An Introduction To Parametric Programming

The CNC Custom Macro Language

By Michael W. Hubbard

Printing History

This manual was produced using Microsoft® Word for Windows 2.0a on CD-ROM. The graphics were created using AutoCad® Version 10 and imported as HPGL files. *Doc-To-Help*®, by WexTech Systems, Inc., was used to create the Microsoft® Windows Help file.

Version 3.0 January 1993

DISCLAIMER

WexTech Systems, Inc.
310 Madison Avenue, Suite 905
New York, NY 10017
(212) 949-9595

Contents

Quick Reference

Getting Started

Explaining the program examples

The Fanuc controls, except the 0 series, allow you to add comments to your program. The characters used for a comment are the open and close parentheses (). In this manual, curly braces are used to add comments that are included in the program. For example, you might see {FEED TO START POINT} in a program. You do not include this statement in your program. This is simply a comment in the manual.

Variable

A variable is a # sign followed by a number. For example, #1 represents variable 1. See "Variables" on page 12 for details.

Local Variable

A variable that is used by only one program. Local variables are #1 through #33. The letters of the alphabet can be used to transfer local variables between programs. Each nested level has it's own local variables. See "Local Variables" on page 13 for more information

Common Variable

A variable that is common to all programs. On the system 6/10/11 common variables #100-#149 are cleared when the power is turned off. On the system 15, parameter 7000 CLV (bit 6), allows #100-#149 to be cleared by pressing Reset. Variables #500-#549 are not cleared when power is turned off. On the system 10-15, #500-#519 can be named in the variable setting page. See "Common Variables" on page 15 for more information.

WHILE-DO

A section of program can be repeated using a while-do loop. For example:

```
WHILE[#33 LT 6]DO1;
PROGRAM STATEMENTS;
END1;
```

WHILE-DO loops can be nested 3 deep. Only 1, 2 or 3 can be used with the do statement. See "Program Iteration" on page 29 for more information.

IF-GOTO

An "IF-GOTO" statement can be used to branch when a variable reaches a preset value. For example: IF[#33 LT 5]GOTO100 will jump to line N100 when variable #33 is less than 5. See "Program Branching" on page 28 for more information.

IF THEN

The "GOTO" can be replaced when "THEN" on the system 10-15. For example: IF[#33EQ#0]THEN#3006=1(STOP). See the example "IF-THEN" on page 29 for more information.

Message Display

The System 10-15 can display a message on the CRT. #3006=1(16 CHARACTER MESSAGE) will stop program execution and display the message. See "Custom Macro Messages" on page 62 for more information.

Custom Alarms

A custom alarm can be displayed. For example: #3000=1(CUSTOM ALARM) will display "MC001 CUSTOM ALARM" on the CRT. The control is placed in the "alarm" state. See "Custom Macro Alarms" on page 63 for more information.

Introduction

Parametric Programming

What Custom Macro Can Do

The 0B and 0C controls have Custom Macro B available as an option. The 0A has a different type of macro language and is not covered.

The Fanuc Family of CNC controls, with the exception of the 0A, have an optional feature known as custom macro or user macro. This a powerful programming language that allows programs to be written using variables instead of fixed numbers. The language includes structures for conditional and unconditional program branching, repetitive looping and mathematical equations. If you are familiar with the BASIC language or FORTRAN for a PC, you will immediately be comfortable with Custom Macro.

Custom Macro is used with the standard "G" codes to enhance your flexibility when programming. With Custom Macro, you are no longer limited to writing a fixed tool path program. You can use mathematical and trigonometric functions to calculate end points. This makes family of part or complex shape programming possible. For example, if you needed to program a parabolic shape, Custom Macro is the obvious choice. You would write a loop that calculates the new end point for each axis and the program would do the rest.

If you used a Computer Aided Manufacturing (CAM) system to write the program, you can imagine how large it would be. The PC would have to write a line of code for each end point. If you use a resolution of even .001", the program would contain thousands of lines of code for a small shape. You would also need a new program for each size of parabola machined. With

Custom Macro, the program is changed at run time by changing the variables that define the equations.

But Custom Macro is not limited to tool path generation. The real power of Custom Macro is that it allows you to interact with the machine. For example, you can write a program that updates the coordinate system automatically or opens the RS232C port and outputs data while the program is running. If you want to add automatic clamping or inspection probes, you need to understand Custom Macro.

For example, you could use Custom Macro to write a bar pulling cycle on a lathe or a wheel dressing cycle on a grinder.

The program examples that follow make use of these structures to simplify patterns common to machining center work. However, Custom Macro can be used on any type of machine tool that has a Fanuc control. These programs reduce CNC programming effort by doing multiple calculations internally and reduce the chance for error while entering code. Custom macro structures make it easier for less experienced personnel to program efficiently.

Fanuc has made custom macro a general purpose programming language. A macro program is different from a simple sub program in many ways. When a branch is made to a sub program the same instructions are repeated regardless of what program or section of a program called for the sub program.

With a custom macro program, the instructions are written using variables and these variables can be changed each time the program is called. For example, if a program is stored in memory to drill a bolt circle, that program can be used with any main program to drill any size or pattern bolt circle. All that is required is for different variables to be defined when the sub program (macro) is called.

Therefore, one program can be left in memory and used anytime a bolt circle is required. This also eliminates the logistic problem of storing all of the sub programs for each part program.

A Canned Cycle is one line of code that results in several operations being completed A the G84 tapping cycle is an example of a Canned Cycle. You use one line of code to produce several operations.

The manner in which a macro is called is different from a simple sub program. A custom macro is called with a "G" code (G65) rather than an M98. If a macro program is used in a lot of main programs, you can define a "G" code to call the macro. This is easier than writing G65 P_ _ _ _ and gives the impression of a "Canned Cycle". More information on the nuts and bolts of custom macro can be found in the next section "Custom Macro Programming".

As with standard "G" code programming, your imagination is the biggest limit to the custom macro programs you can create. While reading the section on programming you will find features that don't appear to have any possible use. Try to keep these in the back of your mind, because someday a need might arise.

This page left blank intentionally

Custom Macro Programming

Using Variables

Variables

The difference between macro programs and normal CNC sub programs is the use of variables. Variables will seem confusing to most CNC programmers at first. However, all the various G codes and M codes used in programming probably seemed a bit confusing when you wrote your first CNC program.

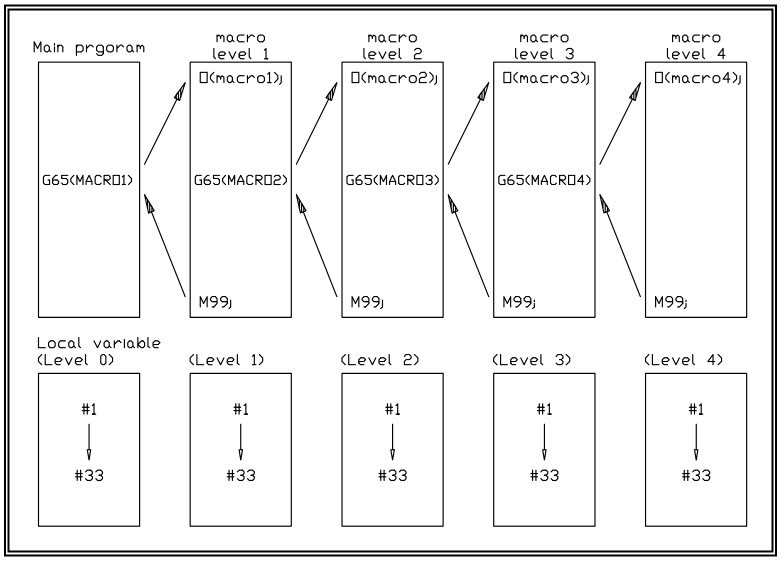
With a little patience and practice, custom macro programming will seem as natural as your normal CNC programming. A variable is just a place holder in the program. It occupies a space until given a value in your program. This allows a program to behave differently each time it is used.

Create a library of tooling programs to simplify setups. For example, the bolt circle program in the application section.

For example, a program can be written using variables to drill a bolt circle. To drill different bolt circle patterns, only the parameters for size, number of holes and starting angle would have to be changed. The parameters would be changed in the main program that called the macro. The variables in the macro program will change to produce the proper pattern.

Fanuc uses three different types of variables→local, common and system. All three types use a # symbol and a number to name the variable. For example, #1 is used to represent variable number one.

Local Variables



VARIABLE LEVELS

Local and System variables cannot be edited manually from the setting page like common variables can. You must set them using program code.

The local variables are #1 through #33. Custom macro programs can be nested four levels deep. The main program and each of the four macro levels has it's own set of local variables. When a macro is called with G65, the local variables of the current level are stored and #1 to #33 are prepared for the new level.

The local variables are automatically cleared for each level when the M99 code is read. Local variables are also cleared by M30, and pressing RESET. This reset feature makes the local variables useful as counters for looping because each time you call the program, they are automatically cleared. See the above figure.

Using local variables, the letters of the alphabet can be used to transfer data from one program to another. The ability to use letters allows you to give custom macro programs a "canned cycle" appearance. For example, to drill a hole with a "G81" drill cycle, the letters Z, R, and F are used to describe the drilling operation→G81 Z__ R__ F__ .

A custom macro can do the same thing. For example, G65 P1 Z-1.0 R.05 F15.0 would branch to program O0001 and set variable #26 to -1.0, #18 to .05 and #9 to 15.0. Naturally, a more complicated program than a G81 drill cycle would be used as the macro.

Not all the letters of the alphabet can be used. The letters G, L, O, N and P are reserved and cannot be used as local variables. Therefore, the number of the variable does not correspond exactly with the letter of the alphabet.

The following table lists the letter of the alphabet and the corresponding variable number:

Variable Address List

ADDRESS	**VARIABLE**	**ADDRESS**	**VARIABLE**
VACANT	#0	V	#22
A	#1	W	#23
B	#2	X	#24
C	#3	Y	#25
D	#7	Z	#26
E	#8	N/A	#10
F	#9	N/A	#12
H	#11	N/A	#14
I	#4	N/A	#15
J	#5	N/A	#16
K	#6	N/A	#27
M	#13	N/A	#28
Q	#17	N/A	#29
R	#18	N/A	#30
S	#19	N/A	#31
T	#20	N/A	#32
U	#21	N/A	#33

The variables #10, #12, #14, #15, #16 and #27-#33 can be used as local variables. Although there is not a letter to call them with, they can still be used in the program. They are still reset automatically. Any of the local variables that are not used with the G65 call are reset and available for use. For example, if:

```
G65 P1 F10.0;
```

were used, only variable #9 is transferred to the macro. Therefore #1 through #33 except #9 are reset and ready to use in the macro program. Variable #0 is always *VACANT*. This means that it has no value, not even the value zero. Why is this important? When a branch is made to a custom macro, local variables that are not used have the value of "VACANT," not zero. This allows you to determine if a variable has been transferred. See "*Custom Macro Alarms*" on page 63 for an example.

Common Variables

The second type of variable is the common variable. These are common to all macro programs and do not get reset when G65 is used. They can be defined in your program to whatever value you need or set in MDI mode through the "SETTING" page. The standard common variables are #100-#149 and #500-#549. On the system 10-15 controls, #100-#199 and #500-#749 are available as options.

Use SETVN to clarify the meaning of a variable. For example, if you are using inspection probes, use SETVN to identify the calibration variables.

For example, #100 can be defined as a 1 by using the program line #100=1. The common variables #100 through #149 are cleared when the power is turned off, #500 to #549 are not. The common variables #500 through #519 can be given a name on the "setting" page by using the SETVN command. This name is not cleared when the power is turned off. The following command given in Manual Data Input (MDI) mode will enter the name:

```
SETVN500[NAME,NAME1,NAME2,...NAME19];
```

where NAME is the name to be given to #500 and NAME19 is the name to be given to #519. All 20 names do not have to be entered at once. If SETVN501 is given, the names will begin at #501. The name can only be 8 characters long. If you use more than eight, an ILLEGAL FORMAT error will occur.

Variable Examples

When a macro program is executed, the variables are replaced by numbers. Local variables are transferred when G65 is executed and common variables are transferred from the common variable page. On the System 10-15 controls, the variable page is accessed through the "setting" mode. On the System 0, the variable page is accessed by pressing the "OFFSET" key

To set the common variables, first press the "SETTING" soft key. Next press the "CHAPTER" key twice. This is the soft key on right. The second soft key from the right will then be the "MACRO" key. With the mode selector set for MDI operation the common variables can be set. Variables can be combined in brackets to create equations. For example, if the sum of #100 and #101 is to be divided by #102, a new variable, #103 can be defined.

```
#103=[#100+#101]/#102;
```

Notice that parentheses are not used, brackets are. Parentheses are used to put comments in the program. Comments inside parentheses are ignored by the control. A number returned in a variable can be redefined as a variable address. If the value of #100 is 2403 and #143=#[#100] then #143 is equal to the value stored at #2403. This can be used to read the system variables that are available. System variables are discussed in section 1.13 on page 31. Any CNC address can be defined with a variable.

For example:

```
X#100  If #100=1.0 then the control will read X1.0.
G#101  If #101=2.0 then the control will read G2.
P#33   If #33=200  then the control will read P200.
```

When a variable is used as a CNC address, its value is rounded to fit the address. If #1 is 1.00993 and X#1 is read, it becomes X1.0099.

Macro Call Instructions

Macro programs can be called in several ways. The first is the "simple call." This is similar to a sub-program call in that a program number is given with a command and the program branches to that program. The simple call is of the form G65 P(program number) L(number of times to repeat) (the variables to set with this call).

For example, if your macro was program O9000 and you were going to set variable #9 to 6.0 the format would be G65 P9000 F6.0. This happens because F is the argument for the G65 command and F corresponds to variable #9. A sample program will help you understand what is going on. Suppose you stored a simple G81 drill cycle in a program using variables. It would look like this:

```
O0001; Any program number
G81 Z#26 R#18 F#9;
M99;
```

When you wanted to use this program the call would look like this:

G65 P1 Z-1.0 R.1 F5.0;

This would be interpreted by the control as:

G81 Z-1.0 R.1 F5.0

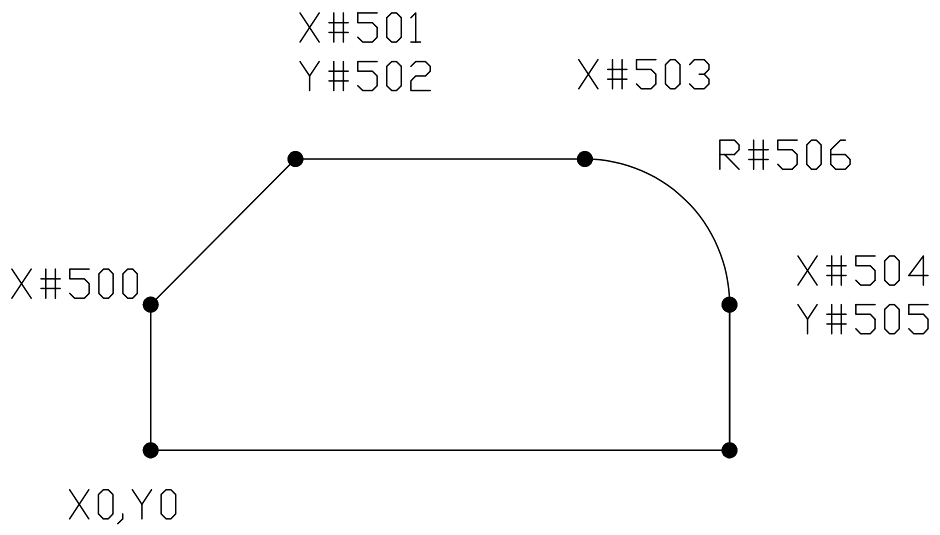
Naturally, you wouldn't make a program this simple because it doesn't save any writing. However, when combined with the other features that macro programs provide a considerable amount of time can be saved. A typical use for a tool path macro program would be a family of parts.

Suppose that you manufacture the widget shown in the next figure. The part is made in ten different sizes that all have the same general shape. Writing and storing 10 different programs would not be the best way to manufacture this family of parts. You could write one macro program, storing the end points as variables. Each time a new size is run only the variables have to be changed.

The following program will help explain.

```
O0001;program number.
N1T1; Select a tool.
N2M6; Change tools.
N3M3 S1000; turn on spindle.
N4G0G54G43H1X0Y-1.0Z0.0; position off the part.
N5G1Y#500 F5.0; feed to start of taper.
N6G1X#501 Y#502; feed up taper.
N7G1X#503; feed to start of radius.
N8G2X#504 Y#505 R#506; feed around radius.
N9G1Y0; feed to bottom of part.
N10G1X-.5; feed off the part.
N11G0G28Z0.0; return to home position.
N12M30; end program.
```

To machine the part, the coordinates of the end points would be stored in variables #500 through #506. To machine a different size part, simply change the variables in the common variable page. No program modification would be required.



WIDGET

Macro Call Using A "G" Code

A macro program can be called by a custom G code. The G code can be a number between 0 and 250. If a program is used often this is the best way to call it. For example, instead of G65 P9010(variables), the program could be called using the statement G100(variables).

