CNC DATA

Macro B Programming Manual

Ben Groves

www.cncdata.co.uk

Although subprograms are useful for repeating the same operation, the custom macro function also allows use of variables, arithmetic and logic operations, and conditional branches for easy development of general programs such as pocketing and user–defined canned cycles. A machining program can call a custom macro with a simple command, just like a subprogram, the only difference being; we can pass information into the sub program and manipulate it as we want.

Main Program	Sub Program
O0001; ; ; G65 P9010 A1. B26. F500. ; ; M30;	O9010; G91; N100 #101=#2/2 G#1 G42 X#101 Y#1 F#9 IF[#5021LT100]GOTO100; M99;

In the world of Macro B, everything revolves around variables, that is because 90% of the information visible on a Fanuc control, has its own variable address, these are called System Variables. Fanuc has also given the end user its own set of variables, two types, local and common, located: [OFFSET] – {MACRO} (see page 5).

Here are some of the System variables available:

- Tool Offsets
- Work Offsets
- Axis Positions
- Modal Information
- PMC Signals
- Alarms
- Automatic Operation Control
- Timers and Counters

Plus many more

An ordinary machining program specifies a G code and the travel distance directly with a numeric value; examples are G01 X100.0
With a custom macro, numeric values can be specified directly or using a variable number. When a variable number is used, the variable value can be changed by a program or using operations on the MDI panel.

```
#2=0
#1=#2+100;
G01 X#1 F200;
```

When specifying a variable, specify a number sign (#) followed by a variable number. General–purpose programming languages allow a name to be assigned to a variable, but this capability is only available for custom macros on a 30xi Series.
Example: #1
An expression can be used to specify a variable number. In such a case, the expression must be enclosed in brackets.
Example: #[#1+#2–12]

Variables are classified into four into four different types.

Variable number	Type of variable	Function
#0	Always null	This variable is always null. No value can be assigned to this variable. It is not a value, it is nothing/empty/null.
#1 – #33	Local variables	Local variables can only be used within a macro to hold data such as the results of operations. When the power is turned off, local variables are initialized to null. When a macro is called, arguments are assigned to local variables. These should only be used to pass values, not for calculations
#100 – #149 (#199) #500 - #531 (#999)	Common Variables	Common variables can be shared among different macro programs. When the power is turned off, variables #100 to #149 are initialized to null. Variables #500 to #531 hold data even when the power is turned off. As an option, common variables #150 to #199 and #532 to #999 are also available.
#1000 +	System variables	System variables are used to read and write a variety of NC data items such as the current position and tool compensation values.

Note
Common variables #150 - #199 and #532 - #999 are a purchasable option from Fanuc GE (J887)

Range of Variables: Local and common variables can have value 0 or a value in the following ranges:
-10^{47} to -10^{-29}
0
10^{-29} to 10^{47}
If the result of calculation turns out to be invalid, a P/S alarm No. 111 is issued.

No decimal point is required with variables.
Example
When #1=123; is defined, the actual value of variable #1 is 123.000.

When the value of a variable is not defined, such a variable is referred to as a "null" variable. Variable #0 is always a null variable. It cannot be written to, but it can be read. If you look at variables #100 - #149 they are empty, this is written as #0.

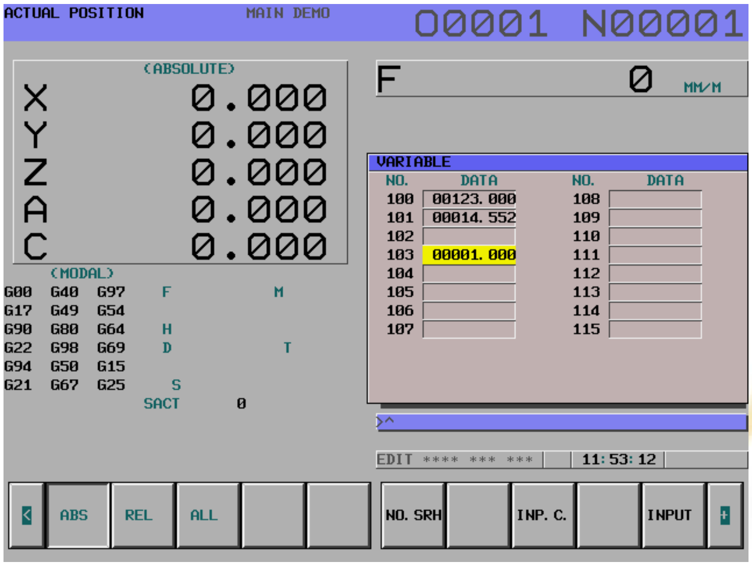
When an undefined variable is quoted, the address itself is also ignored	
When #1 = < vacant >	When #1 = 0
G01 X100 Y #1 ↓ G01 X100	G01 X100 Y #1 ↓ G01 X100 Y0

When < vacant > is the same as 0 except when replaced by < vacant>	
When #1 = < vacant >	When #1 = 0
#2 = #1 ↓ #2 = < vacant >	#2 = #1 ↓ #2 = 0
#2 = #1*5 ↓ #2 = 0	#2 = #1*5 ↓ #2 = 0
#2 = #1+#1 ↓ #2 = 0	#2 = #1 + #1 ↓ #2 = 0

< vacant > differs from 0 only for EQ and NE.	
When #1 = < vacant >	When #1 = 0
#1 EQ #0 ↓ Established	#1 EQ #0 ↓ Not established
#1 NE 0 ↓ Established	#1 NE 0 ↓ Not established
#1 GE #0 ↓ Established	#1 GE #0 ↓ Established

Conditions Expressions	
EQ	EQUAL
NE	NOT EQUAL TOO
LT	LESS THAN
LE	LESS THAN OR EQUAL TOO
GT	GREATER THAN
GE	GREATER THAN OR EQUAL TOO

To display the macro variables press [OFFSET] – {MACRO}



If ******** is displayed then an overflow has occurred. An overflow means the variable is either greater than 99999999 or less than 0.00000001.

System variables can be used to read and write internal NC data such as tool compensation values and current position data. Note, however, that some system variables can only be read. System variables are essential for automation and general–purpose program development.

Interface signals can be exchanged between the programmable machine controller (PMC) and custom macros. In order to use these variables the PMC must be programmed to do this. PMC's should only be written or modified by MTB's. Do not alter your PMC.

Variable number	**Function**
#1000–#1015 #1032	A 16–bit signal can be sent from the PMC to a custom macro. Variables #1000 to #1015 are used to read a signal bit by bit. Variable #1032 is used to read all 16 bits of a signal at one time.
#1100–#1115 #1132	A 16–bit signal can be sent from a custom macro to the PMC. Variables #1100 to #1115 are used to write a signal bit by bit. Variable #1132 is used to write all 16 bits of a signal at one time.
#1133	Variable #1133 is used to write all 32 bits of a signal at one time from a custom macro to the PMC.

For detailed information, refer to the connection manual (B–63523EN–1).

Tool compensation values can be read and written using system variables. Usable variable numbers depend on the number of compensation pairs, whether a distinction is made between geometric compensation and wear compensation, and whether a distinction is made between tool length compensation and cutter compensation. When the number of compensation pairs is not greater than 200, variables #2001 to #2400 can also be used.

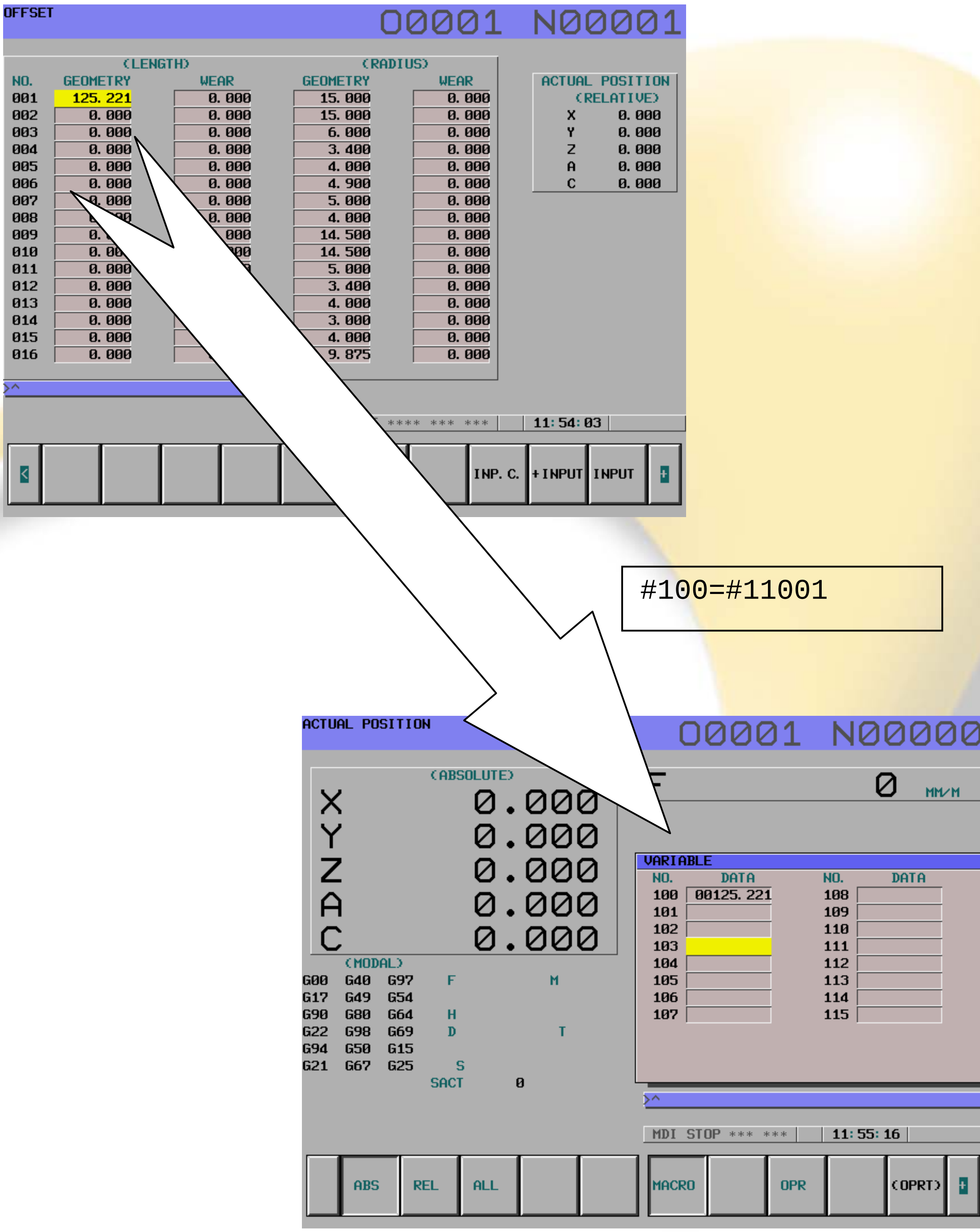
System Variables for Tool Compensation Memory A	
Compensation Number	System Variable
1 : 200 : 999	#10001(#2001) : #10200(#2200) : #10999

