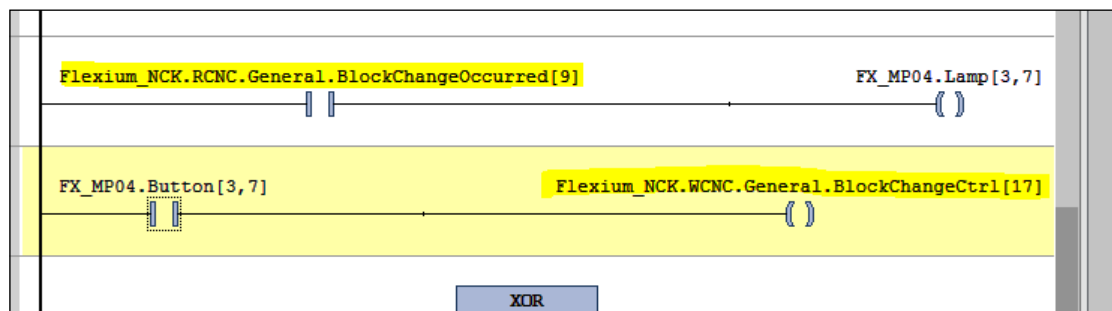
```
X100
X0 G777 D30 Q7 P12 R12      Bit 12 is set and when X<30 and cleared when X=0
```

10.19 **PLC Flags**

The information below is for reference. The flags are further described in the commissioning manual
The following PLC flags Flexium_NCK.RCNC.BlockChangeOccurred[0..31] and Flexium_NCK.WCNC.BlockChangeCtrl[0..31] are available into the NCK memory via the following E parameters.

E91500 to E91531 corresponds to RCNC.BlockChangeOccured[0..31]

E91600 to E91631 corresponds to WCNC.BlockChangeCtrl[0..31]



The bits BlockChangeCtrl are read by the part program with the G777 lxx instruction.

The bits BlockChangeOccurred are set by the part program with the instruction G777 Oxx, G777 Pxx or G777 Rxx.

- **G777 Oxx** sets the NCK output bit when the block change occurs.
- **G777 Pxx** sets the NCK output bit when the conditions on the remaining distance or distance travelled are fulfilled. The Pxx bit, only informs that we are in the zone where EBC would be possible; it does not necessarily mean that the block change has occurred because other conditions may not be fulfilled.

Example:

```

N100 X100
N101 X0 G777 D30 Q6 P12    when X < 70 the bit 12 is set but EBC did not occur
N102 X60                    because X is also programmed in N102.

```

- G777 Rxx reset the NCK output bit when the block change occurs
- G777 lxx controls the block change.

When the instruction G777 lxx is encountered, the program execution will always pass to the next block if and only if the programmed bit is set.

G777 lxx itself means normal block end and wait signal before proceeding (similar to 777+ Q5 : normal block end and wait signal, which will be specified on the block)

```

N100 Z100
N101 G777 I30                I30 should be set before starting N102
N102 M03 S100

```

```

N10 G777+ O2 D10            Early block change 10 mm before end of every block
N11 XYZG                    No EBC because Z is programmed in the next block
N100 Z100                   Block change at Z90
N101 X100 G777 O10          O10 is set when the block change (10 mm before end)
N102 Y200 G777 I10          EBC when Y > 190 and I10 is set
N103 Z0 G777 R10            Position to Z0 and then clear the output bit 10

```

Page deliberately empty

Summary

11	Indexed axes names.....	285
11.1	Declaration.....	285
11.2	Programming	285

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

A

Page deliberately empty

11 Indexed axes names

General

In standard, each channel can contain up to nine programmable axes with the following characteristics:

- Three primary axes X, Y, Z.
- Three secondary axes U, V, W.
- Three rotary axes A, B, C.

For some application, it might be useful to have several axes with identical names only discriminated by an index: e.g. X1, X2, Y1, Y2, Y3 etc. The function "Indexed Axes Names" or "Indexed Programming of axes", evolution of the previous "Axis Address Assignment" names is intended for that.



REMARK

This function is validated by a machine parameter (P7 N4.2). Refer to the machine tool manufacturer documentation. When validated it can be temporary disabled and enabled by parameter E11011.