How is this possible? It is actually very simple. Fanuc has made available 10 parameters to store macro call G codes in. Simply put the number you want to use as your G code in the proper parameter. When the program reads your G code it will then branch to the proper program. The program number of your program must correspond to the parameter where your G code is stored. The following table will clarify this:

SYSTEM 10-15		
PARAMETER		**PROGRAM NUMBER**
7050	G code to call macro program	9010
7051	G code to call macro program	9011
7052	G code to call macro program	9012
7053	G code to call macro program	9013
7054	G code to call macro program	9014
7055	G code to call macro program	9015
7056	G code to call macro program	9016
7057	G code to call macro program	9017
7058	G code to call macro program	9018
7059	G code to call macro program	9019
SYSTEM 6B		
0323	G code to call macro program	9010
0324	G code to call macro program	9011
0325	G code to call macro program	9012
0326	G code to call macro program	9013
0327	G code to call macro program	9014
0328	G code to call macro program	9015
0329	G code to call macro program	9016
0330	G code to call macro program	9017
0331	G code to call macro program	9018
0332	G code to call macro program	9019

For example, if you put a 100 in parameter 7050, the command G100(variables) will branch to program O9010. You cannot use any other program number. There is a one to one correspondence between the parameter number and the program it calls.

System 10-15 Macro Call Using M Codes

Hint: Use custom M codes to call macros that activate machine related functions. This makes your macro appear to be a standard machine function. Automated clamping and air blow macros are good candidates for custom M codes.

In addition, you can set M codes to call your macro programs. The parameters to store the number of your M code begin at 7080. The program called by a number at this parameter is 9020. As with the G codes you can define 10 M codes to call your macro programs. The program numbers will range from 9020 to 9029 and the parameter numbers will range from 7080 to 7089. The numbers, for the M codes, can range from 1 to 97. You would not want to use a code that is already in use by your machine to call a macro program. For example, don't use G01 or M01 as your custom call code.

Differences Between M98 and G65

There are several differences between simple sub-program calls and custom macro calls. However, a program written for a G65 call can still be called with an M98. The local variables will not be reset, so be careful. The following table lists the differences between the two:

1.) When an M98 block includes an axis move, execution stops after the block during single block operation, a G65 does not.
2.) G65 block can include arguments, M98 cannot.
3.) G65 block can only branch, M98 can branch after executing an axis move.
4.) G65 calls can be nested four levels deep.

Special Program Parameters System 10-15

Custom macro programs can be stored at any valid program number. However, if they are stored between O8000→O8999 and O9000→O9899 they can be protected from accidental editing and deletion.

Hint: This works for any program between these numbers, not just macros. Use this feature to protect your tooling programs.

This is accomplished by setting a parameter. For program numbers O8000→O8999 the parameter is 0011. The farthest bit to the right (bit 0) is set to a one. For numbers O9000→O9899 parameter 2201 bit 0 is set to a one. Once this bit is set to one, a program in this range cannot be edited or deleted. However, it can be downloaded through the RS232C port on the control.

If bit one (second from the right) of each of these parameters is set to a one the macro program will not be displayed when it is executed. Only the values it has calculated will be displayed. This is convenient after the program has been debugged and the macro statements do not need to be seen.

You can configure the control to suit yourself by changing parameters. It is a good idea to download the original parameters to a computer before making changes.

There are four more parameters that effect macro program operation. They are very handy while debugging a new program. Normally, macro statements are executed continuously even in single block mode. This means that a group of macro statements before a CNC statement are all executed with one push of "cycle start."

The control will stop when the CNC statement is encountered. When the proper parameter is set, each push of the "cycle start" button will only execute one macro statement. The following table lists the parameters and their function:

1.) 2201 bit 2 (third from right). If set to 1 all macro statements in programs numbered O9000 to O9999 are done in single block.

2.) 0010 bit 3 (fourth from right). If set to a 1 all macro statements in programs numbered O7000 to O7999 are done in single block.

3.) 0010 bit 4 (fifth) from the right) If set to a 1 all macro statements in programs numbered O8000 to O8999 are done in single block.

4.) 0010 bit 5 (sixth from the right). If set to a 1 all macro statements are done in single block.

Functions

The following functions can be used in a custom macro program:

MATHEMATICAL FUNCTIONS IN CUSTOM MACRO	
#1=SIN[#2]	Sine function in degree units.
#1=COS[#2]	Cosine function in degree units.
#1=TAN[#2]	Tangent function in degree units.
#1=ACOS[#2]	Arccosine function in degree units.
#1=ATAN[#2]/[#3]	Arctangent function in degree units.
#1=SQRT[#2]	Square root.
#1=ABS[#2]	Absolute value.
#1=LN[#2]	Natural logarithm.
#1=EXP[#2]	Exponent base e.
#1=ADP[#2]	Add a decimal point to the end of the number.
#1=ROUND[#2]	Round off. Above .5 next higher number.

MATHEMATICAL FUNCTIONS IN CUSTOM MACRO	
#1=FUP[#2]	Round up all decimals.
#1=FIX[#2]	Discard fractions. Ex. 5.5 becomes 5.0
#1=BIN[#2]	Convert from BCD to binary.
#1=BCD[#2]	Convert from binary to BCD.

Where #1, #2 and #3 are any variable or constant value. Notice that brackets are used around the variable/constant and not parentheses. Parentheses are used to place comments in a program and anything inside is ignored.

The trigonometric functions work as expected. If you wrote #100=SIN[30], .5 would be placed in #100. The absolute value, natural logarithm, exponent and square root are also identical to the same functions on a pocket calculator. However, the other functions need some more explanation.

ROUND

The round function works like a standard math function if the variable is used in a standard calculation. For example, if you had #100=ROUND[#101/#102] the result would be rounded down for fractions less than .5 and rounded up for fractions greater than .5. However, if you use round with an address (X, Y, Z), the result is rounded to the least increment of the machine. For example, if G1X[ROUND[#1]]; and the increment system is standard inch (.0001), the value of #1 will be rounded to 4 places. If #1 is 1.63857, the command becomes G1X1.6386. If #1 was used in an arithmetic calculation instead of a CNC address, the value would be 2.0.

ADP

The add decimal point command is used when local variables are transferred to a program. Normally, local variables are transferred in the least increment system of the machine. For example, G65P0001A1; would set #1 to .0001 in a standard inch system machine.

Hint: if you use the ADP function and then move the program to a different control, be sure to check parameter 7000 before using the program. If not, you can spend a lot of time trying to figure out why the program doesn't work.

This is inconvenient for values that are always integers like the number of holes in a bolt circle. With the ADP command, the value would be set to 1.0. This is much more convenient if the program is used a lot. However, Fanuc does not recommend using the ADP command. For ADP to function, bit CVA of parameter 7000 must be set to zero. If it is set to 1, ADP does not work. Because of this, the macro program might not work on all controls. For compatibility, use a decimal point with the address in the main program instead of the ADP command.

BCD

This command is used to output a BINARY CODED DECIMAL number through the system variable #1132. See the section on system variables on page 31 for more information on Input/Output.

The best way to understand BCD is to write a short program and watch the bits change. Use the PC/NC key and the PCDGN mode to view the bits. Be sure to set macro execution to single block, or you wont be able to see the change.

Binary coded decimal is a four bit representation of a decimal number. Each group of four bits can only represent the digits 0-9. Since four bits can represent up to decimal 15, not all bits can be 1 in a BCD representation. For example, to output a BCD 15 to variable #1132, use #1132=BCD[15]. Variable #1132 would be set to 00010101 instead of 1111. Binary Coded Decimal is used to output numbers to switches and other external devices. If you are not familiar with BCD, a digital electronics handbook will be helpful.

BIN

This command is used to read an input from the system variable #1032 and convert it to BCD format. For example, if #1032 is 00010101 (21 binary) and #101=BIN[#1032], #101 will be 15 instead of 21. Remember, a BCD digit is a group of four bits. Bits 0-3 represent the first digit and 4-7 are the second digit.

Be sure to set macro execution to single block, or you wont be able to see the change.

The above example might represent two external switches set to 1 and 5. The following program will provide an example of the BCD and BIN function. The system variables for Input/Output must be decoded and written in the ROM cartridge for this program to work.

See the section on system variables on page 31 for more information on Input/Output variables. The program first writes a BCD 15 to address #1132. This will set #1132 to 00011001. You can view this output in the PC diagnostic page. The program then outputs a decimal 16 to #1132. This sets #1132 to

00010000. Finally, #101 is set to 10 by the BIN[#1132] command. Remember, that BCD is four bits.

```
O0001;
#1132=BCD[15]; {00011001}
M0; { VIEW #1132}
#1132=16; {DEC. 16 OUT}
#101=BIN[#1132];
#1132=0; {RESET #1132}
M30;
```

FIX

This command returns the integer part of a real number. That is, the fractional part of the number is eliminated. For example, if you used the statement #100=FIX[3/2], #100 would be set to 1.0. Without the FIX, obviously #100 would be 1.5. This is a very useful command when writing roughing routines. For example, if you divide the depth of a pocket by the rough step depth you will get the number of rough cuts to make. However, the number will probably not be a whole number. Use the FIX command to clean up the result.

For example; assume you have a pocket that is .5" deep, a reference point of Z.05 and a roughing depth of .075. The number of rough cuts would be NUMBER OF CUTS = (.5 + .05)/.075 = 7.33. Use the FIX command to set the number of cuts to 7.0. This will prevent the roughing loop from over cutting.

FUP

FIX and FUP are very useful functions. It is worth your time to become proficient with them.

The FUP command is similar to FIX except the number is rounded to the next higher integer value. For example, if you used the statement #100=FUP[3/2], #100 would be set to 2.0. Without the FUP, obviously #100 would be 1.5. If you are not comfortable with these two commands, enter the following program and watch the variables change:

```
O0001;
#100=0
WHILE[#100LT2.1]DO1;
#101=FIX[#100];
#102=FUP[#100];
#100=#100+.1;
END1;
M30;
```

Arithmetic Functions

The four arithmetic functions, Addition, Subtraction, Division, Multiplication, as well as three logical functions are available. Constants and variables can be combined to make complex equations.

The priority of execution when more than one operation is performed is Function, Multiplication-Division, Addition-Subtraction. Brackets can be used to modify the priority of mathematical operations. The brackets can be nested five levels deep. A constant inside brackets without a decimal point is treated like it had a decimal point at it's right side. The following table lists the arithmetic functions available:

CUSTOM MACRO MATHEMATICAL OPERATORS	
#1=#2+#3	Sum of two numbers.
#1=#2-#3	Difference of two numbers.
#1=#2*#3	Product of two numbers.
#1=#2/#3	Quotient of two numbers.
#1=#2 AND #3	Logical product at each bit of the 32 bit data.
#1=#2 OR #3	Logical sum at each bit of the 32 bit data.
#1=#2 XOR #3	Exclusive OR at each bit of the 32 bit data.
#1=#2	Substitute one variable or constant for another.

Where #1, #2 and #3 are any variable or constant value.

The AND Logical Operator

The following is an example of the AND function. The AND function is the Boolean algebra logical product 1 x 1, not 1+1 equals two. It has the following truth table:

TRUTH TABLE FOR AN *AND* FUNCTION

2ND VARIABLE	1ST VARIABLE	FUNCTION VALUE
0	0	FALSE (0)
0	1	FALSE (0)
1	0	FALSE (0)
1	1	TRUE (1)

An IF statement can be made into a toggle switch using the AND function. IF[#1 AND 1 EQ 0] THEN #2=-#2. This statement will AND the variable #1(any variable can be used) with the constant 1. When the result is zero #2 will be changed to -#2. When will the result be zero?

Use the IF-THEN statement with the AND function to change directions in roughing routine or anywhere you need to change states.

The AND function compares each value bit for bit. The constant 1 will always be 31 zeroes and a 1 . The variable #1 will change depending on its current value. However, the first bit (the only 1 in the constant it is being compared to) will toggle between 1 and zero. That is, each time #1 increases, the first bit changes either to a 1 or a 0. That is because the numbers are binary, there are only ones and zeroes.

Decimal to Binary Conversion

The binary equivalent of 0→7(8 values in all) for a three bit number is shown in the next table:

DECIMAL VALUE	BINARY VALUE
0	000
1	001
2	010
3	011
4	100
5	101
6	110
7	111

As you can see, the first bit toggles between 0 and 1 each time a greater value is reached. Since the number 1 is a constant it does not change its binary value and the IF statement in the above example will toggle between true and false. The logical functions OR and XOR work in the same manner.

The OR Logical Operator

The following is a truth table for the OR function. The OR function is the Boolean algebra logical addition not 1+1 equals two. It has the following truth table:

TRUTH TABLE FOR AN *OR* FUNCTION		
2ND VARIABLE	1ST VARIABLE	FUNCTION VALUE
0	0	FALSE (0)
0	1	TRUE (1)
1	0	TRUE (1)
1	1	TRUE (1)

The XOR Logical Operator

The following is a truth table for the XOR function:

TRUTH TABLE FOR AN *XOR* FUNCTION		
2ND VARIABLE	1ST VARIABLE	FUNCTION VALUE
0	0	FALSE (0)
0	1	TRUE (1)
1	0	TRUE (1)
1	1	FALSE (0)

Program Branching

Two types of program branching, conditional and unconditional, are available. Unconditional branching is simply a "GOTO line number" statement. When the program reads this command it simply goes to the line number specified. There are no conditions on the jump. This command can be used to jump around a line of code. GOTO 200 would jump from the current line to line N200 regardless where N200 is in the program.

The condition statements are sometimes call Relational Operators in other manuals.

Conditional branching on the other hand, compares two values and branches if the condition is true. The command is of the form IF[condition]GOTO line number. The "line number" is the standard N number used in CNC programming. The following table lists the conditions that can be used:

Conditional Branching Comparisons	
#1 EQ #2	1st value equals 2nd value.
#1 NE #2	1st value does not equal 2nd value.
#1 GT #2	1st value is greater than the 2nd value.
#1 LT #2	1st value is less than the 2nd value.
#1 GE #2	1st value is greater than or equal to 2nd value.
#1 LE #2	1st value is less than or equal to 2nd value

Examples Of "IF-GOTO" Statements

IF[#1 EQ #2] GOTO 100; If #2 equals #1 go to line N100.

IF[#1 EQ 2.0] GOTO 100; If #1 equals 2.0 go to line N100.

IF[[#1+#2] EQ [#6-#5]] GOTO 100; Formulas can also be used in conditional statements.

The IF-GOTO statement can be replaced with an IF-THEN statement on the System 10-15.

IF[#3 EQ #1] THEN #100=10; If #3 equals #1, then #100 becomes 10.

The line number specified in the IF statement must be at the beginning of the line. Also, branching toward the top of the program takes longer than branching towards the bottom.

The lack of a local variable can also be checked. When a local variable is not defined in the G65 line it is considered "vacant." Variable #0 is always considered vacant.

If you want to know if a certain variable is called out in the G65 line you can use the IF command. For example, to check for #3 (C) use the statement IF[#3 EQ #0] GOTO line number;. If variable #3 is not vacant the line is ignored. If it is vacant the program will branch to the line number given. An alarm can be set based on this. See "Custom Macro Alarms" on page 63 for more information on alarms.

Program Iteration

A conditional statement can be used as a looping control also. Instead of an IF statement, a WHILE statement is used. Its form is:

```
WHILE[condition]DOm;   where m = 1, 2 or 3.
(program);
(program);
ENDm;
```

While the condition is false, the lines between WHILE and END are repeated. Depending on the version of software in your control, the WHILE statement can be abbreviated WH[ ]DO1. When the condition is true, control passes to the line following the END.

For example, if you wanted to set the common variables #100 to #120 to vacant, you could use the following loop:

```
WHILE[#33 LT 21]DO1;
#[100+#33]=#0;
#33=#33+1
END1;
```

This type of loop is used to make sure the common variables are set to vacant before your program uses them.

The WHILE[condition]DOm statements can be nested as long as they don't overlap. That is, you can open WHILE[condition]DO1 and then open WHILE[condition]DO2 as long as you END2 before you END1.

WHILE-DO Example I

This example will have an outer loop that repeats 5 times and an inner loop that repeats 30 times.

```
#100=0
#101=0
WHILE[#100 LT 5]DO1
(program)
(program)
WHILE[#101 LT 30]DO2
(program)
(program)
#101=#101+1
END2
#100=#100+1
END1
M30
```

The WHILE[condition] can be omitted. If DO1 is used with END1 the loop will repeat indefinitely. An IF statement can be used to jump out of the loop but you cannot jump inside a DO loop.

WHILE-DO Example II

A WHILE DO statement can be used to machine multiple parts without using sub-programs. A variable is initialized to zero and the number of parts to be machined is set as a limit. In this example six parts will be machined using G54-G59.

```
O9000; program number.
N1 T1; select a tool.
N2 M6; tool change.
N3 M3 S1000; turn on spindle.
N4 G0 G54 G43 H1 X0 Y0 Z.1; position to a start point.
N5 #33=0; initialize #33 to zero.
N6 WHILE[#33 LT 6] DO1; set for six parts.
N7 G0 G[54+#33] X0 Y0; select G54 plus #33.
N8 G98G81Z-1.0R.05F8.0; drill a hole.
N9 X1.0; drill next location.
N10 #33=#33+1; Increment counter to next number.
N11 END1; complete the loop;
N12 G0 G28 Z.1; return home.
N13 M30; end program.
```

The first part will be machined using the G54 system, because #33 is zero. After the part is finished, line N10 increments #33 to 1. The next part will be machined at G[54+1] or G55. Again, the counter will be incremented and the next part will be machined at G54+2 or G56. After the sixth part (G59) #33 will increment to 6. The loop will end and control will pass to line N12.

This eliminates the need to change coordinate systems and branch to a sub-program five times. This makes the program easier to prove out and troubleshoot because you do not have to look at two programs. If you need to machine more parts than you have coordinate systems, see *Multiple Parts Using G52* on page 61.