System Variables for Tool Compensation Memory B		
Compensation Number	Geometry Compensation	Wear Compensation
1 : 200 : 999	#11001(#2201) : #11200(#2400) : #11999	#10001(#2001) : #10200(#2200) : #10999

System Variables for Tool Compensation Memory C				
Compensation Number	Tool Length Compensation (H)		Cutter Compensation (D)	
	Geometric Compensation	Wear Compensation	Geometric Compensation	Wear Compensation
1 : 200 : 999	#11001(#2201) : #11200(#2400) : #11999	#10001(#2001) : #10200(#2200) : #10999	#13001 : #13200 : #13999	#12001 : #12200 : #12999

If the control being used has memory C (below) and we want to read the length of Tool 1 into common variable 100, we need:

#100=#11001

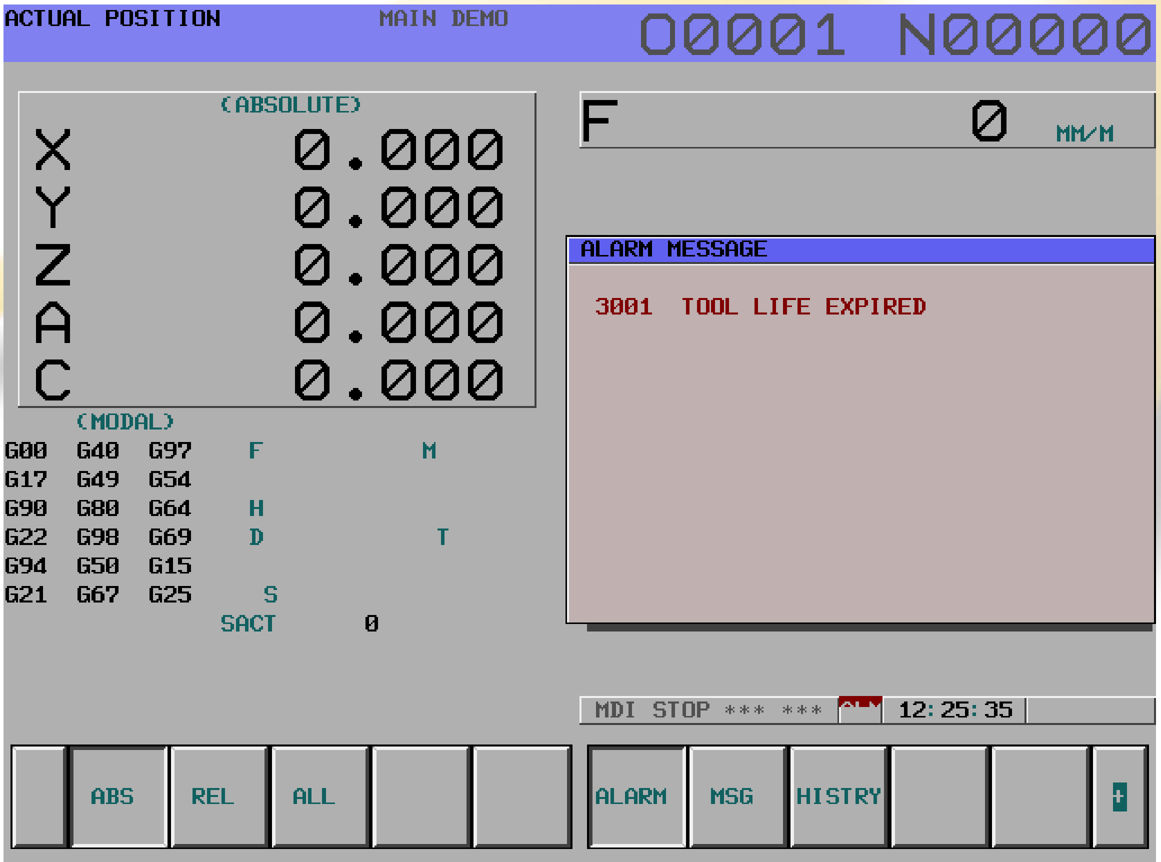


The value of specified in the offset table for the length of tool 1 is now input into variable 100.

Using system variables we can make the machine stop instantly and display a custom message. When a value from 0 to 200 is assigned to variable #3000, the CNC stops with an alarm. After an expression, an alarm message not longer than 26 characters can be described. The CRT screen displays alarm numbers by adding 3000 to the value in variable #3000 along with an alarm message.

Example:

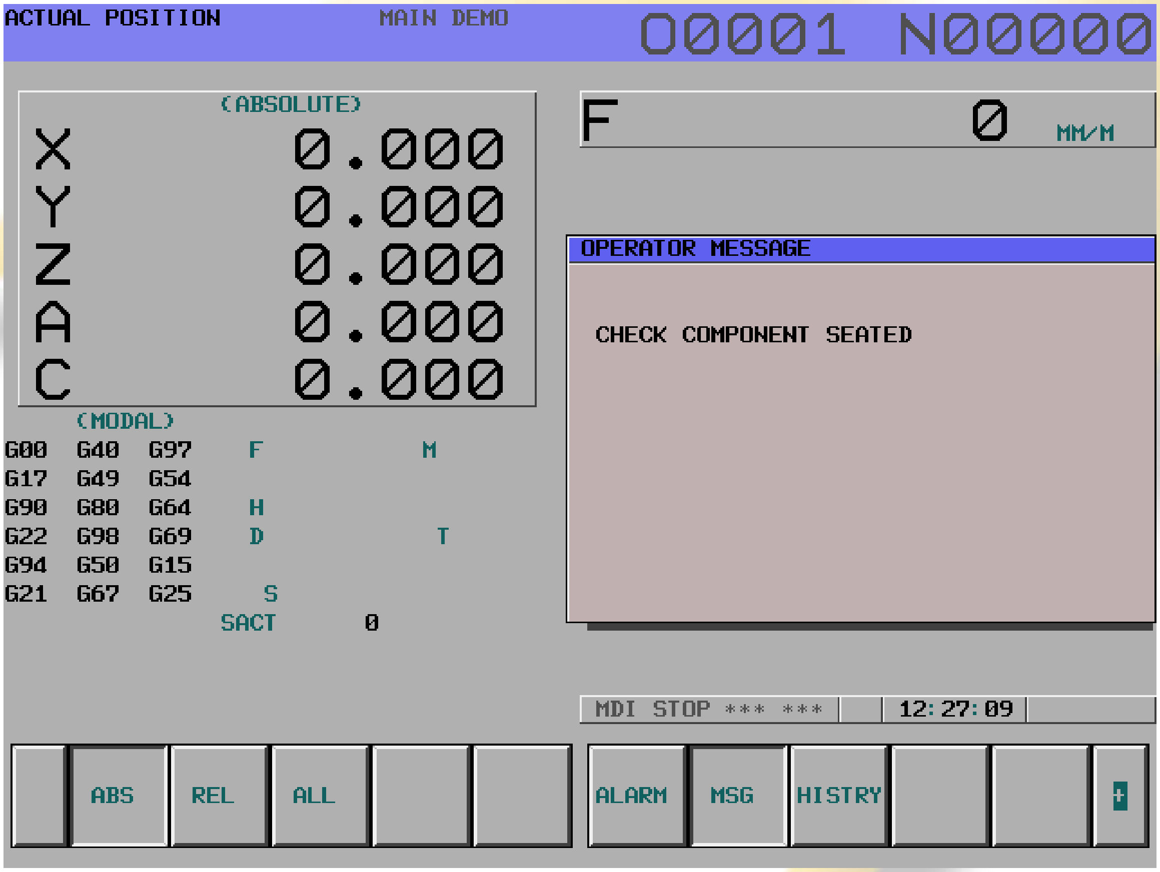
#3000=1(TOOL LIFE EXPIRED)



If you program #3000=23 (TOOL LIFE EXPIRED) then “3023 TOOL LIFE EXPIRED” is dispalyed.

Operator messages are a good way of letting the operator know what is going on in the program and also any checks or inspections they need to make.
When “#3006=1 (MESSAGE);” is commanded in the macro, the program executes blocks up to the immediately previous one and then stops.
When a message of up to 26 characters, which is enclosed by a control–in character (“(”) and control–out character (“)”), is programmed in the same block, the message is displayed on the external operator message screen. The message can be cleared with #3006=0.

#3006=1(CHECK COMPONENT SEATED)



Information regarding time, whether is be the actual time or time to complete something, this can be read using system variables.

System Variables for Time Information	
Variable number	**Function**
#3001	This variable functions as a timer that counts in 1–millisecond increments at all times. When the power is turned on, the value of this variable is reset to 0. When 2147483648 milliseconds is reached, the value of this timer returns to 0.
#3002	This variable functions as a timer that counts in 1–hour increments when the cycle start lamp is on. This timer preserves its value even when the power is turned off. When 9544.371767 hours is reached, the value of this timer returns to 0.
#3011	This variable can be used to read the current date (year/month/day). Year/month/day information is converted to an apparent decimal number. For example, September 28, 2001 is represented as 20010928.
#3012	This variable can be used to read the current time (hours/minutes/seconds). Hours/minutes/seconds information is converted to an apparent decimal number. For example, 34 minutes and 56 seconds after 3 p.m. is represented as 153456.

As #3001 is constantly running, if we want to use it then we must reset it first.

Example:

```
#3001=0;
M98 P1000 (CONTOURING CYCLE);
#500=#3001;
#500=#500/1000;
```

Using these functions it is possible to calculate things such as:

- The percentage of the shift the machine was actually in cycle.
- Cycle time.
- Downtime.

Using system variables we are able to disable and enable program control functions such as:

- SINGLE BLOCK
- FEED RATE OVERRIDE
- FEED HOLD
- EXACT STOP

These groups of variables are called Automatic Operation Control.

System Variable (#3003) for Automatic Operation Control		
#3003	**Single block**	**Completion of an auxiliary function**
0	Enabled	To be awaited
1	Disabled	To be awaited
2	Enabled	Not to be awaited
3	Disabled	Not to be awaited

Example:

#3003=3 – single block is instantly disabled.

