



# OEM Environment

## Programming Description (DP) User Documentation

Valid for:

Product: Kvara SW 162  
from version 3.14

Edition: June 2003  
Code: 91752.DP.1.GB

**Restrictions**

Duplication, transmission and use of this document or its contents shall be considered prohibited unless prior authorization has been received from Esa/Gv. All rights are reserved.

Even when authorized, modification of this document (either by computer or on paper) voids the guarantees specified below.

**Guarantees**

The product may offer performances that are not described in these manuals. Esa/Gv shall neither be obliged to maintain these functions in new versions of the product nor to guarantee the relative assistance.

Checks have been carried out in order to ensure that the contents of these manuals correspond to the documented product. Despite this fact, there may be discrepancies. Esa/Gv therefore offers no guarantees as to the full compliance and completeness of the texts.

The information in this document is periodically revised and new editions are issued when necessary.

This manual has been compiled in partial compliance with ANSI/IEEE std 1063-1987 "IEEE Standard for software User Documentation".

**Editions**

This document is liable to be modified without prior notice. These modifications may involve further editions or revisions of the document.

Further editions imply complete substitution of the document.

Revision involves replacement/addition/elimination of pages of the document.

Each page is identified by the code of the document at the bottom.

**Notes**

---

MS-DOS® Trademark registered by Microsoft Corporation.

---

The chronological list of editions of this document is given in the following table:

### **Evolution of the document**

<b>Edition</b>	<b>Document Code</b>	<b>Release</b>	<b>Type of edition</b>
19/03/2001	91752.DP.0.GB	0	New document
04/06/2003	91752.DP.1.GB	1	Revision

### **Modifications**

<b>Release</b>	<b>Chapters - Pages</b>	<b>Description</b>



# Programming Description

<b>Notes for the reader .....</b>	<b>i</b>
Explanation of the symbols.....	ii
Printer's conventions .....	iii
Glossary .....	iv
<b>1 Introduction .....</b>	<b>1.1</b>
1.1 Definitions .....	1.1
1.2 Conventions used in the description of the syntax .....	1.4
<b>2 Standard Programming .....</b>	<b>2.1</b>
2.1 Writing a program.....	2.1
2.1.1 An C program .....	2.1
2.1.2 Program start .....	2.1
2.1.3 Selecting ISO mode .....	2.2
2.1.4 Program end .....	2.2
2.1.5 Block number .....	2.3
2.1.6 Messages .....	2.3
2.2 Setting machining conditions.....	2.4
2.2.1 Defining a point .....	2.4
2.2.2 Interpolation parameters .....	2.4
2.2.3 Absolute point programming .....	2.5
2.2.4 Incremental programming .....	2.6
2.2.5 Selecting the unit of measurement.....	2.6
2.2.6 Selecting origin translation .....	2.7
2.2.7 Cancelling origin translation .....	2.8
2.2.8 Additive origin translation .....	2.9
2.2.9 Selecting the work plane .....	2.9
2.2.10 Selecting the contour milling plane and the length correction direction .....	2.11
2.3 Programming the feed.....	2.12
2.3.1 Feed command .....	2.12
2.3.2 Rapid feed.....	2.13
2.3.3 Linear interpolation.....	2.14
2.3.4 Circular interpolation .....	2.15
2.3.5 Thread-cutting .....	2.16
2.3.6 Tapping .....	2.18

2.3.7 Timed halt .....	2.19
2.3.8 Positioned halt.....	2.19
2.4 Special interpolations .....	2.20
2.4.1 Auxiliary axis interpolations .....	2.20
2.4.2 Cylindrical programming .....	2.20
2.5 Spindle orientation .....	2.22
2.5.1 M19: Spindle orientation.....	2.22
2.5.2 On-the-fly spindle orientation .....	2.23
2.6 Programming axis and spindle speed .....	2.24
2.6.1 Setting the feed rate .....	2.24
2.6.2 Continuous feed mode .....	2.25
2.6.3 Continuous feed mode with speed look-ahead .....	2.26
2.6.4 Selecting the feed mode .....	2.27
2.6.5 Selecting the spindle rotation mode .....	2.28
2.6.6 Setting the spindle rotation mode .....	2.28
2.6.7 Inverse feed .....	2.31
2.7 Polar axes programming .....	2.33
2.7.1 Polar axes with travel limitation .....	2.33
2.7.2 Round axes .....	2.33
2.7.3 Selection of the direction and rpm rate .....	2.35
2.7.4 Speed programming in the presence of polar axes .....	2.39
2.8 Auxiliary Functions.....	2.41
2.8.1 Functions M and T .....	2.41
2.9 Tool compensation .....	2.43
2.9.1 Tool length correction.....	2.43
2.9.2 Tool radius compensation .....	2.44
<b>3 Advanced Programming.....</b>	<b>3.1</b>
3.1 Selecting Automation Language mode.....	3.1
3.2 Parametric Programming .....	3.2
3.2.1 Global system variables .....	3.2
3.2.2 Pre-defined numeric variables .....	3.3
3.2.3 Assigning a variable pre-defined numeric variables.....	3.3
3.2.4 Symbolic variables .....	3.4
3.2.5 Declaration of Symbolic Variables.....	3.4
3.2.6 Allocation of Symbolic Variables .....	3.6
3.2.7 Expressions.....	3.7

3.2.8 Use of #QNAN .....	3.15
3.3 Program flow control.....	3.16
3.3.1 Unconditional jump.....	3.16
3.3.2 Test .....	3.16
3.3.3 Conditioned Test .....	3.17
3.3.4 WHILE Block .....	3.18
3.3.5 REPEAT Block .....	3.19
3.3.6 FOR Block.....	3.20
3.3.7 Cycle interruption - BREAK.....	3.21
3.3.8 Repeat.....	3.22
3.3.9 Nesting levels: sub-programs and fixed cycles .....	3.23
3.3.10 Subprogram start (fixed cycle) .....	3.24
3.3.11 Subprogram end (fixed cycle) .....	3.25
3.3.12 Program end .....	3.26
3.3.13 Call to subprogram CFU (user fixed cycle) .....	3.26
3.3.14 Subprogram call CFS (system fixed cycle) .....	3.27
3.3.15 Cancels a fixed cycle's mode .....	3.28
3.3.16 Call-back function in "C" language .....	3.29
3.3.17 Forced escape from the interpreter with error.....	3.30
3.4 Special functions .....	3.31
3.4.1 Await termination.....	3.31
3.4.2 Message display in phase with the executor.....	3.31
3.4.3 Memory read/write.....	3.31
3.4.4 Enter a physical or logical output .....	3.32
3.4.5 Enabling hydraulic tapping .....	3.33
3.4.6 Defining path axes .....	3.34
3.4.7 Definition of the Axes belonging to the Channel .....	3.35
3.4.8 Correction by zones .....	3.36
3.4.9 Enabling program simulation.....	3.37
3.4.10 Identifier of the number of the channel being executed .....	3.38
3.4.11 Probe target reading.....	3.39
3.4.12 Axis target reading .....	3.39
3.5 Reference system transformations.....	3.41
3.5.1 PRS - Preset machine origin.....	3.41
3.5.2 DEF - Run-time redefinition of the axis names .....	3.41
3.5.3 SYS – Run-time re-allocation of the axes name .....	3.42
3.5.4 MIR - Mirrored machining.....	3.42

3.5.5 ROT – Rotation of the contouring plane.....	3.45
3.5.6 SHF – Transfer of Origin .....	3.46
3.5.7 MLV - Selection of the matrix conversion level. ....	3.47
3.5.8 Definition of a tilting plane .....	3.49
3.5.9 Tilting plane enabling .....	3.50
3.6 Automatic plane geometry.....	3.51
3.6.1 Chamfer between two linear segments.....	3.51
3.6.2 Radius between two linear segments .....	3.52
3.7 Tool corrector auxiliary functions.....	3.53
3.7.1 Machining allowance management.....	3.53
3.7.2 Tangential feed in/out.....	3.53
3.7.3 Tool life and wear management.....	3.55
3.8 Three dimensional tool corrector .....	3.56
3.8.1 TWI: Spindle settings .....	3.56
3.8.2 THD: Tool holder settings .....	3.56
3.8.3 Conventions regarding angles of rotation .....	3.56
3.8.4 Orientation with polar axis coordinates .....	3.57
3.8.5 Orientation by Euler's Angles .....	3.57
3.8.6 Orientation by roll-pitch-yaw (RPY) .....	3.59
3.8.7 Orientation by the tool axis unit vector .....	3.60
3.8.8 Vectorial tool radius compensation .....	3.62
3.8.9 Automatic determination of the pertinent plane and machining with the axis in tangency .....	3.63
3.8.10 Discriminating between the interchangeable cutting edges of a tool .....	3.67
3.8.11 Machining depth compensation along the tool axis .....	3.69
3.8.12 Machining depth compensation along the tool radius .....	3.70
3.9 Feed in strategies.....	3.71
3.9.1 Definitions.....	3.71
3.9.2 Axial/radial feed in .....	3.71
3.9.3 Axial/radial feed out.....	3.74
3.9.4 Compensation depending on a section of the tool along the trajectory in its in-going point .....	3.78
3.9.5 Compensation depending on a section of the tool along the trajectory in its out-going point .....	3.82
3.10 Feed Control.....	3.83
3.10.1 Modification or Automatic speed control parameters .....	3.83
3.11 Independent axis synchronous/asynchronous positioning .....	3.85
3.11.1 Independent axis synchronous positioning .....	3.85



3.11.2 Independent axis asynchronous positioning .....	3.86
3.11.3 Parallel spindle orientation .....	3.87
3.12 Modifying axis parameters.....	3.88
3.12.1 Following error.....	3.88
3.12.2 Axis acceleration time .....	3.88
3.12.3 Axis deceleration time .....	3.89
3.12.4 Axis emergency deceleration time .....	3.89
3.12.5 Axis maximum speed.....	3.90
3.12.6 Axis ramp to S.....	3.91
3.12.7 Axis jerk settings .....	3.92
3.13 Modification of the interpolation parameters .....	3.93
3.13.1 Interpolation acceleration time setting.....	3.93
3.13.2 Interpolation deceleration time setting .....	3.93
3.13.3 Maximum interpolation speed setting.....	3.94
3.13.4 Interpolation 'S' ramp time setting.....	3.94
3.13.5 Jerk setting during interpolation .....	3.95
<b>4 Fixed Cycle Systems (FCS).....</b>	<b>4.1</b>
4.1 G192 macro Generation of a grid of points .....	4.2
4.2 G193 macro - generation of points distributed along a circumference arc.....	4.5
4.2.1 G81 Fixed Cycle: drilling .....	4.7
4.2.2 G83 Fixed Cycle: drilling with swarf discharge or breaking.....	4.9
4.2.3 G84 Fixed Cycle: Tapping.....	4.12
4.2.4 G86 Fixed Cycle: Boring .....	4.13
4.2.5 G88 Fixed Cycle: drilling cavity walls .....	4.14
4.2.6 G133 Fixed cycle: Thread-cutting .....	4.16
<b>5 Tool parameters .....</b>	<b>5.1</b>
5.1 Definitions .....	5.1
5.2 ER ( <i>Entity Relationship</i> ) diagram of the entities of a machining center of which the geometry and power train are controlled.....	5.3
5.2.1 Key .....	5.3
5.2.2 Diagram.....	5.3
5.3 Other definitions .....	5.5
5.4 Head descriptor .....	5.6
5.4.1 Data structure.....	5.6
5.4.2 Description of the geometry and actuators .....	5.7
5.5 Toolholder descriptor.....	5.13

5.5.1 Data structure.....	5.13
5.5.2 Description of geometry .....	5.13
5.6 Tool or tool cutter descriptor.....	5.15
5.6.1 Data structure.....	5.15
5.6.2 Description of geometry .....	5.15
5.6.3 Description of other technological data .....	5.17
5.6.4 Logic management.....	5.18
5.6.5 Broken tools .....	5.18

## END OF SUMMARY

# Notes for the reader

<b>General information</b>	<p>The information in this manual only applies to the software versions indicated on the frontispiece.</p> <p>Not all the available functions may be described in this manual. In these cases, Esa/Gv shall be obliged to neither guarantee these functions nor include them in future versions.</p>
<b>Purpose</b>	<p>The purpose of this document is to help the operator when programming the processes.</p>
<b>Users</b>	<p>This document contains information for:</p> <ul style="list-style-type: none"><li>• technicians with a good working knowledge of the processing technology and productive process. Basic knowledge of computer work.</li></ul>
<b>Use of the document</b>	<p>The document is divided into chapters that describe the programming operations, the language and basic processing concepts.</p>
<b>Notification of difficulties</b>	<p>Please contact Esa/Gv if any difficulties should arise when this manual is used.</p>

## Explanation of the symbols

Graphic symbols may appear beside the text. These are used to emphasize information of particular importance.



---

**Attention**

This symbol is used when failure to take the appropriate precautions **could cause slight damage to persons and property.**

---



---

**Danger**

This symbol appears when failure to take the appropriate precautions or accomplishment of incorrect manoeuvres **could cause serious damage to persons and/or property.**

---



---

**Important**

This symbol appears in the manual to indicate information of particular importance. It is essential to read these sections in order to fully understand the manual.

---



---

**Option**

This symbol indicates sections of the manual that describe optional functions or parts. Use of optional performances must be established with the machine manufacturer.

---



---

**Manufacturer**

This symbol indicates those sections of the manual reserved to the machine manufacturer.

---



---

**Password**

This symbol indicates sections of the manual that describe functions access to which is safeguarded by software passwords.

---



---

**CN**

This symbol indicates sections of the manual that describe functions only available in CN and not in the PC.

---



---

**PC**

This symbol indicates sections of the manual that describe functions only available in the PC and not in CN.

---

## Printer's conventions

Particular printer's conventions are used to make it easier to identify the information in this manual. These conventions are illustrated below.

### Keyboard and video

The following conventions are used.

- The names of the screen-printed keys are indicated in **boldface** and are enclosed within square brackets. If the name of the key is preceded by "button", reference is being made to a key on the push button panel.
  - **[ENTER]**. Identifies the key that bears the word ENTER.
  - **[+]** indicates the + key of the keyboard, while button **[+]** indicates the + key of the push button panel.
- The names of the function keys are indicated in ***boldface italics*** and are enclosed within square brackets.
  - **[Plc Menu]**. Identifies the function key that bears the words Plc Menu.
- References to fields and/or messages on the video are written in ***boldface italics***.
- The specific text to be digitized by the user is underlined.
  - If the manual indicates "digitize ok", the user must digitize exactly "ok".
- DIRECTION or DIRECTIONAL keys is the collective name used to indicate the UP, DOWN, LEFT and RIGHT keys.
- Pressure, in sequence, on a series of keys is written by separating the identifiers of the required keys with the ">" character.
  - **[Manual] > [START]**. Describes pressure, in sequence, on the **[Manual]** and **[START]** keys.
- Pressure on several keys at the same time is indicated by separating the identifiers of the keys themselves with the "+" character.
  - **[SHIFT] + [→]** Describes contemporaneous pressure on the **[SHIFT]** and **[→]** keys.

### Text

The following conventions are used.

- *Italics* are used to identify specialistic terms.
- **Boldface** is used to emphasize words of particular importance.

## Glossary

### CNC

This is an abbreviation of *Computerized Numerical Control* and indicates the instrument that governs the machine, i.e. the electronic device through which the machining cycles are programmed, the axes moved, etc..  
It corresponds to one of the devices whose operation is described in this manual.

**END OF PREFACE**

# 1 Introduction

This chapter describes certain fundamental concepts that are at the basis of CNC programming.

Examples containing instructions in the programming language will be used to highlight certain aspects, while the explanation of each instruction is given in the dedicated chapter.

## 1.1 Definitions

**Program** A program is a sequence of *blocks*, or program lines, that specify the various machining phases.

**Block** A block consists of one or more words and terminates with the *line feed* character ([RETURN] key)

**Word** A word consists of a literal code and a number, not separated by spaces.

Example of a block formed by three words:

N100 X100.00 Y100.00

Characters *N*, *X* and *Y* identify the type of words in the block.

Example:

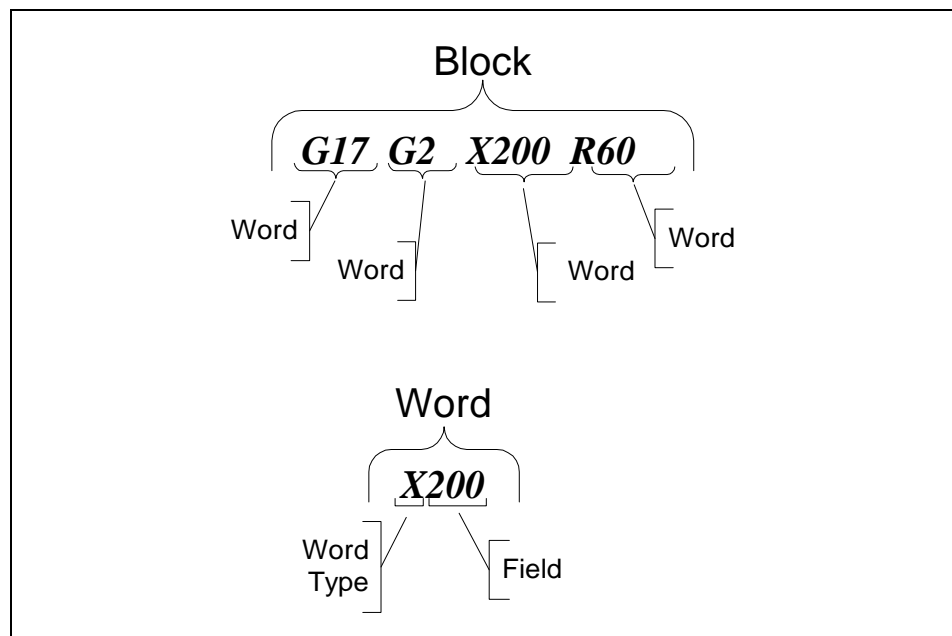


Figure 1.1 - Composition of a Block and a Word



**Do not attempt to execute** the sequence of commands in the previous example on the machine. All the data specified are purely fictitious.

### Program execution

Program execution is the same as executing the blocks that form it in the established order in order to obtain one or more machining operations.

### Execution flow

The execution flow is the sequence with which the instructions in a program or list are executed. In the more simple cases, execution takes place in sequence starting from the first instruction and terminating with the last (sequential flow). In other cases, the flow can be altered, repeating or conditioning the execution of certain instructions (non-sequential flow).

### Axis

An axis is a mechanical and electronic entity thanks to which the NC can control the position of an element of the machine in relation to a reference point.

Three axes square to each other are required too identify a point in the three-dimensional space. They are conventionally called X, Y and Z (main axes).

Z axis is generally parallel to the axis of rotation of the spindle.

The following axes can be defined:

<b>X</b>	Main axis
<b>Y</b>	Main axis
<b>Z</b>	Main axis
<b>U</b>	Auxiliary axis parallel to X
<b>V</b>	Auxiliary axis parallel to Y
<b>W</b>	Auxiliary axis parallel to Z
<b>A</b>	Rotational axis with axis of rotation parallel to X
<b>B</b>	Rotational axis with axis of rotation parallel to Y
<b>C</b>	Rotational axis with axis of rotation parallel to Z

The target of an axis always refers to a point (origin) and can be expressed in:

- Millimeters or inches for linear axes
- degrees for rotational axes

### Origin

An origin is a point in space to which the axes targets refer.

After the sizing cycle, which normally takes place whenever the CNC is powered, the axes are referred to the *machine origin*. However, when it comes to workpiece programming, the targets can be referred to the *workpiece origin* which will be shifted in relation to the machine origin by a value in the *origin* parameters, depending on the position in which the workpiece clamping system is to be found.



**Modal function**

A programming function (or address) is called *modal* if its effect also affects the successive blocks, through to activation of another function that excludes the first.

A non-modal function, that only has effect in the block in which it is programmed, is also called *self-cancelling*.

**Default function**

A modal programming function is *by default* if it is activated automatically at the start of the program.

## 1.2 Conventions used in the description of the syntax

Recourse is made to a conventional notation in the description of the syntax of each instruction:

**Capital text, for example: Y, R, I, J, ...**

Indicates the *name of an address* within a command.

**Target**

Indicates a field in which the following information can be entered:

- A numeric value with sign and decimals used to indicate any coordinate or distance. When the parameter that indicates the display unit is varied, the measurements will be automatically converted from millimeters into inches, and vice versa. For inches, two decimal figures are added as compared to the measurements expressed in millimeters;
- A *variable* (see “Programming concepts”, Parametric Programming” section).

**Angle**

Indicates a field in which the following information can be entered:

- An angle, expressed in degrees/decimal fractions of a degree (from 0.00 to 359.99). The sexagesimal system cannot be used. Values of 360 or more or negative values will be normalized, i.e. they will be brought within the 0 to 359.99 range.
- A variable.

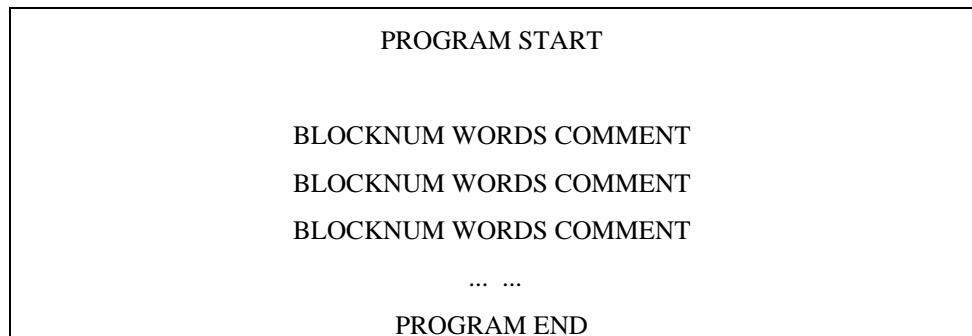
**END OF CHAPTER**

## 2 Standard Programming

### 2.1 Writing a program

#### 2.1.1 An C program

A NC program or partprogram is a sequence of blocks which define the machining of a workpiece (part) on a numerical control machine tool.



#### 2.1.2 Program start

##### Syntax

comment %program\_number

##### Description

The program start block must precede all other blocks of the self program.

Comment

An alphanumerical character string which briefly describes the program.

È a comment must be inserted before the character %.

program\_number

A unique identifying number which corresponds to the partprogram number.

##### Example

```
THREAD 3/8 GAS %1200
....
....
program body
....
....
M30
```

##### See also

**Program end, Subprogram start ( "Advanced programming").**

### 2.1.3 Selecting ISO mode

**Syntax**                   MODE = ISO  
                              MODE = AUTO (*default*)

**Description**             Assign the value ISO to the system variable MODE to select writing a partprogram according to ISO/DIS 6983 (Parts I and II) Draft International Standard, September 1982 (Part I) and July 1988 (Part II).  
In this mode Automation Language-specific features are not available.  
In ISO mode comments are preceded by round brackets and an asterisk.

**Example**

```
( * INSERT WORKING PROGRAM)
N5 G17 ( * WORK PLANE X-Y)
N15 G0 B90 ( * TURN CNC TABLE 90 DEGREES)
N20 T1 M6 (*RECALLS TOOL/SPINDLE )
(*TOOL DIAMETER 10 )
N30 D1 (*TOOL CORRECTOR)
N40 G0 X0 Y0
N50 Z125
```

### 2.1.4 Program end

**Syntax**                   *M2* or *M30*

**Description**             The program end block stops execution of the program.

**Example**

```
THREAD 3/8 GAS %1200
....
....
program body
....
....
M30
```

**See also**                **Program start.**

### 2.1.5 Block number

**Syntax**                      Nblock\_number

**Description**                      The block number is an optional word which must precede the other words in the block. It gives a unique reference to the block itself.

block\_number                      Unique block reference number.

**Example**

```
COMMENT %1
N10 X100 Y100 (feed to point P1(X100,Y100))
N20 Y200 (feed to point P2(X100,Y200))
G04 F1
....
....
M30
```

### 2.1.6 Messages

**Syntax**                      (text)

**Description**                      In ISO programming mode, text in round brackets () is displayed as

text                                      A sequence of alphanumerical characters.

**Example**

```
N100 (ERROR: PRESS RESET)
N110 G04 F1
N120 JMP 110 (loop while awaiting reset)
```

## 2.2 Setting machining conditions

### 2.2.1 Defining a point

**Syntax** *Xpos Ypos Zpos Upos Vpos Wpos Apos Apos Bpos Cpos*

**Description** An axis code followed by a number defines a point's position on the axis.  
To define a point the words for all the defined axes must be contained in a single block.  
If a word relative to a given axis is missing, the system sets that axis to its value in the definition of the preceding point.

*Xpos*

*Ypos*

*Zpos*

*Upos*

*Vpos*

*Wpos*

*Apos*

*Bpos*

*Cpos*

**Example**

X100 Y100 (feed to point P1(X100,Y100))  
Y200 (feed to point P2(X100,Y200))

**See also** **Feed command.**

### 2.2.2 Interpolation parameters

**Syntax** *Ipos* (value on the X axis)  
*Jpos* (value on the Y axis)  
*Kpos* (value on the Z axis)  
*Rpos* (radius of a circle)

**Description** Interpolation parameters define the basis for interpolation.  
The codes I, J and K are used to define the coordinates of the centre of a circular interpolation and the pitch in thread-cutting and tapping.  
The code R defines the radius of a circular interpolation arc.

*I*pos

*J*pos

*K*pos

*R*pos

**Example** See the examples under **Circular interpolation, Thread cutting, Tapping.**

**See also** **Circular interpolation, Thread cutting, Tapping**

### 2.2.3 Absolute point programming

**Syntax** *G90*

**Type of function** Modal, default, mutually exclusive with *G91*

**Description** Absolute programming defines the points relative to the axis origins as defined at the time.

**Example**

```
G90 X200 Y300 (feed to point P1(X200,Y300)
      Y700 (feed to point P2(X200,Y700)
      X1000 (feed to point P3(X1000,Y700)
      Y300 (feed to point P4(X1000,Y300)
      X200 (feed to point P4(X200,Y300)
```

**See also** **Defining a point, Incremental programming.**

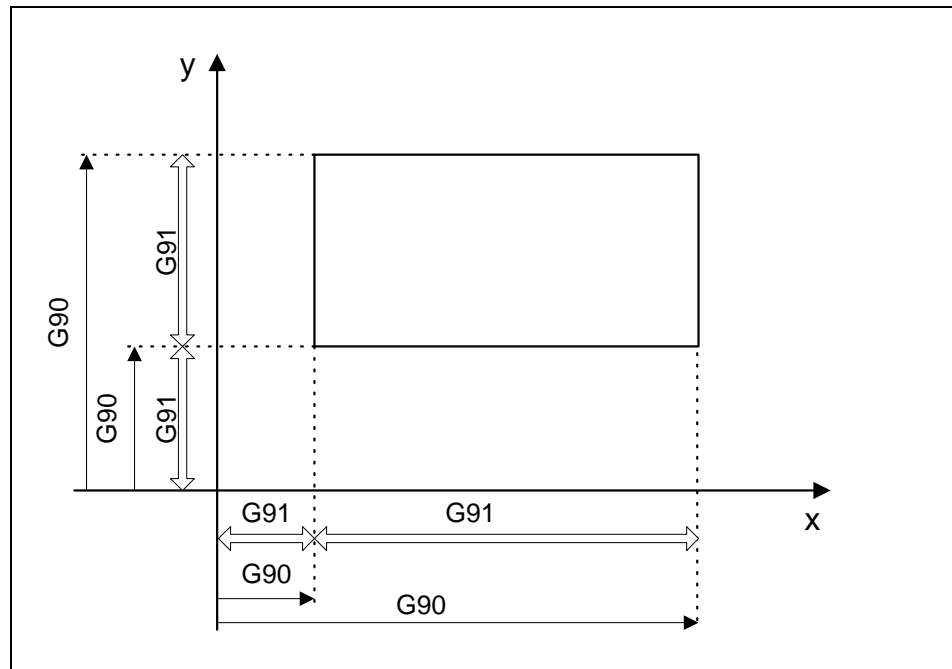


Figure 2.1 - Incremental and absolute programming

## 2.2.4 Incremental programming

**Syntax** *G91*

**Type of function** Modal, mutually exclusive with *G90*

**Description** Incremental programming defines the points relative to the latest preceding point. The I, J and K positions are also relative to the preceding values of the X, Y and Z axes.

### Example

```
X100 Y100 feed to point P1(X200,Y300)
G91 Y100 feed to point P2(X100,Y200)
X100 feed to point P3(X200,Y200)
Y-100 feed to point P4(X200,Y100)
```

**See also** Defining a point, Absolute point programming.

## 2.2.5 Selecting the unit of measurement

**Syntax** *G70* (dimension in inches)  
*G71* (dimension in mm)



<b>Type of function</b>	Modal, default, settable by a machine parameter
<b>Description</b>	Selects programming in imperial or metric dimensions. All axes are thus measured in inches (values after axis codes) except for the rotary axes (A,B,C), the dimensions which define the centre of a circle, and the radius dimension.
<b>Example</b>	<pre>X100 Y100 (feed to point P1(X100,Y100)) G70 G91 Y3 (feed to point P2(X100,Y100+3*25.4))</pre>
<b>See also</b>	<b>Defining a point.</b>

## 2.2.6 Selecting origin translation

**Syntax**

*G54* to select translation of origin 1  
*G55* to select translation of origin 2  
*G56* to select translation of origin 3  
*G57* to select translation of origin 4

**Type of function** Modal, mutually exclusive with *G53*

**Description** Origin translation allows selection of one of the four *part origins* set by the operator via the user interface. In the blocks after the origin selection block, the positions and tool movements refer to the currently selected origin. This function only works in cartesian coordinates XYZ.

**Example**

```
X100 Y100 (feed to point P1(X100,Y100))  
G54 Y200 (feed to point P2(Xinalterato,Y200) relative to  
part origin 1)
```

**See also** Defining a point, Cancelling origin translation.

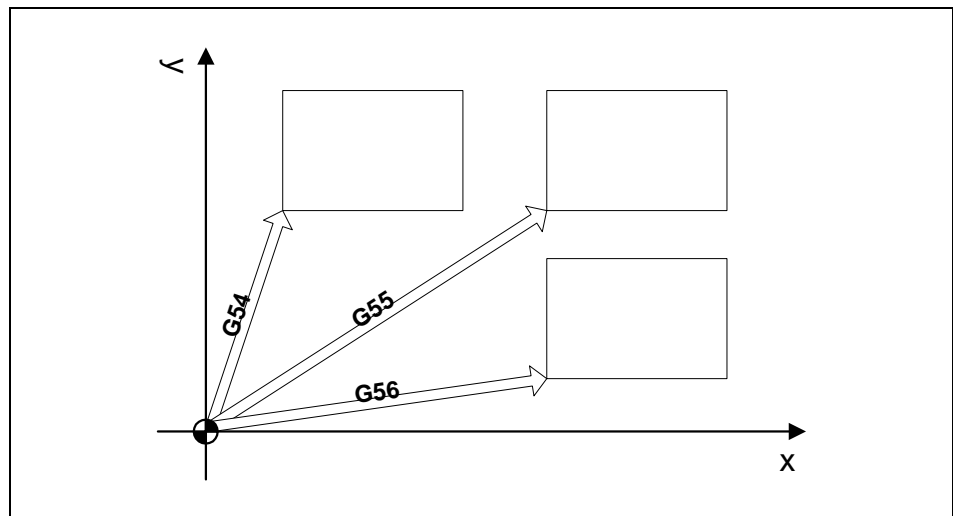


Figure 2.2 - Programming the part origins

## 2.2.7 Cancelling origin translation

**Syntax** *G53*

**Type of function** Self-cancelling, default, mutually exclusive with *G54*, *G55*, *G56*, *G57*

**Description** Inclusion of this function in a block switches off the "origin translation" and "tool compensation" functions. The programmed dimensions are therefore referred to the *machine origin*.  
This function is used for positioning the tool with respect to the machine origin, irrespective of the part origin (for example, for returning to the tool magazine).

**Example**

```
G54 X100 Y100 (feed to point P1(X100,Y100) relative to the
part origin)
G53 X1000 Y2000 (feed to point P2(X1000,Y2000) relative to
the machine origin)
X100 Y200 (feed to point P3(X100,Y200) relative to the
part origin)
```

**See also** **Defining a point, Selecting origin translation.**

### 2.2.8 Additive origin translation

<b>Syntax</b>	<p><i>G58 Xpos Ypos</i> .....for additive translation of origin 1</p> <p><i>G59 Xpos Ypos</i> .....for additive translation of origin 2</p>
<b>Type of function</b>	<p>Modal</p> <p><i>Xpos</i></p> <p><i>Ypos</i></p> <p><i>Zpos</i></p>
<b>Description</b>	<p>Additive origin translation adds origin translation dimensions to the part origin, for defining local origins on the part as required.</p> <p>The translation values are defined for the primary axes (XYZ) or the auxiliary axes (UVW), by the values set in the same block.</p> <p>To cancel additive origin translation, set the translation values to zero.</p>
<b>Example</b>	<pre> X100 Y100    (feed to point P1(X100,Y100)) G58 Y50      (move the origin to P2(X0, Y50)) X100 Y100    (feed to point P3(Xunchanged,Y+50)) G58 Y0       (reset the part origin) </pre>
<b>See also</b>	<b>Defining a point, Selecting origin translation.</b>

### 2.2.9 Selecting the work plane

<b>Syntax</b>	<p><i>G17</i> (select XY plane, compensate tool length on Z)</p> <p><i>G18</i> (select ZX plane, compensate tool length on Y)</p> <p><i>G19</i> (select YZ plane, compensate tool length on X)</p>
<b>Type of function</b>	<p>Modal.</p> <p>Default: G18 if the Y axis does not exist (lathe), otherwise G17.</p>
<b>Description</b>	<p>Selecting the work plane defines:</p> <ul style="list-style-type: none"> <li>• the plane on which the arc is executed in a circular interpolation</li> <li>• the axis along which the tool is compensated</li> <li>• the plane for tool radius correction</li> </ul>

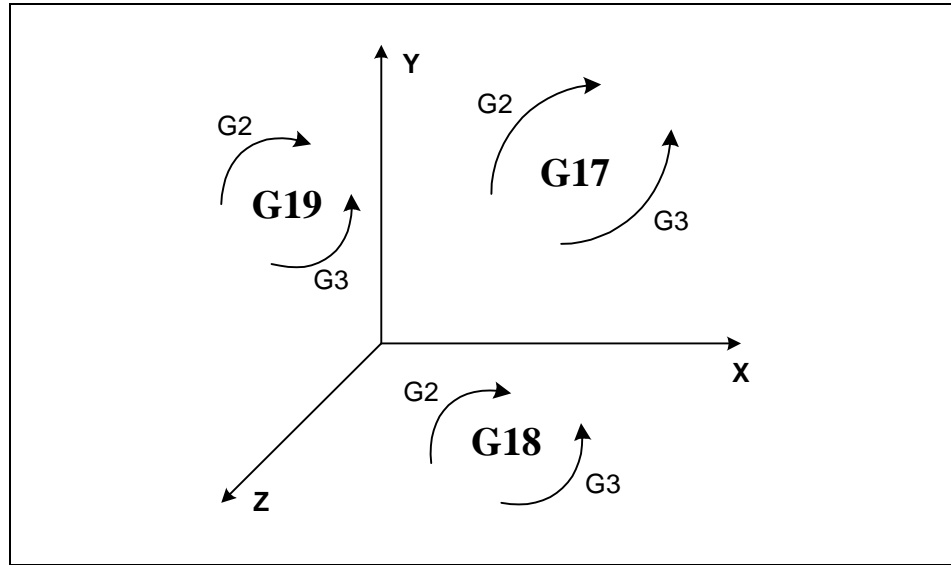


Figure 2.3 - Work plane

#### Example

```
X100 Y100 (feed to point P1(X100,Y100))  
G17 G2 X200 R60 (executes an arc on the XY plane)
```

#### See also

**Circular interpolation, Tool compensation.**

## 2.2.10 Selecting the contour milling plane and the length correction direction

**Syntax** *G16 <name axis I> <name axis II> <name axis III> <direction of length correction>*

**Type of function** Modal.

**Description** Selects the contour milling plane(or where the tool radius correction is applied) by selection of the first and second axis. Any contour milling plane can be specified. Furthermore, G16 enables selection of the direction and axis of length correction via specification of the third axis and sign ('+' / '-'). The third axis can be freely chosen from among the configured cartesian axes. Tool correction cannot therefore be applied directly along an auxiliary (UVW) or polar (ABC) axis.

G16 has the following features:

- definition of the radius correction plane (which is also the machining plane).
- definition of the axis and direction of length correction;

G17 → G16 XYZ+

G18 → G16 ZXY+

G19 → G16 YZX+

**Example**

G16 U Z Y+

## 2.3 Programming the feed

### 2.3.1 Feed command

**Syntax** *Xpos Ypos Zpos Upos Vpos Wpos Apos Apos Bpos Cpos*

**Description** The machine axes are moved to the desired point by simply specifying the point of arrival.  
The way in which the feed is executed depends on the selections made.

*X, Y, Z, U, V, W pos* Linear coordinate (in mm) of the point of arrival.

*A, B, C pos* Angular coordinate (in degrees) of the point of arrival.

**Example**

```
X100 Y100 (feed to point P1(X100,Y100))  
Y200 (feed to point P2(X100,Y200))
```

**See also** **Defining a point, Rapid feed.**

### 2.3.2 Rapid feed

**Syntax** *G0*

**Type of function** Modal, default, mutually exclusive with *G1, G2, G3, G33, G63*

**Description** Rapid feed mode moves the machine's axes to the point of arrival in the shortest time possible within the limits of the machine's operational parameters.  
Rapid feed is used for moving to the machining start point, tool change positions, etc.  
The feed speed set with the F parameter is overridden.  
If ***Rapid feed with independent axes*** ( machine parameters) is enabled, each axis moves autonomously at its maximum speed. If this option is not enabled, the axes move in interpolated mode at the maximum speed, within the limits of the various axes.

Axes with finite travel are handled without modular arithmetic.

For modular axes (infinite travel) the following considerations apply:

- the target dimension (modulo 360), determines the angle to be reached;
- the direction is set so as to reach the target with the shortest possible excursion (max. 180°); if the target and the previous dimension are equal, the axis remains stationary; if the difference is 180° the direction is conventionally set to avoid passing through the origin; if the target is 180° and the previous dimension is 0°, or vice versa, the axis passes through 90°;
- To force the direction of travel add or subtract 360°;
- To rotate the axis several times, add or subtract 360° X (number of full turns + 1). To rotate the axis once, simply add or subtract 720° to the current position of the axis.

#### Example

```
G0 X100 Y100 (rapid feed to point P1(X100,Y100))
Y200 (rapid feed to point P2(X100,Y200))

G0 A120.000
G0 A120.000 (* axis stationary at 120 *)
G0 A120.000
G0 A200.000 (* advance axis to 200, through 80 *)
G0 A120.000
G0 A40.000 (* reverse axis to 40, through 80 *)
G0 A40.000
G0 A320.000 (* reverse axis to 320, through 80 *)
G0 A320.000
```

**See also** **Feed command, Linear interpolation.**



### 2.3.3 Linear interpolation

**Syntax** *G1*

**Type of function** Modal, mutually exclusive with *G0*, *G2*, *G3*, *G33*, *G63*

**Description** Linear interpolation mode enables travel to the target point (with the fine tolerance threshold) via a linear trajectory at the speed set with parameter **F**.  
Whenever rotary axes are involved in the interpolation, as these cannot follow a linear trajectory, they are set to start and terminate their travel at the same time as the linear axes.  
Axes with finite travel are handled without modular arithmetic.  
For modular axes (infinite travel) the following considerations apply:

- the target dimension (modulo 360) determines the angle to be reached;
- the direction to travel is determined by the relative values of the original and target dimensions, including sign; if the target and the previous dimension are equal, the axis remains stationary;
- To rotate the axis several times, add or subtract 360° X (number of full turns). To rotate the axis once, simply add or subtract 360° to the current position of the axis.

**Example**

```
G0 X100 Y100 (rapid feed to point P1(X100,Y100))
G1 X200 Y200 F1000 (feed to point P2(X100,Y200) with linear
travel at 1000 mm/min)

G0 A120.000
G1 A120.000 (* axis stationary at 120 *)
G1 A200.000 (* advance axis to 200 *)
G0 A120.000
G1 A-160.000 (* reverse axis to 200 *)
G0 A120.000
G1 A60.000 (* reverse axis to 60 *)
G0 A120.000
G1 A420.000 (* advance axis to 60 *)

G0 A0.000
G0 A120.000
G1 A480.000 (* rotate axis forwards one full turn and on to
120 *)
G0 A120.000
```

**See also** **Feed command, Rapid feed, Circular interpolation, Setting the feed rate.**

### 2.3.4 Circular interpolation

**Syntax**                    *G2* (clockwise)  
                               *G3* (counterclockwise)

**Type of function**        Modal, mutually exclusive with *G0*, *G1*, *G33*, *G63*

**Description**            Circular interpolation drives the axes to the target point (within the fine tolerance threshold) through a circular arc in the working plane at the speed set with parameter **F**. The arc is defined by the target point and the centre coordinates (codes **I**, **J** and **K**) or by the radius (code **R**).  
 The centre coordinates can be specified either absolutely or incrementally relative to the starting point of the arc, depending on which mode is selected.  
 If the target point is set outside the plane of the arc, the actual path followed by the axes becomes helical.  
 When programming the radius, an arc of angle greater than 180° must be specified with a negative sign.  
 G2-G3 cannot be used with more than 4 axes currently selected as feed axes.

#### Example

```
G0 X0.0 Y0.0 (rapid feed to point P1(X0,Y0))
G2 F100 X100 Y0 I50 J0 (Clockwise circular arc to point
P2(X100, Y0) with centre C(X50, Y0))
G3 X0 Y0 R50 (Rcircular arc as above but
counterclockwise and defining the
radius (50) rather than the centre)
G2 X100 Y0 Z100 I50 J0 (Same arc, with change of plane
(helical))
```

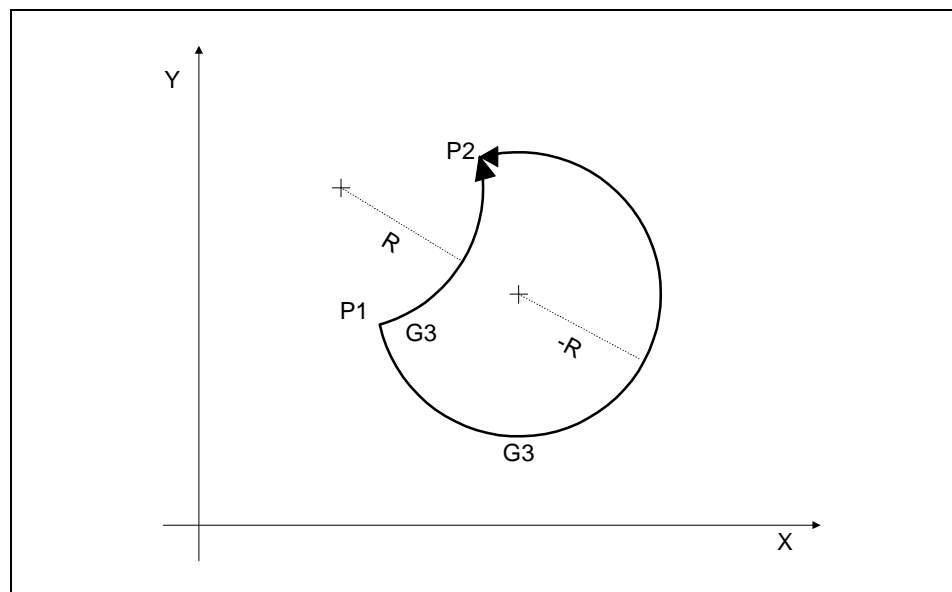


Figure 2.4 - Circular interpolation with positive/negative radius

**See also**                      **Interpolation parameters, Feed command, Selecting the work plane.**

### 2.3.5 Thread-cutting

**Syntax**                      *G33 Zpos Kpitch* (cylindrical)  
                                  *G33 Zpos Xpos Kpitch* (conical)

**Type of function**        Modal, mutually exclusive with *G0, G1, G2, G3, G63*

**Description**              Cuts a cylindrical, conical or planar thread from the current position to the programmed target point, with the programmed pitch.  
                                  Not possible if one of the following functions is enabled: G95, G96.  
                                  During thread-cutting the spindle must be fitted with a position transducer.  
                                  The feed speed of each axis depends on the spindle speed (set by parameter S) and the thread pitch.  
                                  The spindle's starting angle for thread-cutting is always zero.

