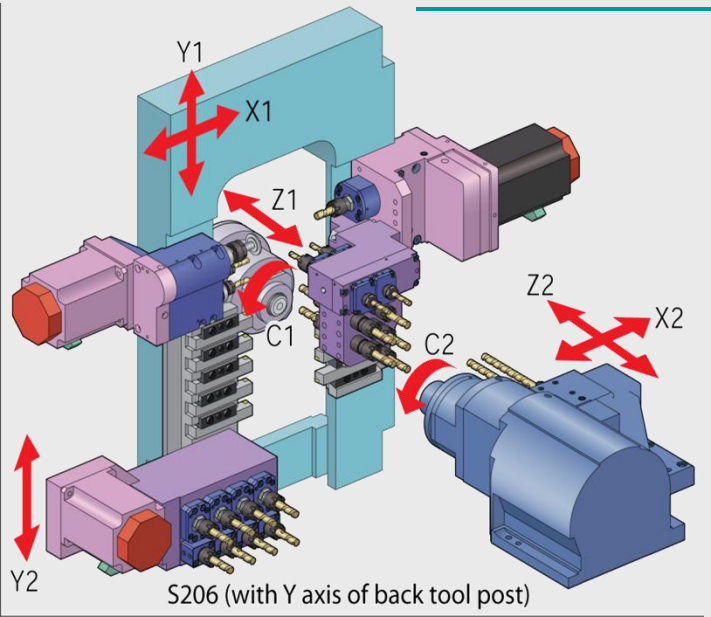
TSUGAMI
Rem Sales, LLC

Custom Macro Programming



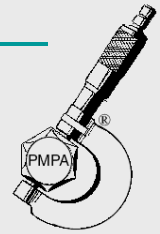
S206 (with Y axis of back tool post)

Parametric Programming

PMPA NTC 2013

Presented by Ron Gainer & Dan Murphy
Tsugami/Rem Sales

Safety



- Every effort has been made to verify the information presented here is correct, however you should assume it is not and exercise caution when putting any of the examples presented here to use.

TSUGAMI
Rem Sales, LLC

Safety

- Macro programs can affect the motions of the machine tool. They can also be used to change coordinate data and parameter settings. Any of these changes can cause unexpected motion, crashes, and other machine malfunctions. Always exercise caution and follow safe procedures for proving out new programs or edits to existing programs.

Back Up Your Data

- Since Macro programming can be used to change parameters, offsets, and work coordinate data, make sure you have recently backed up your parameters, settings, work shifts, and offsets.

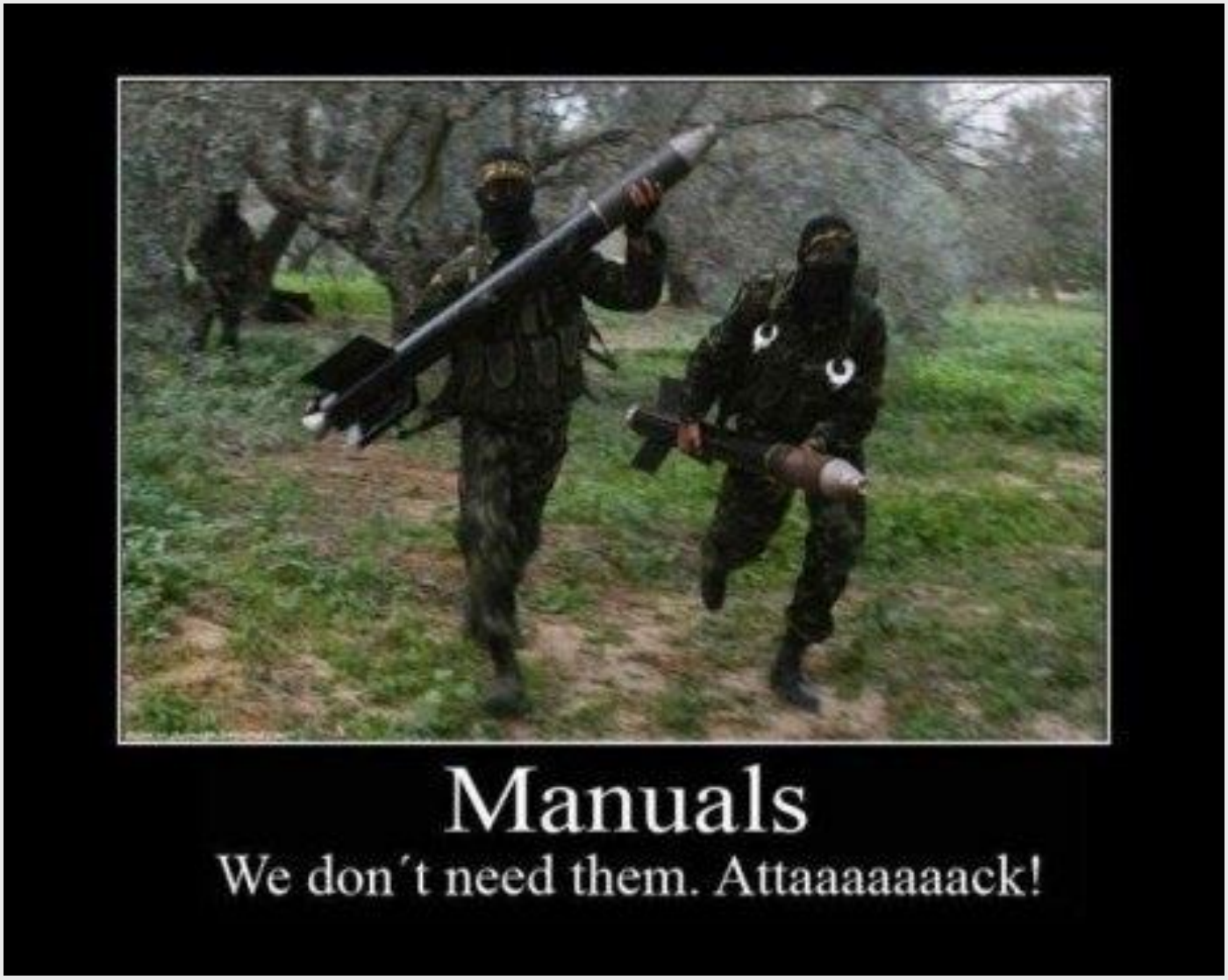
TSUGAMI
Rem Sales, LLC

Introduction

- This Seminar is meant to be an introduction to Custom Macro Programming. There is no way to cover all the material you should be aware of in such a short time period.
- Be sure to read the Custom Macro section of the manual for your particular control

TSUGAMI
Rem Sales, LLC

Don’t Be These Guys

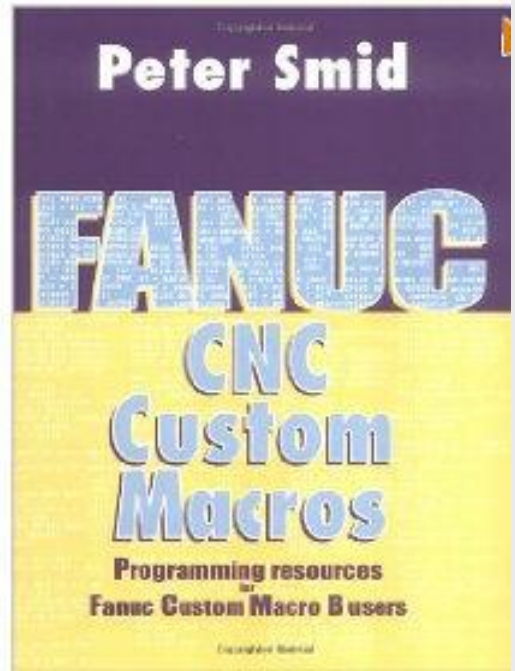


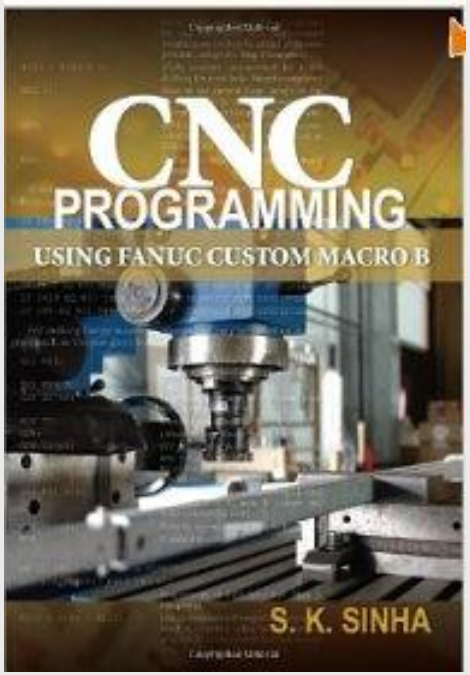
Introduction

- This program concerns the application and use of “Custom Macro “B” on Fanuc controls. Particularly the Fanuc Oi

- The general concepts apply to almost any CNC control that has parametric programming capabilities.

For Further Learning





TSUGAMI
Rem Sales, LLC

For Further Learning

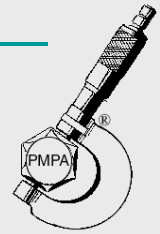
- Fanuc USA offers a course called “Programming Level II” that cover Macro B

- Call - 888-FANUC-US (888-326-8287)

FANUC
FA AMERICA

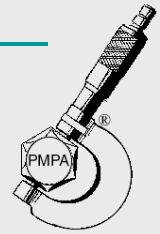
TSUGAMI
Rem Sales, LLC

Introduction



- Applications for Custom Macro Programming
  - Family of parts programming
  - Custom cycles
    - Canned or multiple repetition
  - Program complex motions
    - Parabolas, ellipses, etc
  - Use variables to adjust dimensions that offsets alone can't change
  - "Smart" programs
    - Macro programs can make decisions by using conditional statements
      - IF_THEN, AND, OR, XOR, IF_GOTO, GOTO, WHILE_DO, END
    - Custom alarms and error traps

Introduction

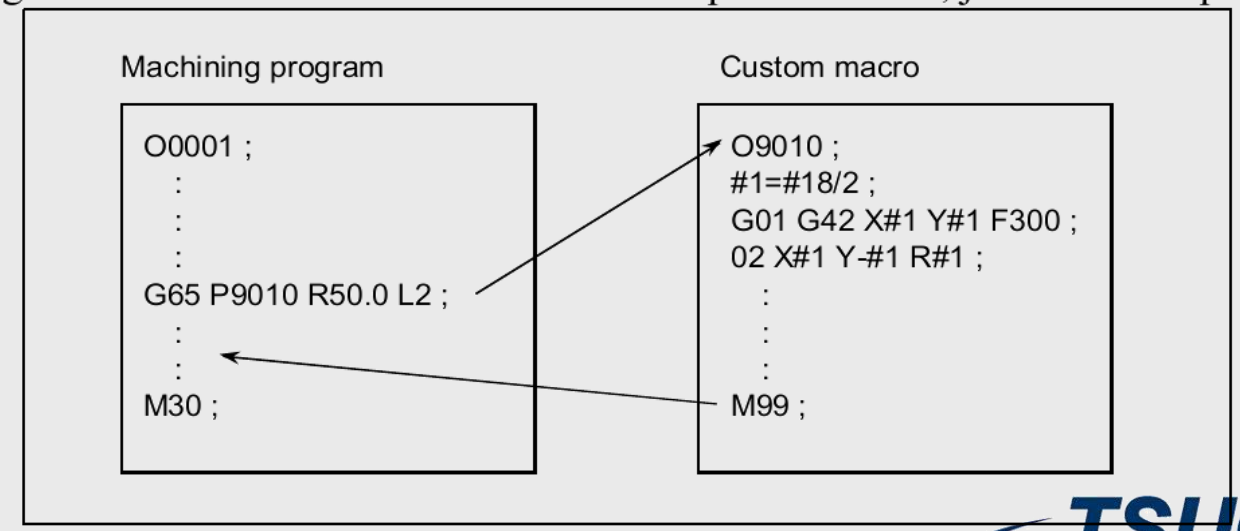


- Applications for Custom Macro Programming
  - Probing and in process gauging
  - Communication with external devices
    - Automation
    - Printers
    - Metrology devices
  - Machine status
    - Check if machine is in proper position
      - “Safe Start” macro
    - Check that offset data and work shifts are within range
    - Check other conditions
      - Cut off tool breakage on Swiss

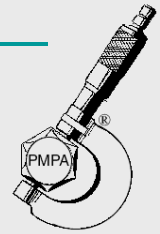
Uses for Macro Programming

- Repetitive operations
  - Similar to using a sub program, but a macro will allow you to change conditions without having to edit multiple lines of code.
  - Example: Milling a pocket taking a number of passes. Using variables will allow you to change the DOC and number of passes without having to rewrite the sub routine.

A machining program can call a custom macro with a simple command, just like a subprogram.



Uses for Macro Programming



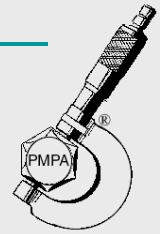
- Family of parts programming
  - One program can run all variations of a part on a tabled drawing.
- Variables are used in place of coordinate numbers in the program.

Conventional program
G01 Z0.5 F0.003

Macro program
G01 Z#501 F0.003

TSUGAMI
Rem Sales, LLC

Variables

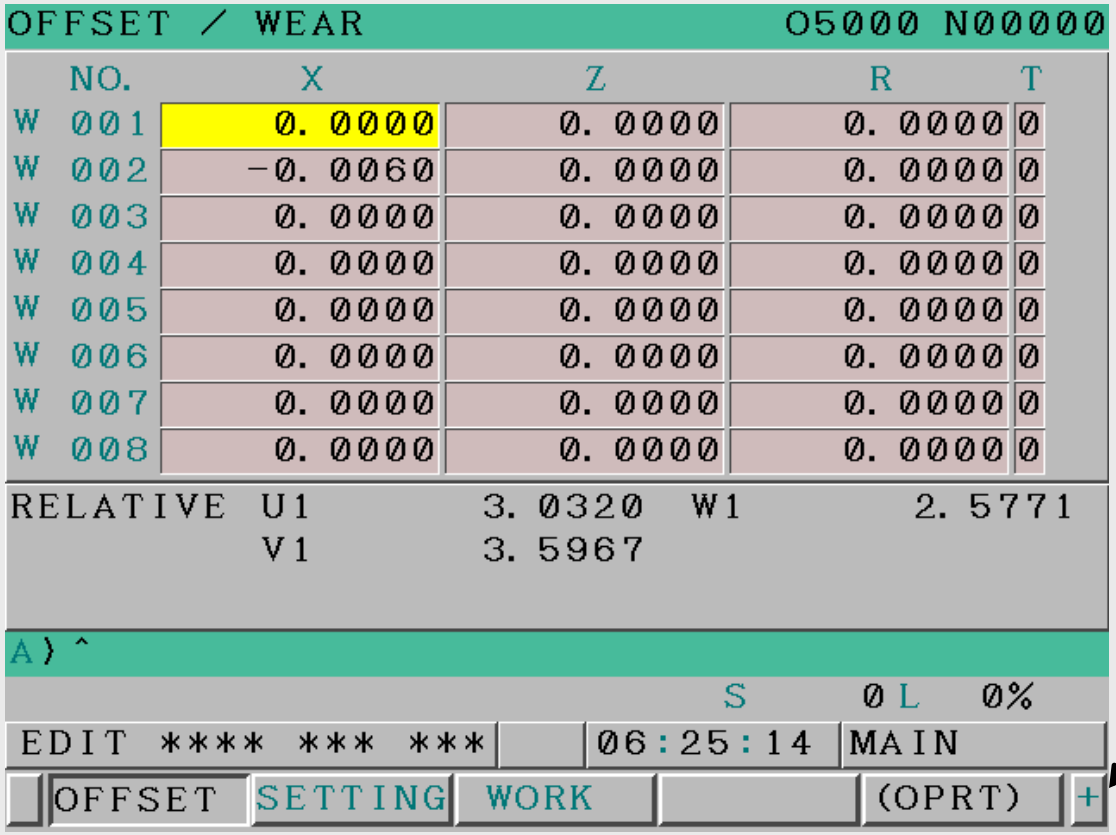


- Variables are expressed in the program by a numerical value preceded by the pound sign
  - #100, #500
- The numerical value corresponds to a register on the macro page of the CNC display

TSUGAMI
Rem Sales, LLC

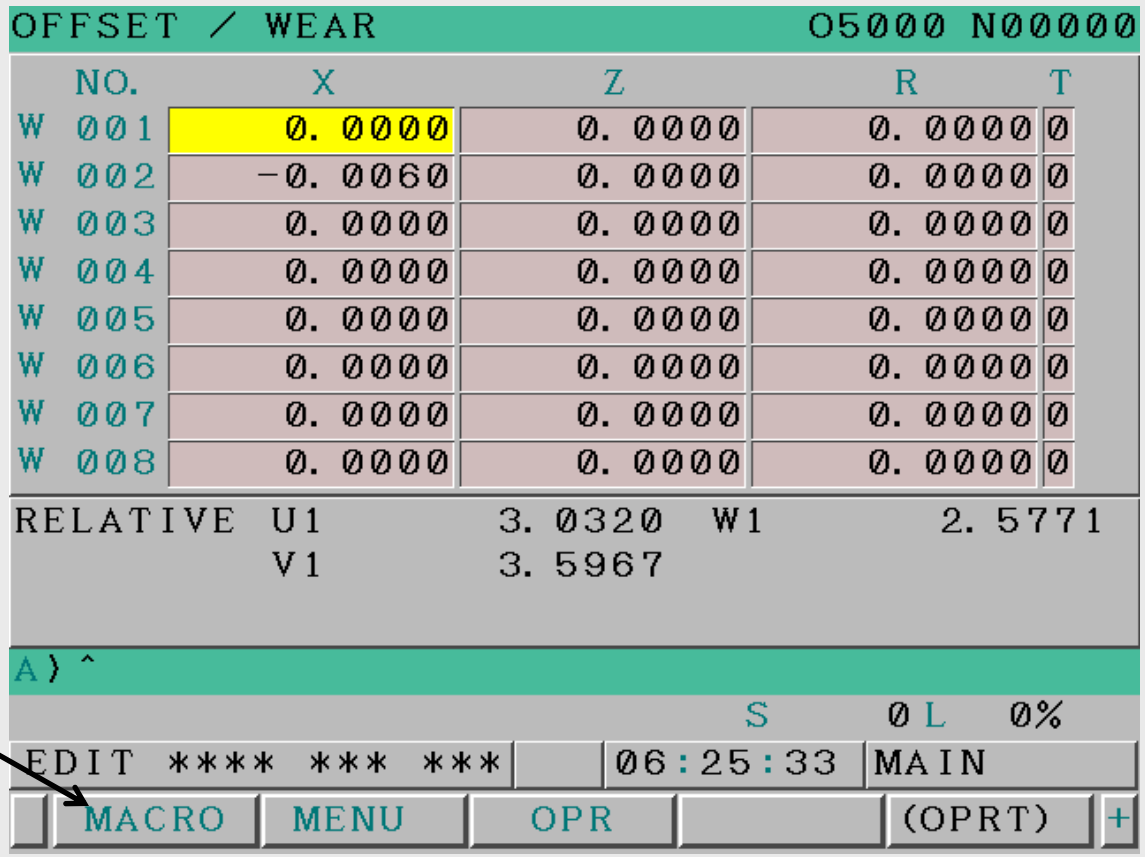
Navigating to the Macro Variable Register

- 1st Press the Offset Key on Main Panel
- 2nd Press the right softkey that has the + symbol.