#3003=2 – single block is instantly enabled.

When using this variable, there are a few things to be aware of:

- When the power is turned on, the value of this variable is 0.
- When single block stop is disabled, single block stop operation is not performed even if the single block switch is set to ON.
- When a wait for the completion of auxiliary functions (M, S, and T functions) is not specified, program execution proceeds to the next block before completion of auxiliary functions. Also, distribution completion signal DEN is not output.

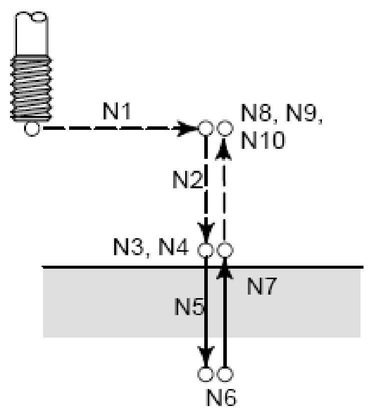
System Variable (#3004) for Automatic Operation Control			
#3004	**Feed hold**	**Feed Rate Override**	**Exact stop**
0	Enabled	Enabled	Enabled
1	Disabled	Enabled	Enabled
2	Enabled	Disabled	Enabled
3	Disabled	Disabled	Enabled
4	Enabled	Enabled	Disabled
5	Disabled	Enabled	Disabled
6	Enabled	Disabled	Disabled
7	Disabled	Disabled	Disabled

Example:

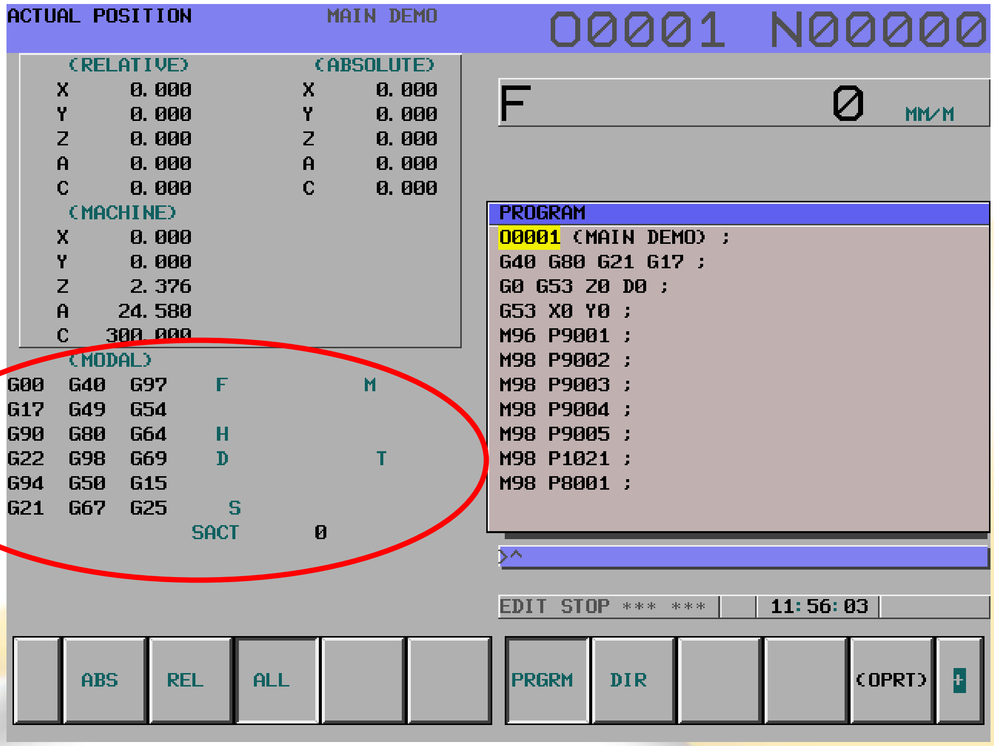
#3004=2 – this will only disable the Feed rate override.

When using this variable, there are a few things to be aware of:

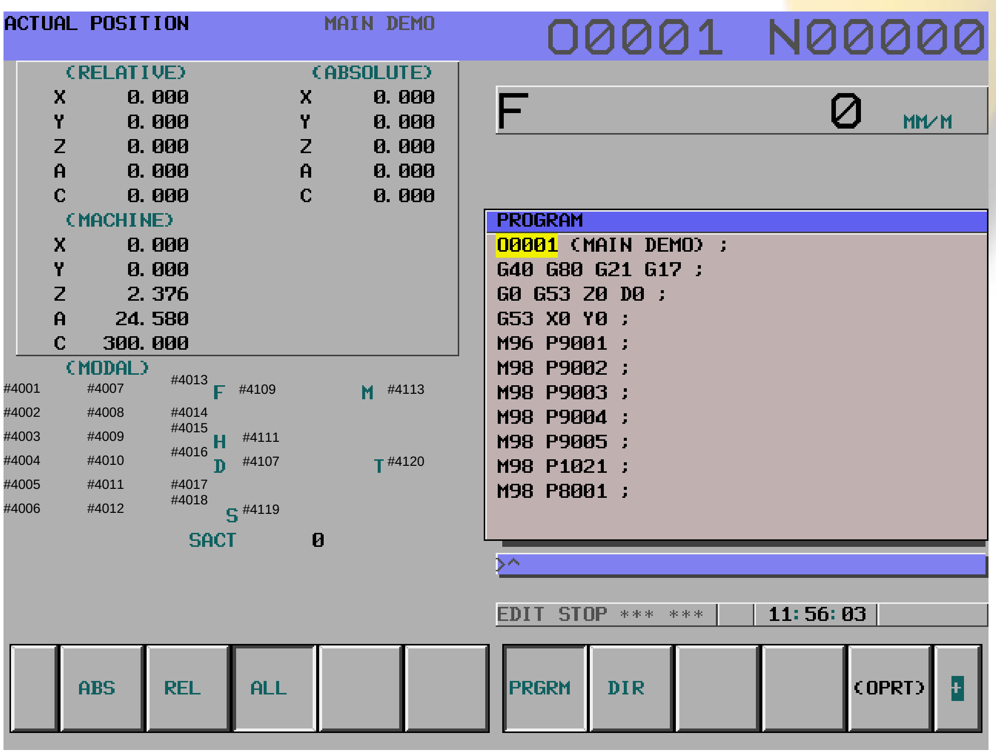
- When the power is turned on, the value of this variable is 0.
- When feed hold is disabled:
  (1) When the feed hold button is held down, the machine stops in the single block stop mode. However, single block stop operation is not performed when the single block mode is disabled with variable #3003.
  (2) When the feed hold button is pressed then released, the feed hold lamp comes on, but the machine does not stop; program execution continues and the machine stops at the first block where feed hold is enabled.
- When feed rate override is disabled, an override of 100% is always applied regardless of the setting of the feed rate override switch on the machine operator's panel.
- When exact stop check is disabled, no exact stop check (position check) is made even in blocks including those which do not perform cutting.




```
O0001 ;
N1 G00 G90 X#24 Y#25
;
N2 Z#18 ;
G04 ;
N3 #3003=3 ;
N4 #3004=7 ;
N5 G01 Z#26 F#9 ;
N6 M04 ;
N7 G01 Z#18 ;
G04 ;
N8 #3004=0 ;
N9 #3003=0 ;
N10M03 ;
```



The image above is a screen shot of a standard Fanuc program display. Below the axis positioning you can see the MODAL information. Modal means active G code or active commands. Everything except the actual spindle speed in the red ring can be read.



System Variables for Modal Information		
Variable Number	**Function**	**Group**
#4001	G00, G01, G02, G03, G33	Group 1
#4002	G17, G18, G19	Group 2
#4003	G90, G91	Group 3
#4004		Group 4
#4005	G94, G95	Group 5
#4006	G20, G21	Group 6
#4007	G40, G41, G42	Group 7
#4008	G43, G44, G49	Group 8
#4009	G73, G74, G76, G80–G89	Group 9
#4010	G98, G99	Group 10
#4011	G98, G99	Group 11
#4012	G65, G66, G67	Group 12
#4013	G96,G97	Group 13
#4014	G54–G59	Group 14
#4015	G61–G64	Group 15
#4016	G68, G69	Group 16
:	:	:
#4022		Group 22
#4102	B code	
#4107	D code	
#4109	F code	
#4111	H code	
#4113	M code	
#4114	Sequence number	
#4115	Program number	
#4119	S code	
#4120	T code	

Example:

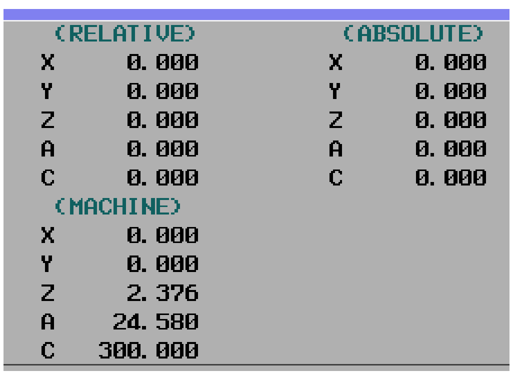
When #1=#4001; is executed, the resulting value in #1 is 0, 1, 2, 3, or 33.
If the specified system variable for reading modal information corresponds to a G code group that cannot be used, a P/S alarm is issued.

Position information can be read but not written.

System Variables for Positioning Information

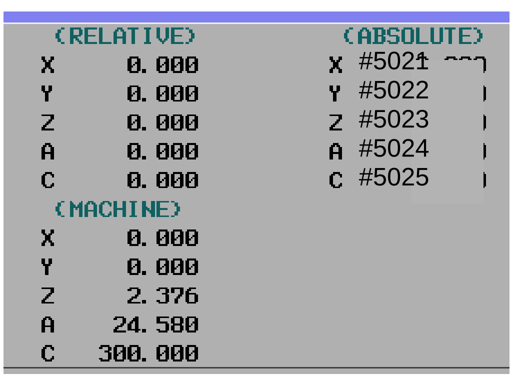
Variable number	Position information	Coordinate system	Tool compensation value	Read operation during movement
#5001–#5008	Block end point	Workpiece coordinate system	Not included	Enabled
#5021–#5028	Current position	Machine coordinate system	Included	Disabled
#5041–#5048	Current position	Workpiece coordinate system		
#5061–#5068	Skip signal position			Enabled
#5081–#5088	Tool length offset value			Disabled
#5101–#5108	Deviated servo position			

The first digit (from 1 to 8) represents an axis number.



Here the axis numbers are as follow:
X=1
Y=2
Z=3
A=4
C=5

Always follow this rule or check parameter 1022.