11.1 Declaration

Declaration is performed via the character @

@X1 = Y : The address X1 refers to axis Y

@X2 = V : The address X2 refers to axis V

The index value is between 1 to 9

The name is necessarily in the list X, Y, Z, U, V, W, A, B, C exclusively

11.2 Programming

Once declaration is made programming can be performed using the "=" sign.
Considering the declaration above

X1=100 // equivalent to Y100 (with the declaration @X1=Y)

X2=200 // equivalent to V200

Page deliberately empty

Summary

12	N/M AUTO function	289
12.1	General	289
12.1.1	Non interpolated axes	289
12.1.2	Errors in N/M AUTO	289
12.2	Using the N/M AUTO function	290
12.2.1	Information for the machine tool builder	290
12.3	Stopping and restarting in N/M AUTO mode	291
12.4	Checks included in N/M AUTO	292
12.4.1	Acceleration checks	292
12.4.2	Speed checks	292
12.4.3	Check of travels	292
12.4.4	Miscellaneous checks	292

Page deliberately empty

12 N/M AUTO function

12.1 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.



NOTE

Please note that the N/M Auto function is used with the MLEGB (Multi Level Electronic Gear Box feature).

12.1.1 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.

12.1.2 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 M00040.

12.2 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.

12.2.1 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].FeedOverride` (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 M00040.

12.3 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 M00040.

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

A

12.4 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.

12.4.1 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.

12.4.2 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 authorized on the driven axes.

12.4.3 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.

12.4.4 Miscellaneous checks

No test is made relative to overspeed (G12).

Homing must be completed on the NMA.

Page deliberately empty

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
A

Page deliberately empty

Summary

13	Polynomial interpolation.....	297
13.1	General	297
13.1.1	Distinction between segmented and smooth polynomial interpolation	297
13.2	Segmented polynomial interpolation	298
13.2.1	Notes on the axes and coefficients programmed	298
13.2.2	Geometric transformations	299
13.2.3	Radius correction using polynomial interpolation	299
13.2.4	Interpolation Feed rate.....	300
13.3	Smooth polynomial interpolation	301
13.3.1	Restrictions on smooth polynomial interpolation	302

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

A

Page deliberately empty

13 Polynomial interpolation

13.1 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.

13.1.1 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 characterized by the end position followed by the coefficients of increasing degrees separated by «/».

13.2 Segmented polynomial interpolation

Syntax

N.. G01 X../C_{x1}/.../C_{xn} Y../C_{y1}/.../C_{yn} Z../C_{z1}/.../C_{zn} ...

G1	Linear and polynomial interpolation function.
X..	X axis end coordinate.
/C_{x1} /... /C_{xn}	First, second, n degree etc. coefficients for X.
Y.. /C_{y1} /... /C_{yn}	Y axis end coordinate and coefficients.
Z.. /C_{z1} /... /C_{zn}	Z axis end coordinate and coefficients.
...	Other axes coordinates and coefficients.

13.2.1 Notes on the axes and coefficients programmed

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

The coefficients are expressed in:

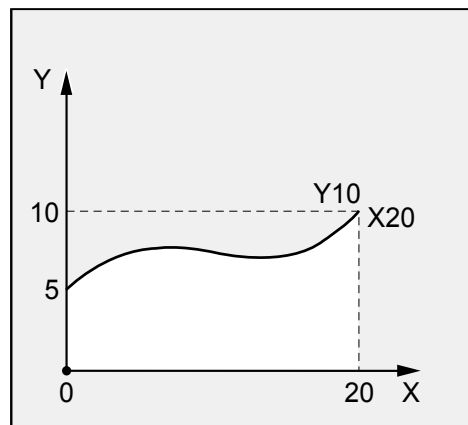
- mm (or inches if G70 is active) for linear axes,
- Degrees for rotary axes.

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

13.2.2 Geometric transformations

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

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

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
```

13.2.3 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.

13.2.4 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.

13.3 Smooth polynomial interpolation

In the block syntax, each polynomial is characterized 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.

Syntax

N.. G01 X../C_{x1}/.../C_{xn} Y../C_{y1}/.../C_{yn} Z../C_{z1}/.../C_{zn} ... I..

G01	Linear and polynomial interpolation function.
X..	X axis end coordinate.
/C_{x1} /... /C_{xn}	First, second, n degree etc. coefficients for X.
Y.. /C_{y1} /... /C_{yn}	Y axis end coordinate and coefficients.
Z.. /C_{z1} /... /C_{zn}	Z axis end coordinate and coefficients.
...	Other axes coordinates and coefficients.
I...	Curve length.



NOTE

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)
```

13.3.1 Restrictions on smooth polynomial interpolation

The following restrictions apply to smooth polynomial interpolation:

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

Page deliberately empty

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
A

Page deliberately empty

Summary

14	B-Spline function.....	307
14.1	General	307
14.2	Restrictions and programming errors	308
14.3	Precision about H parameter with G62.....	309

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

A

Page deliberately empty

14 B-Spline function

14.1 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.