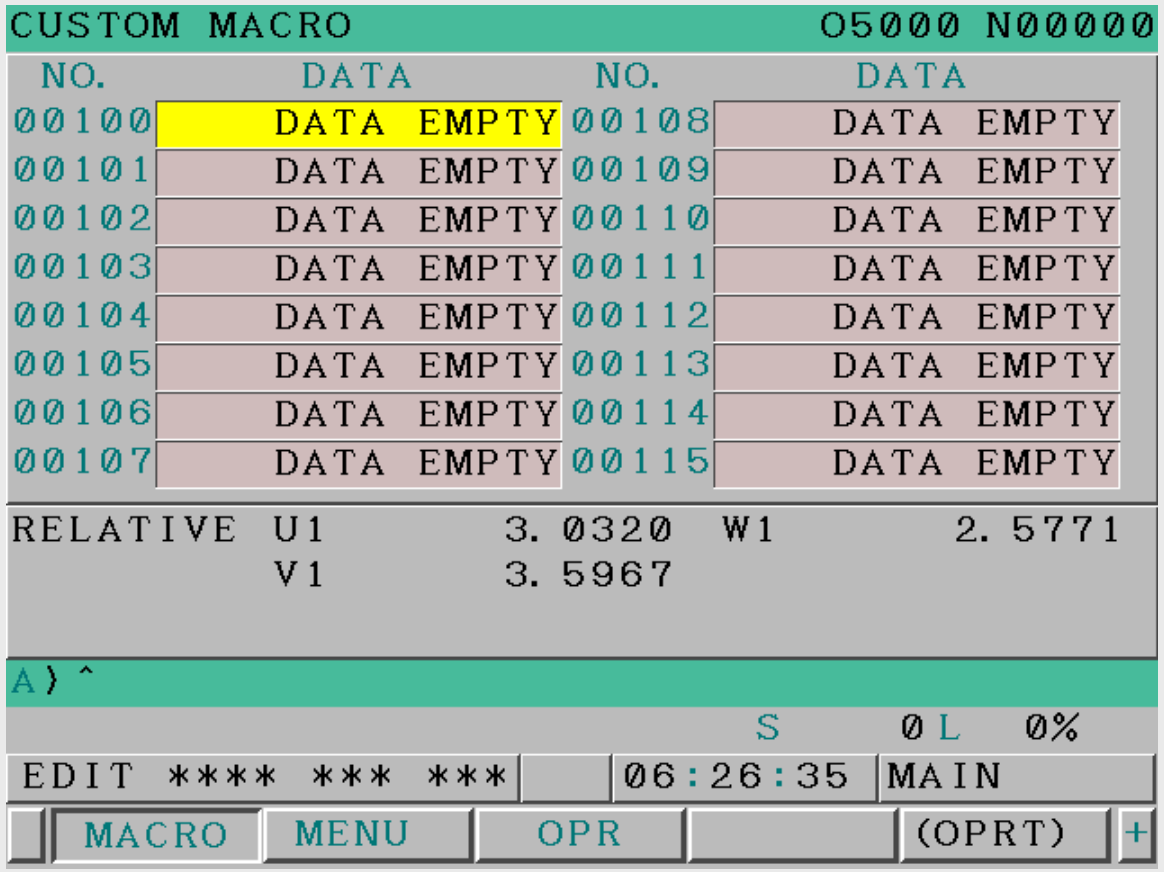
Navigating to the Macro Variable Register

- 3rd Press the Macro Softkey.



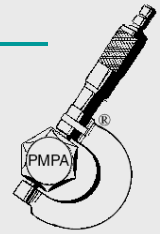
Navigating to the Macro Variable Register

- Macro Register Screen



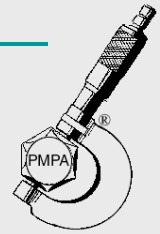
TSUGAMI
Rem Sales, LLC

Assigning a Value to a variable



- Values can be input manually into a variable register
- Values can also be assigned via the NC program
  - #510 = 1.5
  - #100 = #100 + 1
- Range is eight digits maximum
  - #510 = 12345678
  - #510 = 1234.5678
  - #510 = 12345.6789 will generate “TOO MANY DIGITS” alarm
- Decimal points are not considered digits however leading zeros are
  - #510 = 1234.5678 is ok
  - #510 = 0.12345678 is too many digits

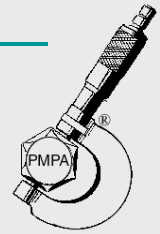
Variables



- Variables Can Only be Mathematical Quantities (numbers)
  - Can not have values of “Yes” or “No”
  - Can’t be “True” or “False”
  - Can’t be letters

TSUGAMI
Rem Sales, LLC

A Variable Can Be Made Conditional




```
O1234;
#10 = 60;
#20 = 30;
IF [#10 GT #20] THEN #1 = #10 - #20;
M#1;
```

(Variable 10 is assigned a value of 60)
(Variable 20 is assigned a value of 30)
(If the value of variable 10 is greater than variable 20 = “True” condition
So the value of variable 1 = 30)

TSUGAMI
Rem Sales, LLC

Expressions

- An expression is either a mathematical statement or a conditional statement.

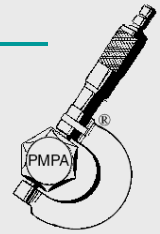
Conditional Statement – IF [#103 EQ 0] GOTO5

If variable number 103 is equal to zero go to line N5

Mathematical Statement - #107 = #106 - #105

Subtract variable number 105 from variable number 106 and place the result in variable number 107

Expressions



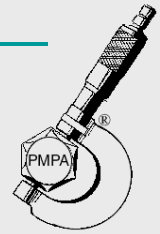
- Enclosing a value within brackets [ ] is considered a macro expression
  - G00 X[.5]
  - G01 X[.5-.2]
- Brackets also denote the hierarchy of operations

IF [#1 LT #2] GOTO 99 (Compare #1 to #2, then if true go to line N99)

#100 = #20 – [#10 * 2] (Multiply #10 by 2, then subtract the result from #20)

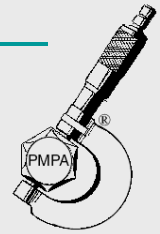
TSUGAMI
Rem Sales, LLC

Expressions



- A macro variable can also be represented as an expression
  - #[#100]
  - #[#1001-1]
  - #[#6/2]

Substituting Variable Values in Program



- All NC “words” can have a variable as their value, except “O” (program number) and “N” (sequence number)

Coordinate word examples:
#100 = 1.2;
G00 X#100; (same as G00 X1.2)

S-codes
#100 = 2500;
G97 M03 S#100; (same as M03 S2500)

F-codes
#100 = .003;
G99 G01 X.5 F#100; (same as F0.003)

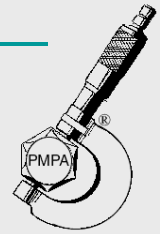
G-codes
#10 = 00
G#10 X1.2 (same as G00)

O#100 (Illegal)

N#100 (Illegal)

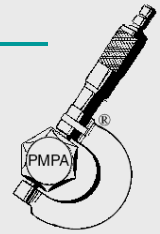
TSUGAMI
Rem Sales, LLC

Substituting Variable Values in Program



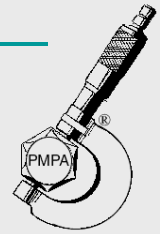
- Caution Must Be Exercised
  - Using variables with G-codes is not good practice
  - Whenever variables are used in place of coordinate values and/or NC words, unexpected machine movement can result

Substituting Variable Values in Program



- Understand how real numbers and integers are treated by the CNC control
  - Real numbers are any number rational or irrational
    - Real numbers include integers
    - 1.25, -.3765, 5, -10 are all real numbers
  - Integers are whole numbers
    - 2500, 3, 20
    - Numbers with decimals are not integers i.e. 1.25
  - Some NC words only allow integers
    - S-commands: M03 S2000 is ok, M03 S2000.5 is not

Substituting Variable Values in Program



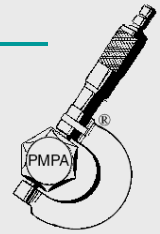
- Understand how real numbers and integers are treated by the CNC control
- Some NC words only allow integers
  - M-codes: M05 is acceptable M05.5 is not
- Use of brackets can in some cases round to nearest integer
- In some cases variables will automatically be rounded
- Always refer to the manual to eliminate any doubt

Example:
M03 S1001.5 (an illegal statement as S-codes have to be integers)

Example:
M03 S[1000.5] (will be treated as M03 S1001)

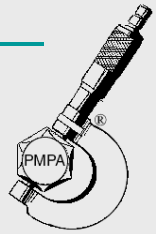
Example:
#1=1000.5
M03 S#1 (will be treated as M03 S1001)

Substituting Variable Values in Program



- G-codes present a special challenge
  - As a rule NEVER USE VARIABLES TO EXPRESS THE VALUE OF A G-CODE
- Despite being able to use real numbers with G-codes (G12.1, G52.1, etc) the control will round up a variable from one place to the right of the decimal and treat the value as an integer only

Variable That Contains No Value



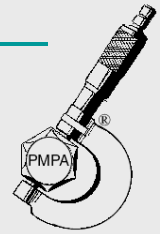
- A variable with no value is considered “Null”
  - Null does not necessarily equal zero
- When a variable of null value is used, the control will, in most cases, ignore the command
- This can be dangerous

Example:
Variable 100 equals null

G97 M03 S#100 (The spindle will start clockwise at the last commanded value)

TSUGAMI
Rem Sales, LLC

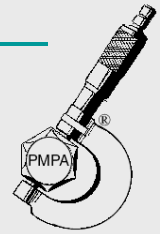
Safety



- An operator was killed by a boring head on a machining center where the RPM was specified by a variable
- The variable was set to null, so the last programmed RPM was used by the control
- The RPM exceeded the rating of the boring head
- The head came apart and the operator was struck by the debris and killed

TSUGAMI
Rem Sales, LLC

Safety



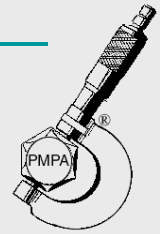
- If you are using a variable and know that if the value is set outside a certain range that danger exists, then add safety checks to your code without fail.

Example: Spindle can run up to 6000 RPM but the boring head in use is rated to 2,500RPM

```
#100 = 6000;
M03 S#100;          <-------(RPM = 6000, Boring Head Blows Up)
```

```
#100 = 6000;
IF [#100 GT 2500] THEN #3000=1 (RPM TOO HIGH);<-(Conditional statement that
M03 S#100;                                       checks that the RPM does not
                                                 exceed 2500. If it does
                                                "ALARM 3001 RPM TOO HIGH"
                                                will be generated)
```

Variable #0



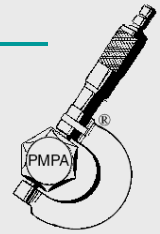
- #0 is a read only System variable permanently set to null so it can be used for comparison
- #0 can also be used to set another variable to null
  - #1 = #0 sets #1 to null
  - #0 = #1 illegal statement because #0 is read only
- In the previous example the RPM check does not prevent a null value being used which would cause the spindle to run at the last programmed RPM.
  - In the case of the boring head, this is what happened
- #0 can be used in a safety check below:

```
#100 = #0;
IF [#100 GT 2500] THEN #3000=1 (RPM TOO HIGH);
IF [#100 EQ #0] THEN #3000=2 (NO RPM SET IN #100);
G97 M03 S#100;
```

← ALARM 3002 will be generated.

TSUGAMI
Rem Sales, LLC

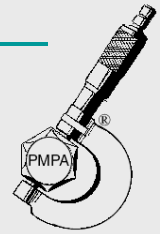
How Null Variables Behave in Arithmetic



- All variables are assumed null in these examples
- In math operations null (no value) is looked at as zero

#2 = #1;	#2 remains null because there is no math or evaluation in this statement
#2 = [#1];	#2 remains null. Even though the brackets make this an expression no math or evaluation occurs
#2 = #1 + #1	#2 will equal zero. No value plus no value has a value of zero
#2 = #1 * 5	#2 will equal zero

How Null Variables Behave in Expressions



- In comparative expressions EQ (equal to) and NE (not equal to), null variables are not considered the same as 0

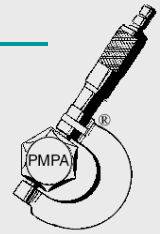
Examples - #1 = null

[#1 EQ 0]	False, null variable is not equal to 0
[#1 NE 0]	True, null variable is not equal to 0
[#1 EQ #0]	True, #0 value is always null
[#2 NE #0]	False, #0 value is always null

Conditional Statement – IF [#1 EQ 0] THEN (some action)

TSUGAMI
Rem Sales, LLC

How Null Variables Behave in Expressions

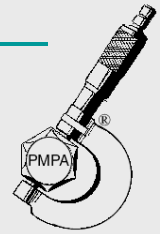


- In all other conditional expressions a null value variable is considered 0
  - LT – less than, LE - less than or equal to
  - GT – Greater than, GE - Greater than or equal to

[#1 LT 0]	False, 0 is not less than 0
[#1 LE 0]	True, 0 is equal to 0
[#1 GT 0]	False, 0 is not greater than 0
[#1 GE 0]	True, 0 is equal to zero

TSUGAMI
Rem Sales, LLC

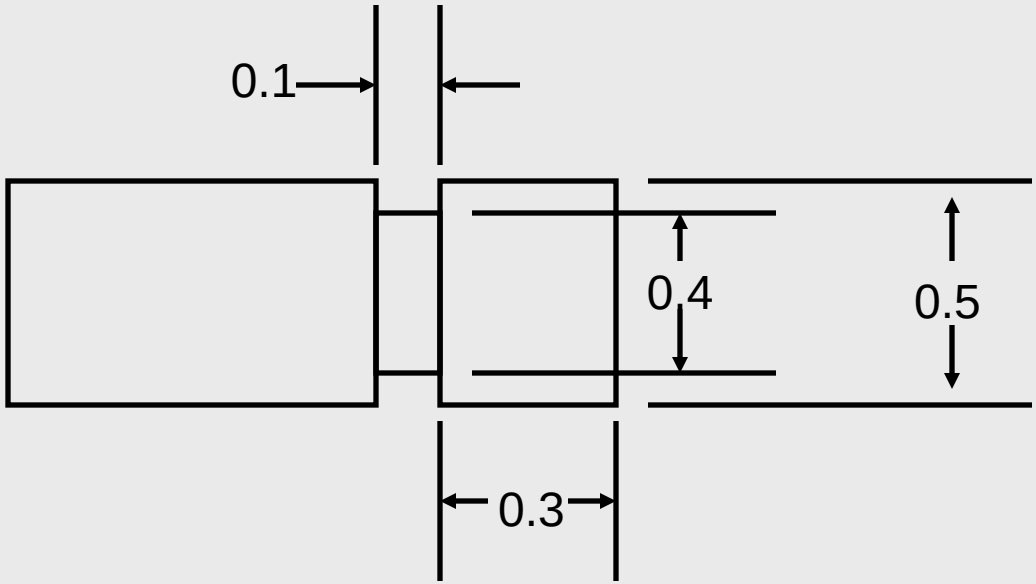
Using Variables



- At this point you have enough knowledge to use macro variables in two of the most common ways.
  - Adjusting feature size
  - Family of part programming

Adjusting Feature Size

- Width of a groove
  - 0.093” groove tool width



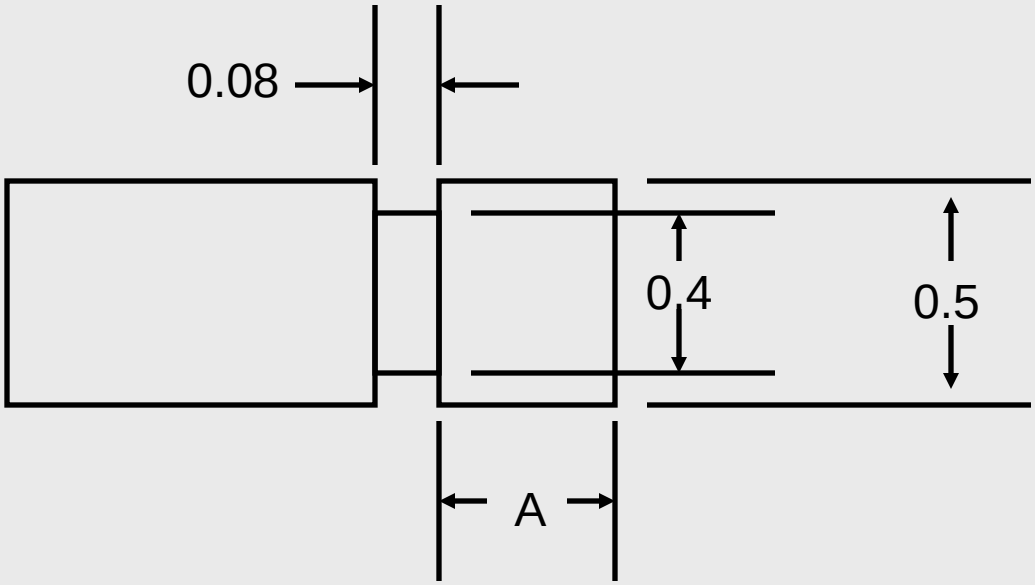

```
T1414;
G96 M03 S200;
G50 S5000;
G00 X.55 Z.3;
G99 G01 X.4 F.003
G00X.55
Z[.307+#514]
G01 X.4
G00X1.18 T0
```

TSUGAMI
Rem Sales, LLC

Family of Part Programming

- Variable feature location
  - 0.08” groove tool width

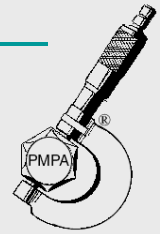
```
T1414;
G96 M03 S200;
G00 X.55 Z.#514;
G99 G01 X.4 F.003
G00X1.18 T0
```



PN	A
-1	.3
-2	.4
-3	.5

TSUGAMI
Rem Sales, LLC

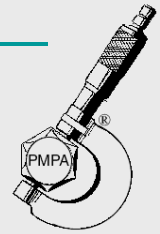
Types of Variables



- Read only permanent value (#0, 3100-3102)
- Local variables (#1 through #33)
- Common variables (#100 through #199)
- Permanent common variable (#500-#599)
  - Can also be made “read only” by parameter setting
- System variables

TSUGAMI
Rem Sales, LLC

Local Variables (#1 - #33)



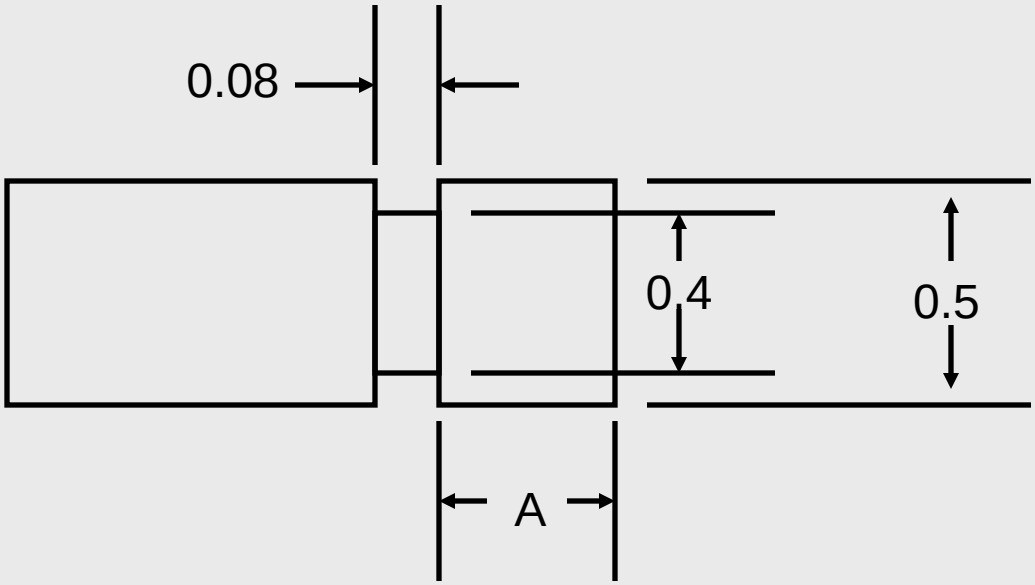
- A local variable is used locally as opposed to globally, that is to say that a local variable called by one macro call at a certain time is different from the same variable called at another time by a different macro call.
- Local variables are used to pass arguments from a macro call
- Variable number #1-#33 are tied to letter addresses in the macro call

Macro Call for Variable Feature

- Variable feature location

Call from main program
G65 P9014 A.3

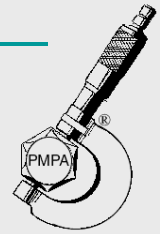
```
O9014
T1414;
G97 M03 S2000;
G00 X.55 Z.#1;
G99 G01 X.4 F.003
G00X1.18 T0
M99
```



PN	A
-1	.3
-2	.4
-3	.5

TSUGAMI
Rem Sales, LLC

Macro Argument Correlation



- Argument specification I

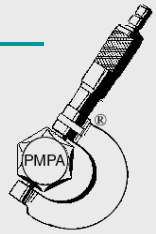
Address	Variable number
A	#1
B	#2
C	#3
D	#7
E	#8
F	#9
H	#11

Address	Variable number
I	#4
J	#5
K	#6
M	#13
Q	#17
R	#18
S	#19

Address	Variable number
T	#20
U	#21
V	#22
W	#23
X	#24
Y	#25
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.
  Argument specification I is always used for I, J, and K by setting bit 7 (IJK) of parameter No. 6008 to 1

Variables



- **Range of variable values**

Local and common variables can have a value in the following ranges. If the result of calculation exceeds the range, an alarm PS0111 is issued.