System Variables

System variables allow you to determine information such as current machine position, current absolute position, skip signal position and work coordinate offsets. In addition, tool length offset, current and preceding "G", "M", "S", "T" and "H" code and other machine related information can be read or written to.

Use this feature of Custom Macro to automate your machine tool.

Programmable inputs/outputs can be added using systems variables. If you have ever wanted to automatically activate clamps or work rests, the programmable outputs are very useful. The programmable inputs can be used to verify that the clamps were activated. If a limit switch is used, you can verify that the part is in the fixture properly. For more information on I/O, see "System Variables for Programmable Inputs/Outputs" on page 46.

The following table is an overview of the system variables available on the System 10-15. The System 0B/C and 6M are similar. The biggest difference is the #3006 message variable. Neither of these controls have this capability. Any macros in the manual that use the #3006 variable must be modified by using an M00 instead of #3006 when used on a 0/6 control.

OVERVIEW OF SYSTEM VARIABLES

VARIABLE NUMBER	VARIABLE NUMBER
#1000-#1035 DI (User supplied inputs to CNC)	#5041-#5055 Work coordinates
#1100-#1135 DO (programmable outputs from CNC)	#5061-#5075 Skip signal position
#2000-#2999 (tool compensation variables)	#5081-#5095 Tool length offset value
#3000 Macro alarm (halts control in alarm condition)	#5101-#5115 Servo deviation
#3006 Macro message (halts operation W/O alarm)	#5201-#5215 Work offset value (common to G54-G59)
#3001 System clock (reset at power on. Always counting)	#5221-#5235 Work offset value for G54 coordinate system
#3002 STL (cycle start) clock (Not reset at power on)	#5241-#5255 Work offset value for G55 coordinate system
#3003 Single block/auxiliary function signal control	#5261-#5275 Work offset value for G56 coordinate system
#3004 Feed Hold/Feed Rate/Exact Stop	#5281-#5295 Work offset value for G57 coordinate system
#3007 Mirror image status	#5301-#5215 Work offset value for G58 coordinate system
#4001-#4120 Modal information	#5321-#5235 Work offset value for G59 coordinate system
#5001-#5015 Block end position	#7001-#7955 Additional work offset values of G54.1P1 through G54.1P48 (48 additional coordinate systems)
#5021-#5035 Machine coordinates	

Its good practice to return the control to its' previous state if the macro is a sub program that changes the state of modal codes.

When branching to a macro program, you might want to store the current state of the modal codes in the machining program. This could be necessary if your macro is positioning to load a pallet or doing anything that might change a modal code that would affect the machining program. When the macro completes, use the stored data to return the control to the previous state.

Program Code Variables-Preceding Block

System variables are available for any code used in the CNC program. In addition, there are system variables for the sequence number, current program number and the number of an additional coordinate system (G54.1P*). The "G" code groups are listed in the Fanuc manual under PREPARATORY

FUNCTION. A partial listing of the "G" code groups follows the table for system variables. The following table lists the system variables for System 10-15 preceding block modal information:

SYSTEM VARIABLE	PRECEDING BLOCK
#4001	G code for group 1
#4002	G code for group 2
...	...
#4021	G code group 21
#4102	B code
#4107	D code
#4108	E code
#4109	F code
#4111	H code
#4113	M code
#4114	Sequence number
#4115	Program number
#4119	S code
#4120	T code
#4130	Additional coordinate system

Program Code Variables-Current Block

The following table lists the system variables for System 10-15 Current block modal information:

SYSTEM VARIABLE	CURRENT BLOCK
#4201	G code for group 1
#4202	G code for group 2
...	...
#4221	G code for group 21
#4302	B code
#4307	D code
#4308	E code
#4309	F code
#4311	H code
#4313	M code
#4314	Sequence number

SYSTEM VARIABLE	CURRENT BLOCK
#4315	Program number
#4319	S code
#4320	T code
#4330	Additional coordinate system

Common G codes and Their Group

G code groups G00-G30

G code	Group	G code	Group
G00	01	G19	02
G01	01	G20	06
G02	01	G21	06
G03	01	G22	04
G04	00	G23	04
G09	00	G25	24
G10	00	G26	24
G15	17	G27	00
G16	17	G28	00
G17	02	G29	00
G18	02	G30	00

G code groups G33-G68

G code	Group	G Code	Group
G33	01	G55	14
G37	00	G56	14
G40	07	G57	14
G41	07	G58	14
G42	07	G59	14
G43	08	G60	00
G44	08	G61	15
G52	00	G65	00
G53	00	G66	12
G54	14	G67	12
G54.1	14	G68	16

G code groups G69-G99

G code	Group	G Code	Group
G69	16	G87	09
G73	09	G88	09
G73	09	G89	09
G76	09	G90	03
G80	09	G91	03
G81	09	G92	00
G82	09	G94	05
G83	09	G95	05
G84	09	G97	13
G85	09	G98	10
G86	09	G99	10

System Variables for Fixture Offsets G54-G59

The system variables for fixture offsets are given in the next table. These variables can be read and the information used to calculate a move distance or they can be assigned a value to set a new coordinate system. This is how spindle mounted probes can adjust the center of a part or update a tool length offset.

Combine multi-part vises & air blowers with manual pallets to increase production and lower cost. Use WHILE-DO to program the multiple parts.

If you are using a pallet changer, the variables can be stored at the beginning of the program. Each time the pallet is switched, the proper values are written to the work coordinate systems. This simplifies the programming and setup. Each pallet is treated as the only setup on the machine. The following table lists the system variables for work coordinate systems G54-G59. See "Fixture Indicating" on page 39 for an example program that shows how to set a new fixture offset.

SYSTEM VARIABLE	CONTENTS	COORDINATE SYSTEM
#5221 #5222 ... #5235	FIXTURE OFFSET OF 1ST AXIS FIXTURE OFFSET OF 2ND AXIS ... FIXTURE OFFSET OF 15TH AXIS	WORK COORDINATE VARIABLES FOR G54
#5241 #5242 ... #5255	FIXTURE OFFSET OF 1ST AXIS FIXTURE OFFSET OF 2ND AXIS ... FIXTURE OFFSET OF 15TH AXIS	WORK COORDINATE VARIABLES FOR G55
#5261 #5262 ... #5275	FIXTURE OFFSET OF 1ST AXIS FIXTURE OFFSET OF 2ND AXIS ... FIXTURE OFFSET OF 15TH AXIS	WORK COORDINATE VARIABLES FOR G56
#5281 #5282 ... #5295	FIXTURE OFFSET OF 1ST AXIS FIXTURE OFFSET OF 2ND AXIS ... FIXTURE OFFSET OF 15TH AXIS	WORK COORDINATE VARIABLES FOR G57
#5301 #5302 ... #5315	FIXTURE OFFSET OF 1ST AXIS FIXTURE OFFSET OF 2ND AXIS ... FIXTURE OFFSET OF 15TH AXIS	WORK COORDINATE VARIABLES FOR G58
#5321 #5322 ... #5335	FIXTURE OFFSET OF 1ST AXIS FIXTURE OFFSET OF 2ND AXIS ... FIXTURE OFFSET OF 15TH AXIS	WORK COORDINATE VARIABLES FOR G59

System Variables for Additional Fixture Offsets G54.1 P*

There are 48 additional sets of fixture offsets available. These are accessed by the system variables #7001-#7955.

System Variable	Contents	Coordinate System
#7001 #7002 ... #7015	FIXTURE OFFSET OF 1ST AXIS FIXTURE OFFSET OF 2ND AXIS ... FIXTURE OFFSET OF 15TH AXIS	1ST ADDITIONAL WORK COORDINATE SYSTEM G54.1P1
#7021 #7022 ... #7035	FIXTURE OFFSET OF 1ST AXIS FIXTURE OFFSET OF 2ND AXIS ... FIXTURE OFFSET OF 15TH AXIS	1ST ADDITIONAL WORK COORDINATE SYSTEM G54.1P2
... ...	FIXTURE OFFSET OF 1ST AXIS FIXTURE OFFSET OF 2ND AXIS ... FIXTURE OFFSET OF 15TH AXIS	1ST ADDITIONAL WORK COORDINATE SYSTEM G54.1P*
#7941 #7942 ... #7955	FIXTURE OFFSET OF 1ST AXIS FIXTURE OFFSET OF 2ND AXIS ... FIXTURE OFFSET OF 15TH AXIS	1ST ADDITIONAL WORK COORDINATE SYSTEM G54.1P48

Not all System 10-15 controls have the additional coordinate systems. If the control does not have the option, and you try to read or write the variable, the error message "VARIABLE OUT OF RANGE" will occur.

If you need to use the additional coordinate systems, contact your machine tool builder. If you need to run more than six parts and don't have the additional coordinates, see the G52 section on Page 61.

The following table lists the variables for absolute position of the preceding block and machine position of the current block. In addition, it is possible to read the variables for the absolute position of the current block and the skip signal position.

Position Variables System 10-15

SYSTEM VARIABLE	CONTENTS	READ WHILE MOVING
#5001	BLOCK END PT ABS. POSITION 1ST AXIS	YES
#5002	BLOCK END PT. ABS. POSITION 2ND AXIS	YES
...	...	YES
#5015	BLOCK END PT. ABS. POSITION 15TH AXIS	YES
#5021	PRESENT MACH. POSITION 1ST AXIS	NO
#5022	PRESENT MACH. POSITION 2ND AXIS	NO
...	...	NO
#5035	PRESENT MACH. POSITION 15TH AXIS	NO
#5041	PRESENT ABSOLUTE POS. 1ST AXIS	NO
#5042	PRESENT ABSOLUTE POS. 2ND AXIS	NO
...	...	NO
#5055	PRESENT ABSOLUTE POS. 15TH AXIS	NO
#5061	SKIP SIGNAL POSITION 1ST AXIS	YES
#5062	SKIP SIGNAL POSITION 2ND AXIS	YES
...	...	YES
#5075	SKIP SIGNAL POSITION 15TH AXIS	YES

The absolute position coordinate end point of the preceding block (ABSIO) can be read with system variables #5001 to #5015. #5001 is the first axis and #5015 is the fifteenth axis. This is the programmed position that the axis will move to. This variable can be read while the axis is moving.

The current machine position coordinate (ABSMT) can be read using #5021 to #5035. This is the actual value for the current machine position of the axis. This variable cannot be read while the axis is moving.

The current absolute coordinate (ABSOT) can be read with system variable #5041 to #5055. Again, this is the current position of the axis and cannot be read while the axis is moving.

The G31 skip position used during probing can be read using system variable #5061 to #5075. The System 6M/10M can only have four axes, System 10M/11M five axes and System 12M fifteen axes. System variables for position are read only and cannot be used on the left side of an arithmetic equation.

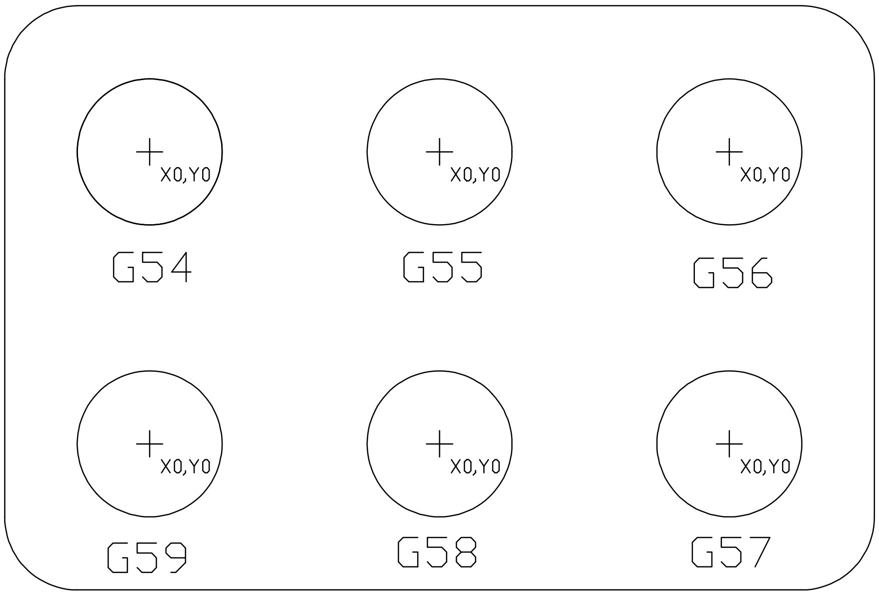
Fixture Indicating Program Example

This example will use system variables to update the coordinates of new fixtures in the G54-G59 fixture offsets. This program will automatically reset the coordinates of a new set of bosses on a six part fixture. After positioning over the old zero point you want to indicate the new boss and update the proper coordinate system (G54-G59). This is easy to do with system variables. For example:

```
O0001(INDICATING PROGRAM);
G90G80G40G49G17G20;
N1G0G54G43H1X0Y0Z3.3;
#103=#5003; {STORE CURRENT Z POINT}
#33=1; (INITIALIZE LOOP COUNTER TO 1}
#130=20; {INITIALIZE COORDINATE COUNTER TO 20}
#3006=1(INDICATE);
#104=#5003; {STORE MANUALLY MOVED Z POINT}
#5221=#5021; {SETS NEW X COORDINATE}
#5222=#5022; {SETS NEW Y COORDINATE}
G0Z#103; {RETURN TO INITIAL POINT}
WHILE[#33LT6]DO1; {SET LOOP FOR SIX FIXTURES}
G0G[54+#33]X0Y0; {POSITION TO NEXT FIXTURE ZERO}
G0Z[#104+.3]; {RAPID ABOVE BOSS}
G1Z#104F10.0; {FEED THE INDICATOR TO THE BOSS}
#3006=1(MANUALLY INDICATE FIXTURE);
#[5221+#130]=#5021; {(RESETS X COORDINATE}
#[5222+#130]=#5022; {RESETS Y COORDINATE}
G0Z#103; {RAPID TO INITIAL POINT}
#130=#130+20.0; {INCREMENT COUNT TO NEXT COORD. SYS}
#33=#33+1; {INCREMENT COUNTER}
END1; {END THE LOOP}
#3006=1(REMOVE INDICATOR);
G0G91G28Z2.0M9;
M30;
```

On the System 0 and 6, you have to change the #3006= lines to M00. These two controls do not have the Message capability.

When this program is executed, the machine rapids to the X0, Y0 of the old boss, and the "Z" axis point set in line N1. The #3006=1(INDICATE) halts the program and displays the prompt "INDICATE" on the CRT screen (see "custom macro messages"). It does not matter what current screen is displayed. The screen will change to the "**OPERATOR MESSAGE**" screen.



INDICATING EXAMPLE

If you are using this program on a System 6 or 0, remember to change the #3006 to M00. You will not see the message to set the tool on these controls. You then mount your indicator and manually indicate the new boss. The Z axis position that is dialed becomes the new "Z0".

The X and Y axis can be moved any amount necessary. Then instead of getting the new location and manually entering it into the proper coordinate system, you simply go back into "memory" and press cycle start.

The current machine position is loaded into the proper coordinates, the head rapids to the initial point and moves to the next boss. The head then rapids to .3" above the point set manually at the first boss. It feeds the final .3" to protect the indicator.

After the last boss is indicated, the #3006=1(REMOVE INDICATOR) causes the screen to change to the "OPERATOR MESSAGE" screen and display the message. The first time you use this program is scary, but it saves time after you are familiar with it. This program could be easily modified to use an edge finder instead of an indicator.

The system variables for work offset can be written to in a program. This can simplify the programming and setup of multiple parts with a pallet changer. The indicating program can be modified to load the machine position into a variable instead of the work offset. You simply change the statements that set the work coordinates to the variable you want to use. For example if you wanted to use variables starting at #500 you would change the statements

```
#[5221+#130]=#5021; {RESETS X COORDINATE}
#[5222+#130]=#5022; {RESETS Y COORDINATE}
```

to

```
#[500+#130]=#5021
#[501+#130]=#5022
```

This will indicate the parts in and load the machine position into variables #500 through #517. The advantage to doing this is that the program does not have to be modified each time the job is run. If the work offsets are slightly different from run to run, the variable values change but the program doesn't. The following program is an example of using variables to load work offsets.

```
%
O0038(SIDE COVER);
(////TOOL-1*.560-STUB DRILL////);
#3006=1(PALLET-#2);
#5221=#500(G54X);
#5222=#501(G54Y);
#5223=#503(G54Z);
#5241=#504(G55X);
#5242=#505(G55Y);
#5243=#506(G55Z);
#5261=#507(G56X);
#5262=#508(G56Y);
#5263=#509(G56Z);
G90M42G80G17G20G49G54;
M3S1500;
G0G54G43H7X-2.254Y2.254Z2.0M8;
#33=#518;
#3002=0;
WHILE[#33LT3.0]DO1;
G0G[54+#33]X-2.254Y2.254Z2.0;
G100C83Z-1.5R.05F15.0Q.4H4.0A30.0; (bolt circle macro)
#33=#33+1;
END1;
G0G91G28Z0Y0M9;
M30;
```

Remember, the #3006 message variable cannot be used on the System 0/6. You must change the #3006 to M00.

System Variables For Tool Offset

The tool offset value for each axis can be read or written to using system variables #2000 through #2800 if the control has from 1 to 200 offsets. If the control has more than 200 offsets the variables are #10001 to #10999. The following tables list the system variables. After the tables is a program example for automatically setting the tool offset.