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.

14.2 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.
L0-L19	LVariables (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

14.3 Precision about H parameter 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 $50\text{mm}/10 = 5$.
- 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" = $1000\text{mm}/1 = 1000$.

Page deliberately empty

Summary

15	Application specific E parameters and G functions	313
15.1	General	313
15.2	Continuous probing E11019	313
15.3	Enabling reference writing for virtual axes E11020.....	314
15.4	Assigning axes to a channel during Reset E11024.....	314
15.5	Inhibiting program modification and deletion E31002	314
15.6	Inhibit writing into memory segments E31003.....	315
15.7	Circle tolerance E32012.....	315
15.8	Tool disengagement in line to line transition E69004.....	315
15.9	Axis duplication and synchronisation E941.., E96.....	316
15.10	Monitoring drives test points E351.., E352.....	317
15.11	Swapping drive parameters E353.....	318
15.12	Recording continuous axes position G751	319

Page deliberately empty

15 Application specific E parameters and G functions

15.1 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 M00044 together with more details about the following.

Reminder

- It is possible to enter values in Hexadecimal or Binary as well as decimal.
- The value is introduced by _HEX or _BIN.

Examples :

E80000 = _BIN1101 equivalent E80000 = 13
E80000 = _HEX5A10 equivalent E80000 = 23056



CAUTION

The space character is significant. E80000 = _HEX 5 A10 means E80000 = 5 and move A to position 10. (Note the space between 5 and A10)

15.2 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 monitoring the change in the values of E7x001 parameters.

For E parameters detailed information please refer to the Flexium+ Programming manual M00040.

15.3 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 virtual 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 M00040.

15.4 Assigning axes to a channel during Reset E11024

A reset normally reassign the axes as they are defined by machine parameters.

This parameter permits to maintain the axes and spindles assignment during a reset.

Default value is = 0.

Standard behavior during reset, the spindles and axes are assigned in channels in accordance with the machine parameters.

- if bit 1 is set (E11024=1) the axes keep staying in their current channel when RAZ is given.
- if bit 2 is set (E11024=2) the spindles keep staying in their current channel when RAZ is given.
- bit 1 and bit 2 can be set simultaneously (E11024=3).

15.5 Inhibiting program modification and deletion E31002

Parameter E31002 is intended to prevent program modification in CNC memory when set.

This parameter is automatically reset at Power up and after a 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 M00040.

15.6 Inhibit writing into memory segments E31003

The function G76+ Sxx (see chapter 4.14) permits to save symbolic variables into NCK memory. When parameter E31003 is set this writing will not be authorized.

15.7 Circle tolerance E32012

In circular interpolation, the system allows a certain difference between the radius defined by start point and center and the radius defined by end point and center.

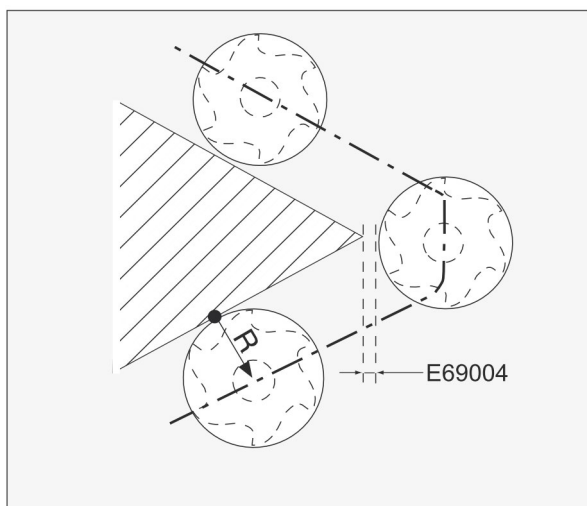
By default the difference is limited to 20 increments. It is defined by a machine parameter but can be edited via E32012. The value for this parameter should be between 20 and 255 increments. A reset restores the value defined in the machine parameter.

For E parameters detailed information please refer to the Flexium+ Programming manual M00040.

