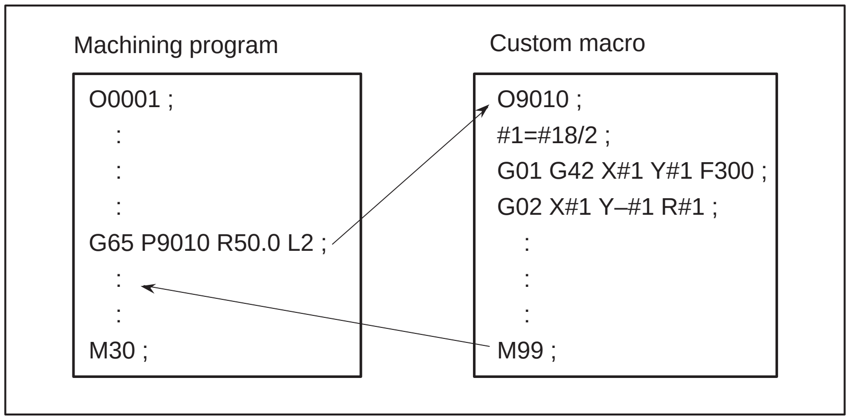
# 17 CUSTOM MACRO

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



## 17.1 VARIABLES

An ordinary machining program specifies a G code and the travel distance directly with a numeric value; examples are G100 and X100.0.
With a custom macro, numeric values can be specified directly or using a variable number. When a variable number is used, the variable value can be changed by a program or using operations on the MDI panel.

```
#1=#2+100 ;

G01 X#1 F300 ;
```

### Explanation

- **Variable representation**

When specifying a variable, specify a number sign (#) followed by a variable number. Personal computers allow a name to be assigned to a variable, but this capability is not available for custom macros.

**Example:** #1

An expression can be used to specify a variable number. In such a case, the expression must be enclosed in brackets.

**Example:** #[#1+#2–12]

- **Range of variable values**

Local and common variables can have value 0 or a value in the following ranges :
**$-10_{47}$ to $-10_{-29}$**
**$10_{-29}$ to $10_{47}$**
If the result of calculation turns out to be invalid, an alarm No. 111 is issued.

- **Omission of the decimal point**

When a variable value is defined in a program, the decimal point can be omitted.

**Example:**
When #1=123; is defined, the actual value of variable #1 is 123.000.

- **Undefined variable**

When the value of a variable is not defined, such a variable is referred to as a “null” variable. Variable #0 is always a null variable. It cannot be written to, but it can be read.

(a) Quotation
When an undefined variable is quotated, the address itself is also ignored.

| When #1 = <vacant > | When #1 = 0 |
|---|---|
| G90 X100 Y#1 ↓ G90 X100 | G90 X100 Y#1 ↓ G90 X100 Y0 |

(b) Operation

< vacant > is the same as 0 except when replaced by < vacant>

| When #1 = < vacant > | When #1 = 0 |
|---|---|
| #2 = #1 ↓ #2 = <vacant > | #2 = #1 ↓ #2 = 0 |
| #2 = #1*5 ↓ #2 = 0 | #2 = #1*5 ↓ #2 = 0 |
| #2 =  #1+#1 ↓ #2 = 0 | #2 = #1 + #1 ↓ #2 = 0 |

(c) Conditional expressions

< vacant > differs from 0 only for EQ and NE.

| When #1 = <vacant > | When #1 = 0 |
|---|---|
| #1 EQ #0 ↓ Established | #1 EQ #0 ↓ Not established |
| #1 NE 0 ↓ Established | #1 NE 0 ↓ Not established |
| #1 GE #0 ↓ Established | #1 GE #0 ↓ Established |
| #1 GT 0 ↓ Not established | #1 GT 0 ↓ Not established |

## ● Types of variables

Variables are classified into four types by variable number.

**Table 17.1 Types of variables**

| Variable number | Type of variable | Function |
|---|---|---|
| #0 | Always null | This variable is always null. No value can be assigned to this variable. |
| #1 to #33 | Local variables | Local variables can only be used within a macro to hold data such as the results of operations. When the power is turned off, local variables are initialized to null. When a macro is called, arguments are assigned to local variables. |
| #100 to #149 (#199)<br>#500 to #531 (#999) | Common variables | Common variables can be shared among different macro programs. When the power is turned off, variables #100 to #149 are initialized to null. Variables #500 to #531 hold data even when the power is turned off. As an option, common variables #150 to #199 and #532 to #999 are also available. However, when these values are using, the length of the tape that can be used for storage decreases by 8.5 m. |
| #1000 to | System variables | System variables are used to read and write a variety of NC data items such as the current position and tool compensation values. |

**NOTE**
Common variables #150 to #199 and #532 to #999 are optional.

## ● Referencing variables

To reference the value of a variable in a program, specify a word address followed by the variable number. When an expression is used to specify a variable, enclose the expression in brackets.

**Example:** G01X[#1+#2]F#3;

A referenced variable value is automatically rounded according to the least input increment of the address.

**Example:**
When G00X#1; is executed on a 1/1000–mm CNC with 12.3456 assigned to variable #1, the actual command is interpreted as G00X12.346;.

To reverse the sign of a referenced variable value, prefix a minus sign (–) to #.

**Example:** G00X–#1;

When an undefined variable is referenced, the variable is ignored up to an address word.

**Example:**
When the value of variable #1 is 0, and the value of variable #2 is null, execution of G00X#1Y#2; results in G00X0;.

## ● Displaying variable values

### Procedure for displaying variable values

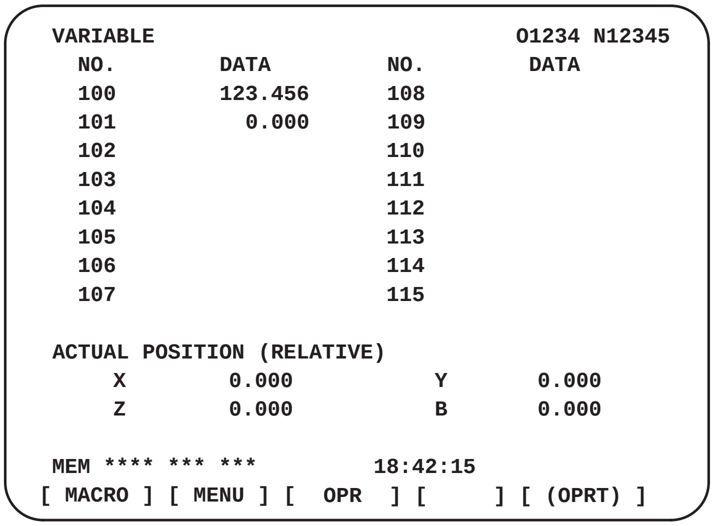
**Procedure**

**1** Press the OFFSET SETTING key to display the tool compensation screen.

**2** Press the continuous menu key ▷.

**3** Press the soft key **[MACRO]** to display the macro variable screen.

**4** Enter a variable number, then press soft key **[NO.SRH]**.
The cursor moves to the position of the entered number.



```
VARIABLE                           O1234 N12345
  NO.        DATA         NO.        DATA
  100        123.456      108
  101          0.000      109
  102                     110
  103                     111
  104                     112
  105                     113
  106                     114
  107                     115

ACTUAL POSITION (RELATIVE)
     X        0.000          Y       0.000
     Z        0.000          B       0.000

MEM **** *** ***         18:42:15
[ MACRO ] [ MENU ] [  OPR  ] [     ] [ (OPRT) ]
```

· When the value of a variable is blank, the variable is null.

· The mark ******** indicates an overflow (when the absolute value of a variable is greater than 99999999) or an underflow (when the absolute value of a variable is less than 0.0000001).

**Limitations**

Program numbers, sequence numbers, and optional block skip numbers cannot be referenced using variables.

**Example:**
Variables cannot be used in the following ways:
O#1;
/#2G00X100.0;
N#3Y200.0;

## 17.2 SYSTEM VARIABLES

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

### Explanations

- **Interface signals**

Signals can be exchanged between the programmable machine controller (PMC) and custom macros.

**Table 17.2(a) System variables for interface signals**

| Variable number | Function |
|---|---|
| #1000 to #1015<br>#1032 | A 16–bit signal can be sent from the PMC to a custom macro. Variables #1000 to #1015 are used to read a signal bit by bit. Variable #1032 is used to read all 16 bits of a signal at one time. |
| #1100 to #1115<br>#1132 | A 16–bit signal can be sent from a custom macro to the PMC. Variables #1100 to #1115 are used to write a signal bit by bit. Variable #1132 is used to write all 16 bits of a signal at one time. |
| #1133 | Variable #1133 is used to write all 32 bits of a signal at one time from a custom macro to the PMC.<br>Note, that values from –99999999 to +99999999 can be used for #1133. |

For detailed information, refer to the connection manual (B–62443E–1).

- **Tool compensation values**

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

**Table 17.2(b)**

| Compensation number | System variable |
|---|---|
| 1 | #10001 (#2001) |
| : | : |
| 200 | #10200 (#2200) |
| : | : |
| 400 | #10400 |

## ● Macro alarms

Table 17.2(c) System variable for macro alarms

| Variable number | Function |
|---|---|
| #3000 | When a value from 0 to 200 is assigned to variable #3000, the NC stops with an alarm. After an expression, an alarm message not longer than 26 characters can be described. The CRT screen displays alarm numbers by adding 3000 to the value in variable #3000 along with an alarm message. |

**Example:**
#3000=1(TOOL NOT FOUND);
→ The alarm screen displays "3001 TOOL NOT FOUND."

## ● Time information

Time information can be read and written.

Table 17.2(d) System variables for time information

| Variable number | Function |
|---|---|
| #3001 | This variable functions as a timer that counts in 1–millisecond increments at all times. When the power is turned on, the value of this variable is reset to 0. When 65535 milliseconds is reached, the value of this timer returns to 0. |
| #3002 | This variable functions as a timer that counts in 1–hour increments when the cycle start lamp is on. This timer preserves its value even when the power is turned off. When 1145324.612 hours is reached, the value of this timer returns to 0. |
| #3011 | This variable can be used to read the current date (year/month/day). Year/month/day information is converted to an apparent decimal number. For example, March 28, 1993 is represented as 19930328. |
| #3012 | This variable can be used to read the current time (hours/minutes/seconds). Hours/minutes/seconds information is converted to an apparent decimal number. For example, 34 minutes and 56 seconds after 3 p.m. is represented as 153456. |

## ● Automatic operation control

The control state of automatic operation can be changed.

Table 17.2(e) System variable (#3003) for automatic operation control

| #3003 | Single block | Completion of an auxiliary function |
|---|---|---|
| 0 | Enabled | To be awaited |
| 1 | Disabled | To be awaited |
| 2 | Enabled | Not to be awaited |
| 3 | Disabled | Not to be awaited |

- When the power is turned on, the value of this variable is 0.
- When single block stop is disabled, single block stop operation is not performed even if the single block switch is set to ON.

- When a wait for the completion of auxiliary functions (M, S, and T functions) is not specified, program execution proceeds to the next block before completion of auxiliary functions. Also, distribution completion signal DEN is not output.

**Table 17.2(f) System variable (#3004) for automatic operation control**

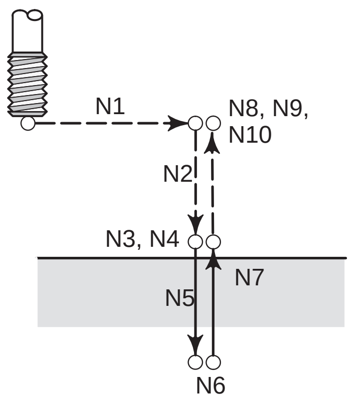
| #3004 | Feed hold | Feedrate Override | Exact stop |
|---|---|---|---|
| 0 | Enabled | Enabled | Enabled |
| 1 | Disabled | Enabled | Enabled |
| 2 | Enabled | Disabled | Enabled |
| 3 | Disabled | Disabled | Enabled |
| 4 | Enabled | Enabled | Disabled |
| 5 | Disabled | Enabled | Disabled |
| 6 | Enabled | Disabled | Disabled |
| 7 | Disabled | Disabled | Disabled |

- When the power is turned on, the value of this variable is 0.
- When feed hold is disabled:

(1) When the feed hold button is held down, the machine stops in the single block stop mode. However, single block stop operation is not performed when the single block mode is disabled with variable #3003.