*Zpos*                              final longitudinal point

*Kpitch*                            constant thread pitch

*Xpos*                              final radial point

#### Example

```
S200 M3      (start spindle)
G0 X20 Z100  (approach to workpiece)
G33 Z70 K2   (cylindrical thread depth 30 mm with pitch
2)
G0 X30      (withdraw from piece)
```

**See also**                      **Interpolations parameters, Tapping.**

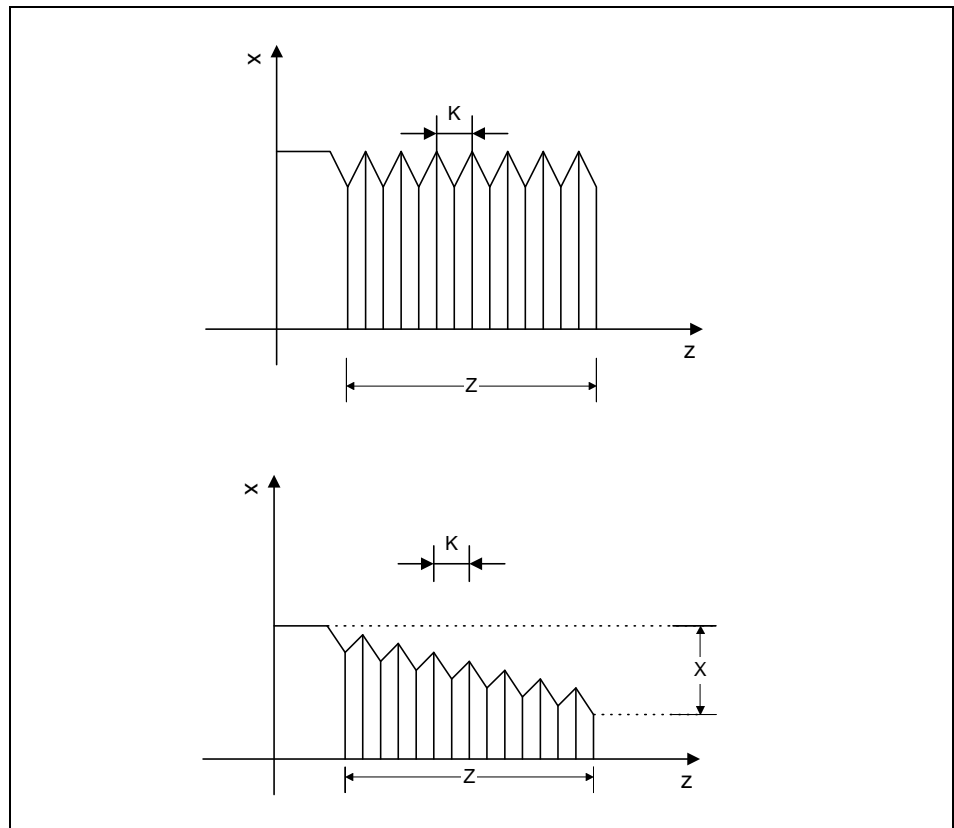


Figure 2.5 - Geometric thread-cutting parameters G33

### 2.3.6 Tapping

**Syntax** *G63 Zpos Kpitch*

**Type of function** Modal, mutually exclusive with *G0, G1, G2, G3, G33*

**Description** Executes rigid tapping from current to programmed target point, with programmed pitch. During tapping the spindle must be fitted with a position transducer. The spindle speed is set by the parameter S. The longitudinal feed speed depends on the spindle speed and the tapping pitch. If the pitch is positive the spindle rotates clockwise for increasing Z and counterclockwise for decreasing Z (righthand thread); vice versa for negative pitch.

*Zpos* end point

*Kpitch*

**Example**

```
M19      (orient spindle)
S200     (select spindle speed)
G0 Z100   (approach to workpiece)
G63 Z70 K-2 (righthand tapping depth 30 mm with pitch 2)
G4 F0.2   (halt for 2 tenths)
G63 Z100 K-2 S1000 (extract tool)
G0 Z200   (withdraw from workpiece)
```

**See also** **Interpolation parameters, Thread cutting.**

### 2.3.7 Timed halt

**Syntax** *G4 Ftime*

**Type of function** Self-cancelling, mutually exclusive with *G0, G1, G2, G3, G33, G63*

**Description** The timed halt function stops the machine for the time set in the parameter F.

*Ftime* [sec]

**Example**

```
G4 F2.5 (halt for 2.5 seconds)
X100 Y200 (feed to point P2(X100,Y200))
```

**See also** **Positioned halt.**

### 2.3.8 Positioned halt

**Syntax** *G9*

**Type of function** Self-cancelling

**Description** The positioned halt function forces stopping at the programmed point within the fine tolerance threshold (set in the machine parameters) in rapid feed mode, in which the rough tolerance would normally apply to allow passage to the next block.

**Example**

```
G0 G9 X100 Y200 (feed to point P1(X100,Y200) with fine
tolerance)
```

**See also** **Rapid feed, Timed halt.**

## 2.4 Special interpolations

### 2.4.1 Auxiliary axis interpolations

#### Descriptin

Interpolations can be programmed using the auxiliary axes UVW just as with the cartesian axes XYZ (see G0, G1, G2, G3, G33, G63, G133).

Just as with the cartesian parameters, we can also apply tool radius and length corrections, machining allowance (SVR/SVL), inverse feed programming (G93) and selection of the system of reference (G58-G59).

To decide which axes are involved in the tool radius correction, for example, the system first checks for the cartesian axes of the selected plane and if they are not enabled checks for the corresponding auxiliary axes, finally checking the rotary axes.

If plane G17 is active, for example, and the axes YUVA are currently enabled (that is, they are in the current axis template, not disabled by CON[ ]=0), U and Y will be selected for tool radius correction.

They are selected with the following priority:

*cartesian axes XYZ => auxiliary axes UVW => polar axes ABC.*

#### Example

```
F100
D1
CON[A] = 0
G16 U Y Z+
G0 U0 Y0 Z0
G2 U200 Z30 R100
G0 U0 Y0 Z0
D0
G0 U0 Y0 Z0
D1
G16 W X Y+
G0 W0 X0 Y0
G2 W200 X0 Y30 R100
G0 W0 X0 Y0
```

### 2.4.2 Cylindrical programming

#### Description

It is possible to interpolate with the linear axes XYZ UVW and at the same time with the polar axes ABC (v. G0, G1, G2, G3).

When programming circular interpolations with one linear and one polar axis, we have adopted an approach of the type "mm = degrees": the polar coordinates are programmed in degrees, but the CNC interprets them as mm. In other words, although the polar coordinates are degrees, they are also mm in practice if the operator has calculated the position correctly in degrees (that is multiplying the position in mm by 57.299 and dividing by the radius of the cylinder).

To avoid rejection of the machining operations by the CNC (which checks whether the parameters specified with G02 actually result in a circle) the coordinates must always be in degrees (which the slave interprets as mm) compatible with the mm specification.

This approach means that, if the numerical information is correct but the operator has not correctly converted from degrees to mm, the interpolation arc is a segment of an ellipse.

Furthermore, the radius of the cylinder can be unknown, and therefore a special mode is not required for cylindrical programming: the ABC axes can be interpolated as if they are cartesian. Note that the radius of the cylinder must be known to the operator doing the calculations, however.

Finally, this mode conforms to the modular arithmetic (mod 360) of the cylinder, inasmuch as the polar coordinates are programmed in degrees.

**Example**

```
F1000
D1G16 B Y X+
G42
G01 Y0 B0
G01 Y120
  B30
G93 F2 G02 Y90 B60 R30
G01 Y70
G03 Y60 B70 R10
G01 B150
G03 Y70 B190 R75
G01 Y110 B230
G02 Y120 B270 R75
G01 B360
G40
G58 Y200
G41
G94 F1000
G01 Y0 B0
G01 Y120
  B30
G93 F2 G02 Y90 B60 R30
G94 F1000
G01 Y70
G03 Y60 B70 R10
G01 B150
G03 Y70 B190 R75
G01 Y110 B230
G02 Y120 B270 R75
G01 B360
G40
```

**See also**

**Programming the feed, Feed command, Rapid feed, Linear interpolation, Circular interpolation.**



## 2.5 Spindle orientation

### 2.5.1 M19: Spindle orientation

**Syntax** *M19 Sangle*

**Type of function** Self-cancelling

**Description** Orients the spindle. The angle of orientation of the spindle is set with the S parameter or, if this is not available, is defined in the machine parameters.  
The spindle must be fitted with a position transducer.  
The speed of rotation for spindle orientation is defined in the parameters.  
This is a closed loop positioning command, which corresponds to the positioning of a polar axis.

*Sangle*

**Example**

```
M19 S45(orient the spindle at 45°)
```

**See also** **On the fly spindle orientation.**

## 2.5.2 On-the-fly spindle orientation

**Syntax** *M19 Sangle*

**Description**

On-the-fly orientation enables switching from a closed to an open loop. If the spindle is rotating in an open loop it can be oriented without having to first stop it.

Phase 1) the PLC zeroes the rotation bit

Phase 2) the spindle decelerates to  $V_{dec}$

Phase 3) the process assumes the programmed acceleration and decelerates to  $V_{stop}$ .

Phase 4)  $V_{stop}$  is maintained until the position error generates the current output voltage.

Phase 5) the loop closes and positioning occurs.

\*) In the current version  $V_{stop} = V_{dec}$  (SP\_VELM19) so the acceleration change in Phase 3 does not occur.

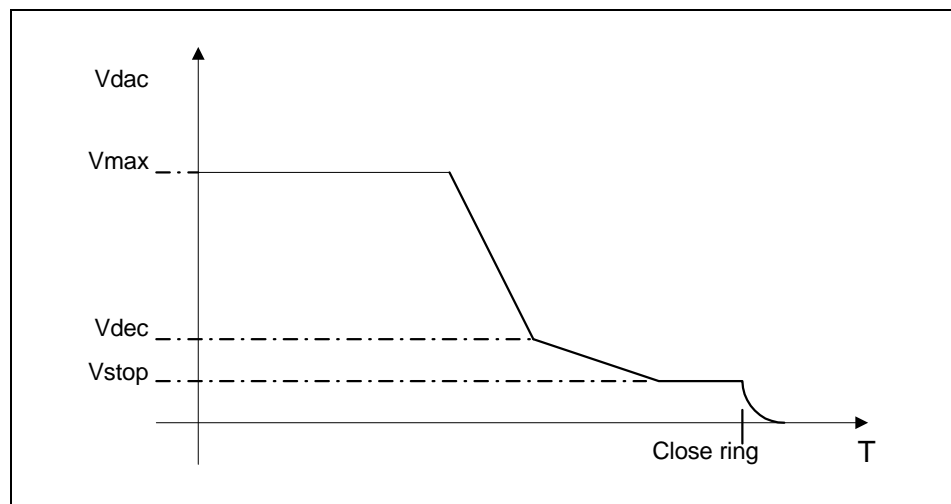


Figure 2.6 - Spindle speed control during on-the-fly orientation

**Example**

```
S1500 M3 ;Open loop rotation
M19 S45.0 ;Closed loop orientation on-the-fly orientation
```

```
S1500 M3 ;Open loop rotation
M5 ;Stop spindle
M19 S45.0 ;Orient spindle Preceding mode
```

**See also**

**M19: spindle orientation.**

## 2.6 Programming axis and spindle speed

### 2.6.1 Setting the feed rate

**Syntax**  $F_{\text{speed}}$

**Type of function** Modal

**Description**

The F parameter sets the interpolation feed rate.

The feed rate is normally expressed in mm/min, but in the case of polar axes the rate is expressed in degrees/min. There is also a mode (G95) in which the feed rate is expressed in mm/revs.

If all axes are cartesian the feed rate for the interpolation is expressed in mm/min (in G94 mode).

For a single polar axis the feed rate is expressed in degrees/min (in G94 mode).

If all axes are polar the unit is still degrees/min (in G94 mode) bearing in mind that the feed rate applies to a n-dimensional vector whose components are the angles of travel in the various polar axes.

If we want the two polar axes A and B to travel through 30° ( $\Delta A$ ) and 40° ( $\Delta B$ ) respectively in 0.20min (t), we first calculate the modulus of the vector (in this case two dimensional) (S, angle of travel in degrees):

$$S = \sqrt{\Delta A^2 + \Delta B^2} = \sqrt{30^2 + 40^2} = 50$$

then we divide by the time (t) to obtain the angular speed (V) in degrees/min:

$$V = \frac{S}{t} = \frac{50}{0.20} = 250.0$$

If both cartesian and polar axes are being used together, the cartesian coordinates are programmed in mm/min (in G94 mode).

The polar axes interpolate *linearly* over the same time of travel required by the cartesian coordinates.

This type of programming is typical of applications which require:

- a tool tangential to the yaw axis (e.g.: non-contour glass cutting with head tangential to plane);
- a polar axis whose rate is not predominant (i.e. negligible relative to the cartesian feed rate).

In the following situations:

- the travel of the X and Y axes is zero or too small to allow modulation of the polar axis feed rate with the required precision;

- the n-dimensional vectorial speed must be computed separately, as the components are not commensurable (linear and angular travel); using the option "programming the feed rate as *inverse feed*" G93.

### Examples

```
G0 X100 Y100 (feed to point P1(X100,Y100))
G1 Y200 F500 (linear to point P2(X100,Y200) with feed rate
500 mm/min)

G00 X0.000 Y0.000
G01 X30.000 Y40.000 F25.0

G00 A0.000      (** ONE SINGLE POLAR AXIS **)
G01 A30.000 F3000 (polar axis feed rate 3000 degrees/min)

G00 A0.000 B0.000 (** SEVERAL POLAR AXES **)
G01 A30.000 B40.000 F250.0 (see example on previous page)

G00 X0.000 Y0.000 A0.000 (** COMBINED CARTESIAN AND POLAR
AXES **)
G01 X30.000 Y40.000 A20.000 F25.0 (feed rate in mm/min)

G00 A0.000 B0.000      (** Inverse feed **)
G93 F250.0 G01 A30.000 B40.000 (positioning in 1/250 sec)
```

**See also**                      **Linear interpolation, Inverse feed, Timed halt.**

## 2.6.2 Continuous feed mode

**Syntax**                      **G64** (enable)  
                                  **G61** (disable)

**Type of function**            Modal, disabled by default

**Description**                In continuous feed mode the system passes from one block to the next without stopping the axes.  
 There are, however, some commands which interrupt the continuous cycle and stop the axes at the end of the block, such as G4, G9, M and T.  
 Disabling continuous feed implies stopping the axes at the end of the block which contains the command G61 itself.

### Example

```
G0 X100 Y100 (feed to point P1(X100,Y100))
G1 G64 Y200 F500 (linear to point P2(X100,Y200) no halt)
G2 G61 X200 R50 (circular arc to point P3(X200, Y200) with
halt)
```

**See also**                      **Setting the feed rate, Continuous feed mode with speed look ahead.**

### 2.6.3 Continuous feed mode with speed look-ahead

#### Description

The machine parameters allow enabling speed look-ahead  
[Channel parameters→Option: Speed Look Ahead].

If speed look-ahead is enabled in continuous feed mode the speed changes continuously to meet that of the next section, i.e.:

$$V_{out}[K] = V_{max}[K+1]$$

The approach to the piece is usually determined by a G0 (positioning), while the actual machining is carried out by interpolation, G1,G2,G3. In continuous feed mode the transition from one feed rate to another sometimes has undesired effects in the machining of the piece. Enabling speed look-ahead makes it possible to anticipate the transition and thus allows smooth machining of the piece..

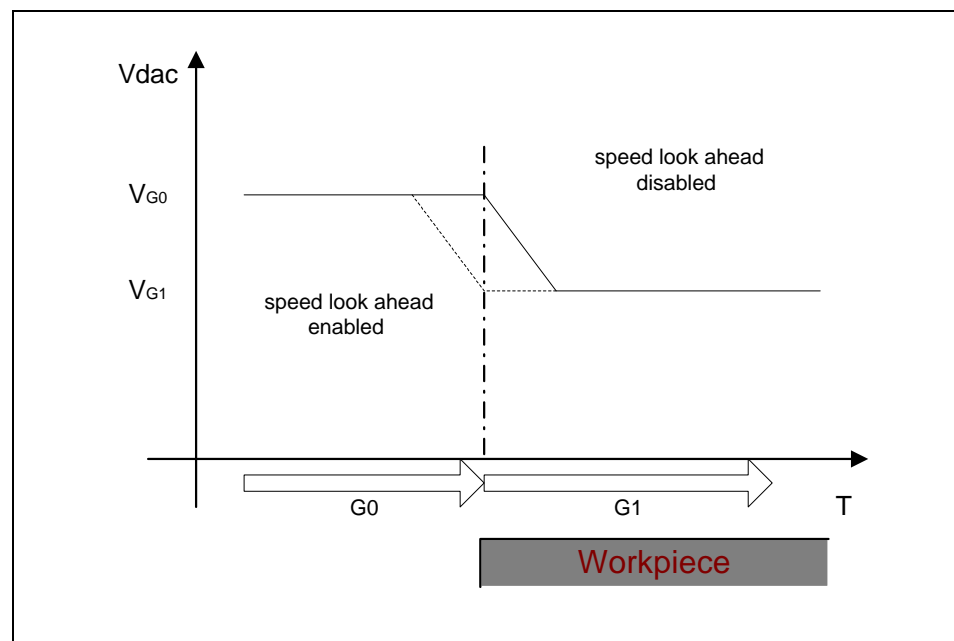


Figure 2.7 - Feed rate transition with/without look-ahead

#### Example

```
;LOOK AHEAD ENABLED
;***** REPORT *****
F10000 G64
G0 Y500 X100 ;Vmax→833333 Vout→166666
G1 X1000 ;Vmax→166666 Vout→166666
G61
G1 X1000 ;Vmax→166666 Vout→0

;LOOK AHEAD DISABLED
;***** REPORT *****
F10000 G64
```

```
G0 Y500 X100 ;Vmax→833333 Vout→833333
G1 X1000 ;Vmax→166666 Vout→166666
G61
G1 X1000 ;Vmax→166666 Vout→0
```

**See also**                      **Continuous feed mode.**

## 2.6.4    Selecting the feed mode

**Syntax**                      **G94** (feed rate in mm/min)  
                                  **G95** (feed rate in mm/rev)

**Type of function**            Modal, in mm/min by default

**Description**                The feed rate is defined by the parameter **F** and can be absolute (in mm/min) or relative to the spindle rotation speed (in mm/rev).  
 If the spindle is not fitted with a position transducer, the nominal rotational speed is used in mm/rev mode.

Using G95, which allows interpretation of the F values as mm/rev, subordinates the rate of the linear axes to the speed of the spindle itself. If the spindle speed increases, so does that of the linear axes.

The vectorial feed rate is determined by the following formula

$$V = S_{rpm} * F \quad \left[ \frac{\text{mm}}{\text{min}} \right]$$

where

$S_{rpm}$  is the speed of the spindle

$F$  advance per rev set by G95

### Example

```
G0 X100 Y100 (feed to point P1(X100,Y100))
S1000 M3 (start spindle)
G95 F2.1 (enable feed in mm/rev)
G1 Y200 (linear to point P2(X100,Y200) with feed rate
bound to spindle speed →
1000 rpm * 2.1mm/rev = 2100 mm/min)
```

**See also**                      **Setting the feed rate, Setting the spindle rotation mode.**

### 2.6.5 Selecting the spindle rotation mode

**Syntax** `Sspeed`

**Type of function** Modal

**Description** Parameter S sets the speed of rotation of the spindle.  
The speed is usually expressed in rpm, but there is a mode in which the speed is expressed as cutting speed in m/min (G96).

Speed speed of rotation, rpm or m/min

**Example** `S500 M3 (start spindle clockwise at 500 rpm)`

**See also** [Setting the spindle rotation mode..](#)

### 2.6.6 Setting the spindle rotation mode

**Syntax** `G96` (constant cutting speed)  
`G97` (constant rpm)

**Type of function** Modal, constant rpm by default (G97)

**Description** This function keeps the cutting speed constant, thus optimising the machining quality.  
The speed of rotation of the spindle is set with the parameter **S** and can be absolute (in rpm) or relative to the position of the linear axis with which it is associated, for example X (cutting speed in m/min).  
This function is characteristic of lathes, in which the position of the X axis determines the diameter of the piece at the point of machining.

The speed of the spindle cannot exceed a maximum.  
When constant cutting speed mode is disabled the speed of rotation of the spindle stays constant at the most recent value.

Selecting G96 forces interpretation of S as m/min, and to keep the cutting speed constant the spindle speed must be modulated by the radius of the contour, for example, if the radius decreases, the spindle speed increases.

$$V_{MAND} [rpm] = \frac{S_{G96} [m/min]}{2pR[m]}$$

where

$R$  is the position on the linear axis,

$S_{G96}$  is the constant cutting speed set with G96

From the above relation we can see that the speed of the spindle is inversely proportional to the position  $R$ .

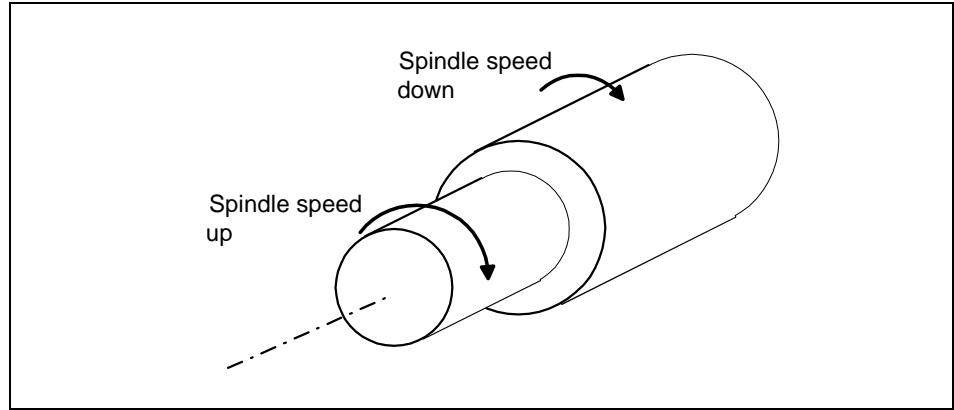


Figure 2.8 - G96 variation of the speed according to the turning radius

This mode can be combined with G95 which programs  $F$  in mm/rev, so as to ensure covering constant areas in constant time.

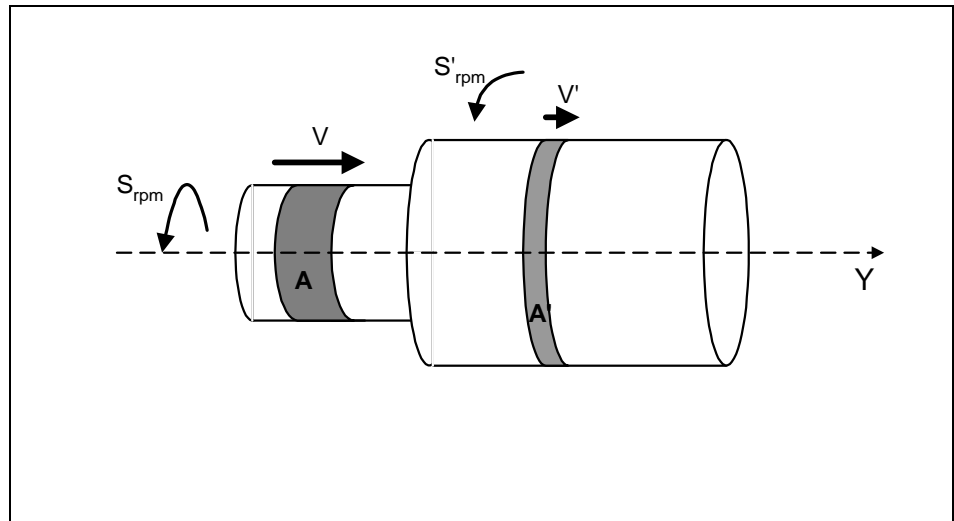


Figure 2.9 - G95-G96 Constant area machining

From the figure it is evident that with constant cutting speed the following relation holds

$$S_{rpm} > S'_{rpm}$$

If we program with G95, the linear feed rate  $V$  is proportional to  $S$  so that

$$V > V'$$

In this case, in a certain interval of time  $\Delta t$ , the machined area is given by:



$$A = 2\pi r \times V \Delta t = 2\pi r \times f(S_{\text{rpm}}) \Delta t = 2\pi r \times f(g(1/r)) \Delta t$$

$$A' = 2\pi R \times V' \Delta t = 2\pi R \times f(S'_{\text{rpm}}) \Delta t = 2\pi R \times f(g(1/R)) \Delta t$$

where  $f(S) = K \times S$  and  $S = g(1/r) = K' \times 1/r$

$$A = A' = 2\pi \times \text{const}$$

### Example

```
(* enable constant cutting speed with G96)

G0 X100 Y100 (*X radius)
S500 M3 (*spindle at 500 rpm)
G96 S100 (*constant cutting speed 100 m/min
  spindle:159 rpm → X=100)
G4 F1
G1 X50 Y50 F1000 (*spindle at 318 rpm → X=50 )
G97
G4 F1
S500 (*spindle at 500 rpm )
G4 F1
M5 (*stop spindle)

(* enable G95 and G96 simultaneously)

G0 X100 Y100
S500 M3 (*spindle at 500 rpm )
G96 S100 (*spindle at 159 rpm )
G4 F5
G95 F1 (*temporary 159 mm/min --> Vmax=2650 Hz)
G1 X50 (*spindle at 318 rpm speed 318 mm/min
  Vmax=5300Hz)
G1 Y50 (*speed 318 mm/min --> Vmax=5300Hz)
G97 G94
G4 F1
S500 (*spindle at 500 rpm )
G4 F1
M5
M30
```

### See also

**Selecting the spindle rotation mode.**

### 2.6.7 Inverse feed

<b>Syntax</b>	<i>G93 F &lt; inverse_feed &gt;</i>
<b>Type of function</b>	Modal, exclusive with G94 and G95, disabled when the CNC is switched on.

**Description** Where *inverse\_feed*, expressed in  $\boxed{\text{min}^{-1}}$ , is the ratio:

$$F = \frac{V}{L}$$

*V* is a speed expressed in any units over time.  
*L* is the length of the section in the same units.

from which the units for the parameter *F* are:

$$\frac{\left(\frac{u}{\text{min}}\right)}{u} = \frac{1}{\text{min}} = \text{min}^{-1}$$

Selects, for the current block, the speed as *the inverse* (or *reciprocal*) of the time.

**Applications** "Inverse feed" is enabled with G93, for instance:

- when programming both polar and linear axes in the same block and the travel of the linear axes is zero or too small for the polar axes to be modulated with the required accuracy;
- when the velocity must be programmed externally as an n-dimensional vector with incommensurable components (e.g.: spaces and angles).

#### Example

A is the yaw axis of the spindle which results in a large displacement of the tool point in contact with the piece even with small angular displacements. To program an operation which involves axes X, Y, Z and A, with a suitable feed rate. The travel of the linear axes is extremely limited so that the physical speed of the fourth axis cannot be accurately regulated (and hence neither can the speed of the tool point) using normal mm/min programming. This type of feed can be extracted from a CAD/CAM program, using calculations which take into consideration the tool point path and the axis movements.

In this case we use *inverse feed* programming (G93).

The axes must move from:

X0.000 Y0.000 Z0.000 A0.000

to:

X0.010 Y0.010 Z0.023 A100.000

interpolating linearly.

We know the path of the tool point during interpolation (different from the XYZ axis paths):

400 mm

We know the linear feed rate of the tool point during the interpolation:

1600 mm/min

We program G93 with the speed:

$$F = \frac{1600}{400} = 4$$

(the reciprocal of the feed time, 0.25min)

We thus obtain:

G00 X0.000 Y0.000 Z0.000 A0.000

G93 **F4** G01 X0.010 Y0.010 Z0.023 A100.000 (positioning in ¼ of a second)

G94 F1000 (resets the speed to mm/min) Auxiliary functions

## 2.7 Polar axes programming

### 2.7.1 Polar axes with travel limitation

The targets expressed are absolute and range from the minimum target to the maximum one, inclusive. In relation to the distance covered, the behaviour is the same as that for a Cartesian axis.

e.g.:

```
G0 C-90.000
G0 C270.000 ; The axis moves forwards through 360
degrees
```

```
G0 C315.000
G1 F5000 C45.000 ; The axis moves backwards through 270
degrees
```

```
G0 C45.000
G1 F3000 C315.000 ; The axis moves forwards through 270
degrees
```

The behaviour described can be modified by means of certain operators that allow the direction and rpm rate of the polar axis to be conditioned. See the *Selection of the direction and rpm rate* section. Note, in that section, how the behaviour described is the same as that of the AC operator.

### 2.7.2 Round axes

#### Operation in the rapid mode

Depending on the counting method, the programmed target determines the position to reach and not the absolute target of the axis. If the target is beyond the [0..mod) range, it is brought back within that range.

The direction is therefore determined so as to reach the programmed position via the shortest possible route. If the programmed position is the same as the one in which the axis actually is, the axis remains at a standstill.

e.g.:

```
G0 C-90.000
G0 C270.000 ;The axis remains at a standstill
```

```
G0 C315.000
G0 C45.000 ; The axis moves forwards through 90 degrees
```

```
G0 C45.000
G0 C315.000 ; The axis moves backwards through 90 degrees
```

The behaviour described can be modified by means of certain operators that allow the direction and rpm rate of the polar axis to be conditioned. See the *Selection of the*

*direction and rpm rate* section. Note, in that section, how the behaviour described is the same as that of the *DC* operator.

### Operation in the feed mode

If a target is programmed within the  $[0..mod)$  range, the axis will reach the required position *without passing via 0*. E.g.:

```
G0 C-90.000
```

```
G1 F5000 C270.000 ; The axis remains at a standstill
```

```
G0 C315.000
```

```
G1 F5000 C45.000 ; The axis moves backwards through 270 degrees
```

```
G0 C45.000
```

```
G1 F5000 C315.000 ; The axis moves forwards through 270 degrees
```

A target beyond the  $[0..mod)$  range can also be programmed without any limitations. In that case, the distance to cover is determined by considering the programmed target and the current position, *only this latter* within the  $[0..mod)$  range. After the block has been executed, the programmed target will be brought back within the  $[0..mod)$  range and will form the new current position. This means that if targets beyond the  $[0..mod)$  range are programmed for a round axis, they will always result in movement of the axis, regardless of the current position.

E.g.:

```
G0 C45.000
```

```
G1 F5000 C405.000 ; The axis moves forwards through 360 degrees
```

```
G1 C405.000 ; The axis moves again forwards through 360 degrees
```

```
G1 C45.000 ; The axis remains at a standstill
```

```
G1 C-315.000 ; The axis moves backwards through 360 degrees
```

```
G1 C-315.000 ; The axis moves again backwards through 360 degrees
```

```
G1 C45.000 ; The axis remains at a standstill
```

### 2.7.3 Selection of the direction and rpm rate

Certain operators are available for conditioning the direction and rpm rate of the polar axis. These operators are implemented like a series of functions that produce values which must be assigned to the address of the polar axes for which conditioned movement is required. For example:

```
G0 C=DC(45.000)
G1 F5000 C=ACP(1,45.000)
```

These operators have the same effect on blocks executed in the rapid and feed modes.

#### Incremental positioning (IC)

<polar axis>=IC(<Incremental target>)

Es.:

```
C=IC(-25.000)
```

The axis is moved so as to cover the distance defined by <incremental\_target> in the positive or negative direction, depending on the sign. The target reached will be given by the algebraic sum of the current target and <incremental\_target>, brought back within the [0..mod) range in the case of a round axis.

#### Absolute positioning (AC)

<polar\_axis>=AC(<absolute\_target>)

E.g.:

```
C=AC(45.000)
```

In the case of *axis with travel limitation*, <absolute\_target>, it is expressed within the minimum target and maximum target range, inclusive. In relation to the distance covered, the behaviour is the same as that for a Cartesian axis. For example:

```
C=AC(-90.000)
C=AC(270.000)           ; The axis moves forwards through 360
degrees
```

```
C=AC(315.000)
C=AC(45.000)           ; The axis moves backwards through 270
degrees
```

```
C=AC(45.000)
C=AC(315.000)          ; The axis moves forwards through 360
degrees
```

In the case of a *round axis*, and if an <absolute\_target> value is programmed within the [0..mod) range, the axis will reach the required position *without passing via 0*. E.g.:

```
C=AC(-90.000)
C=AC(270.000)          ; The axis remains at a standstill
```

```
C=AC(315.000)
C=AC(45.000)           ; The axis moves backwards through 270
degrees
```

```
C=AC(45.000)
C=AC(315.000) ; The axis moves forwards through 270
degrees
```

Again in the case of a *round axis*, and *<absolute\_target>* value can also be programmed beyond the [0..mod) range without any limitations. In this case, the distance to cover is determined considering the *<absolute\_target>* and the current position, *only this latter* within the [0..mod) range. After the block has been executed, the *<absolute\_target>* will be brought back within the [0..mod) range and will form the new current position. This means that programming an AC function with *<absolute\_target>* values beyond the [0..mod) range for a round axis will always result in movement of this latter, regardless of the current position.

E.g.:

```
C=AC(45.000)
C=AC(405.000) ; The axis moves forwards through 360 degrees
```

```
C=AC(405.000) ; The axis moves again forwards through 360
degrees
```

```
C=AC(45.000) ; The axis remains at a standstill
C=AC(-315.000); The axis moves backwards through 360
degrees
```

```
C=AC(-315.000); The axis moves again backwards through 360
degrees
```

```
C=AC(45.000) ; The axis remains at a standstill
```

### **Absolute positioning with the least distance (DC)**

```
<polar_axis>=DC(<absolute_position>)
```

E.g.:

```
C=DC(45.000)
```

Depending on the counting module, *<absolute\_position>* determines the position to reach and not the absolute target of the axis. If *<absolute\_position>* is beyond the [0..mod) range, it will be brought back within this range.

The direction is therefore determined so as to reach the programmed position via the shortest possible route. If the programmed position is the same as the one in which the axis actually is, the axis remains at a standstill.

E.g.:

```
C=DC(-90.000)
C=DC(270.000) ; the axis remains at a standstill
```

```
C=DC(315.000)
C=DC(45.000) ; The axis moves forwards through 90 degrees
```

```
C=DC(45.000)
C=DC(315.000) ; The axis moves again backwards through 90
degrees
```

In the case of *axis with travel limitation* (module 360):

```
C=AC(765.000) ; The position of the axis will be 45 degrees
(765 MOD 360)
C=DC(315.000) ; The axis moves forwards through 90 degrees
and reaches the position of 675
```

### Absolute positioning in the positive direction (ACP)

```
<polar axis>=ACP(<absolute_position>)
<polar axis>=ACP(<n_rev>,<absolute_position>)
```

E.g.:

```
C=ACP(45.000)
C=ACP(2,45.000)
```

Depending on the counting module, *<absolute\_position>* determines the position to reach and not the absolute target of the axis. If *<absolute\_position>* is beyond the [0..mod) range, it will be brought back within this range.

The required position is reached by moving the axis in the positive direction. If the programmed position is the same as the one in which the axis actually is, the axis remains at a standstill.

E.g.:

```
C=DC(-90.000)
C=DC(270.000) ; the axis remains at a standstill

C=DC(315.000)
C=ACP(45.000) ; The axis moves forwards through 90 degrees

C=DC(45.000)
C=ACP(315.000) ; The axis moves forwards through 270
degrees
```

In the case of *axis with travel limitation* (module 360):

```
C=AC(765.000) ; The position of the axis will be 45
degrees(765 MOD 360)

C=ACP(315.000); The axis moves forwards through 270 degrees
and reaches the position of 1035
```

An additional number of revolutions can be expressed *<n\_rev>*. If programmed, the *<absolute\_position>* reached will be the same, but the required number of revolutions will be added to the distance normally covered to reach it in the positive direction.

```
C=DC(-90.000)
C=ACP(1,270.000) ; The axis moves forwards through 360
degrees

C=DC(315.000)
C=ACP(1,45.000) ; The axis moves forwards through 450
degrees

C=DC(45.000)
C=ACP(1,315.000) ; The axis moves forwards through 630
degrees
```



In the case of *axis with travel limitation* (module 360):

C=AC(765.000) ; The position of the axis will be 45 degrees(765 MOD 360)

C=ACP(1,315.000) ; The axis moves forwards through 630 degrees and reaches the position of 1395

### Absolute positioning in the negative direction (ACN)

<polar axis>=ACP(<absolute\_position>)  
<polar axis>=ACP(<n\_rev>,<absolute\_position>)

E.g.:

C=ACN(45.000)  
C=ACN(2,45.000)

Depending on the counting module, <absolute\_position> determines the position to reach and not the absolute target of the axis. If <absolute\_position> is beyond the [0..mod) range, it will be brought back within this range.

The required position is reached by moving the axis in the positive direction. If the programmed position is the same as the one in which the axis actually is, the axis remains at a standstill.

E.g.:

C=DC(-90.000)  
C=DC(270.000) ; the axis remains at a standstill

C=DC(315.000)  
C=ACN(45.000) ; The axis moves backwards through 270 degrees

C=DC(45.000)  
C=ACP(315.000) ; The axis moves backwards through 90 degrees

In the case of *axis with travel limitation* (module 360):

C=AC(765.000) ; The position of the axis will be 45 degrees(765 MOD 360)

C=ACN(315.000); The axis moves forwards through 90 degrees and reaches the position of 675

An additional number of revolutions can be expressed <n\_rev>. If programmed, the <absolute\_position> reached will be the same, but the required number of revolutions will be added to the distance normally covered to reach it in the positive direction.

C=DC(-90.000)  
C=ACN(1,270.000) ; The axis moves backwards through 360 degrees

C=DC(315.000)  
C=ACN(1,45.000) ; The axis moves backwards through 630 degrees

C=DC(45.000)

C=ACP(1,315.000) ; The axis moves backwards through 450 degrees

In the case of *axis with travel limitation* (module 360):

C=AC(765.000) ; The position of the axis will be 45 degrees(765 MOD 360)

C=ACN(1,315.000) ; The axis moves forwards through 450 degrees and reaches the position of 315

## 2.7.4 Speed programming in the presence of polar axes

### Interpolation with one or more Cartesian axes and no polar axis

The speed is programmed on the trajectory described by the Cartesian axes in mm/min (in the G94 mode).

E.g.:

```
G00 X0.000 Y0.000
G01 X30.000 Y40.000 F5000
```

### Interpolation with one or more polar axes and no Cartesian axis

#### Only polar axis

The speed polar axis is programmed in mm/min (in the G94 mode).

Es.:

```
G00 C0.000
G01 C30.000 F3000
```

#### MORE polar axes

Programming is always carried out in degrees/min (in the G94 mode) considering, however, that the speed acts on an n-dimensional vector whose components on the various dimensions are the angles covered by the polar axes.

If two polar axes A and B must cover angles of 30° (ΔA) and 40° (ΔB), respectively, taking 0.20min, (t), first calculate the first module of the vector, two-dimensional in this case (S, angle covered in degrees):

$$S = \sqrt{\Delta A^2 + \Delta B^2} = \sqrt{30^2 + 40^2} = 50$$

then, divide by the time (t) to obtain the angular velocity (V) in degrees/min:

$$V = \frac{S}{t} = \frac{50}{0.20} = 250.0$$

Now program:

```
G00 C0.000 B0.000
G01 C30.000 B40.000 F250.0
```

**Interpolation in the presence of both Cartesian and polar axes**

The speed is programmed on the trajectory described by the *sole Cartesian axes* in mm/min (in the G94 mode).

The polar axes interpolate *linearly* in the same time taken by the Cartesian axes.

E.g.:

```
G00 X0.000 Y0.000 A0.000
G01 X30.000 Y40.000 C20.000 F2500
```

In certain situations, it may be more convenient to program the speed as *inverse of time*, by means of the G93 mode. This sort of situation may occur when the component on the Cartesian axes is extremely small and the speed of the polar axes must be accurately modulated.

**Programming the speed as time reciprocal**

In the G93 mode, no distinctions are made between the types of axes in a block since the reciprocal of the travel time of the block is expressed and not a linear/angular space per unit of time.

## 2.8 Auxiliary Functions

### 2.8.1 Functions M and T

#### M functions

The M auxiliary functions (miscellaneous) are generally plant-specific and are defined by the machine manufacturer, (except for M2, M19 and M30).

When setting the M functions the following standards must be adhered to:

Function	Meaning
M0	Unconditional halt
M1	Optional halt
M2	End of program (not redefinable)
M3	Clockwise spindle rotation
M4	Counterclockwise spindle rotation
M5	Spindle halt
M6	Tool change
M7	Coolant 1 ON
M8	Coolant 2 ON
M9	Coolant OFF
M10	Lock
M11	Unlock
M19	Spindle orientation (not redefinable)
M30	End of program (not redefinable)
M40	Automatic range change
M41	Select range 1
M42	Select range 2
M43	Select range 3
M44	Select range 4
M48	Reset override
M49	Override off
M60	Change piece

The M functions ensure synchronisation of the channel with the rest of the system, via data interchange with the PLC.

The machine parameters allow selection of the control mode (Enable Overlapped M) which enables reduction of deadtime due to waiting for synchronisation signals.

#### T functions

The T auxiliary functions specify and setup the tool to be fitted with the next M6 command.

The T function specifies the tool number, but has no bearing on tool compensation which is handled by the D function.

**Order of execution**

The order of execution of commands in a given block is as follows:

- T functions
- S functions
- M functions
- Feed
- Timing

A block may contain only one command of each type, except for the M functions, of which there may be up to 3 in one block.

## 2.9 Tool compensation

### 2.9.1 Tool length correction

**Tool corrector table** The D function enables selection of the tool corrector, DD disables tool compensation. The CNC contains a table which the operator can access, in which for every corrector a set of data can be specified which defines the mode and properties of the tool compensation. The 0 element of the table is not available. for example, if 10 tools are defined in the *defcn*, the ones available are those from 1 to 9.

The command Dxx acquires the correction data for the tool and automatically enables the length correction along the axis orthogonal to the machining operation. The tool length is added to the position along the specified axis.