When bit 0 (F0C) of parameter No.6008 = 0

Maximum value: approx. $\pm 10^{308}$

Minimum value: approx. $\pm 10^{-308}$

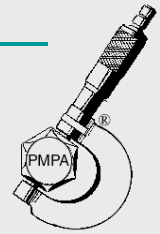
Numeric data handled by a custom macro conforms to the IEEE standard and is handled as a double-precision real number. An error resulting from operation depends on the precision.

When bit 0 (F0C) of parameter No.6008 = 1

Maximum value: approx. $\pm 10^{47}$

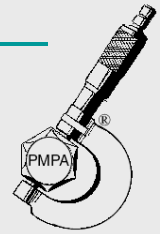
Minimum value: approx. $\pm 10^{-29}$

Common Variables #100-#199



- These variables are global. They have the same value in any macro call.
- These variable are read/write unless protected by parameter change
  - Parameter 6031 and 6032 on FS OiD
- These variables are reset to null upon power down

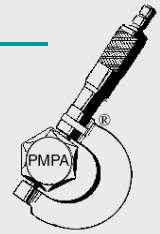
Common Variables #500-#999



- These variables are global. They have the same value in any macro call.
- These variable are read/write unless protected by parameter change
  - Parameter 6031 and 6032 on FS OiD
- These variables will retain their values upon power down, reset, etc.
- These variables are preferred when input from the operator is required
  - Feature size adjustment
  - Family of parts
  - Tool life management count values

TSUGAMI
Rem Sales, LLC

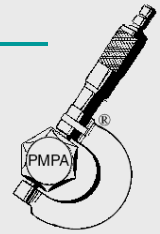
System Variables



- Variable whose usage does not vary in the system
  - The variable's address will always return the same information, though the value will vary
- System variables can be Read only, Read/Write, or Write only depending on the nature of the variable
- System Constants are system variables whose values are fixed. They are Read only.

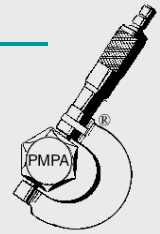
Constant number	Constant name	Description
#0, #3100	[#_EMPTY]	Null
#3101	[#_PI]	Circular constant π= 3.14159265358979323846
#3102	[#_E]	Base of natural logarithm e= 2.71828182845904523536

Variables



- Omission of the decimal point
  - Decimal points can be left out when the value is an integer.
    - #1=123 will be treated the same as #1=123.0000

Specifying a Common Variable by its Name

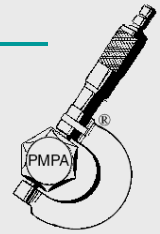


- You can name common variables using the SETVN command.
  - This is useful in that a name that correlates to the variables value is more easily recognizable in the program than a variable number.

Examples:

X[#POS1] Y[#POS2] ; : Specifying a position by the variable name
[#POS1] = #100+#101 ; : Executing a assignment statement by the variable name
#[100+[#ABS]] = 500 ; : Same as above (by a variable number)
#500 = [1000+[#POS2]*10] ; : Reading a variable by a variable name

SETVN Command



- For the 50 common variables, #500 to #549, a name of up to eight characters can be specified by using a command as shown below.

SETVN n [VAR500, VAR501, VAR502,.....] ;

n represents the starting number of a common variable for which the name is specified.
VAR500 is the variable name of variable n, VAR501 is the variable name of variable n+1, and VAR502 is the variable name of variable number n+2, and so on. Each string is delimited by a comma (,). All codes that can be used as meaningful information in a program except control in, control out, [,], EOB, EOR, and : (colon in a program number) can be used. However, each name must begin with an alphabetical character. Variable names are not cleared on switch-off.
Specifying a set variable name allows reading from or writing to the common variable. The command must be specified in the form [#common-variable-name] such as [#VAR500].

[Example] SETVN 510[TOOL_NO, WORK_NO, COUNTER1, COUNTER2];

The command above names the variables as follows.

Variable	Name
#510	#TOOL_NO
#511	#WORK_NO
#512	#COUNTER1
#513	#COUNTER2

The names specified by the command can be used in a program. For example, when 10 is assigned to #510, the expression [#TOOL_NO]=10; can be used instead #510=10;.

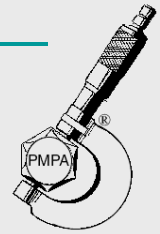
System Variable Names

- Names can be used with system variables
- The name is fixed in the control and can not be changed

Example: Wear Offsets on a T Control

Variable number	Variable name	Position information	Read operation during movement
#5081	[#_TOFSWX]	X-axis tool offset value	Disabled
#5082	[#_TOFSWZ]	Z-axis tool offset value	
#5083	[#_TOFSWY]	Y-axis tool offset value	
#5084	[#_TOFS[4]]	4th axis tool offset value	
#5085	[#_TOFS[5]]	5th axis tool offset value	

System Variable Overview:

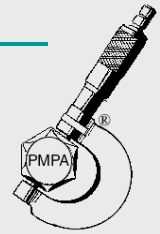


System variables allow you to determine and set machine information. This would include machine position, skip signal position, work & tool offsets. You can also read current “G”, “M”, “S”, “T”, and “H” codes.

There are also System Variables for programmable inputs and outputs.

TSUGAMI
Rem Sales, LLC

Displaying System Variables

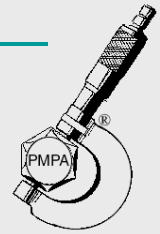


- Unlike common variables, there is no page on the control that will display the value of system variables.
- You can see the value of a system variable by returning it to a common variable

For example to see the system variable for time, command:
#100=#3012
The time will then be stored in variable 100 as Hour/Min/Sec

TSUGAMI
Rem Sales, LLC

List of System Variables



n represents a subscript.
R, W, and R/W are attributes of a variable and indicate read-only, write-only, and read/write enabled, respectively.

- **Interface signals**

System variable number	System variable name	Attribute	Description
#1000-#1031	[#_UI[n]]	R	Interface input signals (BIT), UI000-UI031 NOTE) Subscript n represents a BIT position (0-31).
#1032-#1035	[#_UIL[n]]	R	Interface input signals (LONG), UI000-UI031/ UI100-UI131/ UI200-UI231/UI300-UI331 NOTE) Subscript n (0-3): 0 = UI000-UI031, 1 = UI100-UI131, 2 = UI200-231, 3 = UI300-UI331
#1100-#1131	[#_UO[n]]	R/W	Interface output signals (BIT), UO000-UO031 NOTE) Subscript n represents a BIT position (0-31).
#1132-#1135	[#_UOL[n]]	R/W	Interface output signals (LONG), UO000-UO031/ UO100-UO131/UO200-UO231/UO300-UO331 NOTE) Subscript n (0-3): 0 = UO000-UO031, 1 = UO100-UO131, 2 = UO200-231, 3 = UO300-UO331

System Variable Overview: Descriptions

#1000-#1035=Supplied inputs to CNC

Example: Coolant Off Switch has a supplied input in machine ladder.

IF[#1000EQ1]GOTO881

N881#3000=1(COOLANT OFF ALARM)

TSUGAMI
Rem Sales, LLC

System Variable Overview: Descriptions

#1100-#1135=Programmable outputs to CNC

Example: Hydraulic support activated by macro output

#1100=1(CLOSE SUPPORT)

System Variable Overview: Descriptions

#2000-#2999 Tool Compensation Variables

These values vary depending on the number of offsets and the type of tool offset memory your control has.

They are read/write capable.

System Variable Overview: Descriptions

Tool Offset system example:

- **Tool compensation value**

T

Without tool geometry/wear compensation memory (bit 6 (NGW) of parameter No. 8136 is 1)

System variable number	System variable name	Attribute	Description
#2001-#2064 #10001-#10200	[#_OFSX[n]]	R/W	X-axis compensation value (*1) Note)Subscript n represents a compensation number (1 to 64). The numbers on the left are also allowed. Note)Subscript n represents a compensation number (1 to 200).
#2101-#2164 #11001-#11200	[#_OFSZ[n]]	R/W	Z-axis compensation value (*1) Note)Subscript n represents a compensation number (1 to 64). The numbers on the left are also allowed. Note)Subscript n represents a compensation number (1 to 200).
#2201-#2264 #12001-#12200	[#_OFSR[n]]	R/W	Tool nose radius compensation value Note)Subscript n represents a compensation number (1 to 64). The numbers on the left are also allowed. Note)Subscript n represents a compensation number (1 to 200).
#2301-#2364 #13001-#13200	[#_OFST[n]]	R/W	Virtual tool tip T position Note)Subscript n represents a compensation number (1 to 64). The numbers on the left are also allowed. Note)Subscript n represents a compensation number (1 to 200).
#2401-#2449 #14001-#14200	[#_OFSY[n]]	R/W	Y-axis compensation value (*1) Note)Subscript n represents a compensation number（1 to 49） The numbers on the left are also allowed. Note)Subscript n represents a compensation number (1 to 200).

(*1) X-axis: X-axis of basic three axes, Z-axis: Z-axis of basic three axes, Y-axis: Y-axis of basic three axes

System Variable Overview: Descriptions

System variable number	System variable name	Attribute	Description
#3003	[#_CNTL1]	R/W	Enable or disable the suppression of single block stop. Enable or disable the waiting of the auxiliary function completion signal.
#3003 bit0	[#_M_SBK]	R/W	Enable or disable the suppression of single block stop.
#3003 bit1	[#_M_FIN]	R/W	Enable or disable waiting for the auxiliary function completion signal.
#3004	[#_CNTL2]	R/W	Enable or disable feed hold. Enable or disable feedrate override. Enable or disable exact stop check.
#3004 bit0	[#_M_FHD]	R/W	Enable or disable feed hold.
#3004 bit1	[#_M_OV]	R/W	Enable or disable feedrate override.
#3004 bit2	[#_M_EST]	R/W	Enable or disable exact stop check.
#3005	[#_SETDT]	R/W	Read/write setting data.
#3006	[#_MSGSTP]	W	Stop with a message.
#3007	[#_MRIMG]	R	Status of a mirror image (DI and setting)
#3008	[#_PRSTR]	R	Restarting/not restarting a program

- **Time**

System variable number	System variable name	Attribute	Description
#3011	[#_DATE]	R	Year/Month/Date
#3012	[#_TIME]	R	Hour/Minute/Second

- **Number of parts**

System variable number	System variable name	Attribute	Description
#3901	[#_PRTSA]	R/W	Total number of parts
#3902	[#_PRTSN]	R/W	Number of required parts

System Variable Overview: Descriptions

Commonly used variables:

- #3000 is used to generate a macro alarm. #3000=1(VALUE A TOO LARGE) This will stop the machine in alarm status with alarm 3001(VALUE A TOO LARGE) Message displayed on the screen. W
- #3006 is used to stop the machine in feed hold status with a message. W

  #3006=1(INSTALL SUPPORT).
- #3007=Mirror Image status R

System Variable Overview: Descriptions

Commonly used variables:

- #3011 Date (Year/Month/Date) R
- #3012 Time (Hour/Minute/Second) R
- #3901 Total Number of Parts R/W
- #3902 Number of Required Parts R/W
- #4000 Main Program # R

TSUGAMI
Rem Sales, LLC

System Variable Overview: Descriptions

Modal Information:

- #4001-#4030 Model G-code information (R)

Example #4001=Active G-code of Group 1

Group 1 G-codes include: G00,G01,G02,G03,G32.

#100=#4001 Loads the current active code to #100.

G-Code Groups

T

Table 3.2 (a) G code list

G code system A	G code system B	G code system C	Group	Function
G00	G00	G00	01	Positioning (Rapid traverse)
G01	G01	G01		Linear interpolation (Cutting feed)
G02	G02	G02		Circular interpolation CW or helical interpolation CW
G03	G03	G03		Circular interpolation CCW or helical interpolation CCW
G04	G04	G04	00	Dwell
G05.4	G05.4	G05.4		HRV3 on/off
G07.1 (G107)	G07.1 (G107)	G07.1 (G107)		Cylindrical interpolation
G08	G08	G08		Advanced preview control
G09	G09	G09		Exact stop
G10	G10	G10		Programmable data input
G11	G11	G11		Programmable data input mode cancel
G12.1 (G112)	G12.1 (G112)	G12.1 (G112)	21	Polar coordinate interpolation mode
G13.1 (G113)	G13.1 (G113)	G13.1 (G113)		Polar coordinate interpolation cancel mode
G17	G17	G17	16	XpYp plane selection
G18	G18	G18		ZpXp plane selection
G19	G19	G19		YpZp plane selection
G20	G20	G70	06	Input in inch
G21	G21	G71		Input in mm
G22	G22	G22	09	Stored stroke check function on
G23	G23	G23		Stored stroke check function off
G25	G25	G25	08	Spindle speed fluctuation detection off
G26	G26	G26		Spindle speed fluctuation detection on
G27	G27	G27	00	Reference position return check
G28	G28	G28		Return to reference position
G30	G30	G30		2nd, 3rd and 4th reference position return
G31	G31	G31		Skip function

System Variable Overview: Descriptions

Modal Information Preceding Block (R)

- #4102= B-Code
- #4107= D-Code
- #4108= E-Code
- #4109= F-Code
- #4111= H-Code
- #4113= M-Code
- #4114= Sequence #
- #4115= Program #

System Variable Overview: Descriptions

Modal Information Preceding Block(R)

- #4119= S-Code
- #4120= T-Code
- #4130= Additional Coordinate System

- #4201 Starts Over with Current Block G-code Group 1 Modal Information.

TSUGAMI
Rem Sales, LLC

System Variable Overview: Descriptions

Position Information:

System variable number	System variable name	Attribute	Description
#5001-#5005	[#_ABSIO[n]]	R	End point position of the previous block (workpiece coordinate system) Note) Subscript n represents an axis number (1 to 5)
#5021-#5025	[#_ABSMT[n]]	R	Specified current position (machine coordinate system) Note) Subscript n represents an axis number (1 to 5).

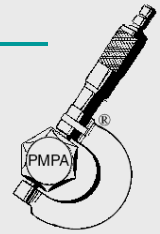
System variable number	System variable name	Attribute	Description
#5041-#5045	[#_ABSOT[n]]	R	Specified current position (workpiece coordinate system) Note) Subscript n represents an axis number (1 to 5).
#5061-#5065	[#_ABSKP[n]]	R	Skip position (workpiece coordinate system) Note) Subscript n represents an axis number (1 to 5).

System Variable Overview: Descriptions

Work Offsets:

System variable number	System variable name	Attribute	Description
#5201-#5205	[#_WZCMN[n]]	R/W	External workpiece origin offset value Note)Subscript n represents an axis number (1 to 5).
#5221-#5225	[#_WZG54[n]]	R/W	G54 workpiece origin offset value Note)Subscript n represents an axis number (1 to 5).
#5241-#5245	[#_WZG55[n]]	R/W	G55 workpiece origin offset value Note)Subscript n represents an axis number (1 to 5).
#5261-#5265	[#_WZG56[n]]	R/W	G56 workpiece origin offset value Note)Subscript n represents an axis number (1 to 5).
#5281-#5285	[#_WZG57[n]]	R/W	G57 workpiece origin offset value Note)Subscript n represents an axis number (1 to 5).
#5301-#5305	[#_WZG58[n]]	R/W	G58 workpiece origin offset value Note)Subscript n represents an axis number (1 to 5).
#5321-#5325	[#_WZG59[n]]	R/W	G59 workpiece origin offset value Note)Subscript n represents an axis number (1 to 5).

Arithmetic and Logic



- Various operations involving arithmetic and/or logic can be performed on variables

#i=<expression>

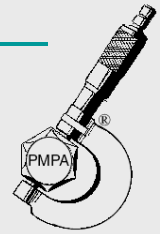
<Expression>

The expression to the right of the arithmetic and logic operation contains constants and/or variables combined by a function or operator. Variables #j and #k below can be replaced with a constant. If a constant used in an expression has no decimal point, it is assumed to end with a decimal point.

Table 14.3 (a) Arithmetic and logic operation

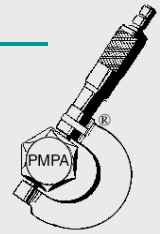
Type of operation	Operation	Description
<1> Definition or replacement	#i=#j	Definition or replacement of a variable
<2> Addition-type operations	#i=#j+#k #i=#j-#k #i=#j OR #k #i=#j XOR #k	Addition Subtraction Logical OR (bit by bit of 32 bits) Exclusive OR (bit by bit of 32 bits)
<3> Multiplication-type operations	#i=#j*#k #i=#j/#k #i=#j AND #k #i=#j MOD #k	Multiplication Division Logical AND (bit by bit of 32 bits) Remainder (A remainder is obtained after #j and #k are rounded to their nearest whole numbers. When #j is a negative value, #i is assumed to be a negative value.)

Arithmetic and Logic



Type of operation	Operation	Description
<4> Functions	#i=SIN[#j]	Sine (in degrees)
	#i=COS[#j]	Cosine (in degrees)
	#i=TAN[#j]	Tangent (in degrees)
	#i=ASIN[#j]	Arc sine
	#i=ACOS[#j]	Arc cosine
	#i=ATAN[#j]	Arc tangent (one argument), ATN can also be used.
	#i=ATAN[#j]/[#k]	Arc tangent (two arguments), ATN can also be used.
	#i=ATAN[#j,#k]	Arc tangent (two arguments), ATN can also be used.
	#i=SQRT[#j]	Square root, SQR can also be used.
	#i=ABS[#j]	Absolute value
	#i=BIN[#j]	Conversion from BCD to binary
	#i=BCD[#j]	Conversion from binary to BCD
	#i=ROUND[#j]	Rounding off, RND can also be used.
	#i=FIX[#j]	Rounding down to an integer
	#i=FUP[#j]	Rounding up to an integer
	#i=LN[#j]	Natural logarithm
	#i=EXP[#j]	Exponent using base e (2.718...)
	#i=POW[#j,#k]	Power (#j to the #kth power)
	#i=ADP[#j]	Addition of a decimal point

Arithmetic and Logic



Explanation

- **Angle units**

The units of angles used with the SIN, COS, ASIN, ACOS, TAN, and ATAN functions are degrees. For example, 90 degrees and 30 minutes is represented as 90.5 degrees.

