B Macro
ISO machines

Edition No. 4
Version 1.0/MC 0703

06.06.2017

Contents

1 Make your programs smart!!

1.1 What is the "B Macro"?

The B macro is a programming language whose parameters are set on FANUC numerical controls.
All your latest-generation ISO machines have this language inbuilt free of charge.
It allows you to make your programs smart. We will provide concrete examples of use in chapter 3.
Most of all, do not trust appearances, as it may appear complex the first time you read about it, but in actual fact it is very simple to use in your everyday programs.

1.2 Did you know?

The B macro is a very simple programming language, which offers you a host of possibilities.
For instance, did you know that the majority of the Tornos macros in your ISO machine consist of more than 20,000 lines of code using this very language?

2 B macro basic instructions

2.1 Variables

In order to make a program smart, you need to be able to replace values with variables.
A variable is a piece of information to which a value can be attributed.
The syntax of a variable is identifiable by a "#" and its identification number.

Variable example: #153
Example of attribution of a value to a variable: #153 = 12.4 *(thus, variable #153 contains the value 12.4)*
Once your variables contain a value, they can for instance be added, multiplied, be used as a position, as a speed, as a feed or even used in a conditional expression.

Here are the variables you can use.

The null variable:
The null variable is a variable that never contains a value.

The variable is: **#0**

Local variables:
Local variables are variables which are reinitialised to "null" as soon as you quit the program in which a value has been attributed to them.

These variables are: **#1 - #33**

Global variables per channel:
Global variables per channel are variables which are not reinitialised to "null" when you quit the program in which a value has been attributed to them.
The fact that they are per channel means that when you attribute a value to a variable, the variable will contain this value only in the channel in which the variable was attributed a value.

These variables are: **#150 - #199**

Common global variables:
Common global variables are variables which are not reinitialised to "null" when you quit the program in which a value has been attributed to them.
The fact that they are common means that when you attribute a value to a variable, the variable will contain this value in all the channels of the machine.

These variables are: **#600 - #699**

System variables:
System variables can be used to read and write the numerical control data, such as tool compensation variables, axes position data. etc.
For more information on system variables, see the FANUC instructions.

These variables are: **> #1000**

2.2 Arithmetical and logical operations

Arithmetical operations:

The most commonly used arithmetical operations are:

Addition	"+"
Subtraction	"-"
Multiplication	"*"
Division	"/"

Use example:
#603 = [#601 + #602] / 4

Functions:

The most commonly used functions are:

Sine	"SIN[#...]"
Cosine	"COS[#...]"
Tangent	"TAN[#...]"
Square root	"SQRT[#...]"
Absolute value	"ABS[#...]"
Power	"POW[#..., #...]"
Rounded down to the next integer	"FIX[#...]"
Round the value	"ROUND[#...]"

Use example:
#603 = COS[#602]

N.B.: the angle unit used is the degree. For example: 90 degrees and 30 minutes is written 90.5 degrees.

Relations operators:

Relations operators allow the comparison of two variables in a conditional expression.

The relations operators are:

Equal to	"EQ"
Not equal to	"NE"
Greater than	"GT"
Greater than or equal to	"GE"
Less than	"LT"
Less than or equal to	"LE"

Logical operations:

Logical operations allow you to test several conditions in a single conditional expression.

The most commonly used logical operations are:

Logical AND	"AND"
Logical OR	"OR"

2.3 Connections and repetitions

The "GOTO" conditional instruction:

The instruction "N*n*" at the beginning of the line allows you to indicate the block number.
The instruction "GOTO*n*" is very simple to understand, GOTO 5 means: skip to block N5.

Use example:

```
...
GOTO 5
G0 X2 T35
M103 S200
G0 Z6
N5 G0 X3 T26
...
```

This part of the program is not run by the NC

The conditional instruction "IF" - "THEN":

The instruction "IF" means: only **if** the conditional expression is satisfied, what follows is run.
The instruction "THEN" means **then**.

Use example:

IF [#600 EQ #601] THEN #602 = 18 *If the values of #600 and #601 are identical then #602 assumes the value 18*

N.B.: the "EQ" can be replaced by other relations operators

The conditional instruction "IF" - "GOTO":

The instruction "IF" means: only **if** the conditional expression is satisfied, what follows is run.
The instruction "GOTO" means **go to**.