Offset Variables for 1-200 Offset

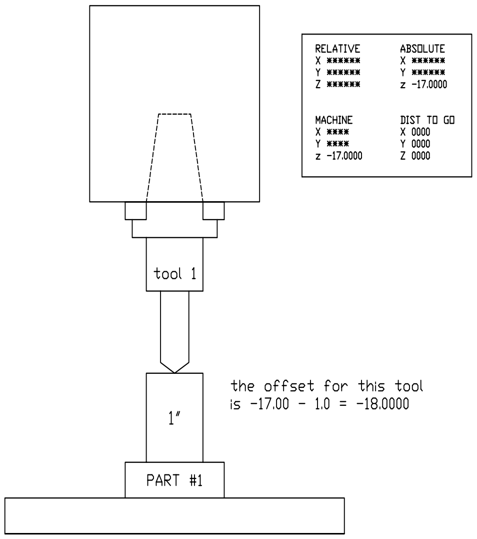
Offset Number	Length (H)		Diameter (D)	
	Geometry	Wear	Geometry	Wear
1	#2001	#2201	#2401	#2601
2	#2002	#2202	#2402	#2602
3	#2003	#2203	#2403	#2603
...	...	...	...	...
199	#2199	#2399	#2599	#2799
200	#2200	#2400	#2600	#2800

As usual, there are different levels of software. If your control has level "A" offset software, the variables only range from #2001 to #2200. If you have level "B" software, the variables range from #2001 to #2400. If you have level "C" software, the table applies in full.

Offset Variables for 1-999 Offset

Offset Number	Length (H)		Diameter (D)	
	Geometry	Wear	Geometry	Wear
1	#10001	#11001	#12001	#13001
2	#10002	#11002	#12002	#13002
3	#10003	#11003	#12003	#13003
...	...	...	...	...
999	#10999	#11999	#12999	#13999

Tool Length Setting Program Example



```
RELATIVE          ABSOLUTE
X ******          X ******
Y ******          Y ******
Z ******          z -17.0000

MACHINE           DIST TO GO
X ****            X 0000
Y ****            Y 0000
z -17.0000        Z 0000
```

Machine Position Offset

This program is used to set tool length offsets. It assumes the tool is set off a 1.0 inch block on the face of the part and the offset is the machine position (a negative value). See the above figure. That is where the #5023-1.0 in line N6 comes from. If you use a different size block or no block, modify this line. If you set Z0 off the face of the spindle and the top of the part and use the absolute position as the offset, change #5023-1.0 to #5003-1.0. See the next two figures. The way to use the program is as follows:

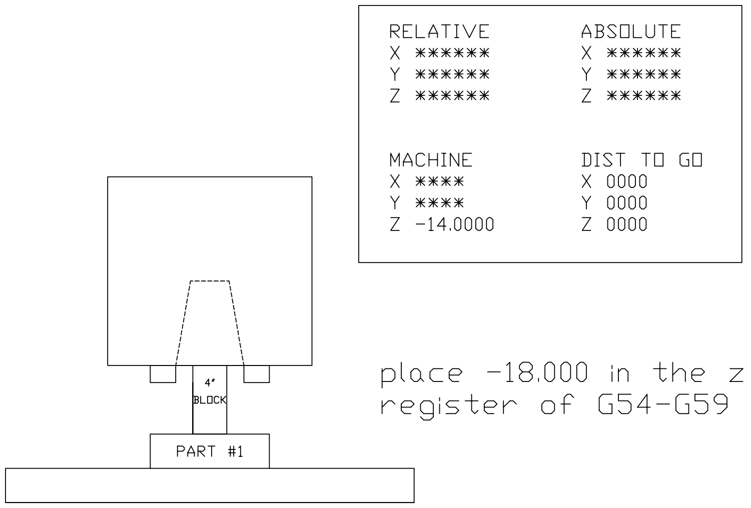
1.) Get all the tools for the job mounted in holders and ready to put in the spindle.

2.) Set #100 to the first tool number you are using. For example #100=1 if the first tool is tool 1.

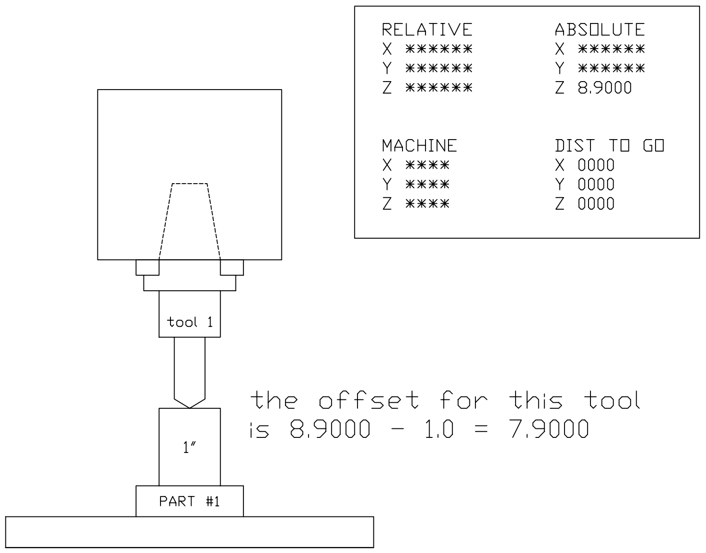
3.) Start the program. When the message appears to set off a 1.0 block go to manual mode and insert tool 1. Then manually move the tool to the block.

4.) Go back to "MEMORY" and press CYCLE START. The new tool length (in machine coordinates) will be stored in the proper offset. The head will then rapid back to the home point and tool change to tool 2.

5.) After the last tool, reset the control and you are ready to machine.



Setting the Spindle Zero



Absolute Position Offset

This program would work very well with one of the spring loaded mechanical or LED (light emitting diode) tool setters on the market. They sell for about $90.00. Used with this program, tool setting would be close to automated. This is the program listing:

```
O0001(TOOL-SETTING PROGRAM);
N1#100=1(set #100 equal to first tool);
N2DO1; {beginning of loop}
N3T[#100]; {select tool}
N4M6; {tool change}
N5#3006=1(SET TOOL OFF 1.0 BLOCK);
N6#[2000+#100]=#[5023]-1.0; {put machine Z minus 1" into offset}
N7G0G91G28Z0.0; {rapid to Z axis zero return}
N8#100=#100+1; {increment to next tool}
N9END1; {end indefinite loop}
N10G90; {switch back to absolute coordinates}
M30;
```

If you are using a 0B/C or 6M control, Change line N5 to M00. The M00 will stop the program after the tool change instead of displaying a message. The 0 series controls do not have the #3006 display variable.

System Variables for Programmable Inputs/Outputs

The programmable inputs/outputs (I/O) are one of the most powerful features of Custom Macro. However, they are not the easiest to implement. The other features of custom macro simply require you to learn the code. Using the I/O feature is a bit more involved.

Unless you plan to buy the proper equipment to write ROM cartridges, get to know a technician at your machine distributor on a first name basis.

The machine tool builder seldom decodes the system variables for the inputs/outputs. This means that the READ ONLY MEMORY (ROM) cartridge in the control does not have an address connecting the custom macro variable with a connector on the mother board. You must know what type of interface the machine tool builder used and have the connection diagrams for that interface. The program stored in the ROM cartridge is sometimes called Firmware because it isn't hardware but isn't quite software either.

The connecting manual is worth buying even if you don't plan on adding I/O. There is a lot of good general information in the manual.

Fanuc sells manuals called *Connecting Manuals*. If you plan to use the I/O feature of custom macro you should purchase the connecting manual for your control. While you are waiting for the connecting manual, contact the machine tool builder and find out which type of interface your machine tool uses. Then ask Fanuc to supply a copy of the interface address list for this interface. This list will give you the address of the signals listed in the System Variables Input and Output table listed next.

An address list for a *Basic Machine Interface* (BMI) is shown in the next table. This is strictly for educational purposes and is not a complete interface list. Do not base your ladder program alterations on this example. You must discuss the interface in your machine with the builder to be sure of the version and exact level of software. There are other types of interfaces than BMI. The "G" and "F" values in the table are the internal signals through the interface.

The Ladder Program is stored in the ROM chip and controls the Programmable Logic Controller in the Fanuc control.

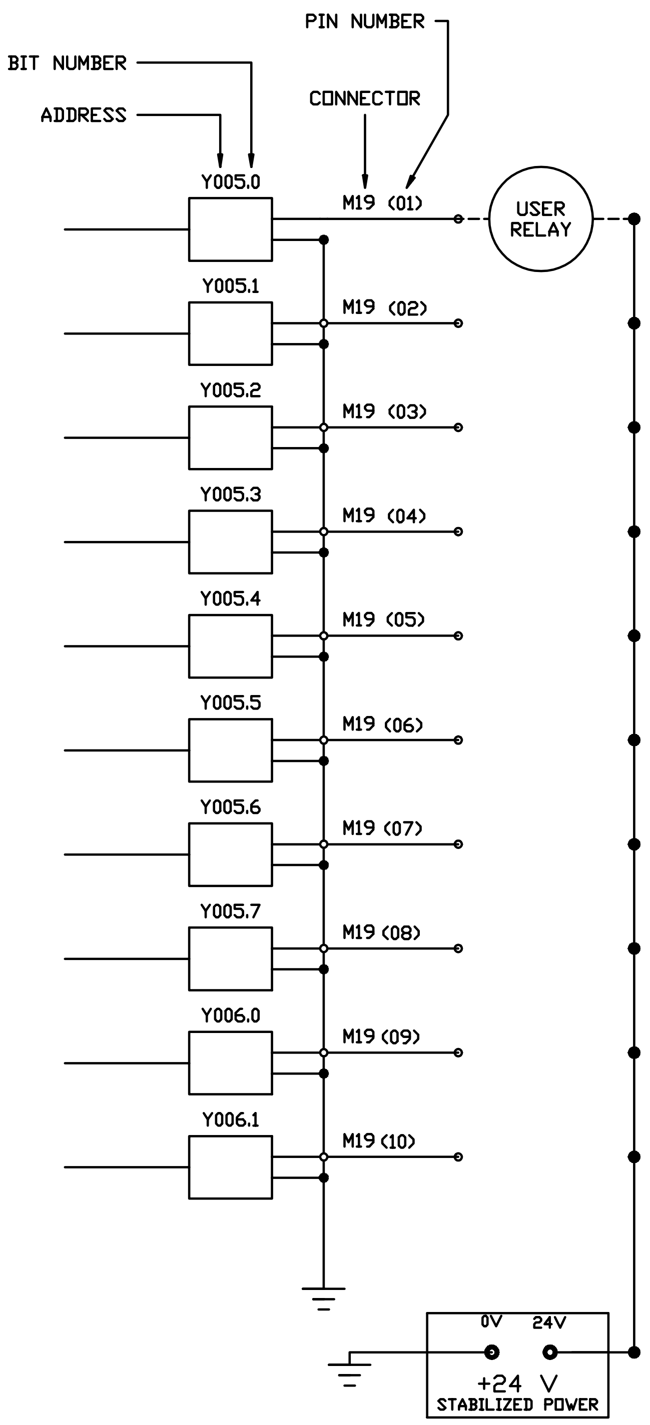
When the ladder program is modified, you will add two lines for each input. The first line will read the "X" value from the connector pin. The second line will write to the "G" value in the interface for an input. For an output, you will read the "F" value from the interface and write to the "Y" value of the connector. The "X", "Y", "F" and "G" values can be seen using the diagnostic page.

Press the key PC/NC. When the screen changes and the soft keys appear, press the soft key "PCDGN". This will bring up a screen with "G" values. To view other addresses, enter the address and press "SEARCH". For example, to view X5.5, enter X5.5 from the keypad and press search. The connecting manual will have complete examples of interface signals.

BASIC MACHINE INTERFACE INPUT SIGNALS								
BIT→ ↓ ADD	7	6	5	4	3	2	1	0
G48	UI031	UI030	UI029	UI028	UI027	UI026	UI025	UI024
G49	UI023	UI022	UI021	UI020	UI019	UI018	UI017	UI016
G50	UI015	UI014	UI013	UI012	UI011	UI010	UI009	UI008
G51	UI007	UI006	UI005	UI004	UI003	UI002	UI001	UI000
BASIC MACHINE INTERFACE OUTPUT SIGNALS								
F48	UO031	UO030	UO029	UO028	UO027	UO026	UO025	UO024
F49	UO023	UO022	UO021	UO020	UO019	UO018	UO017	UO016
F50	UO015	UO014	UO013	UO012	UO011	UO010	UO009	UO008
F51	UO007	UO006	UO005	UO004	UO003	UO002	UO001	UO000

The Electrical Diagrams

You will also need the electrical manual the machine tool builder supplied with your machine. Using the electrical manual, locate unused pins on the interface board. Most machine tools do not use all the available pins in each connector. How much I/O you will have available depends on how many pins your machine tool builder left. If the machine does not have many special features that require I/O you should have at least 5 of each type.



Interface Diagram For Output

The electrical diagrams should have a page that resembles the above figure. This figure shows you a connector with output signals. This particular example is from a machine with a BMI ladder interface and an A3 interface board. The "ADDRESS" is the "Y" from the PC/NC diagnostic page on the CRT. The M19 is the connector number and the number in parentheses is the pin number.

You will look for pins that are not connected. For example, in the figure, Y005.0 is open. This output uses pin 1 on connector M19. You can add your own 24 volt relay to this output.

Fanuc now sells MS-DOS based software to modify the ladder program. You will need a PC and the software sells for about $4000.00

After you have located the unused pins and the address of the interface signals, you will need to have the machine tool builder rewrite the ladder program in the ROM cartridge. This involves removing the cartridge, sending it to the builder and waiting while they rewrite it. The machine will not be usable while this is being done.

After all this is done, actually using the I/O is fairly easy. The I/O variables are like any other system variable. The next table lists the system variables and how they relate to the I/O address.

User Supplied Input Signals #1000-#1015

SYSTEM VARIABLE	BINARY VALUE	DECIMAL VALUE	INTERFACE INPUT SIGNAL NAME
#1000	2^{0}	1	UI000
#1001	2^{1}	2	UI001
#1002	2^{2}	4	UI002
#1003	2^{3}	8	UI003
#1004	2^{4}	16	UI004
#1005	2^{5}	32	UI005
#1006	2^{6}	64	UI006
#1007	2^{7}	128	UI007
#1008	2^{8}	256	UI008
#1009	2^{9}	512	UI009
#1010	2^{10}	1,024	UI010
#1011	2^{11}	2,048	UI011
#1012	2^{12}	4,096	UI012
#1013	2^{13}	8,192	UI013
#1014	2^{14}	16,384	UI014
#1015	2^{15}	32,768	UI015

User Supplied Input Signals #1015-#1031

SYSTEM VARIABLE	BINARY VALUE	DECIMAL VALUE	INTERFACE INPUT SIGNAL NAME
#1016	2^{16}	65,536	UI016
#1017	2^{17}	131,072	UI017
#1018	2^{18}	262,144	UI018
#1019	2^{19}	524,288	UI019
#1020	2^{20}	1,048,576	UI020
#1021	2^{21}	2,097,152	UI021
#1022	2^{22}	4,194,204	UI022
#1023	2^{23}	8,388,608	UI023
#1024	2^{24}	16,777,216	UI024
#1025	2^{25}	33,554,432	UI025
#1026	2^{26}	67,108,864	UI026
#1027	2^{27}	134,217,728	UI027
#1028	2^{28}	268,435,456	UI029
#1029	2^{29}	536,870,912	UI029
#1030	2^{30}	1,073,741,824	UI030
#1031	2^{31}	2,147,483,648	UI031

Programming the Inputs

This table is not as complicated as it looks. For example: if you had an input connected to signal UI000 (through G51 bit 0 with BMI) and the input was active, the value of #1000 would be 1. If this was the feed back from a hydraulic clamp (the clamp is activated in other words) you might use the program line:

```
IF[#1000 NE 1]THEN#3006=1(PARTS NOT CLAMPED);
```

If the clamp is activated, the program will ignore the line. If for some reason the hydraulic system has failed the message PARTS NOT CLAMPED would stop the machine. Remember, the System 0 and 6 do not have the #3006 variable. You will have to use M00.

This would be another application for the logical AND function. You would read #1032, then AND it with the value of the inputs you want to check. One line of code then checks multiple input states.

You can determine the status of all the inputs by reading variable #1032. That is what the value in the table under DECIMAL VALUE is used for. The value of #1032 is the sum of each signal that is high. For example: if you read the value of #1032 and it was 3, inputs UI000 and UI001 would be active. Remember input UI000 is 2^0 and input UI001 is 2^1. Added up this is 2^0(=1) + 2^1(=2) = 3. With 32 inputs, the value of #1032 will get very large.

Programming the Outputs

The output variables work the same way. For example: If the clamp in the above example was connected to UO000 (through F51 bit 0 with BMI) the program line

```
#1100=1;
```

would activate the clamp. The system variables for I/O are not cleared by pressing the *RESET* button. This means that the parts will not be unclamped if the *RESET* button is pressed. You would have to use the program line

```
#1100=0;
```

to release the clamps. This can be frustrating when you interrupt the program and want to release a part. However, it is much safer not to have the I/O variables cleared each time "RESET" is pressed.

Remember that the variables #1000-#1031 and #1100-#1131 are only one bit registers. That means the value can only be 1 or 0. System variables #1032 and #1132 are 32 bit and can have large values.