Here the absolute positions are shown as there variable numbers:
X=#5021
Y=#5022
Z=#5023
A=#5024
C=#5025

Using system variables, zero offset (datum) positions can be read and written too.

Variable number	Function
#5201 : #5208	First–axis external workpiece zero point offset value : Eighth–axis external workpiece zero point offset value
#5221 : #5228	First–axis G54 workpiece zero point offset value : Eighth–axis G54 workpiece zero point offset value
#5241 : #5248	First–axis G55 workpiece zero point offset value : Eighth–axis G55 workpiece zero point offset value
#5261 : #5268	First–axis G56 workpiece zero point offset value : Eighth–axis G56 workpiece zero point offset value
#5281 : #5288	First–axis G57 workpiece zero point offset value : Eighth–axis G57 workpiece zero point offset value
#5301 : #5308	First–axis G58 workpiece zero point offset value : Eighth–axis G58 workpiece zero point offset value
#5321 : #5328	First–axis G59 workpiece zero point offset value : Eighth–axis G59 workpiece zero point offset value

To use variables #2500 to #2806 and #5201 to #5328, optional variables for the workpiece coordinate systems are necessary.
Optional variables for 48 additional workpiece coordinate systems are #7001 to #7948 (G54.1 P1 to G54.1 P48).
Optional variables for 300 additional workpiece coordinate systems are #14001 to #19988 (G54.1 P1 to G54.1 P300).
With these variables, #7001 to #7948 can also be used.

Check the Fanuc operator manual with the machine for additional variables.

The following variables can also be used to read and write zero offset positions.

Axis	Function	Variable number	
First axis	External workpiece zero point offset	#2500	#5201
	G54 workpiece zero point offset	#2501	#5221
	G55 workpiece zero point offset	#2502	#5241
	G56 workpiece zero point offset	#2503	#5261
	G57 workpiece zero point offset	#2504	#5281
	G58 workpiece zero point offset	#2505	#5301
	G59 workpiece zero point offset	#2506	#5321
Second axis	External workpiece zero point offset	#2600	#5202
	G54 workpiece zero point offset	#2601	#5222
	G55 workpiece zero point offset	#2602	#5242
	G56 workpiece zero point offset	#2603	#5262
	G57 workpiece zero point offset	#2604	#5282
	G58 workpiece zero point offset	#2605	#5302
	G59 workpiece zero point offset	#2606	#5322
Third axis	External workpiece zero point offset	#2700	#5203
	G54 workpiece zero point offset	#2701	#5223
	G55 workpiece zero point offset	#2702	#5243
	G56 workpiece zero point offset	#2703	#5263
	G57 workpiece zero point offset	#2704	#5283
	G58 workpiece zero point offset	#2705	#5303
	G59 workpiece zero point offset	#2706	#5323
Fourth axis	External workpiece zero point offset	#2800	#5204
	G54 workpiece zero point offset	#2801	#5224
	G55 workpiece zero point offset	#2802	#5244
	G56 workpiece zero point offset	#2803	#5264
	G57 workpiece zero point offset	#2804	#5284
	G58 workpiece zero point offset	#2805	#5304
	G59 workpiece zero point offset	#2806	#5324

The operations listed in the table below can be performed on variables. The expression to the right of the operator can contain constants and/or variables combined by a function or operator. Variables #j and #K in an expression can be replaced with a constant. Variables on the left can also be replaced with an expression.

Function	Format	Remarks
Definition	#i=#j	
Sum	#i=#j+#k;	
Difference	#i=#j–#k;	
Multiply	#i=#j*#k;	
Divide	#i=#j/#k;	
Sine	#i=SIN[#j];	An angle is specified in degrees. 90 degrees and 30 minutes is represented as 90.5 degrees.
Arcsine	#i=ASIN[#j];	
Cosine	#i=COS[#j];	
Arccosine	#i=ACOS[#j];	
Tangent	#i=TAN[#j];	
Arctangent	#i=ATAN[#j]/[#k];	
Square root	#i=SQRT[#j];	
Absolute value	#i=ABS[#j];	
Rounding off	#i=ROUND[#j];	
Rounding down	#i=FIX[#j];	
Rounding up	#i=FUP[#j];	
Natural logarithm	#i=LN[#j];	
Exponential function	#i=EXP[#j];	
OR	#i=#j OR #k;	A logical operation is performed on binary numbers bit by bit.
XOR	#i=#j XOR #k;	
AND	#i=#j AND #k;	
Conversion from BCD to BIN	#i=BIN[#j];	Used for signal exchange to and from the PMC
Conversion from BIN to BCD	#i=BCD[#j];	

Definition - #i=#j
This is what's used to transfer data from one variable to another. The left variable is where the result is.
So if #1=10 and #2=12
#1=#2
Both variables now equal 12.

Sum - #i=#j+#k
This is what's used to add variables, or values on their own together.
So if #2=12
#1=#2+10
The value of #1 is now 22.

Difference - #i=#j-#k
This is what's used to subtract variables, or values on their own together.
So if #2=12
#1=#2-10
The value of #1 is now 2.

Multiply - #i=#j*#k
This is what's used to multiply variables, or values on their own together.
So if #2=12
#1=#2*10
The value of #1 is now 120.

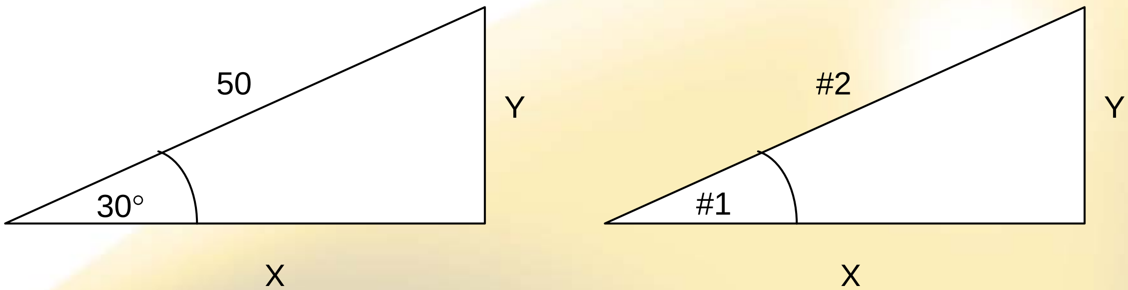
Divide - #i=#j/#k
This is what's used to divide variables, or values on their own together.
So if #2=20
#1=#2/10
The value of #1 is now 2.

All of the above can be put together using brackets to perform larger calculations.
So if #1=2 and #2=5
#100=#1*[#2-3]
The value of #100 is now 4, because 2 x (5 – 3) = 4

For more information on the priority of operations when using brackets see page 23. Macro B also conforms to the Precedence Rule.

In Macro B, Sine, Cosine and Tangent follow the same pattern.

Sine	#i=SIN[#j];
Tangent	#i=TAN[#j];
Cosine	#i=COS[#j];



In the example above, #1=30 and #2=50

In mathematics the equation to calculate the length of:

X is (cos30) x 50 = 43.301
Y is (sin30) x 50 = 25

In Macro B it's the same

X is #100=[cos[#1]*#2]
Y is #101=[sin[#1]*#2]

To actually move the axis incrementally the result of this calculation we can write the following:

G1 G91 X[cos[#1]*#2] Y[sin[#1]*#2]

Or

#100=[cos[#1]*#2]
#101=[sin[#1]*#2]
G1 G91 X#100 Y#101

It is a good idea to use a Zeus book if you're unsure of the formulae.

Arcsine, Arccosine and Arctangent are inverse trigonometric functions of Sine, Cosine and Tangent.

There are sme parameters related to Arcsine, Arccosine and Arctangent, for further details see the manual B–63534EN

Round Function - #i=ROUND[#j];
When the ROUND function is included in an arithmetic or logic operation command, IF statement, or WHILE statement, the ROUND function rounds off at the first decimal place.
When #1=ROUND[#2]; is executed where #2 holds 1.2345, the value of variable #1 is 1.0.

Rounding Up and Down - #i=FUP[#j] & #i=FIX[#j]
With CNC, when the absolute value of the integer produced by an operation on a number is greater than the absolute value of the original number, such an operation is referred to as rounding up to an integer.
Conversely, when the absolute value of the integer produced by an operation on a number is less than the absolute value of the original number, such an operation is referred to as rounding down to an integer.
Be particularly careful when handling negative numbers.

Suppose that #1=1.2 and #2=–1.2.

When #3=FUP[#1] is executed, 2.0 is assigned to #3.
When #3=FIX[#1] is executed, 1.0 is assigned to #3.
When #3=FUP[#2] is executed, –2.0 is assigned to #3.
When #3=FIX[#2] is executed, –1.0 is assigned to #3.

When programming larger calculations, it is important to make sure your calculations are in the correct order, this is called the Priority of Operations.

The priority of operation for Macro B statements is as follows:

1. Functions
2. Operations such as multiplication and division (*,/,AND)
3. Operations such as addition and subtraction (+,-,OR,XOR)

Example

$$\#1=\underbrace{\#2+\underbrace{\#3*\underbrace{\sin[\#4]}_{1}}_{2}}_{3}$$

1,2 and 3 indicate the order of operations.

Brackets are used to change the order of operations. Brackets can be used to a depth of **five** levels including the brackets used to enclose a function.
When a depth of five levels is exceeded, P/S alarm No. 118 occurs.

$$\#1=\underbrace{\sin[\underbrace{\underbrace{\underbrace{[\underbrace{\#2+\#3}_{1}]*\#4}_{2}+\#5}_{3}]*\#6}_{4}]}_{5}$$

1,2,3,4 and 5 indicate the order of operations.

Brackets ([,]) are used to enclose an expression. Note that parentheses (,)are used for comments.
Errors may occur when operations are performed.

Operation	Average error	Maximum error	Type of error
a = b*c	1.55×10^{-10}	4.66×10^{-10}	Relative error(*1) $\left\lvert \frac{\varepsilon}{a} \right\rvert$
a = b / c	4.66×10^{-10}	1.88×10^{-9}	
$a = \sqrt{b}$	1.24×10^{-9}	3.73×10^{-9}	
a = b + c a = b – c	2.33×10^{-10}	5.32×10^{-10}	(*2) Min $\left\lvert \frac{\varepsilon}{b} \right\rvert, \left\lvert \frac{\varepsilon}{c} \right\rvert$
a = SIN [b] a = COS [b]	5.0×10^{-9}	1.0×10^{-8}	Absolute error(*3) $\lvert \varepsilon \rvert$ degrees
a = ATAN [b] / [c] (*4)	1.8×10^{-6}	3.6×10^{-6}	

1 The relative error depends on the result of the operation.
2 Smaller of the two types of errors is used.
3 The absolute error is constant, regardless of the result of the operation.
4 Function TAN performs SIN/COS.
5 If the result of the operation by the SIN, COS, or TAN function is less than 1.0 x 10–8 or is not 0 because of the precision of the operation, the result of the operation can be normalized to 0 by setting bit 1 (MFZ) of parameter No. 6004 to 1.