Use example:

```
IF [#600 EQ #601] GOTO 10
G0 X2 T35
M103 S200
G0 Z6
...
...
N10 G0 X3 T26
```

If the values of #600 and #601 are identical then skip to block N10

This part of the program is run by the NC only if #600 is different from #601

N.B.: the "EQ" can be replaced by other relations operators

The repetition instruction "WHILE":

The instruction "WHILE" means **loop**.
The instruction "DO" means **do**.

Use example:

```
WHILE [#600 GT #601] DO1
G0 X2 T35
M103 S200
G0 Z6
...
...
#601 = #601 + 1
END1
```

This part of the program is run by the NC while the value of #600 is greater than #601

(The increase in the value of #601 will allow you to quit the

N.B.: the "GT" can be replaced by other relations operators

Example of embedded loops:

```
WHILE[...] DO1
...
...
    WHILE[...] DO2
    ...
    ...
        WHILE[...] DO3
        ...
        ...
        ...
        END3
    ...
    ...
    END2
...
...
END1
```

N.B.: up to 3 loops can be embedded into one another

2.4 Sub-programs

The advantages of a sub-program:

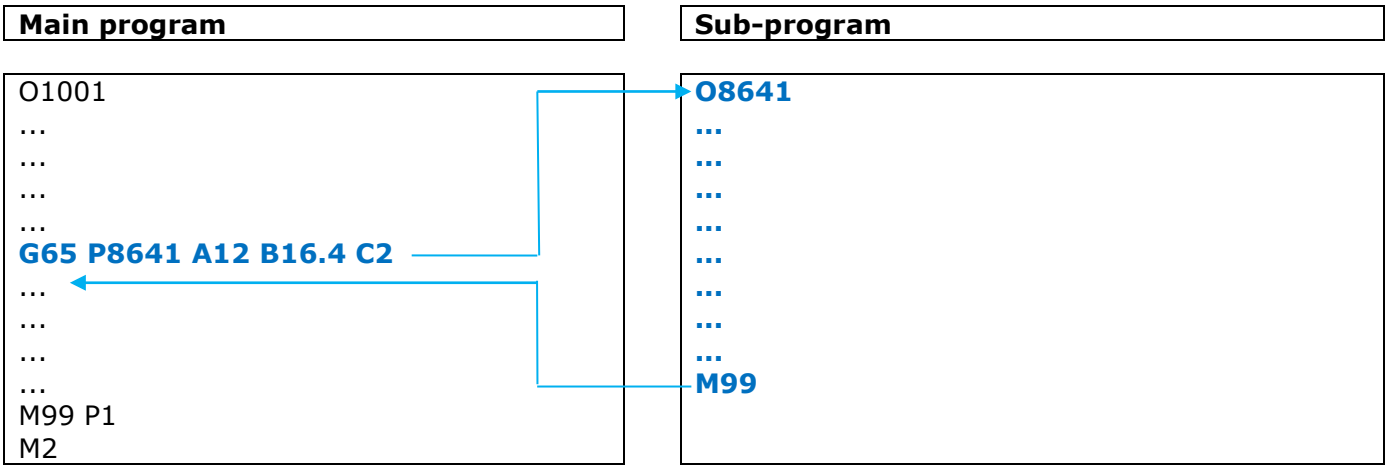
The main advantage of a sub-program lies in the fact that you can call it several times without having to recode it.
It can therefore be called several times in a single program, but it can equally be called from several different programs.

The second advantage of a sub-program is that you can use parameter setting arguments within it.

Sub-program call:

A sub-program is encoded, named and transferred to the machine in the same way as a main program.
Name example "O8641".
A sub-program is called using function G65 P*n* {A*n* B*n* C*n* ...}.

Example:
G65 P8641 A12 B16.4 C2 *Sub-program O8641 is called and arguments A, B, C are sent to it.*



The arguments of a sub-program:

Sending parameter setting arguments to a sub-program is optional.
If you wish to use them, the values of arguments are sent automatically in the local variables according to the table below:

Address	Variables number		Address	Variables number		Address	Variables number
A	#1		I	#4		T	#20
B	#2		J	#5		U	#21
C	#3		K	#6		V	#22
D	#7		M	#13		W	#23
E	#8		Q	#17		X	#24
F	#9		R	#18		Y	#25
H	#11		S	#19		Z	#26

2.5 Displaying an alarm

An alarm can also be displayed on the NC as follows:

#3000 = 1 (ALARM) *When the NC reaches this block the alarm "MC3001 ALARM" will be displayed and will block interpretation.*

Use example:

```
IF [#600 GE 0] GOTO 10
      #3000 = 2 (NEGATIVE VALUE E
N10
```

If #600 is less than 0 the alarm "MC3002 NEGATIVE VALUE ERROR" will be displayed and will block interpretation.