15.8 Tool disengagement in line to line transition E69004

When in cutter compensation mode (G41/G42, see Flexium+ Programming manual M00040) 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).



15.9 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 M00044. 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 and 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 and Intervention)

This is defined via parameter E961.. (32 parameters)

Synchronisation definition

Defined via parameters E962.. (32 parameters)

ex : E96202 = 1 Axis number 2 is synchronized 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 by they follow their master.

Symmetry definition

Defined via parameters E963.. (32 parameters)

ex : E96302 = 1 Axis number 2 will move in symmetry relative to its master.

An axis can not be simultaneously synchronized and symmetrical.

For E parameters detailed information please refer to the Flexium+ Programming manual M00040.

15.10 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 X Parameters manual M00031 or in the NUMDrive C parameter manual AMOMAN007 and into the Flexium Extended NCK Access Manual M00026.

The values 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 Flexium 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	32 bits value of measure points selected by V130 for drive xx
E351(32+xx)	32 bits value of measure points selected by V131 for drive yy

For E parameters detailed information please refer to the Flexium+ Programming manual M00040.

15.11 Swapping drive parameters E353...

For applications where the kinematic response might change according to the situation (switchable, 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 NUMDrive X Parameters manual M00031 for more detailed information.

15.12 Recording continuous axes position G751

For setup purpose, it might be interesting to record axes position.

This function that shall not be used in normal machine operation can be performed by the G751 following function.

G751 stores into the NC memory the position of the axes while a program is running.

The points are stored in a file that you can see in the NC program area, with the same format as an NC part program. This file can be downloaded and exploited by a spreadsheet for example

Syntax

G751 [Hpppppp] [On][Txxx][Qyyy]	
H	Program number where the points will be stored; by default: 20000 + program number in execution.
O	Position point type: 0 = OP, 1 = OM. default: OM.
T	Number of NC sample time when the points are stored ; default 1: every sample time (P50).
Q	Bitmap of the axis with position stored (i.e: Q7 means XYZ) ; default: moving axes.



This function is intended for setup purpose and does not provide advanced diagnostics. The files created are growing rapidly, therefore it must be executed under operator supervision. Recording is continuous until M0, M2 or G750.

Cancellation

Cancellation by G750.

Page deliberately empty

Summary

A	Appendix	323
A.1	Structured programming commands	323
A.2	Operators	324
A.3	String functions	325
A.4	Symbolic variable instructions	326
A.5	Program status access symbols	328
A.5.1	G and M functions.....	328
A.5.2	List of bits	328
A.5.3	Values	329
A.6	Symbols stored in L900 to L951 variables	330
A.6.1	Symbols stored in variables L900 to L925	330
A.6.2	Symbols stored in variables L926 to L951	330
A.7	List of Dynamic Operations	331
A.8	Coordinate conversions.....	333
A.8.1	General	333
A.8.2	Using the coordinate conversion matrix ▶ G24	333
A.8.2.1	Properties of the function.....	334
A.8.2.2	Error numbers and messages	334
A.9	RTCP function.....	335
A.9.1	General	335
A.9.1.1	Control of rotary axes	335
A.9.1.2	Control of rotary axes	335
A.9.2	Programming the RTCP function ▶ G26	336
A.9.2.1	General	336
A.9.2.2	Programming errors	336
A.9.2.3	Other Errors	336
A.9.2.4	Notes.....	337
A.9.3	Structure description	338
A.9.3.1	Twist Heads configuration	338
A.9.3.2	Representation of a twist head configuration	338
A.9.3.3	Turntables configuration	339
A.9.3.4	Representation of a turntable configuration	339
A.9.4	Other processing related to the RTCP function	340
A.9.4.1	Part on inclined plane	340
A.9.4.2	Table off-centering (DAT3)	341
A.9.4.3	Tool length correction	341
A.9.4.4	Tool wear offset	341
A.9.4.5	Tool correction (G29) : 3 or 5-axis tool correction	341
A.9.5	Restrictions and conditions of use	342