Note: before the Dxx function is used, it is obligatory to define the *versors* for both the specified machining plane and the tool. In the simplest case, where a machining operation takes place on plane XY (G17) using a tool that works perpendicularly to the plane along Z axis, it is always necessary to set the following *versors*:

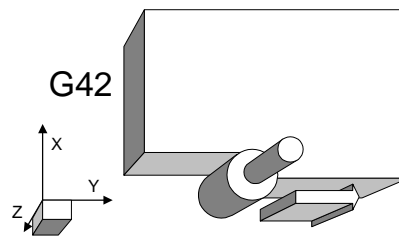
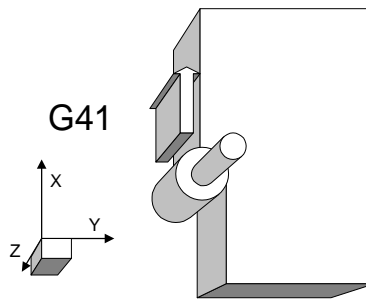
**EI0 EJ0 EK1**      Versor for the machining plane

**EP0 EQ0 ER1**      Versor for the tool

**See also**      **Selecting the work plane, Tool radius compensation.**

## 2.9.2 Tool radius compensation

<b>Syntax</b>	<i>G40</i> (disables radius compensation) <i>G41</i> (enables radius compensation to the left of the tool path) <i>G42</i> (enables radius compensation to the right of the tool path)
<b>Type of function</b>	Modal, disabled by default
<b>Description</b>	<p>Tool radius compensation enables contour milling on a plane with automatic compensation of the tool's radius. The value of the tool radius used to calculate the actual tool path is taken from the D corrector for the tool.</p> <p>Compensation is applied to the selected plane, hence by default on the XY plane(G17).</p> <p>For correct tool radius compensation, proceed as follows:</p> <ul style="list-style-type: none"><li>• select the appropriate tool corrector Dxx;</li><li>• program G41or G42.</li></ul> <p>In active compensation mode:</p> <ul style="list-style-type: none"><li>• Each feed command must involve at least one of the two plane axes, otherwise the compensation algorithm will return an error.</li><li>• The correction direction/milling plane cannot be modified without disabling the compensation itself.</li><li>• Two consecutive rapid feeds cannot be programmed.</li></ul> <p>Two contiguous tool paths, each with radius compensation, can be connected in a different way:</p> <ul style="list-style-type: none"><li>• if the two paths form a convex angle the two paths are connected by a circular arc,</li><li>• if the two paths form a concave angle the first path is continued until it intersects the projection of the second.</li></ul> <p>G40, which deactivates the compensation function, takes effect from the start of the next machining segment. The following two figures show tool paths with G41 and G42 correction using a flat end mill.</p>



*Figure 2.10 - G42 correction with flat end mill:*



The following figure shows G42 correction for a lathed part,

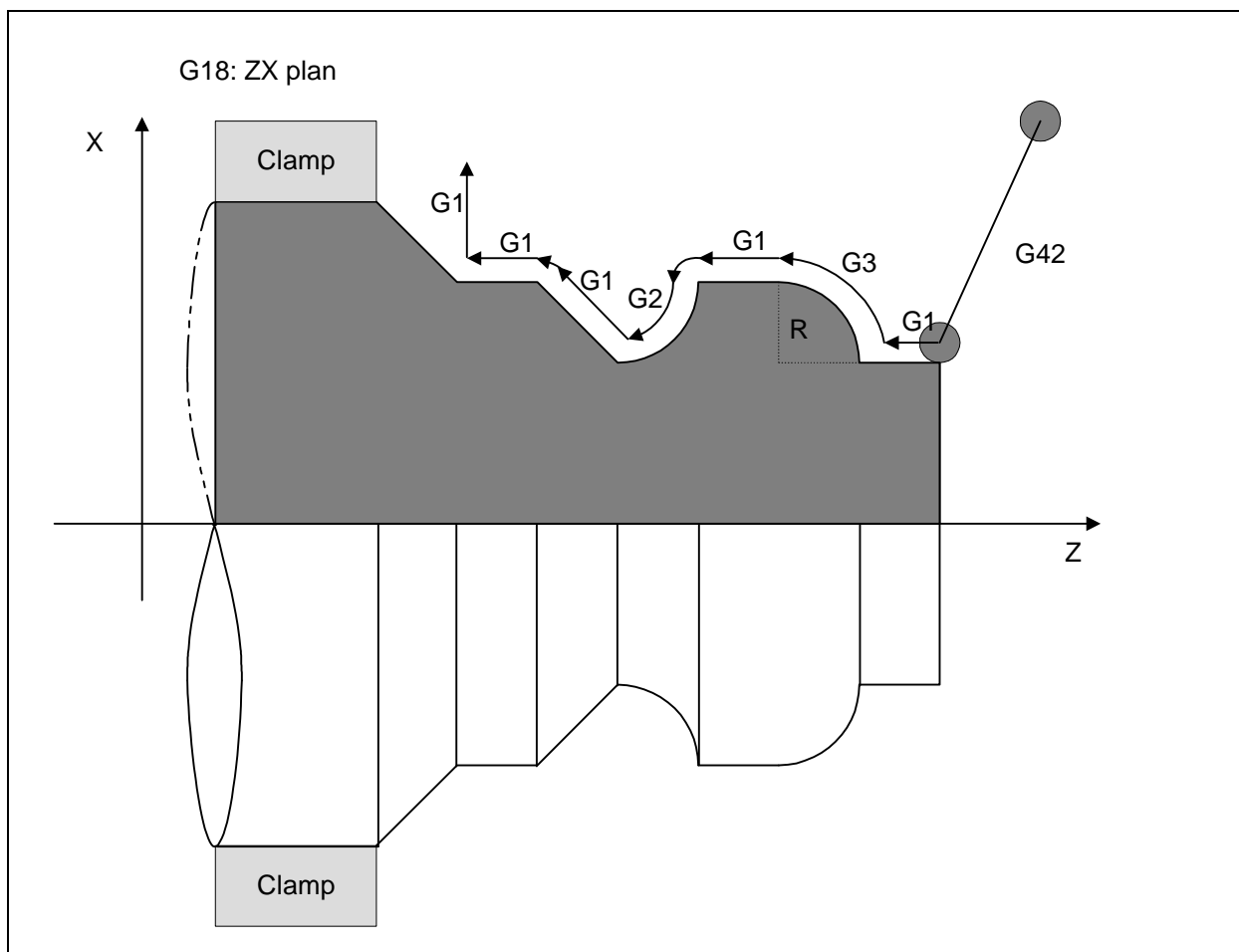


Figure 2.11 -Lathe tool correction:

and in the following for a bored part:

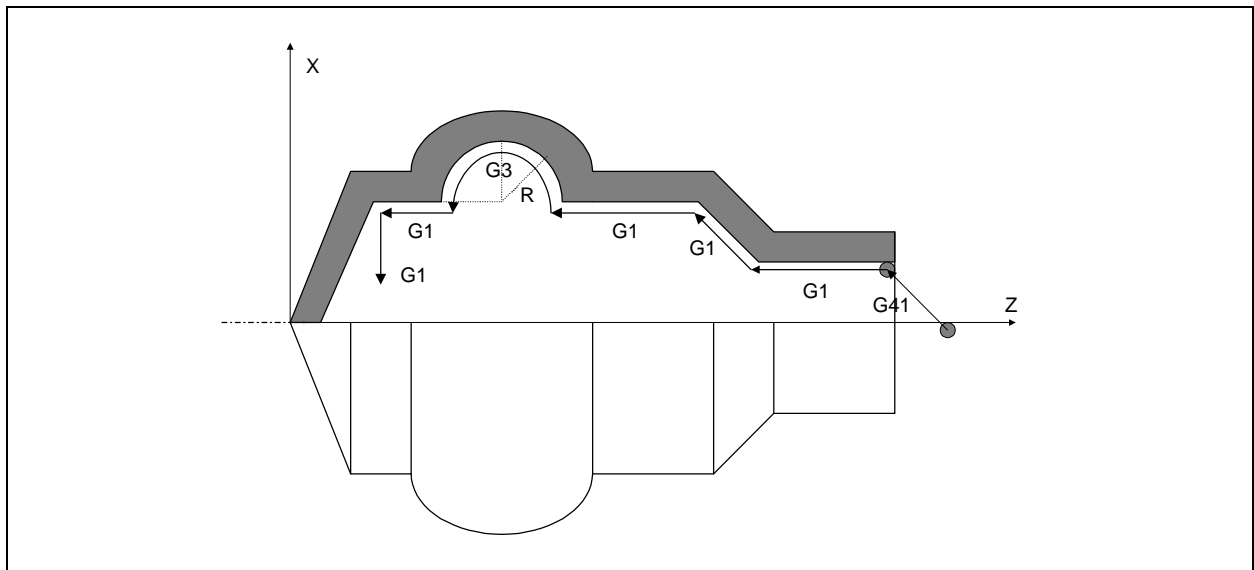


Figure 2.12 - Boring tool correction

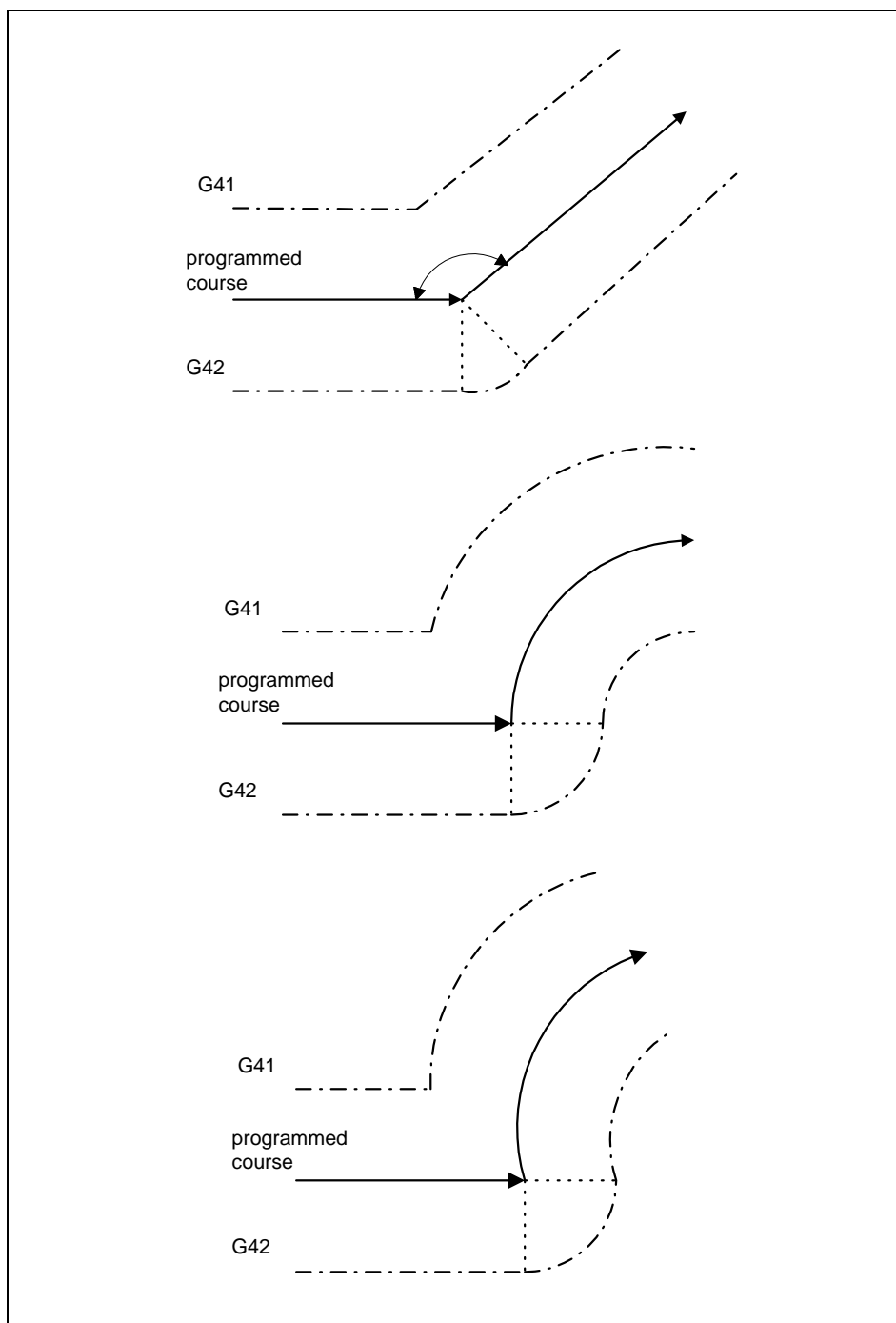


Figure 2.13 - Tool compensation: programmed and corrected path

### Interference check

The tool correction algorithm recognises interference ( or overcutting) on a linear or circular arc segment, only if the direction of the path of the centre of the tool differs from the programmed path by an angle between  $90^{\circ}$  and  $270^{\circ}$ .

For example, the following types of interference are recognised:

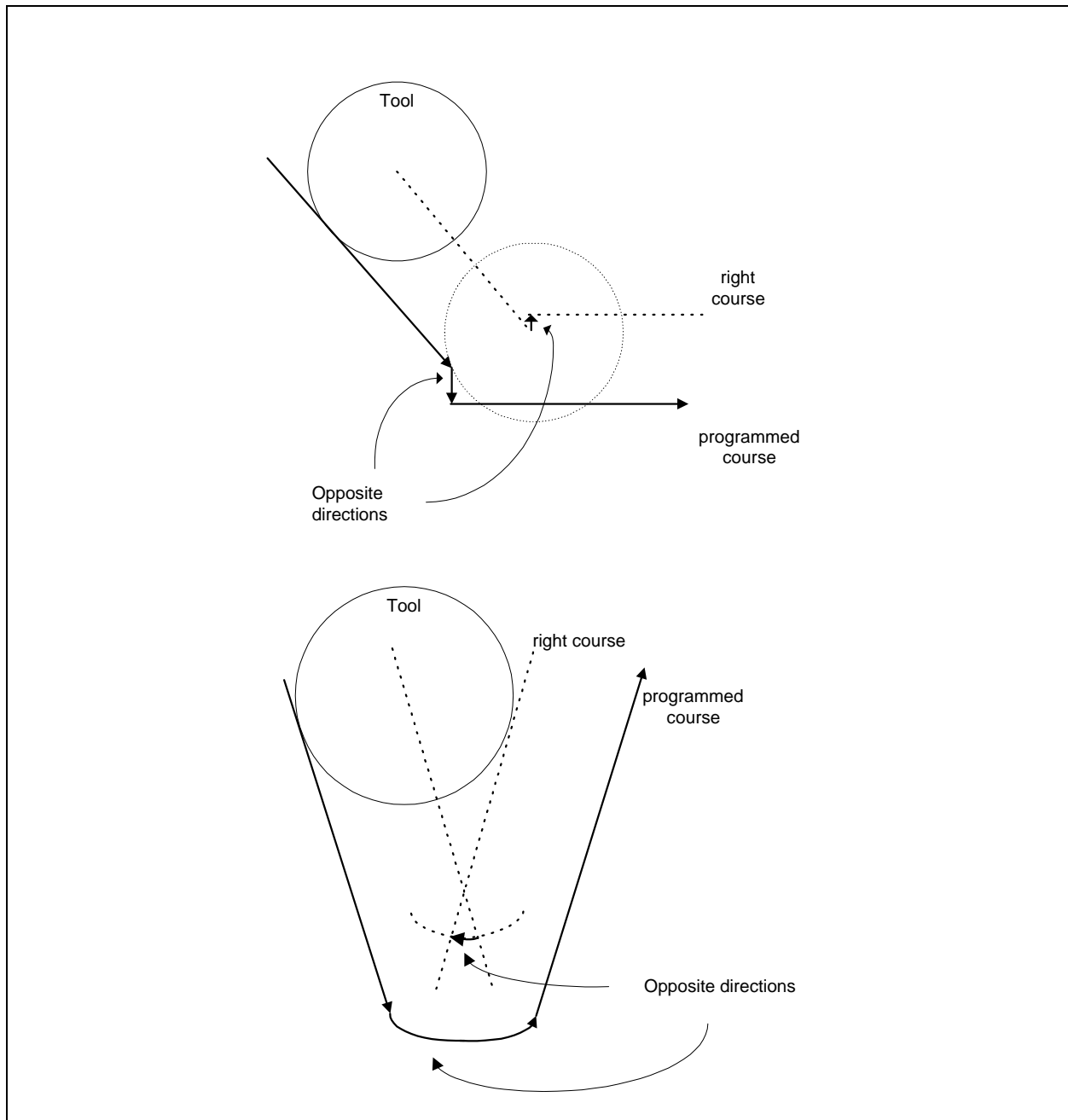


Figure 2.14 - Interference on linear and circular arc paths

#### Example

```
G0 G41 X100 Y100 (feed to point P1(X100-tool radius,Y100))
G1 Y200 F500 (linear to point P2(X100-tool radius,Y200))
G2 G40 X200 R50 (arc to point P3(X200+tool radius, Y200))
```

#### See also

Tool lenght correction.

**END OF CHAPTER**

## 3 Advanced Programming

### 3.1 Selecting Automation Language mode

**Syntax**                      `MODE = AUTO` (*default*)

**Description**                      This command selects the Esa-Gv Automation Language programming mode, with the features listed below.  
In this mode comments are preceded with a semicolon and the reserved ISO character "%" is used to identify variable names. For the rest, Automation Language can be considered an extension of the ISO language.

**See also**                      **Standard Programming: Selecting ISO mode**

## 3.2 Parametric Programming

Parametric programming allows variables to be used instead of numeric values or strings. This makes programming flexible, guaranteeing that the code is clear.

### Types of Variables

Three types of variables are available:

- Global system variables: system or user variables accessible to both the CNC and operator interface.
- Pre-defined variables: variables of the numeric type which can be addressed by means of the V<sub>axx</sub>, VL<sub>xx</sub>, VG<sub>xx</sub> codes.
- Symbolic variables: variables whose names are defined by the user.

### 3.2.1 Global system variables

#### Syntax

%nomregistro

% nomregistro.numerobit

Global variables defined in the shared memory can be accessed within the system (CNC, UI, PLC).

#### Description

To access the registers and structures in the shared memory, the % character must precede the name of the register in question. If the name of the register is followed by .numerobit, this means that only one bit is to be tested or modified. The bit number must be compatible with the size of the register.

Type of register	Tolerated range
BYTE	from 0 to 7
WORD	from 0 to 15
DWORD	from 0 to 31

#### Example

```
%regtool[0].0=1 ;sets bit 0 of reg. %regtool[0]
%regtool[0].1=0 ;resets bit 1 of reg. %regtool[0]
%regtool[0].2=1 ;sets the bit 2 of reg. %regtool[0]
```

### 3.2.2 Pre-defined numeric variables

These are numeric variables pre-defined by the system. They can be accessed using codes VA (automatic variables), VL (local variables) or VG (global variables) followed by the number of the parameter.

All the numerical values in words can be referred to via parameters or expressions, with the exception of the N codes.

Variables fall into the following categories:

VA xxx	Automatic	Only visible inside the program or subprogram
VL xxx	Local	Only visible inside its channel
VG xxx	Global	Visible to all channels

### 3.2.3 Assigning a variable pre-defined numeric variables

**Syntax**                      *VA, VL, VG* number = expression

**Description**                      The variable to the left of the = (assignment) sign is set to the value of the expression on the right.

**Example**

```
G0 X0 Y0 ;Feed to point P1(X0, Y0)
VA1 = 100 VA2 = 200 ;Assigns two variables
G1 F1000 XVA1 YVA2 ;Linear feed to point P2(X100,Y200)
```

**See also**

### 3.2.4 Symbolic variables

The programmer can use personalized variables (the name and type are established by the user) to which numeric values, strings or references can be assigned. Use of this type of variable allows the code to be made more comprehensible. To use the symbolic variables, the machine parameter [*System->Configure Channel->Local variables for Subroutine level*] must be different from zero.

Variables of the following Type:

NUMERIC	variables that contain both real numbers (numbers with the decimal point) and integers.  The numbers are expressed in decimals, possibly with decimal point and sign. Up to 9 whole figures are permitted. The exponential notation is not permitted.  The expressions are processed in binary mode with a 14-figure precision.
STRING	variables that contain sequences of characters
RECORD	variables that enable access to information about another object which can be a symbolic variable, an input/output register, a user register defined in defcn, etc.

### 3.2.5 Declaration of Symbolic Variables

#### Syntax

```
DBL  symb [=numeric value] [,symb [=numeric value]]
STR  symb [= string] [,symb [= string]]
REC  symb [= ^value] [,symb [= ^value]]

symb=nomevar[[elem number]]
```

#### Description

Before the symbolic variables can be used to compile a PP, they must have been previously declared.

DBL declares that the following symbols are numeric

STR declares that the following symbols are the string type

REC declares that the symbols are of the record type

The symbolic variables used must be declared at the beginning of the PP.

The name of a symbolic variable:

- consists of up to 32 alphanumerical characters;
- cannot correspond to a key word of the programming language;
- cannot begin with the ‘\_’ character.

The variables can be initialized during the declaration phase, i.e. their initial value can be specified before they are used in the PP. If this value is not specified, the system assumes a default value, coherent with the type of data item declared.



### TYPE OF DATA DEFAULT VALUE ITEM

numeric	0.0	zero
string	""	empty string
record	NULL	no reference

Several symbols of the same type can be declared (and initialized if required) on the same line by separating them with ',' (comma).

Recognition of the variables discriminates between capitals/lower case letters (e.g. Dimension is not the same as DIMENSION or dimension).

One-dimensional arrays can be declared, for example

```
DBL QuoteX[10] ; vector of 10 Numeric variables
```

This allows homogeneous data to be organized so as to enable access by the index. Since a vector consists of n elements, it cannot be initialized during the declaration phase.

### Visibility of the variables

Personalized variables can be defined within any programming module (main PP, subroutine and fixed cycles).

The variables defined in the PP or its subroutine are activated and visible until its conclusion, thus also during execution of a recalled subroutine.

```
Main Pp %1
DBL NRiga ; number of lines to execute
DBL NFORI, MAXY
...
...

NRiga = 20
  JSR "boring"   Executes a row of holes : boring
N10      .....
          IF (NRiga > 20) JMP 10
          .....
          RET
```

It is not possible for one or more variables declared on the same subroutine level to have the same name, while this is allowed for variables defined at different subroutine levels. The fact of having variables with the same name at different subroutine levels is resolved in the following way:

- a variable is always visible within the module in which it is defined;
- a variable is visible in the modules recalled by the PP/Subroutine if these do not declare a variable with the same name

```
Main % 1

DBL AUX1 = 35,
DBL AUX2 = 2

VL0 = AUX2      ; VL0 = 2

JSR 112 -> LEVEL1 : 112
```

```

VL2 = AUX2 ; VL2 = 10
VL3 = AUX1 ; VL3 = 35

JSR "SUB1" >LEVEL2 :SUB1
DBL AUX1 = 0

VL4 = AUX2 ; VL4 = 10
VL5 = AUX1 ; VL5 = 0

```

**Example**

```

DBL mezz, MIN=0, QuoteX[10]
REC PUNT[3] ;Array of 3 invalid pointers
REC POINTER = ^mezz
STR SUBROUTINE, NAME[3], SUB="example.cfu"

```

**3.2.6 Allocation of Symbolic Variables****Syntax**

```

symb = compatible value
symb = nomevar[[elem number]]

```

**Allocation of variables of the record type**

Following an allocation, the variable of the record type to the left of the allocation symbol (=) contains the reference to the allocated symbol. Once a record variable has been allocated, it can be used instead of the symbol to which it refers. A variable of the record type can be allocated to another variable of the record type.

```

REC PUNT[10]
DBL Port, Limit switch

PUNT[0] = ^%C23 ;PUNT[0] is able to access
               ;logic register C23

PUNT[0] = 15.2 equals %C23=15.2

PUNT[1] = ^%QW3.1 ;PUNT[1] is able to access
               ; bit 1 of output register QW3
PUNT[2] = ^%IW8 ;PUNT[2] is able to access
               ;input register IW8
PUNT[3] = ^%subroutine[0].name
               ;PUNT[3] is able to access the
               ;structure in defcn subroutine[0].name
PUNT[4] = ^Port ;PUNT[4] is able to access the
               ;variable of the numeric type Port
PUNT[5] = ^VA20 ;PUNT[5] is able to access the
               ;pre-defined variable VA20
PUNT[6] = ^VG2 ;PUNT[5] is able to access the
               ;pre-defined variable VG2
PUNT[7] = ^VL13 ;PUNT[5] is able to access the
               ;pre-defined variable VL13

```

Use of record variables can be useful when variables in the shared memory must be accessed frequently, as it speeds up the process.

### Allocation of variables of the numeric type

The contents of the variable on the left of the allocation symbol (=) is set at the value of the expression on the right.

Besides being allocated to a defined variable of the DBL type, a numeric value can also be allocated to a record variable that refers to a symbol of the numeric type.

```
DBL    QuotaX
REC    PUNT

QuotaX = 100          ; QuotaX 100.0

PUNT = ^ QuotaX
PUNT = 300          ; equals QuotaX 300.0
```

### Allocation of variables of the string type

The contents of the variable on the left of the allocation symbol (=) is set at the value of the string on the right.

Besides being allocated to a defined variable of the STR type, a string value can also be allocated to a record variable that refers to a symbol of the string type.

```
STR    SUB= "fora100.cfu"
REC    PUNT

SUB = "1098.cfu"      ; SUB "1098.cfu"

PUNT = ^ SUB
PUNT = "boring.cfu"  ; equals SUB "boring.cfu"
```

## 3.2.7 Expressions

Expressions are created by combining operands and operators as in algebra.

Expressions can be used:

- To assign the resulting value to a variable, the expression appears on the right of the "=" sign that follows the variable to which the calculation is allocated.

E.g.      `MyVar = (VarA ** VarB) * SIN(VarC)/COS(VarD)`

- As condition of an IF instruction; the expression is resolved and the result evaluated by the IF instruction.

Eg.      `IF( ( (VarA ** VarB) * SIN(VarC)/COS(VarD) ) > 0 ) THEN`

- In line with any ISO instruction.

Eg.      `GO X(VarA ** VarB) Y(SIN(VarC)/COS(VarD) )`  
           `M(VarA-VarB)`

<b>Type of operand</b>	<b>Syntax</b>
constant	number
pre-defined variable	VA, VL, VG number
symbolic variable	symbol
0x<number>	Hexadecimal constant
%C <register number>	I/O logic registers
%IW <register number>	physical Input registers
%OW <register number>	physical Output registers
glob. symbolic vector elem.	%name_lowercase_var[el.num.]
global system var.	%name_lowercase_var
Structured glob. system var.	%name_lowercase_var.member
synchronous global system var.	?%name_lowercase_var
<b>operator</b>	
addition (+)	expression+expression
subtraction (-)	expression-expression
multiplication (*)	expression*expression
division (/)	expression/expression
negation (-)	-expression
Raising	expression ** expression
<b>Function</b>	
square root	<i>SQRT</i> (expression)
sine	
cosine	
tangent	
arc sine	
arc cosine	
arc tangent	
absolute value	
rounding off	
truncation	
rounding off to the highest figure	
addition (for dword)	
subtraction (for dword)	
multiplication (for dword)	

Boolean operators	Description
<espress> AND <espress>	<i>and</i> bit to bit operation
<espress> & <espress>	<i>and</i> bit to bit operation
<espress> OR <espress>	<i>and</i> bit to bit operation
<espress>   <espress>	<i>and</i> bit to bit operation
<espress> && <espress>	<i>and</i> logic operation (on entire word)
<espress>    <espress>	operazion <i>or</i> logica ((on entire word)
SHR(<espress1>, <espress2>)	shift right <i>espress1</i> di <i>espress2</i> bits
SHL(<espress1>, <espress2>)	shift left <i>espress1</i> di <i>espress2</i> bits
MOD(<espress1>, <espress2>)	resto divisione <i>espress1</i> modulo <i>espress2</i>

## Description

Expressions are composed of constants and variables combined by the listed operators and functions.

Operators are executed with algebraic priority, with optional parentheses to force priority ( ).

Numbers are expressed in floating point signed decimal arithmetic with up to 9 significant figures; exponential notation is not allowed.

Expressions are evaluated in 14-bit arithmetic; note that approximates numbers with significant figures after the decimal point and hence a certain margin of error applies.

Note also that the number of a variable can itself be an expression, thus enabling indexed variable reference.

## Example

```

VA1 = 2
VA2 = 2000.0
VA3 = VA[VA1] ;indexed variable reference
VA4 = 45.0
VA5 = VA2 + (VA3 * SIN(VA4)) * VA2
G01 F1000 XVA2 YVA5
VA1 = VA1 + 1

N100 %regasync[0] = %nc[1].du[0]
N200 %regasync[1] = %nc[1].du[1]

N600 VA[0] = 1
N600 VL[0] = 2
N600 VG[0] = 3

N900?%regtool[0] = 4.1 * 2 - 1
N900?%regtool[0] = 1
N900?%regtool[0] = 1

N900 %regtool[0].0 = 1
N900 %regtool[0].1 = 0
N900 %regtool[0].2 = 1

N1000?VA[0] = 123.0 * -1
N1000?VA[1] = 123.0 * -2

```

```
N1020 VA0 = %regtool[0]
```

where shared memory is named as follows:

```
dword regasync[10]
```

```
dword regtool[10]
```

```
typedef struct{
    RETAIN dword du[10];
} DU;
```

```
DU nc[5];
```

### Addition

Calculates the algebraic sum of two operands.

Syntax

```
exp1 + exp2
```

### Subtraction

Calculates the difference between two operands.

Syntax

```
exp1 - exp2
```

### Negation

Changes the sign of an operand: an operand with a negative value becomes positive and vice versa.

Syntax

```
- exp1
```

### Multiplication

Calculates the product of two operands.

Syntax

```
exp1 * exp2
```

### Division

Calculates the ratio between two operands.

Syntax

```
exp1 / exp2
```

The value of the dividend (exp2 in the example) must be different from 0.

### Modulus of the division

Calculates the remainder of a division.

Syntax

```
MOD(exp1,exp2)
```

Example:

```
VAR1 = MOD(5,2)
```

The result of this division is 2 and the remainder is 1; thus variable VAR1 equals 1.

<b>Raising</b>	Calculates raising of base A with exponent B;
Syntax	$A ** B$ This operator has certain limits as to the values it can use: <ul style="list-style-type: none"><li>• Generates a domain error if A and B are both 0.</li><li>• Generates a domain error if <math>A &lt; 0</math> and B is not an integer.</li></ul>
<b>Square root</b>	calculates the square root of the associated value.
Syntax	<b>SQRT</b> (expr) The expr value must not be negative.
<b>Absolute value</b>	Calculates the absolute value of the associated operand, i.e. if the value of the operand is positive it remains unchanged, otherwise it is made positive.
Syntax	<b>ABS</b> (expr)
<b>Rounding off to the lower integer</b>	Approximates a non-integer value to the lower integer value nearest to the original value.
Syntax	<b>FIX</b> (expr) Example: <code>VARA = FIX( 3.123 )</code> The result of the operation is 3.
<b>Rounding off to the higher integer</b>	Approximates a non-integer value to the higher integer value nearest to the original value.
Syntax	<b>FUP</b> (expr) Example: <code>VARA = FUP( 3.123 )</code> The result of the operation is 4.
<b>Trigonometric operators</b>	<p>The operators listed below use variables or constants that express angles or that provide an angle as a result, as operands.</p> <p>The numeric characteristic of the angles is that their value is periodic, i.e. values beyond the <math>-360^\circ</math> to <math>+360^\circ</math> range can be brought within this range. The operation that brings the value of an angle back within the illustrated range is called angle normalizing.</p> <p>Angle normalizing is obtained by applying the <math>360^\circ</math> modulus function to the angle to normalize.</p>

Syntax	<p><math>\text{ALPHA} = \text{MOD}(\text{BETA}, 360)</math></p> <p>Normalizes the value of the angle in the BETA variable and sets the result in the ALPHA variable.</p> <p>Normalizing is implicit and must not be calculated, i.e. the <math>\text{SIN}(420)</math> and <math>\text{SIN}(60)</math> functions are equivalent.</p> <p>All the parameters are given in degrees.</p>
<b>Arc-Cosine</b>	<p>Calculates the arc-cosine of the associated value.</p> <p>This is the inverse function of the cosine. The value of the parameter must be between <math>-1</math> and <math>1</math>.</p>
Syntax	<p><math>\text{ACOS}(\text{expr})</math></p> <p>The value calculated is an angle.</p>
<b>Arc-Sine</b>	<p>Calculates the arc-sine of the associated value.</p> <p>This is the inverse function of the sine. The value of the parameter must be between <math>-1</math> and <math>1</math>.</p>
Syntax	<p><math>\text{ASIN}(\text{expr})</math></p> <p>The value calculated is an angle.</p>
<b>Arc-Tangent</b>	<p>Calculates the arc-tangent of the associated value.</p> <p>This is the inverse function of the tangent. The value of the parameter must be between <math>-\infty</math> and <math>+\infty</math>.</p>
Syntax	<p><math>\text{ATAN}(\text{expr})</math></p> <p>The value calculated is an angle.</p>
<b>Cosine</b>	<p>Calculates the cosine of the associated value.</p>
Syntax	<p><math>\text{COS}(\text{expr})</math></p> <p><i>Expr</i> is an angle and the value calculated is between <math>-1</math> and <math>1</math>.</p>
<b>Sine</b>	<p>Calculates the sine of the associated value.</p>
Syntax	<p><math>\text{SIN}(\text{expr})</math></p> <p><i>Expr</i> is an angle and the value calculated is between <math>-1</math> and <math>1</math>.</p>
<b>Tangent</b>	<p>Calculates the tangent of the associated value.</p>
Syntax	<p><math>\text{TAN}(\text{expr})</math></p> <p><i>Expr</i> is an angle and the value calculated is between <math>-\infty</math> and <math>+\infty</math>.</p>
<b>Logic operators (Boolean)</b>	<p>The characteristic of the logic operators is that they only evaluate two values of a variable.</p> <p>The values considered are Boolean values, TRUE or FALSE.</p>



The TRUE value is associated with a variable if its content is not zero. The FALSE value is associated if its content is zero.

The result of a comparison always gives a Boolean result, TRUE or FALSE.

### **Compares if equal**

This operator is significant if used in an expression within a conditioned jump instruction (**IF**).

Compares the two operands and gives TRUE if they are equal or FALSE if they are not equal.

#### **Syntax**

```
expr1 == expr2
IF( expr1 == expr2 ) JMP .....
```

### **Compares if more or equal**

This operator is significant if used in an expression within a conditioned jump instruction (**IF**).

Compares two operands and gives TRUE if the first is the same or more than the second, or FALSE if it is less.

#### **Syntax**

```
expr1 >= expr2
IF( expr1 >= expr2 ) JMP .....
```

### **Compares if more**

This operator is significant if used in an expression within a conditioned jump instruction (**IF**).

Compares two operands and gives TRUE if the first is more than the second, or FALSE if it is less or if the two are equal.

#### **Syntax**

```
expr1 > expr2
IF( expr1 > expr2 ) JMP .....
```

### **Compares if different**

This operator is significant if used in an expression within a conditioned jump instruction (**IF**).

Compares the two operands and gives TRUE if they are different or FALSE if they are equal.

#### **Syntax**

```
expr1 != expr2
IF( expr1 != expr2 ) JMP .....
```

### **Compares if less or equal**

This operator is significant if used in an expression within a conditioned jump instruction (**IF**).

Compares two operands and gives TRUE if the first is the same or less than the second, or FALSE if it is more.

#### **Syntax**

```
expr1 <= expr2
IF( expr1 <= expr1 ) JMP .....
```

### **Compares if less**

This operator is significant if used in an expression within a conditioned jump instruction (**IF**).

Compares two operands and gives TRUE if the first is less than the second, or FALSE if it is more or if the two are equal.

**Syntax**

```
expr1 < expr2
IF( expr1 < expr1 ) JMP .....
```

**And**

Allows several logic expressions to be compiled, thus complex test conditions can be expressed in the conditioned jump.  
The AND operator gives TRUE if both the operators it associates are TRUE. It gives FALSE in all other conditions.

**Syntax**

```
Expr1 && Expr2
```

The following table outlines the results that can be obtained by applying the AND operator.

A	B	Expr1 && Expr2
TRUE	TRUE	TRUE
TRUE	FALSE	FALSE
FALSE	TRUE	FALSE
FALSE	FALSE	FALSE

A conditioned jump instruction with 2 conditions associated with an AND operator is illustrated below.

```
IF (Var1 > Var2) && (Var3 < Var4) JMP .....
```

**Or**

Allows several logic expressions to be compiled, thus complex test conditions can be expressed in the conditioned jump.  
The OR operator gives TRUE if at least one of the operators it associates is TRUE. Only if both are FALSE will the OR operator give FALSE.

**Syntax**

```
expr1 || expr2
```

The following table outlines the results that can be obtained by applying the OR operator.

Expr1	Expr2	Expr1    Expr2
TRUE	TRUE	TRUE
TRUE	FALSE	TRUE
FALSE	TRUE	TRUE
FALSE	FALSE	FALSE

A conditioned jump instruction with 2 conditions associated with an OR operator is illustrated below.

```
IF (Var1 > Var2) || (Var3 < Var4) JMP .....
```

**Negation**

The logic value of an expression is varied by means of this operator.

The operator does not give TRUE if the operand to which it is associated is FALSE and gives FALSE if the operand is TRUE.

**Syntax**

**!** expr1

The following table outlines the results that can be obtained by applying the NOT operator.

Expr1	! Expr1
TRUE	FALSE
FALSE	TRUE

A conditioned jump instruction with 1 NOT condition is illustrated below.

```
IF NOT ( A = B ) GOTO . . . . .
```

**3.2.8 Use of #QNAN**

#QNAN (*Quiet Not A Number*) is a particular configuration a variable can assume to indicate that no tolerated numeric value is present. For example, #QNAN will be found in fixed cycle parameters that have not been explicitly programmed and for which a default value is not available. Since the ISO language does not allocate an address if this contains a #QNAN, instructions that move a set of axes that depend on the combination of programmed parameters without having to check all the combinations one by one, can be programmed within the fixed cycles, for; e.g. if in the following line within a fixed cycle:

```
G0 XVA1 YVA2
```

...VA1 and VA2 are parameters, this line will only be automatically interpreted as above if both the values have been explicitly programmed, or:

```
G0 XVA1
```

or:

```
G0 YVA2
```

...depending on whether *only the first variable* or *only the second variable* are present, and so forth.

If a #QNAN variable is subjected to operations, they will always give #QNAN as a result.

Boolean comparisons between variables of which at least one is a #QNAN, give unforeseeable results. This is why comparing a number with #QNAN cannot be done with something of the IF type ( a == QNAN() )!!! Use the ISQNAN(n) for this purpose, e.g.:

```
IF ( ISQNAN( VA2 ) ) . . .
```

To create a #QNAN use QNAN().

### 3.3 Program flow control

#### 3.3.1 Unconditional jump

**Syntax**

*JMP* number  
*JMP* label

*JMPF* number  
*JMPF* label

**Description**

Allows modification of the program execution sequence by directing execution to the block identified by 'number' . or from the label, Labels are identified by **.alphanumerical string**.  
 Use of the JMPF instruction instead of JMP speeds up the cycle. JMPF only tolerates jumps to lines that follow the current line, i.e. it only tolerates jumps forward.

**Example**

```
N100 ;error signal
N110 G04 F1 ;wait
N120 JMP 110 ;infinite loop
dX=0
.LOOP
  G0 X(100+dX) Y(210+dX)
  dX = dX + 0.5
IF (dX < 10 ) JMP .LOOP
```

**See also**                      **Test, Standard programming: Block number**

#### 3.3.2 Test

**Syntax**

*IF* (expression == expression) <functions> ;equal  
*IF* (expression != expression) <functions> ;not equal  
*IF* (expression > expression) <functions> ;greater than  
*IF* (expression < expression) <functions> ;less than  
*IF* (expression >= expression) <functions> ;greater than or equal to  
*IF* (expression <= expression) <functions> ;less than or equal to  
*IF* (!expression) <functions> ;equal to zero  
*IF* (expression) <functions> ;not equal to zero

**Description**

Compares two expressions and executes the rest of the block if the logical function returns true.  
 N.B.: comparison must be done between commensurable expressions (sign, dimension). In other words a 16-bit signed word (for example a %QW) cannot be compared to an unsigned word (for example 0xFFFF written explicitly). To make the comparison the signed variable must be cast as follows:

```
;IF(%QW2 == 0xFFFF) JMP 110 incorrect!!!

IF((%QW2 & 0xFFFF) == 0xFFFF) JMP 110 ; correct
```

### Example

```
N100 VA1 = 100.0 ;initialise variable
N200 G0 XVA1 ;position X
N300 M80 ;execute machining sequence
N400 VA1 = VA1 + 10.0 ;increment variable
N500 IF (VA1 < 205.0) JMP 200 ;repeat 10 times, then
      continue

IF(%C2.1==1) JMP 100
IF((%C2 && %C3 && %C4) || %C2.1 || %C2) JMP 110

IF(%IW1 > 100) JMP 10
; IF(%QW2 == 0xFFFF) JMP 10 incorrect!!!
IF((%QW2 & 0xFFFF) == 0xFFFF) JMP 10 ; correct
IF(%QW2 == 0x1B) JMP 10
IF(%QW2 == %cnerr.err) JMP 10
IF(%QW2.15) JMP 10
IF(%C2.1 == 1) JMP 10
```

**See also**                      **Conditioned Test**

### 3.3.3 Conditioned Test

**Syntax**                      *IF* condition *THEN*  
                                 Block 1  
                                 *ELSE*  
                                 Block 2  
                                 *ENDIF*

**Description**                      Checks the condition. If this is true, the following instruction block will be executed (block 1 of the syntax). If the condition is false, the instruction block that follows the ELSE key will be executed (block 2 of the syntax, if present). The process joins up after the key word ENDIF.  
 A block of instructions within an IF-THEN-ELSE can contain other IF-THEN-ELSE instructions. The maximum nesting level is 5.

### Example

```
IF( MyVar > 100 ) THEN
  G0 X100 Y 100
  G4 F2
  M19 S45
  MyVar = 0
ELSE
  MyVar = MyVar + 1
  G0 Z0
ENDIF
```

**Also see**                      **Test**

### 3.3.4 WHILE Block

**Syntax**

```
WHILE (condition)
    Block
ENDW
```

**Description**

Executes the block of instructions for as long as the expression remains true. As soon as the condition becomes false, the controls passes on to the block that follows *ENDW*. If the condition is false at the first iteration, the block will not even be executed once.

**Example**

```
WHILE( MyVar != 100 )
    G0 X100 Y 100
    G4 F2
    M19 S45
    MyVar = MyVar+1
ENDW
```

Since this block of instructions is evaluated by a pre-processing stage carried out before accomplishment of the machine movement in real time, it is advisable to use conditions that cannot vary in real time. This construction is used to create parametric iterations but not to block the processing phase until an event occurs (e.g. activation of an input signal). The following example is not an example of good programming.

**Example of improper use**

```
G0 X100 Y100
WHILE( "%IW0.0" == 0 )
    G4 F1
ENDW
```

This example shows how a physical input is tested in the pre-processing phase. The previous instruction will be executed as soon as the real-time executor examines it. The *WHILE* block therefore has time to reiterate several times before the *G0* block is executed, waiting for the event to occur. Every time the *WHILE* cycle is reiterated, the process prepares a *G4* instruction to send to the executor block, while the pre-processing block continues to iterate without a pause, waiting for the end of iteration event.

One of the ways to synchronize the pre-processing block with the executor is to use a *SYN* instruction.

```
G0 X100 Y100
WHILE( "%IW0.0" == 0 )
    SYN
    G4 F1
ENDW
```

This method blocks the pre-processing stage until the execution stage has accomplished each instruction. In the example, this means waiting for the *G0* block to have terminated before executing block *G4*. In the successive iterations, a new *G4* block will only be sent

to the executor when this has finished with the previous one. In actual fact, a time setting is included in the pre-processing block.

**Also see**                      **REPEAT, FOR, BREAK**

### 3.3.5 REPEAT Block

**Syntax**                      *REPET*  
                                     Block  
                                     *UNTIL* (condition)

**Description**                      Executes the block of instructions for as long as the expression remains false. As soon as the condition becomes true, the controls passes on to the block that follows UNTIL. This construction guarantees execution of the block of instructions it contains at least once. In actual fact, the block is first executed and its condition is only evaluated afterwards. If the condition is true, the block will not be reiterated. Vice versa, it will be executed again until the escape condition occurs.

#### Example

```
REPEAT
  G0 X100 Y 100
  G4 F2
  M19 S45
  MyVar = MyVar+1
UNTIL ( MyVar == 100 )
```

Since this block of instructions is evaluated by a pre-processing stage carried out before accomplishment of the machine movement in real time, it is advisable to use conditions that cannot vary in real time. This construction is used to create parametric iterations but not to block the processing phase until an event occurs (e.g. activation of an input signal). The following example is not an example of good programming.

#### Example of improper use

```
G0 X100 Y100
REPEAT)
  G4 F1
UNTIL ( "%IW0.0" != 0 )
```

This example shows how a physical input is tested in the pre-processing phase. The previous instruction will be executed as soon as the real-time executor examines it. The WHILE block therefore has time to reiterate several times before the G0 block is executed, waiting for the event to occur. Every time the WHILE cycle is reiterated, the process prepares a G4 instruction to send to the executor block, while the pre-processing block continues to iterate without a pause, waiting for the end of iteration event.

One of the ways to synchronize the pre-processing block with the executor is to use a SYN instruction.

```

G0 X100 Y100
REPEAT
  SYN
  G4 F1
UNTIL ( "%IW0.0" != 0 )

```

This method blocks the pre-processing stage until the execution stage has accomplished each instruction. In the example, this means waiting for the G0 block to have terminated before executing block G4. In the successive iterations, a new G4 block will only be sent to the executor when this has finished with the previous one. In actual fact, a time setting is included in the pre-processing block.

**See also** **WHILE, FOR, BREAK**

### 3.3.6 FOR Block

#### Syntax

```

FOR index=start TO end BY step
  Block
ENDFOR

```

Where:

**index** is the name of the variable that uses the number of iterations as a counter

**start** is the initial value of the counter

**end** is the final value of the counter, i.e. the condition that determines the end of the iteration

**step** this is the increase applied to the counter on each iteration. It can also have decimal and/or negative values. In this case, start must be greater than end.

#### Description

Repeats the block of instructions for the number of iterations required to bring the **index** variable from the **start** value to the **end** value, increasing it on each iteration of the **step** value.

If the "**BY step**" part is omitted, the process assumes that the increase is unitary.

#### Example

The following example accomplished 25 iterations (from 0 to 100 at step 2.5)

```

DBL ind;

FOR ind=0 TO 100 BY 2,5
  G0 X100+(ind*2) Y(ind/2)
  G1 X(ind/2) Y100+(ind*2)
  G4 F (ind/10)
ENDFOR

```

The following example is identical to the previous one but uses a unitary increase, thus makes 100 iterations.

```

DBL ind;

FOR ind=0 TO 100 BY 2,5
  G0 X100+(ind*2) Y(ind/2)
  G1 X(ind/2) Y100+(ind*2)
  G4 F (ind/10)
ENDFOR

```



**See also**                    **WHILE, REPEAT, BREAK**

### 3.3.7    Cycle interruption - BREAK

**Syntax**                    ***BREAK***

**Description**              Interrupts any iteration cycle, bringing the execution to the first instruction that follows the iteration cycle that contains **BREAK**.  
Interruption of a nested iteration cycle with one or more iteration cycles will not stop the iteration of the outermost cycles.  
The **BREAK** instruction can be used for **FOR**, **WHILE** and **REPEAT** cycles.

**Example**

```
DBL ind;
FOR ind=0 TO 100 BY 2,5
  G0 X100+(ind*2) Y(ind/2)
  G1 X(ind/2) Y100+(ind*2)

  IF( %C0.0 == 1 ) BREAK

  G4 F (ind/10)
ENDFOR
```

**See also**                    **WHILE, REPEAT, FOR**

### 3.3.8 Repeat

#### Syntax

**RPT** start block, end block, number of repeats

#### Description

Allows repetition of a specified block sequence for the specified *number of repeats* . The sequence of blocks to repeat is defined by the *start block* and *end block*. Start and end blocks can be identified by:

- numeric constant;
- expression
- label.

Number of repeats. Can be specified by an expression  
Up to 3 levels of nesting are allowed.

#### Example

```
G0 X0 Y0 Z200
G0 B0           ;position table at 0°
RPT 10, 70, 1   ;repeat N10-N70 once
10, 70, 1       ;repeat N10-N70 once
G0 B45          ;position table at 45°
RPT G0 B90 B45  ;position table at 90°
RPT 10, 70, 1   ;repeat N10-N70 once

;(Definition of technical parameters for drilling)
N10 T1 M6
N20 G81 DR125 DE100 DH125 FW150 FR15000
N30 X10 Y10      ;drilling
N40 G80          ;cancel G81 mode
N50 T1 M6
;(Definition of technical parameters for tapping)
N60 G84 DR125 DE100 DH125 RW150 RR15000
N70 RPT 30, 30, 1 ;executes tapping
                    ;at the same position where
                    ;drilling has been done

N100 M30
```

#### See also

See use with **Fixed system cycles and macro instructions**

### 3.3.9 Nesting levels: sub-programs and fixed cycles

#### Nesting levels: sub-programs and fixed cycles

Compilation of a program can be made modular by using the calls to fixed cycles or subroutines (sub-programs). Besides making the code more legible, this type of programming enables already written code parts to be reused. A sub-program can recall another sub-program through to a maximum of 5 nested calls.