2.6 Displaying a message

A MESSAGE can be displayed on the NC as follows:

#3006 = 1 (MESSAGE) *When the NC reaches this block, the message "MESSAGE" will be displayed but interpretation will not be blocked.*

Use example:

```
IF [#600 GE 0] GOTO 10
      #3006 = 2 (CAUTION NEGATIVE VA
N10
```

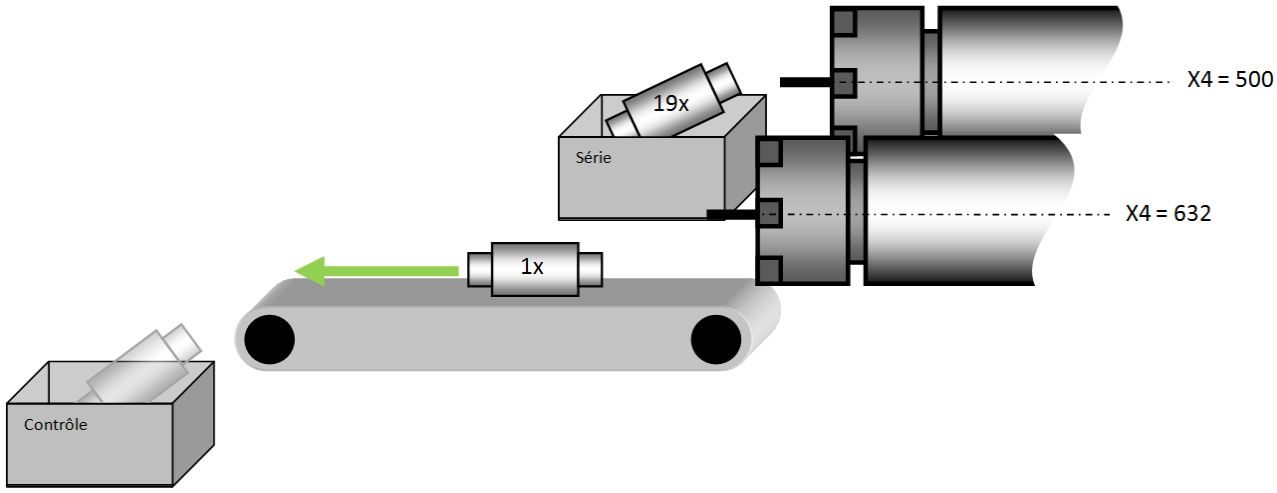
If #600 is less than 0 the message "CAUTION NEGATIVE VALUE" will be displayed but interpretation will not be blocked.

3 Concrete use example

3.1 Part sampling

Let's imagine you are producing a series of parts that requires a part check every 20 cycles.
Let's see how the B macro can help.

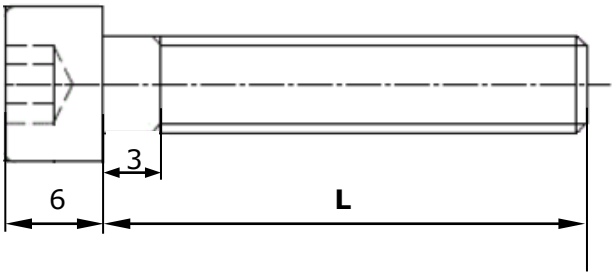
Principle:
The principle lies in the fact that the part is ejected 19 times into the catcher inside the machine and a single time onto the part conveyor so that it can be checked outside the machine.



Programming:

Counter spindle channel	
#600 = 0	**(CYCLE COUNTER INITIALISATION)**
#601 = 20	**(PART TO BE CHECKED EVERY NUMBER OF CYCLES)**
#602 = 500	**(POSITION X4 EJECT SERIES)**
#603 = 632	**(POSITION X4 EJECT FOR CHECK)**
…	
…	
N1 M120	**(LOOP START)**
IF [#600 EQ #601] THEN	**(AT CYCLE 20 RESET COUNTER TO 0)**
#600 = 0	**(COUNTER INCREMENTATION)**
#600 = #600 + 1	
…	
…	
IF [#600 EQ #601] GOTO 10	**(CYCLE DIFFERENT FROM 20TH EJECT AT X500)**
G53 X500	
GOTO 11	
N10	**(CYCLE NUMBER 20 EJECT AT X630)**
G53 X632	
N11	**(EJECTION)**
M84	
…	
…	**(LOOP END)**
M121	

3.2 Family of parts



Let's imagine you are producing a range of screws.
All the screws are identical with the exception of their length "L".
It may be interesting to retain a single program for all the screws instead of having one program for each screw.