Page deliberately empty

A Appendix

A.1 Structured programming commands

CONT	Next increment of a loop
Syntax:	CONT
EXIT	Exit from a loop
Syntax:	EXIT  Remark EXIT can only be included in IF /ENDI sequences.
FOR/ENDF	Loops with control variable
Syntax:	FOR (variable) = (expression 1) TO/DOWNTO (expression 2) BY (value) DO (instructions) ENDF
IF/ENDI	Conditional execution of instructions
Syntax:	IF (condition) THEN (instructions 1) ELSE (instructions 2) ENDI
SWITCH/ENDS	Selective Jump
Syntax:	SWITCH expression CASE expression 1 statements to execute until " break" BREAK CASE constant expression 2 statements to execute until " break" BREAK DEFAULT statements to execute in other cases ENDS end of switch statement
REPEAT/UNTIL	REPEAT UNTIL loops
Syntax:	REPEAT (instructions) UNTIL (condition)
WHILE/ENDW	WHILE loops
Syntax:	WHILE (condition) DO (instructions) ENDW

A.2 Operators

+	Addition
-	Subtraction
*	Multiplication
/	Division
 	Remainder of the division
S (or _SIN)	Sine(degrees)
C (or _COS)	Cosine(degrees)
T (or _TRUNC)	Truncation
A (or _ATAN)	Arctangent in $]-90^{\circ}, +90^{\circ}[$
_ATAN2	Arctangent in $]-180^{\circ}, 180^{\circ}[$
_ATAN3	Arctangent in $]0^{\circ}, 360^{\circ}[$
R (or _SQR)	Square Root
&	Logical AND
!	Logical OR
_ASIN	Arcsine
_ACOS	Arccosine
_EXP	Exponential
_LN	Natural logarithm
_LOG	Base10 logarithm
_ABS	Absolute value
_TAN	Tangent
_HSIN	Hyperbolic sine
_HCOS	Hyperbolic cosine
_HTAN	Hyperbolic Tangent
_RAD	Conversion to radians
_DEG	Conversion to degrees
_BIN	Conversion of a binary value to decimal
_HEX	Conversion of an hexadecimal value to decimal
U	Convert to unit of linear axes
M	Convert to unit of rotary axes
_CEIL	Smallest integer value greater than or equal
_FLOOR	Largest integer value less than or equal
_ROUND	Return the nearest integer
_MAX	Search the maximum
_MIN	Search the minimum
_PI	3.142653....

A.3 String functions

_SLEN	StringLength
_SCMP	StringCompare
_SCPY	StringCopy
_SCAT	StringConcat
_PUTSTR	PutString
_GETSTR	GetString
_SETSPLC	StringToPlc
_GETSPLC	StringFromPlc
_SISTR	StringIndex
_SNDEL	StringDelChar
_SNINS	StringInsert
_SSUB	StringExtract

A.4 Symbolic variable instructions

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)]
C.INIT	Initialises the C.OFF function
Syntax:	C.INIT INP=[tab(M,N)] / OUT=[out] / SIDE=[side] / POLAR=[polar]
C.OFF	Normal offset to a closed contour
Syntax:	C.OFF OFFSET= [offset] or C.OFF OFFS_S= [start_offset] / OFFS_E=[end_offset]
CUT	Eliminates grooves with the tool relief angle
Syntax:	CUT * [Pa(7,NB)] / Angle
DELETE	Suppress variables from the stack
Syntax:	DELETE [IX]/[TAB1]/[PROF1] Deletion of variables from current level. DELETE *[IZ]/[TAB3]/[PROF2] Deletion of variables from the stack.
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
PUSH	Push a variable or set of variables onto the stack
Syntax:	PUSH L0 - L7
PULL	Retrieve a variable or set of variables from the stack
Syntax:	PULL L0 - L7
P.BUILD	Create a table for storing the dimensions of a profile
Syntax:	P.BUILD [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..
VAR/ENDV	Declaration of symbolic variables or arrays
Syntax:	VAR [A]=[.RX(7)] [B]=[.RX(8)] [C]=[.RX(9)] [K11] [K12] [K13] [V] [X] [Y] [Z] ENDV



Unlike Flexium it is necessary to specify the full keyword; it is no more possible to limit to the first four digits.