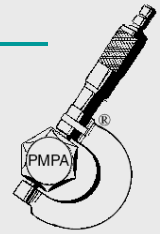
- **ARCSIN #i = ASIN[#j];**

- The solution ranges are as indicated below:
  When the bit 0 (NAT) of parameter No.6004 is set to 0: 270° to 90°
  When the bit 0 (NAT) of parameter No.6004 is set to 1: -90° to 90°
- When #j is beyond the range of -1 to 1, an alarm PS0119 is issued.
- A constant can be used instead of the #j variable.

- **ARCCOS #i = ACOS[#j];**

- The solution ranges from 180° to 0°.
- When #j is beyond the range of -1 to 1, an alarm PS0119 is issued.
- A constant can be used instead of the #j variable.

Arithmetic and Logic



- **ARCTAN #i = ATAN[#j]/[#k]; (two arguments)**

- ATAN[#j,#k] is equivalent to ATAN[#j]/[#k].
- When point (#k,#j) on plane X-Y is given, this function returns the value of the arc tangent for the angle made by the point.
- A constant can be used instead of the #j variable.
- The solution ranges are as follows:

  When the bit 0 (NAT) of parameter No.6004 is set to 0: 0° to 360°

  Example:

  When #1 = ATAN[-1]/[-1]; is specified, #1 is 225.0.

  When the bit 0 (NAT) of parameter No.6004 is set to 1: -180° to 180°

  Example:

  When #1 = ATAN[-1]/[-1]; is specified, #1 is -135.0.

- **ARCTAN #i = ATAN[#j]; (one argument)**

- When ATAN is specified with one argument, this function returns the main value of arc tangent ($-90° \leq ATAN[\#j] \leq 90°$). In other word, this function returns the same value as ATAN in calculator specifications.
- To use this function as the dividend of a division, be sure to enclose it with brackets ([ ]). If this function is not enclosed, ATAN[#j]/[#k] is assumed.

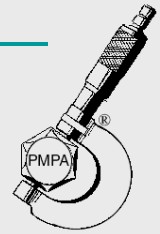
  Example:

  #100 = [ATAN[1]]/10 ; : Divides ATAN with one argument by 10.

  #100 = ATAN[1]/[10] ; : Executes ATAN with two arguments.

  #100 = ATAN[1]/10 ; : Assumes ATAN with two arguments, but issues an alarm PS1131 because the X coordinate specification is not enclosed with brackets ([ ]).

Arithmetic and Logic



- **Exponential function #i = EXP[#j];**

- When the result of the operation overflows, an alarm PS0119 is issued.
- A constant can be used instead of the #j variable.

- **ROUND function**

- 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.

  Example:

  When #1=ROUND[#2]; is executed where #2 holds 1.2345, the value of variable #1 is 1.0.

- When the ROUND function is used in NC statement addresses, the ROUND function rounds off the specified value according to the least input increment of the address.

  Example:

  Creation of a drilling program that cuts according to the values of variables #1 and #2, then returns to the original position

  Suppose that the increment system is 1/1000 mm, variable #1 holds 1.2345, and variable #2 holds 2.3456. Then,

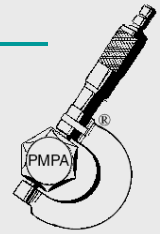
  G00 G91 X-#1 ; Moves 1.235 mm in negative direction.

  G01 X-#2 F300 ; Moves 2.346 mm in negative direction.

  G00 X[#1+#2] ; Since 1.2345 + 2.3456 = 3.5801 in positive direction, the travel distance is 3.580, which does not return the tool to the original position.

  This difference comes from whether addition is performed before or after rounding off. G00X-[ROUND[#1]+ROUND[#2]] ; must be specified to return the tool to the original position.

Arithmetic and Logic



- **Rounding up and down to an integer (FUP and FIX)**

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.

Example:

Suppose that #1=1.2 and #2=-1.2.

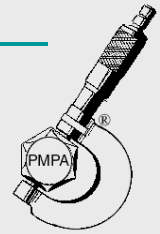
When #3=FUP[#1] ; is executed, 2.0 is assigned to #3.

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.

Arithmetic and Logic



- Order of Operations
  - Algebraic order of operations
- P.E.M.D.A.S.
  1. Parenthesis - In macro programming [ ] = ( )
  2. Exponents
  3. Multiplication
  4. Division
  5. Addition
  6. Subtraction

TSUGAMI
Rem Sales, LLC

Arithmetic and Logic

- **Priority of operations**

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

Example) #1=#2+#3*SIN[#4];

<1>
<2>
<3>

<1>, <2> and <3> indicate the order of operations.

- **Bracket nesting**

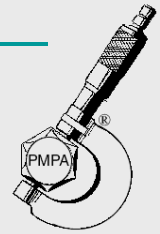
Brackets are used to change the order of operations. Brackets can be used to a depth of five levels including the brackets used to enclose a function. When a depth of five levels is exceeded, an alarm PS0118 occurs.

Example) #1=SIN [[[#2+#3] *#4 +#5] *#6];

<1>
<2>
<3>
<4>
<5>

<1> to <5> indicate the order of operations.

Arithmetic and Logic



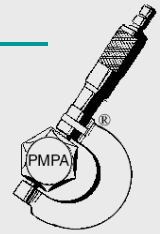
- **Brackets**

Brackets ([ ]) are used to enclose an expression.
Note that parentheses ( ) are used for comments.

- **Divisor**

When a divisor of zero is specified in a division, an alarm PS0112 occurs.

Arithmetic and Logic



- Precision and Errors

- Range of variable values

Local and common variables can have a value in the following ranges. If the result of calculation exceeds the range, an alarm PS0111 is issued.

When bit 0 (F0C) of parameter No.6008 = 0

Maximum value: approx. $\pm 10^{308}$

Minimum value: approx. $\pm 10^{-308}$

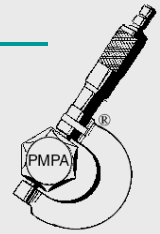
Numeric data handled by a custom macro conforms to the IEEE standard and is handled as a double-precision real number. An error resulting from operation depends on the precision.

When bit 0 (F0C) of parameter No.6008 = 1

Maximum value: approx. $\pm 10^{47}$

Minimum value: approx. $\pm 10^{-29}$

Arithmetic and Logic



Limitation

- **Caution concerning decreased precision**
  When bit 0 (F0C) of parameter No. 6008 is set to 0

- Addition and subtraction

Note that when an absolute value is subtracted from another absolute value in addition or subtraction, the relative error may become 10^{-15} or greater.
For example, assume that #1 and #2 have the following true values in the process of operation.
(The following values are examples in the process of operation and cannot actually be specified from any program.)

#1=9876543210.987654321
#2=9876543210.987657777

You cannot obtain the following result with operation #2-#1:

#2-#1=0.000003456

This is because the precision of custom macro variables is 15 decimal digits. With this precision, the values of #1 and #2 become:

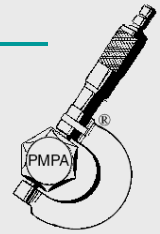
#1=9876543210.987650000
#2=9876543210.987660000

(Precisely, the actual values are slightly different from the above values because they are internally processed in binary.) Therefore, the result is:

#2-#1=0.000010000

A large error occurs.

Arithmetic and Logic



Limitation

- **Caution concerning decreased precision**
  When bit 0 (F0C) of parameter No. 6008 is set to 0

```
#1=9876543210.987654321
#2=9876543210.987657777
```

- Logical expressions
  Be aware of errors that can result from conditional expressions using EQ, NE, GT, LT, GE, and LE because they are processed basically in the same way as addition and subtraction. For example, if the following statement is used to decide whether #1 is equal to #2 in the above example, a correct decision may not be resulted because errors may occur:

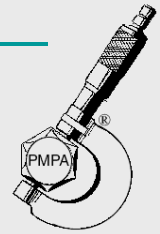
  ```
  IF [#1 EQ #2]
  ```

  Evaluate the difference between #1 and #2 with:

  ```
  IF [ABS [#1-#2]LT 0.1]
  ```

  Then, assume that the values are equal when the difference does not exceed the allowable error range.

Arithmetic and Logic



Limitation

- **Caution concerning decreased precision**
  When bit 0 (F0C) of parameter No. 6008 is set to 0

- Trigonometric functions
  The absolute error is guaranteed for trigonometric functions. However, the relative error is 10^{-15} or greater. Carefully perform multiplication or division after executing a trigonometric function.
- FIX function
  When using the FIX function for the result of an operation, be careful with the precision. For example, when the following operations are performed, the value of #3 may not always be 2.

```
N10 #1=0.002;
N20 #2=#1*1000;
N30 #3=FIX[#2];
```

  This is because an error may occur in operation N20 and the result may not be

```
#2=2.0000000000000000
```

  but a value a little smaller than 2 such as the following:

```
#2=1.9999999999999997
```

  To prevent this, specify N30 as follows:

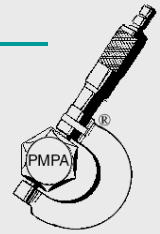
```
N30 #3=FIX[#2+0.001];
```

  Generally, specify the FIX function as follows:

  FIX[expression] → FIX[expression ±ε]

  (Specify +ε when the value of the expression is positive or -ε when it is negative, and 0.1, 0.01, 0.001, ... for ε as required.)

Arithmetic and Logic



When bit 0 (F0C) of parameter No. 6008 is set to 1

Errors may occur when operations are performed.

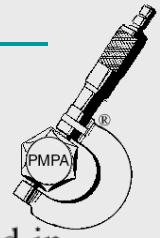
Table 14.3 (b) Errors involved in operations

Operation	Average error	Maximum error	Type of error
a = b*c	1.55×10^{-10}	4.66×10^{-10}	Relative error $\left\lvert\frac{\varepsilon}{a}\right\rvert$
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}	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 $\lvert\varepsilon\rvert$ degrees
a = ATAN [b] / [c]	1.8×10^{-6}	3.6×10^{-6}	

NOTE

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 Note that, in the case of natural logarithm #i=LN[#j]; and exponential function #i=EXP[#j];, the relative error may become 10^{-8} or greater.
6 The operation result of exponential function #i=EXP[#j]; overflows when #j exceeds about 110.

Arithmetic and Logic



- 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.)

- Also be aware of errors that can result from conditional expressions using EQ, NE, GE, GT, LE, and LT.

  Example:

  IF[#1 EQ #2] is effected by errors in both #1 and #2, possibly resulting in an incorrect decision.
  Therefore, instead find the difference between the two variables with IF[ABS[#1-#2]LT0.001].
  Then, assume that the values of the two variables are equal when the difference does not exceed an allowable limit (0.001 in this case).

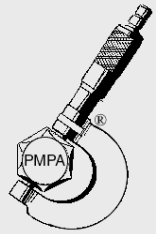
- Also, be careful when rounding down a value.

  Example:

  When #2=#1*1000; is calculated where #1=0.002;, the resulting value of variable #2 is not exactly 2 but 1.99999997.
  Here, when #3=FIX[#2]; is specified, the resulting value of variable #3 is not 2.0 but 1.0.
  In this case, round down the value after correcting the error so that the result is greater than the expected number, or round it off as follows:

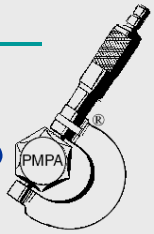
  #3=FIX[#2+0.001]
  #3=ROUND[#2]

TSUGAMI
Rem Sales, LLC



Questions Regarding Arithmetic and Logic?

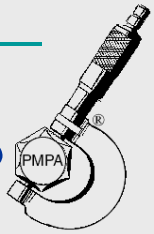
Macro Statements and NC Statements



- The following blocks are referred to as macro statements:
  - Blocks containing arithmetic or logic operations
  - Blocks containing a control statement
    - GOTO, DO, END
  - Blocks containing a macro call command:
    - G65, G66, G67
    - Custom G and M codes that command a macro call

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

Macro Statements and NC Statements



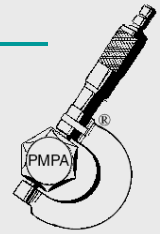
- Macro and NC Statement differences
  - When the machine is in single block mode, the machine does not stop when executing a macro
    - When parameter 6000 bit 5 (SBM) = 1 single block is effective in a macro
- Macro blocks are not regarded as blocks with no movement in cutter compensation mode (G41, G42)

Branch and Repetition Statements

- The flow of the program can be changed using branch and repetition statements.

Branch and repetition	┬	GOTO	(unconditional branch)
	├	IF	(conditional branch: if ..., then...)
	└	WHILE	(repetition while ...)

Unconditional Branch GOTO



- GOTO n causes an unconditional branch to sequence number n

Example:

GOTO 10; (causes the program to jump ahead or behind to sequence number N10)

> ⚠ **WARNING**
> Do not specify multiple blocks with the same sequence number in a single program. It is very dangerous to specify such blocks because the destination of a branch from the GOTO statement is undefined.

> **NOTE**
> 1 A backward branch takes more time as compared with a forward branch.
> 2 In the block with sequence number n, which is the branch destination of the GOTO n command, sequence number n must be located at the beginning of the block. Otherwise, the branch cannot be executed.

GOTO Statement Using Stored Sequence Numbers

When the GOTO statement is executed in a custom macro control command, a sequence number search is made for sequence numbers stored at previous execution of the corresponding blocks at a high speed. Sequence numbers stored at previous execution indicate the sequence numbers for a subprogram call and the sequence numbers that are unique in the same program of the sequence numbers at previous execution, and the CNC records these sequence numbers.
The storage type differs depending on the values of the following parameters.

(1) When bit 1 (MGO) of parameter No. 6000 is set to 1
- Fixed type: Up to 20 sequence numbers stored at execution of the corresponding blocks from the start of operation

(2) When bit 4 (HGO) of parameter No. 6000 is set to 1
- Variable type: Up to 30 sequence numbers stored at execution of the corresponding blocks before execution of the GOTO statement
- History type: Up to 10 sequence numbers stored by a sequence number search previously made using the GOTO statement

The stored sequence numbers are canceled in the following cases:
- Immediately after power-on
- After a reset
- Operation after program registration or editing (including background editing and MDI program editing)

GOTO Statement Using Stored Sequence Numbers

⚠ WARNING

Do not specify multiple blocks with the same sequence number in a single program.

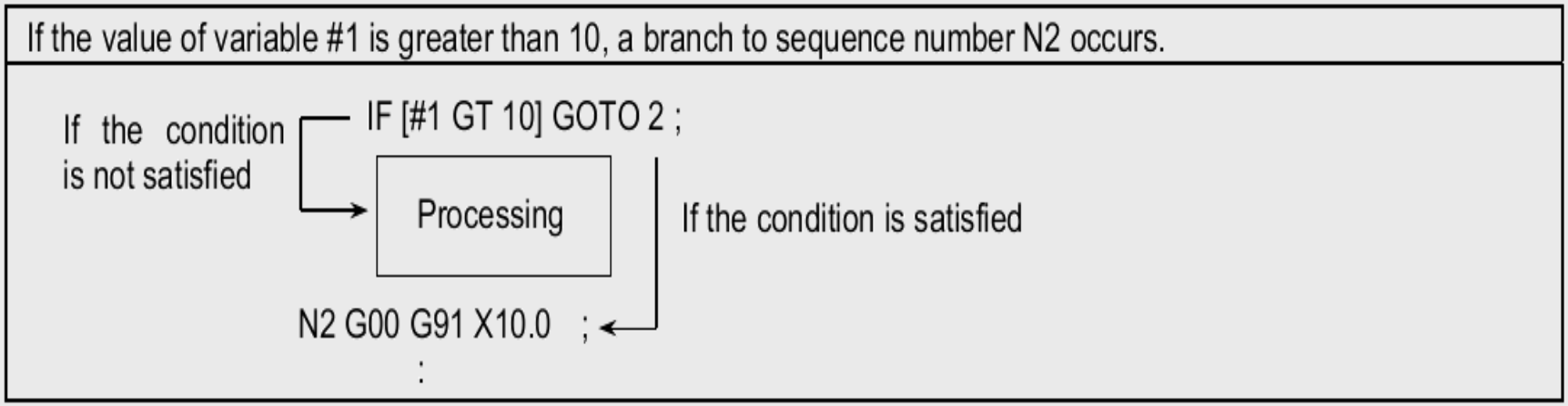
It is very dangerous to specify the sequence number of the branch destination before and after the GOTO statement and execute the GOTO statement because the branch destination changes according to the values of the parameters as shown below:

When bit 1 (MGO) or 4 (HGO) of parameter No. 6000 is set to 1	When both bits 1 (MGO) and 4 (HGO) of parameter No. 6000 are set to 0
: N10; : GOTO10; : N10; A branch to N10 before the GOTO statement occurs.	: N10; : GOTO10; : N10; A branch to N10 after the GOTO statement occurs.

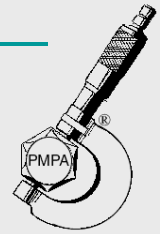
When bit 1 (MGO) or 4 (HGO) of parameter No. 6000 is set to 1 and the GOTO statement is executed, the sequence number of the branch destination may not be contained in the sequence numbers stored at previous execution of the corresponding blocks. In this case, a branch to the sequence number in a block following the GOTO statement occurs (the destination is the same as when both bits are set to 0).

Conditional Branch (IF Statement)

- IF[<conditional statement>] GOTO n
  - If the <conditional statement> is true then a branch to sequence number n occurs



Conditional Branch (IF Statement)



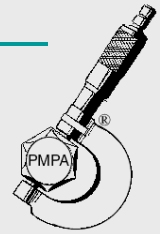
- IF[<conditional statement>] THEN
  - If the conditional expression is satisfied (true), the macro statement after THEN will be executed.
  - Only a single macro statement can be executed

If the values of #1 and #2 are the same, 0 is assigned to #3.
IF[#1 EQ #2] THEN #3=0 ;

If the values of #1 and #2 are the same and those of #3 and #4 are also the same, 0 is assigned to #5.
IF[[#1 EQ #2] AND [#3 EQ #4]] THEN #5 = 0 ;

If the values of #1 and #2 are the same or those of #3 and #4 are the same, 0 is assigned to #5.
IF[[#1 EQ #2] OR [#3 EQ #4]] THEN #5 = 0 ;

Conditional Branch (IF Statement)



Explanation

- <Conditional expression>

<Conditional expressions> are divided into <simple conditional expressions> and <complex conditional expressions>. In a <simple conditional expression>, a relational operator described in Table 14.5 (a) is specified between two variables or between a variable and constant to be compared. An <expression> can be used instead of a variable. With a <complex conditional expression>, an AND (logical AND), OR (logical OR), or XOR (exclusive OR) operation is performed for the results (true or false) of multiple <simple conditional expressions>.

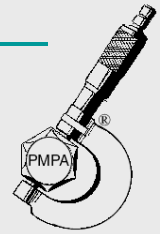
- Relational operators

Relational 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 equal sign (=) and inequality sign (>, <) cannot be used as a relational operator.

Table 14.5 (a) Relational operators

Operator	Meaning
EQ	Equal to(=)
NE	Not equal to(≠)
GT	Greater than(>)
GE	Greater than or equal to(≥)
LT	Less than(<)
LE	Less than or equal to(≤)

Sample Program

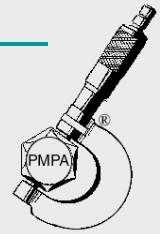


The sample program below finds the total of numbers 1 to 10.

```
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
```

#1 = 1+2+3+4+5+6+7+8+9+10

Macro Examples: (Tool Life Management)



This example counts tool use frequency and stops the machine once the tool life is expired.