(2) When the feed hold button is pressed then released, the feed hold lamp comes on, but the machine does not stop; program execution continues and the machine stops at the first block where feed hold is enabled.

- When feedrate override is disabled, an override of 100% is always applied regardless of the setting of the feedrate override switch on the machine operator's panel.
- When exact stop check is disabled, no exact stop check (position check) is made even in blocks including those which do not perform cutting.



```
O0001 ;
N1  G00 G91 X#24 Y#25 ;
N2  Z#18 ;
    G04 ;
N3  #3003=3 ;
N4  #3004=7 ;
N5  G01 Z#26 F#9 ;
N6  M04 ;
N7  G01
Z–[ROUND[#18]+ROUND[#26]] ;
    G04 ;
N8  #3004=0 ;
N9  #3003=0 ;
N10 M03 ;
    M99 ;
```

**Fig. 17.2(a) Example of using variable #3004 in a tapping cycle**

## • Settings

Settings can be read and written. Binary values are converted to decimals.

**#3005**

| | #15 | #14 | #13 | #12 | #11 | #10 | #9 | #8 |
|---|---|---|---|---|---|---|---|---|
| Setting | | | | | | | | |

| | #7 | #6 | #5 | #4 | #3 | #2 | #1 | #0 |
|---|---|---|---|---|---|---|---|---|
| Setting | | | SEQ | | | INI | ISO | TVC |

| | | |
|---|---|---|
| #5 (SEQ) | : | Whether to automatically insert sequence numbers |
| #2 (INI) | : | Millimeter input or inch input |
| #1 (ISO) | : | Whether to use EIA or ISO as the output code |
| #0 (TVC) | : | Whether to make a TV check |

## • Mirror image

The mirror–image status for each axis set using an external switch or setting operation can be read through the output signal (mirror–image check signal). The mirror–image status present at that time can be checked. (See Section 4.7 in III.)
The value obtained in binary is converted into decimal notation.

**#3007**

| | #7 | #6 | #5 | #4 | #3 | #2 | #1 | #0 |
|---|---|---|---|---|---|---|---|---|
| Setting | 8th axis | 7th axis | 6th axis | 5th axis | 4th axis | 3th axis | 2th axis | 1th axis |

For each bit, 0 (mirror–image function is disabled) or 1 (mirror–image function is enabled) is indicated.

Example : If #3007 is 3, the mirror–image function is enabled for the first and second axes.

- When the mirror–image function is set for a certain axis by both the mirror–image signal and setting, the signal value and setting value are ORed and then output.
- When mirror–image signals for axes other than the controlled axes are turned on, they are still read into system variable #3007.
- System variable #3007 is a write–protected system variable. If an attempt is made to write data in the variable, P/S 116 alarm “WRITE PROTECTED VARIABLE” is issued.

## • Number of machined parts

The number (target number) of parts required and the number (completion number) of machined parts can be read and written.

**Table 17.2(g) System variables for the number of parts required and the number of machined parts**

| Variable number | Function |
|---|---|
| #3901 | Number of machined parts (completion number) |
| #3902 | Number of required parts (target number) |

**WARNING**
Do not substitute a negative value.

- **Modal information**

Modal information specified in blocks up to the immediately preceding block can be read.

**Table 17.2(h) System variables for modal information**

| Variable number | Function | |
|---|---|---|
| #4001 | G00, G01, G02, G03, G33 | (Group 01) |
| #4002 | G17, G18, G19 | (Group 02) |
| #4003 | G90, G91 | (Group 03) |
| #4004 | | (Group 04) |
| #4005 | | (Group 05) |
| #4006 | G20, G21 | (Group 06) |
| #4007 | G40, G41, G42 | (Group 07) |
| #4008 | | (Group 08) |
| #4009 | | (Group 09) |
| #4010 | | (Group 10) |
| #4011 | G50, G51 | (Group 11) |
| #4012 | G65, G66, G67 | (Group 12) |
| #4014 | G54–G59 | (Group 14) |
| #4015 | G61–G64 | (Group 15) |
| #4016 | G84, G85 | (Group 16) |
| : | : | : |
| #4022 | | (Group 22) |
| #4102 | B code | |
| #4109 | F code | |
| #4111 | H code | |
| #4113 | M code | |
| #4114 | Sequence number | |
| #4115 | Program number | |
| #4119 | S code | |
| #4120 | T code | |
| #4130 | P code (number of the currently selected additional workpiece coordinate system) | |

**Example:**

When #1=#4001; is executed, the resulting value in #1 is 0, 1, 2, 3, or 33.

## ● Current position

Position information cannot be written but can be read.

**Table 17.2(i) System variables for position information**

| Variable number | Position information | Coordinate system | Tool compensation value | Read operation during movement |
|---|---|---|---|---|
| #5001 to #5008 | Block end point | Workpiece coordinate system | Not included | Enabled |
| #5021 to #5028 | Current position | Machine coordinate system | Included | Disabled |
| #5041 to #5048 | Current position | Workpiece coordinate system | | |
| #5061 to #5068 | Skip signal position | | | Enabled |
| #5081 to #5088 | Tool offset value | | | Disabled |
| #5101 to #5108 | Deviated servo position | | | |
| #6251 to #6258 | Pattern base position | Workpiece coordinate system | Not included | Enabled |
| #6261 to #6268 | Multi–piece machining coordinate system | Workpiece coordinate system | Not included | Enabled |
| #6271 to #6278 | Local coordinate system | Workpiece coordinate system | Not included | Enabled |

- The first digit (from 1 to 8) represents an axis number.
- The tool offset value currently used for execution rather than the immediately preceding tool offset value is held in variables #5081 to 5088.
- The tool position where the skip signal is turned on in a G33 (skip function) block is held in variables #5061 to #5068. When the skip signal is not turned on in a G33 block, the end point of the specified block is held in these variables.
- When read during movement is “disabled,” this means that expected values cannot be read due to the buffering (preread) function.

- **Workpiece coordinate system compensation values (workpiece zero point offset values)**

Workpiece zero point offset values can be read and written.

**Table 17.2(j) System variables for workpiece zero point offset values**

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

**NOTE**

Variables #5201 to #5328 are optional variables for the workpiece coordinate systems.

## 17.3 ARITHMETIC AND LOGIC OPERATION

The operations listed in Table 16.3(a) can be performed on variables. The expression to the right of the operator can contain constants and/or variables combined by a function or operator. Variables #j and #K in an expression can be replaced with a constant. Variables on the left can also be replaced with an expression.

**Table 17.3(a) Arithmetic and logic operation**

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

### Explanations

#### ● Angle units

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

#### ● ATAN function

After the ATAN function, specify the lengths of two sides separated by a slash. A result is found where $0 \leqq \text{result} < 360$.

**Example :**
When #1=ATAN[1]/[–1], the value of #1 is 135.0

#### ● 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.
G01 X–#2 F300; Moves 2.346 mm.
G00 X[#1+#2];
Since 1.2345 + 2.3456 = 3.5801, 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.

### ● Rounding up and down to an integer

With NC, 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.

### ● Abbreviations of arithmetic and logic operation commands

When a function is specified in a program, the first two characters of the function name can be used to specify the function.

**Example:**

ROUND → RO
FIX → FI

### ● Priority of operations

$\boxed{1}$ Functions
$\boxed{2}$ Operations such as multiplication and division (*, /, AND, MOD)
$\boxed{3}$ Operations such as addition and subtraction (+, –, OR, XOR)

Example) $\#1=\underbrace{\#2+\underbrace{\#3*\underbrace{SIN[\#4]}_{\boxed{1}}}_{\boxed{2}}}_{\boxed{3}};$

$\boxed{1}$, $\boxed{2}$ and $\boxed{3}$ 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, alarm No. 118 occurs.

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

$$\#1=\underbrace{\mathrm{SIN}\,\underbrace{[\,\underbrace{[\,\underbrace{\underbrace{[\#2+\#3]}_{\boxed{1}}*\#4}_{\boxed{2}}+\#5]}_{\boxed{3}}*\#6]}_{\boxed{4}}}_{\boxed{5}}\;;$$

$\boxed{1}$ to $\boxed{5}$ indicate the order of operations.

## Limitations

### ● Brackets

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

### ● Operation error

Errors may occur when operations are performed.

**Table 17.3(b) Errors involved in operations**

| Operation | Average error | Maximum error | Type of error |
|---|---|---|---|
| a = b*c | $1.55\times10^{-10}$ | $4.66\times10^{-10}$ | Relative error(*1) $\left\lvert \frac{\varepsilon}{a} \right\rvert$ |
| a = b / c | $4.66\times10^{-10}$ | $1.88\times10^{-9}$ | |
| $a = \sqrt{b}$ | $1.24\times10^{-9}$ | $3.73\times10^{-9}$ | |
| a = b + c<br>a = b – c | $2.33\times10^{-10}$ | $5.32\times10^{-10}$ | (*2) $\mathrm{Min}\left\lvert \frac{\varepsilon}{b} \right\rvert , \left\lvert \frac{\varepsilon}{c} \right\rvert$ |
| a = SIN [ b ]<br>a = COS [ b ] | $5.0\times10^{-9}$ | $1.0\times10^{-8}$ | Absolute error(*3) $\lvert \varepsilon \rvert$ degrees |
| a = ATAN [ b ] / [ c ]<br>(*4) | $1.8\times10^{-6}$ | $3.6\times10^{-6}$ | |

**WARNING**
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.

- 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 #1 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]

## ● Divisor

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

## 17.4 MACRO STATEMENTS AND NC STATEMENTS

The following blocks are referred to as macro statements:

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

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

### Explanations

#### ● Differences from NC statements

- Even when single block mode is on, the machine does not stop. Note, however, that the machine stops in the single block mode when bit 5 of parameter 6000 is 1.
- Macro blocks are not regarded as blocks that involve no movement in the cutter compensation mode (see Section 16.7).

#### ● NC statements that have the same property as macro statements

- NC statements that include a subprogram call command (such as subprogram calls by M98 or other M codes, or by T codes) and also include an O, N, P, or L address have the same property as macro statements.
- NC statements that include M99 and an O, N, L, or P address have the same property as macro statements.

## 17.5 BRANCH AND REPETITION

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

Branch and repetition
- GOTO statement (unconditional branch)
- IF statement (conditional branch: if ..., then...)
- WHILE statement (repetition while ...)

### 17.5.1 Unconditional Branch (GOTO Statement)

A branch to sequence number n occurs. When a sequence number outside of the range 1 to 99999 is specified, alarm No. 128 occurs. A sequence number can also be specified using an expression.

```
GOTO n ;     n:  Sequence number (1 to 99999)
```

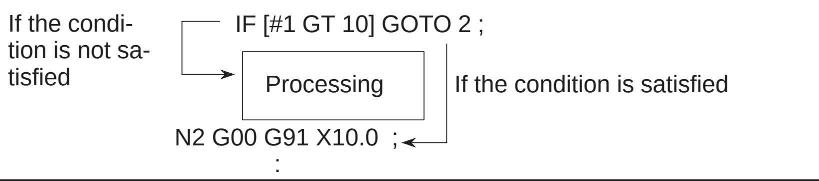
**Example:**

```
GOTO1;
GOTO#10;
```

### 17.5.2 Conditional Branch (IF Statement)

Specify a conditional expression after IF. If the specified conditional expression is satisfied, a branch to sequence number n occurs. If the specified condition is not satisfied, the next block is executed.