The precision of variable values is about 8 decimal digits. When very large numbers are handled in an addition or subtraction, the expected results may not be obtained.
Example:
When an attempt is made to assign the following values to variables
#1 and #2:
#1=9876543210123.456
#2=9876543277777.777
the values of the variables become:
#1=9876543200000.000
#2=9876543300000.000
In this case, when #3=#2–#1; is calculated, #3=100000.000 results.
(The actual result of this calculation is slightly different because it is performed in binary.)

When a divisor of zero is specified in a division or TAN[90], P/S alarm No. 112 occurs.

The following blocks are referred to as macro statements:

- Blocks containing an arithmetic or logic operation (=)
- Blocks containing a control statement (such as GOTO, DO, END)
- Blocks containing a macro call command (such as macro calls by G65, G66, G67, or other G codes, or by M codes)

Any block other than a macro statement is referred to as an NC statement.

Differences from NC Statements

Even when single block mode is on, the machine does not stop. Note, however, that the machine stops in the single block mode when bit 5 of parameter SBM No. 6000 is 1.

Macro blocks are not regarded as blocks that involve no movement in the cutter compensation mode (seeII–15.7).

NC statements that have the same property as macro statements

NC statements that include a subprogram call command (such as subprogram calls by M98 or other M codes, or by T codes) and not include other command addresses except an O,N or L address have the same property as macro statements.

The blocks not include other command addresses except an O,N,P or L address have the same property as macro statements.

In a program, the flow of control can be changed using the GOTO statement and IF statement. Three types of branch and repetition operations are used:

Branch and Repetition
- GOTO statement (unconditional branch)
- IF statement (conditional: IF…,THEN…)
- WHILE statement (repetition)

Unconditional Branch (GOTO Statement)

IF[<conditionalexpression>]GOTOn

Unconditional Branch (GOTO Statement)	Specify a conditional expression after IF.
IF[<conditional expression>]GOTOn	If the specified conditional expression is satisfied, a branch to sequence number n occurs. If the specified condition is not satisfied, the next block is executed.

If the value of variable #100 is not equal to 20, a branch to sequence number N5 occurs.
If the condition is not satisfied — IF[#100 NE 20] GOTO 5 → Processing → N5 G0 G54 X50. — If the condition is satisfied

IF[<conditional expression>]THEN	If the specified conditional expression is satisfied, a predetermined macro statement is executed. Only a single macro statement is executed.

If #1 is empty (no value in it), then the following statement is satisfied.
IF[#1EQ#0] THEN #3000=1(TOOL NOT ENGAGED);

A conditional expression must include an operator inserted between two variables or between a variable and constant, and must be enclosed in brackets ([,]). An expression can be used instead of a variable.

Operators each consist of two letters and are used to compare two values to determine whether they are equal or one value is smaller or greater than the other value. Note that the inequality sign cannot be used.

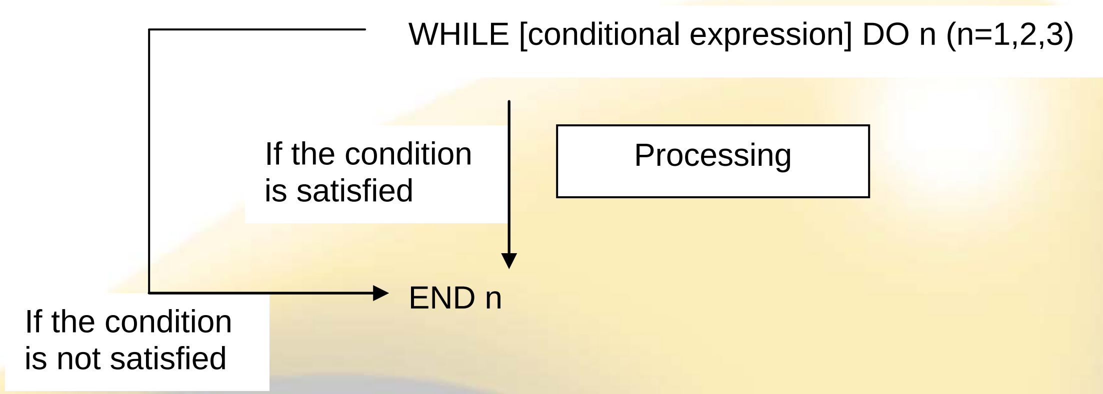
Operator	Meaning
EQ	Equal to(=)
NE	Not equal to(&#23;)
GT	Greater than(>)
GE	Greater than or equal to(&#24;)
LT	Less than(<)
LE	Less than or equal to(&#11;)

The sample program below finds the total of numbers 1 to 10.

```
O9500;
#1=0; . . . . . . . . . . . . . . . . . . Initial value of the variable to hold the sum
#2=1; . . . . . . . . . . . . . . . . . . Initial value of the variable as an addend
N1 IF[#2 GT 10] GOTO 2; . . Branch to N2 when the addend is greater than 10
#1=#1+#2; . . . . . . . . . . . . . . Calculation to find the sum
#2=#2+1; . . . . . . . . . . . . . . . Next addend
GOTO 1; . . . . . . . . . . . . . . .  Branch to N1
N2 M30; . . . . . . . . . . . . . . . . End of program
```

Repetition (WHILE statement)

Specify a conditional expression after WHILE. While the specified condition is satisfied, the program from DO to END is executed. If the specified condition is not satisfied, program execution proceeds to the block after END.



While the specified condition is satisfied, the program from DO to END after WHILE is executed. If the specified condition is not satisfied, program execution proceeds to the block after END. The same format as for the IF statement applies. A number after DO and a number after END are identification numbers for specifying the range of execution. The numbers 1, 2, and 3 can be used. When a number other than 1, 2, and 3 is used, P/S alarm No. 126 occurs.

The sample program below finds the total of numbers 1 to 10.

```
O0001;
#1=0;
#2=1;
WHILE[#2 LE 10]DO 1;
#1=#1+#2;
#2=#2+1;
END 1;
M30;
```

The identification numbers (1 to 3) in a DO–END loop can be used as many times as desired. Note, however, when a program includes crossing repetition loops (overlapped DO ranges), P/S alarm No. 124 occurs.

The identification numbers (1 to 3) can be used as many times as required.	DO loops can be nested to a maximum depth of three levels.
WHILE […] DO 1; Processing END 1; : WHILE […] DO 1; Processing END 1;	WHILE […] DO 1; : WHILE […] DO 2; : WHILE […] DO 3; Processing END 3; : END 2; : END 2;
DO ranges cannot over lap.	Control can be transferred to the outside of a loop.
WHILE […] DO 1; Processing WHILE […] DO 2; END 1; Processing END 1;	WHILE […] DO 1; Processing IF […] GOTO n; END 1; Processing Nn;

Macro Call

A macro program can be called using the following methods:

Macro Call
- Simple call (G65)
- Modal call (G66,G67)
- Macro call with G code
- Macro call with M code
- Subprogram call with M code
- Subprogram call with T code

Both G65 and M98 will call up and open a subprogram.

The main difference between a Macro Call (G65) and a subprogram call (M98) is that G65 can pass information from the G65 line into a subprogram as variables.

When an M98 block contains another NC command (for example, G01 X100.0 M98Pp), the subprogram is called after the command is executed. On the other hand, G65 unconditionally calls a macro.

Think of a normal canned cycle as a macro call(G81 – Drilling). The information you specify (example X and Y coordinates, depth of hole, return point, etc) is then passed into a macro program, the data is manipulated, that then drills your holes. This is what happens on CNC controls, but as Fanuc or the MTB have written the cycles, they have also hidden all the “behind the scenes” activities. It is also possible in to do this, once the Macro is complete.

G65 P9010 X10 Y15 Z-10 R2

X10 → #24   Y15 → #25   Z-10 → #26   R2 → #18

#24=10
#25=15
#26#=-10
#18=2

Simple Call (G65)

When G65 is specified, the custom macro specified at address P is called. Data (argument) can be passed to the custom macro program.

```
G65 Pp Ln
        P: Number of the program to call
        L: Repetition count
```

```
O0001;
    :
G65 P9010 L2 A1 B2;
    :
M30;
```

```
O9010;
#3=#1+#2;
IF[#3GT360]GOTO99;
G0 G54 X10;
M99;
```

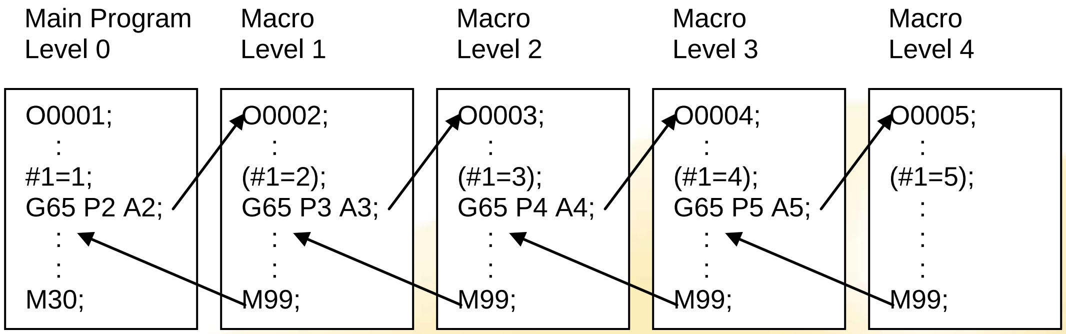
After a G65, a P (program number) must be specified, this program is the macro program needed. When repetitions are required, a L must be specified.
Any other information on a G65 line is passed into the macro program as variables. This is what we call an argument. The information passed is the argument.

Two types of argument specification are available. Argument specification 1 uses letters other than G, L, O, N, and P once each.
Argument specification 2 uses A, B, and C once each and also uses I, J, and K up to ten times. The type of argument specification is determined automatically according to the letters used. See the manual B-63534 for further details.

Address	Variable Number	Address	Variable Number	Address	Variable 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

- Addresses G, L, N, O, and P cannot be used in arguments.
- Addresses that need not be specified can be omitted. Local variables corresponding to an omitted address are set to null.
- Addresses do not need to be specified alphabetically. They conform to word address format.
  I, J, and K need to be specified alphabetically, however.