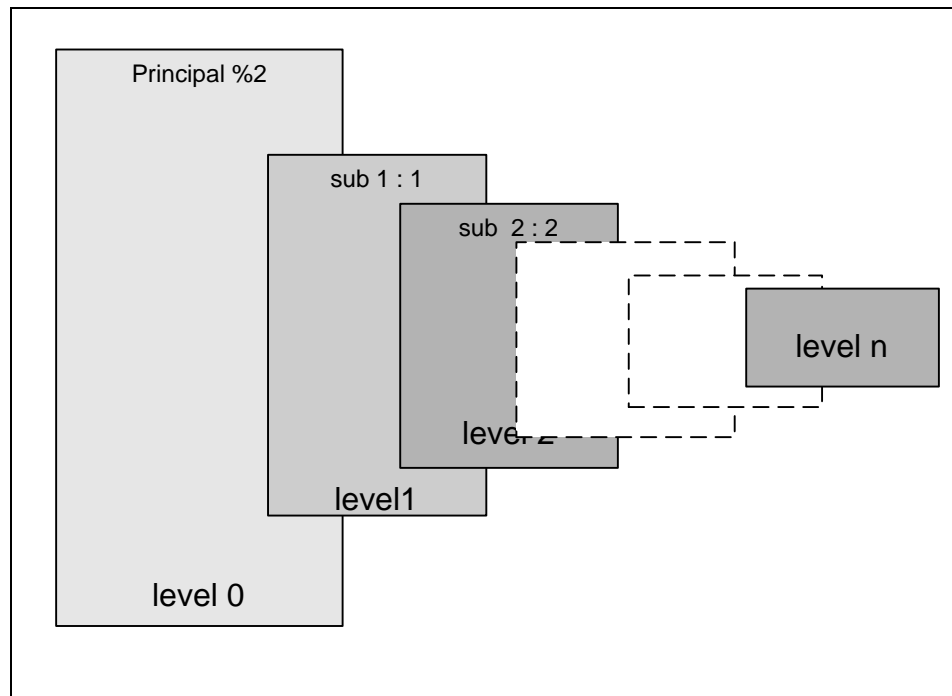


Figure 3.1 -

### 3.3.10 Subprogram start (fixed cycle)

#### Syntax

*prototype: fixed cycle name*  
where:

*prototype = var1,var2, ...*

*var n =[variable]default value or variable*

#### Description

The start subprogram block (fixed user cycle) contains a *comment*, the start subprogram character ":" and the *subprogram number* (fixed cycle name). User fixed cycles (*filename.cfu*) can be freely written by the user.

The following description only regards the system fixed cycles.

The subprogram start block (system fixed cycle) contains the *prototype*, the subprogram start character ":" and the *subprogram number* (fixed cycle name). System fixed cycles (*filename.cfs*) are factory set. Each subprogram starts with a start block.

**Prototype** Variables can be listed in any order. They can have any single-letter address (so that up to 26 variables (A-Z) are allowed). Contain all the numerical values, except for T and E, which can be followed by a string in brackets.

Variables with numerical values are passed to the fixed cycle by automatic variables of type *double* VA0 => VA25.

**Number** A unique identifying number which corresponds to the partprogram  
**fixed cycle** number.

A variable declared as **self-cancelling** in the prototype must be declared between square brackets, and *when called* can be:

- by value => the actual value is passed to the fixed cycle;
- omitted => if the prototype contains a *default value* for this variable, the default value is passed; if *no default value* is specified in the prototype, *Nan* is passed (**Not A Number**).

If the variable is assigned the value *Nan*, a special ISO command option can be used: when an ISO address is followed by *Nan*, the address is ignored. If a feed command for a given axis is followed by *Nan*, for example, the axis is ignored (it can therefore not be one of the currently available axes as listed in the machine parameters). If a G, M or F address is followed by *Nan*, the feed or PLC command or speed setting is *not* executed (in the case of feed commands, any following commands with valid addresses are correctly executed in the active mode). This type of functionality, with *Nan* values, has been extended to the following addresses:

**G, M, T, D, F, S,**  
**X, Y, Z, U, V, W, A, B, C,**  
**R, I, J, K,**  
**TGR** (the three values which follow TGR), **SVR, SVL,**  
**CHA** (chamfer), **RAD** (radius),

A **modal** parameter in the prototype must not be in square brackets and, *when called*, can be:

- by value => the actual value is passed to the fixed cycle;
- omitted => if this is the first call of that fixed cycle at that level of nesting, an error is returned, if *not*, then the last evaluated value for that fixed cycle at that level of nesting is returned.

Note that this mode is maintained only at the given level of nesting.

A prototype must be inserted before the symbol:.

### Example

A sample part program follows (at nesting level 0):

```
N30 F100
N40 G17 G0 X 0 Y 0
N50 G100 B 20.0 E "string" C 30.0 N197
      B 60.0 ;modal call of G100
N58 G80          ;cancel modality of G100
N59 JMP 197      ;jump forward two lines
N60 G01 X VL100 Y VL104
N197 M10
N199 M30
N200 END
```

fixed cycle G100(nesting level 1):

```
[B] C E [F] 30.0 [G] [N]:100 ;prototype
G1 X VA1 Y VA2 ;uses values 20.0, 30.0 of variables B and C
G1 X VA6 Y VA1 A VA6 ;uses Nan, 30.0, Nan of parameters G,
C, G
CALL MrlChangeTool(3,10) ;call C
VL104 = 33
VL100 = 66
G101 A 10 B 11 C 12
RET
```

and fixed cycle G101(level of nesting 2):

```
[A] 10.0 B [C] 25.2:101 ;prototype
G1 X VA0 Z VA1 ;uses values 10.0, 11.0 of parameters A and
B
RET
```

### See also

Call to subprogram, CFU (User fixed cycle) Advanced Programming.

### 3.3.11 Subprogram end (fixed cycle)

#### Syntax

***RET***

#### Description

Returns execution to the calling program.

### 3.3.12 Program end

**Syntax** *END*

**Description** Terminates program execution.

### 3.3.13 Call to subprogram CFU (user fixed cycle)

**Syntax** *JSR* number expression.1 expression.2 ... expression.n  
*JSR* "name" expression.1 expression.2 ... expression-.n  
*JSR* variable expr. 1 expr.2..expr.n

**Description** The named subprogram is executed before passing to the next block in the linear sequence.

The subprogram can be specified by

- from the number;
- from the alphanumerical name between inverted commas "";
- from the content of a *string* type of variable (symbolic system variable -%nomesub- or symbolic variable defined by the user).

Variables can be passed to the subprogram by value; the numerical values of the variables are located by the subprogram in the automatic registers VA0, VA1, ... VAn, in the order in which they are defined.  
 Up to 5 levels of nesting are possible.

**Example**

Main program:

```
STR NOMESUB
JSR 1 10.0 VA12 (VA10+VA20)
```

```
Drilling subprogram: 1
G0 XVA1 YVA2 ;feed to hole position
G1 ZVA0 F1000 ;execute drilling
G0 Z100 ;retract tool from piece
RET ;return to main program
```

```
NOMESUB = "holes"
JSR NOMESUB 10.0 VA12 (VA10+VA20)
JSR "holes" 10.0 VA12 (VA10+VA20)
```

```
Drilling subprogram: holes
G0 XVA1 YVA2 ;feed to hole position
G1 ZVA0 F1000 ;execute drilling
G0 Z100 ;extract tool from piece
RET ;return to main program
```

**See also** Subprogram start (fixed cycle), Subprogram end (fixed cycle)

### 3.3.14 Subprogram call CFS (system fixed cycle)

<b>Syntax</b>	<p><i>G</i> number <i>var</i> expr <i>var</i> expr ...</p> <p>where</p> <p><i>var</i>: [A, B, ..., Z] variable in the prototype</p>
<b>Type of function</b>	Modal
<b>Description</b>	<p>The named subprogram is executed before passing to the next block in the linear sequence.</p> <p>The number following the G code indicates the subprogram in the user-defined system directories.</p> <p>The variables which appear in this block must be among those declared in the fixed cycle prototype, otherwise an error is returned. Variables not explicitly passed to the subprogram are assigned their default values (self-cancelling variables) or their previous values (modal variables).</p> <p>The variables are accessed by the fixed cycle via the automatic variables VA0..VA26. VA0 contains the value of variable A, VA1 the value of variable B, etc.</p> <p>Up to 5 levels of nesting are possible.</p> <p>After the first call of a fixed cycle only those variables which have changed relative to the first call need be passed explicitly in successive calls.</p>
<b>Example</b>	<p>Main program:</p> <pre> N30 F100 N40 G17 G0 X 0 Y 0  N50 G100 B 20.0 E "string" C 30.0 N197       B 60.0 ;modal call of G100  N58 G80          ;cancel G100 mode N59 JMP 197      ;jump forwards two lines N60 G01 X VL100 Y VL104 N197 M10 N199 M30 N200 END  [B] C E [F] 30.0 [G] [N]:100 ;prototype </pre>
<b>See also</b>	<p><b>Subprogram start (fixed cycle), Subprogram end (fixed cycle), Cancels a fixed cycle's mode</b></p>

### 3.3.15 Cancels a fixed cycle's mode

**Syntax** *G80*

**Description** In the blocks following a fixed cycle call, declaring one or more variables involved in the fixed cycle causes it to execute.

G80 cancels the fixed cycle mode, that is, it allows execution of successive blocks without automatically calling the fixed cycle itself

In the same way it is possible to cancel a fixed cycle mode with G0, G1, G2 etc...

**Example**

Main program:

```
N30 F100
N40 G17 G0 X 0 Y 0

N50 G100 B 20.0 E "string" C 30.0 N197
    B 60.0 ;modal call of G100

N58 G80          ;cancel G100 mode
    B 45.0 ;position polar axis B

or
G0 B 45.0 ;position polar axis B

[B] C E [F] 30.0 [G] [N]:100 ;prototype
```

**See also** Subprogram start (fixed cycle), Subprogram end (fixed cycle)



### 3.3.16 Call-back function in “C” language

**Syntax**

**CALL** *<function name>* (<var1>, <var2>,...)

<var>=<expression>|^<output variable>

**Description**

Where:

*function name* is the name of the function in "C".

The parameters passed to the function are listed between round brackets (max 32 parameters).

The fixed cycle C access the parameters by means of a parameter counter and a list of the parameters themselves, respectively *iArgc* and *Args*. *Args* specifies the type and value of each parameter. This information allows checking whether the passed parameter is type-compatible with the function; the correctness of the call can thus be checked by the cycle itself.

Finally, it will have access to the ISO channel descriptor and a limited series of functions .

**Example**

```
CALL MrlPtiDrill (3, 10)
```

**See also**

### 3.3.17 Forced escape from the interpreter with error

**Syntax** ERROR( err\_nr )

**Description** This instruction stops program execution and displays the error message that depends on the **err\_nr** value.  
It can be used to signal faults that have occurred in the automation part created in ISO language.  
The **ERROR** only has a direct effect on the channel that is executing the program that invoked it. Other channels in parallel execution continue their execution unless there is an intervention from the PLC.  
The error code can be from 0 to 9999, even though it is advisable to use values over 768 to prevent conflicts with signals used by the firmware of the CNC.  
To customize the message to associate with the error, customize the RunXXX.err file, where XXX means the three-letters that distinguish the language of the translation (ITA, GER, FRA, ENG ...). The generated alarm is managed in exactly the same way as the alarms generated autonomously by the NC.

**Esempio**

```
IF( (VL10 < 1000) AND (VL10 > 0) ) THEN
  G0 X VL10
ELSE
  ERROR( 800 )
ENDIF
```

In the example, the field of existence of the VL10 variable before using it for positioning. A fault is signalled if the variable is beyond the field of existence.

## 3.4 Special functions

### 3.4.1 Await termination

**Syntax** *SYN*

**Type of function** Self-cancelling

**Description** Halts program calculations until the preceding block has terminated execution. If this command is not specified, program calculations always run several blocks ahead of actual execution for higher output. This command is specified in particular to ensure that the information used in parameters (e.g. information from axis position sensors or limit switches) are checked in realtime and not in advance.

**Example**

```
N100 G1 G91 X-0.5 Y-0.5 ;approach by one step
      SYN ;await axis positioning
      IF (VG500 == 0) JMP 100 ;if the limit switch has not been
      tripped, approach by a further step
```

### 3.4.2 Message display in phase with the executor

**Syntax** \$ (text)

**Type of function** Self-cancelling

**Description** Displays the text between brackets in synchrony with execution of the blocks that precede it.

### 3.4.3 Memory read/write

**Syntax** *VA, VL, VG* parameter\_number = %*identifier*  
[?]%*identifier* = *VA, VL, VG* parameter\_number

**Description** Enables reading and writing to a register in shared memory .  
[?] indicates writing synchronous with execution.  
Note that the unit of measurement used by the whole value variable is crucial, in the case of a linear axis position it is a thousandth of a mm ( $\mu\text{m}$ ), hence, to convert to a floating point value it must be divided by 1000.

**Example**

```
?%MYMEAS = VA11
```

**See also** Enter a physical or logical output

### 3.4.4 Enter a physical or logical output

**Syntax**

```
?%QW <output number> = 1 ;sets the physical output to synchronous mode
?%QW <output number> = 0 ;resets the physical output in synchronous mode
[?]%C <output number> = 1 ;sets the logical output in synchronous or asynchronous mode
[?]%C <output number> = 0 ;resets the logical output in synchronous or asynchronous mode
```

**Description** The character '?' indicates that the operation must be executed synchronously with the other blocks of the program.

**?%QW10.4 = 1** ; sets the bit 5 of physical output 10

**?%QW10.4=0** ; resets bit 5 of physical output 10

**?%QW10=10** ; assigns the value 0x01010 to physical output 10

Writing to physical outputs must be synchronous with execution to ensure the integrity of the IO states which can also be accessed by the PLC.

**?%C10.2 = 1** ; sets bit 3 of logical output 10

**?%C10.2 = 0** ; resets bit 3 of logical output 10

**?%C10= 14** ; assigns the value 0x01110 to logical output 10

The value of a physical or logical input can be tested with an IF statement.

Writing to physical registers (%QW) must be done synchronously (so that the registers are not overwritten by the PLC refresh cycle). Writing to logical registers (%C) can also be asynchronous if the register in question is not used by the PLC.

### Example

```
?%C2.0 = 1
?%C2 = 13
%C2.1 = 0
%C2 = 14
?%QW2.1 = 1
```

```
IF(%QW2 > 100) JMP 110
```

```
IF(%IW2.1==1) JMP 200
```

```
IF(%C2.1==1) JMP 100
```

```
IF((%C2 && %C3 && %C4) || %C2.1 || %C2) JMP 120
```

**See also** Reading and writing memory's register

### 3.4.5 Enabling hydraulic tapping

**Syntax** *ADP*[Axis, Mode]

**Description** This command is intended to resolve problems deriving from the non-linearity of the hydraulic axes. Tapping can be reduced to a linear interpolation between a linear axis and a spindle; the non-linear behaviour of a hydraulic axis, especially during acceleration and deceleration, and the fact that the tool and the piece are rigidly coupled, results in defective machining. This problem is surmounted by the use of a self-learning algorithm which monitors the realtime error on the two axes; this is then used to correct the piston control voltage to reduce the errors to a minimum.

Axis Hydraulic axis name .

Mode This can have one of the following values:

- **INIT** initialises the state of the function
- **START** synchronises the start of the self-learning cycle
- **STOP** stops the self-learning cycle

**Example**

```
ADP[Z,INIT]  
ADP[Z,START]  
G63 S1200 Z0 K5.0 ; tapping forwards  
G63 S1500 Z80 K-5.0 ; tapping reverse  
ADP [Z,STOP]
```

### 3.4.6 Defining path axes

<b>Syntax</b>	<code>CON [Axis1, Axis2, ..., Axisn] = 1 ; couple ON</code> <code>CON [Axis1, Axis2, ..., Axisn] = 0 ; couple OFF</code>				
<b>Type of function</b>	Modal				
<b>Description</b>	<p>CON[<i>Axis list</i>] = 1: Couple ON  Updates the register that contains the <b>mask of path axes</b>, entering 1 on a level with the axes specified in the <i>Axes List</i>.</p> <p>CON[<i>Axis list</i>] = 0: Couple OFF  Updates the register that contains the <b>mask of path axes</b>, by resetting the bits on a level with the axes specified in the <i>Axes List</i>.</p> <p>The Path Axis Template contains:</p> <table> <tr> <td>1</td><td>on a level with the axes that must be torqued since they are involved in the programmed interpolation;</td></tr> <tr> <td>0</td><td>on a level with the axes that are not involved in the programmed interpolation;</td></tr> </table> <p>Axis list</p> <p>A list of the axes which must be coupled for execution of the programmed interpolation.</p> <p>When the channel is reset the coupled axes coincide with the axes configured in the channel.</p> <p>N.B. The CON commands (just like G16) have no direct electromechanical consequences; these are handled by the PLC: CON simply instructs the PLC to couple/uncouple the axes in question. Vice versa, the PLC could initiate the operation and communicate the fact to the ISO channel which uses the CON to adjust internally.</p>	1	on a level with the axes that must be torqued since they are involved in the programmed interpolation;	0	on a level with the axes that are not involved in the programmed interpolation;
1	on a level with the axes that must be torqued since they are involved in the programmed interpolation;				
0	on a level with the axes that are not involved in the programmed interpolation;				
<b>Example</b>	<pre> N100 G0 X100 Y200 Z300 ;linear3d N200 CON[Y] = 0 N300 X200 ;linear2d N400 CON[Z] = 0 N500 X300 ;single N600 CON[Y,Z] = 1 N700 Z0 ;linear3d </pre>				

### 3.4.7 Definition of the Axes belonging to the Channel

<b>Syntax</b>	<p><i>GRP</i> [Axis1, Axis2, ..., Axis n] = 1 ; Axes associated with the channel</p> <p><i>GRP</i> [Axis1, Axis2, ..., Axis n] = 0 ; Axes disassociated from the channel</p>
<b>Type of function</b>	Modal.
<b>Description</b>	<p><i>GRP</i>[<i>Axes List</i>] = 0</p> <p>The axes in the list are released from the channel that will no longer use them. Successive instructions that require a previously released axis to be moved, cause an error signal.</p> <p><i>GRP</i>[<i>Axes List</i>] = 1</p> <p>declares that the channel is managing the list of axes defined by the instruction. The list of defined axes adds to the axes that were already being managed by the channel. The axes to which the process refers must be associated with the channel in the static mode in the channel configuration table. This command is used to reset a previous releasing action of one or more axes (<i>GRP</i>[..]=0).</p>
<b>Shared axes</b>	<p>This command is used to handle resources shared amongst several channels. In this case, there must be a configuration in which two or several channels have one or more shared axes.</p> <p>An axis is shared by several channels when the same logic number (system wide) appears in the static configuration of several channels. The shared axes can have different names in different channels (e.g. X shared in CH 0 and U in CH 1).</p> <p>The automation must be designed (part program and PLC) so that each channel only links to an axis when this is really necessary (<i>GRP</i>[ax]=1) and releases it as soon as it is no longer required (<i>GRP</i>[ax]=0).</p> <p>Processing of a <i>GRP</i> function causes the axes associated with the channel (<i>CO_PATAX</i>, <i>cn</i>[..]<i>rc</i>[30]) to be publicized besides activating a strobe signal (<i>M_STB_GET</i>, <i>cn</i>[..]<i>rc</i>8.17). execution is suspended while awaiting the acknowledge synchronism of the PLC (<i>M_ACK_GET</i>, <i>cn</i>[..]<i>rc</i>0.29).</p> <p>See the “<i>Interface registers</i>” manual for greater details.</p>
<b>Example</b>	<pre>G0 X100 Y200 Z300 ;linear3d GRP[Y] = 0 G1 X200 Z100 Z0 X0 GRP[Y] = 1 G0 X100 Y200</pre>

### 3.4.8 Correction by zones

**Syntax** *CZO[Axis] = 0* ;correction by zones disabled  
*CZO[Axis] = 1* ;correction by zones enabled

**Type of function** Modal

**Description** Correction by zones reduces errors due to mechanical non-linearity in the drive. 255 zones (defined by 256 points) of equal length are available, between a minimum and maximum value, for each of which suitable correction can be specified. Positions below the programmed minimum are corrected with the value of the first programmed point; positions above the programmed maximum are corrected with the value of the last programmed point.

Only after a first self-learning phase, in which the compensation values are memorised, can correction by zones be activated. In this mode, each calculated position is displaced by the zone-relative compensation.

#### Example

```
Z AXIS COMPENSATION LEARNING CYCLE %1101

CZO[Z] = 0 ; Disable correction by zones

VA0 = 0 ; Minimum position [mm]
VA1 = 1000 ; Maximum position [mm]
VA2 = 255 ; Number of zones
VA3 = ( VA1 - VA0 ) / VA2 ; Size of zone i
VA4 = 0 ; Sample index
VA5 = 0 ; Increment dimension
VA10 = %ax[2].pa[40] ; Associated CZone Buffer index

G0 ZVA0 AVA0

N100 POS[Z] = VA5 POS[A] = VA5 FA[Z] = 10000 FA[A] = 10000
N200 SYN
N300 VA6 = QTA(Z)
N400 VA7 = QTA(A)
N500 %czone[VA10].sam[VA4] = ( VA7 - VA6 ) * 1000
N600 VA4 = VA4 + 1
N700 VA5 = VA5 + VA3

RPT 100, 700, VA2

%czone[VA10].qtamin = 0
%czone[VA10].qtamax = VA1 * 1000
%czone[VA10].nzone = VA2

M10 ; Couple Off
CZO[Z] = 1 ; Send buffer and enable correction by zones
M11 ; Couple On
M30
```



### 3.4.9 Enabling program simulation

**Description**

The channel sets to TEST mode and executes:

- a syntax check of the program lines
- parameter calculations
- radius and length compensations
- machining feasibility analysis(Path allowed by CUT, SW limit positions)
- runs a parametric flow check (IF, JMP)

**Example**

```
/N100 IF (VG500 != 0) JMP 1000
```

### 3.4.10 Identifier of the number of the channel being executed

**Syntax**                      ChanNr = WHO();

**Description**                      Gives the number of the channel in which the program is running.  
This function must be used for generic programs that, however, must distinguish between one channel and the other.

**Example**

```
DBL ChanNr, ChanSpeed;

ChanNr = WHO()

    ; Rear speed programmed for channel
ChanSpeed = %cn[ChanNr].rc[26]

    ;Distinguish cycle for channel 0 and for other channels
IF( ChanNr == 0 ) THEN
    .....
ELSE
    .....
ENDIF
```

### 3.4.11 Probe target reading

**Syntax**                      Sample =GET (Ax);

**Description**                      Gives the target acquired by a touch probing cycle for the indicated axis.  
**Ax** can have one of the envisaged axes names (X, Y, Z, U, V, W, A, B, C).  
 When an automation language command is used (`_singfc(..)`), this freezes the target of the axis to which it refers when an event occurs (making or breaking of a logic or physical contact). After this, The GET function is used to make the frozen value available for parametric calculations.  
 A typical use is for acquiring information about the position of the workpiece to be machined and for then re-processing calculations about the machining process.

**Example**

```
DBL ChanNr, ChanSpeed;

_singfc( VMAX, X, 2, 1000, 0, "%IW1.15", 0 )

SYN
VA0 = GET( X )
SFH[X] = VA0
```

In the example, a positioning is accomplished with X axis until the contact associated with the IW1.15 input is released. The X axis target is frozen as soon as this event occurs. The SYN instruction that follows, forces suspension of the pre-processing stage until the probing cycle is completed (`_singfc` command). In this way, the GET function only accesses the sampled value when this has really been acquired. Without the SYN instruction, GET would work in advance of the probing function, causing a non-valid value to be acquired.

### 3.4.12 Axis target reading

**Syntax**                       $VLO = QTA(Ax)$      ; Theoretic Target  
                                   $VLO = QCALC(Ax)$  ; Theoretic Target  
                                   $VLO = QMEAS(Ax)$  ; Real Target

**Description**                      These three functions give the targets of the axes to which they refer. The first two functions have the same name, thus both give the theoretic value of the axis. The third gives the measured target.  
 The fact that the first two functions have the same name is due for reasons of compatibility with old programs.  
 The value given by all three functions is expressed in millimeters, while the data item is the double precision type.

**Example**

```
DBL QTeor, QReal, FwError;
```

```
SYN
QTeor = QCALC(X)
QReal = QMEAS(X)
FwError = ABS(QTeor - QReal)
```

An IMPROPER function is made for an ISO channel in the example, i.e. the follow error is calculated by acquiring the theoretic target and measured target of X axis. The operation is improper because the axis will certainly change its position between the two acquisitions. It is therefore only considered for explanatory purposes. The instantaneous value of the follow error is published in real time in a register of the NC.

## 3.5 Reference system transformations

### 3.5.1 PRS - Preset machine origin

**Syntax** *PRS*[Axis] = <expr>

**Description** Loads the specified value (position) into the axis position register (this is also called immediate setting). The effect is to force the axis origin to the value given by the expression.  
The machine origin preset operation does not depend on the MLV level, that is, whenever a new axis origin is set, it applies to every level of transformation of the MLV coordinates.

**Example**

```
PRS[X]=100
PRS[Y]=100 ;forces the machine origin to the point
P0(100,100)
G0 X200 Y200 ;feed to point P1(X200,Y200) of the new
;system of reference
```

**See also** **ROT.**

### 3.5.2 DEF - Run-time redefinition of the axis names

**Syntax** **DEF** [first axis name] = <second axis name>

**Type of function** Modal

**Description** Maps the *second axis* onto the *first axis* name. From the next block after the DEF command, whenever the name of the *first axis* occurs, it is interpreted as the *second axis*. To redefine the *first axis* to be the *first axis*, use the following syntax:  
*DEF* [<first axis name>] = <first axis name>

**Example**

```
DEF[X] = U
G33 Z-32.9 X 7 K 3 ;interpreted as G33 Z-32.9 U 7 K 3
DEF[X] = X
```

**See also** **SYS.**

### 3.5.3 SYS – Run-time re-allocation of the axes name

**Syntax**                      **SYS** [name of first axis] = <name of second axis>

**Type of function**            Modal.

**Description**                Re-allocates the name of the *first axis* to the name of the *second axis*. From the block following that of SYS programming onwards, the interpreter reads the name of the *first axis* as though it were the *second axis*. Use the following syntax to re-define the *first axis* as *first axis*:

*SYS* [<name of first axis>] = <name of first axis>

Unlike the DEF instruction, SYS is sensitive to the matrix conversion established with MLV, i.e. if a SYS is programmed as MLV=2, if MLV=1 is entered it will be disabled. Moreover, SYS is able to correct the direction of circular interpolations when the axes of the contouring plane are switched with each other.

**Example**

```
SYS[X] = U
G33 Z-32.9 X 7 K 3 ;interpreted as G33 Z-32.9 U 7 K 3
SYS[X] = X
```

**Also see**                      **DEF, ROT.**

### 3.5.4 MIR - Mirrored machining

**Syntax**                      **MIR** [axis name] = 0, 1

**Type of function**            Modal

**Description**                *MIR* [axis] = 1  
Enter mirroring working as regards axis. Mirroring is programmed inverting axis direction  
*MIR* [axis] = 0  
cancels mirroring for that axis

When mirroring is active the programmed direction of rotation is automatically inverted (G2, G3) as well as the programmed tool radius compensation (G41, G42). Once a piece contour has been machined it can be mirrored using the MIR instruction. This results in more readable code.

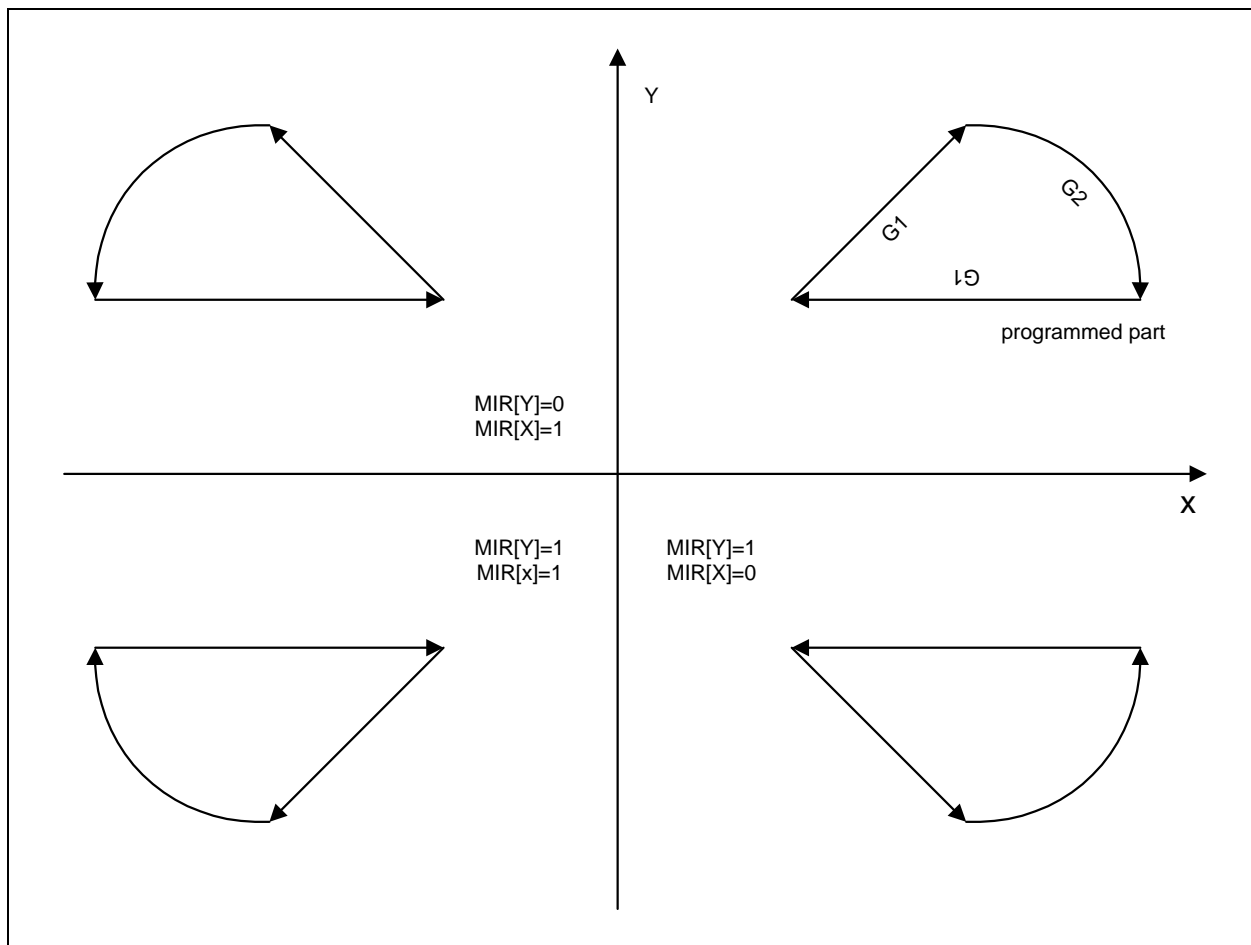


Figure 3.2 - Mirrored machining

**Example**

```
Mirroring test %1

F60000 D1

JSR "part1"

MIR[X]=1
JSR "part1"

MIR[Y]=1
JSR "part1"

MIR[X]=0
JSR "part1"

M30

;CAUTION: to ensure that the direction
;of radius compensation is correct
;compensation itself must be activated
;and deactivated in the part machining sequence

mirroring:part1
G41
G0 X0 Y0
G1 X500 Y500
G1 X1000 Y1000
G2 X1500 Y500 R500
G1 X500
G40 G0 X0 Y0
RET
```

**See also****ROT MLV.**



### 3.5.5 ROT – Rotation of the contouring plane

#### Syntax

***ROT[plane] = angle***

The rotation plane must be selected from amongst:

- EA      Rotation around X axis
- EB      Rotation around Y axis
- EC      Rotation around Z axis

The angle is given in degrees

#### Type of function

Modal.

#### Description

Enters an angle of rotation that will be applied to axes moving instructions that follow ROT.

Rotation occurs in relation to the workpiece origins. Any transfers from the origin (obtained with SHF instructions) will be applied after rotation. If transfer is to be carried out first and then rotation, this latter must be programmed with a lower matrix level (MLV) than the one used to program the transfer.

#### Example

```
ROT[EC]=45
SHF[X]=100
G0 X 0 Y 0 Z 0 F 3000
G1 X 300 F 3000
G1 X 0 Y 300
G1 X 0 Y 0
M30
```

On the other hand, if a transfer is to be carried out first and then rotation

```
ROT[EC]=45
MLV=1
SHF[X]=100
G0 X 0 Y 0 Z 0 F 3000
G1 X 300 F 3000
G1 X 0 Y 300
G1 X 0 Y 0
M30
```

#### See also

**MLV, SHF.**

### 3.5.6 SHF – Transfer of Origin

**Syntax** *SHF[Axis name] = target*

It only applies to axes X, Y and Z.

**Type of function** Modal.

**Description** Transfers the reference system. It is applied at activated conversion matrix level. Programming transfers at different conversion matrix levels means applying all the transfers programmed in the levels between 0 and *i* to the *i*-esimal level. Matrix level selection ( $MLV=k$ ) involves application of the sole transfers activated between level 0 and level *k*. Transfers programmed at levels beyond *i* and *k* are not lost. This means that restoration of a deeper matrix level reactivates the transfers through to level *i*.

**Example**

```
G0 X0 Y0 F300
MLV = 1
SHF[X] = 100
G1 X0 Y0
MLV = 2
SHF[Y] = 100
G1 X0 Y0
MLV = 0
G1 X0 Y0
MLV = 2
G2 X0 Y0 I-50 J-50
M30
```

The following interpolation occurs in the example:

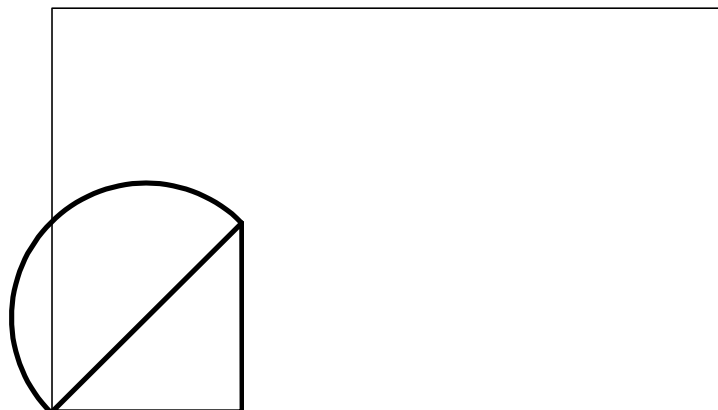


Figure 3.3 -

### 3.5.7 MLV - Selection of the matrix conversion level.

**Syntax** *MLV = 0...5*

**Type of function** Modal

**Description** MLV enables locking the systems of reference obtained from the transformation instructions SHF, MIR and SYS and also resetting them.  
By default MLV level 0 is active, and programmed transformations are applied to the machine's system of reference if no other levels are specified.

Programming *MLV=n* (*n* new level) all successive transformations apply to level *n*. Each time the system passes to a new level the new level inherits the system of reference defined in the preceding level, hence new transformations are summed to those already applied.

Up to 6 MLV levels are available, from 0 to 5.

Part origin translations (G54...G57) and local origin translations (G58, G59) are active no matter what MLV level is currently active.

#### Example

```

; LEVEL 0
SHF[X] = 500 ; X axis translation
MIR[X] = 1   ; mirror X axis
G0 X100 Y100
G0 Y200
G0 X200
G0 Y100
G0 X100

; LEVEL 1: inherits the mirroring and translations
previously applied to X

MLV=1
SHF[Y]=300 ; Y axis translation
MIR[Y]=1   ; mirror Y

MIR[X]=1 ; compensates the X axis mirroring inherited
; from LEVEL 0

G0 X100 Y100
G0 Y200
G0 X200
G0 Y100
G0 X100

;LEVEL 0
MLV=0 ; resets the level 0 coordinate system
; SHF and MIR only on X

SHF[X]=0 ;cancels the X axis translation
MIR[X]=0 ;cancels the level 0 mirroring
G0 X100 Y100
G0 Y200

```

```
G0 X200  
G0 Y100  
G0 X100
```

```
M30
```

**See also**

**SYS DEF.**

### 3.5.8 Definition of a tilting plane

**Syntax**

$$\begin{aligned} MAT[asse1] &= [a11 \ a12 \ a13 \ a14] \\ MAT[asse2] &= [a21 \ a22 \ a23 \ a24] \\ MAT[asse3] &= [a31 \ a32 \ a33 \ a34] \end{aligned}$$

**Type of function** Modal.

**Description** A tilting plane is defined by programming the conversion matrix obtained from the known geometric formulas.  
Three terms (lines) of the matrix , one for each axis of the Cartesian triad:

$$\begin{aligned} MAT[X] &= \dots \\ MAT[Y] &= \dots \\ MAT[Z] &= \dots \end{aligned}$$

These can be expressed in any order: The number of columns is 4 to also specify the rotation fulcrum, i.e. the offset of the tilting plane in relation to the reference system.

After a tilting plane has been defined, copying profiles are programmed as though the process were taking place on the XY plant (G17).

#### Example

```
DBL a11, a12, a13, a14
DBL a21, a22, a23, a24
DBL a31, a32, a33, a34
DBL alfa = 45

; Printing < DROTX >
a11 = 1.0 a12 = 0.0      a13 = 0.0      a14 = 0.0
a21 = 0.0 a22 = COS(alfa) a23 = -SIN(alfa) a24 = 0.0
a31 = 0.0 a32 = SIN(alfa) a33 = COS(alfa) a24 = 0.0

MAT[ X ] = [ a11 a12 a13 a14 ]
MAT[ Y ] = [ a11 a12 a13 a14 ]
MAT[ Z ] = [ a11 a12 a13 a14 ]

G41
G61
G0 X0 Y0 Z0
G1 X300 F3000
G2 X0 Y0 I150 J0 F1000
```

**See also** **DYNMOD**

### 3.5.9 Tilting plane enabling

**Syntax** *DYNMOD = 0/1*

**Description** This function enables (DYNMOD=1) or disables (DYNMOD=0) control of the conversion matrixes for defining a tilting plane.  
If a matrix is disabled, it need not be re-programmed when successively restored.

**Example**

```

DBL a11, a12, a13, a14
DBL a21, a22, a23, a24
DBL a31, a32, a33, a34
DBL alfa = 45

; Printing < DROTX >
a11 = 1.0 a12 = 0.0      a13 = 0.0      a14 = 0.0
a21 = 0.0 a22 = COS(alfa) a23 = -SIN(alfa) a24 = 0.0
a31 = 0.0 a32 = SIN(alfa) a33 = COS(alfa) a24 = 0.0

MAT[ X ] = [ a11 a12 a13 a14 ]
MAT[ Y ] = [ a11 a12 a13 a14 ]
MAT[ Z ] = [ a11 a12 a13 a14 ]

G41
G61
G0 X0 Y0 Z0
G1 X300 F3000
G2 X0 Y0 I150 J0 F1000

DYNMOD = 0      ; ; MATRIX disabling

G41
G61
G0 X0 Y0 Z0
G1 X300 F3000
G2 X0 Y0 I150 J0 F1000

DYNMOD = 1

G0 X1000 Y1000
G1 X600 F4000
G3 X1000 I800 J1000

M30

```

## 3.6 Automatic plane geometry

### 3.6.1 Chamfer between two linear segments

**Syntax** *CHA* <expr>

**Type of function** Self-cancelling

**Description** Machines a chamfer between two linear segments. The size of the chamfer (expressed after the CHA instruction) is the distance between the two new corners resulting from machining of the chamfer itself.

**Example**

```
N100 G01 X100.0 CHA 10.0  
N200 G01 X200.0 Y 100.0
```

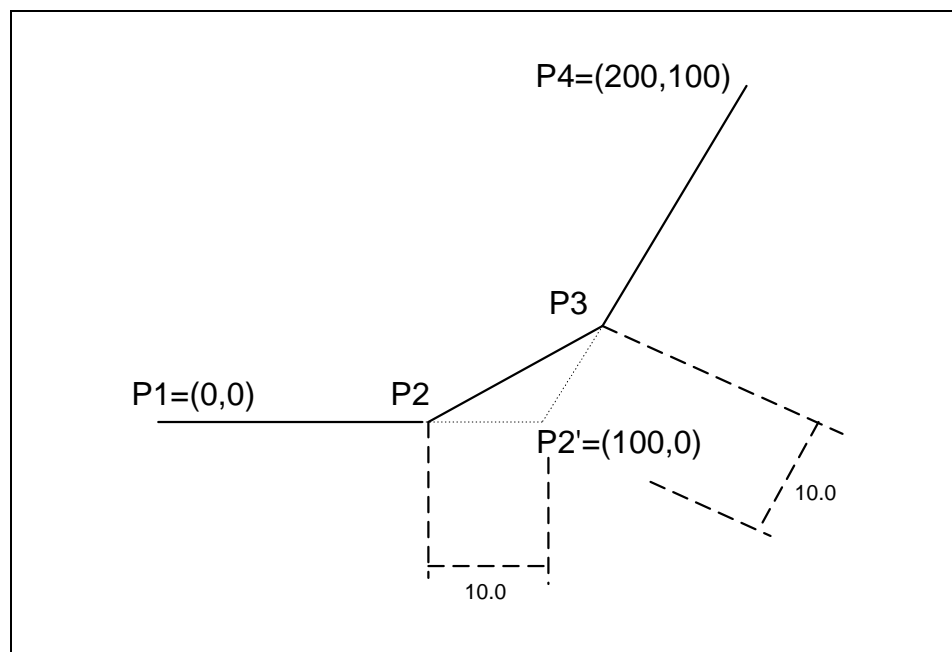


Figure 3.4 - Chamfer between two linear segments

**See also** Radius between two linear segments.

### 3.6.2 Radius between two linear segments

**Syntax** *RAD* <expr>

**Type of function** Self-cancelling

**Description** Machines a radius between two linear segments. The size of the radius (expressed after the RAD instruction) is the geometric radius of the arc which forms the cut. The radius, by definition, is tangential to both linear segments.

**Example**

```
N100 G01 X100.0 RAD 10.0  
N200 G01 X200.0 Y 100.0
```

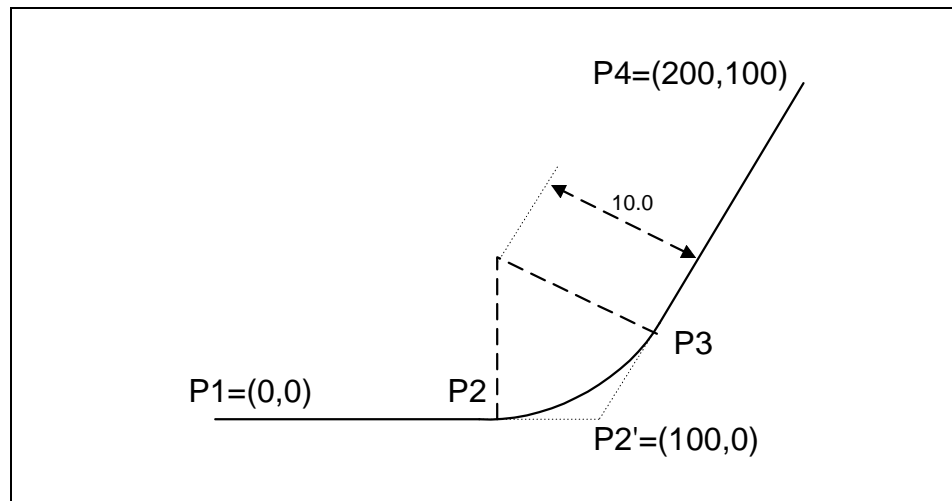


Figure 3.5 - Radius between two linear segments

**See also** Chamfer between two linear segments.



### 3.7 Tool corrector auxiliary functions

#### 3.7.1 Machining allowance management

<b>Syntax</b>	$SVR <expr>$ $SVL <expr>$
<b>Type of function</b>	Modal
<b>Description</b>	<p>SVR: radial or contour machining allowance.  SVL: length or depth machining allowance.</p> <p>SVR: programs the machining allowance on the machined contour.  The machining allowance modifies the tool radius compensation value (if enabled) (G41, G42) → Tool Radius + SVR.  Disabled by Svr 0.0 or G40 which disables tool radius compensation.</p> <p>SVL: programs the machining allowance on the machining depth axis, that is the length-compensated axis.  Activated by the selection of a D corrector, applies a length compensation equal to SVL orthogonal to the selected contour milling plane (G17, G18, G19).  Disabled by D0 or Svl 0.0</p>
<b>Example</b>	<p><b>N.B.</b> SVL 1 - the space is important !</p> <pre> D1 F2000 G18 G42 <b>SVR 10.0</b> G00 X250.0 Z250.0 G01 X250.0 Z400.0 G40 G0 X0 Z0 </pre>
<b>See also</b>	Selecting the machining plane, Selecting the contour milling plane and the direction of length correction, Tool length correction, Tool radius compensation.

#### 3.7.2 Tangential feed in/out

<b>Syntax</b>	$TGR K<Kr> <Alfa> <Beta>$ $TGR R<r> <Alfa> <Beta>$
<b>Type of function</b>	Modal
<b>Description</b>	<p><b>K</b> <i>Kr</i> is the number of tool radii which determines the radius of curvature of the tool feed in,  <b>R</b> <i>r</i> is the radius of curvature of the tool feed in  <i>Alfa</i> is the width of the feed in arc,</p>

*Beta* is the width of the feed out arc.

Program TGR to enable tangential feed in/out. This is automatic if the tool corrector is active.

Tangential feed in is executed every time G0 is followed by G1, G2 or G3, so that the tool describes an arc before engaging with the part surface tangentially so as not to damage it. The arc depends on the feed in angle (*Alfa*) and the radius of curvature.