Variables:

#505=Tool Life for 1st Tool

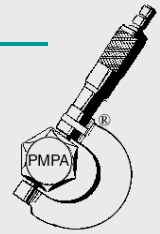
#506=1st Tool Life Counter

#507=Tool Life for 2nd Tool

#508=2nd Tool Life Counter

When a tool has reached its life expectancy the counter is set back to zero and an alarm message is generated to inform the operator which tool to change.

Macro Examples: (Tool Life Management)

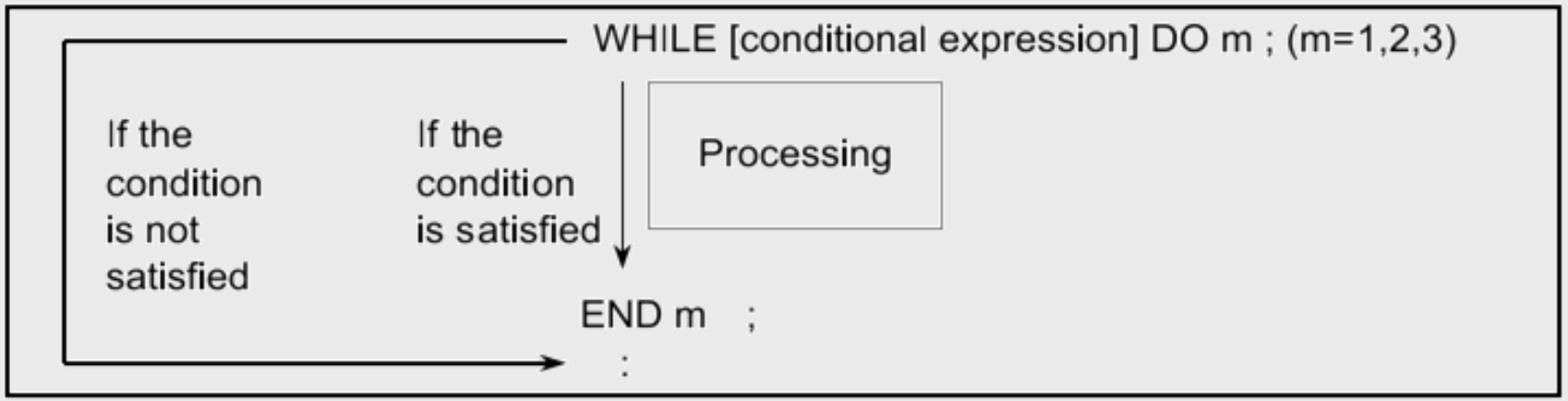



```
O1204(PROGRAM)
(VARIABLES TO SET BY OPERATOR)
(#505= 1ST TOOL LIFE AMOUNT)
(#507= 2ND TOOL LIFE AMOUNT)

(CHECK TOOL LIFE)
IF[#506GE#505]GOTO881
IF[#508GE#507]GOTO882

G00G40G18G99G20G97

T1515(1ST TOOL)
G0G99X.550Z.05M3S3000
G1G42X.520Z.015F.005
G1X.490Z0F.001
G1X-.015
G0Z-.05
G28U0
#506=[#506+1](ADD COUNT)

T1616 (2ND TOOL)
G0G99X0Z-.05M3S3000
G1Z.1F.002
G0Z-.05T0
G28U0
#508=[#508+1](ADD COUNT)

(RESET COUNT AND ALARMS)
GOTO999(NO TOOLS EXPIRED)

N881(1ST TOOL ALARM)
#506=0
#3000=1(T1515 IS EXPIRED)

N882(2ND TOOL ALARM)
#508=0
#3000=2(T1616 IS EXPIRED)

N999
M30
```

TSUGAMI
Rem Sales, LLC

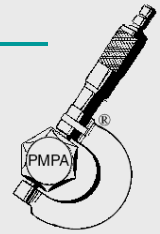
Repetition (WHILE Statement)

- A conditional statement is specified after WHILE. As long as the conditional statement is satisfied (true) the program from DO to END is executed.



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, an alarm PS0126 occurs.

Repetition (WHILE Statement)



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
```

Takes longer to search backward

Example using WHILE_DO to do the same thing:

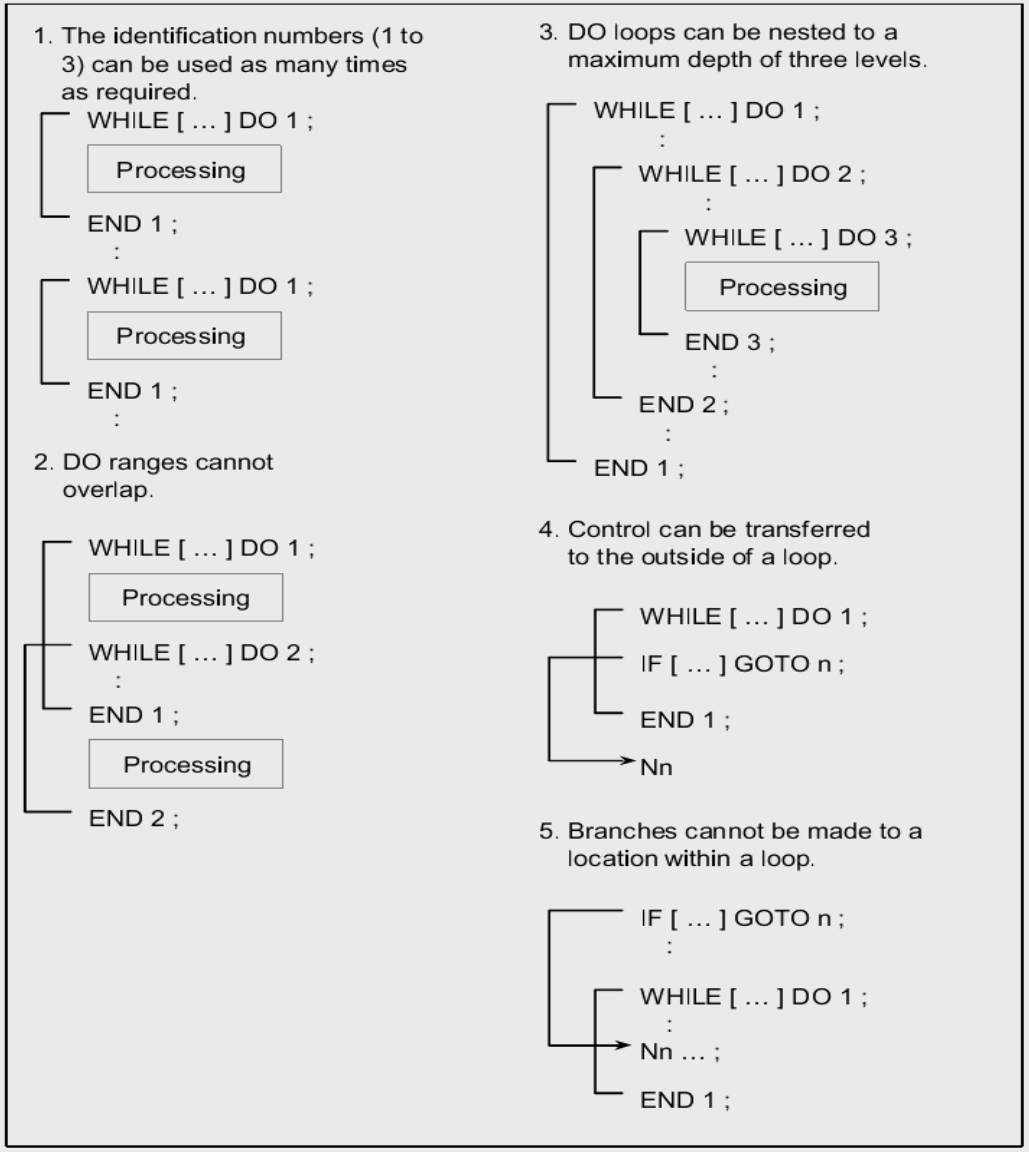
```
O0001;
#1=0;
#2=1;
WHILE[#2 LE 10] DO 1;
#1=#1+#2;
#2=#2+1;
END 1;
M30;
```

Processes faster

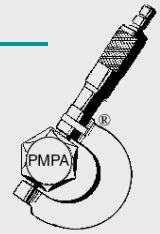
TSUGAMI
Rem Sales, LLC

Repetition (WHILE Statement)

- Nesting
  - The identification numbers (1 to 3) in a DO_END loop can be used as many times in a program as desired (reuse).
  - When a program contains overlapping DO ranges alarm PS0124 is generated.



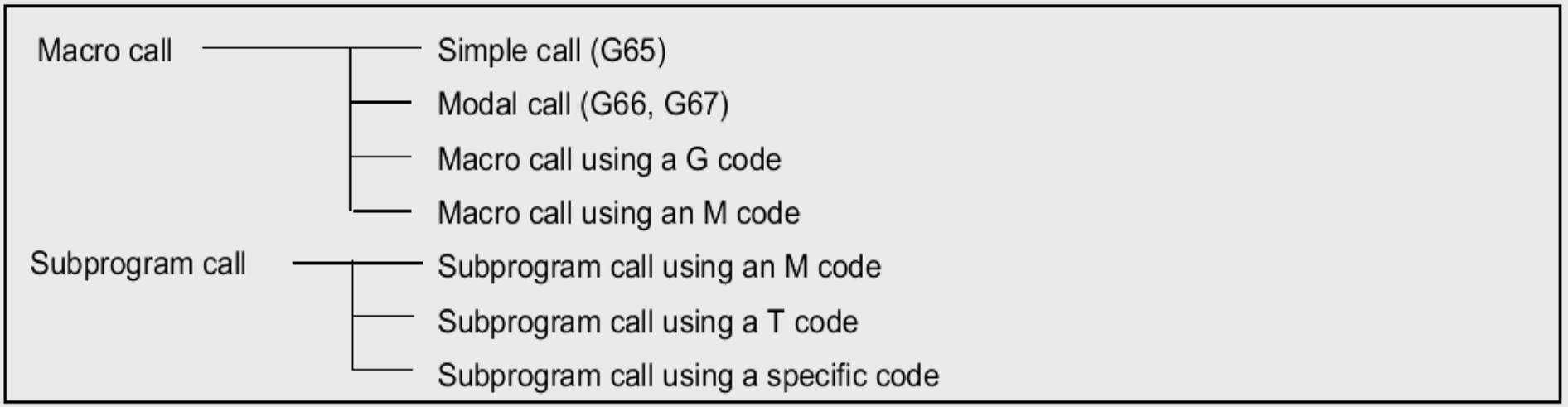
Repetition (WHILE Statement)



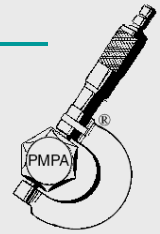
- Limitations
  - Infinite loops – When DO m is specified without a WHILE statement preceding, an infinite loop is created between the DO and END commands.
  - Processing time – When a branch to the sequence number specified in a GOTO statement occurs, the sequence number is searched for. In the case of processing in the reverse direction use a While statement to reduce processing time.
  - Null variable – In conditional expressions that use EQ and NE, a null variable is not treated the same as 0. In all other conditional expressions null = 0

Macro Call

- A macro program can be called using any of the following methods.
- There are two types of calling methods: macro call and subprogram call



Macro Call



Limitation

- Call nesting

Macro calls can be nested to a depth of up to five levels and subprogram calls can be nested to a depth of up to ten levels; calls can be nested to a depth of up to 15 levels in total.

- Differences between macro calls and subprogram calls

Macro call (G65, G66, Ggg, or Mmm) differs from subprogram call (M98, Mmm, or Ttt) as described below.

- With a macro call, an argument (data passed to a macro) can be specified. A subprogram call does not have this capability.
- If a macro call block contains another NC command (such as G01 X100.0 G65 Pp), an alarm PS0127 occurs.
- If a subprogram call block contains another NC command (such as G01 X100.0 M98 Pp), the subprogram is called after the command is executed.
- In any macro call block, the machine does not stop in the single block mode.
  If a subprogram call block contains another NC command (such as G01 X100.0 M98 Pp), the machine stops in the single block mode.
- With a macro call, the level of local variables changes. With a subprogram call, the level of local variables does not change. (See "Local variable levels" in Limitation of Subsection 14.6.1.)

Simple Macro Call (G65)

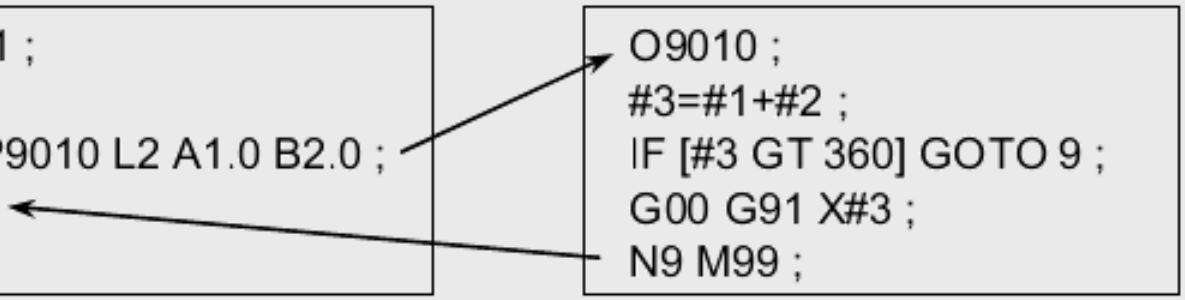
- When G65 is specified, the custom macro sub program at address P is called. Data (arguments) can be passed to the macro program.

G65 P p L ℓ <argument-specification> ;

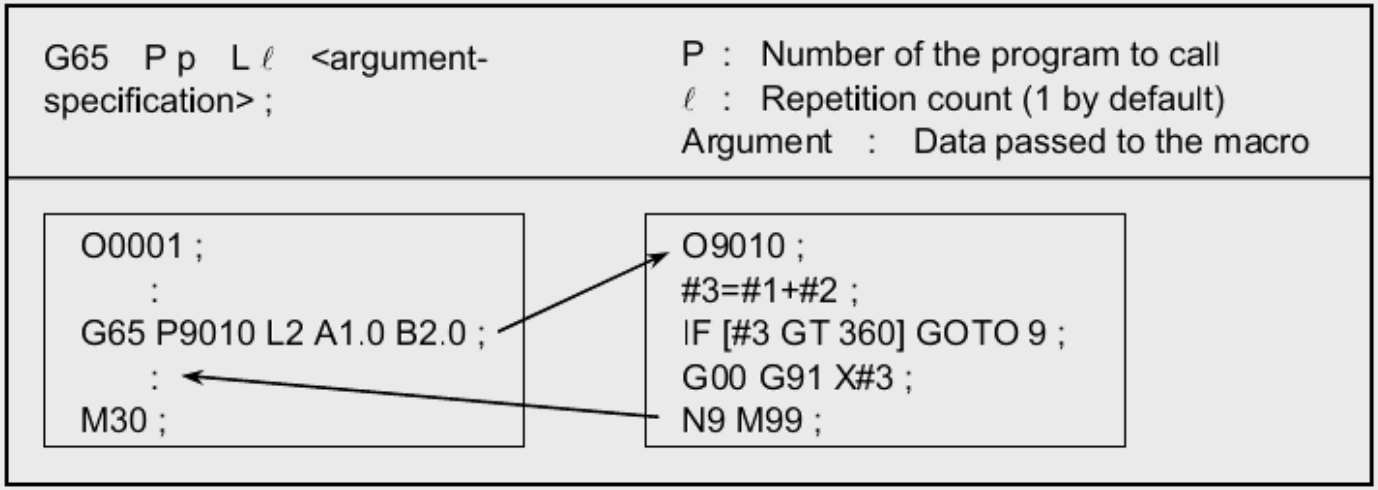
P : Number of the program to call
ℓ : Repetition count (1 by default)
Argument : Data passed to the macro