Calls can be nested to a depth of four levels including simple calls (G65) and modal calls (G66). This does not include subprogram calls (M98).



- Local variables from level 0 to 4 are provided for nesting.
- The level of the main program is 0.
- Each time a macro is called (with G65 or G66), the local variable level is incremented by one. The values of the local variables at the previous level are saved in the CNC.
- When M99 is executed in a macro program, control returns to the calling program. At that time, the local variable level is decremented by one; the values of the local variables saved when the macro was called are restored.

Modal Call (G66)

Once G66 is issued to specify a modal call a macro is called after a block specifying movement along axes is executed. This continues until G67 is issued to cancel a modal call.

G66 Pp Ln
P: Number of the program to call
L: Repetition count

```
O0001;
     :
G66 P9010 L2 A1 B2;
G00 X100.;
Y300.
M30;
```

```
O9010;
G00 Z-#1
G01 Z-#2

M99;
```

- After G66, specify at address P a program number subject to a modal call.
- When a number of repetitions is required, a number from 1 to 9999 can be specified at address L.
- As with a simple call (G65), data passed to a macro program is specified in arguments.
  When a G67 code is specified, modal macro calls are no longer performed in subsequent blocks.
- Calls can be nested to a depth of four levels including simple calls (G65) and modal calls (G66). This does not include subprogram calls (M98).
- Modal calls can be nested by specifying another G66 code during a modal call.

Macro Call Using G Code

By setting a G code number used to call a macro program in a parameter, the macro program can be called in the same way as for a simple call (G65).
By setting parameter 6050 to 100, G65 Pn is now replaced by G100

G65 Pp = G100

```
O0001;
    :
G100 L2 A1 B2;
    :
M30;
```

```
O9010;
#3=#1+#2;
IF[#3GT360]GOTO99;
G0 G54 X10;
M99;
```

By setting a G code number from 1 to 9999 used to call a custom macro program (O9010 to O9019) in the corresponding parameter (N0.6050 to No.6059), the macro program can be called in the same way as with G65.
For example, when a parameter is set so that macro program O9010 can be called with G81, a user–specific cycle created using a custom macro can be called without modifying the machining program.

The following table shows the correspondence between program number and parameter. If for example your macro program is O9010, enter the value of the G code you want in parameter 6050. I.E if you want G125 to open O9010 then 6050 must be 125.

Program Number	Parameter Number
O9010	6050
O9011	6051
O9012	6052
O9013	6053
O9014	6054
O9015	6055
O9016	6056
O9017	6057
O9018	6058
O9019	6059

Macro Call Using M Code

By setting an M code number used to call a macro program in a parameter, the macro program can be called in the same way as for a simple call (G65). By setting parameter 6080 to 100, G65 Pn is now replaced by M100

G65 Pp = M100

```
O0001;
   :
M100 L2 A1 B2;
   :
M30;
```

```
O9020;
#3=#1+#2;
IF[#3GT360]GOTO99;
G0 G54 X10;
M99;
```

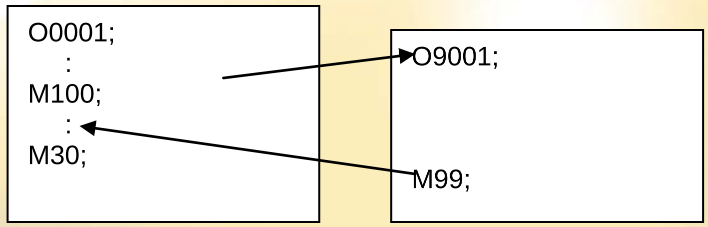
By setting an M code number from 1 to 99999999 used to call a custom macro program (9020 to 9029) in the corresponding parameter (No.6080 to No.6089), the macro program can be called in the same way as with G65.

Program Number	Parameter Number
O9020	6080
O9021	6081
O9022	6082
O9023	6083
O9024	6084
O9025	6085
O9026	6086
O9027	6087
O9028	6088
O9029	6089

Subprogram Call Using M Code

By setting an M code number used to call a subprogram (macro program) in a parameter, the macro program can be called in the same way as with a subprogram call (M98).
By setting parameter 6071 to 100, M98 Pn is now replaced by M100

M98 Pp = M100



By setting an M code number from 1 to 99999999 used to call a subprogram in a parameter (No.6071 to No. 6079), the corresponding custom macro program (O9001 to O9009) can be called in the same way as with M98.

Program Number	Parameter Number
O9001	6071
O9002	6072
O9003	6073
O9004	6074
O9005	6075
O9006	6076
O9007	6077
O9008	6078
O9009	6079

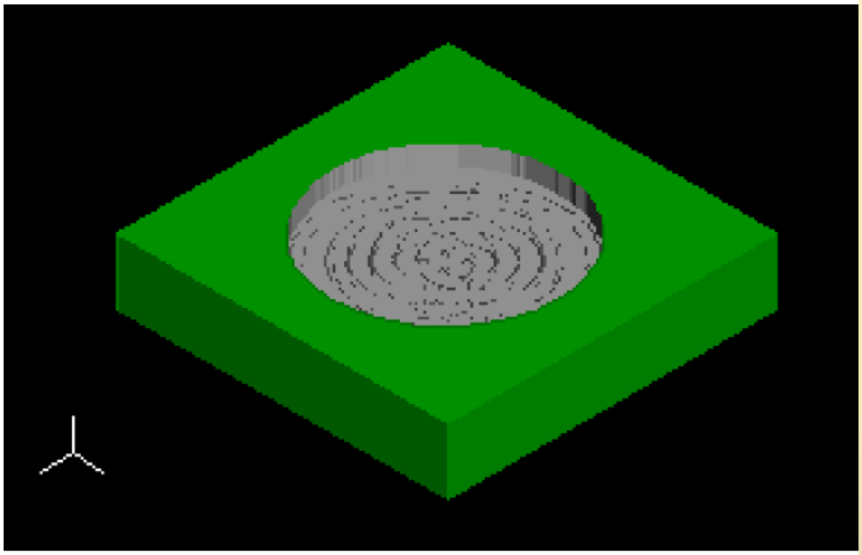
Joint Exercise

Scenario

You have a customer that wants you to machine circular holes into a square billet. Problem is there are over 50 variations of this job. All different hole sizes, depths and centre points.

Process

1. Move the tool to centre point
2. Move the tool down into the job
3. Interpolate out several times until diameter is met
4. Return tool to the centre point
5. Repeat steps 2 and 3 until depth and diameter is met.

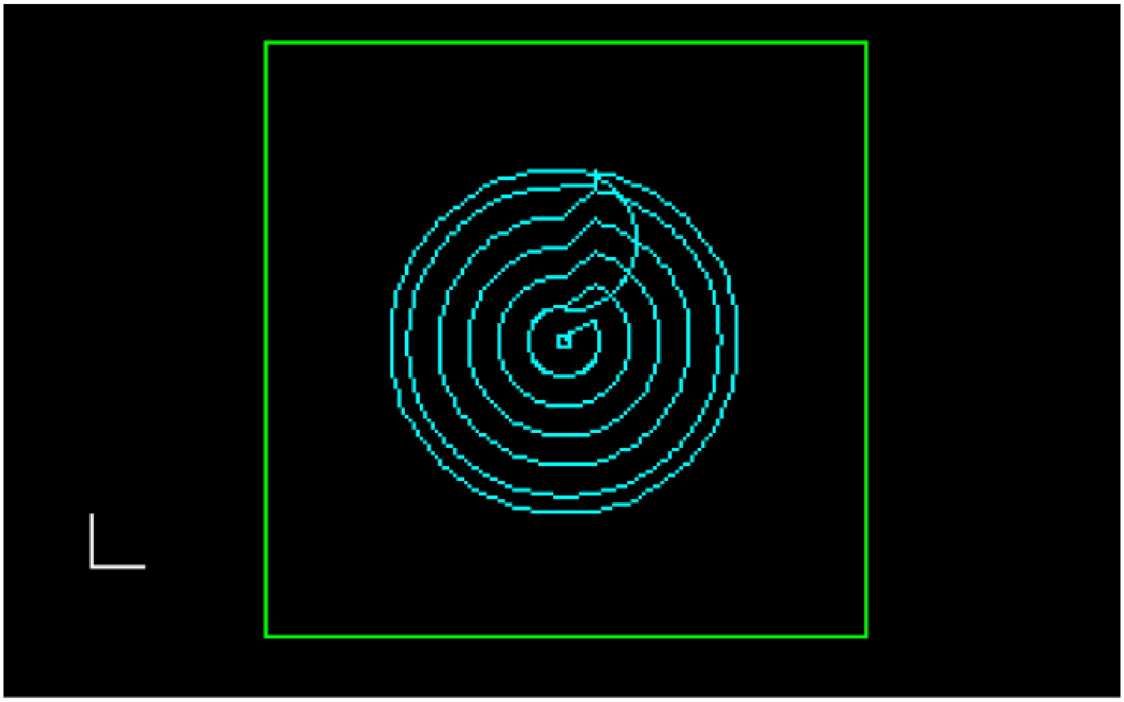


Now we have to think about every possibilty and options available to us, to come up with the best method. Here are a few things to think about:

- Where is the datum point going to be?
- Absolute or Incremental?
- Climb milling/direction?
- What letters to use on the Macro call?
- What information shall we require?
- Cutter compensation, yes/no?
- What error checks can we make?
- What G code to create?
- What material is the component?
- What variables shall we use, #100-#149 or #500-#531?

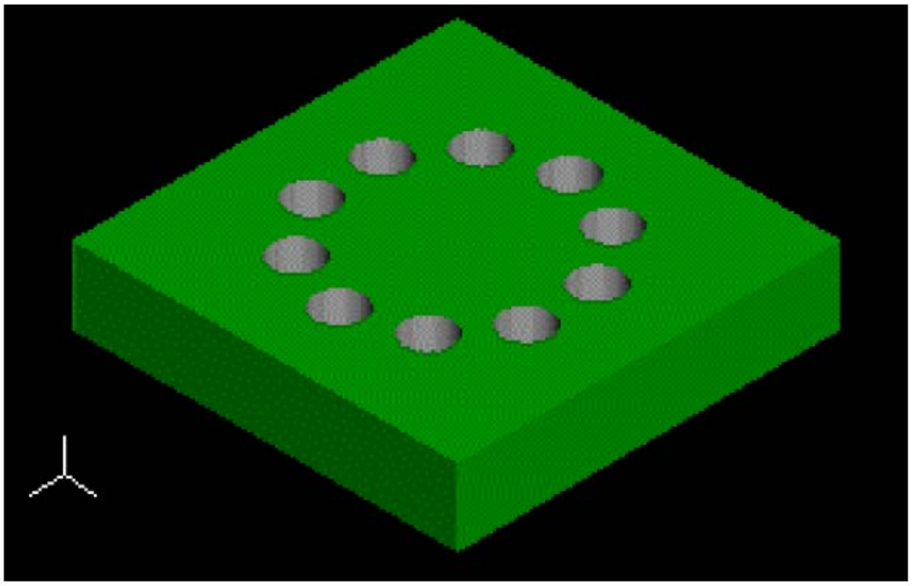
It' always a good idea to have a pen and paper to hand to make notes on all of the above when you're writing Macro B programs.