In the same way tangential feed out is executed every time G1, G2 or G3 are followed by G0.

To disable tangential feed in/out, program

TGR K0 0 0

or G40 which deactivates the tool corrector.

### Example

TGR K3.0 45.0 30.0 (feed in at 45°, feed out at 30°, width three tool radii)

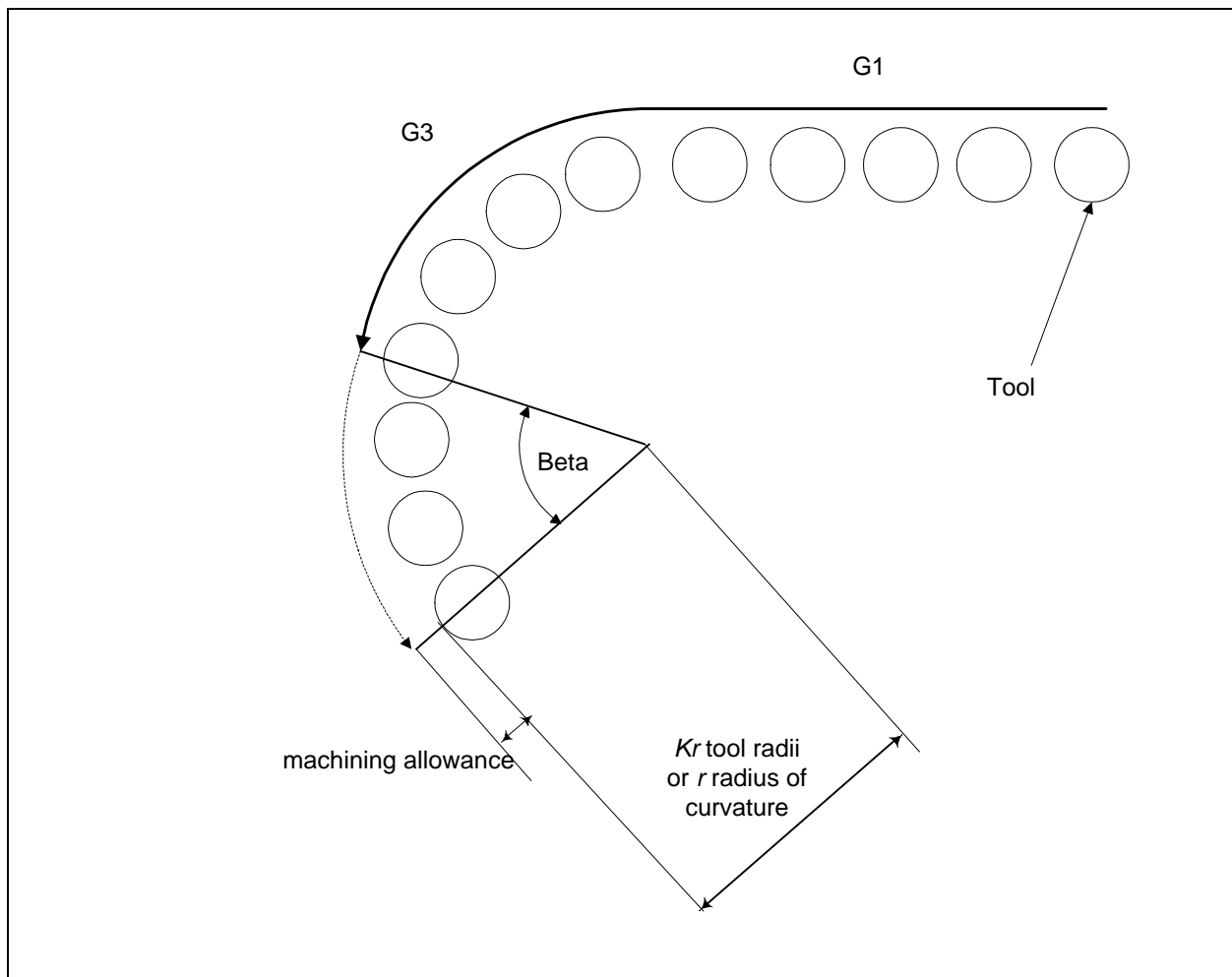


Figure 3.6 - Tangential radius

### 3.7.3 Tool life and wear management

**Syntax** *VTL* <expression>

**Description** Increments the value in the Current Tool Life field (of the selected corrector) by the value of the expression.  
If the value is nil (VTL0) the Current Tool Life field is zeroed.  
Handles tool life in the Counting mode using two fields in the corrector record:

- Preset Tool Life
- Current Tool Life

An alarm is generated when the Current Life is > than the Preset Life.

Each field that geometrically characterizes the tool is associated with a register for making fine corrections to these dimensions.

These compensations will be enabled:

- from selection of corrector Dn (length compensation),
- from modal functions G41/G42 (Tool radius compensations).

This compensation is manual in release 0.0.

**Example**

```
D1
G17 G1 X100 Y200
VTL1
```

### 3.8 Three dimensional tool corrector

#### 3.8.1 TWI: Spindle settings

**Syntax** *TWI* number

**Description** Declares the current spindle.

**Example**

*TWI* 9

#### 3.8.2 THD: Tool holder settings

**Syntax** *THD* number

**Description** Declares the current tool holder.

**Example**

*THD* 16

#### 3.8.3 Conventions regarding angles of rotation

We use the following convention:

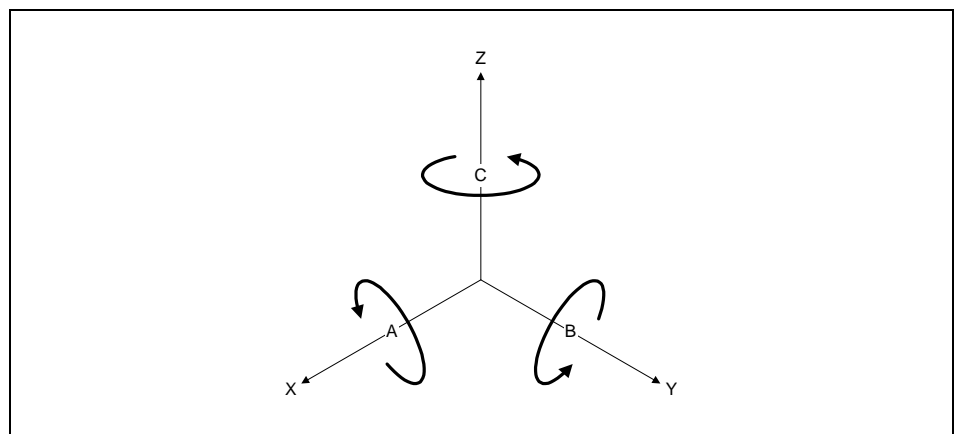


Figure 3.7 - Conventions for programming angles

### 3.8.4 Orientation with polar axis coordinates

Name		Significance	
		Rotary Head	Fork Rotary Head
<i>A</i>	Depends on feed	(rotation around Z axis)	(rotation around Z axis)
<i>B</i>	Depends on feed	(rotation around X axis)(*)	(rotation around X axis)

(\*) In reality the *B* axis cannot be programmed, because in this configuration there is no physical actuator capable of driving it. The set position must correspond to the tool set up angle; this is used in a machining feasibility analysis.

If a head has been selected with TWI, *A* and *B* are referred to the actuator axes for that head and are taken into account by the three dimensional tool compensation. To drive the physical *A* and *B* axes, use *TWI0*.

If the actuator axes cannot reach the specified positions an error is returned.

Matrix transformations cannot be applied to the coordinates of the polar axes. To apply transformations (e.g.: to rotate the angle when a rotation/translation is active) the tool orientation must be programmed with a notation which is independent of the actual feed instructions (e.g.: Euler's angles).

#### Example

```
TWI4THD2
G0 X10.00 Y10.00 Z8.00 A0.00 B45.00
```

#### See also

**Orientation by Euler's angles, Orientation by roll pitch yaw (RPY), Orientation by the tool axis unit vector.**

### 3.8.5 Orientation by Euler's Angles

#### Syntax

*MVD*=EUL (default)

#### Description

Setting the mode of interpretation of *EA*, *EB*

*EA* = <expr>

*EB* = <expr>

*EA* → Rotation around Z axis, parallel to the tool axis.

*EB* → Rotation around X axis.

To vary the angle without changing the tool feed in point, the contour milling axis coordinates must be programmed at the same positions; otherwise no movement occurs.

Use *G16* to reset axial compensation.

**Example**

TWI4THD2

**MVD=EUL**G0 X10.00 Y10.00 Z8.00 **EC0.00 EA45.00**

The head programmed with TWI can be of any type with a tool holder pointing at the upper face of the part and with the polar axes in the reference position.

Face	Programming	
<i>upper</i>	EA0.00	EB0.00
<i>front</i>	EA0.00	EB90.00
<i>right</i>	EA90.00	EB90.00
<i>rear</i>	EA180.00	EB90.00
<i>left</i>	EA270.00	EB90.00

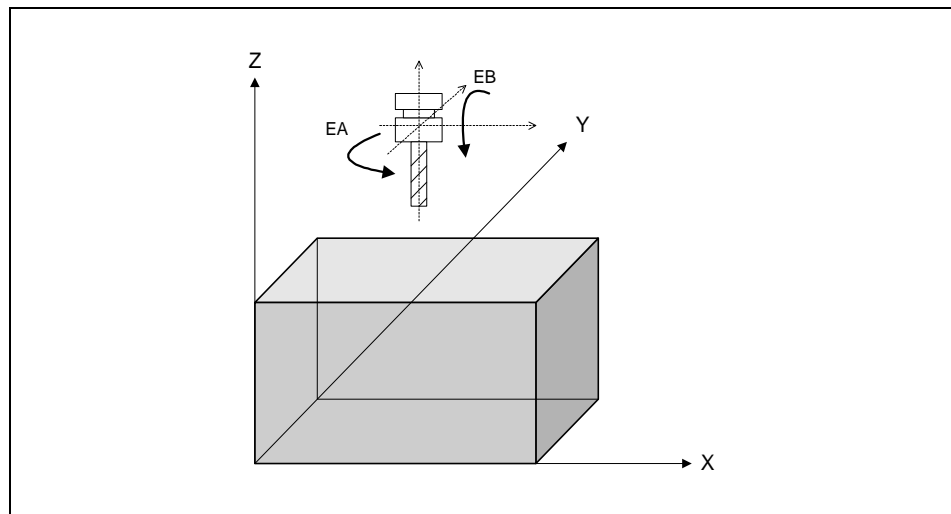


Figure 3.8 - Parallelepiped to which the machining parameters refer

In some applications it may be convenient to combine matrix transformations with these operations to redefine the axes on the various faces and refer them to local origins to facilitate programming.

**See also**

**Orientation with polar axis coordinates, Orientation by roll pitch yaw (RPY), Orientation by the tool axis unit vector..**

### 3.8.6 Orientation by roll-pitch-yaw (RPY)

**Syntax** *MVD* = RPY

**Description** Setting the interpretation of EA, EB, EC

*EA* = <expr>

*EB* = <expr>

*EC* = <expr>

EA → Rotation about X (Roll)

EB → Rotation about Y (Pitch)

EC → Rotation about Z, parallel to the tool axis (Yaw)

To vary the angle without changing the tool feed in point, the contour milling axis coordinates must be programmed at the same positions; otherwise no movement occurs.

If at the next feed command the selected tool cannot reach the specified orientation an error is returned.

Use G16 to reset axial compensation

#### Example

TWI4THD2

**MVD=RPY**

G0 X10.00 Y10.00 Z8.00 **EC0.00 EB17.93 EA24.32**

The head programmed with TWI can be of any type with a tool holder pointing at the upper face of the part and with the polar axes in the reference position.

Face	Programming		
<i>upper</i>	EC0.00	EB0.00	EA0.00
<i>right</i>	EC0.00	EB90.00	EA0.00
<i>rear</i>	EC90.00	EB90.00	EA0.00
<i>left</i>	EC180.00	EB90.00	EA0.00
<i>front</i>	EC270.00	EB90.00	EA0.00

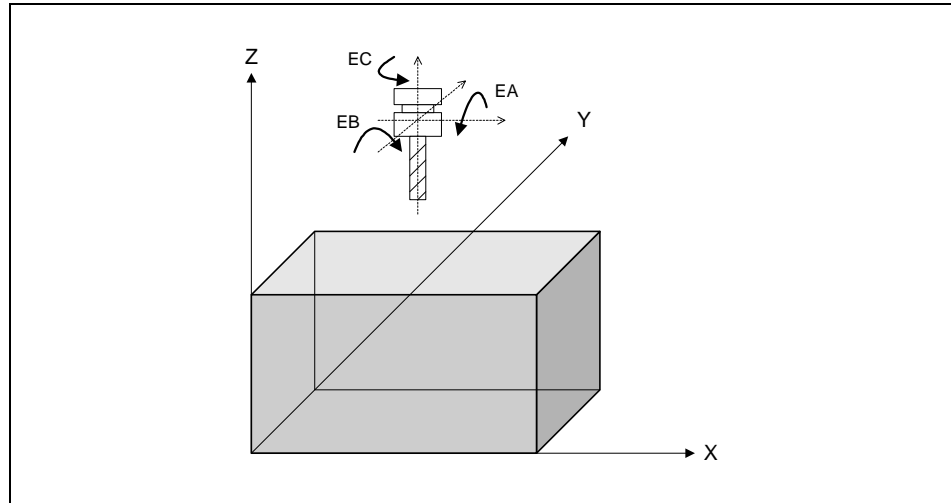


Figure 3.9 - Parallelepiped to which the machining parameters are referred

#### ORDER OF ROTATION:

- 1) EC rotation about Z
- 2) EB rotation about Y
- 3) EA rotation about X

#### See also

**Orientation with polar axis coordinates, Orientation by roll pitch yaw (RPY), Orientation by the tool axis unit vector.**

### 3.8.7 Orientation by the tool axis unit vector

#### Syntax

**EP** <expr>  
**EQ** <expr>  
**ER** <expr>

#### Description

If tool length compensation is activated ( $Dn$ ), the tool orientation must be specified with the components of the tool axis unit vector  $\hat{\mathbf{o}}$ :

EP  $\rightarrow$  X component of the tool axis unit vector

EQ  $\rightarrow$  Y component of the tool axis unit vector

ER  $\rightarrow$  Z component of the tool axis unit vector

Only the orientation of the axis is thus taken into account. This allows us to use the three cosines (known to the user) which specify the direction of the tool axis from the tool point towards the base.

To vary the angle without changing the tool feed in point, the contour milling axis coordinates must be programmed at the same positions; otherwise no movement occurs.

If at the next feed command the selected tool cannot reach the specified orientation an error is returned.

Use G16 to reset axial compensation.



**Example**

```
G0 X10.00 Y10.00 Z8.00 EP0.707 EQ0.707 ER0.00
```

The head programmed with TWI can be of any type with a tool holder pointing at the upper face of the part and with the polar axes in the reference position.

Face	Programming	
<i>upper</i>	EP0.00 ER1.00	EQ0.00
<i>right</i>	EP1.00 ER0.00	EQ0.00
<i>rear</i>	EP0.00 ER0.00	EQ1.00
<i>left</i>	EP-1.00 ER0.00	EQ0.00
<i>front</i>	EP0.00 ER0.00	EQ-1.00

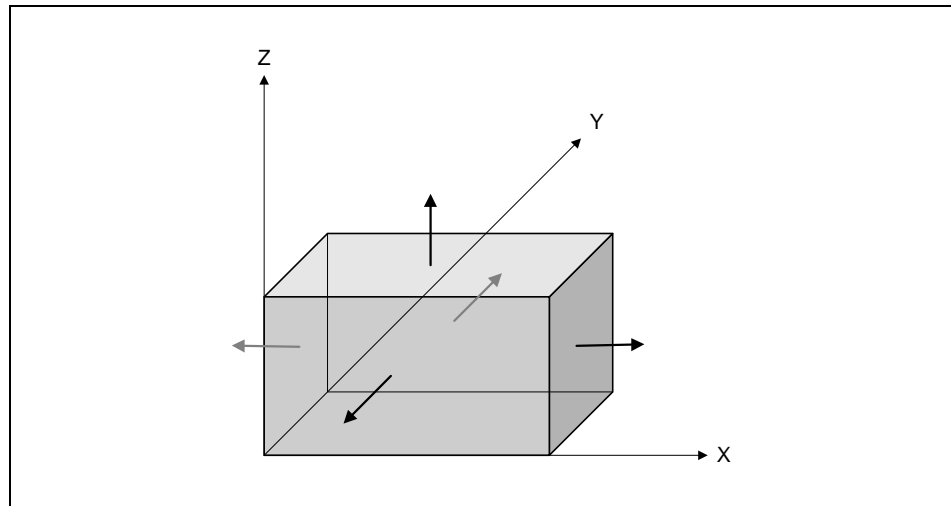


Figure 3.10 - Parallelepiped to which the tool axis unit vector components refer

In some applications it may be convenient to combine matrix transformations with these operations to redefine the axes on the various faces and refer them to local origins to facilitate programming.

**See also**

**Orientation with polar axis coordinates, Orientation by roll pitch yaw (RPY), Orientation by the tool axis unit vector.**

### 3.8.8 Vectorial tool radius compensation

**Syntax**

$EI = \langle \text{expr} \rangle$   
 $EJ = \langle \text{expr} \rangle$   
 $EK = \langle \text{expr} \rangle$

**Description**

Three dimensional tool radius compensation depends on the tool feed in angle. The direction of compensation can be specified explicitly using the machining surface normal vector  $\mathbf{n}$ .

$EI \rightarrow$  X component of the radius compensation vector

$EJ \rightarrow$  Y component of the radius compensation vector

$EK \rightarrow$  Z component of the radius compensation vector

Only the orientation of the vector is taken into account. We can therefore use three quantities which have the significance of being the cosines which give the direction of the tool radius compensation.

If EI,EJ,EK are nil, three-dimensional compensation is cancelled.

**Example**

G1 X10.00 Y10.00 Z8.00 **EI0.707 EJ0.00 EK0.707**

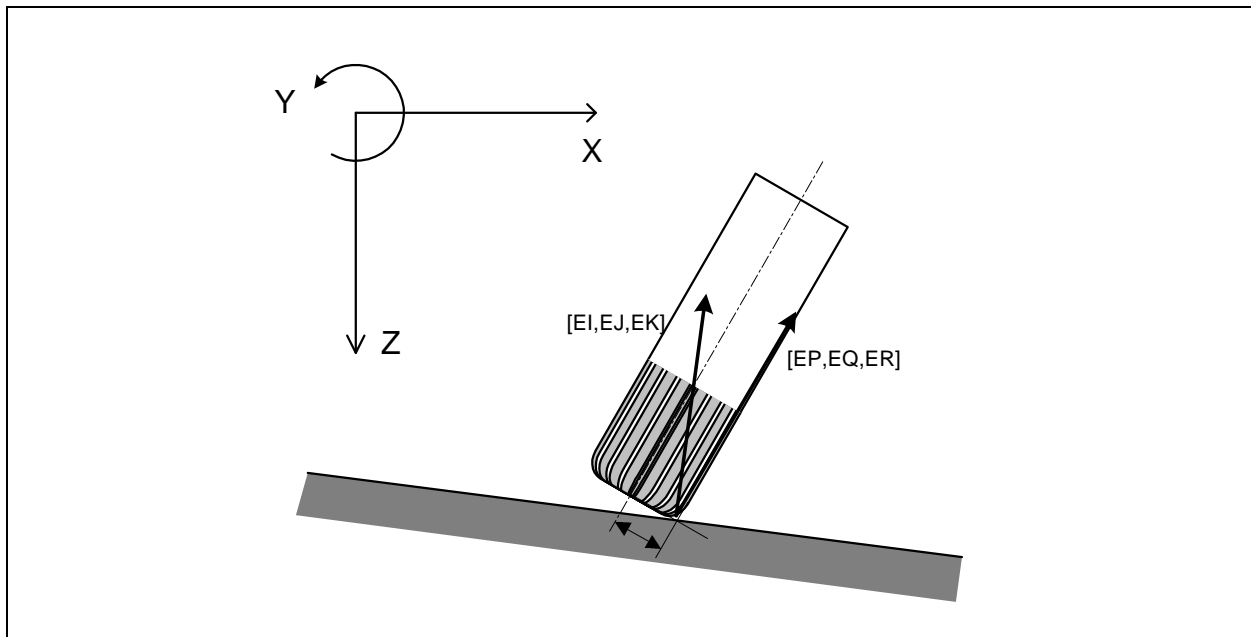


Figure 3.11 - Information required for three dimensional tool compensation: machining surface normal vector and tool axis unit vector

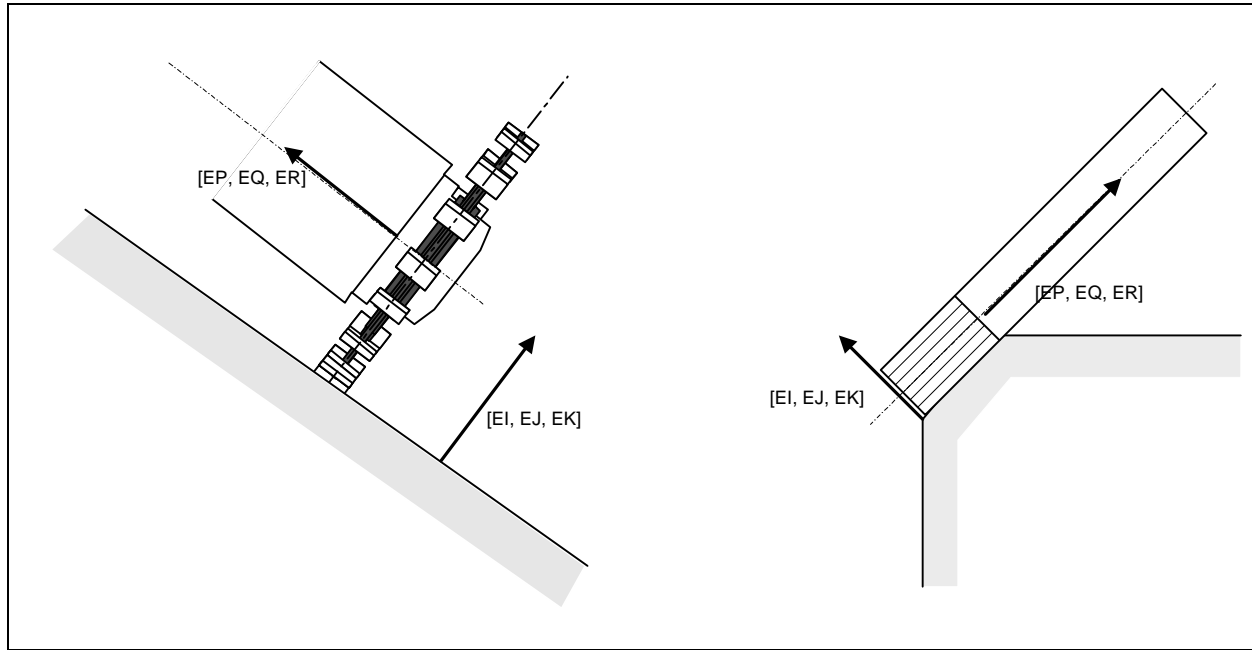


Figure 3.12 - programming example of normal vector  $[EI, EJ, EK]$  to the surface and of the vector  $[EP, EQ, ER]$  that describes tool positioning

### 3.8.9 Automatic determination of the pertinent plane and machining with the axis in tangency

**Syntax** PADSID = 0..4

#### Description

Establishes the position of the tool axis in relation to the programmed path.

To indicate the value of PADSID, an auxiliary reference system must be identified with  $Y'$  axis coinciding with the programmed path,  $X'$  axis square to  $Y'$  and belonging to the selected interpolation plane, and  $Z'$  axis square to the  $X', Y'$  plane. The value of PADSID indicates on which of the four quadrants, determined by axes  $Z', Y'$ , the tool axis lies.

PADSID = 0 tool length compensation not determined automatically

PADSID = 1 the tool axis remains in quadrant I (at the top right)

PADSID = 2 the tool axis remains in quadrant II (at the top left)

PADSID = 3 the tool axis remains in quadrant III (at the bottom left)

PADSID = 4 the tool axis remains in quadrant IV (at the bottom right)

If PADSID is activated, addresses EA and EB have the following meaning, regardless of the MVD selector setting:

EB = Angle the tool forms with the normal interpolation plane

EA = Additional rotation around Z axis (normal in relation to the interpolation plane)

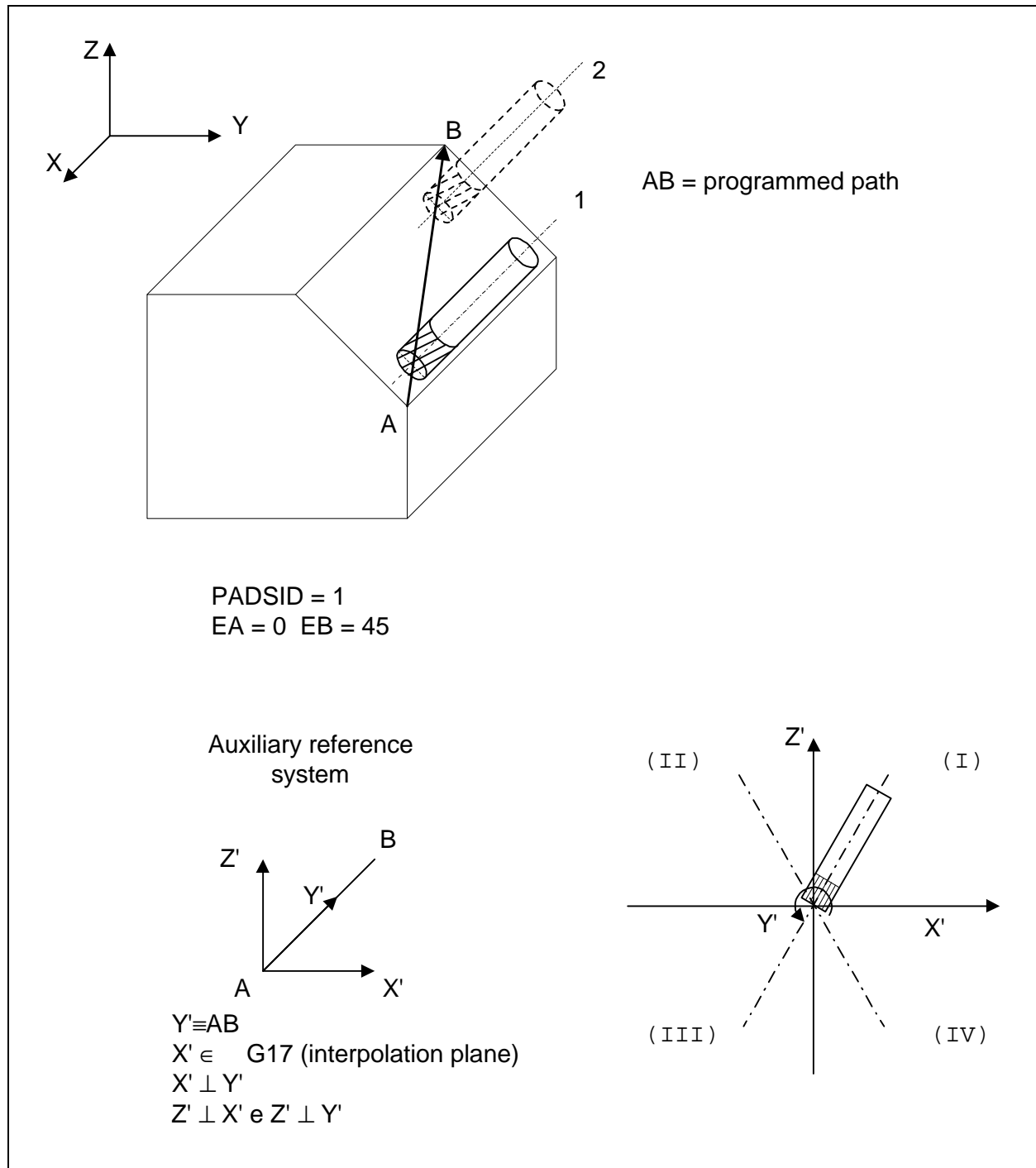


Figura 3.13 -

Certain conditions of indefiniteness or impossibilities may occur when this mode is used: if the plane selected is G17 and a movement along Z axis has been programmed, it will be impossible to determine the auxiliary reference system in an univocal way.

If the EA value programmed is different from 0, the tool is turned the specified angle around the axis in the normal way in relation to the interpolation plane beginning from the self-determined position. Consequently, the tool axis will no longer be normal in relation to the machining process (with the exception of singularity with EB at 0 or at 90 degrees).

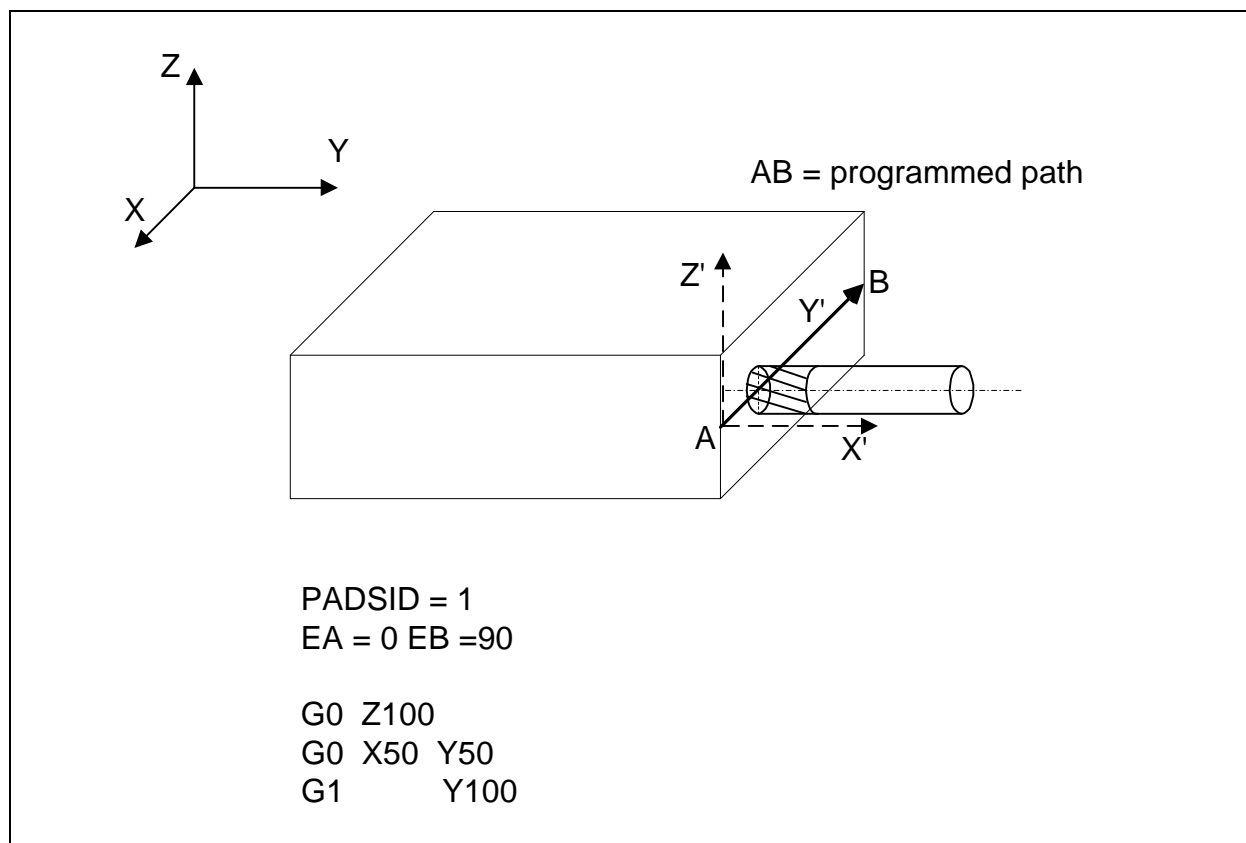


Figura 3.14 -

**Warning**

When automatic determination of the pertinent plane is activated, addresses EP, EQ and ER must not be programmed.

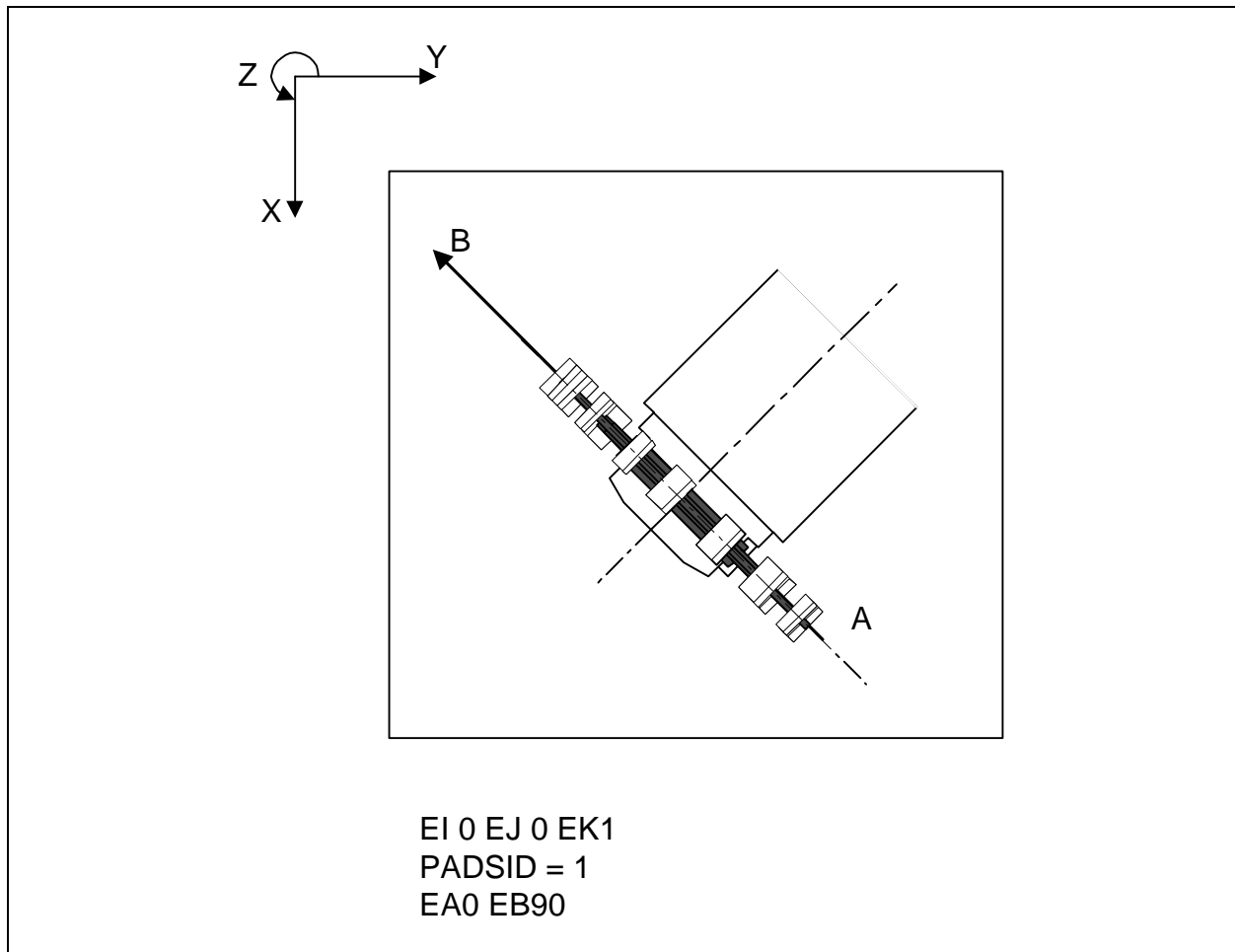


Figura 3.15 -

### Warning

If a disk-type milling cutter is used, the vector [EI, EJ, EK] must be specified and the value of EA must always be zero (EA=0).

### Example

```

PADSID=1
EA0.00 EB45.00
G0 X10.00 Y10.00 Z8.00
G2 X10.00 Y10.00 Z8.00 I40.00 J40.00
  
```

### 3.8.10 Discriminating between the interchangeable cutting edges of a tool

**Syntax** *ICDSID =0..5*

**Description**

Determines which of the available interchangeable cutting edges (e.g.: of a blade) will follow the programmed path.

It is possible to determine whether the tool is to cut to the left or right of the path, just as with the 2D tool corrector(G41,G42). If this automation is used (in the table: *the tool on the left/right of the programmed path*), the tool radius unit vector must be programmed ( *EI, EJ, EK*).

Another possibility is that of automatically determining the cutting edge to use as a function of the material surface normal vector. If this automation is used (in the table: *the tool selects a suitable cutting edge from those available*), the tool radius unit vector must be programmed ( *EI, EJ, EK*).

If, with the three dimensional corrector activated, material cannot be cut with the selected cutting edge, an error is returned.

ICDSID=0 the centre of the blade follows the programmed path

ICDSID=1 the tool is on the left of the programmed path

ICDSID=2 the tool is on the right of the programmed path

ICDSID=3 the tool uses a suitable cutting edge among those available

ICDSID=4 the tool uses the cutting edge furthest from the tool holder (DEFAULT)

ICDSID=5 the tool uses the cutting edge closest to the tool holder

To avoid damage if an unsuitable cutting edge is selected, when the channel is reset the *cutting edge furthest from the tool holder* setting is activated.

When *cutting edge closest to the tool holder* or *blade centre* are selected, the length-wise cutting allowance (SVL) and axial depth (DEA) change sign or are cancelled, respectively.

**Reversibility**

If one of the following modes is selected:

- ICDSID=0 *the centre of the blade follows the programmed path;*
- ICDSID=1 *the tool is on the left of the programmed path;*
- ICDSID=2 *the tool is on the right of the programmed path;*

...the CNC assumes that the tool is *reversible*. This means that if the feed commands required to orient the tool as specified cannot be executed, *the system will try using the opposite orientation*.

This behaviour is useful if the spindle has insufficient degrees of freedom and orientation is automatic or it can be adjusted by matrix transformations. The machining may be feasible with the selected tool even though a different orientation would have been preferable in the case of more degrees of freedom.

Note that *blade centre* mode has no variations, so that the tool is not reversible; use reversibility only if the tool really is reversible to avoid problems.

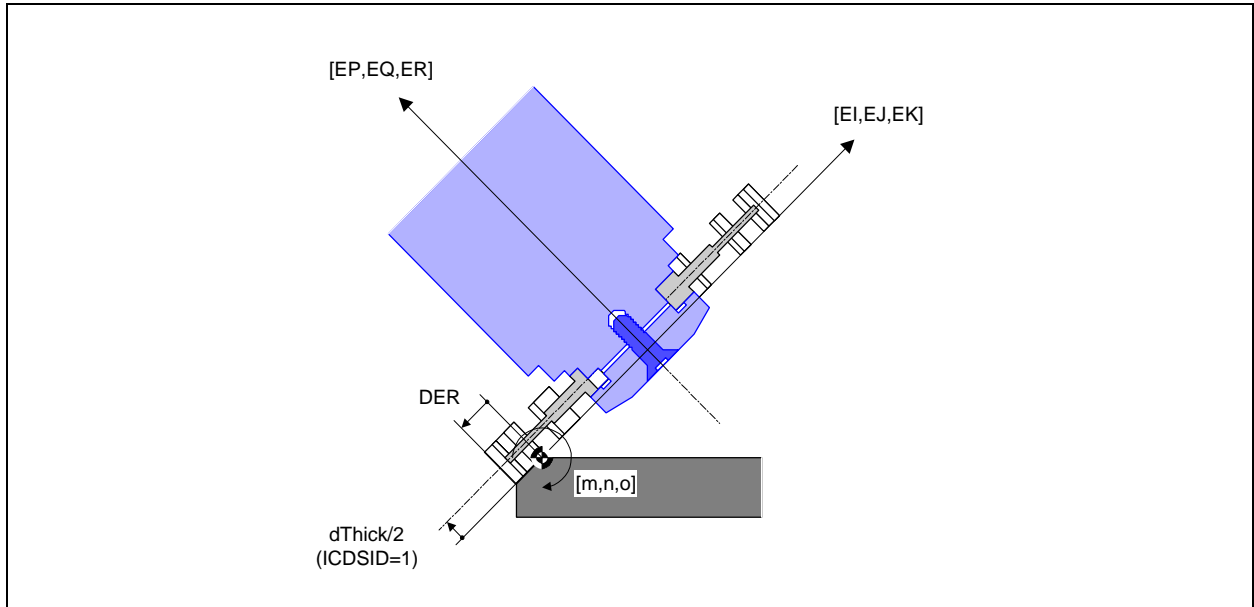


Figure 3.16 -

The disc tool is automatically corrected by half its width ( $dThick/2$ ), and stays to the left of the programmed path (direction shown as  $[m,n,o]$  vector in figure) when  $ICDSID=1$  has been programmed. Uses the cutting edge furthest from the tool holder.

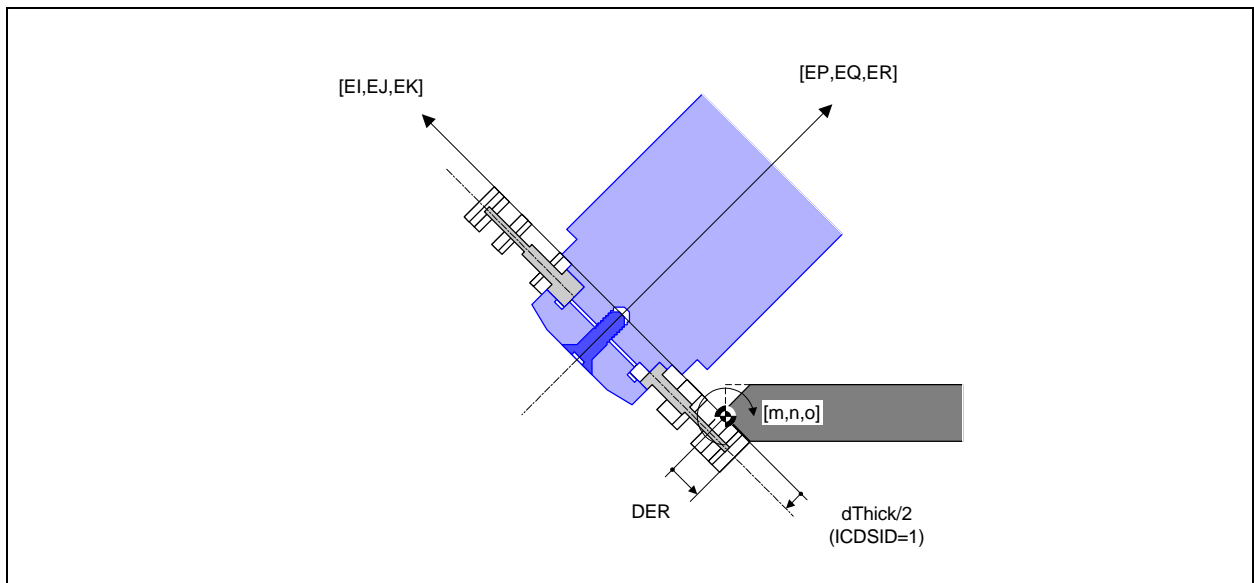


Figure 3.17 -

The disc tool is automatically corrected by half its width ( $dThick/2$ ), and stays to the left of the programmed path (direction shown as  $[m,n,o]$  vector in figure) when  $ICDSID=1$  has been programmed. Uses the cutting edge closest to the tool holder.



### 3.8.11 Machining depth compensation along the tool axis

**Syntax** *DEA* <expr>

**Type of function** Modal

**Description** DEA expresses the machining depth along the tool axis. The coordinates are automatically compensated, taking into account this component.

The compensation can be variable along the path of a machining operation. In the present version the variable depth compensation is not available if the machining operation, as compensated, is not a straight line or circular arc (in general if the path is not rectilinear and the tool axis is not perpendicular to the plane of the arc).

Note that to vary the depth without altering the tool feed in point (e.g.: drilling), the coordinates of the contour milling axes must be programmed to the same positions; otherwise no movement occurs

**Example**

```
EP0.707 EQ0.707 ER0.00 ; tool corrector vector
G0 X10.0 Y10.0 Z8.0 DEA-5.00 ; setting DEA -5.00
DEA5.00 G0 X10.00 Y10.00 ; only modifies the depth
                        ; of machining

G17 ; selects the machining plane
G0 X10.0 Y10.0 Z8.0 DEA-5.00
DEA5.00 G0 X10.00 Y10.00
```

**See also** **Machining allowance management.**

### 3.8.12 Machining depth compensation along the tool radius

**Syntax** *DER* <expr>

**Type of function** Modal

**Description** *DER* expresses the machining depth along the tool radius.  
If *DER* is non-zero, three dimensional tool compensation must be activated (*ECLMOD=1*) or the tool radius vector must be described with *EI*, *EJ* and *EK*.  
  
The compensation can be variable along the path of a machining operation. Variable depth compensation is not available if the machining operation, as compensated, is not a straight line or circular arc.  
Note that to vary the depth without altering the tool feed in point, the coordinates of the contour milling axes must be programmed to the same positions; otherwise no movement will occur.