A.5 Program status access symbols

Symbols to access 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.5.1 G and M functions

[.BGxx]	G function addressing
[.BMxx]	M function addressing
[.IMab(xx)]	Spindle specific M functions with ab = {03,04,05,19,40,41,42,43,44,45}

A.5.2 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.5.3 Values

[.RD]	Tool correction number
[.RDX]	Tool axis orientation (G16)
[.REC]	Spindle orientation
[.REA]	Angle of trajectory
[.RED]	Angular offset
[.RF]	Feed rate
[.RG4]	Programmed dwell (G04 F..)
[.RG80]	G code value calling the subroutine
[.RLX]	List of axes programmed in the block
[.RLN]	Line number
[.RN]	Number of the last sequence (block) found
[.RNC]	Value of CNC function
[.RS]	Spindle speed
[.RT]	Tool number
[.RXH]	Current subroutine nesting level
[.IBO]	Spindle programmed
[.IRO]	Programmed value of abstract axes
[.IRDI 1...5]	Value of programmed angular offsets (G59 I.. J.. K..)
[.IREC]	Programmed spindle position
[.IRH 1...16]	Current program or subroutine number
[.IRI 1...5]	I, J and K arguments values
[.IRP 1...3]	P, Q and R arguments values
[.IRTX 1...9]	Offsets programmed on the axes
[.IRX 1...9]	Dimensions programmed on the axes
[.IRSL(1..4)]	Main to AUX3 spindle logical address
[.IRSR(1)]	Measure reference spindle logical address
[.IRSS(1..4)]	Main to AUX3 spindle physical address
[.IRS(1..32)]	Programmed speed of spindle 1..32

A.6 Symbols stored in L900 to L951 variables

A.6.1 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.6.2 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.7 List of Dynamic Operations

Cancel an operation (see chapter 8.1.4)	
Syntax:	On = 0
Add (see chapter 8.4)	
Syntax:	On = 1 Ea / Eb / Ec {/I}
Subtract (see chapter 8.5)	
Syntax:	On = 2 Ea / Eb / Ec {/I}
Multiply (see chapter 8.6)	
Syntax:	On = 3 Ea / Eb / Ec {/I}
Rotate a vector (see chapter 8.7)	
Syntax:	On = 4 Ea / Eb / Ec / I
Load a parameter with or without mask (see chapter 8.8)	
Syntax:	On = 5 Ea / Eb {/Ec} {/I}
Load a peripheral memory with or without mask (see chapter 8.9) Load an axis address (see chapter 8.9.1)	
Syntax:	On = 6 Ea / @.. {/Eb} {/I}
Load from a PLC Data (see chapter 8.9.2)	
Syntax:	On = 6 Ea / Eb {/Ec} {/I}
Indexed load of a parameter (see chapter 8.10)	
Syntax:	On = 7 Ea / Eb / Ec / I
Indexed storage of a parameter (see chapter 8.11)	
Syntax:	On = 8 Ea / Eb / Ec / I
Compare two parameters (see chapter 8.12)	
Syntax:	On = 9 Ea / Eb / Ia {/Ib {/Ic}}
Conditions on the sum of two parameters (see chapter 8.13)	
Syntax:	On = 10 Ea / Eb / Ec / I On = 10 Ea / Eb / Ec / E37xxx
Store data in a peripheral memory (see chapter 8.14) Storage at an axis address (see chapter 8.14.1)	
Syntax:	On = 11 @.. / E
Store in a PLC memory (see chapter 8.14.2)	
Syntax:	On = 11 Ea / Eb / I

Increment with modulo (see chapter 8.15)	
Syntax:	On = 12 Ea / Eb / I
Multiply by a program variable (see chapter 8.16)	
Syntax:	13 Ea {/Eb} / L..
Arc tangent function (see chapter 8.17)	
Syntax:	On = 14 Ea / Eb / Ec / I
Axes manual control (see chapter 8.18)	
Syntax:	15 @.. / E37xxx / Eb / I
Third-degree polynomial and derivative (see chapter 8.19)	
Syntax:	16 Ea / Eb / Lxx {/*}
Speed on trajectory (see chapter 8.20)	
Syntax:	17 Ea / @.. {/@@..} {/@@..}
Divide (see chapter 8.21)	
Syntax:	On = 18 Ea / Eb / Ec {/I}
Square root (see chapter 8.22)	
Syntax:	On = 19 Ea / Eb / I}
Axis speed synchronisation (see chapter 8.23)	
Syntax:	On = 20 @e / @m / E200xx / E37xxx / I
Oscillation cycles (see chapter 8.24)	
Syntax:	On = 21 @x = [TX(5,2)] / @y = [TY(5,2)] / @z = [TZ(5,2)] {/E37xxx}
Execute a C dynamic operator (see chapter 8.25)	
Syntax:	On = 22 COperateurName { / P ₁ / P ₂ / P ₃ / ... / P _n }
Electronic cam (see chapter 8.26)	
Syntax:	On = 23 @<axis> =<dim>/<dim>/<dim> @<axe> =<dim>/<dim>/.... @<axis> =<dim>/<dim> E37xxx
Differential shift (see chapter 8.27)	
Syntax:	On = 24 @a ₀ / @a ₁ / @ a _i / ... / Wm {/*}
Read current block data (see chapter 8.28)	
Syntax:	Oi= 25 Ea/Eb/#Symbol value/#Index value/Ec/I

A.8 Coordinate conversions

A.8.1 General

This chapter is for information only. NUM provides the tools to exploit this function easily and efficiently. Please refer to the chapter "Inclined plane and RTCP" in this manual.