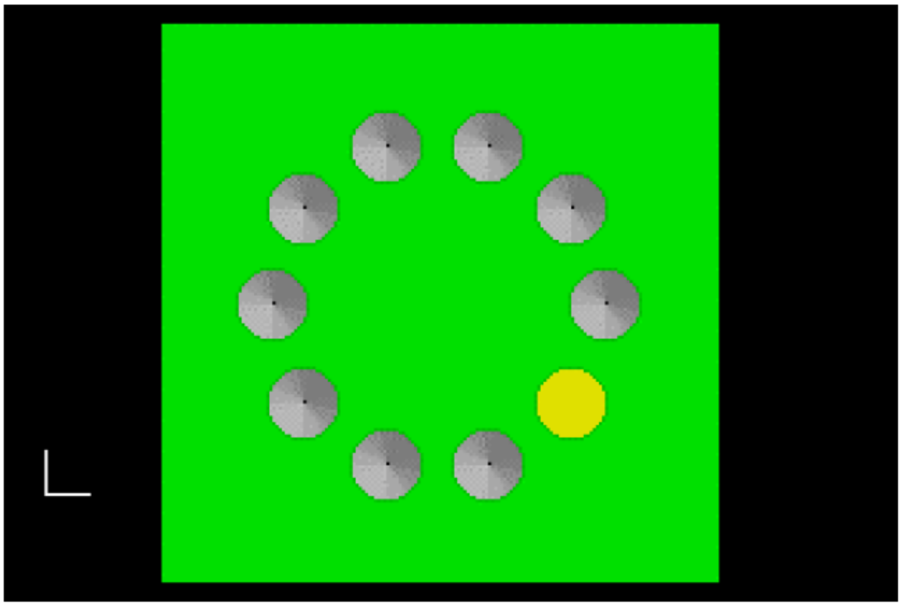
Using the joint the joint exercise just completed, we need to make the macro machine to the correct sizes specified. Ensuring the macro doesn't cut oversize, radially or in depth. We also need to put in place measures to prevent the macro running without all the necessary information. For example if the user forgets to input the diameter of te circle, then the macro cannot run. This macro should run with G100.



Scenario
You have a customer that wants you to create a G-Code to enable him to drill various PCD’s. These comes with various depths, diameters and the amount of holes vary.

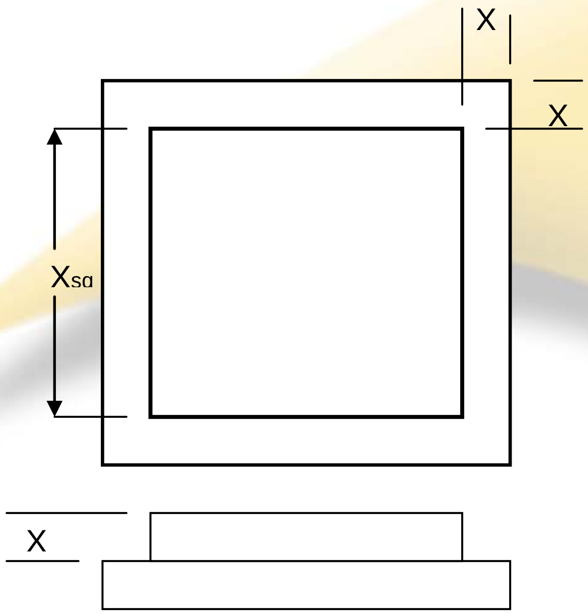
Process
1. Move the tool to the centre point
2. Using Trigonometry calculate hole position 1
3. Drill the hole
4. Using a WHILE statement repeat steps 2 & 3 until all holes are drilled.

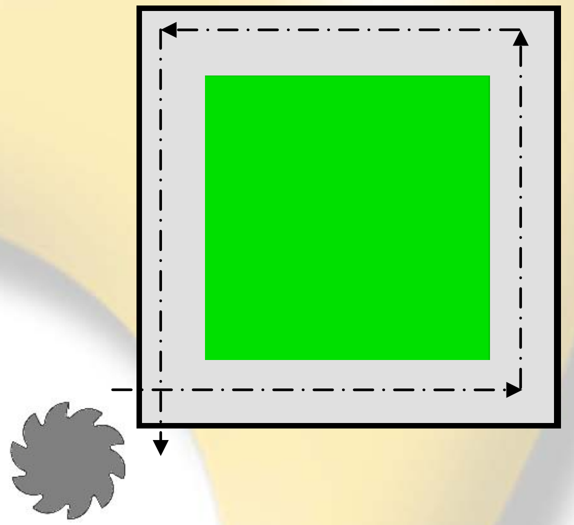




Scenario

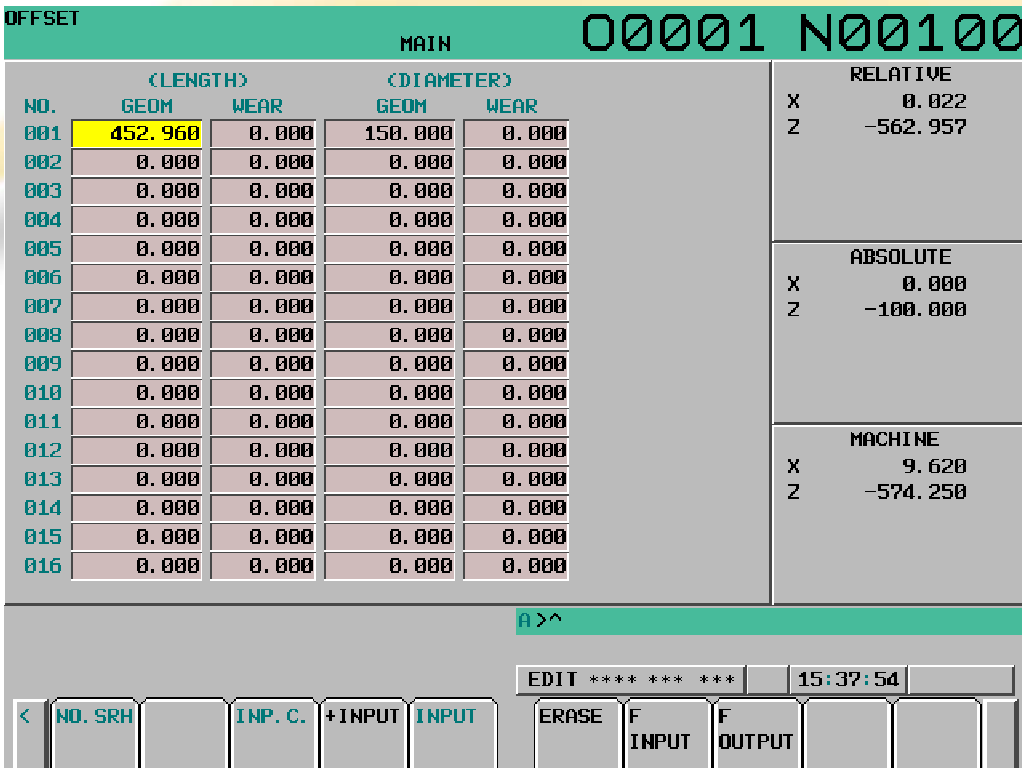
We have just received an order for several thousand components. Each component has a raised square face on it. There are ten different types of component, where features such as the height or square size of the component differ. Rather than write ten different NC programs, we can write one Macro program instead.





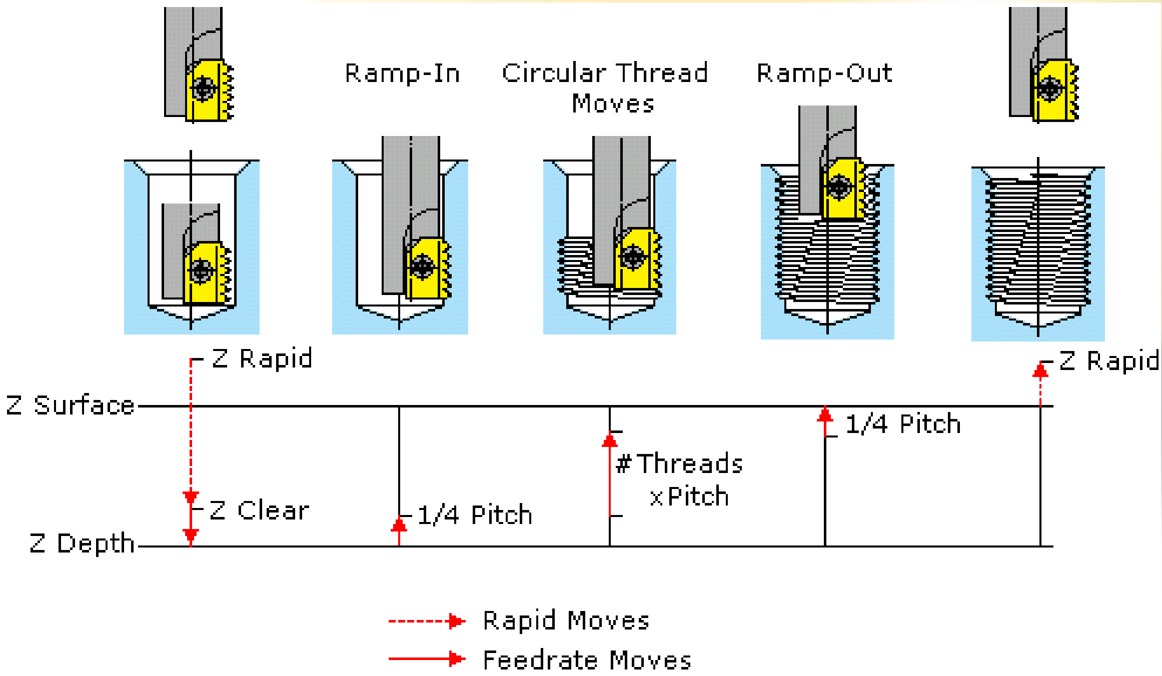
Scenario

You have just written several macro programs on a cylindrical grinder. All of these programs use the offsets of Tool 1, as there is only one wheel and the datum's positions on G54. If the operator sets any other offsets then your macro has a problem. The control has 300 tool offsets and 6 work piece offsets. Again if the operator sets any offset other than G54, your macro has a problem. So we have to create a check program to make sure no unnecessary information is set, for tool length, tool radius and work pieces. Also if the external offset is, display a message so the operator is aware the EXT offset is active.