If the value of variable #1 is greater than 10, a branch to sequence number N2 occurs.



```
IF [#1 GT 10] GOTO 2 ;
Processing
N2 G00 G91 X10.0 ;
:
```

If the condition is not satisfied
If the condition is satisfied

#### Explanations

- **Conditional expression**

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

## ● Operators

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

**Table 17.5.2 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(≦) |

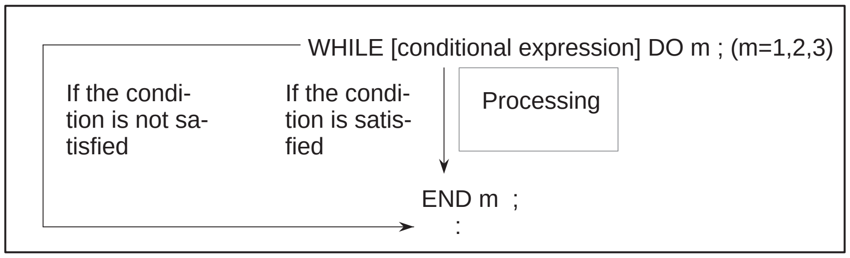
## Sample program

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
```

## 17.5.3 Repetition (While Statement)

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

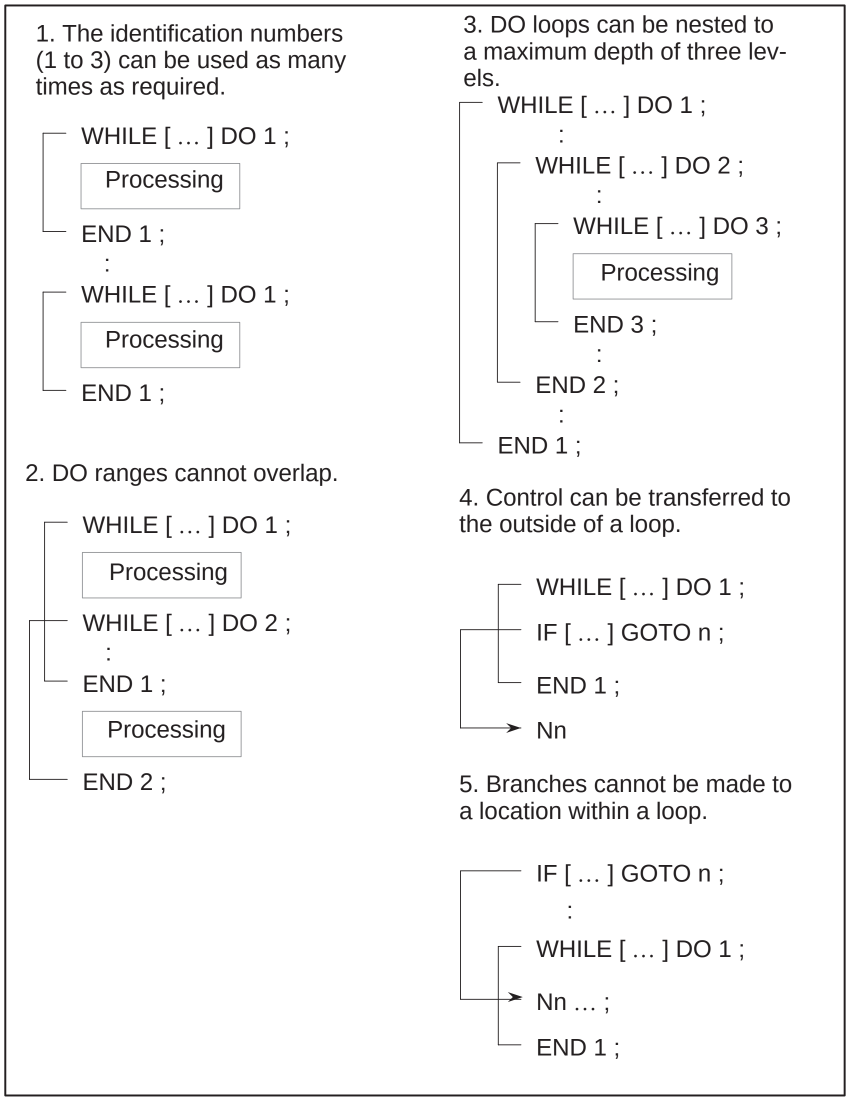

WHILE [conditional expression] DO m ; (m=1,2,3)
If the condition is not satisfied
If the condition is satisfied
Processing
END m ;
:

## Explanations

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, alarm No. 126 occurs.

- **Nesting**

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



1. The identification numbers (1 to 3) can be used as many times as required.

```
WHILE [ … ] DO 1 ;
Processing
END 1 ;
  :
WHILE [ … ] DO 1 ;
Processing
END 1 ;
```

2. DO ranges cannot overlap.

```
WHILE [ … ] DO 1 ;
Processing
WHILE [ … ] DO 2 ;
  :
END 1 ;
Processing
END 2 ;
```

3. DO loops can be nested to a maximum depth of three levels.

```
WHILE [ … ] DO 1 ;
      :
  WHILE [ … ] DO 2 ;
        :
    WHILE [ … ] DO 3 ;
    Processing
    END 3 ;
        :
  END 2 ;
      :
END 1 ;
```

4. Control can be transferred to the outside of a loop.

```
WHILE [ … ] DO 1 ;
IF [ … ] GOTO n ;
END 1 ;
Nn
```

5. Branches cannot be made to a location within a loop.

```
IF [ … ] GOTO n ;
    :
WHILE [ … ] DO 1 ;
Nn … ;
END 1 ;
```

## Limitations

- **Infinite loops**

When DO m is specified without specifying the WHILE statement, an infinite loop ranging from DO to END is produced.

- **Processing time**

When a branch to the sequence number specified in a GOTO statement occurs, the sequence number is searched for. For this reason, processing in the reverse direction takes a longer time than processing in the forward direction. Using the WHILE statement for repetition reduces processing time.

- **Undefined variable**

In a conditional expression that uses EQ or NE, a null value and zero have different effects. In other types of conditional expressions, a null value is regarded as zero.

## Sample program

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

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

## 17.6 MACRO CALL

A macro program can be called using the following methods:

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

### Limitations

- **Differences between macro calls and subprogram calls**

Macro call (G65) differs from subprogram call (M98) as described below.

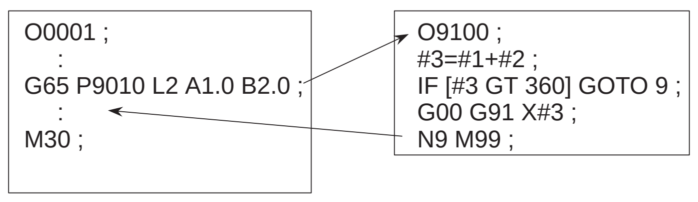
- With G65, an argument (data passed to a macro) can be specified. M98 does not have this capability.
- When an M98 block contains another NC command (for example, G01 X100.0 M98Pp), the subprogram is called after the command is executed. On the other hand, G65 unconditionally calls a macro.
- When an M98 block contains another NC command (for example, G01 X100.0 M98Pp), the machine stops in the single block mode. On the other hand, G65 does not stops the machine.
- With G65, the level of local variables changes. With M98, the level of local variables does not change.

## 17.6.1 Simple Call (G65)

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

G65 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 ;
```

```
O9100 ;
#3=#1+#2 ;
IF [#3 GT 360] GOTO 9 ;
G00 G91 X#3 ;
N9 M99 ;
```

### Explanations

- **Call**

- After G65, specify at address P the program number of the custom macro to call.
- When a number of repetitions is required, specify a number from 1 to 9999 after address L. When L is omitted, 1 is assumed.
- By using argument specification, values are assigned to corresponding local variables.

### ● Argument specification

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

- Addresses G, L, N, O, and P cannot be used in arguments.
- Addresses that need not be specified can be omitted. Local variables corresponding to an omitted address are set to null.

**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 | Address | Variable number | Address | Variable number |
|---|---|---|---|---|---|
| A | #1 | $K_3$ | #12 | $J_7$ | #23 |
| B | #2 | $I_4$ | #13 | $K_7$ | #24 |
| C | #3 | $J_4$ | #14 | $I_8$ | #25 |
| $I_1$ | #4 | $K_4$ | #15 | $J_8$ | #26 |
| $J_1$ | #5 | $I_5$ | #16 | $K_8$ | #27 |
| $K_1$ | #6 | $J_5$ | #17 | $I_9$ | #28 |
| $I_2$ | #7 | $K_5$ | #18 | $J_9$ | #29 |
| $J_2$ | #8 | $I_6$ | #19 | $K_9$ | #30 |
| $K_2$ | #9 | $J_6$ | #20 | $I_{10}$ | #31 |
| $I_3$ | #10 | $K_6$ | #21 | $J_{10}$ | #32 |
| $J_3$ | #11 | $I_7$ | #22 | $K_{10}$ | #33 |

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

## Limitations

### ● Format

G65 must be specified before any argument.

### ● Mixture of argument specifications I and II

The NC 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.

### ● Position of the decimal point

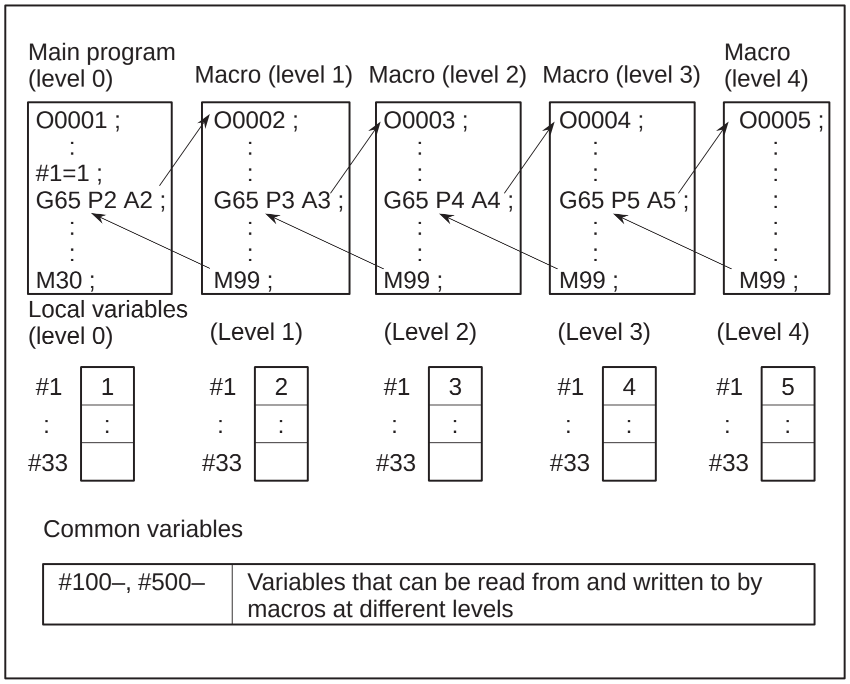
The units used for argument data passed without a decimal point correspond to the least input increment of each address. The value of an argument passed without a decimal point may vary according to the system configuration of the machine. It is good practice to use decimal points in macro call arguments to maintain program compatibility.