Coordinate conversions are 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.

A.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).

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.

Cancellation

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

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

A.8.2.1 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**.



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-).

A.8.2.2 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.

A.9 RTCP function

A.9.1 General

This chapter is for information only. NUM provides the tools to program RTCP in an easy and efficient manner. Please refer to the chapter 9 Inclined plane and RTCP in this manual.

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.

A.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)

A.9.1.2 Control of rotary 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.

A.9.2 Programming the RTCP function ► G26

A.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.

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).

Cancellation

The RTCP function (G26+) is cancelled by:

- G26- (without arguments)
- A reset.

A.9.2.2 Programming errors

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

- G26- (without arguments)
- A reset.

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.

A.9.2.3 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.

A.9.2.4 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.

A.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.

A.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.

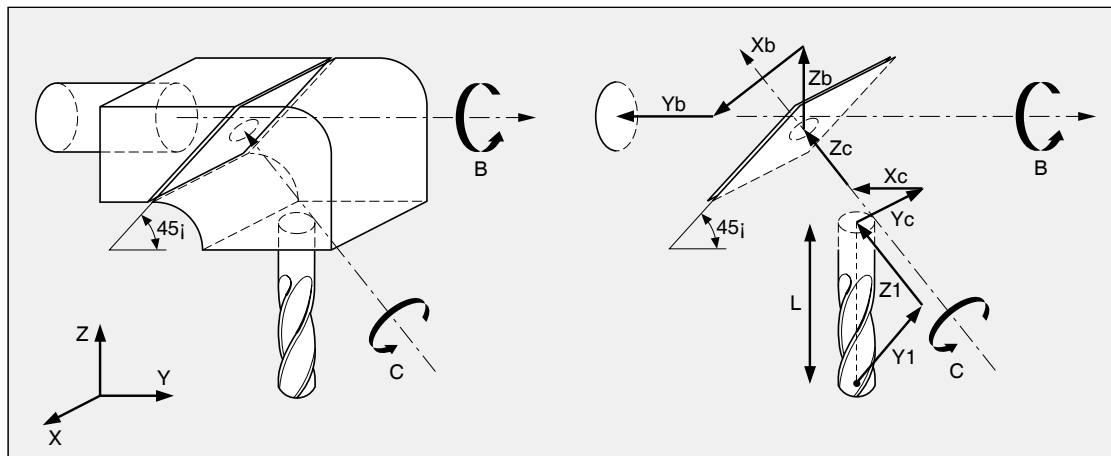
The axis of rotation is declared by TA, TB or TC followed the three associated translations each separated by the character «/».

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+).