User Supplied Output Signals #1100-#1115

SYSTEM VARIABLE	BINARY VALUE	DECIMAL VALUE	INTERFACE OUTPUT SIGNAL NAME
#1100	2^{0}	1	UO000
#1101	2^{1}	2	UO001
#1102	2^{2}	4	UO002
#1103	2^{3}	8	UO003
#1104	2^{4}	16	UO004
#1105	2^{5}	32	UO005
#1106	2^{6}	64	UO006
#1107	2^{7}	128	UO007
#1108	2^{8}	256	UO008
#1109	2^{9}	512	UO009
#1110	2^{10}	1,024	UO010
#1111	2^{11}	2,048	UO011
#1112	2^{12}	4,096	UO012
#1113	2^{13}	8,192	UO013
#1114	2^{14}	16,384	UO014
#1115	2^{15}	32,768	UO015

User Supplied Output Signals #1115-#1131

SYSTEM VARIABLE	BINARY VALUE	DECIMAL VALUE	INTERFACE OUTPUT SIGNAL NAME
#111 6	2^{16}	65,536	UO016
#1117	2^{17}	131,072	UO017
#1118	2^{18}	262,144	UO018
#1119	2^{19}	524,288	UO019
#1120	2^{20}	1,048,576	UO020
#1121	2^{21}	2,097,152	UO021
#1122	2^{22}	4,194,304	UO022
#1123	2^{23}	8,388,608	UO023
#1124	2^{24}	16,777,216	UO024
#1125	2^{25}	33,554,432	UO025
#1126	2^{26}	67,108,864	UO026
#1127	2^{27}	134,217,728	UO027
#1128	2^{28}	268,435,456	UO029
#1129	2^{29}	536,870,912	UO029
#1130	2^{30}	1,073,741,824	UO030
#1131	2^{31}	2,147,483,648	UO031

Suppression Of Single Block & Auxiliary Function End Signal (#3003)

This variable allows you to disable single block mode and to ignore the auxiliary end signal from S, T, M and B codes. The single block ignore is used after a custom canned cycle has been debugged. Your canned cycle will operate like one of Fanuc's drill cycles. However, this is a dangerous variable to use before the cycle is debugged. You are expecting to be in single block and the control behaves like it is in automatic mode. The state of #3003 is cleared by pressing "RESET" but you should include the statement #3003=0 at the end of your cycle.

The auxiliary end signal (FIN) can also be ignored by setting #3003. This is useful under the right conditions but is a very dangerous mode. If you do not completely understand why you are using #3003 to ignore the FIN signal ***DON'T USE IT***.

#3003 EXAMPLE:

```
O0001;
G90G80G40G49G17G20; {INITIALIZE CONTROL}
G0G54G43H1X0Y0Z3.3; {POSITION TO START}
#3003=1; {TURN OFF SINGLE BLOCK W/O DISABLING END SIGNAL}
G0Z#18; {MAKE AN AXIS MOVE W/O SINGLE BLOCK ACTIVE}
G1Z#26F#9;
G0Z#18;
#3003=0; {CANCEL SINGLE BLOCK SUPPRESSION}
G0G91G28Z0;
M30;
```

Table of #3003 Settings

VALUE OF #3003	SINGLE BLOCK STOP	AUXILIARY FUNCTION END SIGNAL
0	Not suppressed	Awaited
1	Suppressed	Awaited
2	Not suppressed	Not awaited
3	Suppressed	Not awaited

System Variable For Feed Functions (#3004)

System variable #3004 gives you control over FEED HOLD, FEED RATE OVERRIDE and the EXACT STOP function. The setting is modal and remains active until reset by another #3004=0 or RESET is pressed. If your macro requires you to disable one of these functions, be sure to reset #3004 to zero afterward. Forgetting to enable FEED HOLD could obviously be dangerous.

Pressing Feed hold when suppression is active:

1. When the feed hold button is held, FEED HOLD occurs at the start of the first block outside the suppression range.
2. When FEED HOLD is pressed and released, Feed hold occurs at the end of the first block outside the suppression range.

The following table lists the possible values for #3004:

Table of #3004 Settings

VALUE OF #3004	FEED HOLD	FEED RATE OVERRIDE	EXACT STOP CHECK

VALUE OF #3004	FEED HOLD	FEED RATE OVERRIDE	EXACT STOP CHECK
0	0	0	0
1	X	0	0
2	0	X	0
3	X	X	0
4	0	0	X
5	X	0	X
6	0	X	X
7	X	X	X

0 = MODE IS EFFECTIVE X = MODE IS NOT EFFECTIVE

#3004 Example:

```
O0001;
G90G80G40G49G17G20; {INITIALIZE CONTROL}
G0G54G43H1X0Y0Z3.3; {POSITION TO START}
#3004=2; {DISABLE FEEDRATE OVERRIDE W/O DISABLING FEED HOLD or EXACT STOP}
G0Z#18; {MAKE AN AXIS MOVE W/O FEED RATE OVERRIDE ACTIVE}
G1Z#26F#9;
G0Z#18;
#3004=0; {CANCEL FEED RATE OVERRIDE SUPPRESSION}
G0G91G28Z0;
M30;
```

System Variable For Mirror Image (#3007)

This read only system variable allows you to check the status of mirror image on individual axes. This is an 8 bit binary register. If you are not familiar with the binary system see “Decimal to Binary Conversion” on page 27. For each axis 0 is mirror image off and 1 is mirror image on.

Mirror Image example:

If the value of #3007 is 1 mirror image is on for the 1st axis.

If the value of #3007 is 2 mirror image is on for the 2nd axis.

If the value of #3007 is 3 mirror image is on for the 1st and 2nd axis.

This variable does not turn mirror image on or off, it simply reports the status of mirror image.

Table of #3007 Values

#3007	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
	8TH AXIS	7TH AXIS	6TH AXIS	5TH AXIS	4TH AXIS	3RD AXIS	2ND AXIS	1ST AXIS

System Variable For Time & Date (#3011 3012)

The System 15 and 0C have system variables for the current time and date. These variables are especially useful if you are using the **DPRNT** command to output information about the program. To use these variables, simply set a common variable equal to #3011 for the date or #3012 for the time. For example:

#100=#3011; will store the current date in variable #100

#101=#3012; will store the current time in variable #101

See "DPRNT Output Statement 0 and 10-15" on page 63 for an example of using the time and date variables with the **DPRNT** command.

Reading An Endmill Radius

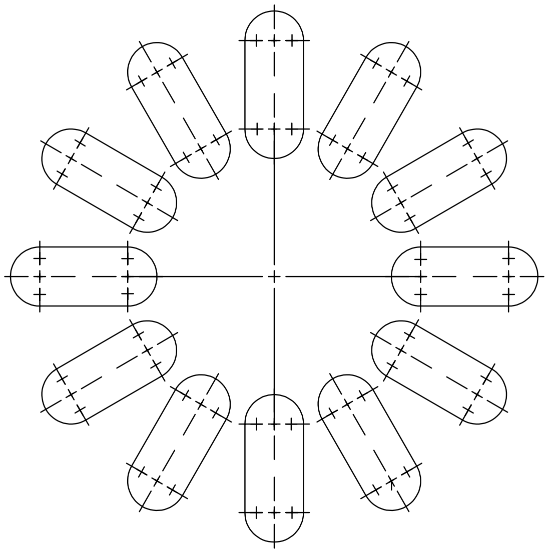
The radius of the cutter can be determined using system variables. This can be used if a circular interpolation program is required and you do not want to turn G41, G42 on and off. The following program will read the value stored in the offset page:

```
N1T1;select a tool.
N2M6; tool change.
N3S1000M3;
N4G0G43H1D2X0Y0Z.1;set tool length offset 1 & radius offset 2.
N5#100=#[#4107+2000]; #100 is equal to the radius of the cutter (value stored in offset 2)
```

The value of #100 is added to the address where compensation is needed. For example, G1 X[1.0+#100]. When the radius is stored at offset 2 the cutter will move to X1.0 plus the radius of the endmill. If a negative value is stored in the offset page the cutter will move to X1.0 minus the radius of the endmill.

Custom Macro Slotting Example

A family of parts with slots, as shown in the next figure, is a good place to use a custom macro program. The number of slots, diameter or width can easily be changed if custom macro is used. If an off-line programming system is used, several problems arise. First, the program becomes very large because the off-line system will write the code for each slot. If the number of slots is small this is no problem. The part that was used for this example actually had 144 slots on two different diameters. The off-line system easily wrote the code, but the resulting tape was so long it had to be broken up into two parts. You can imagine that the operator was not too happy about that!



Radial Slots

Editing is another problem. The off-line system writes the CNC program using actual X and Y values. If the amount of finish stock or other cutting parameter needs to be changed after the first part, the program must be regenerated off-line. This means stopping an expensive machine tool and possibly losing any other changes that were made to the program. Custom macro will eliminate these problems. This example will show you how to write the program.

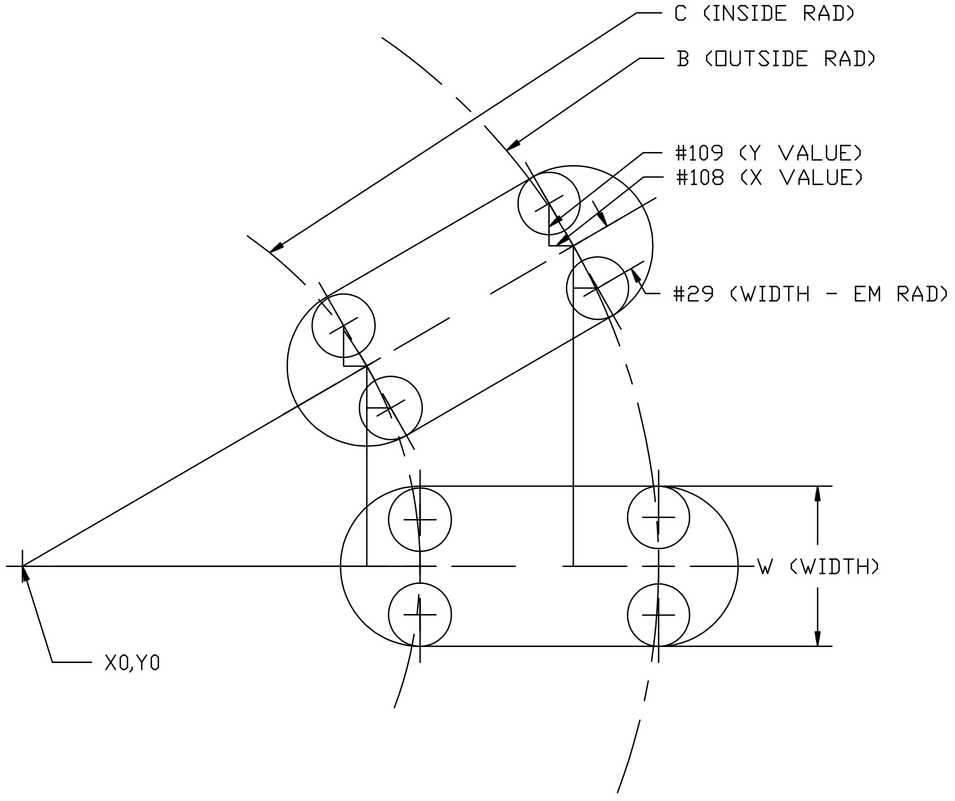
First, we will assume some initial conditions:

1. The slot has two holes predrilled at each end. This is done using a custom macro bolt circle.
2. The slot will be climb milled.
3. The endmill used is wider than half the width of the slot.

With these initial conditions determined, the program can be started.

If the slot is centered on the X or Y axis, the code is easy to figure out. Simply offset the center of the cutter by the endmill radius and sweep a radius that is smaller by the endmill radius at each end. Of course, if there is a finish allowance, it must be taken into account. That is easily done after a method is determined to make the slots that are not on an X or Y axis.

The best way to get started on the off axis slots is to make a sketch of the problem. See the next figure. As you can see, the problem begins to fall apart.



Slot Mill Trigonometry

Four identical triangles are generated at the radius of the inscribed circle and a line are perpendicular to the slot. These triangles are similar to the triangle created between the radius of the inscribed circle, the X axis and a line perpendicular to the X axis. Since we know the angle between the X axis and the radius (360°/# of slots for the first slot) and the length of the hypotenuse (the radius of the inscribed circle) we can solve the triangle.

The four similar triangles have the same angles. Their hypotenuse is half the width of the slot minus the radius of the endmill. We learned how to read the radius of an endmill in the last example, so getting that value is no problem. Since the triangles have been rotated 90°, the X axis leg can be calculated using COS(90-A)=SIN(A) and the Y axis value by SIN(90-A)=COS(A).

The only problem left is to find an expression for the angle. This is actually very easy. We know the number of slots needed. Dividing 360° by the number of slots gives us the angle between the slots. Multiplying this value by the number of the slot gives the angle from the X axis for that slot. If we also want to be able to start the slots at some angle other than zero, we can add a variable for that.

Now we are ready to define the macro variables needed to write the program:

```
#1=Angle to shift first slot if needed.
#2=Radius of inside inscribed circle.
#3=Radius of outside inscribed circle.
#21=Finish allowance (U)
#23=Width of slot (W).
#27=#23/2 Width of slot/2
#28=#27-#144-#21
#29=#27-#144
#100=The counter to loop by.
#101=Angle between the slots.
#102=SIN[[#100*#101]+#1] "X" leg.
#103=COS[#100*#101]+#1] "Y" leg.
#104=#2*#103 X value of outside inscribed circle.
#105=#2*#102 Y value of outside inscribed circle.
#106=#3*#103 X value of inside inscribed circle.
#107=#3*#102 Y value of inside inscribed circle.
#108=#28*#102 X value of small triangle.
#109=#28*#103 Y value of small triangle.
#110=#104-#108 X start point of 1st radius
#111=#105+#109 Y start point of 1st radius
#112=#106-#108 X start point top of 2nd radius
#113=#107+#109 Y start point top of 2nd radius
#114=#106+#108 X start point bottom of 2nd radius
#115=#107-#109 Y start point bottom of 2nd radius
#116=#104+#108 X start point bottom of 1st radius
#117=#105-#109 Y start point bottom of 1st radius
#144=#[#4107+2000] Endmill radius.
```

This looks like a lot of variables to simply mill some slots. But when you consider that any number of slots starting at any angle can be milled, it is not so bad. If you have doubts, write the code to rough and finish 72 slots using only standard CNC code.

The complete program is listed next:

```
%
O9001;
IF[#2EQ#0]THEN#3000=1(NO OUTSIDE RADIUS);
IF[#3EQ#0]THEN#3000=2(N0 INSIDE RADIUS);
IF[#11EQ#0]THEN#3000=3(NUMBER OF SLOTS);
IF[#9EQ#0]THEN#3000=4(NO MILL FEEDRATE);
IF[#18EQ#0]THEN#3000=5(NO REF POINT);
IF[#23EQ#0]THEN#3000=6(NO WIDTH GIVEN);
IF[#5EQ#0]THEN#3000=7(MILL RADIUS FEED RATE);
IF[#8EQ#0]THEN#3000=8(NO Z FEEDRATE);
IF[#26EQ#0]THEN#3000=9(NO Z DEPTH);
#140=#5003; {STORE INITIAL Z POINT}
#100=0; {SET COUNTER TO ZERO}
#27=#23/2.0; {DIVIDE DIAMETER BY 2}
#144=#[#4107+2000]; {D+2000 READS RADIUS OF ENDMILL}
#28=#27-#144-#21; {WIDTH - RADIUS - FINISH ALLOWANCE}
#29=#27-#144; {WIDTH-RADIUS}
#101=360/#11; {NO. OF DEGREES BETWEEN SLOTS}
WHILE[#100LTABS[#11]]DO1; {BEGIN LOOP}
#102=SIN[[#100*#101]+#1]; {X leg}
#103=COS[#100*#101]+#1]; {Y leg}
#104=#2*#103; {X VALUE OF OUTSIDE INSCRIBED CIRCLE}
#105=#2*#102; {Y VALUE OF OUTSIDE INSCRIBED CIRCLE}
#106=#3*#103; {X VALUE OF INSIDE INSCRIBED CIRCLE}
#107=#3*#102; {Y VALUE OF INSIDE INSCRIBED CIRCLE}
#108=#28*#102; {X VALUE OF SMALL TRIANGLE}
#109=#28*#103; {Y VALUE OF SMALL TRIANGLE}
#110=#104-#108; {X START POINT OF 1ST RAD}
#111=#105+#109; {Y START POINT OF 1ST RAD}
#112=#106-#108; {X START POINT TOP OF 2ND RAD}
#113=#107+#109; {Y START POINT TOP OF 2ND RAD}
#114=#106+#108; {X START POINT BOT OF 2ND RAD}
#115=#107-#109; {Y START POINT BOT OF 2ND RAD}
#116=#104+#108; {X START POINT BOT OF 1ST RAD}
#117=#105-#109; {Y START POINT BOT OF 1ST RAD}
G0X#104Y#105; {POSITION TO ENTRY POINT}
G0Z#18; {RAPID T0 REFERENCE POINT}
G1Z#26F#8; {FEED TO Z VALUE}
G1X#110Y#111F#5; {FEED TO START POINT}
G1X#112Y#113F#9; {FEED TO BEGINNING OF RADIUS}
G3X#114Y#115R#28F#5; {CUT ARC}
G1X#116Y#117F#9; {FEED TO BEGINNING OF RADIUS}
G3X#110Y#111R#28F#5; {CUT ARC}
G1X#104Y#105; {FEED TO ENTRY POINT}
IF[#21EQ#0]GOTO2; {IF NO FINISH ALLOW}
(*********);
(BEGINNING OF FINISH CUT);
(*********);
#108=#29*#102;
#109=#29*#103;
#110=#104-#108;
#111=#105+#109;
#112=#106-#108;
#113=#107+#109;
#114=#106+#108;
#115=#107-#109;
#116=#104+#108;
#117=#105-#109;
G0X#104Y#105;
G1X#110Y#111;
G1X#112Y#113F#9;
G3X#114Y#115R#29F#5;
G1X#116Y#117F#9;
G3X#110Y#111R#29F#5;
G1X#104Y#105;
N2G0Z#18;
G0Z#140;
#100=#100+1;
END1;
M99;
```

```
(A=START ANGLE OF FIRST SLOT)
(B=OUTSIDE RADIUS CENTER POINT)
(C=INSIDE RADIUS CENTER POINT)
(E=Z FEEDRATE)
(F=FEEDRATE TO MILL)
(H=NUMBER OF SLOTS)
(J=FEEDRATE AROUND RADIUS)
(U=FINISH ALLOWANCE-RADIAL VALUE)
(W=WIDTH OF SLOT)
(Z=Z AXIS DEPTH)
M30%
```

The program looks big, but there is a lot of documentation the could be deleted. There are also several alarms in the beginning of the program.