```
O0001 ;
      :
G65 P9010 L2 A1.0 B2.0 ;
      :
M30 ;
```

```
O9010 ;
#3=#1+#2 ;
IF [#3 GT 360] GOTO 9 ;
G00 G91 X#3 ;
N9 M99 ;
```



Simple Macro Call (G65)



- After G65, address P is the program number of the macro program.
- When a number of repetitions are required, specify the number after address L. Range is 1-999999999. When L is omitted the value of L is assumed to be 1.
- By specifying arguments (A n, B n, Cn) values are passed to the corresponding variable.

Argument Specification I

Two types of argument specification are available. Argument specification I uses letters other than G, L, O, N, and P once each. Argument specification II 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.

- Argument specification I

Address	Variable number
A	#1
B	#2
C	#3
D	#7
E	#8
F	#9
H	#11

Address	Variable number
I	#4
J	#5
K	#6
M	#13
Q	#17
R	#18
S	#19

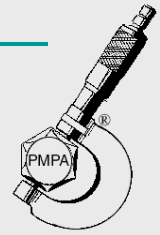
Address	Variable number
T	#20
U	#21
V	#22
W	#23
X	#24
Y	#25
Z	#26

- 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.
  Argument specification I is always used for I, J, and K by setting bit 7 (IJK) of parameter No. 6008 to 1

Example

- When bit 7 (IJK) of parameter No. 6008 is 0, I_ J_ K_ means that I = #4, J = #5, and K = #6 while K_ J_ I_ means K = #6, J = #8, and I= #10 because argument specification II is used.
- When bit 7 (IJK) of parameter No. 6008 is 1, K_ J_ I_ means that I = #4, J = #5, and K = #6, which is the same as with I_ J_ K_, because argument specification I is used.

Argument Specification II



Argument specification II uses A, B, and C once each and uses I, J, and K up to ten times. Argument specification II is used to pass values such as three-dimensional coordinates as arguments.

Address	Variable number
A	#1
B	#2
C	#3
I_1	#4
J_1	#5
K_1	#6
I_2	#7
J_2	#8
K_2	#9
I_3	#10
J_3	#11

Address	Variable number
IK_3	#12
I_4	#13
J_4	#14
K_4	#15
I_5	#16
J_5	#17
K_5	#18
I_6	#19
J_6	#20
K_6	#21
I_7	#22

Address	Variable number
J_7	#23
K_7	#24
I_8	#25
J_8	#26
K_8	#27
I_9	#28
J_9	#29
K_9	#30
I_{10}	#31
J_{10}	#32
K_{10}	#33

- Subscripts of I, J, and K for indicating the order of argument specification are not written in the actual program.

NOTE
When bit 7 (IJK) of parameter No. 6008 is 1, argument specification II cannot be used.

Simple Macro Call (G65)

Limitation

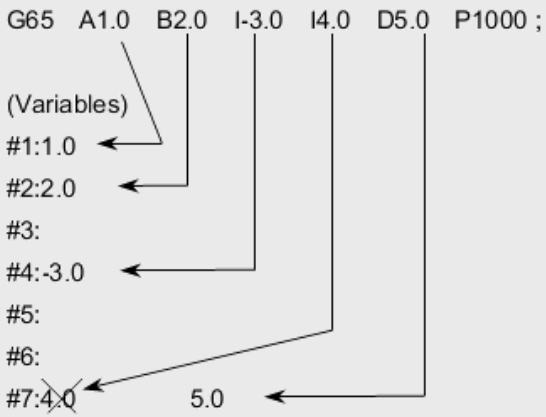
- Format

G65 must be specified before any argument.

- Mixture of argument specifications I and II

The CNC internally identifies argument specification I and argument specification II. If a mixture of argument specification I and argument specification II is specified, the type of argument specification specified later takes precedence.

[Example]



When both the I4.0 and D5.0 arguments are commanded for variable #7 in this example, the latter, D5.0, is valid.

- Position of the decimal point

The units used for argument data passed without a decimal point correspond to the least input increment of each address.

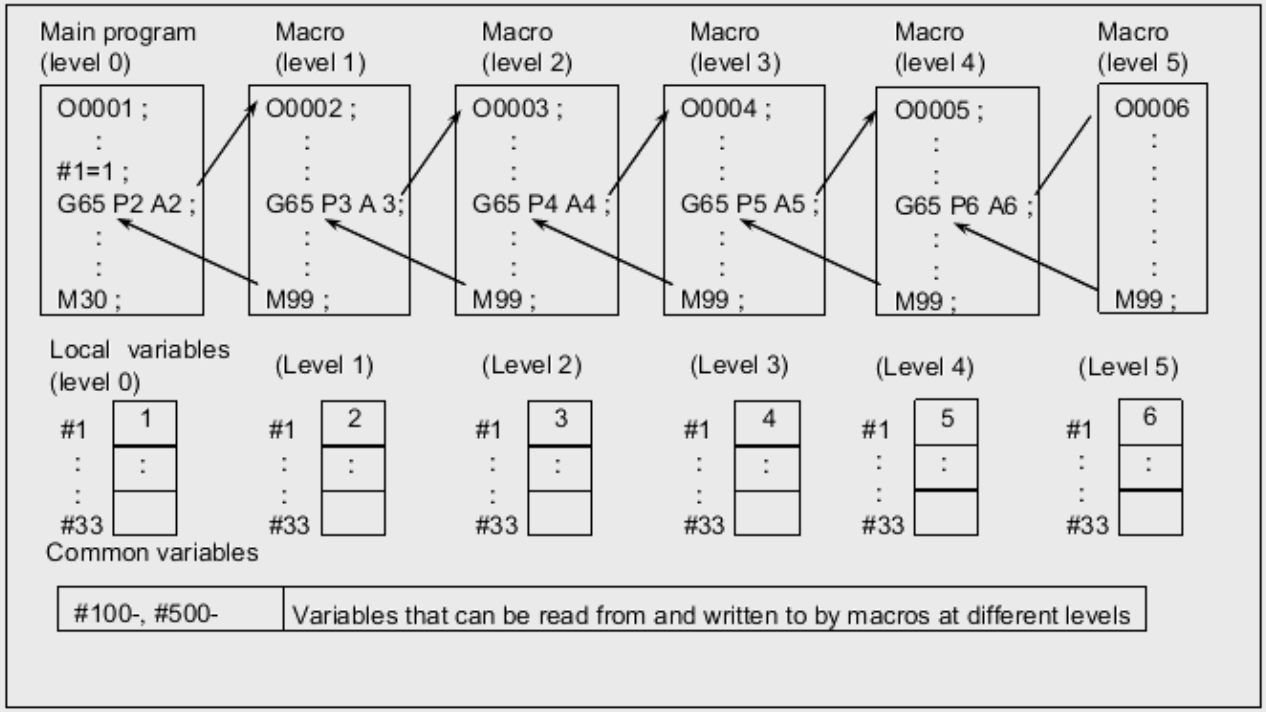
> ⚠ **CAUTION**
> The value of an argument passed without a decimal point may vary according to the system configuration of the machine. It is good practice to use decimal points in macro call arguments to maintain program compatibility.

Macro Call Nesting

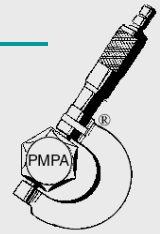
Macro calls can be nested to a depth of up to five levels including simple calls (G65) and modal calls (G66). Subprogram calls can be nested to a depth of up to 15 levels including macro calls.
A macro program can also be called during MDI operation in the same way.

- **Local variable levels**

- Local variables from level 0 to 5 are provided for nesting.
- The level of the main program is 0.
- Each time a macro is called (with G65, G66, Ggg, or Mmm), 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.



Macro Examples: Macro G & M Code Sub Program Calls



•Macro sub-programs can also be called with G and M codes by setting parameters.

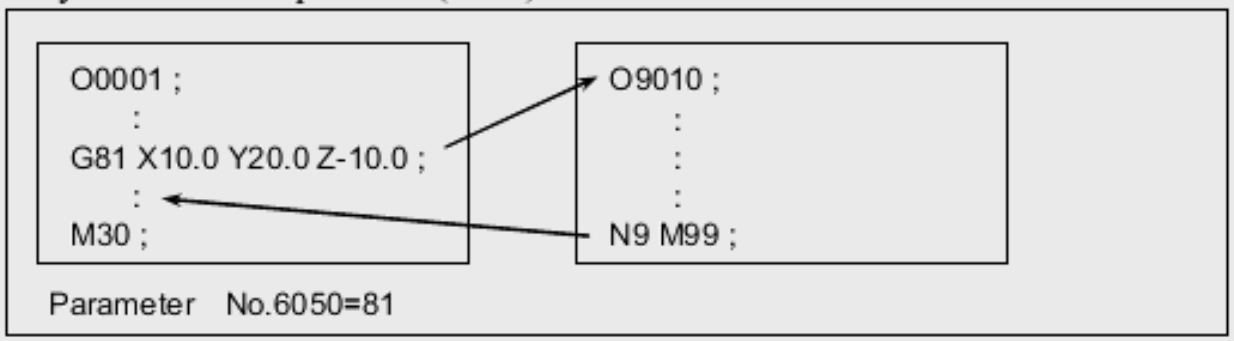
•Any value can be set to these parameters as long as it is not an existing G or M code.

•Parameters 6050-6059 correspond with program numbers 9010-9019 for macro call by G-code.

•Parameters 6080-6089 correspond with program numbers 9020-9029 for macro call by M-code.

Using a 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).

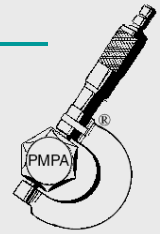


Explanation

By setting a G code number from -9999 to 9999 used to call a custom macro program (O9010 to O9019) in the corresponding parameter (No.6050 to No.6059), the macro program can be called in the same way as with G65.

If a negative G code is set, a modal call (equivalent to G66) is made.
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.

Using a G-Code



- **Correspondence between parameter numbers and program numbers**

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

- **Repetition**

As with a simple call, a number of repetitions from 1 to 99999999 can be specified at address L.

- **Argument specification**

As with a simple call, two types of argument specification are available: Argument specification I and argument specification II. The type of argument specification is determined automatically according to the addresses used.

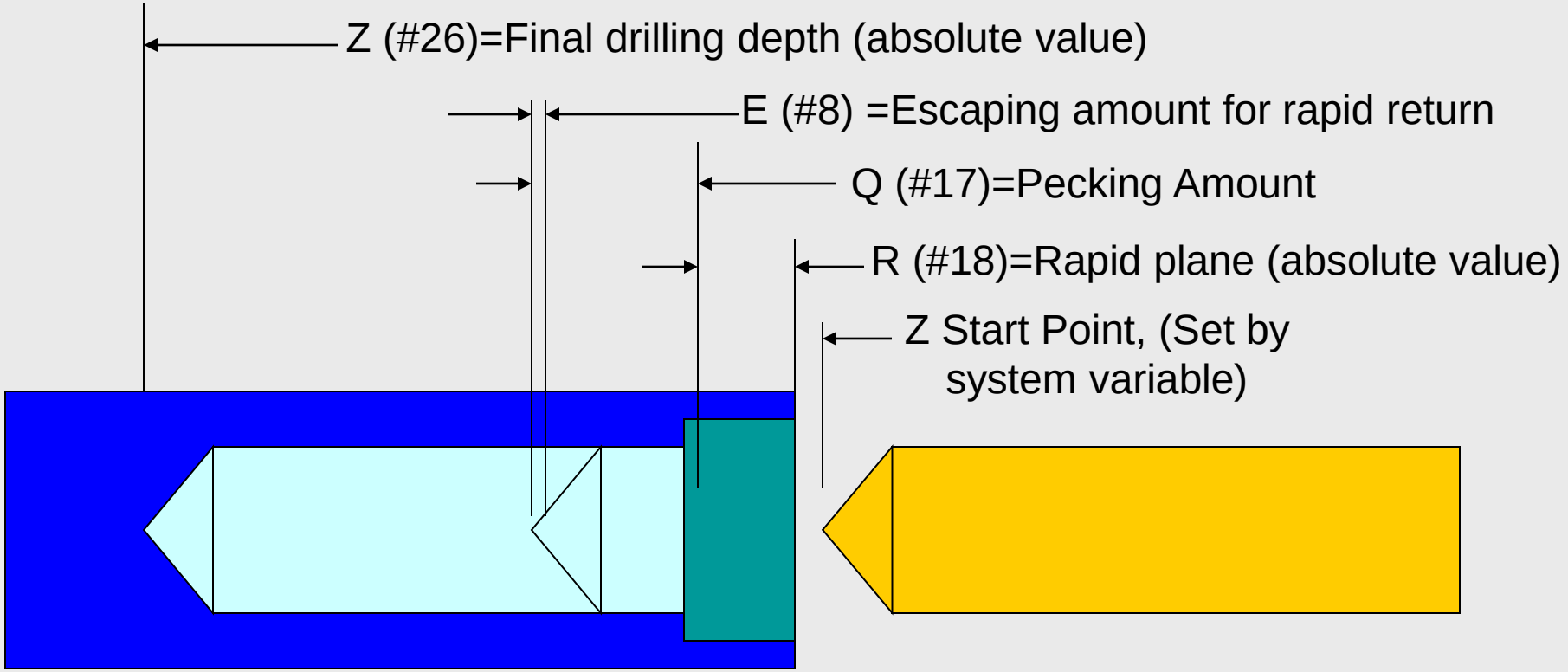
Limitation

- **Nesting of calls using G codes**

- To call another program in a program called using a G code, only G65, M98, or G66 can be used normally.
- When bit 6 (GMP) of parameter No. 6008 is set to 1, a call using an M code, T code, or specific code can be performed in a program called using a G code.

Macro Examples: Custom G-Code Macro

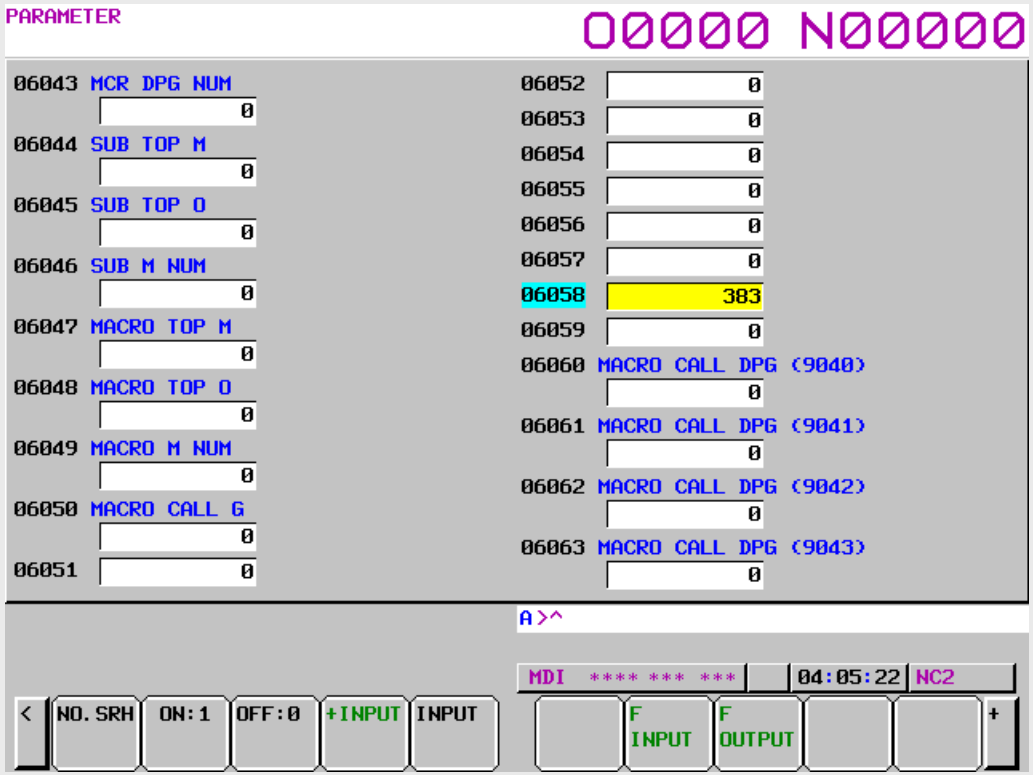
- Start with a desired application and decide how much control you need.
- List out your arguments.
- Document it carefully once it is proven.
- Name it: Set parameter 6058=383, G383 now calls program O9018.



TSUGAMI
Rem Sales, LLC

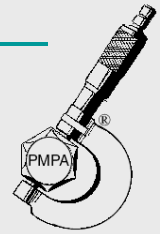
Macro Examples: Custom G-Code Macro

- Parameter Screen set for G383 Macro:



TSUGAMI
Rem Sales, LLC

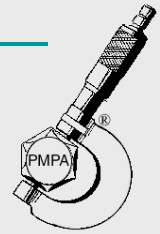
Macro Examples: Custom G-Code Macro



Macro Arguments:
R (#18)= Rapid plane start point. Absolute Z value
Z (#26)= Final drill depth.
Q (#17)= 1st drilling peck depth
E (#8)= Escaping amount for pecking return
T (#20)= Dwell time at last peck
F (#9)= Drilling feed rate
A (#1)= Peck reduction percentage
B (#2)= Minimum pecking amount

Program notes:
*Z axis returns to start point after each peck.
*The R rapid plane is an absolute Z-location.
*A= peck reduction by percentage. Example .2=
20% peck reduction each time.
*B= Minimum pecking amount allowed.
*If A and B are omitted the pecking amount is the “Q” value throughout the cycle.
*Select G98 or G99 mode prior to G383.

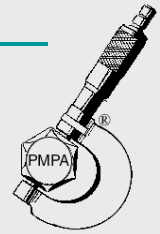
Macro Examples: Custom G-Code Macro




```
%
O9018(G383 PECK DRILL CYCLE)

(BEGINNING CALCULATIONS)
#110=#5002(SET START Z)
#112=[1.0-#1](PECK MULTIPLIER)
#113=#17(SET 1ST PECK)

(DECIDE Z- OR Z+)
IF[#5002GT#26]GOTO200(Z- DIRECTION)

(**********)
(Z+ DIRECTION)
#111=[#18+#8](SET DRILL START)

IF[#17EQ0]GOTO100(NO PECKING)

(WHOLE PECKS)
WHILE[[#111+#113]LT#26]DO1
G0Z[#111-#8](RAPID IN)
G1Z[#111+#113]F#9(DRILLING PECK)
#111=#5002(READ DRILL END Z)
G0Z#110(GO BACK TO START POS.)
#113=[#113*#112](REDUCE PECK)

IF[#113LT#2]THEN#113=#2(PECK CLAMP)
END1

N100(FINAL DEPTH)
G0Z[#111-#8]
G1Z#26F#9
G4U#20
G0Z#110
GOTO300(GO TO END)
```

TSUGAMI
Rem Sales, LLC

Macro Examples: Custom G-Code Macro

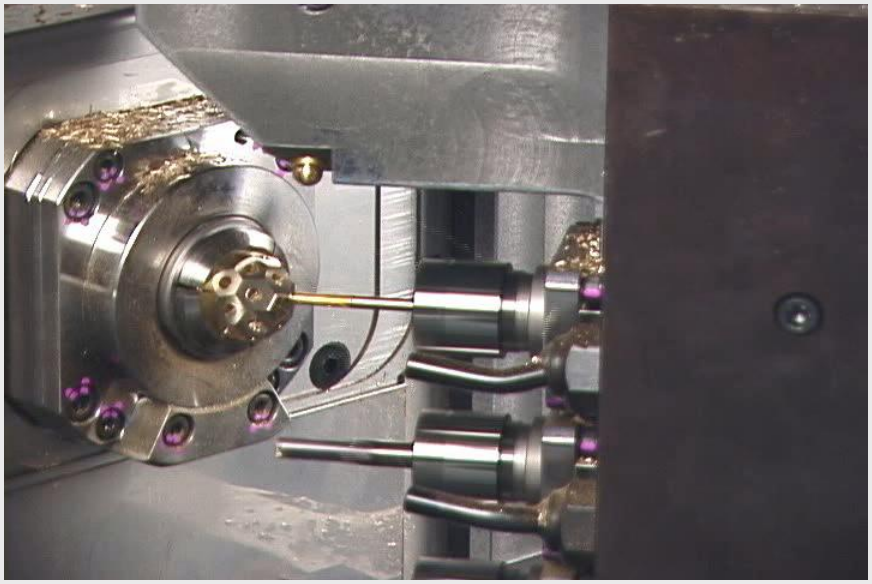



```
N200(Z- DIRECTION)
#111=[#18-#8](SET DRILL START)

IF[#17EQ0]GOTO201(NO PECKING)

(WHOLE PECKS)
WHILE[[#111-#113]GT#26]DO1
G0Z[#111+#8](RAPID IN)
G1Z[#111-#113]F#9(DRILLING PECK)
#111=#5002(READ DRILL END Z)
G0Z#110(GO BACK TO START POS.)