**Example**

```
EI0.707 EJ0.707 EK0.00 ; tool correction vector  
G0 X10.00 Y10.00 Z8.00 DER-5.00  
DER5.00 G0 X10.00 Y10.00
```

**See also** **Machining allowance management.**

### 3.9 Feed in strategies

#### 3.9.1 Definitions

- **Radial in-going (RAM):** workpiece in-going mode that includes an approach path to the tool in a radial direction. It can be very useful for approaching material with a fluted mill or with a disk-type milling cutter treated like a fluted mill.
- **Axial in-going (AAM):** workpiece in-going mode that includes an approach path to the tool in an axial direction. It can be very useful for approaching material with a fluted mill in order to make a pocket or slot.

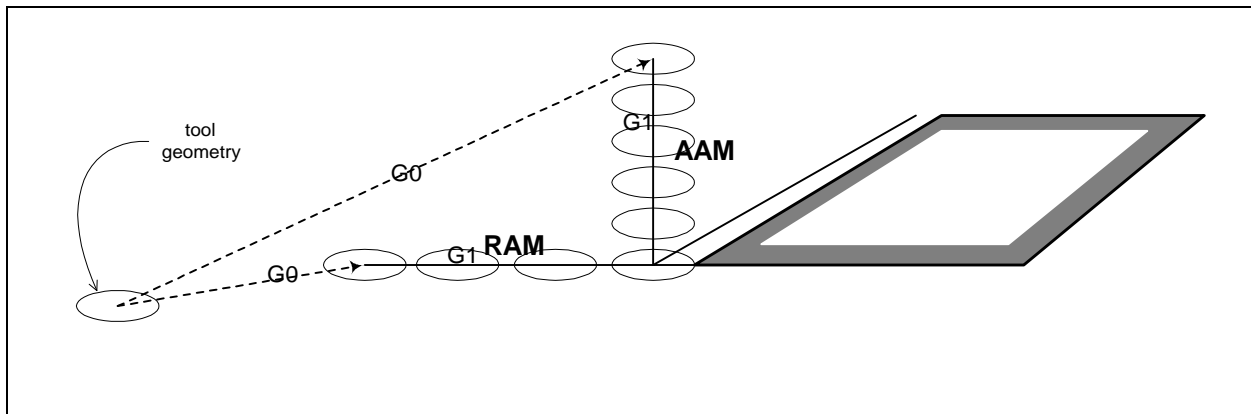


Figure 3.18 - Axial and radial feed in

#### 3.9.2 Axial/radial feed in

**Syntax** *EAMMOD = 0 / 1*

**Type of function** Modal

**Description** Enables/disables feed in along the axis/radius of the tool.  
If EAMMOD=1, whenever G0 is specified the tool is fed into the part radially or axially according to the values of the variables EAMAAH and EAMRAH.

**Syntax** *EAMFED = <expr>*

**Type of function** Modal

**Description** Selects the speed at which the tool is approached to the part. If the parameter is not set, the default value is 0.

**Syntax** *EAMAAH* = <expr>

**Type of function** Modal

**Description** Enables axial in-going on to the workpiece if different from zero. The value programmed establishes the length of the linear section approaching the workpiece, i.e. the distance from the surface of the top of the tool before it enters the material. The distance is not influenced by the machining depth selected with *DEA*, while it is influenced by the machining allowance in length selected with *SVL*.  
If only this parameter is given a value the feed in is axial.

**Syntax** *EAMRAH* = <expr>

**Type of function** Modal

**Description** Enables radial in-going on to the workpiece if different from zero. The value programmed establishes the length of the linear section approaching the workpiece, i.e. the distance of the tool from the point identified on the surface before it enters the material.  
This dimension is not affected by the machining depth set in *DER*, although it is affected by the radial machining allowance set in *SVR*.  
If only this parameter is set, the feed in is radial.  
When *EAMRAH* is non-zero, the tool radius vector must be programmed ( *EI*, *EJ*, *EK*).

#### Example

```
EAMMOD=1 ; Enables axial/radial feed in
EAMFED=1000 ; sets the feed in speed

EAMAAH=10.00
G0 X10.00 Y10.00 Z8.00 DEA 20.00 ; AXIAL feed

EAMRAH=10.00
G0 X10.00 Y10.00 Z8.00 DER 20.00 ; RADIAL feed
```

**Syntax** *EAMAIH* = <expr>

**Type of function** Modal

**Description** The linear feed in path (*EAMAAH*) can be divided into two sections at different speeds specified by *EAMFED* and *EAMFAI*.  
*EAMAIH* expresses the second linear feed section along the axis of the tool, measured from the machining surface.

**Syntax**  $EAMFAI = \langle \text{expr} \rangle$

**Type of function** Modal

**Description** The first section is executed at the speed set in EAMFED, while the second section is executed at the speed set in EAMFAI.

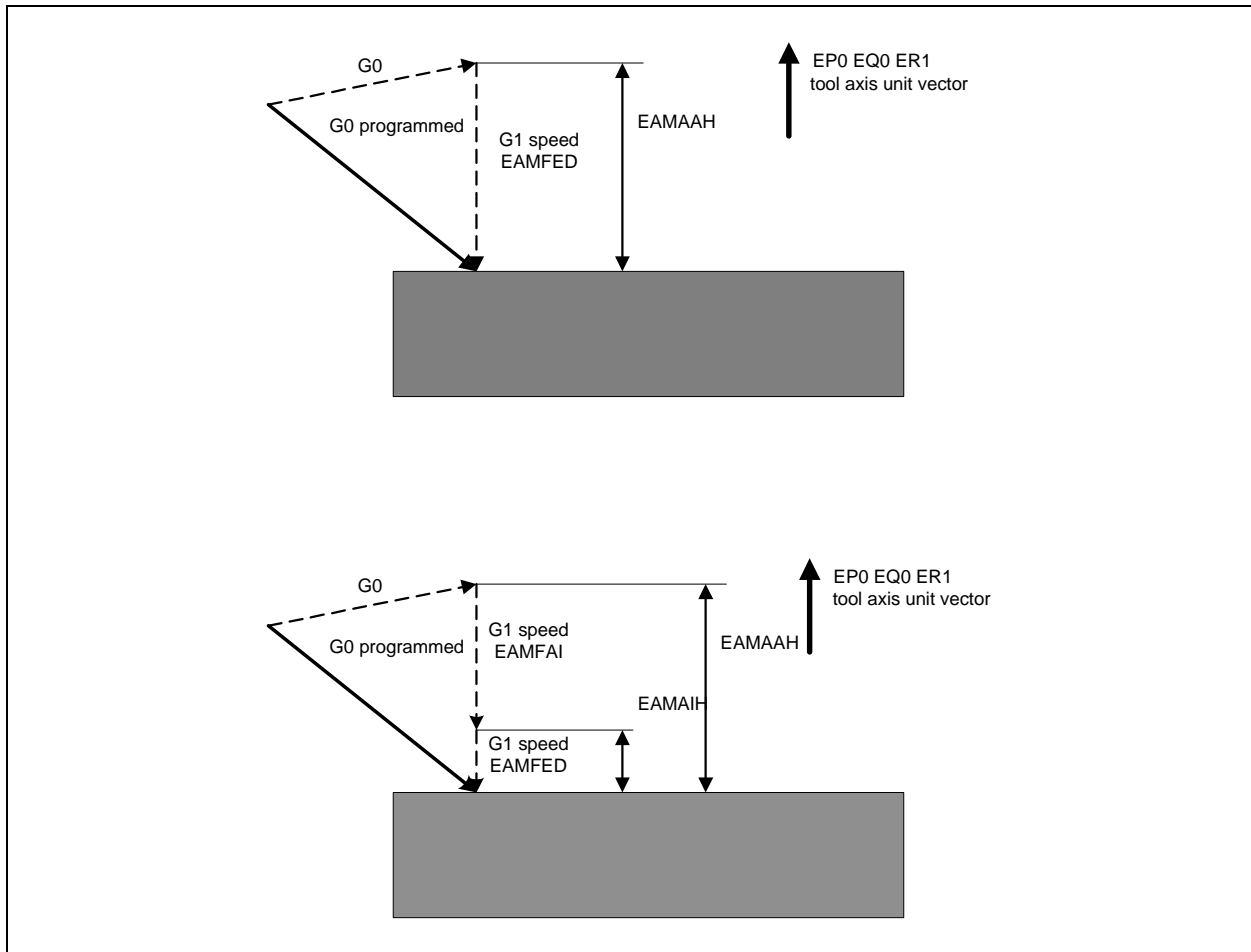


Figure 3.19 - Axial feed parameters (one/two section approach)

### 3.9.3 Axial/radial feed out

**Syntax** *XAMMOD = 0 / 1*

**Type of function** Modal

**Description** Activated or disactivated axial/radial feed out according to the values of XAMAAH and XAMRAH.

The feed out mode must be programmed before the feed in.

Once radial or axial feed out has been specified, after the first G0 which specifies the feed in, every time the system reads a G1, G2 or G3 followed by G0 the tool is fed out in the specified manner.

XAMMOD=0 specifies tool feed out of the part.

**Syntax** *XAMFED = <expr>*

**Type of function** Modal

**Description** Selects the speed for the linear feed out section. If the parameter is not specified the default value of 0 applies.

**Syntax** *XAMAAH = <expr>*

**Type of function** Modal

**Description** Enables axial out-going from the workpiece if different from zero. The value programmed establishes the length of the linear section furthering from the workpiece, or  
This dimension is not affected by the machining depth set in *DEA*, although it is affected by the axial machining allowance set in *SVL*.  
If only this parameter is set, the feed out is along the tool axis.

**Syntax** *XAMRAH = <expr>*

**Type of function** Modal.

<b>Description</b>	<p>Enables radial out-going from the workpiece if different from zero. The value programmed establishes the length of the linear section furthering from the workpiece, i.e. the distance of the tool of the tool from the point identified on the surface after leaving the material. The distance is not influenced by the machining depth selected with <i>DER</i>, while it is influenced by the machining allowance on the radius selected with <i>SVR</i>.</p> <p>Out-going takes place along a radius of the tool if only this parameter has been set. The tool radius versor (<i>EI</i>, <i>EJ</i>, <i>EK</i>) must be programmed when <i>XAMRAH</i> is set at different values from 0..</p>
<b>Syntax</b>	<i>XAMAIH</i> = <expr>
<b>Type of function</b>	Modal
<b>Description</b>	<p>The linear feed out path (<i>XAMAAH</i>) can be divided into two sections at different speeds specified by <i>XAMFED</i> and <i>XAMFAI</i>. <i>XAMAIH</i> expresses the second linear feed section along the axis of the tool, measured from the machining surface.</p>
<b>Syntax</b>	<i>XAMRIH</i> = <expr>
<b>Type of function</b>	Modal
<b>Description</b>	<p>The linear section for radial out-going from the workpiece programmed with <i>XAMRAH</i> can be divided into two segments to cover at different speeds, one specified by <i>XAMFED</i> and the other by <i>XAMFAI</i>. <i>XAMRIH</i> expresses the length of the first linear section for furthering from the workpiece along the tool axis, beginning from the surface of the workpiece itself.</p>
<b>Syntax</b>	<i>XAMFAI</i> = <expr>
<b>Type of function</b>	Modal
<b>Description</b>	The first section is executed at the speed set in <i>XAMFED</i> , while the second section is executed at the speed set in <i>XAMFAI</i> .
<b>Example</b>	<pre>XAM and EAM activated %1 F900000</pre>

```
G58 X100 Z200 ;set auxiliary origin.

EAMMOD =1 ;Activate axial feed in
EAMFED =9000
EAMAAH =100

XAMMOD =1 ;activate axial feed out
XAMFED =9000
XAMAAH =500

D1 ;select tool corrector

EP0 EQ0 ER1 ;orient the tool

VA0 = 0
VA1 = 400

N10 G0 X100 Y100 Z10 ;feed out/in
N20 G2 Y300 R100
N30 G1 X300 Z10
N40 G1 Y100
N50 G1 X100

VA0=VA0+VA1
N60 SHF[X]=VA0

RPT 10,60,2

XAMMOD = 0 ; feed out
M30
```



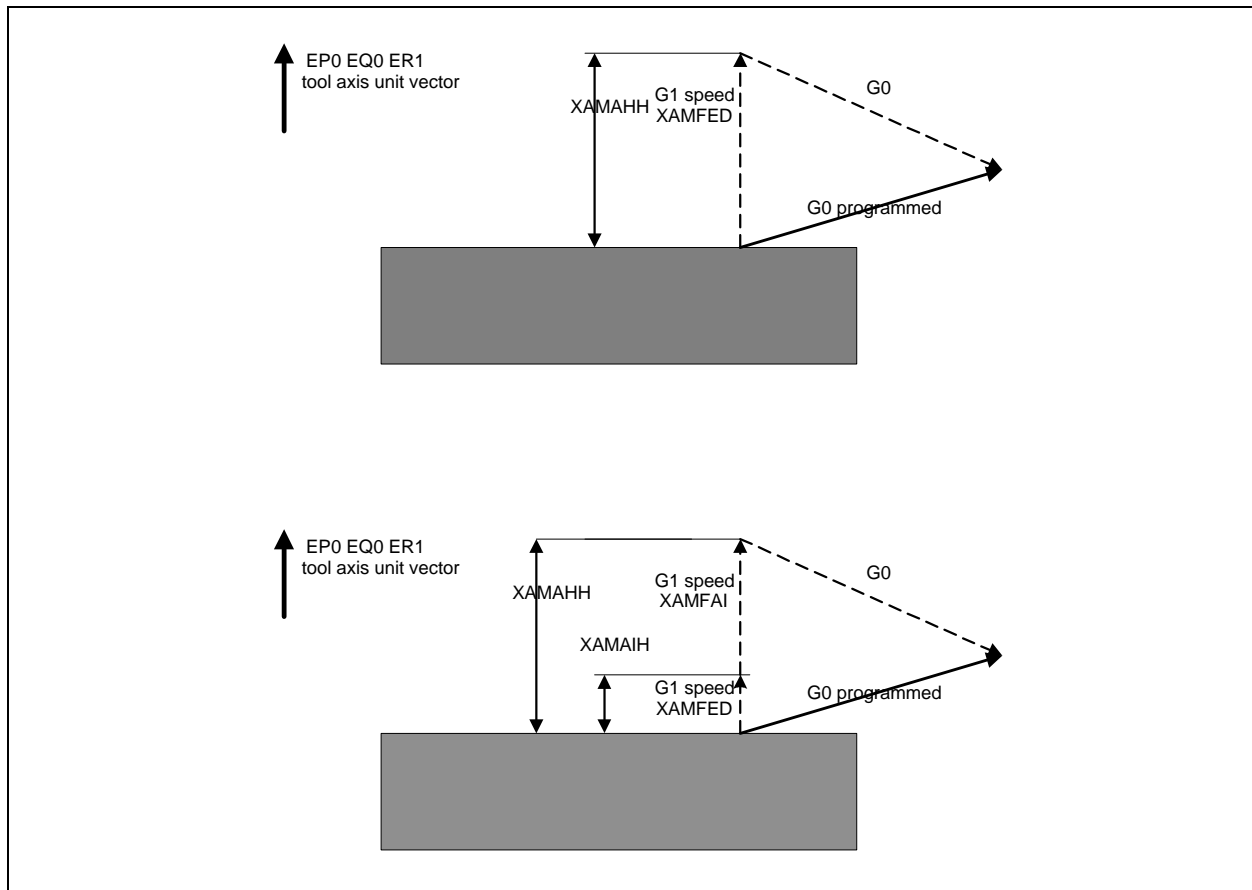


Figure 3.20 - Axial feed out parameters (one/two sections)

### 3.9.4 Compensation depending on a section of the tool along the trajectory in its in-going point

**Syntax** *ESCMOD =ON/OFF*

**Type of function** Modal

**Description** Activates/deactivates compensation depending on a cutting section of the tool along the trajectory in its in-going point  
 The tool section that touches the material depends on the depth programmed for the machining process (*DER*).  
 When ESCMOD is activated, the in-going targets of the machining process are compensated *along the trajectory* to suit that section, so that the programmed machining starting point is on the surface of the tool.  
 When ESCMOD is activated, the tool radius versor must be described with *EI*, *EJ* and *EK* and the tool rotation axis and vector that are normal in relation to the surface must also be normal in relation to the trajectory within a certain tolerance margin.  
 The trajectory must be linear and the machining section sufficiently long, so as to allow tool compensation in the in-going point. However, if the machining process consists of a single block G1 and compensation in the out-going point is activated, the machining section must be at least as long as the sum of the two compensations.

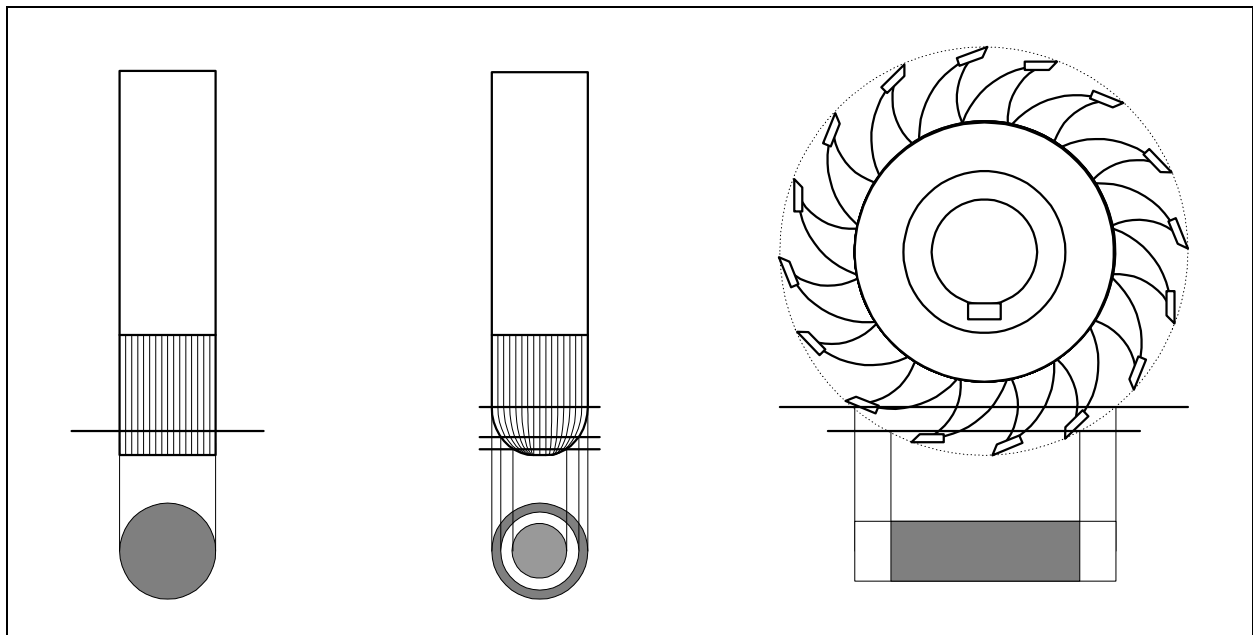


Figure 3.21 -

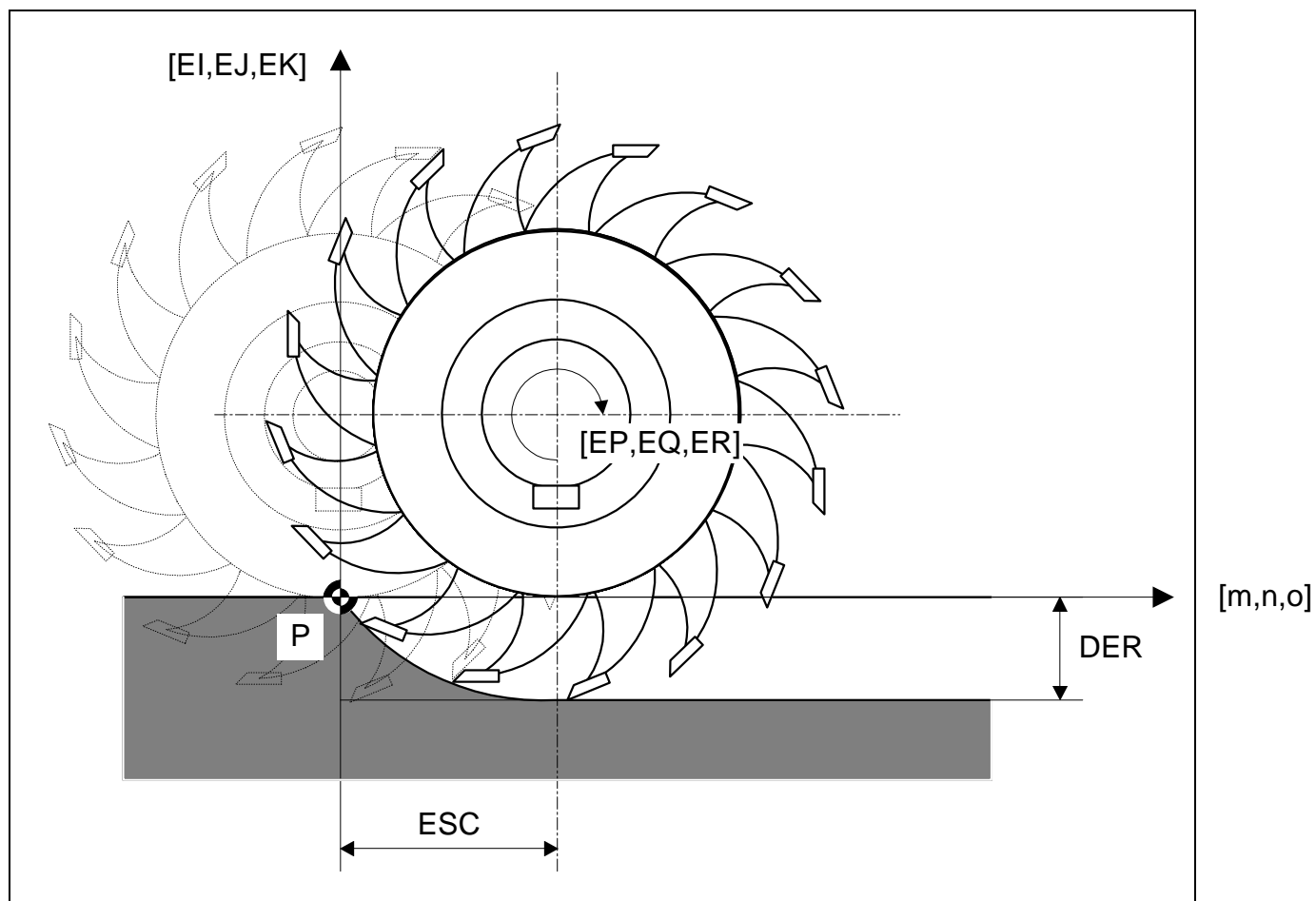


Figura 3.22 -

Parameters for Compensation along the trajectory depending on the section of a disk-type cutting tool.  $P$  is the machining starting point,  $ESC$  is the compensation along the trajectory,  $[m, n, o]$  the infeed vector.

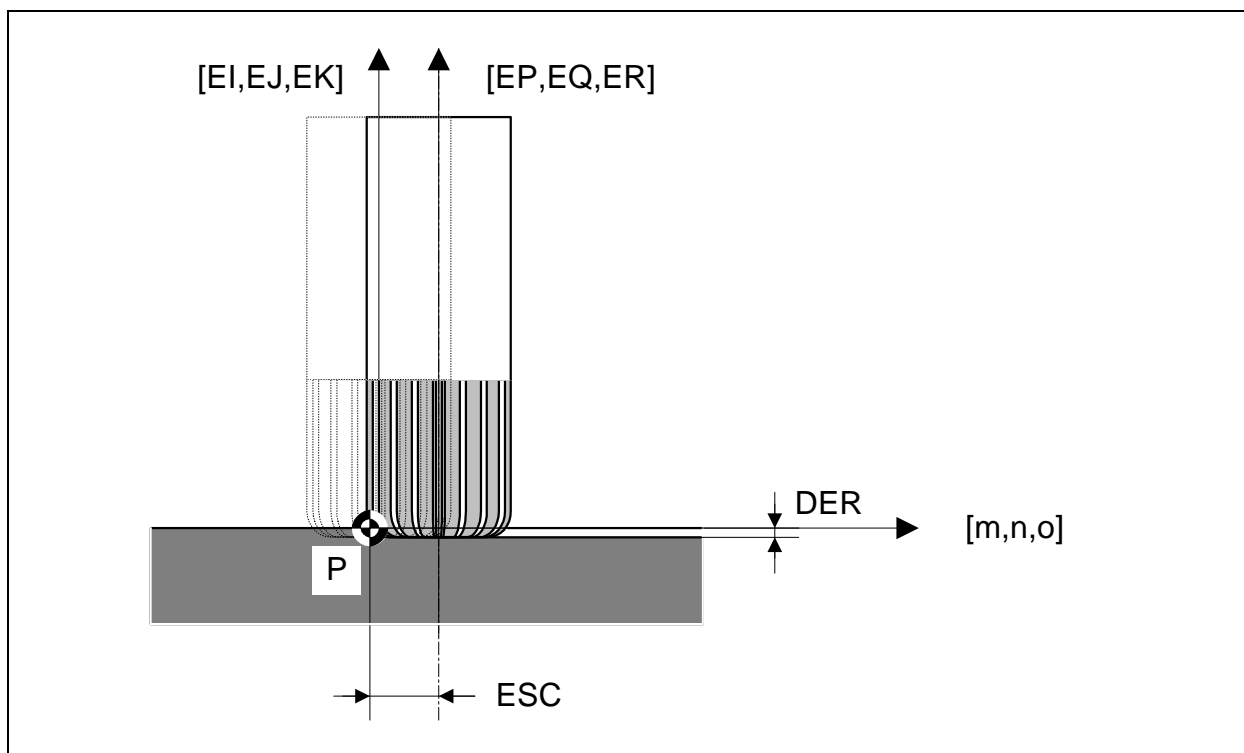


Figura 3.23 - Compensation along the trajectory depending on the section of a toroidal tool.

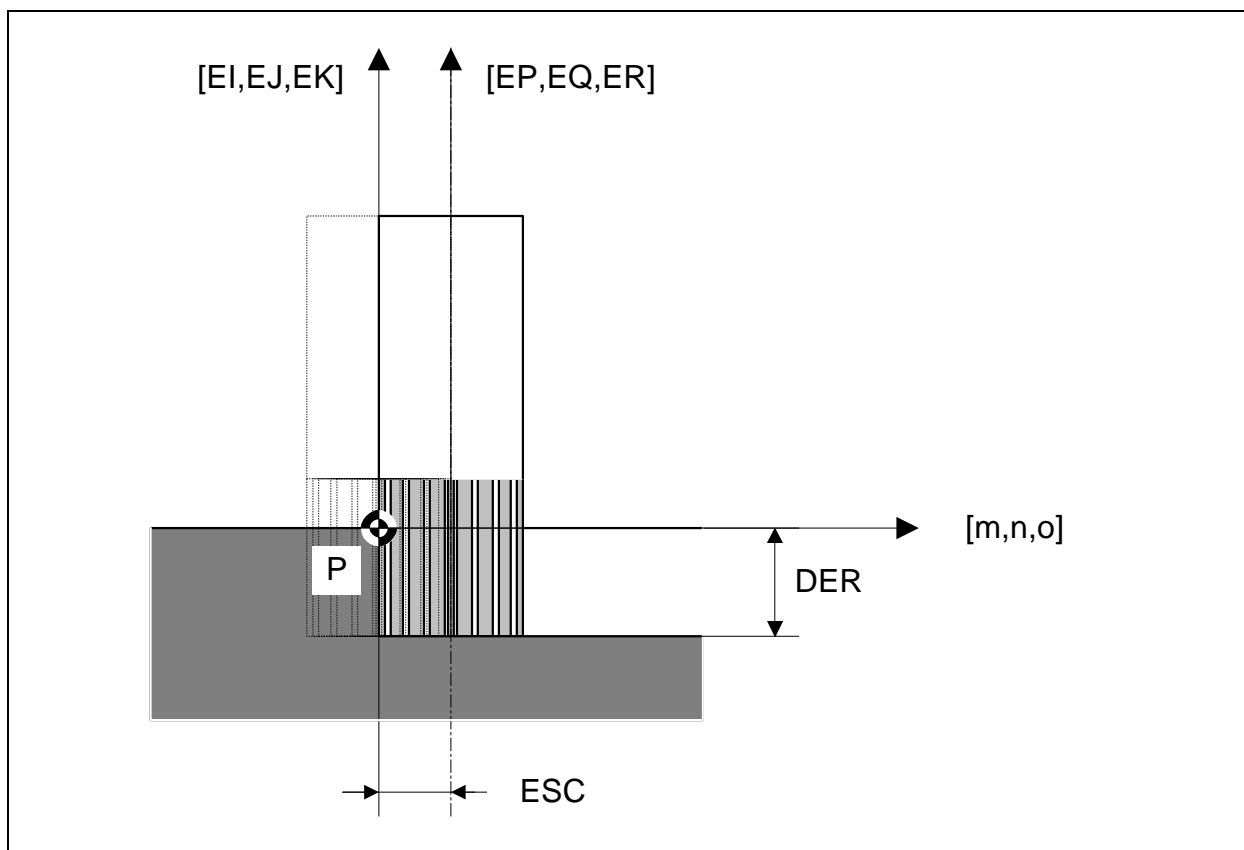


Figure 3.24 -

Compensation along the trajectory depending on the section of a cylindrical tool. The DER setting is not significant since the tool section is constant (so long as the values are higher than 0).

### 3.9.5 Compensation depending on a section of the tool along the trajectory in its out-going point

**Syntax** *XSCMOD =ON/OFF*

**Type of function** Modal

**Description** Activates/deactivates compensation depending on a cutting section of the tool along the trajectory in its out-going point  
When activated, the out-going targets of the machining process are compensated *along the trajectory* so that the programmed end of machining point lies on the tool surface, *considering the machining depth specified with DER*.

When XSCMOD is activated, the tool radius versor must be described with *EI*, *EJ* and *EK*. The tool's axis of rotation must be normal in relation to the trajectory and within a certain tolerance. The trajectory must be linear and the machining section sufficiently long, so as to allow tool compensation in the out-going point. However, if the machining process consists of a single block G1 and compensation on the in-going point is activated, the machining section must be at least as long as the sum of the two compensations.

## 3.10 Feed Control

### 3.10.1 Modification or Automatic speed control parameters

**Syntax** *FDCDLE= 0/1*

**Type of function** Modal

**Description** *Deceleration Look-Ahead Enable.*  
FDCDLE =1 Enables the automatic speed control algorithm, which enables determination of the section feed speed, taking into account the axis movements and feed rates and the machining operations within a certain number of lines (in the present version the number of steps cannot be set).

FDCDLE=1 Enable  
FDCDLE=0 Disable

**Syntax** *FDCTC0=<expr>*  
*FDCTC1= <expr>*  
*FDCTC2= <expr>*  
*FDCTC3= <expr>*

**Type of function** Modal

**Description****FDCTC0 →** *90° angle Reference deceleration [mm/min]*

Sets the almost instantaneous decelerations of the cartesian axes.

Too low values result in rough angles, while too high values affect the machine's mechanisms, resulting in oscillations and overshoot with sharp angles.

**FDCTC1 →** *Arc-line angle Reference Deceleration [°/min]*

Sets the almost instantaneous decelerations of the polar axes.

Too low values result in discontinuous contours with rough tangential axes, whereas too high values result in delays in stopping the vector axis while passing from curved to straight sections, or to larger radius curves.

**FDCTC2 →** *0° Angle Reference Acceleration → [mm/min]*

Sets the almost instantaneous accelerations of the cartesian axes .

Too low values result in rough angles, while too high values affect the machine's mechanisms, resulting in oscillations and overshoot even with slight angles.

**FDCTC3 →** *Line-Arc angle Reference Acceleration [°/min]*

Sets the almost instantaneous accelerations of the polar axes.

Too low values result in discontinuous contours with rough tangential axes, whereas too high values result in delays in stopping the vector axis while passing from curved to straight sections, or to smaller radius curves.



### 3.11 Independent axis synchronous/asynchronous positioning

#### 3.11.1 Independent axis synchronous positioning

**Syntax**                      **POS**[Axis] = pos  
                                 **FA**[Axis] = speed

**Type of function**            Self-cancelling

**Description**                The selected axis is positioned not by the channel's main process, but by the process corresponding to the axis, using the axis's characteristic parameters .  
FA[ ] specifies the positioning speed in mm/min.

Both the positioning and the contour milling axes can be programmed in the same block. The axes start at the same time, but can reach their targets at different times since each axis programmed with POS has a different speed and acceleration.  
The ISO block is switched when all axes have reached their targets.

**Example**

```
G1 X100 Y50 F1000 POS[U]=200 POS[B]=50 FA[U]=500 FA[B]=2000
G0 X0 Y0 ;this block is executed only when
         ;X,Y,A and B have reached their targets
```

**See also**                      **Indipendent axis asynchronous positioning.**

### 3.11.2 Independent axis asynchronous positioning

**Syntax**

**POSA**[Axis] = pos  
**FA**[Axis] = speed  
**WAITP**[Axis List]

**Type of function** Self-cancelling

**Description**

The axis is driven not by the main channel process, but by the process corresponding to the axis.

Both the contour milling and positioning axes can be programmed in the same ISO block (POSA[]), in this situation the axes all start at the same time and arrive at their targets at different times.

Using POSA instead of POS the ISO does not wait until the axis has reached its target to switch.

FA[ ] sets the positioning speed [mm/min].

WAITP[] enables resynchronisation by waiting for the specified axis/axes to reach their targets.

**Example**

```
G0 X200 POSA[U]=300 POSA[B]=200 FA[U]=150 FA[B]=300

WAITP[B]    ; waits for B to reach its target
X300 POSA[B]=300
WAITP[U]    ; waits for U to reach its target
POSA[U]=350
X400
WAITP[U,B]   ; waits for U and B to reach their targets
```

**See also** **Indipendent axis asynchronous positioning.**

### 3.11.3 Prallel spindle orientation

**Syntax**                    **POSA**[S] = pos  
                              **WAITP**[Axis List]

**Type of function**        Self-cancelling

**Description**             POSA[S] enables orienting the spindle.  
                              The ISO block switches without waiting for the spindle to reach its target.  
                              WAITP[S] forces waiting for positioning to terminate.

**Example**

```
G1 B0 F1000 POSA[S] = 45.0
Z200
X0 Y0
WAITP[S]
```

**See also**                 **Spindle orientation, On-the-fly spindle orientation.**

## 3.12 Modifying axis parameters

The parameters that characterize the dynamics of an axis are set up in the machine parameters ([*Parameters*] [*System*] [*Axes Parameters*]).

The following instructions allow these parameters to be modified straight from the PartProgram

### 3.12.1 Following error

**Syntax** `FWR[Axis] = 0|1`

**Type of function** Modal

**Description** Enables/disables synchronously following error monitoring.  
Axis Axis name.

**Mode: 0** Disables following error monitoring.

**Mode: 1** Enables following error monitoring.

**Example**

```
FWR[X] = 0
FWR[X] = 1
```

### 3.12.2 Axis acceleration time

**Syntax** `ACC[Axis] = <expr>`

**Type of function** Modal

**Description** Modifies the acceleration of the specified axis synchronously.  
The set acceleration expresses the time [in seconds] required to reach the maximum speed.

**Example**

```
ACC[X] = 1.2
```

### 3.12.3 Axis deceleration time

**Syntax**                    **DEC**[Axis] = expr

**Type of function**        Modal

**Description**             Modifies the deceleration of the specified axis synchronously.  
The expression gives the time [in seconds] required to complete a deceleration at the maximum delta(speed)

**Example**

DEC[X] = 1.5

### 3.12.4 Axis emergency deceleration time

**Syntax**                    **DEE**[Axis] = <expr>

**Type of function**        Modal

**Description**             Modifies the emergency deceleration of the specified axis synchronously.  
The expression gives the time in seconds required to complete a deceleration at the maximum delta(speed) in an emergency.

**Example**

DEE[X] = 1.5

### 3.12.5 Axis maximum speed

**Syntax** `VEL[Axis] =<expr>`

**Type of function** Modal

**Description** Modifies the maximum speed in mm/min of the specified axis synchronously.

**Example**

`VEL[X] = 1000`

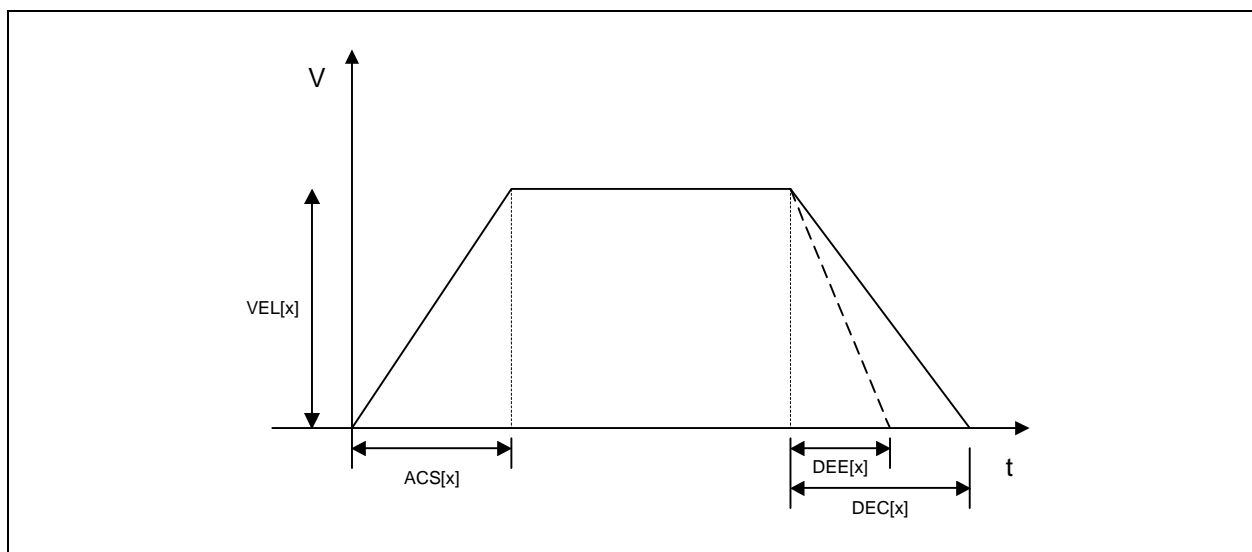


Figure 3.25 - Axis characteristic parameters

### 3.12.6 Axis ramp to S

**Syntax** ACS[Axis] =<expr>

**Type of function** Modal

**Description** Synchronously sets the parameter which determines the ramp to S of the specified axis. The ramp time is the time required to accelerate to the maximum. The ramp to S delay time must not exceed half the acceleration time.

**Example**

ACS[X] = 1.5

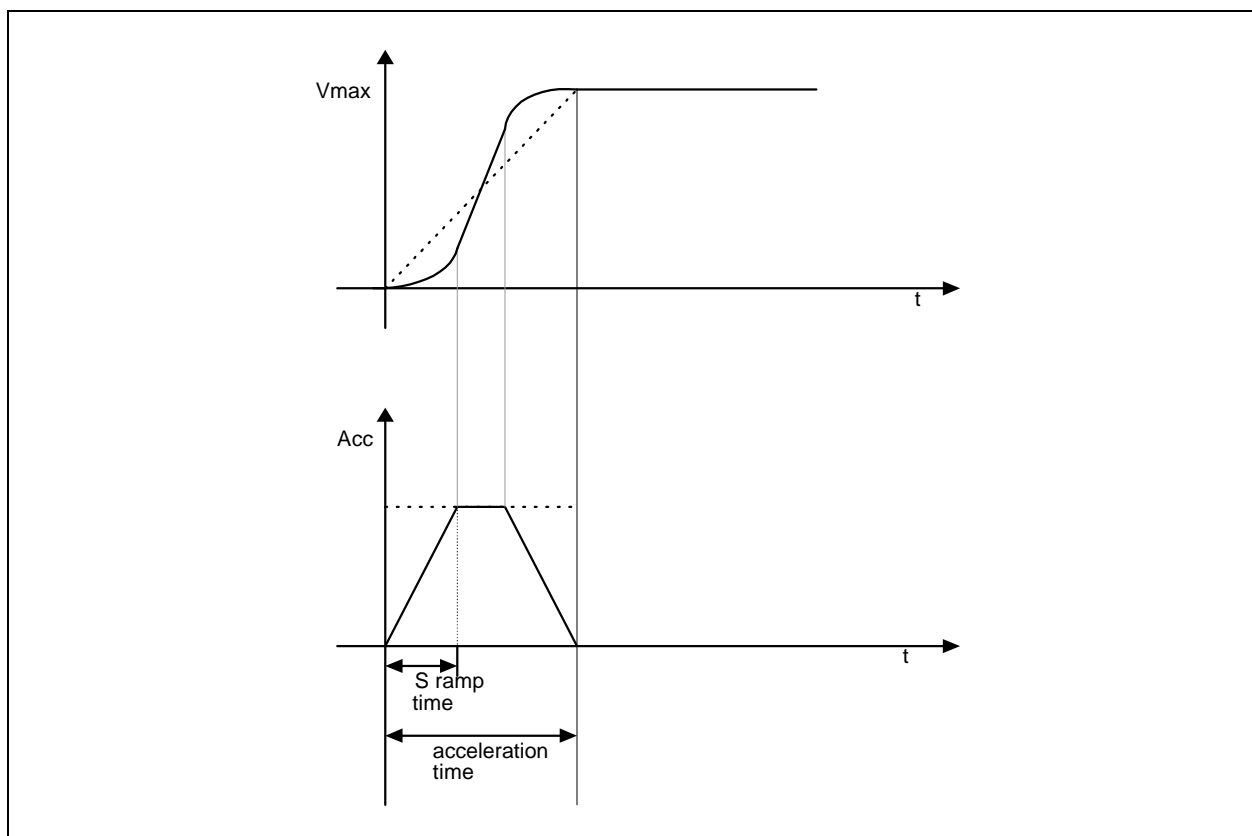


Figure 3.26 - Ramp to S

### 3.12.7 Axis jerk settings

**Syntax**                      JRK[Axis] =<expr>

**Type of function**            Modal.

**Description**                Sets up the parameter that characterizes the variation of the acceleration (JERK) of the associated axis in the synchronized mode. The value entered indicates the acceleration variation in the unit of time, and is expressed in mm/sec<sup>3</sup>.

**Example**                      JRK[X] = 22500



### 3.13 Modification of the interpolation parameters

The parameters that characterize the interpolation dynamics are set up in the machine parameters ([Parameters][System][Channel Parameters]).

The following instructions allow these parameters to be modified straight from the PartProgram

#### 3.13.1 Interpolation acceleration time setting

**Syntax** `ACCTRJ =<expr>`

**Type of function** Modal

**Description** Modifies the characteristic acceleration of the channel interpolations in the synchronous mode. The acceleration set expresses the time [in seconds] required to reach the maximum speed.

**Example** `ACCTRJ =1 . 2>`

#### 3.13.2 Interpolation deceleration time setting

**Syntax** `DECTRJ =<expr>`

**Type of function** Modal

**Description** Modifies the characteristic deceleration of the channel interpolations in the synchronous mode. The expression set expresses the time [in seconds] required to make a deceleration equal to the maximum speed delta.

**Example** `DECTRJ = 1 . 2>`

### 3.13.3 Maximum interpolation speed setting

**Syntax** VELTRJ =<expr>

**Type of function** Modal

**Description** Modifies the value in mm/min or the maximum channel interpolation speed in the synchronous mode.

**Example** VELTRJ = 1000

### 3.13.4 Interpolation 'S' ramp time setting

**Syntax** ACSTRJ =<expr>

**Type of function** Modal

**Description** Sets up the parameter that characterizes the channel's interpolation 'S' ramp in the synchronized mode. The ramp time defines the time that the acceleration takes to reach the maximum programmed time. The 'S' ramp delay value must not be more than half the acceleration time.

**Example** ACS [X] . 1 . 5

### 3.13.5 Jerk setting during interpolation

**Syntax**                      JRKTRJ =<expr>

**Type of function**            Modal

**Description**                Sets up the parameter that characterizes the variation of the acceleration (JERK) during a channel interpolation. The value entered indicates the acceleration variation in the unit of time, and is expressed in mm/sec<sup>3</sup>.

**Example**                      JRKTRK = 22500

**END OF CHAPTER**



## 4 Fixed Cycle Systems (FCS)

### Definition

The Fixed Cycle Systems comprise the preparatory functions G from G81 to G89 which enable the definition of technical processing parameters for drilling, tapping and boring. The fixed cycle is activated in the first ISO block which evaluates the feed target (X, Y). Macros G192-G193 are supplied define the geometry (with the cartesian points at which the technological process is performed).

G192: Hole grid.

G193: Holes on a circular arc.

## 4.1 G192 macro Generation of a grid of points

### Syntax

**G192** *X*<> *Y*<> *IA*<> *JA*<> *NRA*<> *IB*<> *JB*<> *NRB*<>

G192 Fixed Cycle Interface:

**X**      P0x coordinate  
**Y**      P0y coordinate  
**IA**     Line: X increment component  
**JA**     Line: Y increment component  
**NRA**    Line: Number of holes  
**IB**     Column: X increment component  
**JB**     Column: Y increment component  
**NRB**    Column: Number of holes

### Type of function

Self-cancelling

### Description

Defines geometry for grid points.