- **Call nesting**

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

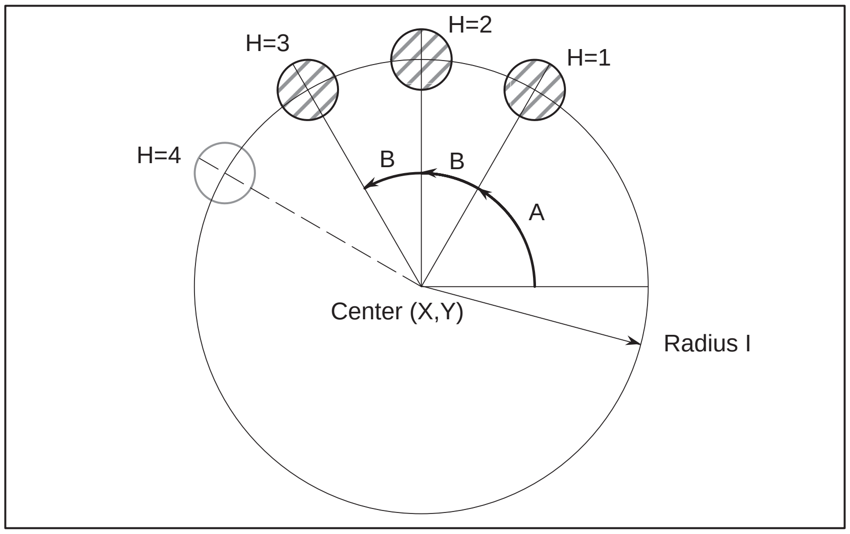
- **Local variable levels**

- Local variables from level 0 to 4 are provided for nesting.
- The level of the main program is 0.
- Each time a macro is called (with G65 or G66), the local variable level is incremented by one. The values of the local variables at the previous level are saved in the NC.
- 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.



## Sample program (bolt hole circle)

A macro is created which drills H holes at intervals of B degrees after a start angle of A degrees along the periphery of a circle with radius I. The center of the circle is (X,Y). Commands can be specified in either the absolute or incremental mode. To drill in the clockwise direction, specify a negative value for B.



### ● Calling format

```
G65 P9100 X x Y y Z z R r F f I i A a B b H h ;
```

X: X coordinate of the center of the circle (absolute or incremental specification) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . (#24)
Y: Y coordinate of the center of the circle (absolute or incremental specification) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . (#25)
Z : Hole depth . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . (#26)
R: Coordinates of an approach point . . . . . . . . . . . . . . . . . . . . . (#18)
F : Cutting feedrate . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . (#9)
I : Radius of the circle . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . (#4)
A: Drilling start angle . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . (#1)
B: Incremental angle (clockwise when a negative value is specified) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . (#2)
H: Number of holes . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . (#11)

- **Program calling a macro program**

```
O0002;
G90 G92 X0 Y0 Z100.0;
G65 P9100 X100.0 Y50.0 R30.0 Z–50.0 F500 I100.0 A0 B45.0 H5;
M30;
```

- **Macro program (called program)**

```
O9100;
  #3=#4003;  . . . . . . . . . . . . . . . . . . . . . . . . . Stores G code of group 3.
  G81 Z#26 R#18 F#9 K0; (Note)\  . . . . . . . . . . . . . . . . . . Drilling cycle.
     . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .  Note:  L0 can also be used.
  IF[#3 EQ 90]GOTO 1; . . . . . . . . . .  Branches to N1 in the G90 mode.
  #24=#5001+#24; . . . . . . . . .  Calculates the X coordinate of the center.
  #25=#5002+#25; . . . . . . . . .  Calculates the Y coordinate of the center.
N1 WHILE[#11 GT 0]DO 1;
 . . . . . . . . . . . . . . . . . . .  Until the number of remaining holes reaches 0
  #5=#24+#4*COS[#1]; . . .  Calculates a drilling position on the X–axis.
  #6=#25+#4*SIN[#1];  . . .  Calculates a drilling position on the Y–axis.
  G90 X#5 Y#6;  .  Performs drilling after moving to the target position.
  #1=#1+#2;  . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .  Updates the angle.
  #11=#11–1; . . . . . . . . . . . . . . . . . . . . Decrements the number of holes.
  END 1;
  G#3 G80;   . . . . . . . . . . . . . .  Returns the G code to the original state.
  M99;
```

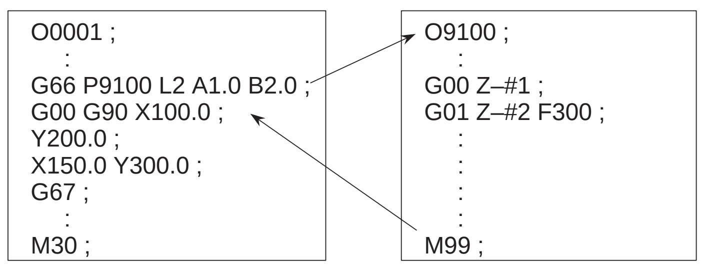
Meaning of variables:

#3: Stores the G code of group 3.
#5: X coordinate of the next hole to drill
#6: Y coordinate of the next hole to drill

## 17.6.2 Modal Call (G66)

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

```
G66  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 ;                          O9100 ;
   :                                :
G66 P9100 L2 A1.0 B2.0 ;         G00 Z–#1 ;
G00 G90 X100.0 ;                 G01 Z–#2 F300 ;
Y200.0 ;                            :
X150.0 Y300.0 ;                     :
G67 ;                               :
   :                                :
M30 ;                            M99 ;
```

### Explanations

#### • 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 9999 can be specified at address L.
- As with a simple call (G65), data passed to a macro program is specified in arguments.

#### • Cancellation

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

#### • Call nesting

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

#### • Modal call nesting

Modal calls can be nested by specifying another G66 code during a modal call.

### Limitations

- In a G66 block, no macros can be called.
- G66 needs to be specified before any arguments.
- No macros can be called in a block which contains a code such as a miscellaneous 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.

## Sample program

The drilling cycle is created using a custom macro and the machining program makes a modal macro call. For program simplicity, all drilling data is specified using absolute values.

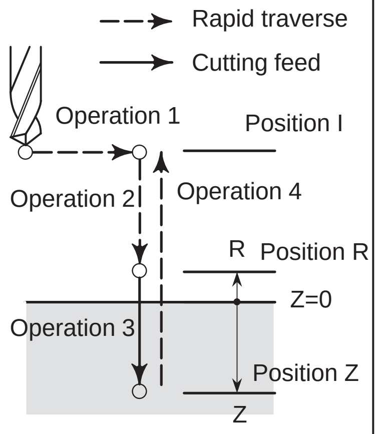
The canned cycle consists of the following basic operations:

Operation 1:
Positioning along the X–axis and Y–axis

Operation 2:
Rapid traverse to point R

Operation 3:
Cutting feed to point Z

Operation 4:
Rapid traverse to point R or I



### ● Calling format

```
G65 P9110 X x Y y Z z R r F f L l ;
```

| | | |
|---|---|---|
| X: | X coordinate of the hole (absolute specification only) . . . . | (#24) |
| Y: | Y coordinate of the hole (absolute specification only) . . . . | (#25) |
| Z : | Coordinates of position Z (absolute specification only) . . . | (#26) |
| R: | Coordinates of position R (absolute specification only) . . . | (#18) |
| F : | Cutting feedrate . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | (#9) |
| L : | Repetition count | |

### ● Program that calls a macro program

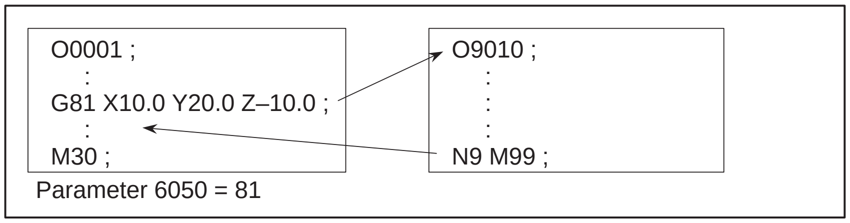
```
O0001;
G28 G91 X0 Y0 Z0;
G92 X0 Y0 Z50.0;
G00 G90 X100.0 Y50.0;
G66 P9110 Z–20.0 R5.0 F500;
G90 X20.0 Y20.0;
X50.0;
Y50.0;
X70.0 Y80.0;
G67;
M30;
```

### ● Macro program (program called)

```
O9110;
  #1=#4001; . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . Stores G00/G01.
  #3=#4003; . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . Stores G90/G91.
  #4=#4109; . . . . . . . . . . . . . . . . . . . . . . . . . Stores the cutting feedrate.
  #5=#5003; . . . . . . . . . . Stores the Z coordinate at the start of drilling.
  G00 G90 Z#18; . . . . . . . . . . . . . . . . . . . . . . Positioning at position R
  G01 Z#26 F#9; . . . . . . . . . . . . . . . . . . . . . . Cutting feed to position Z
  IF[#4010 EQ 98]GOTO 1; . . . . . . . . . . . . . . . . . . . Return to position I
  G00 Z#18; . . . . . . . . . . . . . . . . . . . . . . . . . . Positioning at position R
  GOTO 2;
N1 G00 Z#5; . . . . . . . . . . . . . . . . . . . . . . . . . . Positioning at position I
N2 G#1 G#3 F#4; . . . . . . . . . . . . . . . . . . . . Restores modal information.
  M99;
```

## 17.6.3 Macro Call Using G Code

By setting a G code number used to call a macro program in a parameter, the macro program can be called in the same way as for a simple call (G65).



### Explanations

By setting a G code number from 1 to 255 used to call a custom macro program (9010 to 9019) in the corresponding parameter (6050 to 6059), the macro program can be called in the same way as with G65.
For example, when a parameter is set so that macro program O9010 can be called with G81, a user–specific cycle created using a custom macro can be called without modifying the machining program.

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

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

- **Repetition**

As with a simple call, a number of repetitions from 1 to 9999 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.

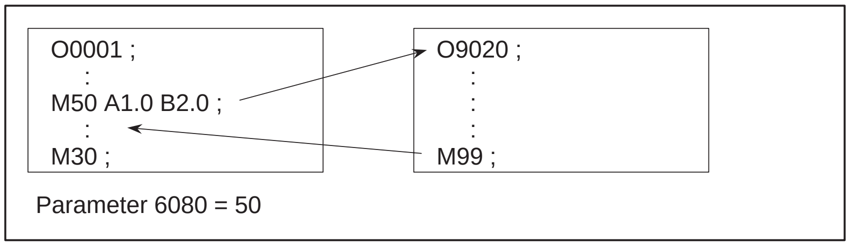
### Limitations

- **Nesting of calls using G codes**

In a program called with a G code, no macros can be called using a G code. A G code in such a program is treated as an ordinary G code. In a program called as a subprogram with an M or T code, no macros can be called using a G code. A G code in such a program is also treated as an ordinary G code.

## 17.6.4 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).



```
O0001 ;                    O9020 ;
   :                          :
M50 A1.0 B2.0 ;               :
   :                          :
M30 ;                      M99 ;

Parameter 6080 = 50
```

### Explanations

By setting an M code number from 1 to 255 used to call a custom macro program (9020 to 9029) in the corresponding parameter (6080 to 6089), the macro program can be called in the same way as with G65.

#### • Correspondence between parameter numbers and program numbers

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

#### • Repetition

As with a simple call, a number of repetitions from 1 to 9999 can be specified at address L.

#### • Argument specification

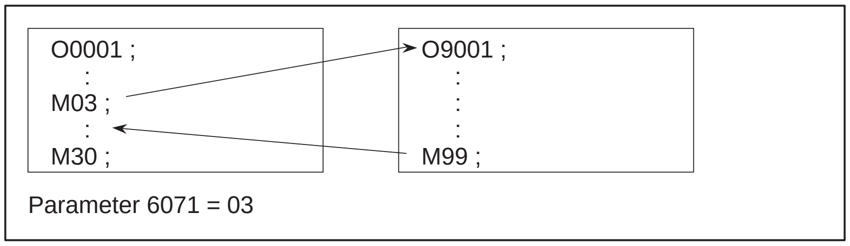
As with a simple call, two types of argument specification are available: Argument specification I and argument specification II. The type of argument specification is determined automatically according to the addresses used.