#113=[#113*#112](REDUCE PECK)

IF[#113LT#2]THEN#113=#2(PECK CLAMP)
END1
```

```
N201(FINAL DEPTH)
G0Z[#111+#8]
G1Z#26F#9
G4U#20
G0Z#110


N300(END)
M99
%
```

TSUGAMI
Rem Sales, LLC

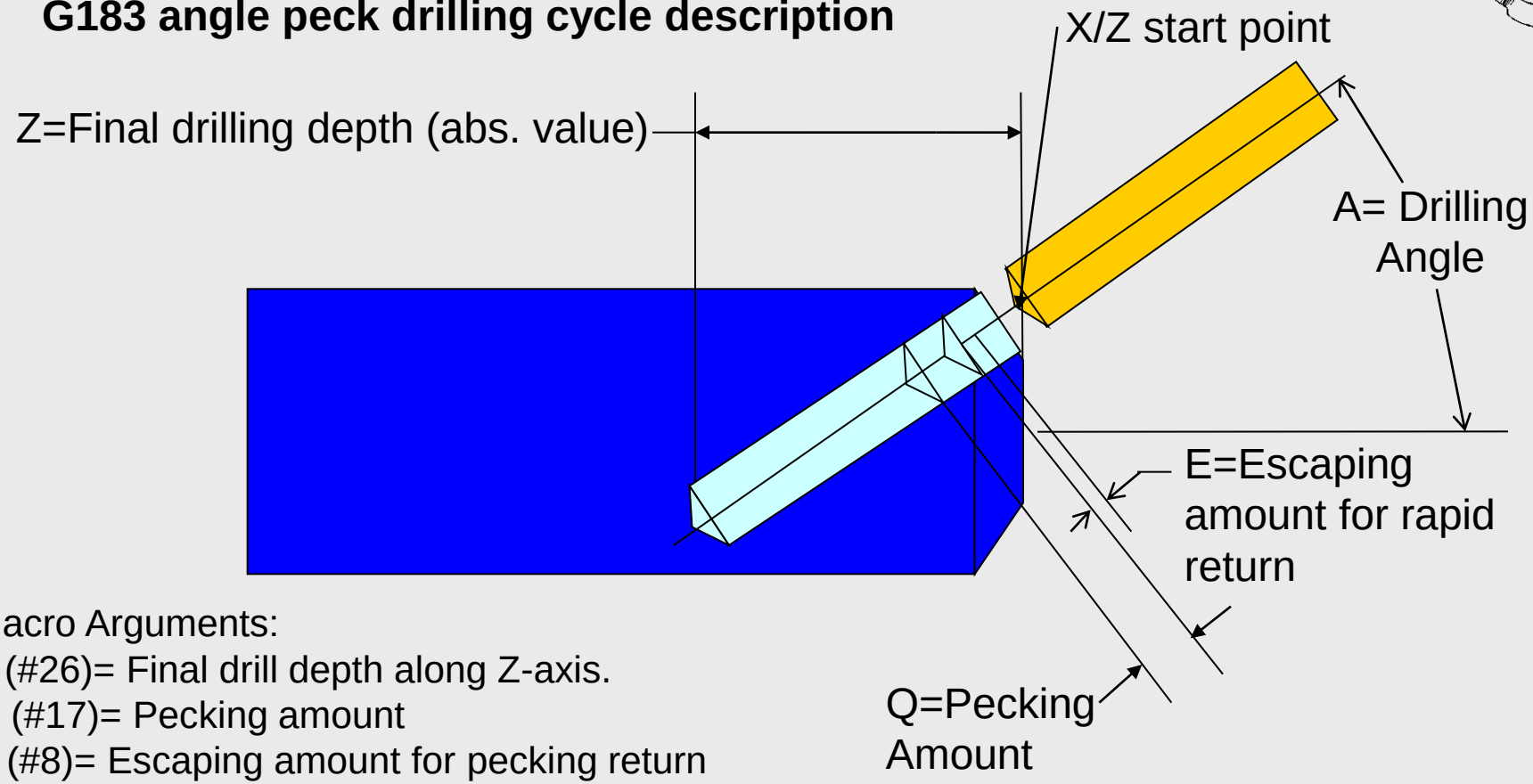
Example Custom Angle Drilling Cycle For a Live Tool Lathe



TSUGAMI
Rem Sales, LLC

Macro Examples: Custom G-Code Macro

G183 angle peck drilling cycle description

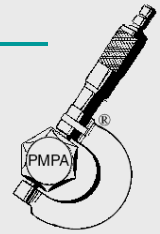


Macro Arguments:
Z (#26)= Final drill depth along Z-axis.
Q (#17)= Pecking amount
E (#8)= Escaping amount for pecking return
T (#20)= Dwell time at last peck
F (#9)= Drilling feed rate
A (#1)= Angle for drilling
R (#18)= Retract feed rate

Program notes:
*X and Z axes return to start point after each peck.
*If “Q” value is set to 0 then no pecking is done.

TSUGAMI
Rem Sales, LLC

Macro Examples: Custom G-Code Macro



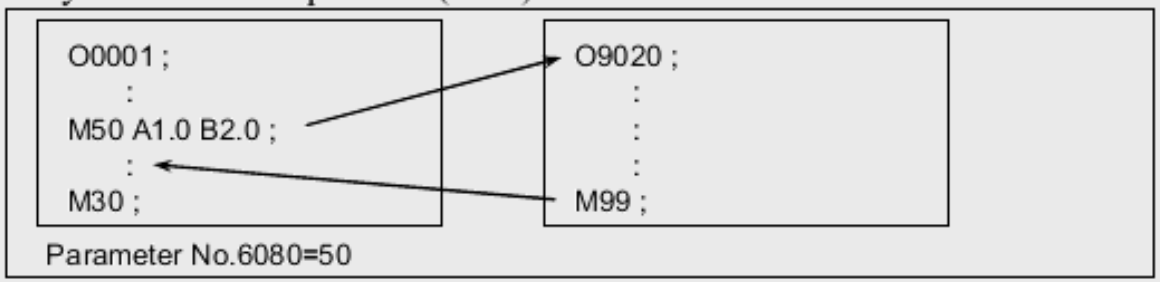
G183 angle peck drilling cycle description

```
%
O9019(G183 ANGLED PECKING)
(Z+ DIRECTION DRILLING)
(SET PARAMETER 6059=183)
(G283A,Z,Q,E,F,R,T)
(Z#26=ZENDPOINT FOR DRILLING)
(A#1=ANGLE FOR DRILLING)
(Q#17=PECK AMOUNT)
(E#8=ESCAPING AMOUNT)
(F#9=CUTTING FEED RATE)
(T#20=LAST PECK DWELL)
(R#18=RETRACT FEEDRATE)
(IF Q=0 THEN NO PECKING)

#108=#5004(START Y)
#109=#5002(START Z)
#110=#5004(Y ESCAPE)
#111=#5002(Z ESCAPE)
#112=#5004(Y DRILL END)
#113=#5002(Z DRILL END)

#115=[#26-#5002](Z-LEG)
#116=[#115/COS[#1]](HYP. DISTANCE)

IF[#17EQ0]GOTO300(NO PECKING)

#117=[FIX[#116/#17]](WHOLE PECKS)
#107=0(COUNTER)
```

```
(WHOLE PECKS)
WHILE[#117GT#107]DO1
G98G1Y#110Z#111F#18(GO TO ESCAPING POS.)
G1Y#112Z#113F#9(FEED TO END)
G1V-[[#17]*SIN[#1]]W[[#17]*COS[#1]]F#9(DRILLING PECK)
#112=#5004(READ DRILL END Y)
#113=#5002(READ DRILL END Z)
G1V[#8*SIN[#1]]W-[#8*COS[#1]]F#9(ESCAPING AMOUNT)
#110=#5004(READ ESCAPE Y)
#111=#5002(READ ESCAPE Z)
G1Y#108Z#109F#18(GO BACK TO START POS.)
#107=[#107+1](ADD COUNT)
END1

N300(FINAL DEPTH)
G98G1Y#110Z#111F#18
G1Y#112Z#113F#9(FEED TO END)
G1V-[[#26-#113]*TAN[#1]]Z#26F#9
G4U#20
G1Y#108Z#109F#18
M99
%
```

Macro Call Using an 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 with a simple call (G65).



Explanation

By setting an M code number from 3 to 99999999 used to call custom macro program O9020 to O9029 in the corresponding parameter (Nos. 6080 to 6089), the macro program can be called in the same way as with G65.

Macro Call Using an M-Code

- **Correspondence between parameter numbers and program numbers**

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

Example)

When parameter No. 6080 is set to 990, O9020 is called using M990.

- **Repetition**

As with a simple call, a number of repetitions from 1 to 99999999 can be specified at address L.

- **Argument specification**

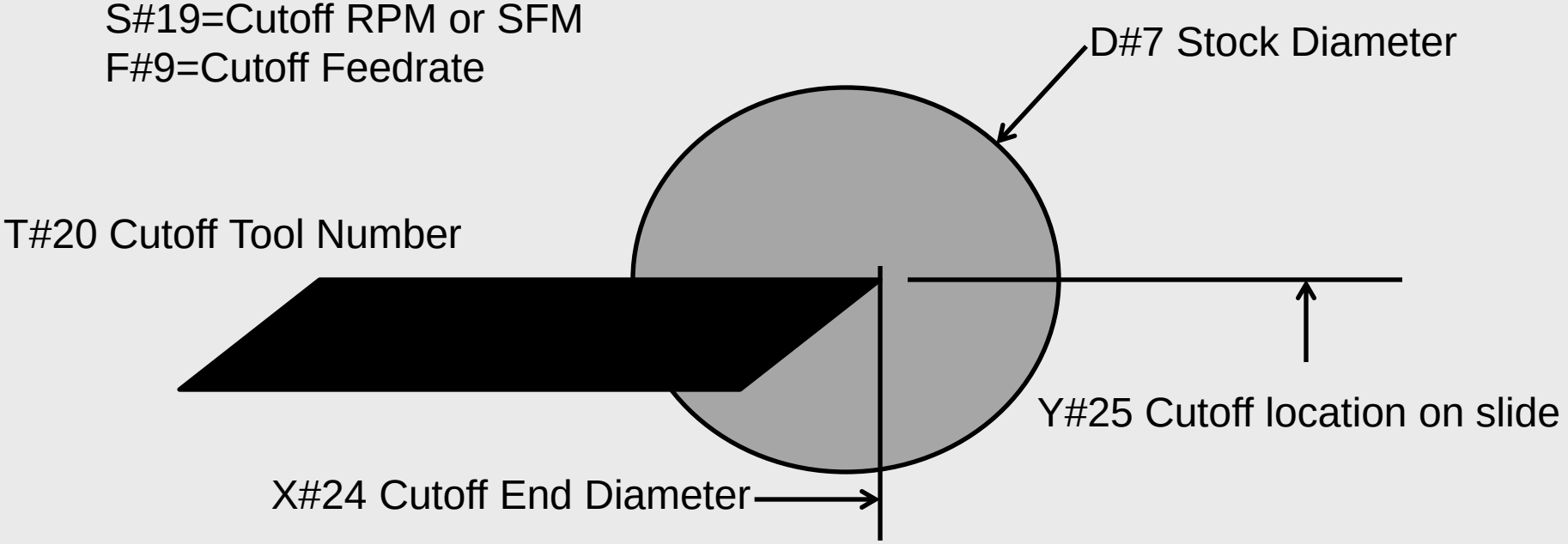
As with a simple call, two types of argument specification are available: Argument specification I and argument specification II. The type of argument specification is determined automatically according to the addresses used.

Macro Examples: Custom M-Code Macro

Automatic cut-off macro;
Set parameter 6080 to 910. M910 call program O9020.

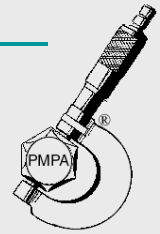
Notes: Cutoff location and offset must match
Example T7=T0707

C#3=G96 or G97 mode
I#4=G68 or G69 status
S#19=Cutoff RPM or SFM
F#9=Cutoff Feedrate



TSUGAMI
Rem Sales, LLC

Macro Examples: Custom M-Code Macro



Automatic Cutoff Macro

M-Code call: M910 I69 T7 D1.0 X-.04 C96 S400 F.002

```
O9020(AUTO CUT-OFF)
G50S3000(SPINDLE SPEED LIMIT)
(ROUTINE START)
G#4(G68 OR G69)
T#20(OFFSET CALL)

(CHECK X-AXIS ABS. POSITION)
IF[#5001GT0]GOTO10
(CHECK Y-AXIS MACH. POSITION)
IF[#5024LT[#25-.02]]GOTO10
IF[#5024GT[#25+.02]]GOTO10

M99
N10(CUTOFF)

#3006=1(CHECK BEFORE CUTOFF)
M8
G28U0
M10
M60
T[[#20*100]+#20]
G#3M3S#19

G0X[#7+.05]
G99G1X#24F#9
M9
M5

#3006=1(CUTOFF FINISH)

M99
%
```

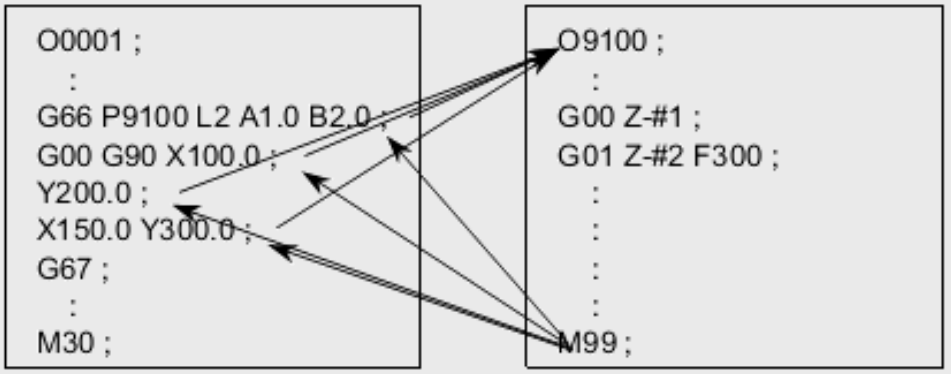
TSUGAMI
Rem Sales, LLC

Modal Macro Call

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 P p L ℓ <argument-specification> ;

P : Number of the program to call
ℓ : Repetition count (1 by default)
Argument : Data passed to the macro



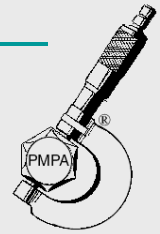
- **Call**

- 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 999999999 can be specified at address L.
- As with a simple call (G65), data passed to a macro program is specified in arguments.
- In the G66 mode, a macro can be called.

- **Cancellation**

When a G67 code is specified, modal macro calls are no longer performed in subsequent blocks.

Modal Macro Call



- **Call nesting**

Macro calls can be nested to a depth of up to five levels including simple calls (G65) and modal calls (G66). Subprogram calls can be nested to a depth of up to 15 levels including macro calls.

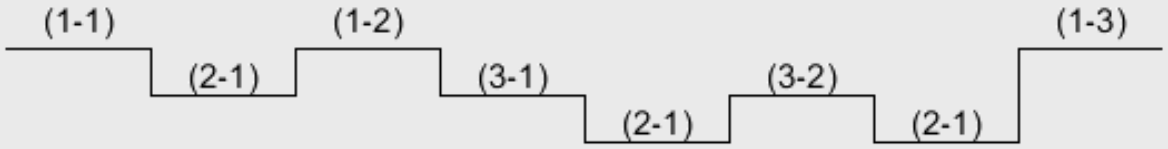
- **Modal call nesting**

For a single modal call (when G66 is specified only once), each time the move command is executed, the specified macro is called. When nested modal macro calls are specified, the macro at the next higher level is called each time the move command for a macro call is executed.
Macros are called in reverse order in which they are specified. Each time G67 is issued, the macros are canceled one by one in reverse order in which they are specified.

[Example]

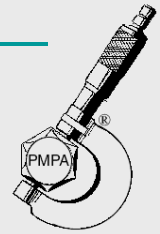
G66 P9100 ;		O9100 ;		O9200 ;	
X10.0 ;	(1-1)	Z50.0 ;	(2-1)	X60.0 ;	(3-1)
G66 P9200 ;		M99 ;		Y70.0 ;	(3-2)
X15.0 ;	(1-2)			M99;	
G67 ;	Cancels P9200.				
G67 ;	Cancels P9100.				
X-25.0 ;	(1-3)				

Execution order of the above program (blocks not containing the move command omitted)



* No modal call is performed after (1-3) because the mode is not the macro call mode.

Modal Macro Call



Limitation

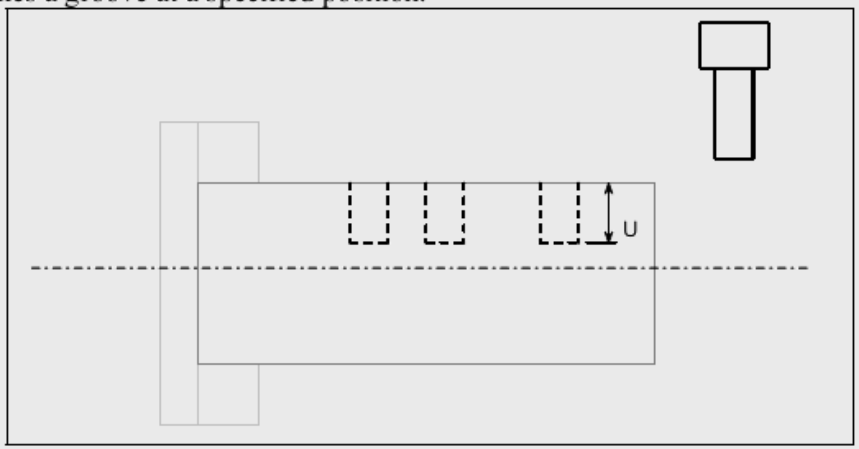
- G66 and G67 blocks are specified in pairs in the same program. If a G67 code is specified not in the G66 mode, an alarm PS1100 occurs. Bit 0 (G67) of parameter No. 6000 can be set to 1 to specify that the alarm does not occur in this case.
- In a G66 block, no macros can be called. Local variables (arguments) are set, however.
- G66 needs to be specified before any arguments.
- No macros can be called in a block which contains a code such as a auxiliary function that does not involve movement along an axis.
- Local variables (arguments) can only be set in G66 blocks. Note that local variables are not set each time a modal call is performed.

NOTE
If M99 is specified in a block in which a call is performed, it is executed after the call is performed.

Sample program

T

This program makes a groove at a specified position.



- **Calling format**

G66 P9110 Uu Ff ;

U : Groove depth (incremental programming)
F : Cutting feed of grooving

- **Program that calls a macro program**

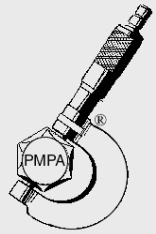
```
O0003 ;
G50 X100.0 Z200.0 ;
S1000 M03 ;
G66 P9110 U5.0 F0.5 ;
G00 X60.0 Z80.0 ;
Z50.0 ;
Z30.0 ;
G67 ;
G00 X00.0 Z200.0 M05 ;
M30 ;
```

- **Macro program (program called)**

```
O9110 ;
G01 U-#21 F#9 ;..... Cuts the workpiece.
G00 U#21 ; ..... Retracts the tool.
M99 ;
```

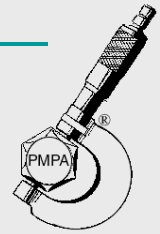
TSUGAMI
Rem Sales, LLC

TSUGAMI
Rem Sales, LLC



Macro Program Examples

Macro Examples: (Tool Life Frequency with Tool Change)



This example uses two tools of the same type. When the 1st tool has expired the machine starts using the 2nd tool.

Variables:

#510=Tool in Use (Machine Decision)

#511=Tool Life Expectancy (Operator Set)

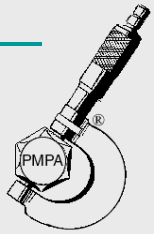
#512=1st Tool Life Counter

#513=2nd Tool Life Counter

Once both tool counters have expired the counters will reset and an alarm will stop the machine with a tool change message.

TSUGAMI
Rem Sales, LLC

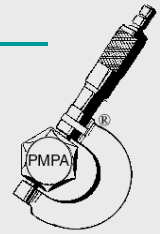
Macro Examples: (Tool Life with Tool Change)




```
O1204(PROGRAM)
(VARIABLES TO SET BY OPERATOR)
(#511= TOOL LIFE AMOUNT)
(TOOL LIFE 1ST GROUP)
IF[#512GE#511]GOTO30
#510=1616
GOTO100
N30(2ND TOOL)
IF[#513GE#511]GOTO881
#510=1717

N100(MAIN PROGRAM)
G00G40G18G99G20G97
T#510(TOOL CALL )
G0G99X.550Z.05M3S3000
G1G42X.520Z.015F.005
G1X.490Z0F.001
G1X-.015
G0Z-.05
G28U0
```

```
(1ST TOOL)
IF[#512GE#511]GOTO50
#512=[#512+1] (ADD COUNT)
GOTO60
N50(2ND TOOL)
#513=[#513+1] (ADD COUNT)
N60


(RESET COUNT AND ALARMS)
GOTO999(NO TOOLS EXPIRED)
N881(1ST GROUP ALARM)
#512=0(RESET 1ST COUNT)
#513=0(RESET 2ND COUNT)
#3000=1(GROUP 1 EXPIRED)