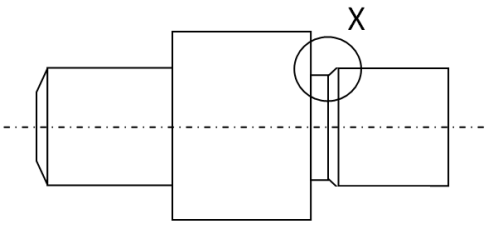
Channel 1

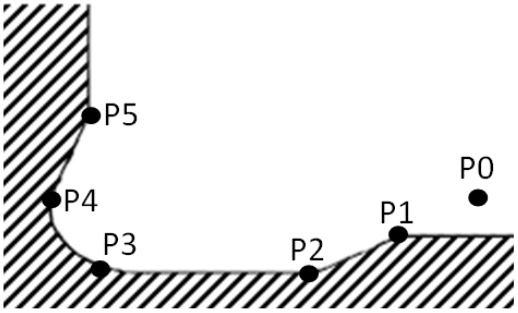
```
#600 = 53427                          (SCREW ID NUMBER 53426, 53427, 53428, ...)
IF [#600 EQ 53426] GOTO 5
IF [#600 EQ 53427] GOTO 10
IF [#600 EQ 53428] GOTO 15
IF [#600 EQ 53429] GOTO 20
IF [#600 EQ 53430] GOTO 25
N5 #601 = 10                          (LENGTH "L" FOR SCREW 53426)
GOTO 30
N10 #601 = 12                         (LENGTH "L" FOR SCREW 53427)
GOTO 30
N15 #601 = 15                         (LENGTH "L" FOR SCREW 53428)
GOTO 30
N20 #601 = 20                         (LENGTH "L" FOR SCREW 53429)
GOTO 30
N25 #601 = 22                         (LENGTH "L" FOR SCREW 53430)
N30
G800 A10 B[6+#601] C[#601+2]          (CYCLE B DATA: PART LENGTH, C: (PART PICKOFF
...                                   LENGTH)
...
N1 M120
...                                   (LOOP START)
...
G0 X5 Z2 T12 D0
G1 Z-#601 F0.06
G1 X12                                (TURNING OF SPAN)
...
...
G78 P020060 Q500 R0.02
G78 X4.2 Z-[#601-3] R0 P4300          (THREADING)
Q900 F0.7                             (THREADING)
...
...
M121                                  (LOOP END)
...
```

Thus, you simply need to change the ID number of the screw on the first line of the program to change screw.