### Limitations

- An M code used to call a macro program must be specified at the start of a block.
- In a macro called with a G code or in a program called as a subprogram with an M or T code, no macros can be called using an M code. An M code in such a macro or program is treated as an ordinary M code.

## 17.6.5 Subprogram Call Using an M Code

By setting an M code number used to call a subprogram (macro program) in a parameter, the macro program can be called in the same way as with a subprogram call (M98).



```
O0001 ;
   :
M03 ;
   :
M30 ;
```

```
O9001 ;
   :
   :
   :
M99 ;
```

Parameter 6071 = 03

### Explanations

By setting an M code number from 1 to 255 used to call a subprogram in a parameter (6071 to 6079), the corresponding custom macro program (9001 to 9009) can be called in the same way as with M98.

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

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

- **Repetition**

As with a simple call, a number of repetitions from 1 to 9999 can be specified at address L.

- **Argument specification**

Argument specification is not allowed.

- **M code**

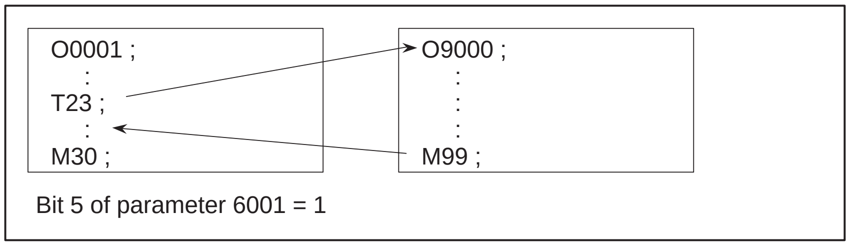
An M code in a macro program that has been called is treated as an ordinary M code.

### Limitations

In a macro called with a G code or in a program called with an M or T code, no subprograms can be called using an M code. An M code in such a macro or program is treated as an ordinary M code.

### 17.6.6 Subprogram Calls Using a T Code

By enabling subprograms (macro program) to be called with a T code in a parameter, a macro program can be called each time the T code is specified in the machining program.



#### Explanations

**● Call**

By setting bit 5 of parameter 6001 to 1, the macro program O9000 can be called when a T code is specified in the machining program. A T code specified in a machining program is assigned to common variable #149.

#### Limitations

In a macro called with a G code or in a program called with an M or T code, no subprograms can be called using a T code. A T code in such a macro or program is treated as an ordinary T code.

## 17.6.7 Sample Program

By using the subprogram call function that uses M codes, the cumulative usage time of each tool is measured.

### Conditions

- The cumulative usage time of each of tools T01 to T05 is measured. No measurement is made for tools with numbers greater than T05.
- The following variables are used to store the tool numbers and measured times:

| | |
|---|---|
| #501 | Cumulative usage time of tool number 1 |
| #502 | Cumulative usage time of tool number 2 |
| #503 | Cumulative usage time of tool number 3 |
| #504 | Cumulative usage time of tool number 4 |
| #505 | Cumulative usage time of tool number 5 |

- Usage time starts being counted when the M03 command is specified and stops when M05 is specified. System variable #3002 is used to measure the time during which the cycle start lamp is on. The time during which the machine is stopped by feed hold and single block stop operation is not counted, but the time used to change tools and pallets is included.

### Operation check

#### ● Parameter setting

Set 3 in parameter 6071, and set 05 in parameter 6072.

#### ● Variable value setting

Set 0 in variables #501 to #505.

#### ● Program that calls a macro program

```
O0001;
T01 M06;
M03;
G04 X20.0;
M05; . . . . . . . . . . . . . . . . . . . . . . . . . . Changes #501.
T02 M06;
M03;
G04 X20.0;
M05; . . . . . . . . . . . . . . . . . . . . . . . . . . Changes #502.
T03 M06;
M03;
G04 X20.0;
M05; . . . . . . . . . . . . . . . . . . . . . . . . . . Changes #503.
T04 M06;
M03;
G04 X20.0;
M05; . . . . . . . . . . . . . . . . . . . . . . . . . . Changes #504.
T05 M06;
M03;
G04 X20.0;
M05; . . . . . . . . . . . . . . . . . . . . . . . . . . Changes #505.
M30;
```

## Macro program (program called)

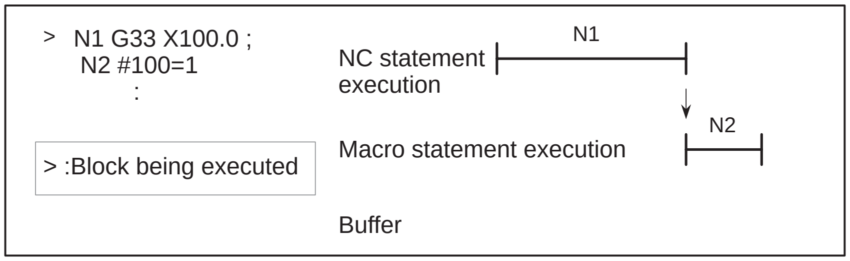
```
O9001(M03); . . . . . . . . . . . . . . . . . . . . . . . . . . Macro to start counting
  M01;
  IF[#4120 EQ 0]GOTO 9; . . . . . . . . . . . . . . . . . . . . . . No tool specified
  IF[#4120 GT 5]GOTO 9; . . . . . . . . . . . . . . . Out–of–range tool number
  #3002=0; . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . Clears the timer.
N9 M03; . . . . . . . . . . . . . . . Rotates the spindle in the forward direction.
  M99;

O9002(M05); . . . . . . . . . . . . . . . . . . . . . . . . . . . Macro to end counting
  M01;
  IF[#4120 EQ 0]GOTO 9; . . . . . . . . . . . . . . . . . . . . . No tool specified
  IF[#4120 GT 5]GOTO 9; . . . . . . . . . . . . . . . Out–of–range tool number
  #[500+#4120]=#3002+#[500+#4120]; . . . . Calculates cumulative time.

N9 M05; . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . Stops the spindle.
  M99;
```

## 17.7 PROCESSING MACRO STATEMENTS

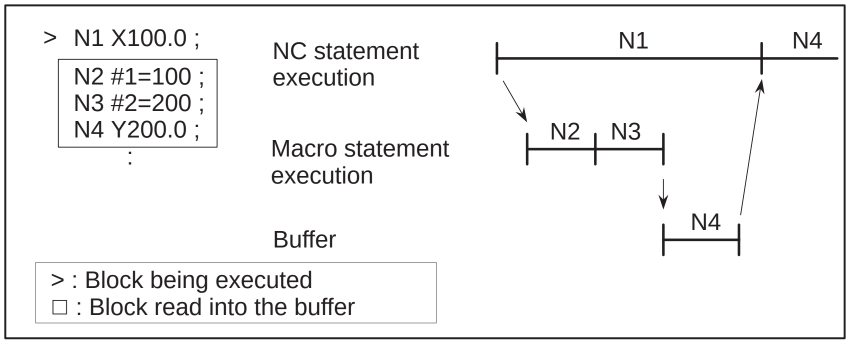
For smooth machining, the NC prereads the NC statement to be performed next. This operation is referred to as buffering. In cutter compensation mode (G41, G42), the NC prereads NC statements two or three blocks ahead to find intersections. Macro statements for arithmetic expressions and conditional branches are processed as soon as they are read into the buffer. Blocks containing M00, M01, M02, or M30, blocks containing M codes for which buffering is suppressed by setting parameters 3411 to 3420, and blocks containing G33 are not preread.

### Explanations

- **When the next block is not buffered (M codes that are not buffered, G33, etc.)**



- **Buffering the next block in other than cutter compensation mode (G41, G42) (normally prereading one block)**

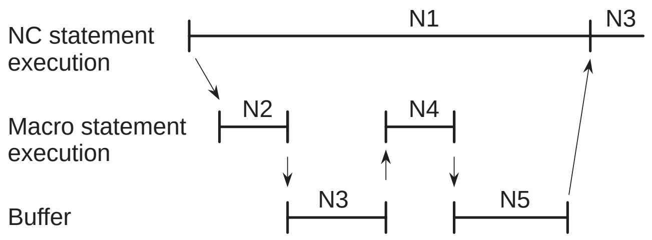


When N1 is being executed, the next NC statement (N4) is read into the buffer. The macro statements (N2, N3) between N1 and N4 are processed during execution of N1.

## • Buffering the next block in cutter compensation mode (G41, G42)

```
>  N1 G01 G41 G91 X50.0 Y30.0 F100 Dd ;
   N2 #1=100 ;
   N3 X100.0 ;
   N4 #2=200 ;
   N5 Y50.0 ;
        :
```

> : Block being executed
□ : Blocks read into the buffer

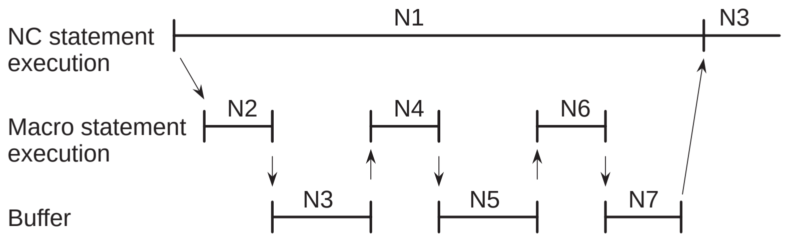


When N1 is being executed, the NC statements in the next two blocks (up to N5) are read into the buffer. The macro statements (N2, N4) between N1 and N5 are processed during execution of N1.

## • When the next block involves no movement in cutter compensation C (G41, G42) mode

```
>  N1 G01 G41 X100.0 G100 Dd ;
   N2 #1=100 ;
   N3 Y100.0 ;
   N4 #2=200 ;
   N5 M08 ;
   N6 #3=300 ;
   N7 X200.0 ;
        :
```

> : Block being executed
□ : Blocks read into the buffer



When the NC1 block is being executed, the NC statements in the next two blocks (up to N5) are read into the buffer. Since N5 is a block that involves no movement, an intersection cannot be calculated. In this case, the NC statements in the next three blocks (up to N7) are read. The macro statements (N2, N4, and N6) between N1 and N7 are processed during execution of N1.

## 17.8 REGISTERING CUSTOM MACRO PROGRAMS

Custom macro programs are similar to subprograms. They can be registered and edited in the same way as subprograms. The storage capacity is determined by the total length of tape used to store both custom macros and subprograms.

## 17.9 LIMITATIONS

- **MDI operation**

The macro call command can be specified in MDI mode. During automatic operation, however, it is impossible to switch to the MDI mode for a macro program call.

- **Sequence number search**

A custom macro program cannot be searched for a sequence number.

- **Single block**

Even while a macro program is being executed, blocks can be stopped in the single block mode (except blocks containing macro call commands, arithmetic operation commands, and control commands).
A block containing a macro call command (G65, G66, or G67) does not stop even when the single block mode is on. Blocks containing arithmetic operation commands and control commands can be stopped in single block mode by setting SBKM (bit 5 of parameter 6000) to 1.
Single block stop operation is used for testing custom macro programs. When SBKM (bit 5 of parameter 6000) is set to 1, a single block stop takes place at every macro statement.
Note that when a single block stop occurs at a macro statement in cutter compensation C mode, the statement is assumed to be a block that does not involve movement, and proper compensation cannot be performed in some cases. (Strictly speaking, the block is regarded as specifying a movement with a travel distance 0.)

- **Optional block skip**

A / appearing in the middle of an <expression> (enclosed in brackets [ ] on the right–hand side of an arithmetic expression) is regarded as a division operator; it is not regarded as the specifier for an optional block skip code.

- **Operation in EDIT mode**

Registered custom macro programs and subprograms should be protected from being destroyed by accident. By setting NE8 (bit 0 of parameter 3202) and NE9 (bit 4 of parameter 3202) to 1, deletion and editing are disabled for custom macro programs and subprograms with program numbers 8000 to 8999 and 9000 to 9999. When the entire memory is cleared (by pressing the RESET and DELETE keys at the same time to turn on the power), the contents of memory such as custom macro programs are deleted.