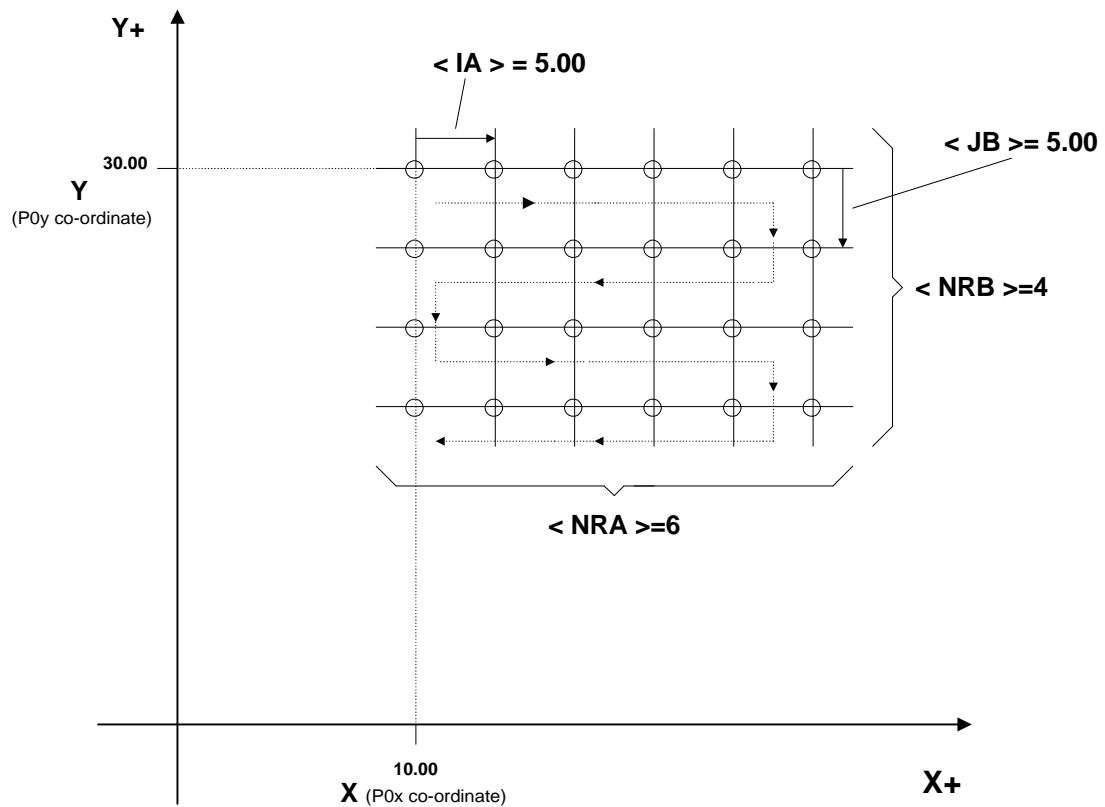
The parameters transferred in the call have the following characteristics:

IA: if undeclared IA = 0  
 JA: if undeclared JA = 0  
 NRA: if undeclared, NRA = 1  
 IB: if undeclared IB = 0  
 JB: if undeclared JB = 0  
 NRB: if undeclared NRB = 1

### Example

```
Drilling, centring %1181
N10 G0 X0 Y0 Z200
(Technological parameter definition)
N20 G81 DR125 DE100 DH125 FW150 FR15000
(Fixed Drilling Cycle Activation)
N30 G192 X-25 Y-25 IA10 JA5 NRA5 IB5 JB10 NRB6
N50 G80
N60 M10
N70 M30
N80 END
```

## G192 Macro ( hole grid )



**G192 X<> Y<> IA<> JA<> NRA<> IB<> JB <> NRB<>**

**X** P0x Co-ordinate

**Y** P0y Co-ordinate

**IA** Line: X increment component

**JA** Line: Y increment component

**NRA** Line: Number of holes

**IB** Column: X increment component

**JB** Column: Y increment component

**NRB** Column: Number of holes

In this case:

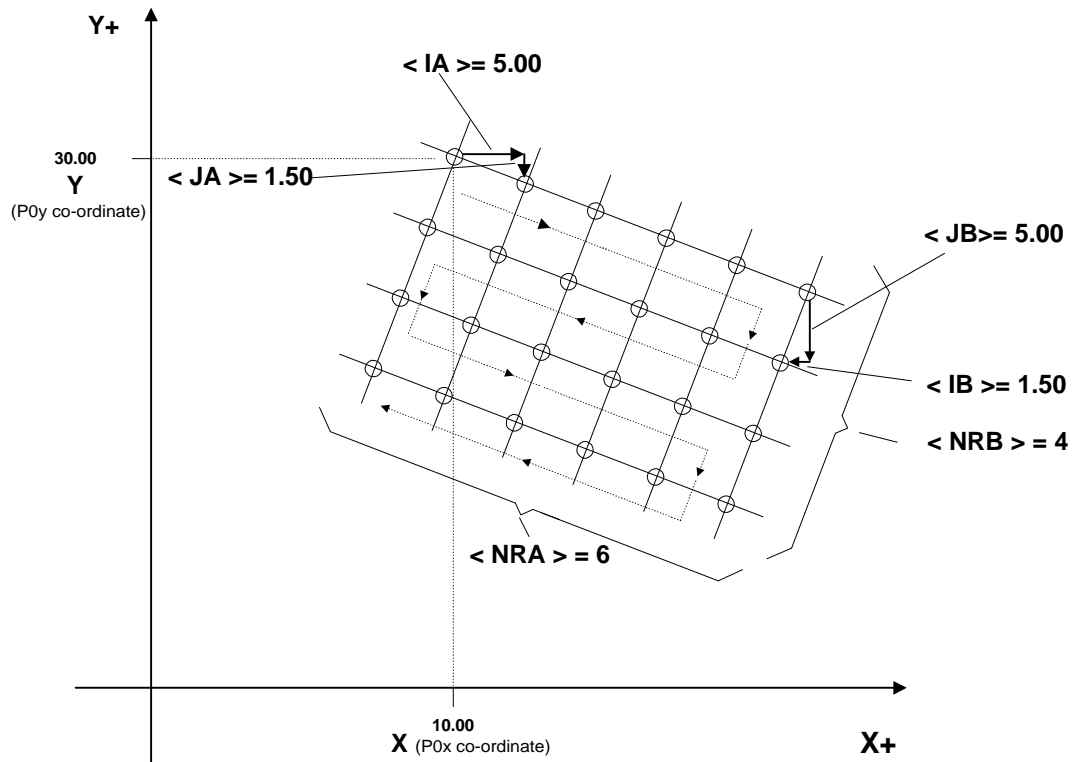
**G192 X 10.00 Y 30.00 IA 5.00 NRA 6 JB 5.00 NRB 4**

JA: if undeclared JA = 0

IB: if undeclared JB = 0

*Figura 4.1 - Profile executed by G192 Macro*

## G192 Macro ( Hole grid )



**G192** X<> Y<> IA<> JA<> NRA<> IB<> JB <> NRB<>

**X** P0x Co-ordinate

**Y** P0y Co-ordinate

**IA** Line: X increment component

**JA** Line: Y increment component

**NRA** Line: Number of holes

**IB** Column: X increment component

**JB** Column: Y increment component

**NRB** Column: Number of holes

In this case:

**G192 X 10.00 Y 30.00 IA 5.00 JA 1.50 NRA 6 JB 1.50 JB 5.00 NRB 4**

Figura 4.2 - Profile executed by G192 Macro



## 4.2 G193 macro - generation of points distributed along a circumference arc

**Syntax** *G193 X<> Y<> AR<> I<> J<> NR<>*

G193 Fixed Cycle interface:

*X*      P0x coordinate  
*Y*      P0y coordinate  
*AR*     Angle between two holes  
*I*       Centre Cx  
*J*       Centre Cy  
*NR*     Number of holes

**Type of function**      Self-cancelling

**Description**            Defines the geometry required to generate points distributed on an arc.  
 The parameters transferred in the call have the following characteristics:

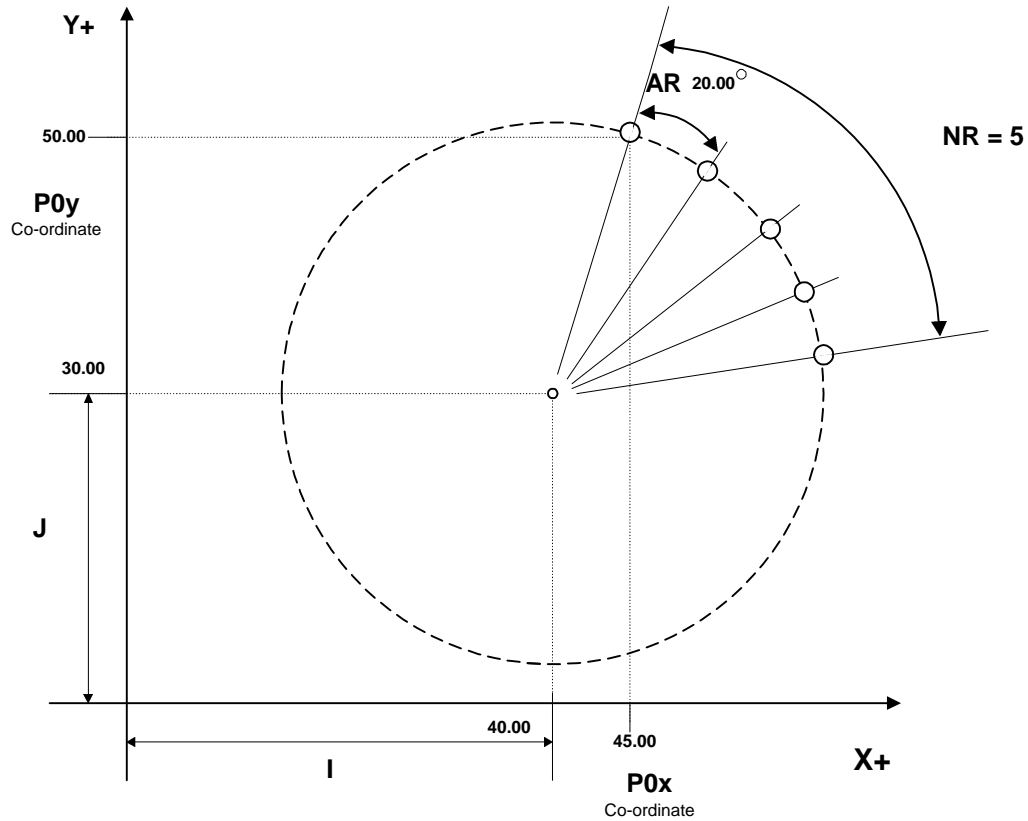
AR: if undeclared AR = 360  
 I: compulsory  
 J: compulsory  
 NR: if undeclared NR = 1

### Example

```
Drilling, centring %1181
N10 G0 X0 Y0 Z200
      (Technological parameter definition)
N20 G81 DR125 DE100 DH125 FW150 FR15000
      (Fixed Drilling Cycle Activation)
N30 G193 X0 Y0 AR90 I100 J0 NR3
N50 G80
N60 M10
N70 M30
N80 END
```

**G193 Macro**

(Holes on a circular arc)

**G193 X<> Y<> AR<> I<> J<> NR<>****X** P0x Co-ordinate**Y** P0y Co-ordinate**AR** Angle between two holes (if positive drilling is anticlockwise; if negative drilling is clockwise).**I** Centre Cx**J** Centre Cy**NR** Number of holes

In this case:

**G193 X 45.00 Y 50.00 AR -20.00 I 40.00 J 30.00 NR 5**

AR: if undeclared AR = 360

NR: if undeclared NR = 1

*Figura 4.3 - Profile executed by G193 Macro*

### 4.2.1 G81 Fixed Cycle: drilling

#### Syntax

**G81 DR<> DE<> DH<> FW<> FR<> WE<>**

G81 Fixed Cycle interface:

<b>DR</b>	(VA23) Position Z reached in fast mode	(Fast mode position)
<b>DE</b>	(VA3) End of hole position Z reached in Feed	(End position)
<b>DH</b>	(VA4) Home return position Z	(Home position)
<b>FW</b>	(VA5) Z axis work speed	(Feed Work)
<b>FR</b>	(VA6) Feed return speed	(Feed Return)
<b>WE</b>	(VA7) End of hole wait time	(Wait End)

#### Type of function

Modal

#### Description

Defines the technological parameters for drilling.  
The function G80 (applied on PP line only), or a G0 Z feed (position) cancel G81 mode explicitly.  
The parameters transferred in the call have the following characteristics:

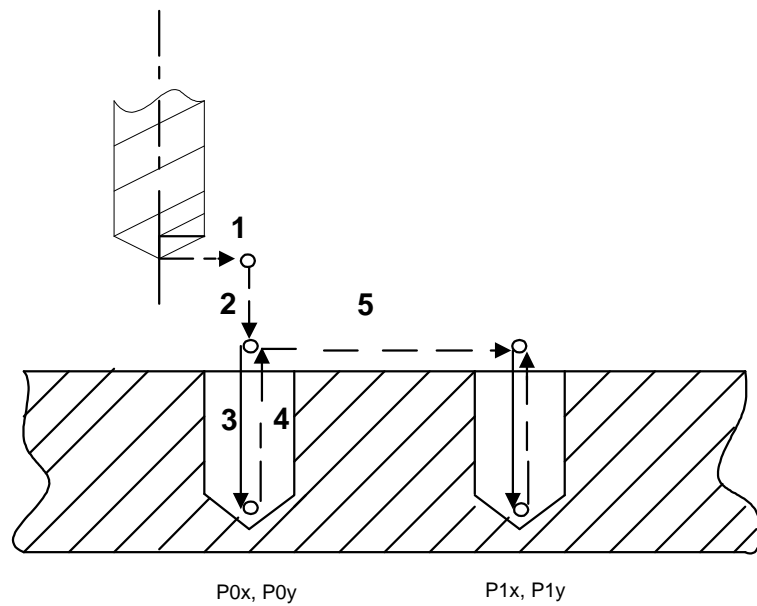
DR: modal  
DE: modal  
DH: if undeclared, DH = DR  
FW: modal  
FR: if undeclared, max. speed (machine data).  
WE: if undeclared, WE = 0

#### Example

```
Drilling, centring %1181

N10 G0 X0 Y0 Z200
    (Technological parameter definition)
N20 G81 DR125 DE100 DH125 FW150 FR15000
    (Fixed Drilling Cycle Activation)
    (P0x, P0y = 10, 10)
    (P1x, P1y = 40, 50)
N30 X10 Y10
N40 X40 Y50
N50 G80 (applied on line only)
N60 M10
N70 M30
N80 END
```

## G81 Fixed Cycle: Centring and Drilling



- 1) Positioning of axes X and Y (P0x, P0y); first hole.
- 2) Rapid Z descent to machining start position **DR**.
- 3) Z end of hole position reached in Feed **DE**.
- 4) Z home return position **DH**.
- 5) Positioning of axes X and Y (P1x, P1y).

Figura 4.4 - Profile executed by fixed cycle G81

#### 4.2.2 G83 Fixed Cycle: drilling with swarf discharge or breaking

##### Syntax

**G83 DR<> DE<> DH<> FW<> FR<> WE<> NI<> WI<>**

G83 Fixed Cycle interface:

<b>DR</b>	(VA23) Position Z reached in fast mode	(Fast mode position)
<b>DE</b>	(VA3) End of hole position Z reached in Feed	(End position)
<b>DH</b>	(VA4) Home return dimension Z	(Home position)
<b>FW</b>	(VA5) Z axis work speed	(Feed Work)
<b>FR</b>	(VA6) Feed return speed	(Feed Return)
<b>WE</b>	(VA7) End of hole wait time	(Wait End)
<b>NI</b>	(VA19) Number of increments	(Number Increment)
<b>WI</b>	(VA23) Wait time between increments	(Wait Increment)

Defines the technological parameters to use for boring with swarf unloading or breaking

##### Type of function

Modal

##### Description

Defines the technological parameters used in drilling

The function G80 (applied on PP line only), or a G0 Z feed (position) cancel G83 mode explicitly.

Parameters transferred in the call are self-cancelling.

Parameter DE determines the hole depth and DH defines the pre-set position of axis Z to exit the workpiece.

NI determines the number of processing phases. The depth value is taken and divided by the number in NI, to calculate the increment automatically for each processing phase.

The first drilling phase is then carried out on the first point calculated for axis Z. At this point return is activated in fast mode to the position specified in DR .

Axis Z restarts at high speed to reach a new point at a specific distance from the previous one, minus 1mm. From this point, using a G1 type feed, drilling is resumed up to the new point calculated automatically by the cycle.

These phases are repeated until the position specified in DE is reached.

The parameters transferred in the call have the following characteristics:

DR: modal

DE: modal

DH: if undeclared, DH = DR

FW: modal

FR: if undeclared, max. speed (machine data).

WE: if undeclared, WE = 0

NI: modal

WI: if undeclared, WI = 0

**Example**

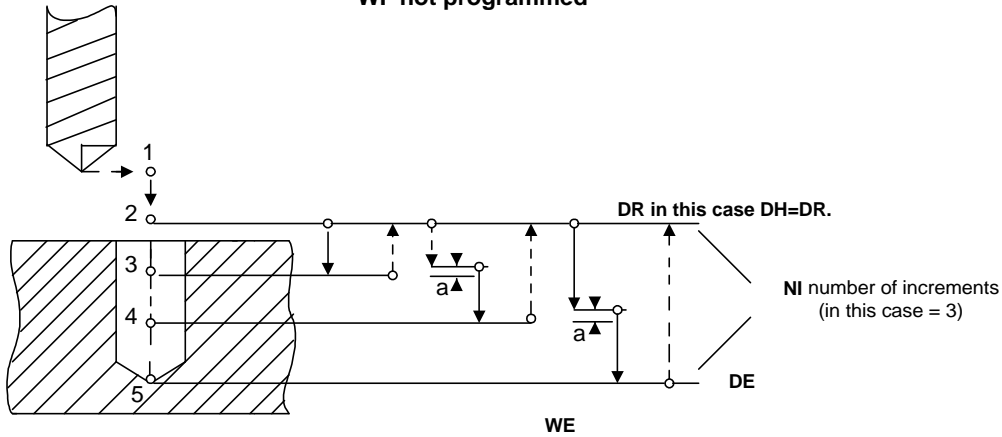
Deep drilling divided into 5 phases.  
Drilling test in sections %1183

N10 G0 X0 Y0 Z200  
(Technological parameter definition)  
N20 G83 DR125 DE100 DH125 FW150 FR15000 NI5  
(Fixed Drilling Cycle Activation)  
(P0x, P0y = 10, 10)  
(P1x, P1y = 40, 50)  
N30 X10 Y10  
N40 X40 Y50  
N50 G80(applied on line only)  
N60 M10  
N70 M30  
N80 END

## G83 Fixed Cycle: drilling with swarf discharge or breaking

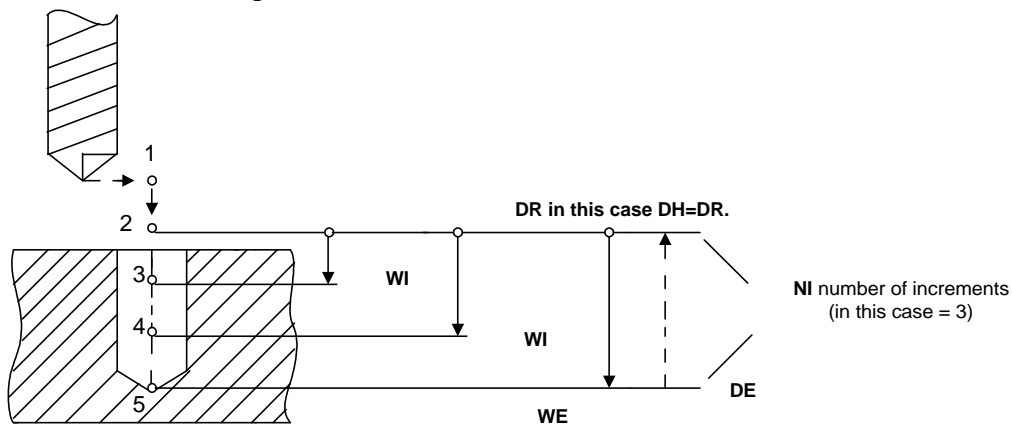
- With swarf discharge with WE other than 0.

WI not programmed



- 1) Positioning of axes X and Y (hole axis).
  - 2) Positioning of Z in fast mode DR.
  - 3-4-5) Programmed number of increment NI.
  - 6) Programmed end of hole position DE, with programmed wait time WE.
- At the end of each increment, the tool is recalled in fast mode to the position set in DH (unless DH is programmed DH=DR) to return in Feed mode at the previously machine depth minus value < a > set at 1mm.

- With swarf breaking with WE = 0



- 1) Positioning of axes X and Y (hole axis).
- 2) Positioning of Z in fast mode DR.
- 3-4-5) Programmed number of increment NI.
- Increments 3 and 4 at the same time as programmed wait time WI.
- 5) Programmed end of hole position DE, with programmed wait time WE.

Figura 4.5 - Profile executed by fixed cycle G83

### 4.2.3 G84 Fixed Cycle: Tapping

#### Syntax

**G84 DR<> DH<> DE<> RW<> RR<> WE<> P<>**

G84 Fixed Cycle Interface:

<b>DR</b>	(VA23) Position Z reached in fast mode	(Fast mode position)
<b>DE</b>	(VA3) End of hole position Z reached in Feed	(End position)
<b>DH</b>	(VA4) Home return position Z	(Home position)
<b>DW</b>	(VA5) Z axis work speed	(Revolution Work)
<b>RR</b>	(VA6) Feed return speed	(Revolution Return)
<b>WE</b>	(VA7) End of hole wait time	(Wait End)
<b>P</b>	(VA19) Pitch	(Pitch)

#### Type of function

Modal

#### Description

Performs fixed tapping cycle.

The function G80 (applied on PP line only), or a G0 Z feed (dimension) cancel G84 mode explicitly.

Parameter DE identifies the tapping end position.

Once hole positions X and Y are established as well as the workpiece approach distance, the work speed for axis Z RW must be defined; in fact a positive value sets clockwise spindle rotation and negative sets anti-clockwise rotation by entering the minus sign (-) in front of the speed RW.

Once the tapping end position is reached, a timed shutdown is activated and then spindle rotation is reversed to return to the point as specified in parameter DH at the new speed set in RH.

The parameters transferred in the call have the following characteristics:

DR: modal

DE: modal

DH: if undeclared, DH = DR

RW: modal

RR: if undeclared, value RW

WE: if undeclared, WE = 0

P modal

#### Example

Hole tapping with clockwise spindle rotation on entry, and anti-clockwise rotation on exit; therefore RW is positive.

```
TAPPING test %1184
N10 G0 X0 Y0 Z200
      (Technological parameter definition)
N20 G84 DR125 DE100 DH125 RW150 RR15000 P0.75
      (Fixed Drilling Cycle Activation)
      (P0x, P0y = 10, 10)
      (P1x, P1y = 40, 50)
N30 X10 Y10
N40 X40 Y50
N50 G80(applied on line only)
N60 M10
N70 M30
N80 END
```



#### 4.2.4 G86 Fixed Cycle: Boring

##### Syntax

**G86 DR<> DE<> DH<> FW<> FR<> WE<>**

G86 Fixed Cycle interface:

<b>DR</b>	(VA23) Position Z reached in fast mode	(Fast mode position)
<b>DE</b>	(VA3) End of hole position Z reached in Feed	(End position)
<b>DH</b>	(VA4) Home return position Z	(Home position)
<b>FW</b>	(VA5) Z axis work speed	(Feed Work)
<b>FR</b>	(VA6) Feed return speed	(Feed Return)
<b>WE</b>	(VA7) End of hole wait time	(Wait End)

##### Type of function

Modal

##### Description

Performs fixed boring cycle with wait at end of hole and return with spindle stationary.

The function G80 (applied on PP line only), or a G0 Z feed (position) cancel G86 mode explicitly.

DR: modal

DE: modal

DH: if undeclared, DH = DR

FW: modal

FR: if undeclared, max. speed (machine data)

WE: if undeclared, WE = 0

##### Example

Two holes are drilled, followed by boring by means of G86 fixed cycle

```

BORING WITH STATIONARY SPINDLE RETURN test %1185
N10 G0 X0 Y0 Z200
    (Technological parameter definition)
N20 G86 DR125 DE100 DH125 FW150 FR15000
    (Fixed Drilling Cycle Activation)
    (P0x, P0y = 10, 10)
    (P1x, P1y = 40, 50)
N30 X10 Y10
N40 X40 Y50
N50 G80(applied on line only)
N60 M10
N70 M30
N80 END
  
```

#### 4.2.5 G88 Fixed Cycle: drilling cavity walls

##### Syntax

**G88 DRA<> DEA<> DH<> DRB<> DEB<> FW<> FR<> WE<>**

G88 Fixed Cycle Interface:

<b>DRA</b>	(VA23) First wall position Z reached in fast mode	(PtoA Fast mode position)
<b>DEA</b>	(VA3) End of hole position Z reached in Feed	(PtoA end position)
<b>DH</b>	(VA4) Home return dimension Z	(Home position)
<b>DRB</b>	(VA23) Second wall position Z reached in fast mode	(Fast mode position PtoB)
<b>DEB</b>	(VA3) End of hole position Z reached in Feed	(End position PtoB)
<b>FW</b>	(VA5) Z axis work speed	(Feed Work)
<b>FR</b>	(VA6) Feed return speed	(Feed Return)
<b>WE</b>	(VA7) End of hole wait time	(Wait End)

##### Type of function

Modal

##### Description

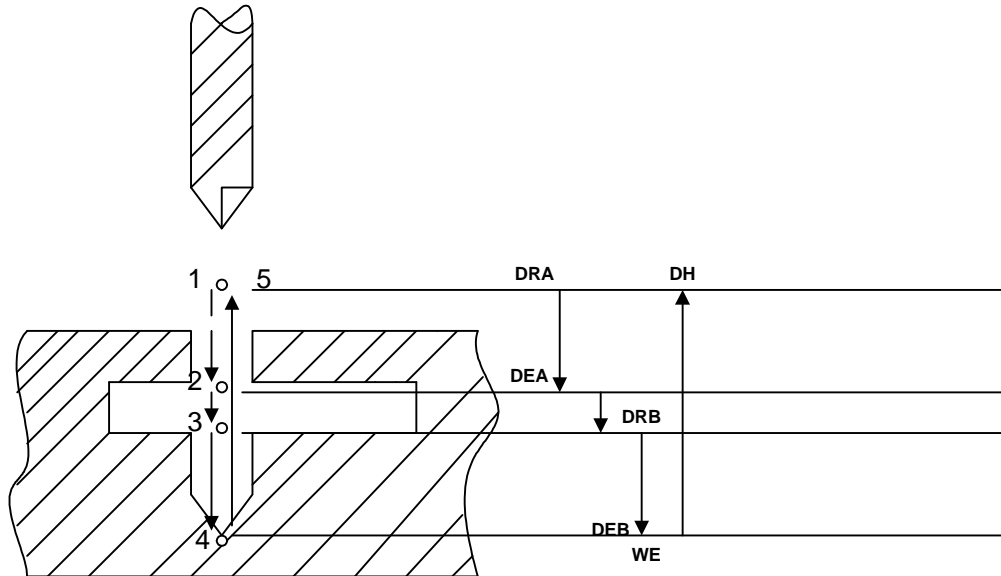
Performs fixed drilling cycle on cavity walls.  
 The function G80 (applied on PP line only), or a G0 Z feed (position) cancel G88 mode explicitly.  
 Parameters transferred in the call are self-cancelling.  
 Following approach, the first phase is started by drilling the first wall at the speed set in FW to the depth specified by DEA.  
 On completion of the first drilling phase, axis Z is activated in fast mode to reach the position set in DRB. Following this, the second wall is drilled at the speed set in FW, until the position set in DEB is reached.  
 Axis Z is then withdrawn at high speed to the position set in DH.

##### Example

Drilling on two cavity walls.

```
Drilling cavity walls:%1188
N10 G0 X0 Y0 Z200
      (Technological parameter definition)
N20 G88 DRA125 DEA100 DH125 DRB80 DEB60 FW150 FR15000
      (Fixed Drilling Cycle Activation)
      (P0x, P0y = 10, 10)
      (Plx, Plx = 40, 50)
N30 X10 Y10
N40 X40 Y50
N50 G80(applied on line only)
N60 M10
N70 M30
N80 END
```

## G88 Fixed cycle: (drilling spaced walls)



- 1) Axis **Z** approach in fast mode to position **DRA**.
- 2) Drilling to first wall to position **DEA** at speed **FW**.
- 3) **Z** fast mode to position **DRB**.
- 4) Drilling of second wall to position **DEB** at speed **FW**.
- 5) After wait time **WE**, return at speed **FR** to position **DH**.

**G88 DRA<> DEA<> DH<> DRB<> DEB<> FW<> FR<> WE<>**

*Figura 4.6 - Profile executed by G88 fixed cycle*

#### 4.2.6 G133 Fixed cycle: Thread-cutting

**Syntax**

*G133 X<exp> Z<exp> K<exp> P<exp> H<exp> A<exp> B<exp> S<exp> L<exp>  
T<exp> F<exp> V<exp> R<exp>*

*or:*

*G133 U<exp> Z<exp> K<exp> P<exp> H<exp> A<exp> B<exp> S<exp> L<exp>  
T<exp> F<exp> V<exp> R<exp>*

G133 Fixed Cycle interface:

<b>Z</b>	[VA1] Pf coordinate, Z axis
<b>X or U</b>	[VA0] Pf coordinate, X or U axis
<b>K</b>	[VA2] Thread-cutting pitch
<b>P</b>	[VA3] Thread-cutting depth ( X Machining start pos. - X Machining end pos.)
<b>H</b>	[VA4] Return height (Partial return pos. X - Coord Pf X)
<b>A</b>	[VA5] Angle of attack
<b>B</b>	[VA6] Angle (if 0: plunge threading) (if other than zero: chipping)
<b>S</b>	[VA7] Number of roughing runs
<b>L</b>	[VA8] Number of polishing runs
<b>T</b>	[VA9] Type (if 0: Constant Depth) (if 1: Constant Removal Area)
<b>F</b>	[VA10] X axis fast mode
<b>V</b>	[VA11] Z axis fast mode
<b>R</b>	[VA15] Bevel exit (if 0: through exit) (if 0.1: pull-out exit) (if greater than 0.5: bevel exit pitch)
<b>W</b>	Additional depth after the last polishing run. This is added to the target dimension of the radial axis (X or U) in order to make a further run after polishing.
*	[VA12] X axis, Pi coordinate
*	[VA13] Z axis, Pi coordinate
*	[VA14] X axis programming mode (0: radial, 1: diametric)
*	[VA16] Selected Plane (if 0: Lathe Programming plane ZX) (if 1: Lathe Programming plane ZU)

\*: registers used internally and not available for the fixed cycle call.

## Description

Performs fixed thread-cutting cycle (CHIPPING, PLUNGE-THREADING) in the case of bevel or pull-out exit, coupling a tapered thread with a length equal to the window set with parameter R.

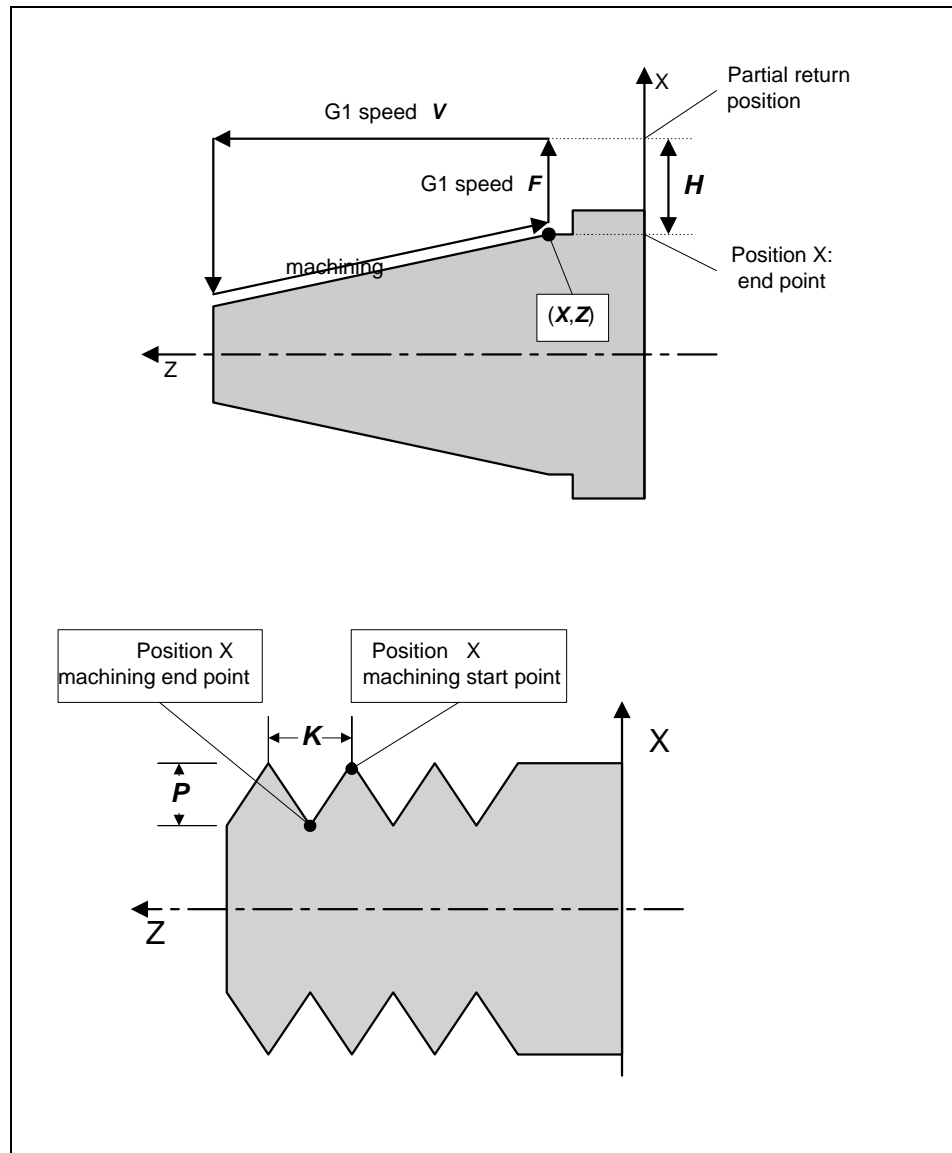
Function G80 cancels G133 mode explicitly.

Parameters transferred in the call are compulsory if G133 is displayed on the line (an alarm is activated if any are missing), the order in the list is not important.

However they are modal if G133 is not displayed and if it is still active (G80 or feed function G has not been performed).

The parameters to be used in the fixed cycle are illustrated in the figures.

The parameters regarding the positions of axis X are modified accordingly if the axis is programmed diametrically, for example in this case depth P, based on the difference between machining start point X and machining end point X, is halved automatically.



*Figura 4.7 - Geometrical processing parameters*

The figures illustrate the various types of processing according to pull-out exit parameter R.

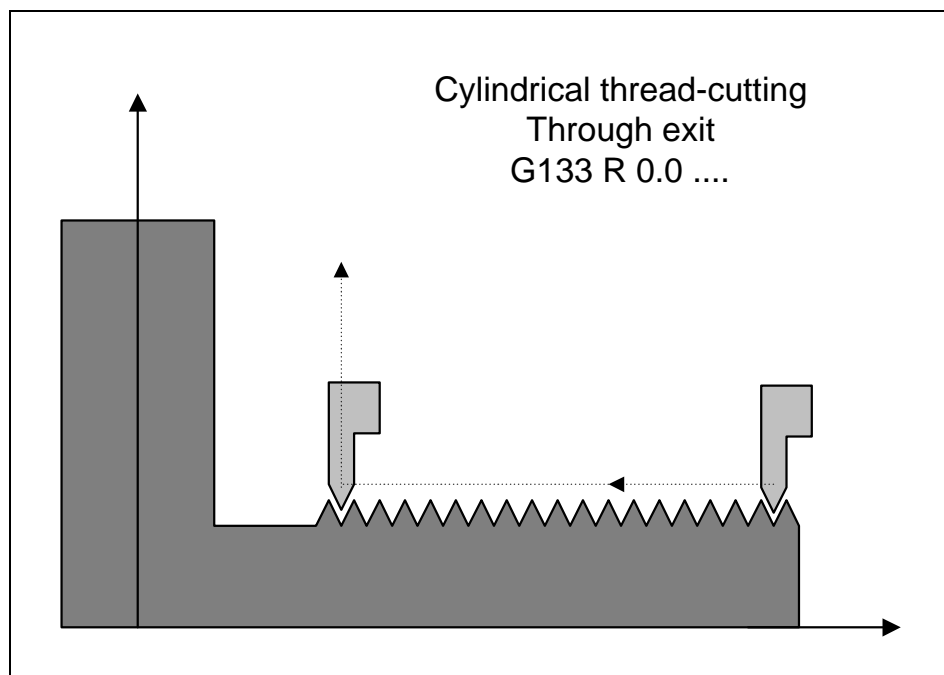


Figura 4.8 - Thread-cutting: through exit

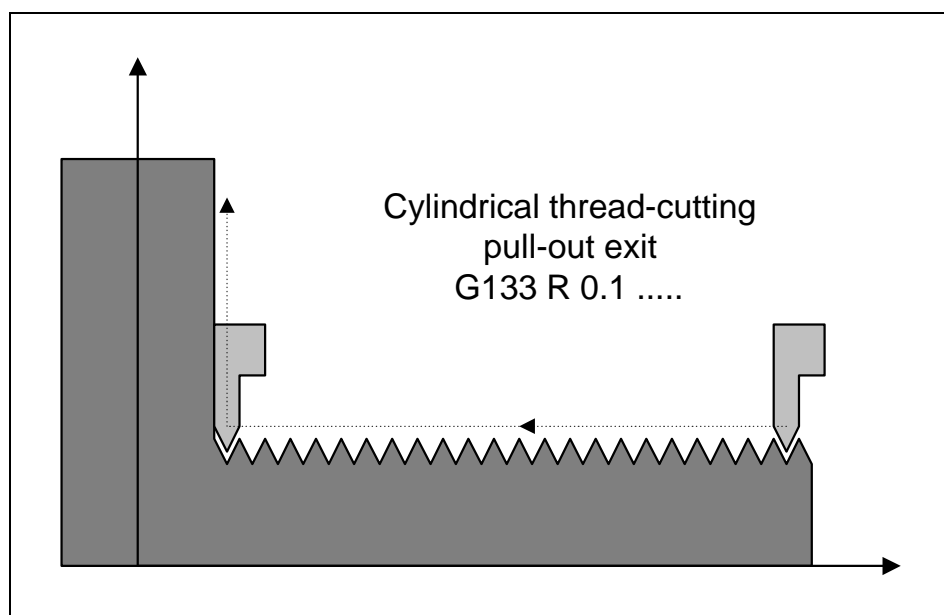


Figura 4.9 - Thread-cutting: pull-out exit

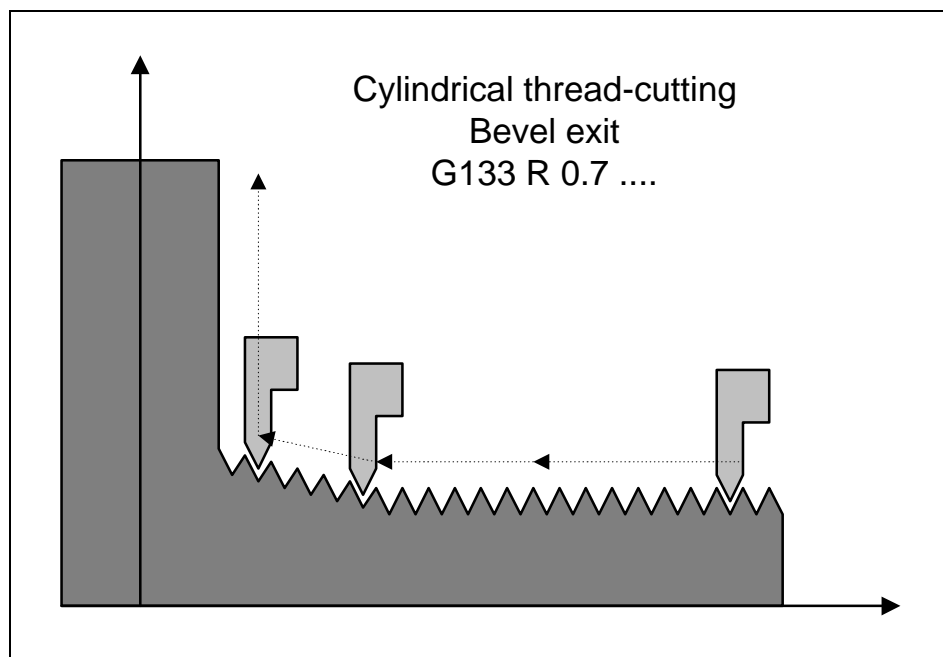


Figura 4.10 - Thread-cutting: bevel exit

The figures illustrate plung threading or chipping processes according to settings in parameter B.

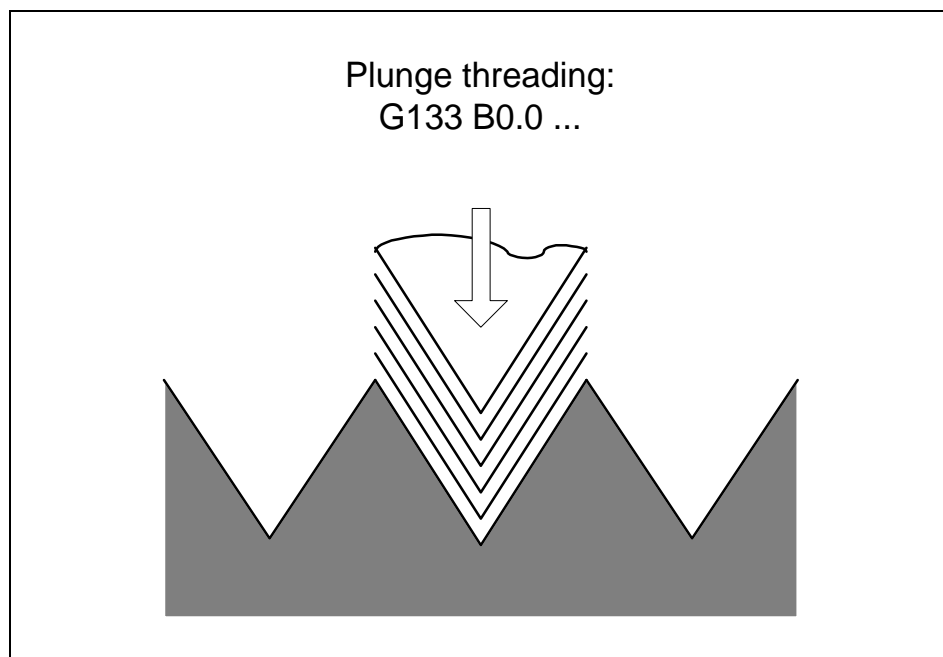
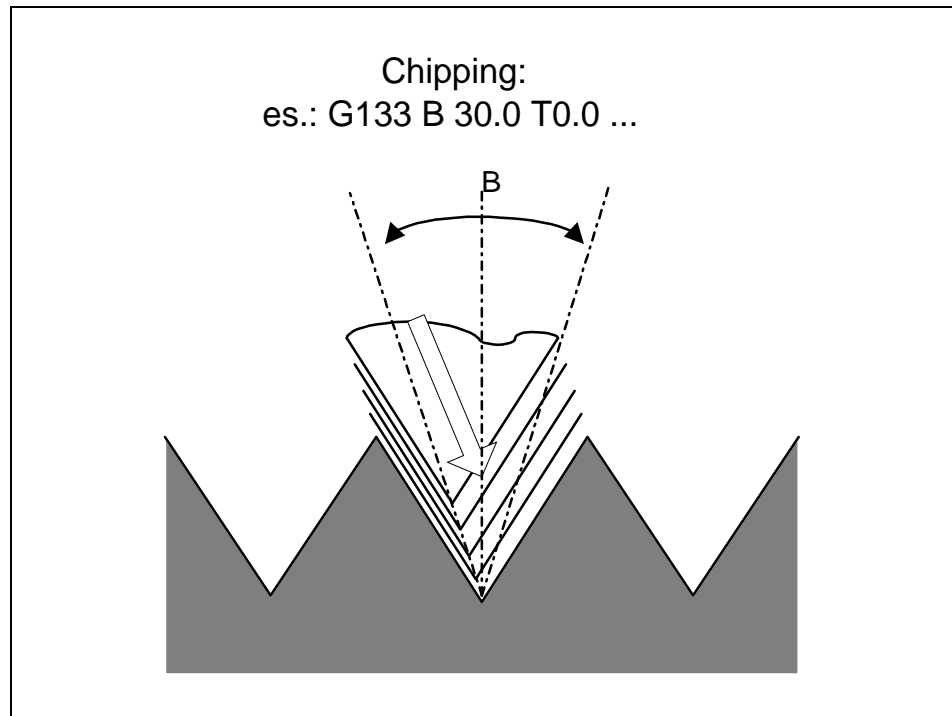


Figura 4.11 - Plunge threading





*Figura 4.12 - Thread-cutting: typical chipping parameter*

The figures illustrate constant area or depth processing according to the settings in parameter T.