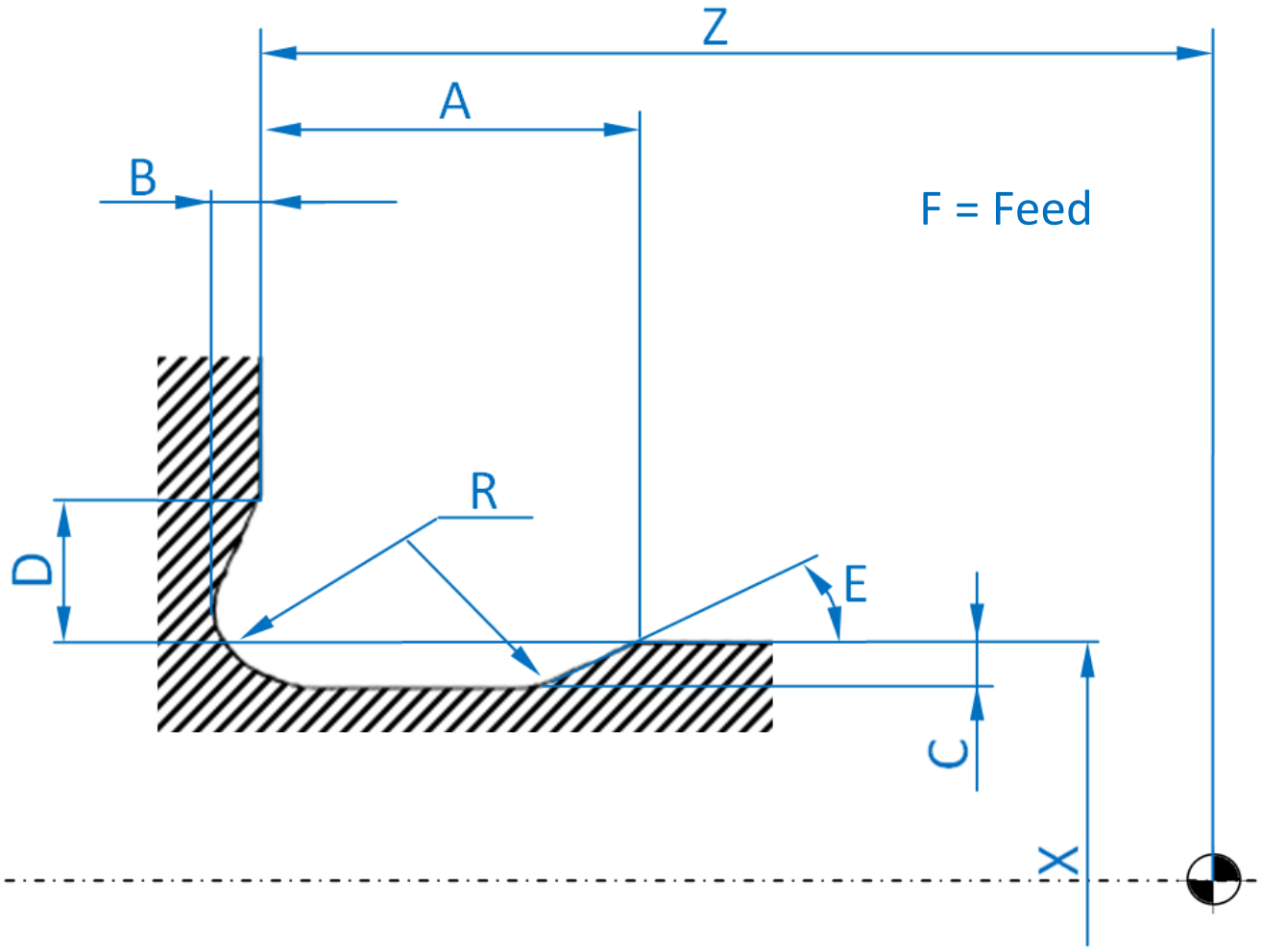
3.3 Your own machining macro

Let's imagine you are required to program regularly on your parts some channels on which you need to calculate several points. To make your life simpler, you could create your very own machining macro.





Programming a macro:



Main program

```
...
G65 P8512 A2 B0.1 C0.25 D1.15 E30 F0.05      (SUB-PROGRAM O8512 CALL)
R0.6 X12 Z36
...
```

Sub-program O8512 *(Machining macro)*

```
(*** STORAGE OF ARGUMENTS ***)
#600 = #1                                (ARGUMENT A)
#601 = #2                                (ARGUMENT B)
#602 = #3                                (ARGUMENT C)
#603 = #7                                (ARGUMENT D)
#604 = #8                                (ARGUMENT E)
#605 = #9                                (ARGUMENT F)
#606 = #18                               (ARGUMENT R)
#607 = #24                               (ARGUMENT X)
#608 = #26                               (ARGUMENT Z)
#609 = 2                                 (SAFETY CONSTANT)

(*** CALCULATION OF P0 ***)
#611 = #607 + #609                       (P0 X)
#612 = -[ABS[#608] - #600 -              (P0 Z)
[[#609/2]/[TAN[#604]]]]
(*** CALCULATION OF P1 ***)
#613 = #607                              (P1 X)
#614 = -[ABS[#608] - #600]               (P1 Z)

(*** CALCULATION OF P2 ***)
#615 = #607 - [#602 * 2]                 (P2 X)
                                         (P2 Z)
#616 = -[ABS[#608] - #600 +
[#602/TAN[#604]]]
(*** CALCULATION OF P3 ***)               (P3 X)
#617 = #615                              (P3 Z)
#618 = -[ABS[#608] + #601 - #606]

(*** CALCULATION OF P4 ***)               (P4 X)
#619 = #615 + [#606 * 2]                 (P4 Z)

#620 = -[ABS[#608] + #601]
                                         (P5 X)
(*** CALCULATION OF P5 ***)               (P5 Z)
#621 = #607 + [#603 * 2]

#622 = -[ABS[#608]]

(*** ISO CODE ***)                        (P0)
G90 G95                                  (P1)
G0 Y0                                    (P2)
G0 X#611 Z#612                           (P3)
G1 X#613 Z#614 F#605                     (P4)
G1 X#615 Z#616 ,R#606 F#605              (P5)
G1 X#617 Z#618 F#605
G2 X#619 Z#620 I#606 K0 F#605            (SUB-PROGRAM EXIT)
G1 X#621 Z#622 F#605

M99
```