- **Reset**

When memory is cleared with a reset operation, local variables and common variables #100 to #149 are cleared to null values. They can be prevented from being cleared by setting, CLV and CCV (bits 7 and 6 of parameter 6001). System variables #1000 to #1133 are not cleared.
A reset operation clears any called states of custom macro programs and subprograms, and any DO states, and returns control to the main program.

- **Display of the PROGRAM RESTART page**

As with M98, the M and T codes used for subprogram calls are not displayed.

- **Feed hold**

When a feed hold is enabled during execution of a macro statement, the machine stops after execution of the macro statement. The machine also stops when a reset or alarm occurs.

- **Constant values that can be used in <expression>**

+0.0000001 to +99999999
–99999999 to –0.0000001
The number of significant digits is 8 (decimal). If this range is exceeded, alarm No. 003 occurs.

## 17.10 EXTERNAL OUTPUT COMMANDS

In addition to the standard custom macro commands, the following macro commands are available. They are referred to as external output commands.

- **BPRNT**
- **DPRNT**
- **POPEN**
- **PCLOS**

These commands are provided to output variable values and characters through the reader/punch interface.

### Explanations

Specify these commands in the following order:

**Open command: POPEN**
Before specifying a sequence of data output commands, specify this command to establish a connection to an external input/output device.

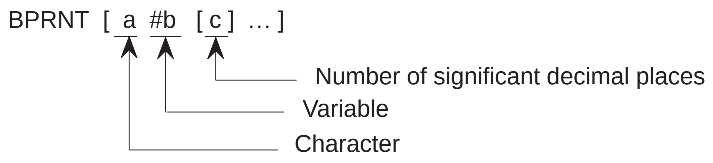
**Data output command: BPRNT or DPRNT**
Specify necessary data output.

**Close command: PCLOS**
When all data output commands have completed, specify PCLOS to release a connection to an external input/output device.

#### ● Open command POPEN

**POPEN**
POPEN establishes a connection to an external input/output device. It must be specified before a sequence of data output commands. The NC outputs a DC2 control code.

#### ● Data output command BPRNT



The BPRNT command outputs characters and variable values in binary.

(i) Specified characters are converted to corresponding ISO codes according to the setting (ISO) that is output at that time.
Specifiable characters are as follows:
- **Letters (A to Z)**
- **Numbers**
- **Special characters (*, /, +, –, etc.)**

An asterisk (*) is output by a space code.

(ii) All variables are stored with a decimal point. Specify a variable followed by the number of significant decimal places enclosed in brackets. A variable value is treated as 2–word (32–bit) data, including the decimal digits. It is output as binary data starting from the highest byte.

(iii) When specified data has been output, an EOB code is output according to the ISO code settings on the parameter screen.

(iv) Null variables are regarded as 0.

**Example)**

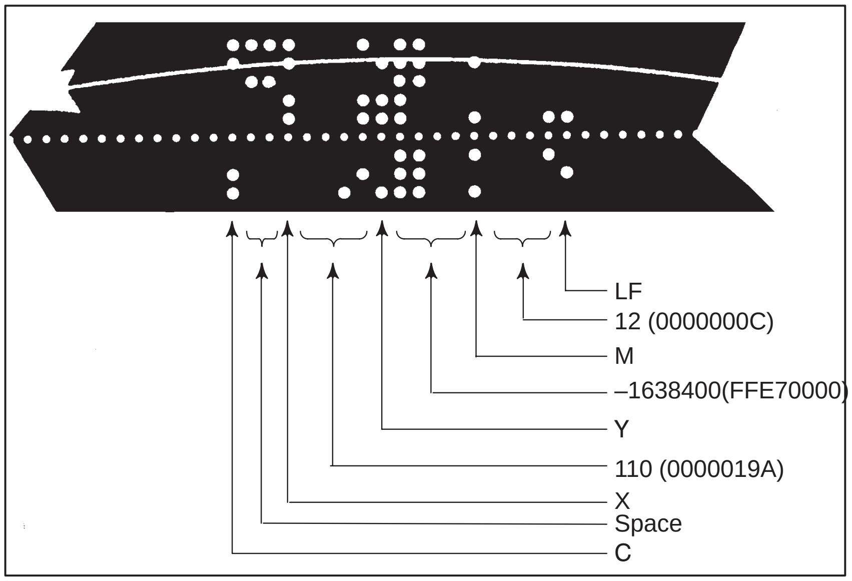
BPRINT [ C** X#100 [3] Y#101 [3] M#10 [0] ]

Variable value

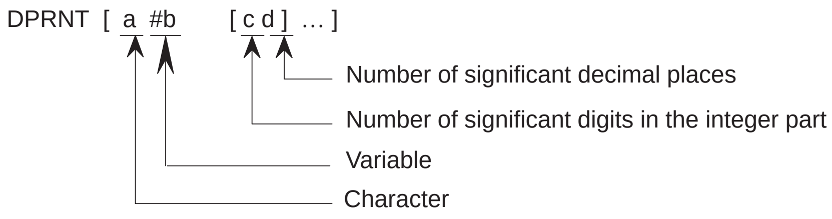
#100=0.40596

#101=–1638.4

#10=12.34



- **Data output command DPRNT**


DPRNT [ a #b [ c d ] … ]
Number of significant decimal places
Number of significant digits in the integer part
Variable
Character

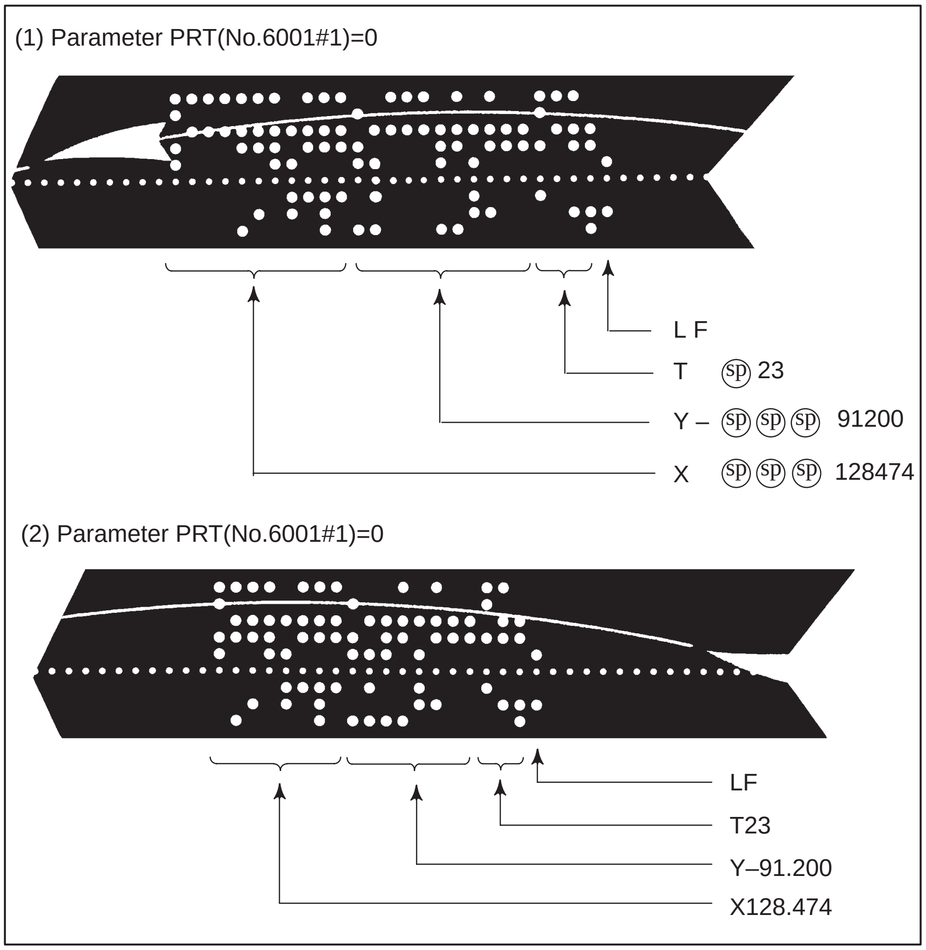
The DPRNT command outputs characters and each digit in the value of a variable according to the code set in the settings (ISO).

(i) For an explanation of the DPRNT command, see Items (i), (iii), and (iv) for the BPRNT command.

(ii) When outputting a variable, specify # followed by the variable number, then specify the number of digits in the integer part and the number of decimal places enclosed in brackets.
One code is output for each of the specified number of digits, starting with the highest digit. For each digit, a code is output according to the settings (ISO). The decimal point is also output using a code set in the settings (ISO).
Each variable must be a numeric value consisting of up to eight digits. When high–order digits are zeros, these zeros are not output if PRT (bit1 of parameter 6001) is 1. If PRT is 0, a space code is output each time a zero is encountered.
When the number of decimal places is not zero, digits in the decimal part are always output. If the number of decimal places is zero, no decimal point is output.
When PRT (bit 1 of parameter 6001) is 0, a space code is output to indicate a positive number instead of +; if PRT is 1, no code is output.

**Example)**

DPRINT [ X#2 [53] Y#5 [53] T#30 [20] ]
Variable value
#2=128.47398
#5=–91.2
#30=123.456



## ● Close command PCLOS

**PCLOS ;**

The PCLOS command releases a connection to an external input/output device. Specify this command when all data output commands have terminated. DC4 control code is output from the CNC.

## ● Required setting

Specify the channel use for parameter 020. According to the specification of this parameter, set data items (such as the baud rate) for the reader/punch interface.

**I/O channel 0 : Parameters 101 and 103**
**I/O channel 1 : Parameters 111 and 113**
**I/O channel 2 : Parameters 121 and 123**

Specify parameter 102, 112 or 122 so that the reader/punch interface is used as the output device for punching. (Never specify output to the Fanuc Cassette or floppy disks.)

When specifying a DPRNT command to output data, specify whether leading zeros are output as spaces (by setting PRT (bit 1 of parameter 6001) to 1 or 0).

To indicate the end of a line of data in ISO code, specify whether to use only an LF (NCR, of bit 3 of parameter 0103 is 0) or an LF and CR (NCR is 1).

**NOTE**

1 It is not necessary to always specify the open command (POPEN), data output command (BPRNT, DPRNT), and close command (PCLOS) together. Once an open command is specified at the beginning of a program, it does not need to be specified again except after a close command was specified.
2 Be sure to specify open commands and close commands in pairs. Specify the close command at the end of the program. However, do not specify a close command if no open command has been specified.
3 When a reset operation is performed while commands are being output by a data output command, output is stopped and subsequent data is erased. Therefore, when a reset operation is performed by a code such as M30 at the end of a program that performs data output, specify a close command at the end of the program so that processing such as M30 is not performed until all data is output.
4 Abbreviated macro words enclosed in brackets [ ] remains unchanged. However, note that when the characters in brackets are divided and input several times, the second and subsequent abbreviations are converted and input.
5 O can be specified in brackets [ ]. Note that when the characters in brackets [ ] are divided and input several times, O is omitted in the second and subsequent inputs.

## 17.11 INTERRUPTION TYPE CUSTOM MACRO

When a program is being executed, another program can be called by inputting an interrupt signal (UINT) from the machine. This function is referred to as an interruption type custom macro function. Program an interrupt command in the following format:

### Format

| | |
|---|---|
| M96 P○○○○ ; | Enables custom macro interrupt |
| M97 ; | Disables custom macro interrupt |

### Explanations

Use of the interruption type custom macro function allows the user to call a program during execution of an arbitrary block of another program. This allows programs to be operated to match situations which vary from time to time.

(1) When a tool abnormality is detected, processing to handle the abnormality is started by an external signal.

(2) A sequence of machining operations is interrupted by another machining operation without the cancellation of the current operation.