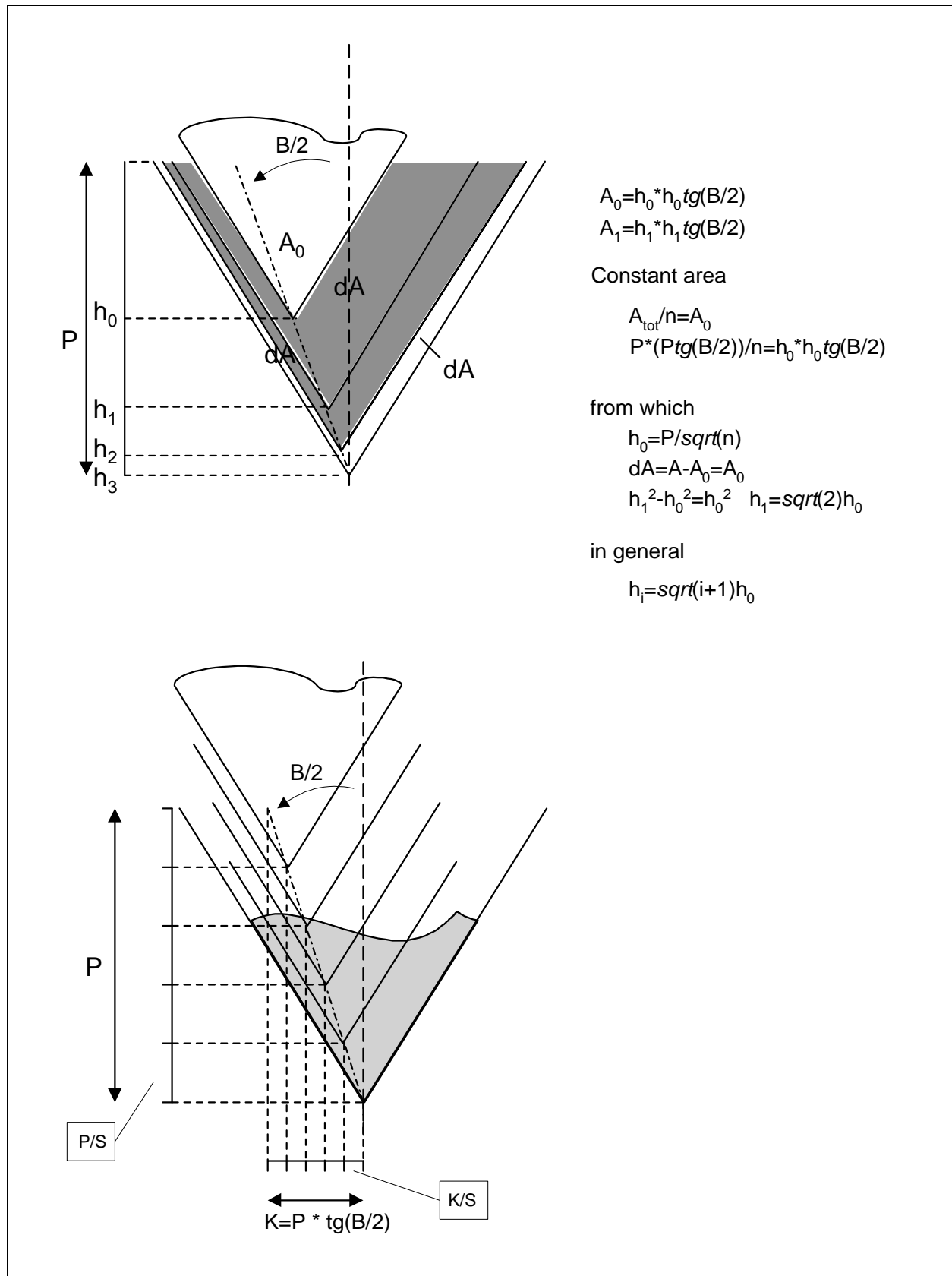


Figura 4.13 - Constant area (above) and constant depth type thread-cutting (below)

**Example**

```
3-principle thread-cutting %100
N100 G1 X<start position> Z<start position> F<>
N200 G133 X<> Z<> K<> P<> H<> A 0 B<> S<> L<> T<> F<> V<>
N300 A120
N400 A240
N500 G1 X<outside dimensions> Z<outside dimensions> F<>
```

**END OF CHAPTER**



## 5 Tool parameters

### 5.1 Definitions

<b>Machining center</b>	<p>Also <i>Machine tool</i>. Machine that includes one or more fixing devices able to clamp the same number of workpieces during the machining phase and one or more devices that actually carry out the machining operations by removing material from the workpieces.</p> <p>Machining centers are used for milling or turning, depending on whether the tool or workpiece turns to remove the material from this latter.</p>
<b>Paraxial machine tool</b>	<p>This tool machines workpieces moved in a paraxial way and includes one or more <i>heads</i>.</p>
<b>Paraxial table</b>	<p>Table with fixing devices able to clamp and locate a <i>workpiece</i>. It is sometimes driven in the paraxial mode. It is generally installed on milling machines only.</p> <p>For mechanical reasons, the table sometimes provides movement along some of the Cartesian axes and the <i>paraxial machining unit</i> along the remaining ones.</p>
<b>Head</b>	<p>Kinematical device able to position a single <i>spindle for tool changes</i> or several <i>toolholders</i> in space, and to transmit rotational movement in order to remove material from the workpiece during the machining process.</p> <p>It may have one or more degrees of freedom, or none.</p> <p>When the part being positioned is a single spindle, the actual head itself is often called <i>spindle</i>.</p>
<b>Polar table</b>	<p>Kinematical device able to position in space a surface that locates and clamps a <i>workpiece</i>, provided with a sufficient number of degrees of freedom.</p> <p>For mechanical reasons, the polar table sometimes provides some of the degrees of freedom and the <i>head</i> the remaining degrees required.</p>
<b>Tool change device</b>	<p>Device able to transmit rotational movement to a <i>coupling cone for tool changing</i> that, in turn, is able to transmit the movement to one or more toolholders.</p>
<b>Tool change cone</b>	<p>Device able to connect one or more tools clamped by means of <i>toolholders</i> to a spindle for tool changing purposes, and to transmit the rotational drive of the spindle to the former.</p> <p>The coupling cones with the relative toolholders can generally be interchanged with each other and are matched to the spindle to suit the machining operations required.</p>

<b>Toolholder</b>	Device able to clamp a single <i>tool</i> and to transmit the rotational movement of the spindle to it. Drive transmission often makes use of a mechanism that misaligns the tool to allow multiple machining operations to be carried out in combination with other tools (the entire toolholder unit is often called <i>multiple toolholder</i> ), or tilts it so that tilted machining operations can be carried out (the entire toolholder unit is often called <i>tilting head</i> ).
<b>Tool with one or more cutters</b>	Device able to remove material from the workpiece during the machining operation by means of one or more <i>cutters</i> that operate either individually or at the same time and that receive a rotational movement (milling machine) or that remain in a fixed position while the workpiece turns round them (lathe).
<b>Tool or Cutter</b>	<p>Device able to remove material from the workpiece during the machining operation. The cutter is the final part of the power train that begins with the <i>paraxial machining unit</i> and is the true cutter that generally gives rise to a <i>machining process</i> with the programmed characteristics.</p> <p><i>Tool</i> is the same as the <i>cutting edge of a tool</i> when there is only one.</p>
<b>Machining process</b>	Controlled removal of material from a <i>workpiece</i> carried out by the cutters of a tool in order to produce a surface with the required characteristics.
<b>Workpiece</b>	Part temporarily fixed to a table so that it can be machined by removal of material.

## 5.2 ER (*Entity Relationship*) diagram of the entities of a machining center of which the geometry and power train are controlled

### 5.2.1 Key

The annotation may not comply with the drawing standards.

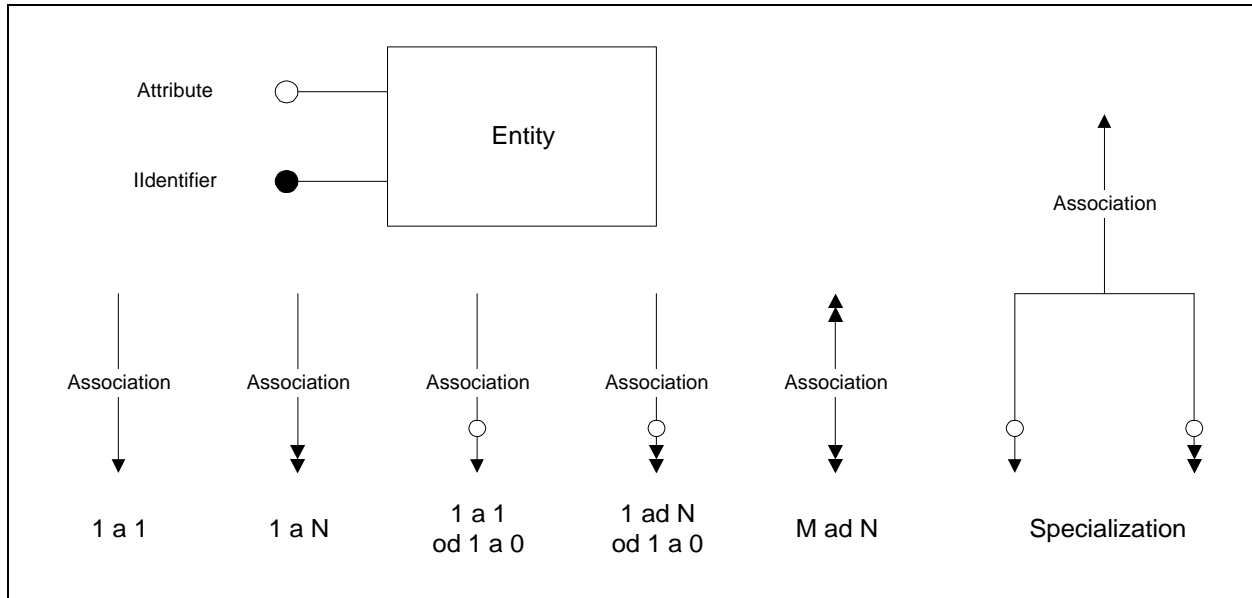


Figure 5.1 – Key to the symbols in ER (*Entity Relationship*) diagrams

### 5.2.2 Diagram

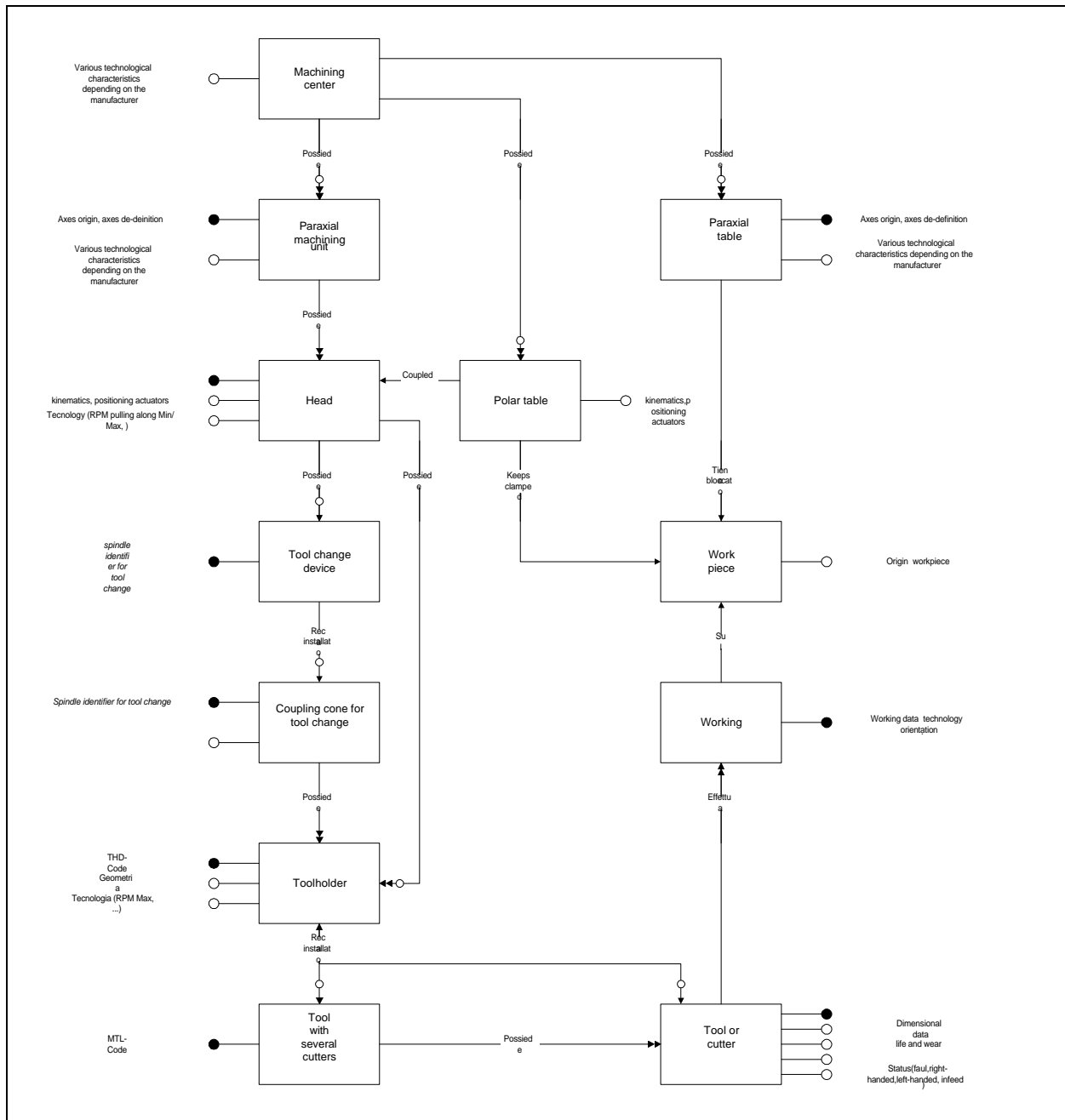


Figure 5.2 – ER diagram of the entities of a machining center of which the geometry and power train are controlled



### 5.3 Other definitions

#### **Pivot**

*Pivots* are the rotation centers of the kinematical motions that position the tool.

The terminal kinematical device controls tool tip movement while the non-terminal ones control the pivot of the next kinematical device.

When there are two kinematical devices for example, the first is the one that controls the movement of the pivot of the second one while the second and last control the movement of the tool tip.

#### **Tool Center Point Programming**

*Tool Center Point Programming* (abbreviated as *TCP* in the text) is one of the denominations commonly used for the type of management that allows the point at the top of the tool that touches the material and the machining angles to be directly programmed. All the complex operations required to establish the positions of the kinematical devices are carried out by the CNC.

The main advantages are simple programming, while the machining operations do not depend on the tool or the power train of the machine.

## 5.4 Head descriptor

This is selected from a table by means of the *TWI* command.

### 5.4.1 Data structure

Property of a head
Structure array identifier
<code>twi[n]</code>

It is advisable to specify the actual identifier in capital letters as structure label.

### 5.4.2 Description of the geometry and actuators

constant in Shared.H	field of the structure	Parameter in defcn	Meaning
	<b>XCLTWIDATA</b>		
TWI_KINE	TWI_KINEMATICS_CLASS Kine	byte yKine	Type of kinematical device: <ul style="list-style-type: none"> <li>• 1 = ROTARY_HEAD</li> <li>• 2 = FORK_ROTARY_HEAD</li> <li>• 3 = OBLIQUE_AXIS_ROTARY_HEAD</li> <li>• 4 = ROTARY_TABLE_AND_TILT_HEAD</li> <li>• 5 = FORK_TABLE</li> </ul>
TWI_MINRPM	ULONG ulMinRPM;	dword dwMinRPM	Minimum rpm rate
TWI_MAXRPM	ULONG ulMaxRPM;	dword dwMaxRPM	Maximum rpm rate
TWI_CCW	BOOLEAN bCCW;	byte yCCW	Normal left-hand rotation direction
	type: <b>ROTARY_HEAD_DATA</b>		
RH_X0	double dX0	double dX	First pivot X coordinate (ref. head zero)
RH_Y0	double dY0	double dY	First pivot Y coordinate (ref. head zero)
RH_Z0	double dZ0	double dZ	First pivot Z coordinate (ref. head zero)
RH_ORIENT	XCL_ORIENTATION Orient	byte yOrient	Z axis head positioning: <ul style="list-style-type: none"> <li>• 6 = X+</li> <li>• 4 = X-</li> <li>• 8 = Y+</li> <li>• 2 = Y-</li> <li>• 9 = Z+</li> <li>• 1 = Z</li> </ul>
RH_EC	double Dec	double dEC	Rotation around Z axis
RH_ECREV	BOOLEAN bECCRev	byte bECCRev	Reversal towards actuator of rotation around Z axis.
RH_ECAX	XCL_DRIVER ECAx	byte yECAx	Actuator of rotation around Z axis, chosen from amongst: <ul style="list-style-type: none"> <li>• 0 = NC</li> <li>• 7 = A</li> <li>• 8 = B</li> <li>• 9 = C</li> <li>• 10 = Spindle Axis</li> </ul>
	type: <b>FORK_ROTARY_HEAD_DATA</b>		

constant in Shared.H	field of the structure <b>XCLTWIDATA</b>	Parameter in defcn	Meaning
FRH_X0	double dX0	double dX	First pivot X coordinate (ref. head zero)
FRH_Y0	double dY0	double dY	First pivot Y coordinate (ref. head zero)
FRH_Z0	double dZ0	double dZ	First pivot Z coordinate (ref. head zero)
FRH_ORIENT	XCL_ORIENTA TION Orient	byte yOrient	Z axis head positioning: <ul style="list-style-type: none"> <li>• 6 = X+</li> <li>• 4 = X-</li> <li>• 8 = Y+</li> <li>• 2 = Y-</li> <li>• 9 = Z+</li> <li>• 1 = Z</li> </ul>
FRH_EC	double dEC	double dEC	Rotation around Z axis
FRH_ECREV	BOOLEAN bECRev	byte bECRev	Reversal towards actuator of rotation around Z axis.
FRH_ECAX	XCL_DRIVER ECAx	byte yECAx	Actuator of rotation around Z axis, chosen from amongst: <ul style="list-style-type: none"> <li>• 0 = NC</li> <li>• 7 = A</li> <li>• 8 = B</li> <li>• 9 = C</li> <li>• 10 = Spindle Axis</li> </ul>
FRH_X1	double dX1	double dXx	Second pivot X' coordinate
FRH_Y1	double dY1	double dYy	Second pivot Y' coordinate
FRH_EB	double dEB	double dEB	Rotation around Y axis
FRH_EBREV	BOOLEAN bEBRev	byte bEBRev	Reversal towards actuator of rotation around Y axis.
FRH_EBAX	XCL_DRIVER EBAx	byte yEBAx	Actuator of rotation around Y axis, chosen from amongst: <ul style="list-style-type: none"> <li>• 0 = NC</li> <li>• 7 = A</li> <li>• 8 = B</li> <li>• 9 = C</li> <li>• 10 = Spindle Axis</li> </ul>
FRH_X2	double dX2	double dXX	Toolholder X'' coordinate.
	double _dUnused	--	-
FRH_Z2	double dZ2	double dZZ	Toolholder Z'' coordinate.

These descriptors can be used for three-dimensional tool compensation in order to automatically calculate the position of the Cartesian axes depending on the machining coordinates, the position of the angular spindle actuators and toolholder geometry.

When an actuator is not connected, checks are made during the machining phase to make sure that the target to which an actuator should be position, net of additional rotation, is 0 (within a tolerance margin established in the machine parameters). This allows the feasibility of the machining operations to be checked when there are manually set up tools such as tilting heads, for example.

The heads can also be positioned according to machining parameters that do not depend on the kinematics.

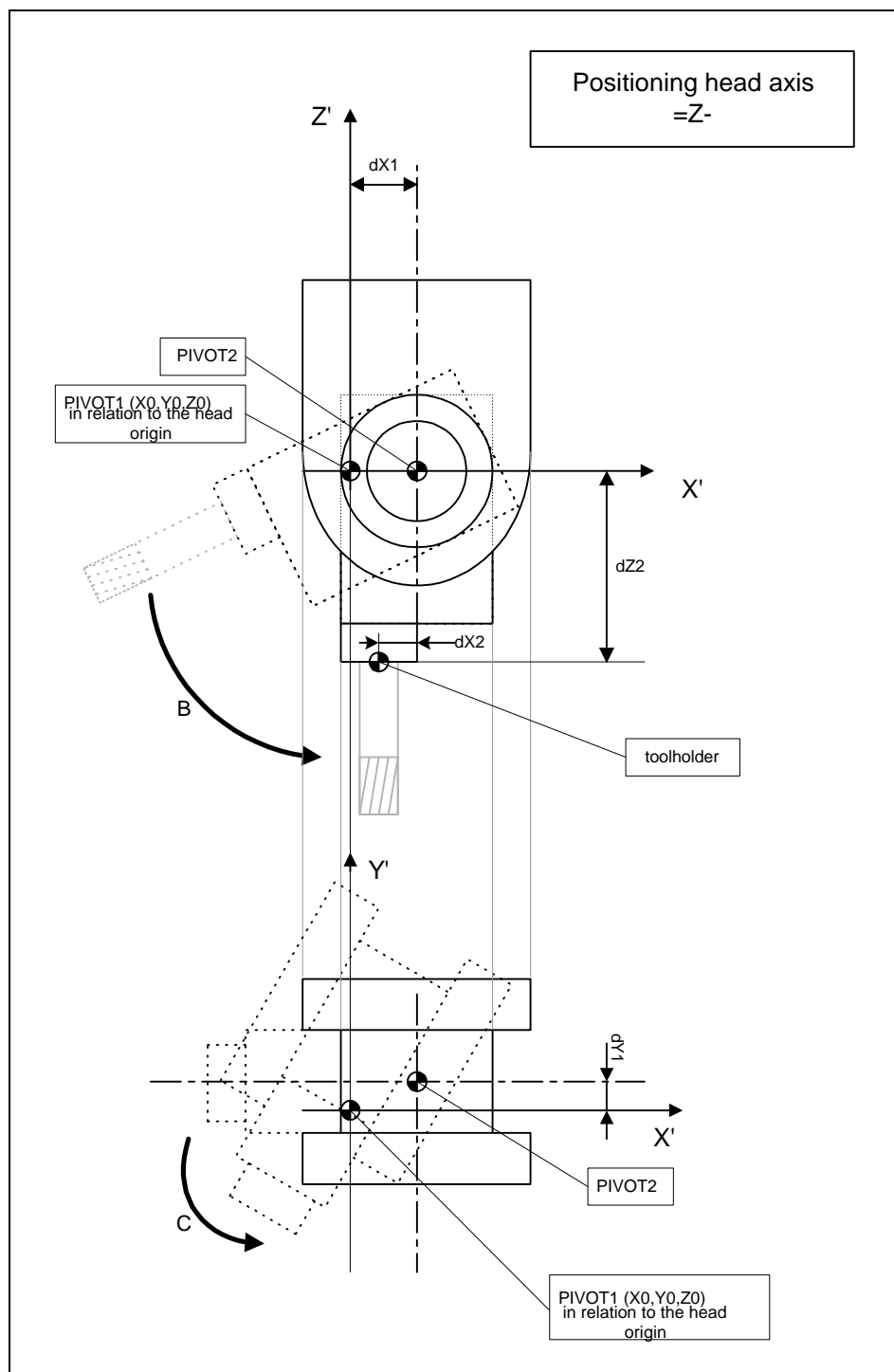


Figure 5.3 – Characteristic parameters of a FORK\_ROTARY\_AHEAD head

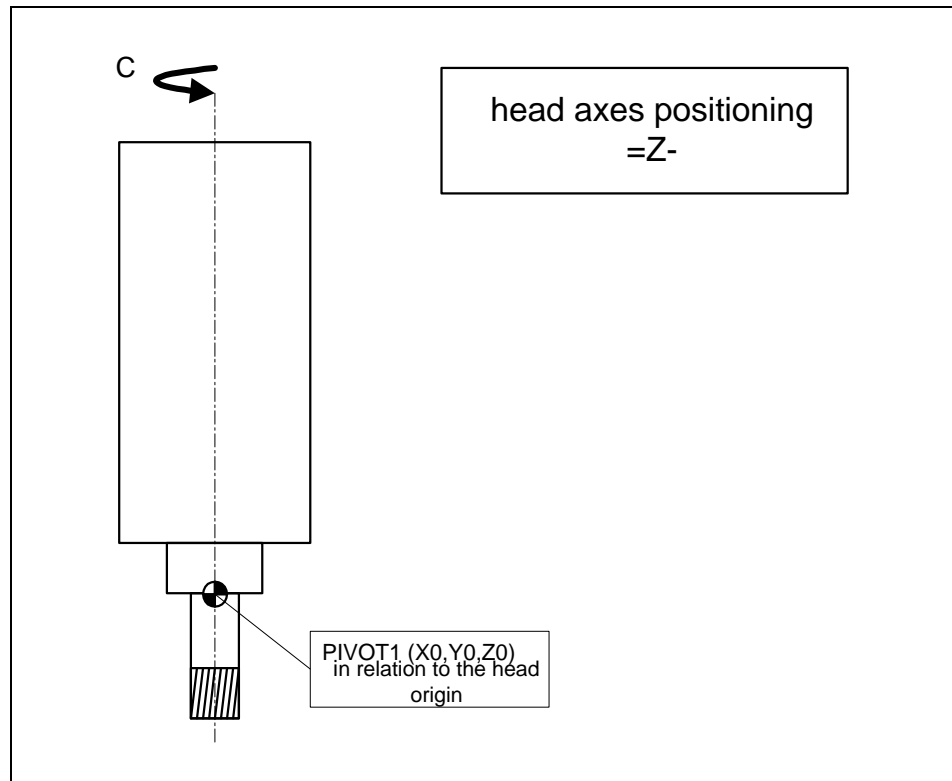


Figure 5.4 – Characteristic parameters of a rotary head

The *Positioning* field establishes the order and direction of the axes of the system of coordinates of the spindle, by identifying the new Z axis of the spindle. This indicates the tool positioning vector (that points from the top of the tool towards the toolholder) with the spindle in the reference position. 6 settings are available.

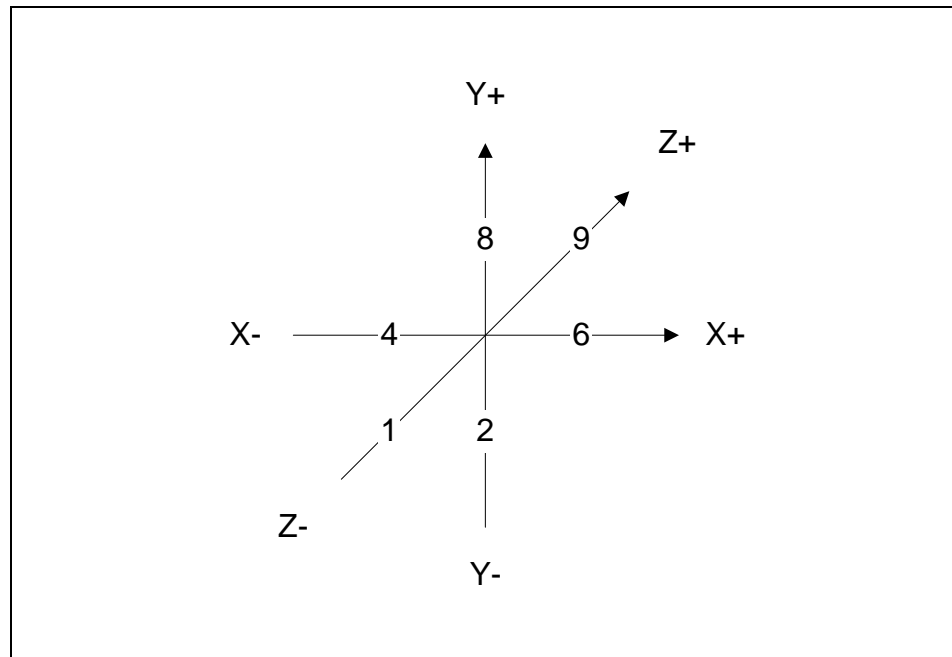


Figure 5.5 -

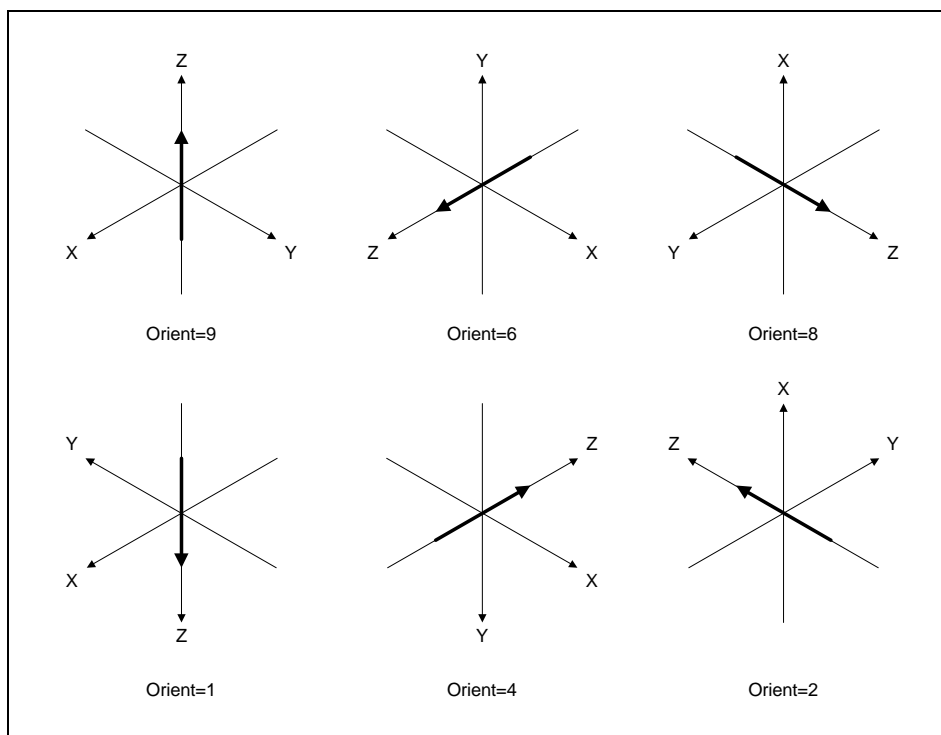


Figure 5.6 -



## 5.5 Toolholder descriptor

This is selected from a table by means of the *THD* command.

### 5.5.1 Data structure

Property of a toolholder
Structure array identifier
thd[n]

It is advisable to specify the actual identifier in capital letters as structure label.

### 5.5.2 Description of geometry

constant Shared.H	in field of the structure type <b>XCLTHDDATA</b>		Meaning
THD_Z0	double dZ0	double dZzero	Pivot Z coordinate (it is added to Z'')
THD_FLG	THD_TRANSFORMATION Transformation	byte yFlg	Order of transformations concerning pivot: <ul style="list-style-type: none"> <li>1 = First rotation around Z axis, then pivot X' Y' transfer (see <sup>(1)</sup>);</li> <li>2 = First transfer, then rotation (recommended).</li> </ul>
THD_C	Double dC	double dC	Rotation around Z axis
THD_X1	double dX1	double dTXx	Pivot X' coordinate
THD_Y1	double dY1	double dTYy	Pivot Y' coordinate
THD_B	double dB	double dB	Rotation around Y axis
• THD_X2	• double dX2	double dXX	X'' coordinate of the tool base (it is added to X')
• THD_Z2	• double dZ2	double dZZ	Z'' coordinate of the tool base (it will be added to the tool length)
<b>TECHNOLOGICAL DATA</b>			
	T H	dword U dwMaxRPM	Maximum rpm rate
	T	byte yRev	Transmission with rotation direction reversal
<b>CONTROL LOGIC</b>			
THD_IDX		word wToolIdx	Installed tool descriptor index (0=absent). Used depending on the application.
THD_CONE		dword dwCone	Univocal identifier of the coupling cone for tool changing purposes, A multiple toolholder differs from a single one owing to the presence of several toolholder cones with the same tool change coupling cone.

<sup>(1)</sup> By and large, it is used for configuring non-vertical toolholders on a head without either a numeric actuator or a manual positioning system. For this reason, it is often easier to configure the *absolute position of the toolholder and the relative positioning* than to configure the *position the toolholder would have if the head were turned so as to*

point it towards the reference axis. If it is true that the ideal solution would be to introduce a TW1 head with the toolholder's coordinates, it is also true that when there are lots of toolholders (e.g.: a boring machine), there would be a large number of head tables with the sole purpose of differentiating the Cartesian coordinates of the toolholder. However, if the toolholder was mechanically positionable and rotation introduced compensations on  $dX1$  and  $dY1$  it would be definitely necessary to configure a head.

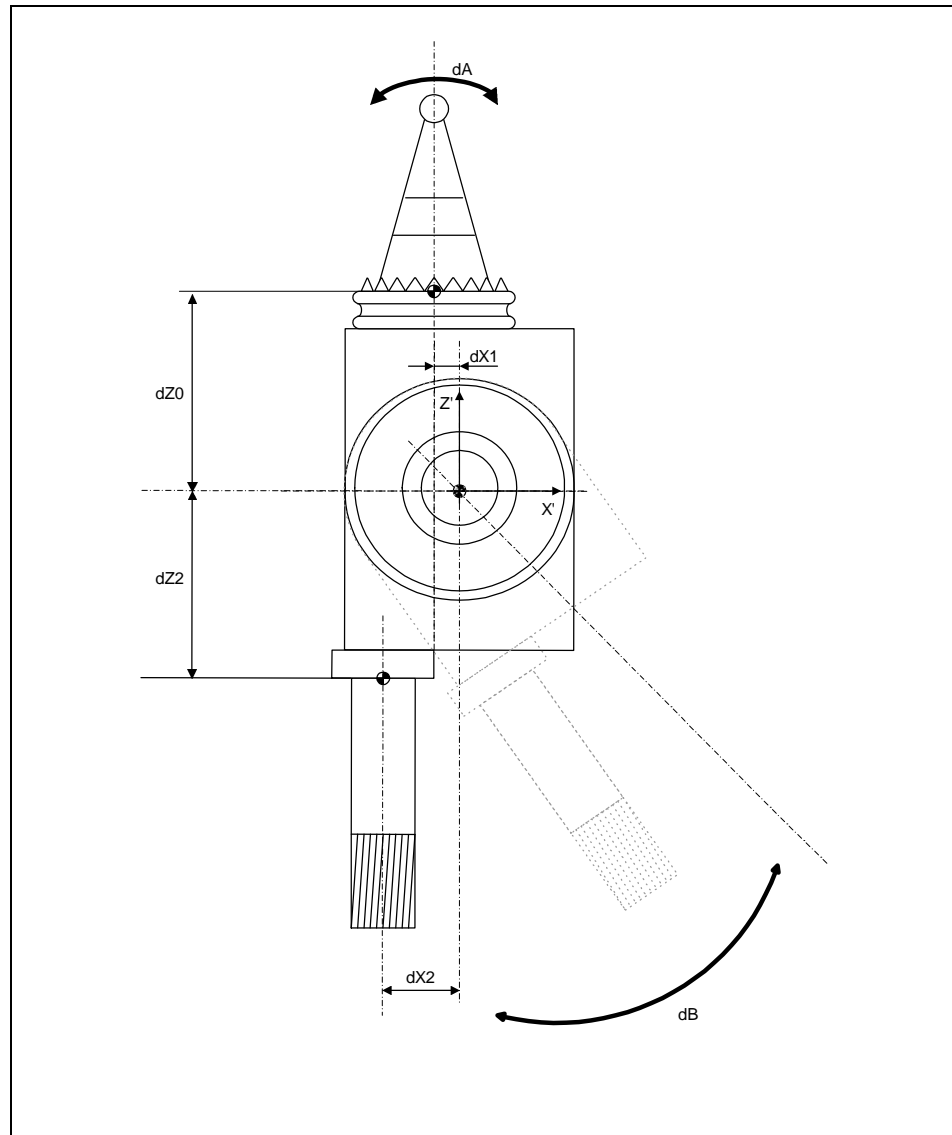


Figure 5.7 -

## 5.6 Tool or tool cutter descriptor

This is selected from a table by means of the *D* command.

To be as brief as possible, the word *tool* will be used to define both a *tool* and the *cutting edge of a tool*.

### 5.6.1 Data structure

Property of a tool
Structure array identifier
ced[n]

It is advisable to specify the actual identifier in capital letters as structure label.

### 5.6.2 Description of geometry

Property of a tool			
constant in SHARED.H	field of the structure <b>XCLCEDDATA</b>	Parameter in defcn	Meaning
CED_RAD	double dRad	double dRad	Radius
CED_TRAD	double dTRad	double dTRad	Union radius for toric mill
CED_LENZ	double dLenZ	double dLenZ	Length of the boring/milling tool (will be added to the Z'' of the tool base)
CED_THICK	double dThick	double dThick	Thickness/Distance between interchangeable cutters;
CED_LENX	double dLenX	double dLenX	Auxiliary length for turning tools (will be added to the Y'' of the tool base)
CED_LEN Y	double dLenY	double dLenY	Auxiliary length for turning tools (will be added to the X'' of the tool base)
CED_WEAIDX	--	word wWeaIdx	Tool data extension index for wear management

Several descriptors can be used in combination with one single toolholder in order to handle several cutters of the same tool of which the geometry, wear and so forth, must be differentiated.

Descriptors *dLenX*, *Y* and *Z* may or may not be used depending on the type of tool and have the meanings shown in the following figure:

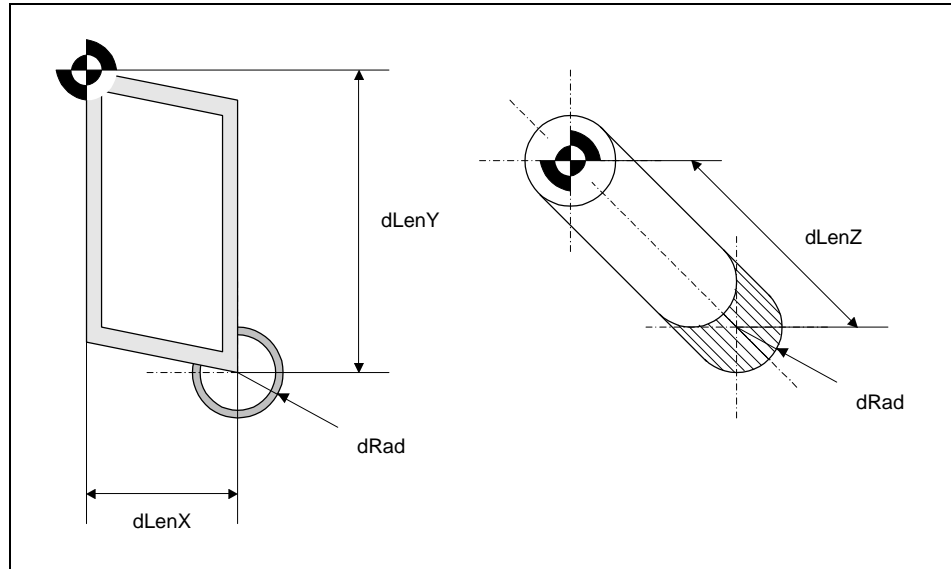


Figure 5.8 – Use of offsets on the compensation plane for lathe and milling tools

**+ = Center of the toolholder base**

Descriptor *dTRad* is used to describe the geometry of spherical and toric mills, as illustrated in the figure below:

$dTRad=0$	<i>cylindrical tool</i>
$0 < dTRad < dRad$	<i>toric tool</i>
$dTRad = dRad$	<i>spherical tool</i>

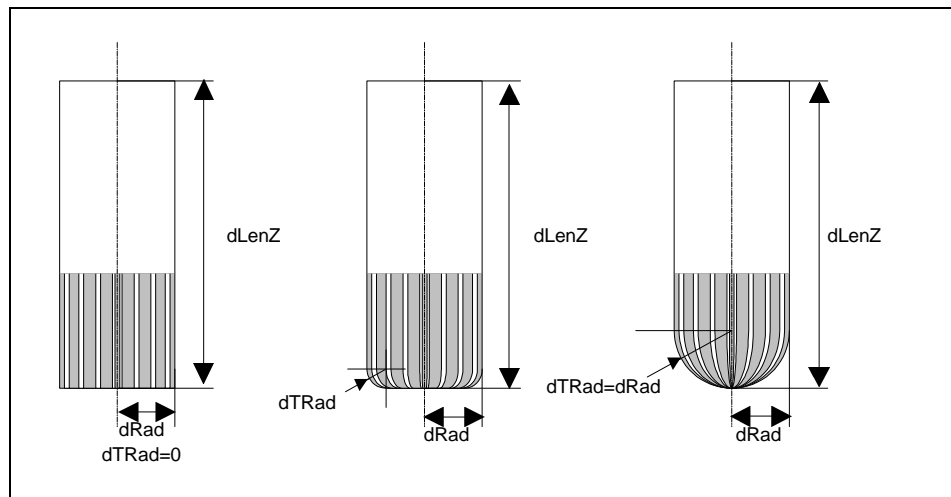


Figure 5.9 – From the left towards the right, parameters of a cylindrical tool, a toric tool and a spherical tool

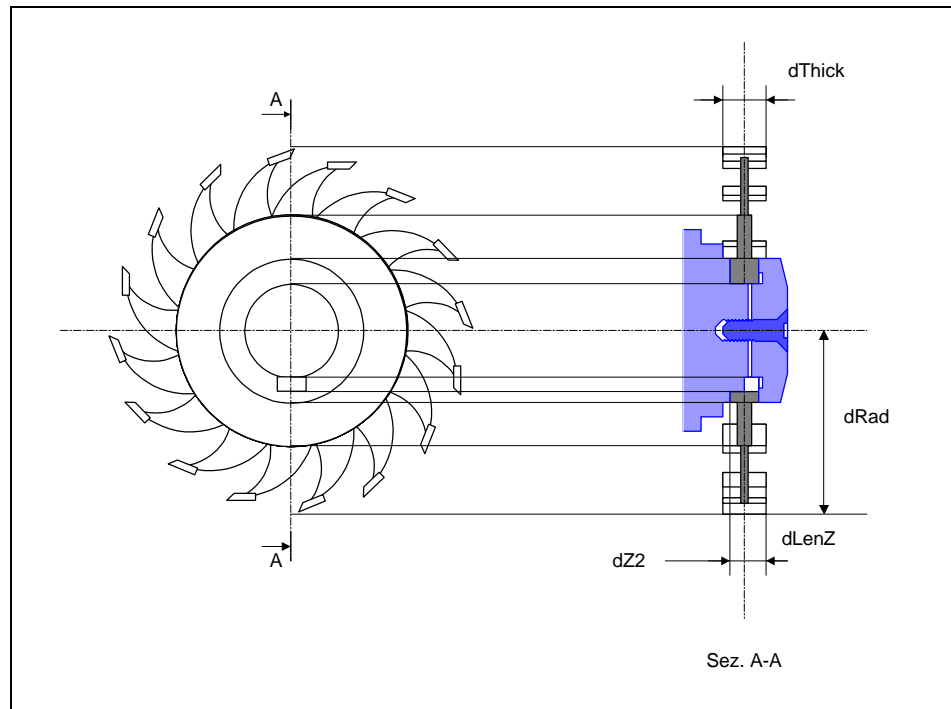


Figure 5.10 – Configuration of a tool with opposed cutting edges that can be interchanged during the machining process (disk type cutter)

### 5.6.3 Description of other technological data

Property of a tool			
constant in SHARED.H	field of the structure <b>XCLCEDDATA</b>	Parameter in defcn	Meaning
CED_MINRPM	ULONG ulMinRPM	dword dwMinRPM	Minimum rpm rate
CED_MAXRPM	ULONG ulMaxRPM	dword dwMaxRPM	Maximum rpm rate
CED_STDRPM	ULONG ulStdRPM	dword dwStdRPM	Rpm rate if not specified differently (0= do not set automatically on tool selection)
CED_CCW	BOOLEAN bCCW	byte bCCW	Left-handed tool
CED_MAXFEE D	double dMaxFeed	double dMaxFeed	Maximum linear infeed speed
CED_STDFEED	double dStdFeed	double dStdFeed	Linear infeed speed if not specified differently (0= do not set automatically on tool selection)
CED_DIR	BOOLEAN bDir	byte bDir	Enables cutting direction monitoring function
CED_TG	BOOLEAN bTg	byte bTg	Enables tool tangency to path monitoring function (e.g.: shaped blade or cutter)
CED_TGTHRE	double dTgThre	double dTgThre	Max. angle of deviation in relation to path

#### 5.6.4 Logic management

Property of a tool					
constant in SHARED.H	field of the structure <b>XCLCEDDATA</b>	Parameter in defcn	Meaning		
CED_STATUS	--	word wTStat	Events that can condition tool use		
			D2-Dn	D1	D0
			Not used.	Tool faulty ( <sup>1</sup> )	Tool not present

(<sup>1</sup>) Entered by means of a tool measuring cycle failure. Check with laser radius, etc..  
Reset when tool is changed.

#### 5.6.5 Broken tools

Locator systems could inform the CNC about tool breakage. For example, a laser could detect tool breakage once the tool has been installed on the spindle. Depending on the application, this:

- sets the CNC channel to the alarm status;
- causes searching and installation of a new tool. In this case, the CNC must be informed about how the tools are organized into groups within which the tools themselves can be interchanged with each other (tool families).

**END OF CHAPTER**

*We would be grateful for opinion about the following aspects of this manual:*

Aspect in question	Excellent	Good	Sufficient	Insufficient	Very bad
Ease with which the data can be consulted					
Way in which the descriptions of the various subjects are given					
Completeness					
Accuracy					
Readability					
Correctness of the terminology					
Page layout					
Number of illustrations					
Quality of the illustrations					

[illegible]

*You can send this form via FAX, our number is +39-059-851313, or post it to our address: Esa-Gv - Ufficio Documentazione Tecnica - 15 via Zamboni, Cp 43 - 41011 Campogalliano (Italy)*  
*<http://www.esagv.it> - E-mail: [info@esagv.it](mailto:info@esagv.it)*