This example might have appeared difficult at first, but you can see that all that you need to do is define the problem and then proceed step by step. This example demonstrated WHILE-DO loops, IF-THEN alarm statements, system variables and trigonometric calculations. There isn't much else you can use in a program!

Multiple Parts Using G52

Using a WHILE-DO loop to machine multiple parts is very convenient. You don't have all the logistic problems required to keep track of the sub-programs and debugging is easier because there is only one program. However, if you have more parts to machine than you have coordinate systems, looping is not so convenient.

This next example provides one solution to this problem. The standard G52 daughter coordinate system is used with common variables as the X, Y, Z coordinates. This type of program works well if you are using multiple vises in your setup. The first time the parts are made, the common variables are known in advance because you just machined the vise jaws. The next time the parts are run, you simply use the indicating macro to update the common variables. If you are using a spindle mounted probe, the setup is completely automatic. Here is the example program:

```
O0001;
(SET Z0 ON TOP OF TURNED BLANK);
N1;
G0G91G28Z0;
T1M6;
S4000M3;
G90G43H1G54X-1.0Y-1.0Z.1; {POSITION TO FIRST HOLE}
M98P0002; {CALL SUB FOR FIRST OP}
G52X#100Y#101Z#102; {CALL THE DAUGHTER SYSTEM}
M98P0002; {CALL SUB FOR FIRST OP}
G52X#103Y#104Z#105; {CALL THE DAUGHTER SYSTEM}
M98P0002;
G52X0Y0Z0; {RESET G52 TO ZERO}
G0G91G28Z0;
M1
```

This type of program allows you to machine as many parts as you have common variables. If you use this type of program, you should add comment lines to let the operator and setup person know which variables machine which part. This will not add to the cycle time, and the next time the parts are run, there shouldn't be any confusion.

If you still run out of coordinate systems, you can add more common variables. The System 10/11/12 can have up to 300 common variables. The System 15 can have up to 650. Remember that adding variables reduces the amount of free memory. If you add variables, you may have to add memory. Fanuc publishes lists of options for each control. They are free of charge and can be ordered by calling Fanuc's Chicago office.

Custom Macro Messages

Except for the System 0 and 6, an address has been set aside for messages and alarms. The message halts program operation and displays up to 26 characters on the CRT screen. The current display will be changed to the "OPERATOR MESSAGE" screen and the message will be displayed in the upper left corner of the CRT.

The form to use is #3006=1(26 character message). The number 1 is not displayed with the message and can be changed to any number between 1 and 999. This variable address halts program operation similar to a M00 but does not stop the spindle or the coolant.

The message can be used for any purpose. If chips need to be cleared from a shallow hole before a tap is used, a message can be displayed for the operator. If only a certain number of parts need to be machined before a change over, a counter can be set and a message displayed when the correct number of parts are finished.

Custom Macro Alarms

If you suspect an error might occur you can set an error trap. Suppose you had a bolt circle macro that used the letter D for diameter. When the program is called and no diameter is given there is a problem.

If the program stopped and said **"THERE IS NO DIAMETER VALUE"** it would be easy to troubleshoot. Actually this is easy to accomplish. Fanuc has set aside an address for alarms. Up to 999 different alarms can be set. The address of the alarm is #3000.

The format is #3000=n(alarm message) where n is your alarm number and (alarm message) is your description. This example uses an IF statement to see if #7 is vacant. If it is, the #3000=1(no diameter was given) after the THEN statement will cause an alarm. The message can be 26 characters long.

```
O9010;
N1IF[#7 EQ #0]THEN #3000=1(NO DIAMETER GIVEN);
(program)
(program)
(program)
N50M99;
```

If the variable "D" is not present, the screen will clear and the message **MC001 NO DIAMETER GIVEN** will appear. If #3000=2(no diameter was given) is used the alarm will be **MC002 NO DIAMETER GIVEN** will be given. This allows alarms to be given unique numbers for documentation purposes.

DPRNT Output Statement 0 and 10-15

The **DPRNT** statement allows you to output program data while the program is running. If the control is interfaced to an off-line computer, you can download data from the control, while the program is running. An example would be downloading Statistical Process Control data collected by a spindle mounted probe. The process control charts could be updated in real time on a workstation computer.

A panel mounted printer can connected to the spare serial port on the mother board of the control. This would allow you to output offsets or any other information to the operator while the program is running.

The System 15 and System OC have a system variable for time and date. You could output the time and date each time the cycle ended. If a spindle mounted probe is being used, the measurements could be printed while the probe is inspecting the part or they could transmitted an the off-line computer. If tool life management is being used, a message could be sent to the office when a tool reaches it's useful life, instead of informing the operator. As another example, if a spindle mounted probe is being used, the measurements could be transmitted to the off-line computer.

The disadvantages to sending data are an increase in cycle time. If an off-line computer is used to monitor the CNC control, the computer must be on-line while the machine is in operation.

If you want to install panel mounted printer and use the serial connector on the mother board, you will need a Honda 20 pin male serial connector. Fanuc or GE Fanuc both sell these connectors. You will then have to make up a cable.

The only pins required are transmit (RD) and signal ground (SG). If you set the "foreground" device on the **SETTING** page to 2 for output and leave the foreground input and background input/output device as 1, you can output to the printer and still upload to the control in the foreground. You will have to use "Background Edit" to punch a program to your off-line computer. These settings are on the "SETTING" page.

To send data during program execution follow these steps:

1. Place the statement POPEN before the data to be sent.
2. Use DPRNT[message #i[44]] to send the message.
3. Place the statement PCLOS to close the port.

This sample program outputs the value of X#103, Z#510 and #100.

```
O1000(DPRNT-EXAMPLE);
POPEN;
G90G43H1X0Y0Z1.0G54;
#3002=0;
G81Z-1.0R.05F10.0;
DPRNT[X#103[24]Z-#510[24]];
#100=#100+#3002;
DPRNT[CYCLE TIME IS #100[44]];
PCLOS;
G0G91G28Z0;
M30;
```

The X#103[24] transmits X and the value of #103 with 2 places before the decimal and 4 after. The #100[44] transmits the value of #100 with 4 places before and after the decimal point. For example, if #103 is equal to 10.5 and #510 is equal to 12.1234 when the DPRNT statement is read, the output will be X10.5Z-12.1234 from the port.

If you use the DPRNT statement, do not forget to close the port after the transmission. The **POPEN** and **PCLOS** statements are only needed once. The DPRNT is needed for each line of data. Spaces are output using an asterisk. For example DPRNT[*****test]; will print the word test with 5 leading spaces. The statement DPRNT[-----]; will output a short line. Use your imagination to design the output layout you need.

The following program outputs the time, date and part number as a header and then outputs the X, Y center and Z value of the selected work offset system. The G54.1 is the additional work offset option that is available on later controls. This option gives you a total of 54 work offsets. The 6000 number variables are from the menu programming option. This option allows you to write a program that excepts input from the operator and then passes these variables to your macro program. The next section discusses menu programming.

```
O9425(PRINT  WORK OFFSETS);
POPEN;
DPRNT[--------------------------------];
DPRNT[DATE*IS*#3011[80]];
DPRNT[TIME*IS*#3012[60]];
DPRNT[PART*NUMBER*IS*#6003[80]];
DPRNT[--------------------------------];
#100=[#6001-1]*20;
#101=0;
#33=#6001;
WH[#32LT#6002]DO1;
DPRNT[G54.1*P#33[20]];
DPRNT[X*CENTER*#[7001+#100+#101][24]];
DPRNT[Y*CENTER*#[7002+#100+#101][24]];
DPRNT[Z*VALUE*IS*#[7003+#100+#101][24]];
DPRNT[--------------------------------];
#101=#101+20;
#32=#32+1;
#33=#33+1;
END1;
PCLOS;
M30;
```

The output of this program would like this:

```
--------------------------------
DATE IS  19911017
TIME IS  135508
PART NUMBER IS 40230007
--------------------------------
G54.1 P 1
X CENTER  10.9933
Y CENTER - 9.7267
Z VAL -18.2034
--------------------------------
G54.1 P 2
X CENTER  19.4968
Y CENTER - 9.7281
Z VAL -18.2013
--------------------------------
G54.1 P 3
X CENTER  27.9907
Y CENTER - 9.7238
```

```
Z VAL -18.2048
-------------------------------
```

As you can see, the output can be formatted to be very readable.

Menu Programming

The System 10-15 controls have an option for writing menu driven programs. This option allows you to write programs that prompt the operator for input at run time. An example of this would be inspection probing. You can write a menu program that asks the operator for the number of parts to probe and the required data for the probe program.

Or you might use a menu for the tools to set using the earlier example of a tool setting macro. See "Tool Length Setting" on page 43. The biggest drawback to menus are the memory required. The option requires several meters of memory before you even write any code. If you have this option turned on, you will need at least eighty meters of memory.

The CRT screen shown below is an example of a simple menu. In this example, Program one allows the operator to enter the variables for a probing program. The variables required will be explained later. This particular program probes parts on the zero degree side of a tombstone fixture. The next two programs probe parts at 90° and 270° respectively. The fourth program is for a tool setting macro and the final program opens the serial port and outputs the current work coordinates.

```
MENU PROGRAM01234 N0000

PROGRAM 01

N01:              01: PROBE 0 DEG
N02:              02: 90 DEGREE MENU
N03:              03:180 DEGREE MENU
N04:              04:TOOL SET MENU
N05:              05:WORK OFFSET PRNT
N06:              06:
N07:              07:
N08:              08:

EDIT *** STOP *** *** *** ***
INSERT  DELETE       SRC_PRG   BLK_DAT
```

CRT Screen for a simple menu

The following program shows how to create a menu for the screen in shown above:

```
O9419; {any valid program number)
G10L60P01(PROBE MENU 0 DEG); {item1 displayed on the menu page}
G10L61P01Q01(Y POS. OF  0 SET); {displayed on the sub menu}
G10L61P01Q02(DIA. TO PROBE); {displayed on the sub menu}
G10L61P01Q03(Z VALUE TO PROBE); {displayed on the sub menu}
G10L61P01Q04(R VALUE - FOR ID); {displayed on the sub menu}
G10L61P01Q05(NUM. OF PARTS); {displayed on the sub menu}
(***);
G10L60P02(90 DEGREE MENU); {item 2 displayed on the menu page}
G10L61P02Q01(Y POS. OF  0 SET); {displayed on the sub menu}
G10L61P02Q02(DIA. TO PROBE); {displayed on the sub menu}
G10L61P02Q03(Z VALUE TO PROBE); {displayed on the sub menu}
G10L61P02Q04(R VALUE - FOR ID); {displayed on the sub menu}
G10L61P02Q05(NUM. OF PARTS); {displayed on the sub menu}
(***);
G10L60P03(270 DEGREE MENU); {item 3 displayed on the menu page}
G10L61P03Q01(Y POS. OF  0 SET); {displayed on the sub menu}
G10L61P03Q02(DIA. TO PROBE); {displayed on the sub menu}
G10L61P03Q03(Z VALUE TO PROBE); {displayed on the sub menu}
G10L61P03Q04(R VALUE - FOR ID); {displayed on the sub menu}
G10L61P03Q05(NUM. OF PARTS); {displayed on the sub menu}
(***);
G10L60P04(TOOL SET MENU); {item 4 displayed on the menu page}
G10L61P04Q01(START TOOL NUM.); {displayed on the sub menu}
G10L61P04Q02(NUMBER OF TOOLS); {displayed on the sub menu}
G10L61P04Q03(PART NUMBER); {displayed on the sub menu}
(***);
G10L60P05(WORK OFFSET PRNT); {item 5 displayed on the menu page}
G10L61P05Q01(1ST OFFSET NO.); {displayed on the sub menu}
G10L61P05Q02(NUMBER TO PRINT); {displayed on the sub menu}
G10L61P05Q03(PART NUMBER)); {displayed on the sub menu}
M30;
%
```

In this program, the lines G10L60P01, G10L60P02, G10L60P03, G10L60P04, G10L60P05 display the titles in the menu when the program is run. The titles can have a maximum of 16 characters. In this example the menu will look like this:

01: PROBE MENU 0 DEG

02: 90 DEGREE MENU

03: 270 DEGREE MENU

04: TOOL SET MENU

05: WORK OFFSET PRNT

When a menu choice is selected and you press BLK_DAT, the sub menu appears. The format for the sub menu is G10**L61**P01Q01, G10**L61**P01Q02..., G10**L61**P01Q0*n*. The sub menu uses L61 instead of L60. Each menu item has it's own sub menu. Each sub menu item uses G10L61 and the same P value as the menu item. The L value starts at one and is incremented for each sub menu item. There can be a maximum of 16 sub menu items. For example, if the first menu item has two sub menu choices, the lines for the menu will be:

G10L60P01(ITEM 1 MENU TITLE); {item 1 displayed on the menu page}

G10L61P01***Q01***(SUB MENU 1 TITLE); {displayed on the sub menu}

G10L61P01***Q02***(SUB MENU 2 TITLE); {displayed on the sub menu}

If there are two menu items, the second sub menu lines will become:

G10L60P02(ITEM 2 MENU TITLE); {item 2 displayed on the menu page}

G10L61***P02***Q01(SUB MENU 1 TITLE); {displayed on the sub menu}

G10L61***P02***Q02(SUB MENU 2 TITLE); {displayed on the sub menu}

As you can see, the P parameter changes to two, but the Q parameter still starts at one and increments for each sub menu item.

The following program is used to display the menu:

Menu Display Program

```
%O9420;
#6091=1;
N2IF[#6000 EQ 0]GOTO7;
M98 P[#6000 +9500]; {add menu number to 9500}
#6091=#6091+1;
IF[6091GT16]GOTO7;
GOTO2;
N7M30;
%
```

This example assumes the program to be executed is numbered 9500 plus the value of the menu choice. For example, program 9501 would be executed if menu choice one was selected. That is equal to 9500 plus the value of #6000 which is one in this case.

Custom Macro Characters

#	Number sign. Used to name a variable. For example #1 is variable number 1. See "VARIABLES".
[	Open bracket. Used to change the sequence of mathematical operations. Can be nested five levels. The inner level is solved first followed by the next level until all open brackets are done. Must be used in pairs with the close bracket.
]	Close bracket. Used to close an equation that has been given priority.
( )	Parentheses. Used as a pair to enclose comments in a program. Statements inside parentheses are ignored by the control. Fanuc calls the open parenthesis "control out" and the close parenthesis "control in."
+	Addition sign. Used to add two variables, a constant and a variable or two constants.
--	Minus sign. Used to subtract two variables, a constant and a variable, or two constants.
*	Asterisk. Used to multiply two variable, a variable and a constant or two constants.
/	Slash. Used to divide two variables, a variable and a constant or two constants.
DO m	Creates an unlimited loop when used with ENDm. See "PROGRAM ITERATION".
GOTO n	Used as an unconditional branch or jump in a program. See "PROGRAM BRANCHING".
IF	The beginning of an IF[condition] statement. Can be used for conditional branching, checking for presence of a variable or conditional setting of a variable . See "PROGRAM BRANCHING".
THEN	Can be used with IF statement to make a conditional decision. See PROGRAM BRANCHING.
WHILE	Beginning of a WHILE[condition]DO statement. See "PROGRAM ITERATION".

System 6B Supplement

The system 6MB control has two types of custom macro, A and B. Custom macro B has all of the features of custom macro A plus these additional features:

- WHILE[<CONDITION IS TRUE>]DOm where m is 1,2, or 3
- ENDm
- Use of system variables.
- Can use multiplication and division.
- The following functions:

#1=SIN[#2]	Sine function in degree units.
#1=COS[#2]	Cosine function in degree units.
#1=TAN[#2]	Tangent function in degree units.
#1=ATAN[#2]/[#3]	Arctangent function in degree units.
#1=SQRT[#2]	Square root.
#1=ABS[#2]	Absolute value.
#1=ROUND[#2]	Round off. Above .5 next higher number.
#1=FUP[#2]	Add 1 for fractions less than 1.
#1=FIX[#2]	Discard fractions less 1.
#1=BIN[#2]	Convert from BCD to binary.
#1=BCD[#2]	Convert from binary to BCD.