(3) At regular intervals, information on current machining is read.
Listed above are examples like adaptive control applications of the interruption type custom macro function.

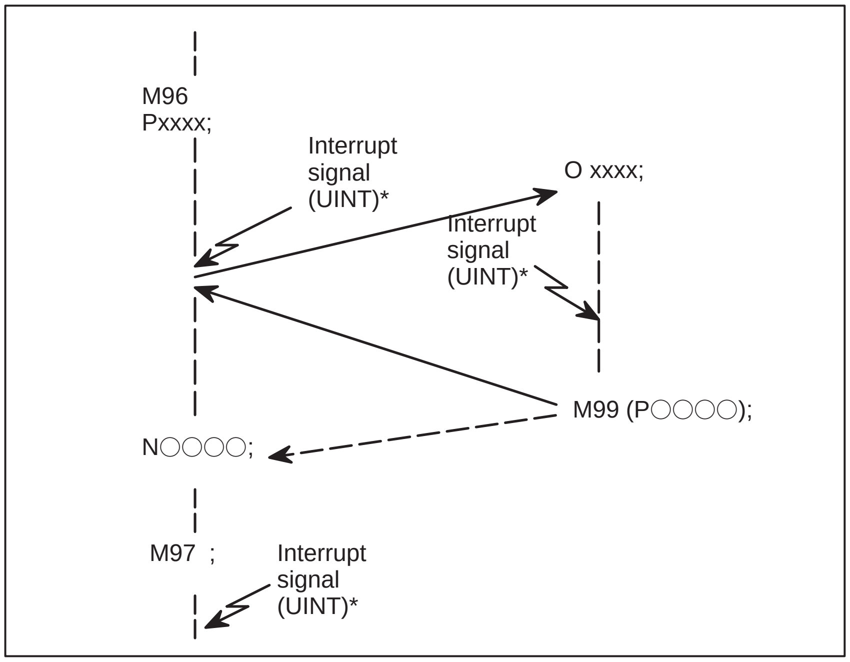


**Fig 17.11 Interruption type sustom macro function**

When M96Pxxxx is specified in a program, subsequent program operation can be interrupted by an interrupt signal (UINT) input to execute the program specified by Pxxxx.
When the interrupt signal (UINT, marked by * in Fig. 17.11 is input during execution of the interrupt program or after M97 is specified, it is ignored.

## 17.11.1 Specification Method

### Explanations

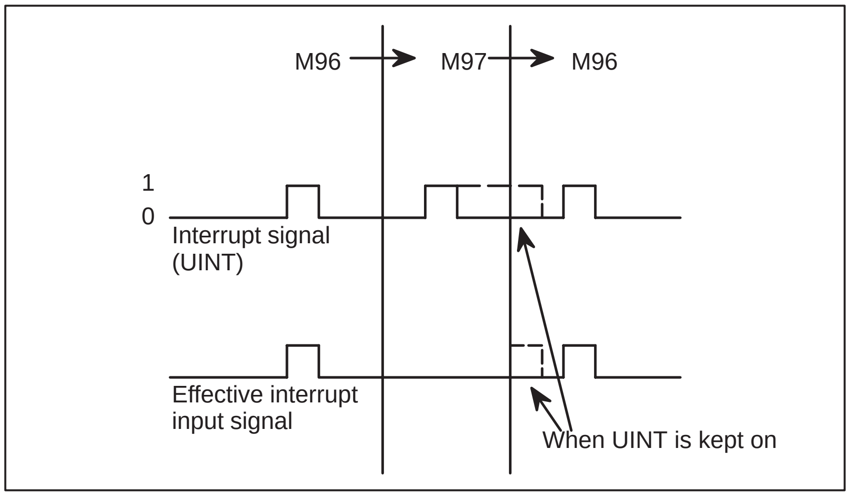
- **Interrupt conditions**

A custom macro interrupt is available only during program execution. It is enabled under the following conditions

**– When memory operation or MDI operation is selected**
**– When STL (start lamp) is on**
**– When a custom macro interrupt is not currently being processed**

- **Specification**

Generally, the custom macro interrupt function is used by specifying M96 to enable the interrupt signal (UINT) and M97 to disable the signal.
Once M96 is specified, a custom macro interrupt can be initiated by the input of the interrupt signal (UINT) until M97 is specified or the NC is reset. After M97 is specified or the NC is reset, no custom macro interrupts are initiated even when the interrupt signal (UINT) is input. The interrupt signal (UINT) is ignored until another M96 command is specified.



The interrupt signal (UINT) becomes valid after M96 is specified. Even when the signal is input in M97 mode, it is ignored. When the signal input in M97 mode is kept on until M96 is specified, a custom macro interrupt is initiated as soon as M96 is specified (only when the status–triggered scheme is employed); when the edge–triggered scheme is employed, the custom macro interrupt is not initiated even when M96 is specified.

**NOTE**
For the status–triggered and edge–triggered schemes, see Item “Custom macro interrupt signal (UINT)” of Subsec. 17.11.2.

## 17.11.2 Details of Functions

### Explanations

- **Subprogram–type interrupt and macro–type interrupt**

There are two types of custom macro interrupts: Subprogram–type interrupts and macro–type interrupts. The interrupt type used is selected by MSB (bit 5 of parameter 6003).

**(a) Subprogram–type interrupt**
An interrupt program is called as a subprogram. This means that the levels of local variables remain unchanged before and after the interrupt. This interrupt is not included in the nesting level of subprogram calls.

**(b) Macro–type interrupt**
An interrupt program is called as a custom macro. This means that the levels of local variables change before and after the interrupt. The interrupt is not included in the nesting level of custom macro calls. When a subprogram call or a custom macro call is performed within the interrupt program, this call is included in the nesting level of subprogram calls or custom macro calls. Arguments cannot be passed from the current program even when the custom macro interrupt is a macro–type interrupt.

- **M codes for custom macro interrupt control**

In general, custom macro interrupts are controlled by M96 and M97. However, these M codes, may already being used for other purposes (such as an M function or macro M code call) by some machine tool builders. For this reason, MPR (bit 4 of parameter 6003) is provided to set M codes for custom macro interrupt control.
When specifying this parameter to use the custom macro interrupt control M codes set by parameters, set parameters 6033 and 6034 as follows:
Set the M code to enable custom macro interrupts in parameter 6033, and set the M code to disable custom macro interrupts in parameter 6034.
When specifying that parameter–set M codes are not used, M96 and M97 are used as the custom macro control M codes regardless of the settings of parameters 6033 and 6034.
The M codes used for custom macro interrupt control are processed internally (they are not output to external units). However, in terms of program compatibility, it is undesirable to use M codes other than M96 and M97 to control custom macro interrupts.

- **Custom macro interrupts and NC statements**

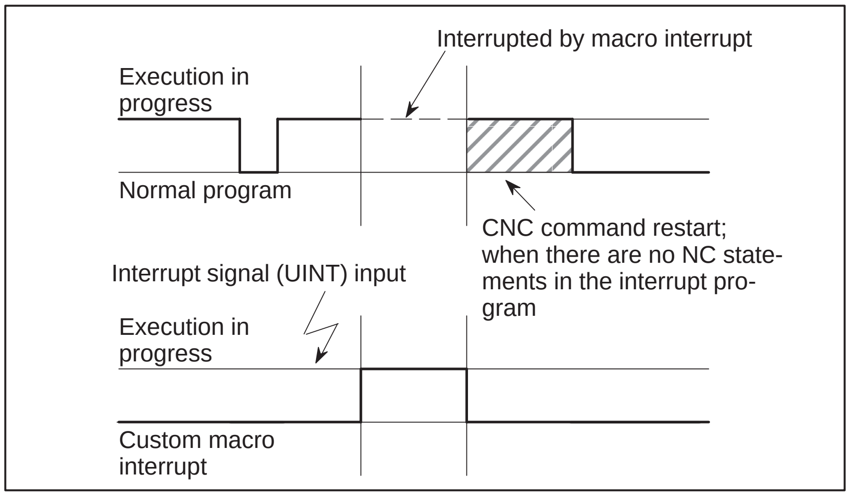
When performing a custom macro interrupt, the user may want to interrupt the NC statement being executed, or the user may not want to perform the interrupt until the execution of the current block is completed. MIN (bit 2 of parameter 6003)is used to select whether to perform interrupts even in the middle of a block or to wait until the end of the block.

**Type I**
**(when an interrupt is performed even in the middle of a block)**

(i) When the interrupt signal (UINT) is input, any movement or dwell being performed is stopped immediately and the interrupt program is executed.

(ii) If there are NC statements in the interrupt program, the command in the interrupted block is lost and the NC statement in the interrupt program is executed. When control is returned to the interrupted program, the program is restarted from the next block after the interrupted block.

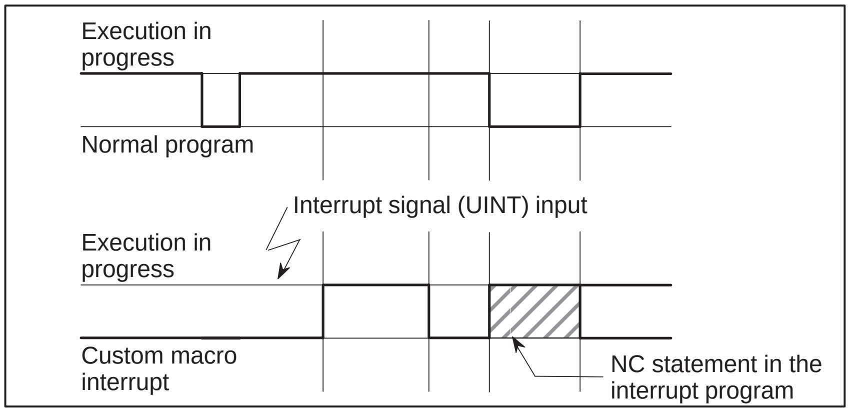
(iii) If there are no NC statements in the interrupt program, control is returned to the interrupted program by M99, then the program is restarted from the command in the interrupted block.


Interrupted by macro interrupt
Execution in progress
Normal program
CNC command restart; when there are no NC statements in the interrupt program
Interrupt signal (UINT) input
Execution in progress
Custom macro interrupt

**Type II**
**(when an interrupt is performed at the end of the block)**

(i) If the block being executed is not a block that consists of several cycle operations such as a pattern function and automatic reference position return (G28), an interrupt is performed as follows:
When an interrupt signal (UINT) is input, macro statements in the interrupt program are executed immediately unless an NC statement is encountered in the interrupt program. NC statements are not executed until the current block is completed.

(ii) If the block being executed consists of several cycle operations, an interrupt is performed as follows:
When the last movement in the cycle operations is started, macro statements in the interrupt program are executed unless an NC statement is encountered. NC statements are executed after all cycle operations are completed.



- **Conditions for enabling and disabling the custom macro interrupt signal**

The interrupt signal becomes valid after execution starts of a block that contains M96 for enabling custom macro interrupts. The signal becomes invalid when execution starts of a block that contains M97.
While an interrupt program is being executed, the interrupt signal becomes invalid. The signal become valid when the execution of the block that immediately follows the interrupted block in the main program is started after control returns from the interrupt program. In type I, if the interrupt program consists of only macro statements, the interrupt signal becomes valid when execution of the interrupted block is started after control returns from the interrupt program.