3.4 Discontinuous machine cleaning

Let's imagine you are producing a series of parts, for which you need to clean on a regular basis the inside of the machine in order to eject the chips.
Let's see how the B macro can help.

Principle:
The principle lies in cleaning your cutting tools regularly using a high pressure pump but without the latter operating continuously.
The advantage of using this process is that you reduce the noise and electricity consumption of your workshop.
In the example below, we are also taking advantage of cleaning the counter spindle collet with air.

Counter spindle channel

```
#600 = 0                    (CYCLE COUNTER INITIALISATION)
#601 = 100                  (MACHINE TO BE CLEANED EVERY NUMBER OF CYCLES)
...
N1 M120                     (LOOP START)
IF [#600 EQ #601] THEN      (AT CYCLE 100 RESET COUNTER TO 0)
#600 = 0                    (COUNTER INCREMENTATION)
#600 = #600 + 1
...
M11                         (EJECTION OF PART FROM COUNTER SPINDLE COLLET)
M84
...
...                         (EVERY 100 CYCLES THE CODE IS RUN UP TO N10)
IF [#600 NE #601] GOTO      (VALVE 1 OPENING)
10                          (START HIGH PRESSURE PUMP)
M532 M11 M1                 (TOOL SYSTEM 1 CLEANING)
M532 M1 M1                  (VALVE 1 CLOSING)
G4 X4                       (VALVE 2 OPENING)
M532 M11 M0                 (TOOL SYSTEM 2 CLEANING)
M532 M12 M1                 (VALVE 2 CLOSING)
G4 X4                       (VALVE 3 OPENING)
M532 M12 M0                 (TOOL SYSTEM 3 CLEANING)
M532 M13 M1                 (VALVE 3 CLOSING)
G4 X4                       (STOP HIGH PRESSURE PUMP)
M532 M13 M0                 (COUNTER SPINDLE CENTRE BLOWER - 3 SEC. FOR COLLET
M532 M1 M0                  CLEANING)
M841 M2 M3000
N10
...
...                         (LOOP END)
M121
...
```

3.5 Default machine stop

Let's imagine your machine is producing all week but that you need to stop production on Saturday evening to avoid tool wear from producing parts out of tolerance until Monday morning.
It may be interesting for you not to have to go back tot he factory on Saturday evening to stop the machines. Let's see how the B macro can help.

Principle:
The principle lies in checking at each cycle the current date and time of the NC, and of stopping the machine when the configured date and time are reached.

In the example below, we will stop the machine
on: 24.06.2017 (#600)
at: 20:35:00 (#601)

Channel 1	
#600 = 203500	**(STOP TIME: HOUR - MINUTE - SECOND)**
#601 = 20170624	**(MACHINE STOP DATE: YEAR - MONTH - DAY)**
…	
…	
N1 M120	**(LOOP START)**
…	
…	
IF [#601 NE #3011] GOTO 10	**(CHECK WHETHER TODAY IS THE MACHINE STOP**
IF [#600 LT #3012] GOTO 10	**DATE)**
M105	**(CHECK WHETHER THE MACHINE STOP TIME HAS**
M405	**PASSED)**
M1105	**(SPINDLES STOP)**
M9	
M0	
N10	**(OIL STOP)**
M121	**(CYCLE STOP)**
…	
	(LOOP END)

N.B.: #3011 = system variable that indicates the current date on the NC (year, month, day)
#3012 = system variable that indicates the current time on the NC (hour, minute, second)

4 Good to know

4.1 Cycle time

We recommend you encode in B macro everything you can before the machining loop *(before M120)*. This will allow you to minimise as much cycle time wastage as possible tied to the processing of conditions and calculations.

4.2 Tornos training

You may be interested to know that Tornos offers parameter programming training schemes, so you can become a genuine specialist and use all the possibilities of this language in the best possible way.

4.3 FANUC instruction

FANUC instruction B-63944 explains all the language possibilities in full.