In addition, the System 6B does not have the #3006=N(MESSAGE) and IF[<CONDITION IS TRUE>]THEN features of the System 10-15.

The examples in the manual are based on the System 10/11 to highlight custom macro. The system 6B user will not be able to use these two features.

The alarms used in the application programs will need to be changed to the following format:

IF[#1EQ#0]GOTOn where #1 is the proper variable and n is any CNC line number.

For example:

IF[#3EQ#0]THEN#3000=3(NO DRILL CYCLE); could be changed to

IF[#3EQ#0]GOTO200;

Line N200 would be placed after the M99 and would have the format #3000=3(NO DRILL CYCLE).

To edit protect a macro program in the System 6MB:

O8000 TO O8999 SET PARAMETER 0319 BIT 7 TO 1

O9000 TO O9899 SET PARAMETER 0318 BIT 7 TO 1

To stop a macro program execution from being displayed:

O8000 TO O8999 SET PARAMETER 0319 BIT 5 TO 1

O9000 TO O9899 SET PARAMETER 0318 BIT 5 TO 1

To set macro statement execution to single block:

O1 to O7999 and O9900 to O9999 set parameter 0319 bit 0 to 1

O8000 TO O8999 SET PARAMETER 0319 BIT 6 TO 1

O9000 TO O9089 SET PARAMETER 0318 BIT 6 TO 1

VARIABLE	CONTENTS	SYSTEM
#2500	FIXTURE OFFSET OF 1ST AXIS	G54
#2600	FIXTURE OFFSET OF 2ND AXIS	G54
#2700	FIXTURE OFFSET OF 3ND AXIS	G54
#2501	FIXTURE OFFSET OF 1ST AXIS	G55
#2601	FIXTURE OFFSET OF 2ND AXIS	G55
#2701	FIXTURE OFFSET OF 3ND AXIS	G55
#2502	FIXTURE OFFSET OF 1ST AXIS	G56
#2602	FIXTURE OFFSET OF 2ND AXIS	G56
#2702	FIXTURE OFFSET OF 3ND AXIS	G56
#2503	FIXTURE OFFSET OF 1ST AXIS	G57
#2603	FIXTURE OFFSET OF 2ND AXIS	G57
#2703	FIXTURE OFFSET OF 3ND AXIS	G57
#2504	FIXTURE OFFSET OF 1ST AXIS	G58
#2604	FIXTURE OFFSET OF 2ND AXIS	G58
#2704	FIXTURE OFFSET OF 3ND AXIS	G58
#2505	FIXTURE OFFSET OF 1ST AXIS	G59
#2605	FIXTURE OFFSET OF 2ND AXIS	G59
#2705	FIXTURE OFFSET OF 3ND AXIS	G59

The sample program that uses fixture offsets is based on the System 10/11. To use these programs on the System 6M, change the system variables for X and Y (#5221,#5222 to #2500,#2600), change #130=20 to 130=1 and change #130=#130+20 to #130=#130+1.

Installing the Programs

System 10-15 instructions

Using the included custom macro programs should increase the productivity of your machining center by reducing programming time and effort. Once you are familiar with these programs, the amount of time spent on program prove out should be reduced also. Using the programs should be very easy and require little modification to your present programming method. These programs have been tested under many conditions and are well developed.

However, the Fanuc control has several parameters related to custom macro. The following section will assist you in making any changes that may be necessary to fit the programs to your particular control. System 0MB/6MB users should look at the supplement for additional information.

System Requirements

Hint: You can also enter the following in MDI mode.
#100=0
and press cycle start. If you don't have Macro, you will see an alarm.

Your control must have the Custom Macro option. There is no way to run a custom macro program on a control that does not have the option installed. You can tell if your control has the option by pressing the function menu key (the soft key on the far left), until the "setting" soft key appears and then pressing the "chapter" key (the key on the far right). If custom macro is available the soft key second from the right will change to "macro". If the control does not have the option contact Fanuc.

Installation Procedure System 10-15

1. Set parameter 8000 bit 0 (farthest to the right) to a one. To do this you must be in "MDI" mode. Press the function menu key (the farthest soft key to the left) until the "setting" soft key is visible. Press the "setting" soft key until the setting data screen appears. Press the chapter key (the farthest soft key to the right) until the "inp-no." key is visible. Then enter 8000 from the keyboard and press the "inp-no." soft key. Press the "on: 1" soft key. At this point an alarm will occur and clear the screen. This alarm is to remind you that all parameters can now be changed, so be careful.
2. Press the function menu key until the "service" soft key is visible. Enter 2201 from the keyboard. Press the chapter key until the "inp-no." key is visible. Press the key.
3. Set parameter 2201 bit 0 (farthest to the right) to a zero.
4. Set parameter 2201 bit 1 (second from the right) to a zero.
5. Set parameter 2201 bit 2 (third form the right) to a one.
6. Enter 7050 and press the "inp-no." soft key. Enter 100 and press the "input" soft key.
7. Enter 7051 and press the "inp-no." soft key. Enter 101 and press the "input" soft key.
8. Enter 7052 and press the "inp-no." soft key . Enter 102 and press the "input" soft key.
9. Repeat this for parameters 7053 to 7079.

Entering the programs

You are now ready to input the programs. If you are entering the programs in the "MDI" mode there are some common mistakes to avoid. First, the letters used with the GOTO statement are not zeros. They are letters. Second, brackets are used with equations, not parentheses. Parentheses are used to put comments in a program and are ignored by the control. Finally, with all the "shift" key use required to enter one of these programs by hand, you can easily make a mistake. If the program does not run properly, carefully check for typographic errors.

The programs are available as "ASCII" files on 5¼" or 3½" IBM format floppy disks. The order form is at the end the manual. If you are using a computer to load programs, simply copy these files into your communications program and upload. The programs have the extensions .10M and .6M. Therefore, COPY A:\macro\*.* C: would copy the programs from the A floppy drive to the C hard drive. Be sure to read the up to date

README.DOC file on the disk. Change to the A: drive, enter TYPE README.DOC and press return to view the file.

Program Modification System 10-15

Fanuc offers three tool offset memory types. In memory type "A" there is only an offset for tool length. In memory "B" there is an offset for tool length and for tool length wear. In memory "C" there is an offset for tool length, tool length wear, tool radius and tool radius wear.

With memory type "A" the radius of the cutter is stored in a different number offset than the tool length. For example, tool 1 might use H1 for the tool length and D11 for the cutter radius.

The radius of the cutter is available to the program through a system variable. The pocketing program reads a particular system variable to calculate the compensation required to cut the proper size.

If your control has either of the memory types the program will still run. However, if you have type "B" or "C" you will want to modify the program to take advantage of the extra offset for wear. The following table lists the modification to the circle pocketing program for Type B and C:

MEMORY TYPE "B"	MEMORY TYPE "C"
N7(DELETE)	N7(DELETE)
N8#144=#[#4107+2000]	N8#144=[#4107+2400]
#145=#[#4107+2200]	#145=#[#4107+2600]
#147=#144+#145	#147=#144+#145
N9#10=#18-#147	N9#10=#18-#147

With these modifications, both the geometry and wear offsets will work with the circle pocketing program. For the counterbore command make the same modifications except for line N9. N9 will remain the same. N10 will become #100=#27-#147 for both memory types.

Application Programs

Using Custom Macro in the Shop

The Bolt Circle Cycle

The bolt circle command allows quick and easy programming of a circular pattern. Any of the standard "G" code drilling cycles can be used with the bolt circle command. The pattern can be centered anywhere on the part and the first hole can be at any angle. The information needed to use the command is listing in the following table:

X	X axis center.
Y	Y axis center.
Z	Depth to drill.
R	Reference point above part.
D	Diameter of bolt circle.
C	"G" code to be used.
H	Number of holes in the pattern.
A	Starting angle of first hole.
F	Feed rate in inches per minute.
T	Dwell value if G82 cycle is used.
Q	Peck interval if G73, G83 is used.

If the bolt circle pattern is centered at X0, Y0 a value is not required for X or Y. A value for T and Q are not required for a G81 drilling cycle. All other addresses are necessary to drill the pattern. The holes are normally drilled in a counter clockwise direction. However, if the value used for H is negative the direction will be clockwise. For example, if a four hole pattern is drilled H4.0 will proceed CCW. If H-4.0 is used the direction will be CW.

Decimal points must be used with all addresses except "T". If no decimal point is used the value will be taken as the least increment of the address. For example, if you use A15 the control will calculate based on .0015 degrees. The least increment in the inch system is .0001.

ALARM GENERATION

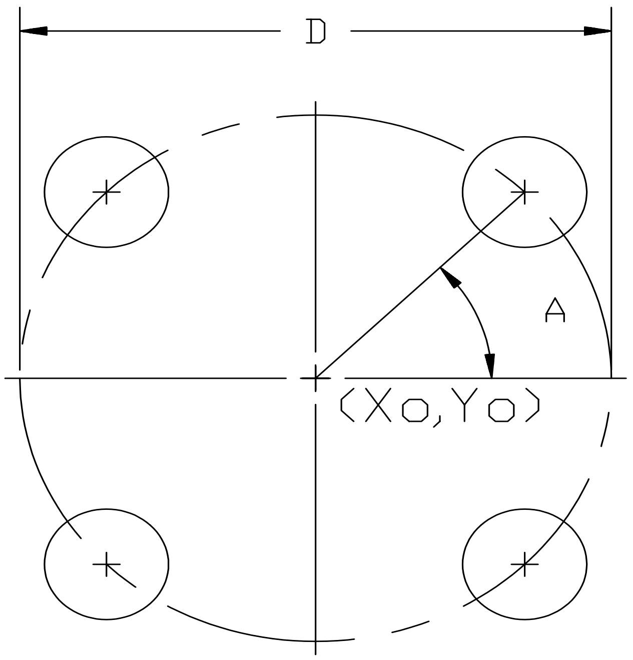
If a required address is not given an alarm will occur when the command is executed. See the Alarm list for a complete list of alarms.

HOW THE COMMAND WORKS

The next figure shows a typical bolt circle. The tool will rapid to the first hole. The particular cycle selected by "C" will be executed. The tool will return to the reference point given by "R" if G99 is in effect or to the initial point if G98 is in effect. The tool will then rapid to the next hole. The sequence will be repeated until all holes are completed.

The complete command for a 1.0 radius pattern located at X0.0 Y0.0 drilling 1.0 deep referenced .1 above the part with 4 holes starting at 45 degrees would be:

```
G100 X0.0Y0.0Z-1.0D2.0R.1C81A45.0H4.0F10.0
```



BOLT CIRCLE

The following program will machine the bolt circle in the previous example:

```
O0001;
N1 T1;
N2 M6;
N3 M3 S1000;
N4 G0 G43 G98 H1 X.707 Y.707 Z1.;
N5 G100 X0 Y0 Z-1. R.1 C81 D2. A45. H4. F10.;
N6 G0 G80 G28 Z1.0;
N7 M30;
```

Note: Make sure the tool doesn't hit a clamp or part of the fixture as it moves to the first hole.

Notice that G98 (return to initial point) was called in the G43 position line. G99 (return to reference point) could be used as well. The proper position in line N4 is not necessary. If the tool was drilling at a different location prior to the bolt circle, only the G100 would be necessary. The program will calculate the correct position and move to the first hole.

If a second bolt circle with completely different characteristics is to be machined with the same tool, a new line could be inserted after line N5 with the new parameters. For example, if a second circle with a diameter of 3.0 and a starting angle of 300.0 and 10 holes was required the following program would work:

```
O0001; program number.
N1 T1;
N2 M6;
N3 M3 S1000;
N4 G0 G43 G98 H1 X.707 Y.707 Z1.0;
N5 G100 X0 Y0 Z-1. R.1 C81 D2. A45. H4. F10.;
N6 G100 X0 Y0 Z-1. R.1 C81 D3. A300. H10. F10.;
N7 G0 G80 G28 Z1.0;
N8 M30;
```

Bolt Circle Skip Cycle G101 Program O9011

The bolt circle skip command allows quick and easy programming of a circular pattern with an certain number of holes missing. Any of the standard "G" code drilling cycles can be used with the bolt circle command. The pattern can be centered anywhere on the part and the first hole can be at any angle. The information needed to use the command is listing in the following table:

X	X-axis center.
Y	Y-axis center.
Z	Depth to drill.
R	Reference point above part.
D	Diameter of bolt circle.
C	"G" code to be used.
H	Number of holes in the pattern if none were skipped.
A	Starting angle of first hole.
F	Feed rate in inches per minute.
T	Dwell value if G82 cycle is used.
Q	Peck interval if G73,G83 is used.
B	Number of holes per group.
U	Number of holes to skip.

If the bolt circle pattern is centered at X0,Y0 a value is not required for X or Y. A value for T and Q are not required for a G81 drilling cycle. All other addresses are necessary to drill the pattern. The holes are normally drilled in a counter clockwise direction. However, if the value used for H is negative the direction will be clockwise. For example, if a four hole pattern is drilled H4.0 will proceed CCW. If H-4.0 is used the direction will be CW.

Decimal points must be used with all addresses except "T". If no decimal point is used the value will be taken as the least increment of the address. For example, if you use A15 the control will calculate the angle based on .0015 degrees. The least increment in the inch system is .0001.

ALARM GENERATION

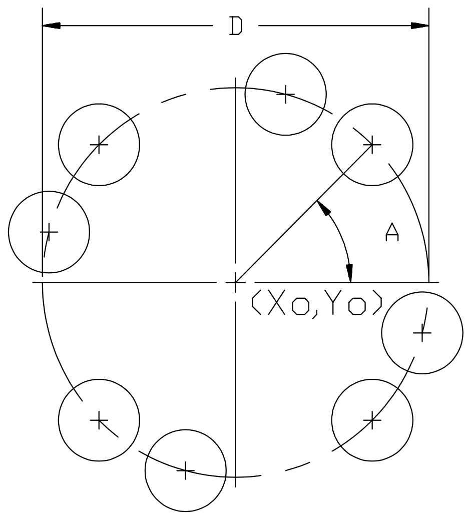
If a required address is not given an alarm will occur when the command is executed. See the Alarm list for a complete list of alarms.

HOW THE COMMAND WORKS.

The next figure shows a typical bolt circle with a skip pattern. The tool will rapid to the first hole. The particular cycle selected by "C" will be executed. The tool will return to the reference point given by "R" if G99 is in effect or the initial point if G98 is in effect. The tool will then rapid to the next hole. The sequence will be repeated until all holes in the first group are completed. The tool will then skip the number of holes given by "U" and repeat the process.

The complete command for the skip pattern shown in the figure is listed next. The circle is 2.5 inches in diameter located at X0, Y0. The holes will be drilled 1.0" deep. The tool is referenced .1" above the part. The pattern has 8 holes in a 12 hole spacing. The holes will be drilled in groups of 2 starting at 0.0 degrees.

G101Z-1.0D2.5R.1C81A45.H12.0B2.0U1.0F10.0



Bolt Circle Skip

The following program will machine the bolt circle in the previous example:

```
O0001;
N1 T1;
N2 M6;
N3 M3 S1000;
N4 G0 G43 G98 H1 X.866 Y.5 Z1.;
N5 G101 X0 Y0 Z-1. D2.5 R.1 C81 A45. H12. B2. U1. F10.;
N6 G0 G80 G28 Z1.0;
N8 M30;
```

Note: Make sure the tool doesn't hit a clamp or part of the fixture as it moves to the first hole.

Notice that G98 (return to initial point) was called in the G43 position line. G99 (return to reference point) could be caused as well. The proper position in line N4 is not necessary. If the tool was drilling at a different location prior to the bolt circle, only the G101 would be necessary. The program will calculate the correct position and move to the first hole.

If a second bolt circle will different characteristics is to be machined with the same tool, a line can be inserted after line N5 with the new parameters. For example, If a second circle with a diameter of 3.0 and a start angle of 0.0 is needed the following program will work:

```
O0001;
N1 T1;
N2 M6;
N3 M3 S1000;
N4 G0 G43 G98 H1 X.866 Y.5 Z1.;
N5 G101 X0 Y0 Z-1. D2.5 R.1 C81 A45. H12. B2. U1. F10.;
N6 G101 X0 Y0 Z-1. D3. R.1 C81 A0. H12. B2. U1. F10.;
N7 G0 G80 G28 Z1.0;
N8 M30;
```

Circular Pocket Cycle G102 Program O9012

The circular pocket command allows a complete pocket to be machined with one line of programming. The pocket can be centered anywhere on the part and a finish allowance can be left on the bottom and sides for a final pass. The command compensates for the size of the endmill. The radius of the endmill is stored in the offset page along with the tool length. The information needed to use the command is listing in the following table:

X	X axis center. If X=0 it is not required.
Y	Y axis center. If Y=0 it is not required.
Z	Depth of the pocket.
A	Reference point above the part.
R	Radius of the pocket.
F	Feed rate to be used while cutting.
C	Feed rate to use while plunging to depth.
D	Direction to cut. D2 CW. D3 CCW.
I	Allowance to be left for final cut. It is a radial value.
J	Finish allowance on bottom of pocket.
K	Peck interval if Z depth is to be roughed.
W	Width of cut. This is a decimal percent of the endmill diameter.

If a there is no predrilled hole in the center, the plunge feed rate can be lower than the cutting feed rate. The value given for "C" is the plunge feed rate. The direction of cutting can be changed from climb cutting (CCW) to conventional cutting with address "D". A value of 3 (D3) is climb cutting and a value of 2 (D2) is conventional. If a finish allowance is left on the bottom, the tool will return to center after the diameter is roughed out. The tool will then feed to the final depth and recut the bottom. This increases the machining time drastically.