- **Custom macro interrupt during execution of a block that involves cycle operation**

**For type I**

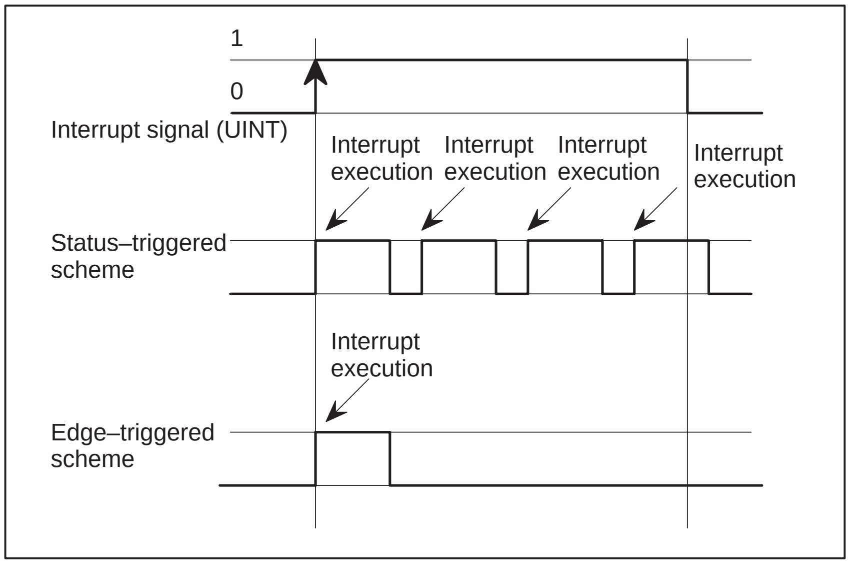
Even when cycle operation is in progress, movement is interrupted, and the interrupt program is executed. If the interrupt program contains no NC statements, the cycle operation is restarted after control is returned to the interrupted program. If there are NC statements, the remaining operations in the interrupted cycle are discarded, and the next block is executed.

**For type II**

When the last movement of the cycle operation is started, macro statements in the interrupt program are executed unless an NC statement is encountered. NC statements are executed after cycle operation is completed.

- **Custom macro interrupt signal (UINT)**

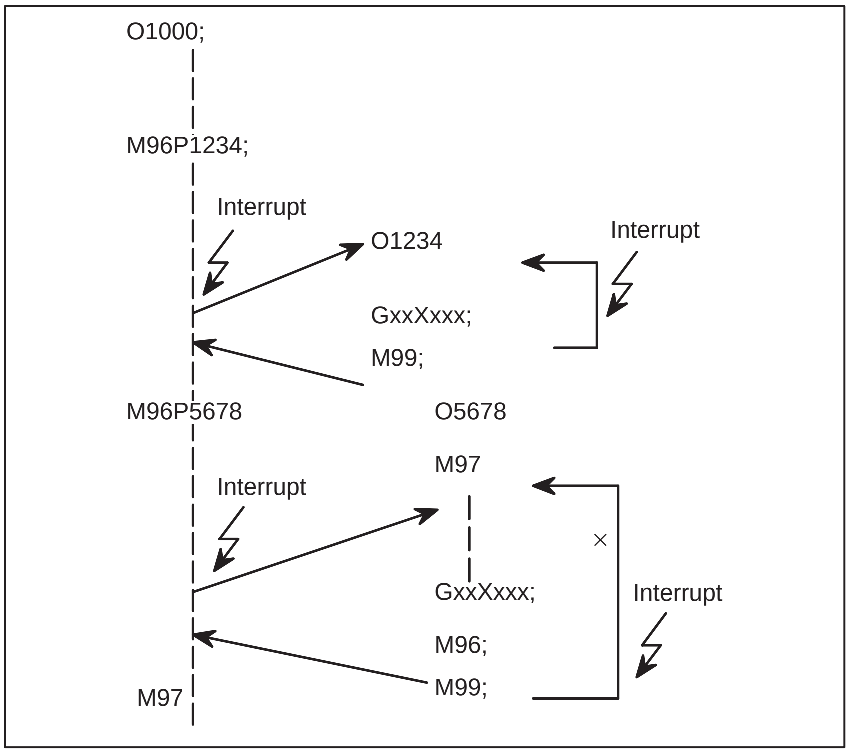
There are two schemes for custom macro interrupt signal (UINT) input: The status–triggered scheme and edge– triggered scheme. When the status–triggered scheme is used, the signal is valid when it is on. When the edge triggered scheme is used, the signal becomes valid on the rising edge when it switches from off to on status.
One of the two schemes is selected with TSE (bit 3 of parameter 6003). When the status–triggered scheme is selected by this parameter, a custom macro interrupt is generated if the interrupt signal (UINT) is on at the time the signal becomes valid. By keeping the interrupt signal (UINT) on, the interrupt program can be executed repeatedly.
When the edge–triggered scheme is selected, the interrupt signal (UINT) becomes valid only on its rising edge. Therefore, the interrupt program is executed only momentarily (in cases when the program consists of only macro statements). When the status–triggered scheme is inappropriate, or when a custom macro interrupt is to be performed just once for the entire program (in this case, the interrupt signal may be kept on), the edge–triggered scheme is useful.
Except for the specific applications mentioned above, use of either scheme results in the same effects. The time from signal input until a custom macro interrupt is executed does not vary between the two schemes.



In the above example, an interrupt is executed four times when the status triggered scheme is used; when the edge– triggered scheme is used, the interrupt is executed just once.

- **Return from a custom macro interrupt**

To return control from a custom macro interrupt to the interrupted program, specify M99. A sequence number in the interrupted program can also be specified using address P. If this is specified, the program is searched from the beginning for the specified sequence number. Control is returned to the first sequence number found.
When a custom macro interrupt program is being executed, no interrupts are generated. To enable another interrupt, execute M99. When M99 is specified alone, it is executed before the preceding commands terminate. Therefore, a custom macro interrupt is enabled for the last command of the interrupt program. If this is inconvenient, custom macro interrupts should be controlled by specifying M96 and M97 in the program.
When a custom macro interrupt is being executed, no other custom macro interrupts are generated; when an interrupt is generated, additional interrupts are inhibited automatically. Executing M99 makes it possible for another custom macro interrupt to occur. M99 specified alone in a block is executed before the previous block terminates. In the following example, an interrupt is enabled for the Gxx block of O1234. When the signal is input, O1234 is executed again. O5678 is controlled by M96 and M97. In this case, an interrupt is not enabled for O5678 (enabled after control is returned to O1000).



**NOTE**

When an M99 block consists only of address O, N, P, L, or M, this block is regarded as belonging to the previous block in the program. Therefore, a single–block stop does not occur for this block. In terms of programming, the following (1) and (2) are basically the same. (The difference is whether Gff is executed before M99 is recognized.)

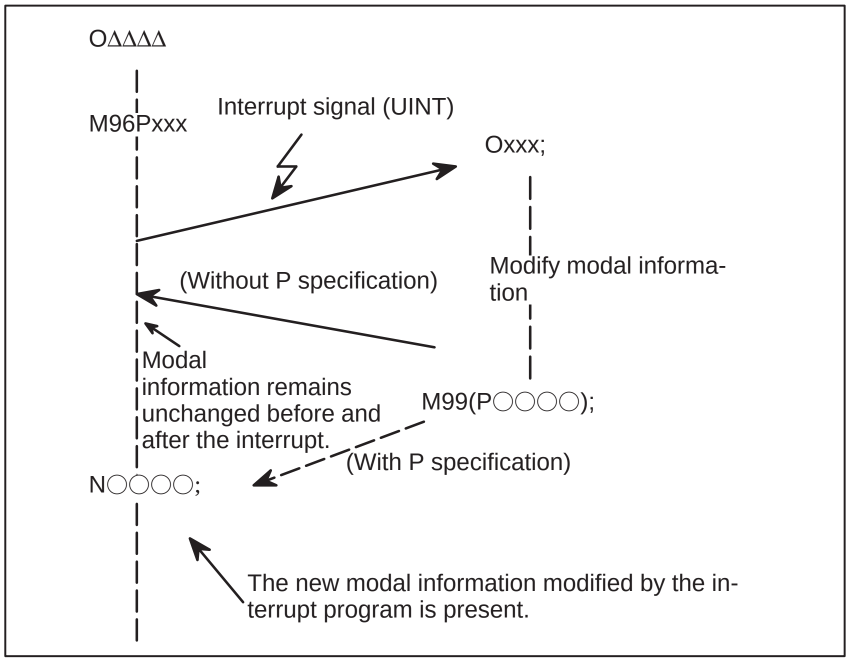
(1) G○○ X○○○ ;
M99 ;
(2) G○○ X○○○ M99 ;

- **Custom macro interrupt and modal information**

A custom macro interrupt is different from a normal program call. It is initiated by an interrupt signal (UINT) during program execution. In general, any modifications of modal information made by the interrupt program should not affect the interrupted program.
For this reason, even when modal information is modified by the interrupt program, the modal information before the interrupt is restored when control is returned to the interrupted program by M99.
When control is returned from the interrupt program to the interrupted program by M99 Pxxxx, modal information can again be controlled by the program. In this case, the new continuous information modified by the interrupt program is passed to the interrupted program. Restoration of the old modal information present before the interrupt is not desirable. This is because after control is returned, some programs may operate differently depending on the modal information present before the interrupt. In this case, the following measures are applicable:

(1) The interrupt program provides modal information to be used after control is returned to the interrupted program.

(2) After control is returned to the interrupted program, modal information is specified again as necessary.


OΔΔΔΔ
M96Pxxx
Interrupt signal (UINT)
Oxxx;
(Without P specification)
Modify modal information
Modal information remains unchanged before and after the interrupt.
M99(P○○○○);
(With P specification)
N○○○○;
The new modal information modified by the interrupt program is present.

**Modal information when control is returned by M99**

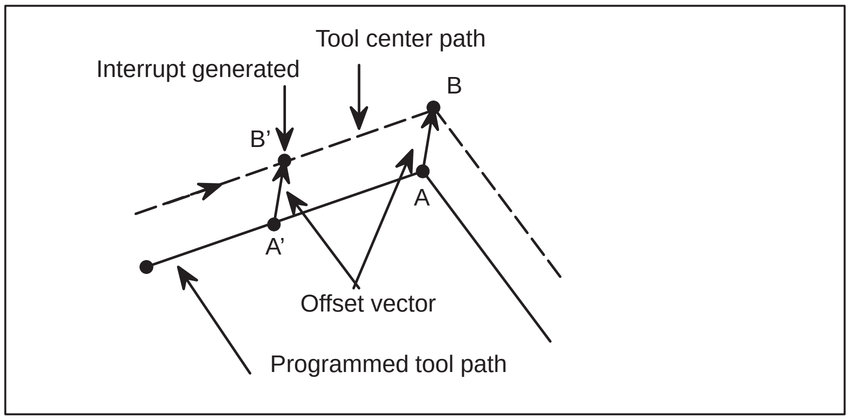
The modal information present before the interrupt becomes valid. The new modal information modified by the interrupt program is made invalid.

**Modal information when control is returned by M99 P○○○○**

The new modal information modified by the interrupt program remains valid even after control is returned. The old modal information which was valid in the interrupted block can be read using custom macro system variables #4001 to #4120.
Note that when modal information is modified by the interrupt program, system variables #4001 to #4120 are not changed.

- **System variables (position information values) for the interrupt program**

- The coordinates of point A can be read using system variables #5001 and up until the first NC statement is encountered.
- The coordinates of point A' can be read after an NC statement with no move specifications appears.
- The machine coordinates and workpiece coordinates of point B' can be read using system variables #5021 and up and #5041 and up.



- **Custom macro interrupt and custom macro modal call**

When the interrupt signal (UINT) is input and an interrupt program is called, the custom macro modal call is canceled (G67). However, when G66 is specified in the interrupt program, the custom macro modal call becomes valid. When control is returned from the interrupt program by M99, the modal call is restored to the state it was in before the interrupt was generated. When control is returned by M99Pxxxx;, the modal call in the interrupt program remains valid.

- **Custom macro interrupt and program restart**

When the interrupt signal (UINT) is input while a return operation is being performed in the dry run mode after the search operation for program restart, the interrupt program is called after restart operation terminates for all axes. This means that interrupt type II is used regardless of the parameter setting.