N999(PROGRAM END)
M30
```

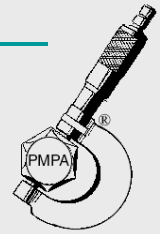
TSUGAMI
Rem Sales, LLC



TSUGAMI
Rem Sales, LLC

Macro Examples: (Tool Life by Time Macro)

Supplied by Caron Engineering



This macro requires the use of sub-program calls by M-code. This feature is set up in the custom macro parameters.
Parameter 6071-6079 correspond with programs O9001-O9009.

1) Command M201 after every T-code command (O9001 program)

2) At the end of the cutting sequence (before any T0) command M202 (O9002 program)

This program will accumulate time to the commanded T-code.

3) At the beginning and end of your main program command M203 (O9003 program)

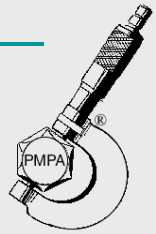
This program will determine if any tools have expired.

4) If no preset value is set the tool life will not be measured.

TSUGAMI
Rem Sales, LLC

Macro Examples: (Tool Life by Time Macro)

Supplied by Caron Engineering



Macro Variables #501-#524 are used to preset and accumulate cutting time for up to 12 tools,

Life expectancy is set and measured in hours.

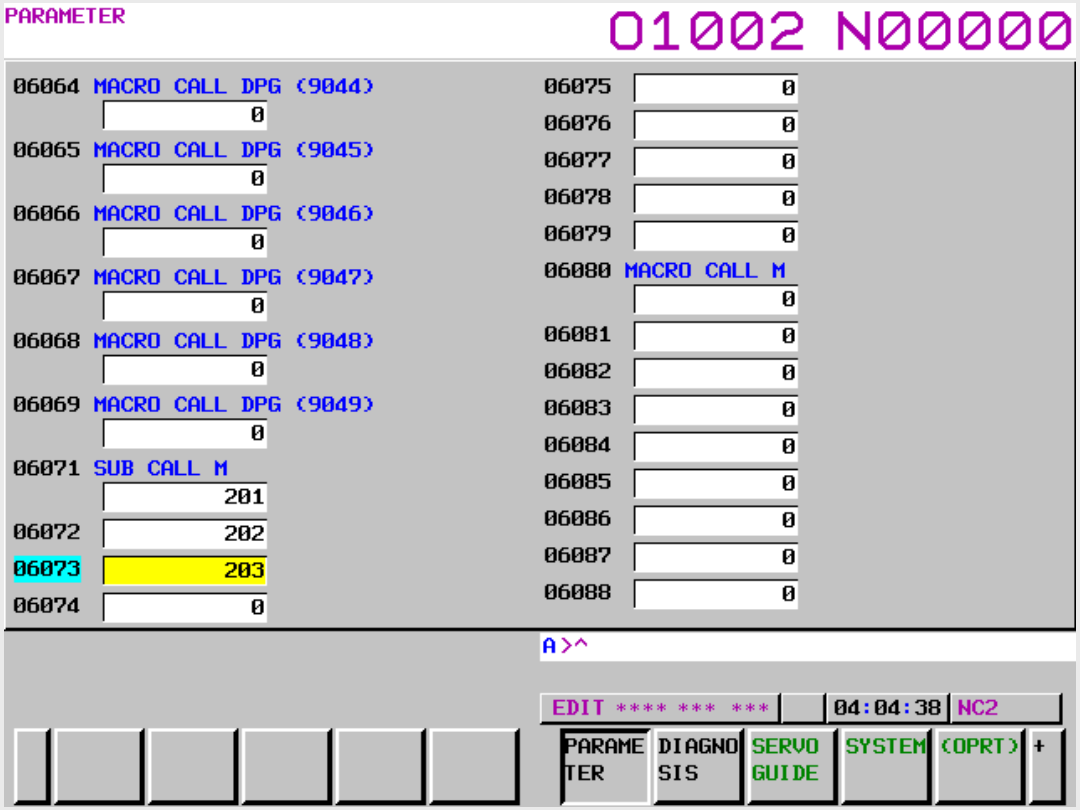
Tool 1	Preset #501	Accumulator #502
Tool 2	#503	#504
Tool 3	#505	#506
Tool 4	#507	#508
Tool 5	#509	#510
Tool 6	#511	#512
Tool 7	#513	#514
Tool 8	#515	#516
Tool 9	#517	#518
Tool 10	#519	#520
Tool 11	#521	#522
Tool 12	#523	#524

TSUGAMI
Rem Sales, LLC

Macro Examples: (Tool Life by Time Macro)

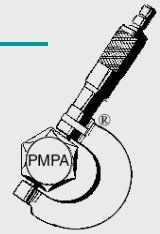
Supplied by Caron Engineering

This shows the correct parameter settings for this Macro Routine.
M201=O9001, M202=O9002, and M203=O9003.



Macro Examples: (Tool Life by Time Macro)

Supplied by Caron Engineering



START TIMER

```
O9001
(M201 START TIMER)
#105=0
#106=0
#107=0
#3002=0
#105=#3002
M99
```

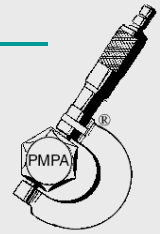
MEASURE TIME OF CUTTING PROCESS

```
O9002
(M202 ACCUMULATE TIME)
#106=#3002
#107=#106-#105 (TIME FROM M201 TO M202)
#103=FIX[#4120/100] (FIX T-CODE TO INTEGER)
IF[#103EQ0]GOTO5 (IF T-CODE IS ZERO DON'T USE)
IF[#103EQ#0]GOTO5 (IF T-CODE IS NULL DON'T USE)
#102=#103*2 (#102=T-CODE INTEGER * 2)
#100=500+#102 (DETERMINE ACCUMULATOR #)
#[#100]=#[#100]+#107 (ADD TIME TO ACCUMULATOR)
N5M99
```

TSUGAMI
Rem Sales, LLC

Macro Examples: (Tool Life by Time Macro)

Supplied by Caron Engineering

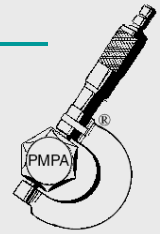



```
O9003 (M203 TOOL LIFE ALARMS)
(TOOL 1)
IF[#501EQ0]GOTO20 (IF PRESET IS ZERO SKIP)

IF[#501EQ#0]GOTO20 (IF PRESET IS NULL SKIP)

IF[#502LT#501]GOTO20 (IF ACCUMULATOR IS LESS THAN PRESET GOTO NEXT TOOL)

#502=0 (SET ACCUMULATOR TO ZERO)

#3000=1(HD1 TOOL 1 LIFE EXPIRED) (ALARM)
(TOOL 2)
N20IF[#503EQ0]GOTO30
IF[#503EQ#0]GOTO30
IF[#504LT#503]GOTO30
#504=0
#3000=2(HD1 TOOL 2 LIFE EXPIRED)
(TOOLS 3-12)
N999M99
%
```

TSUGAMI
Rem Sales, LLC

Macro Examples: (Tool Life by Time Macro)

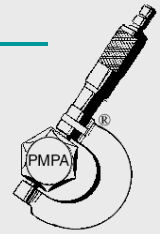
Supplied by Caron Engineering



Part Program Format:

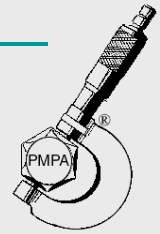
```
O1234
M203(TOOL LIFE CHECK)
G0G80G99

T0101
G0G99X1.0Z0M3S1000
M201(TIME RESET)
(CUTTING PROCESS)
M202(MEASURE TIME)
G28U0
T0
T0303
G0G99X1.0Z0M3S1000
M201(TIME RESET)
(CUTTING PROCESS)
M202(MEASURE TIME)
G28U0
T0
M203(TOOL LIFE CHECK)
M30
```

TSUGAMI
Rem Sales, LLC



TSUGAMI
Rem Sales, LLC

Macro Examples: M-Code Sub Program



This program uses a system variable to calculate on and off time for a chip conveyor.

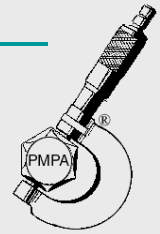
```
O9005(M2900 CONVEYOR ON-OFF)
(SET PARA 6075 TO 2900)
(AT BEGINNING OF PART PRG.)
(COMMAND THE FOLLOWING TWO LINES)
(#3001=0)
(M#120)
(COMMAND M2900 AT END OF PART PRG)
(SET #525 OFF TIME MINUTES)
(SET #527 ON TIME MINUTES)

(OFF TIME DECISION)
IF[#526GE#525]GOTO201
#120=291
#526=[[#3001/1000/60]+#526]
GOTO299

N201(ON TIME DECISION)
IF[#528GE#527]GOTO202
#120=290
#528=[[#3001/1000/60]+#528]
GOTO299

N202(RESET TIMERS)
#526=0(OFF TIMER RESET)
#528=0(ON TIMER RESET)

N299(END)
M99
```

Macro Examples: M-Code Sub Program

The program part program sets the timer to zero, commands the chip conveyor and timer sub-routine.

```
O1234(PART PROGRAM)
 #3001=0(ZERO MILLI SECOND TIMER)
M#120(CHIP CONVEYOR ON/OFF)

(PART PROGRAM)

M2900(CALLS PROGRAM O9005)
M30
```

Macro Examples: Tool/Offset Change Macro



This macro example alternates between two roughing tools for 100 cycles.

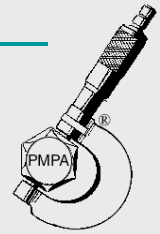
It will then generate a tool change alarm for the two roughing tools.

It also subtracts .001” from the finish tool offset every 10 parts for 100 cycles.

It will then generate a tool change alarm for the finish tool.

TSUGAMI
Rem Sales, LLC

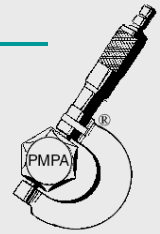
Macro Examples: Tool/Offset Change Macro




```
%
O1234(TEST ALTERNATE TOOL MACRO)
(ADJUST FINISH TOOL OFFSET)
(9/30/2012 REV 1A)

N20 (FRONT FACE & ROUGH TURN)          (#523 is the toggle counter for selecting the tool)
IF[#523EQ0]THEN#505=2323               (First Rough Tool Number)
IF[#523EQ1]THEN#505=2222               (Second Rough Tool Number)
M03 S3000 P1 G00 Z0.0 T#505            (Tool Call)
#522=[#522+1]                          (TOOL LIFE COUNTER)
X0.83
G01 X-0.04 F0.004
Z-0.04 F0.08
G41 X0.62
Z0.748 F0.004
X0.742
U0.016 W0.008
G40 G00 X4.7244 W0.0 T0
IF[#523EQ0]GOTO21
IF[#523EQ1]GOTO23
N21#523=1
GOTO25
N23#523=0
N25
```

TSUGAMI
Rem Sales, LLC

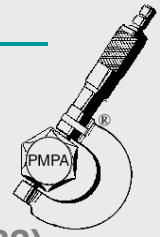
Macro Examples: Tool/Offset Change Macro




```
N30(FINISH TURNING)
M03 S3000 P1 G00 Z0.0T2121
#521=[#521+1]                    (TOOL LIFE COUNTER)
IF[#521LT10]GOTO38               (How many parts to make before you change the offset)
IF[#521EQ10]THEN#100=#2021       (Wear offset of Tool 21 in X-Axis)
#101=[#100-.001]                 (How much to change the offset)
#2021=#101
#521=0
N38X0.83
G01 X-0.04 F0.004
Z-0.04 F0.08
G41 X0.458
G01 X0.6 Z0.031 F0.004
Z0.75
X0.742
X0.83 W0.044
G40 G00 X4.7244 W0.0 T0

(AFTER CUT-OFF)
M95

/M92(BAR CHANGE)
M96
M97
```

TSUGAMI
Rem Sales, LLC

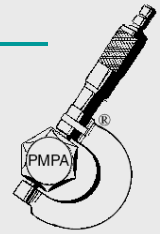
Macro Examples: Tool/Offset Change Macro




```
IF[#522LT100]GOTO9021                (TOOL LIFE TOTAL COUNT BOTH TOOL 22 & TOOL 23)
#522=0
G28U0
T0
#3000=1(CHANGE INSERT #T22& #T23)   (ALARM GENERATED)

N9021
IF[#521LT100]GOTO9022                (TOOL LIFE TOTAL COUNT FOR TOOL 21)
#521=0
G28U0
T0
#3000=1(CHANGE INSERT #T21)          (ALARM GENERATED)
N9022

N99M599
#501=0
M30
%
```

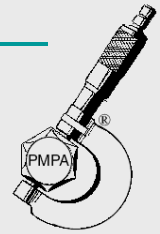
TSUGAMI
Rem Sales, LLC

Custom Macro and Probing



- The most common use for probing on a CNC machine is to determine part location for machining.
- Probes can also be used to measure features.
- When selecting a probe for your machine it is very important to work with the supplier to determine what your software requirements are.
- Most vendors will have a standard probing software package, but often these may need to be customized to suit your needs.
- You will also need to determine the control options required. (High Speed Skip, Additional Macro Variables, Work Offsets, Ect.)

Custom Macro and Probing



The probe is basically a spring loaded stylus with an electronic switch.

The machine moves the probe at a specified feed rate with a skip signal motion (G31).

Once the switch is tripped the machine motion stops. Now machine coordinates can be read to macro variables and decisions and calculations made accordingly.

Example:

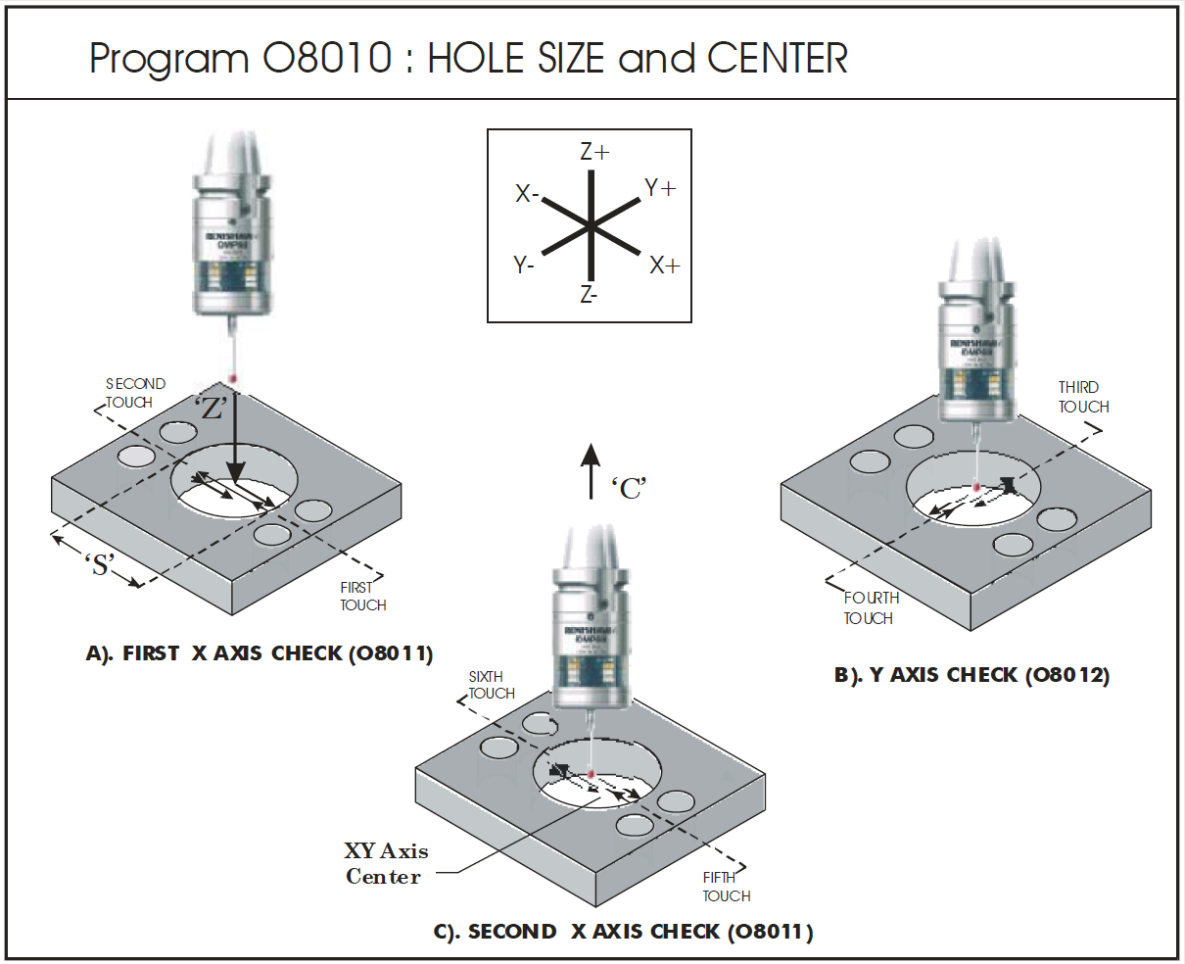
G0Z1.5;

G31Z1.0F4.0;

The machine will travel from Z1.5 to Z1.0 at 4 IPM in High Speed Skip Mode. Once the probe trips the machine motion stops. At this time the actual Z location can be recorded.

Custom Macro and Probing

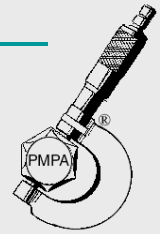
Caron Engineering Machining Center Probe Example:



TSUGAMI
Rem Sales, LLC

Custom Macro and Probing

Caron Engineering Machining Center Probe Example:



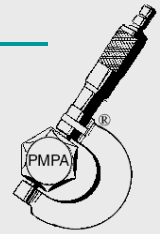
Macro arguments for program O8010

C		an **ABSOLUTE** safe **RETRACT** position in the **Z-axis**.
(D)	optional	the **cutter compensation offset number** used to cut the **HOLE**. (SET TO 0 OR VACANT, IF NO OFFSET IS TO BE ADJUSTED)
(E)	optional	the **+/- true position tolerance**. (If set to zero or vacant, true position is not checked.)
S		the **NOMINAL Hole Size.**
T		the **upper (+) tolerance** allowed from nominal size.
(U)	optional	the **lower (-) tolerance** allowed from nominal size. (If vacant, 'U is set to minus value of 'T'.)
(V)	optional	the **extra work offset groups G54 P1-P48** to store the XY axes position. (V = 1. to 48. for G54 P1 to P48; set V = 0 or #0, if no offset is to be set.)
(W)	optional	the **work offset groups G54 to G59** to store the XY axes position. (W = 1. to 6. for G54 to G59; set W = 0 or #0, if no offset is to be set.)
Z		the **ABSOLUTE Z-axis depth** to probe the **Hole**.

TSUGAMI
Rem Sales, LLC

Custom Macro and Probing

Caron Engineering Machining Center Probe Example:



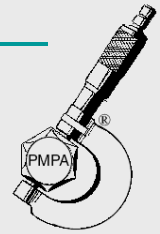
Programming example for O8010;
C=safe Z retract 1.0
S=Nominal hole size 1.4
T=Upper hole tolerance of .002
W=5th work offset update (G58)
Z=Depth inside hole to probe

```
T30 M6
G58 G90 G0 X0 Y0
M98 P8099
G90 G43 G31 H30 Z1. F80.
G65 P8010 C1. S1.4 T.002 W5. Z-.25
G91 G28 Z0
M98 P8098
M30
```

SUGAMI
Rem Sales, LLC

Custom Macro and Probing

Caron Engineering Machining Center Probe Example:



During the probing cycle all of the calculated information is loaded to variables. Here it can be accessed for any additional calculations or decisions.

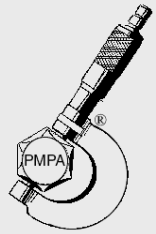
#102	---	The X-axis HOLE SIZE.
#112	---	The Y-axis HOLE SIZE.
#104	---	The AVERAGE HOLE SIZE.
#105	---	The HOLE SIZE ERROR
#106	---	The X-axis MACHINE Hole Center position.
#107	---	The Y-axis MACHINE Hole Center position.
#108	---	The X-axis ABSOLUTE Hole Center position.
#109	---	The Y-axis ABSOLUTE Hole Center position.
#118	---	The X and Y axes TRUE POSITION ERROR.

TSUGAMI
Rem Sales, LLC

Open Discussion

Questions?

TSUGAMI
Rem Sales, LLC



End

Thank You