A.9.3.2 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

A.9.3.3 Turntables configuration

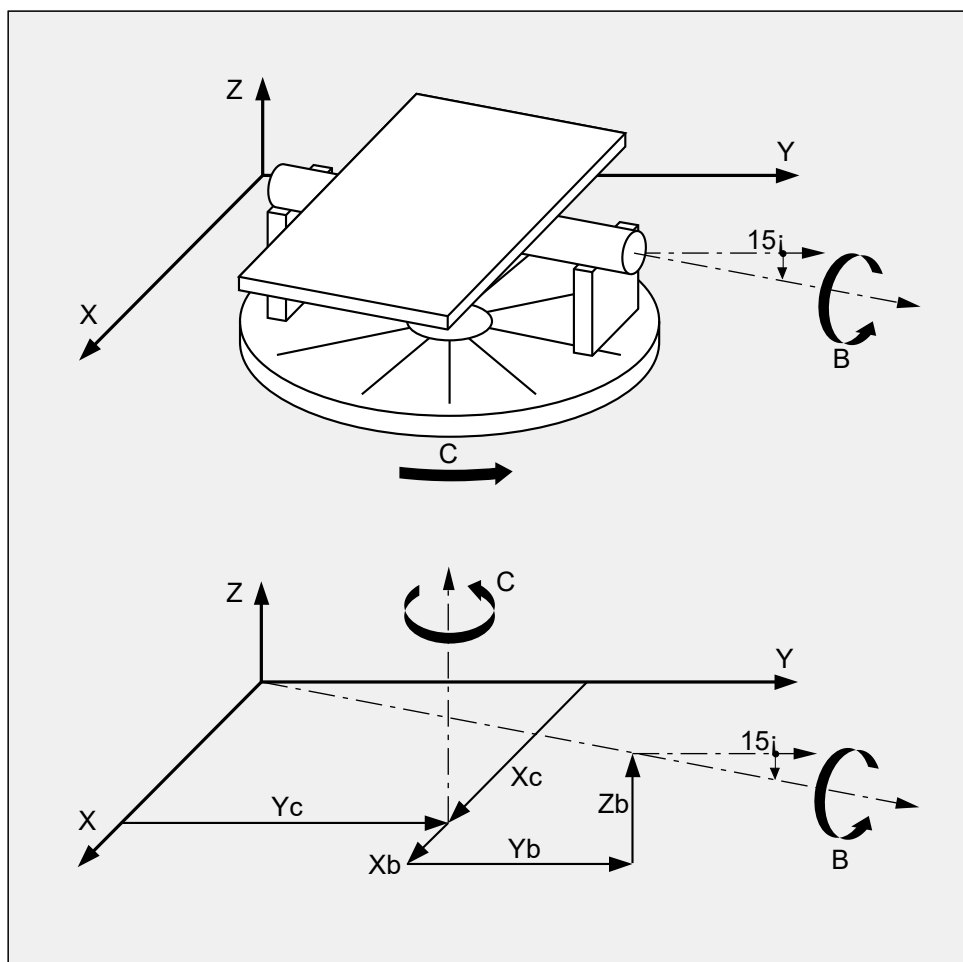
Each articulation is described by an axis of rotation and a translation vector.

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.

A.9.3.4 Representation of a turntable configuration



Programming

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

A.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.

A.9.4.1 Part on inclined plane

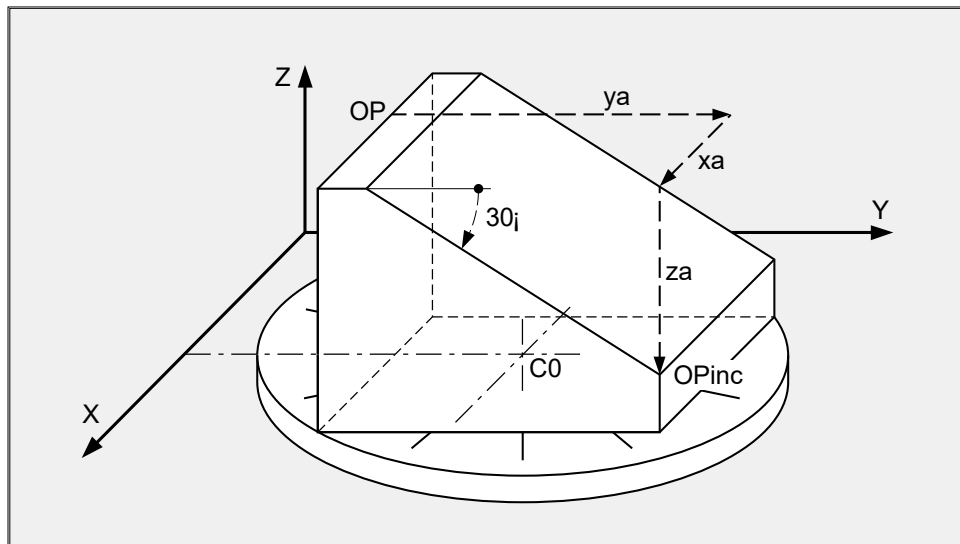
When the part to be machined rests on an inclined plane, this plane is specified by:

- 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]
```

A.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.

A.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»).

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-).

A.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»).

A.9.4.5 Tool correction (G29) : 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../... ..
```

A.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 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.

For E parameters detailed informations please refer to the Flexium⁺ Programming manual M00040.

Page deliberately empty

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

A

Page deliberately empty

Published by NUM Spa

Via F. Somma 62
20012 Cuggiono (MI) (Italy)
Tel. : +39 02 979 691
website: www.num.com

Printed October 2022