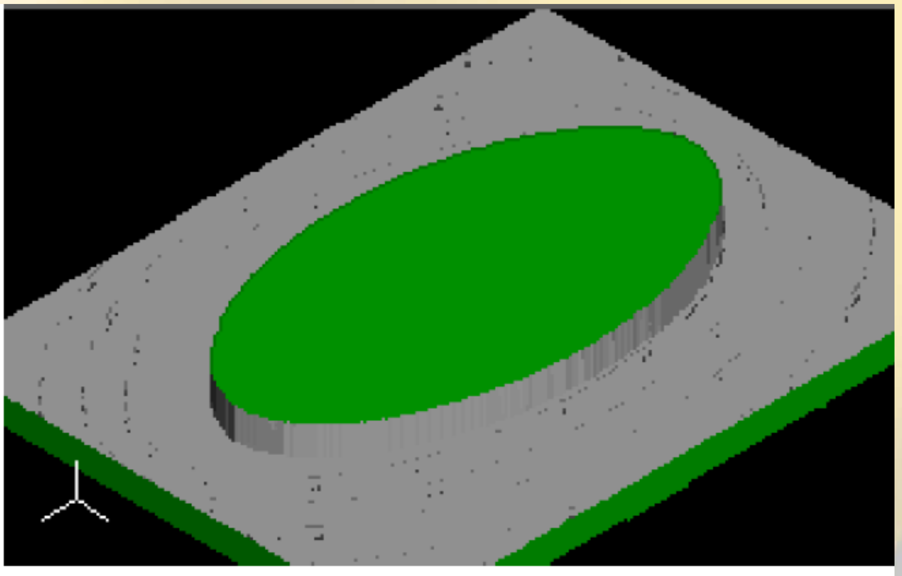
Scenario

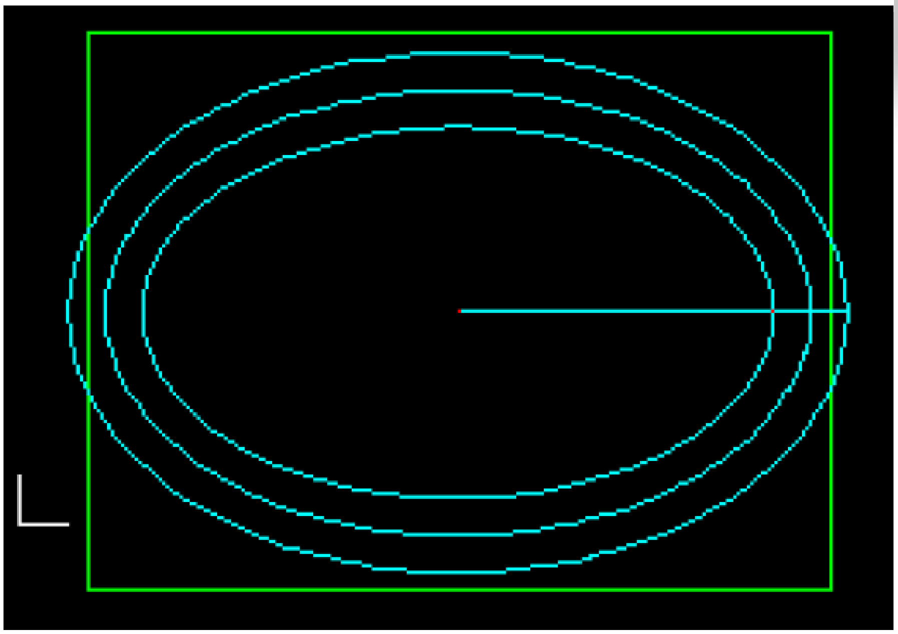
Thread milling at your place of work is a common operation. Currently for every cycle a new helical interpolation program is written, consuming a lot of time. Your task is to create a cycle for thread milling, using G184 to call up the macro; the G180 line should look similar to a G84 line. Once the tool enters the component, it must not be stopped, Be sure to rad on and rad off.



Scenario
You have a customer that wants you to machine elliptical bosses into a square billet. Problem is there are over 20 variations of this job. All different major and minor diameters and some are not complete ellipses, i.e start at 90 degrees and finish at 180 degrees.

Process
1. Move the tool to centre point
2. Move the tool down into the job
3. Interpolate (varying radiuses throughout) out several times until diameter is met
4. Return tool to the centre point
5. Repeat steps 2 and 3 until depth and diameter is met.





Variable	Description
#1	A
#2	B
#3	C
#4	I
#5	J
#6	K
#7	D
#8	E
#9	F
#10	
#11	H
#12	
#13	M
#14	
#15	
#16	
#17	Q
#18	R
#19	S
#20	T
#21	U
#22	V
#23	W
#24	X
#25	Y
#26	Z

Variable	Description
#100	Common Variable
#101	Common Variable
#102	Common Variable
#103	Common Variable
#104	Common Variable
#105	Common Variable
#106	Common Variable
#107	Common Variable
#108	Common Variable
#109	Common Variable
#110	Common Variable
#111	Common Variable
#112	Common Variable
#113	Common Variable
#114	Common Variable
#115	Common Variable
#116	Common Variable
#117	Common Variable
#118	Common Variable
#119	Common Variable
#120	Common Variable
#121	Common Variable
#122	Common Variable
#123	Common Variable
#124	Common Variable
#125	Common Variable
#126	Common Variable
#127	Common Variable
#128	Common Variable
#129	Common Variable
#130	Common Variable
#131	Common Variable
#132	Common Variable
#133	Common Variable
#134	Common Variable
#135	Common Variable
#136	Common Variable
#137	Common Variable
#138	Common Variable
#139	Common Variable
#140	Common Variable
#141	Common Variable
#142	Common Variable
#143	Common Variable
#144	Common Variable
#145	Common Variable
#146	Common Variable
#147	Common Variable
#148	Common Variable
#149	Common Variable

All of these are variables are cleared either on reset, at the end of the program or at power off.

Variable	Description
#500	Common Variable
#501	Common Variable
#502	Common Variable
#503	Common Variable
#504	Common Variable
#505	Common Variable
#506	Common Variable
#507	Common Variable
#508	Common Variable
#509	Common Variable
#510	Common Variable
#511	Common Variable
#512	Common Variable
#513	Common Variable
#514	Common Variable
#515	Common Variable
#516	Common Variable
#517	Common Variable
#518	Common Variable
#519	Common Variable
#520	Common Variable
#521	Common Variable
#522	Common Variable
#523	Common Variable
#524	Common Variable
#525	Common Variable
#526	Common Variable
#527	Common Variable
#528	Common Variable
#529	Common Variable
#530	Common Variable
#531	Common Variable
#1000	PMC Bit Read
#1001	PMC Bit Read
#1002	PMC Bit Read
#1003	PMC Bit Read
#1004	PMC Bit Read
#1005	PMC Bit Read
#1006	PMC Bit Read
#1007	PMC Bit Read
#1008	PMC Bit Read
#1009	PMC Bit Read
#1010	PMC Bit Read
#1011	PMC Bit Read
#1012	PMC Bit Read

Variable	Description
#1013	PMC Bit Read
#1014	PMC Bit Read
#1015	PMC Bit Read
#1032	PMC Word Read
#1100	PMC Bit Write
#1101	PMC Bit Write
#1102	PMC Bit Write
#1103	PMC Bit Write
#1104	PMC Bit Write
#1105	PMC Bit Write
#1106	PMC Bit Write
#1107	PMC Bit Write
#1108	PMC Bit Write
#1109	PMC Bit Write
#1110	PMC Bit Write
#1111	PMC Bit Write
#1112	PMC Bit Write
#1113	PMC Bit Write
#1114	PMC Bit Write
#1115	PMC Bit Write
#1132	PMC Word Write
#1133	PMC Double Word Write

Variable	Description
#3000	Alarm & Stop
#3001	Timer (m/s)
#3002	Timer (hourly)
#3003	Single Block
#3004	Feed control
#3005	
#3006	Operator Message
#3007	
#3008	
#3009	
#3010	
#3011	Date
#3012	Time

#3901	Machine Parts
#3902	Required Parts

#4001	Modal Group 1
#4002	Modal Group 2
#4003	Modal Group 3
#4004	Modal Group 4
#4005	Modal Group 5
#4006	Modal Group 6
#4007	Modal Group 7
#4008	Modal Group 8
#4009	Modal Group 9
#4010	Modal Group 10
#4011	Modal Group 11
#4012	Modal Group 12
#4013	Modal Group 13
#4014	Modal Group 14
#4015	Modal Group 15
#4016	Modal Group 16
#4017	Modal Group 17
#4018	Modal Group 18
#4019	Modal Group 19
#4020	Modal Group 20
#4021	Modal Group 21
#4022	Modal Group 22
#4102	Modal B Code
#4107	Modal D Code
#4109	Modal F Code
#4111	Modal H Code
#4113	Modal M Code
#4114	Modal Sequence No
#4115	Modal Program No

Variable	Description
#4119	Modal S Code
#4120	Modal T Code
#4130	Modal P Code

#5001	Workpiece Position 1st Axis (B)
:	:
#5008	Workpiece Position 8th Axis (B)
#5021	Machine Position 1st Axis
:	:
#5028	Machine Position 8th Axis
#5041	Workpiece Position 1st Axis (C)
:	:
#5048	Workpiece Position 8th Axis (C)
#5061	Skip Signal Position 1st Axis
:	:
#5068	Skip Signal Position 8th Axis

#5201	1st Axis EXT Zero Offset
:	:
#5208	8th Axis EXT Zero Offset
#5221	1st Axis G54 Zero Offset
:	:
#5228	8th Axis G54 Zero Offset
#5241	1st Axis G55 Zero Offset
:	:
#5248	8th Axis G55 Zero Offset
#5261	1st Axis G56 Zero Offset
:	:
#5268	8th Axis G56 Zero Offset
#5281	1st Axis G57 Zero Offset
:	:
#5288	8th Axis G57 Zero Offset
#5301	1st Axis G58 Zero Offset
:	:
#5308	8th Axis G58 Zero Offset
#5321	1st Axis G59 Zero Offset
:	:
#5328	8th Axis G59 Zero Offset