Decimal points must be used with all addresses. If no decimal point is used the value will be taken as the least increment of the address. For example, if you use A15 the control will calculate based on .0015 degrees. The least increment in the inch system is .0001.

ALARM GENERATION

If a required address is not given an alarm will occur when the command is executed. See the alarm list for a complete list of alarms.

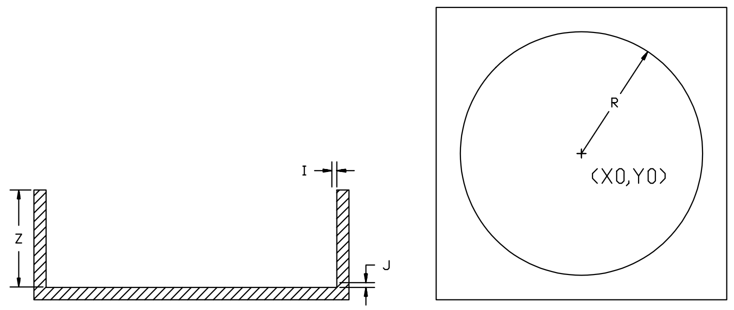
HOW THE COMMAND WORKS.

The next figure shows a typical pocket. The tool should be positioned in the center of the pocket. When the command is called the tool will feed to the Z depth using the plunge feed rate "C". If a peck cycle is selected the tool will feed "K" in depth. The tool will then feed one endmill diameter times the value in address "W" at the feed rate "F".

For example if a one inch endmill is used and "W" is .8 the tool will feed .8 inches. A series of circles will be cut until the finish allowance is reached. If the peck cycle is being used the tool will return to the center and plunge "K" again. A final rough cut will be taken and then the tool will feed to the final value and a cut will be taken. The tool will then feed out of the cut in a circular move. It will then rapid to the A plane and finally to the center of the pocket.

The information needed to machine the pocket in the figure is listed next. The diameter is 2.0". The pocket is .5" deep and is centered at X0 Y0. The finish allowance is .02", the plunge feed is 5.0 ipm, and the cutting feed rate is 20.0 ipm. Finally, the width of cut is eighty percent.

```
G102Z-.5R1.0A.05I.02J.02W.8F20.C5.0D3.
```



Circular Pocket

The following program will machine the circular pocket in the previous example:

```
O0001;
N1 T1;
N2 M6;
N3 M3 S1000;
N4 G0 G43 H1 D11 X0 Y0 Z.1;
N5 G102 Z-.5 R1. A.05 I.02 W.8 F20. C5. D3;
N6 G0 G28 Z.1;
N7 M30;
```

When I, J and K are used they must be in alphabetical order. The tool length would be stored in offset 1 and the endmill radius would be in offset 11. If extended tool offset memory (FANUC option) is available see "installation instructions" for a program modification.

Arc Mill Cycle G103 Program O9013

This command makes it simple to mill circular slots around the part. The number of slots required, the starting angle and the diameter of the slots are needed. In addition, the slot can ramp down in the Z axis. The pattern must be centered at X0,Y0. The information needed to use the command is listing in the following table:

D	Diameter of the slots.
A	The half angle of the slot. For a 30 degree long slot A=15.0
Z	The start depth of the slots.
R	The reference point above the part.
F	The feed rate to use while cutting.
H	The number of slots to cut.
B	The Z depth to end if cutting a ramped slot.
C	The feed rate to use while plunging.
S	The start angle of the first slot.

Decimal points must be used with all addresses. If no decimal point is used the value will be taken as the least increment of the address. For example, if you use A15 the control will calculate based on .0015 degrees. The least increment in the inch system is .0001.

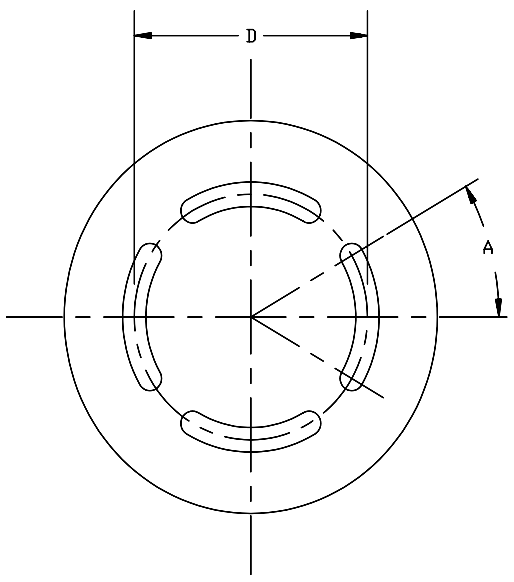
ALARM GENERATION

If a required address is not given an alarm will occur when the command is executed. See the alarm list for a complete list of alarms.

HOW THE COMMAND WORKS.

The next figure is a typical part with radial slots. The tool will rapid to the start point of the first slot. It will then rapid to the reference point. The feed rate given by "C" will be used to plunge to depth. This feed rate is usually lower than the feed used to mill the slot. The slot will then be cut at the rate given by "F". If a ramped slot is to be cut, the tool will move to a final Z depth given by "B". The tool will then rapid to the initial Z value and cut the next slot. The complete command to cut four 30.0 degree long slots is listed next. The slots are on a 3.0" diameter, .5" deep and start at 0.0 degrees. The reference point is .05" above the part.

G103D3.0A15.0Z-.5R.05F6.0C3.0H4.0S0.0



CIRCULAR SLOT CYCLE

The following program will machine the arcs in the previous example:

```
O0001;
N1 T1;
N2 M6;
N3 M3 S1000;
N4 G0 G43 H1 X1.5 Y0 Z1.0;
N5 G103 D3. A15. Z-.5 R.05 F6. C3. H4. S0.;
N6 G0 G28 Z1.0;
N7 M30;
```

The tool does not have to be positioned to the proper start Point. The program will calculate the correct point and move to it if necessary. If an arc that ramps down is required, address B" would be included. In this case the tool would feed down to depth "Z" and then cut an arc ending at depth "B". The tool moves in a counter clockwise direction. The starting angle is the angle at the center of the slot. In the example, the stating angle is 0. The slot is cut from 345 degrees to 15 degrees. This makes the slot 30 degrees total in length.

Ramped Arc Cycle G104 Program O9014

The ramp command makes it easy to program circular slots that ramp down to the center and then return to the start point. The information needed is the diameter of the slots, the start and end depth, the reference point, the number of degrees from the center of the slot (half of the included angle), the number of slots, and the feed rate. The information needed to use the command is listing in the following table:

D	The center diameter of the slots.
A	The angle from center. For a 30 degree slot A15.0
Z	The start depth of the slot.
F	The feed rate to cut.
C	The feed rate to plunge.
R	The reference point above the part.
J	The depth in the center of the arc.
S	The start angle of the first slot
H	The number of slots to cut.

Decimal points must be used with all addresses. If no decimal point is used the value will be taken as the least increment of the address. For example, if you use A15 the control will calculate based on .0015 degrees. The least increment in the inch system is .0001.

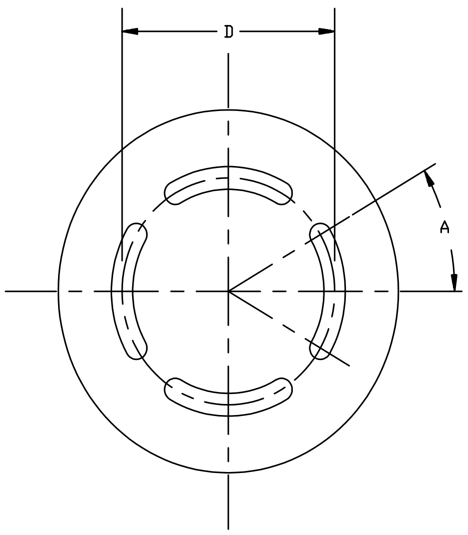
Alarm Generation

If a required address is not given an alarm will occur when the command is executed. See the alarm list for a complete list.

HOW THE COMMAND WORKS.

The following figure shows a slotted part. The angle A is 15.0 degrees. The radius is 1.50". The start depth is -.138". The end depth is -.188". The reference point is .05" above the part. The tool will rapid to the start point of 345.0 degrees. It will then rapid to .05" and feed to -.138". The first half of the slot will be cut to -.188" ending at 0.0 degrees. The second half will then be cut back to -.138" ending at 15.0 degrees. It will then rapid to .05" above the part. The feed and rapid moves will be repeated for the remaining slots. Finally the tool will rapid to the initial Z value. The complete command would be:

G104 D3.0 A15.0 Z-.138 J-.188 H4.0 R.05 S0.0 C3.0 F10.0



Ramped Arc Cycle

The following program will machine the arcs in the previous example:

```
O0001;
N1 T1;
N2 M6;
N3 M3 S1000;
N4 G0 G43 H1 X1.5 Y0.0 Z1.0;
N5 G104 Z-.138 J-.188 R.05 H4. S0. C3. F10. A15. D3.;
N6 G0 G28 Z1.0;
N7 M30;
```

The tool does not have to be positioned to the proper start point. The program will calculate the correct point and move to it if necessary. If the tool was milling at another location it will rapid to the start point.

Array Drilling Cycle G105 Program O9015

The array command provides an easy method of drilling an array of holes. Any of the standard "G" codes for drilling, boring and tapping can be used with the command. The pattern can be centered anywhere on the part and contain as many hole as necessary. The information needed to use the command is listing in the following table:

X	X axis start point. This is the lower left corner.
Y	Y axis start point. This is the lower left corner.
A	The number of holes in the Y axis direction.
B	The number of holes in the X axis direction.
I	The distance between holes in the X axis direction.
J	The distance between holes in the Y axis direction.
D	The offset in the "Y" axis if a zig zag pattern is needed.
F	The feed rate to use.
C	The "G" code to use.

Decimal points must be used with all addresses. If no decimal point is used the value will be taken as the least increment of the address. For example, if you use A15 the control will calculate based on .0015 degrees. The least increment in the inch system is .0001.

Alarm Generation

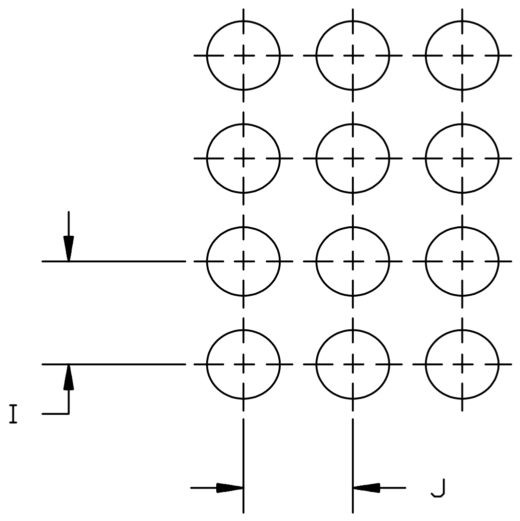
If a required address is not given an alarm will occur when the command is executed. See the alarm list for a complete list of alarms.

HOW THE COMMAND WORKS.

The next figure shows a typical array of holes. The tool will rapid to the hole in the lower left corner defined by X and Y. The selected "G" cycle will be executed and the next hole in the positive Y axis will be drilled. When all holes in the first column are drilled the tool will move to the next hole in the positive X direction. The holes in this column will the be completed. This motion is repeated until all holes are drilled.

The complete command to drill an array of 3 columns by 4 rows is listed next. The hole spacing is 1.0" in both X and Y. The depth is .5" and the reference point is .05".

G105X2.0Y3.0Z-.5R.05F5.0A4.0B3.0I1.0J1.0C81



Array Drilling Cycle

The following program will drill the array pattern in the previous example:

```
O0001;
N1 T1;
N2 M6;
N3 M3 S1000;
N4 G0 G98 G43 H1 X2. Y3. Z1.;
N5 G105 X2. Y3. A4. B3. I1. J1. C81 R.05 Z-.5 F5.;
N6 G0 G28 G80 Z1.;
N7 M30;
```

Notice that G98 (return to initial point) was called in the G43 position line. G99 (return to reference point) could be used as well. If an offset in the Y axis is needed address "D" would be inserted. By putting in an offset every other column of holes will have Y axis centers higher or lower than the previous column. The columns will be higher if "D" is positive and lower if "D" is negative.

Counterbore Cycle G106 Program O9016

The counterbore command uses an endmill to counterbore an existing hole. The counterbore can be located anywhere on the part. The command compensates for the radius of the endmill. The radius is stored in the offset page along with the tool length offset. The information needed to use the command is listing in the following table:

Z	Depth of the counterbore.
R	Reference point above the part.
D	Diameter of the counterbore.
F	The feed rate to use.

Decimal points must be used with all addresses. If no decimal point is used the value will be taken as the least increment of the address. For example, if you use A15 the control will calculate based on .0015 degrees. The least increment in the inch system is .0001.

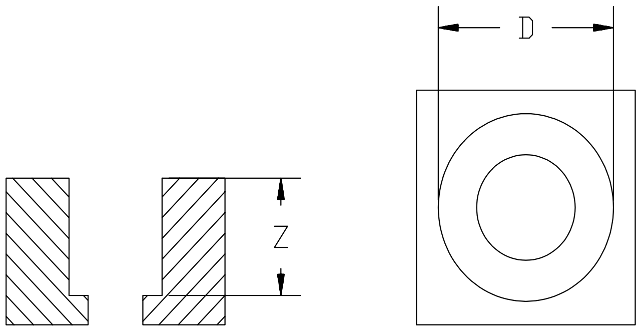
Alarm Generation

If a required address is not given an alarm will occur when the command is executed. See the alarm list for a complete list of alarms.

HOW THE COMMAND WORKS.

The next figure shows a typical counterbore. The tool must be positioned to the hole center. The tool will rapid to the reference point (R). It will then feed at the given feed rate to the bottom of the counterbore(Z). The tool will feed out the required distance and circular interpolate the counterbore. The tool will then feed back to the center of the hole at 20 IPM. Finally, The tool will rapid to the initial Z value. The complete command for a .75 diameter counterbore is listed next. The counterbore is .50" deep, referenced .10" above the part.

```
G106 Z-.50 R.10 D.75 F5.0
```



Counter Bore Cycle

The following program will machine the counterbore in the previous example at a location of X2.0 Y3.0:

```
O0001;
N1 T1;
N2 M6;
N3 M3 S1000;
N4 G0 G43 H1 D11 X2. Y3. Z1.0;
N5 G106 Z-.5 R.1 D.75 F5.0;
N6 G0 G28 Z1.0;
N7 M30;
```

This is a simple, easy to use program. However, it can save a lot of time if the proper size counter bore is not available. Simply put the endmill diameter in the proper offset and produce the hole. A program using Fanuc's cutter compensation or fixed G3 codes doesn't have to be written and tested. Variations in endmill diameter or tool wear can be easily corrected using a tool offset.

Line At Angle Cycle G107 Program O9017

The line at angle makes it easy to drill a series of holes on a line. The line can be at any angle. All the standard "G" codes for drilling and boring can be used with this command. The information needed to use the command is listing in the following table:

X	X axis position of first hole.
Y	Y axis position of first hole.
Z	Z depth to drill.
A	Angle of line from X axis.
H	Number of holes to drill.
I	Distance between holes.
C	"G" code to be used.
F	Feed rate to use.
T	Dwell value if G82 is used.
Q	Peck interval if G73, G83 is used.

Decimal points must be used with all addresses except "T". If no decimal point is used the value will be taken as the least increment of the address. For example, if you use A15 the control will calculate based on .0015 degrees. The least increment in the inch system is .0001.

ALARM GENERATION

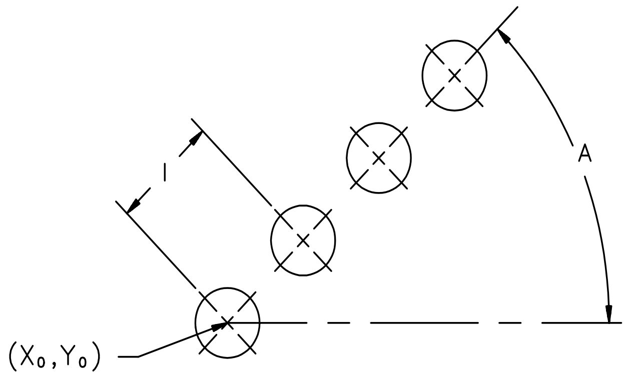
If a required address is not given an alarm will occur when the command is executed. See the alarm list for a complete list of alarms.

HOW THE COMMAND WORKS.

The next figure is a typical line at an angle. The tool will rapid to the first hole. The particular cycle selected by "C" will be executed. The tool will then rapid to the next hole. The sequence will be repeated until all holes are completed. The tool will then return to the reference point.

The complete command for a line with four holes 1 inch apart is listed. The holes are to be drilled 1.0 inch deep, referenced .1 inch above the part. The angle is 26 degrees.

G107 X0.0 Y0.0 A26.0 I1.0 C81 Z-1.0 R.1 F10.0



Line At Angle

The following program will drill the line in the previous example:

```
O0001;
N1 T1;
N2 M6;
N3 M3 S1000;
N4 G0 G43 G98 H1 X0 Y0 Z1.0;
N5 G107 X0 Y0 Z-1. R.1 F10. C81 A26. I1.;
N6 G0 G91 G28 Z0;
N7 M30;
```

Notice that G98 (return to initial point) was called on the line that positions the tool. G99 (return to reference point) could be used as well.

Slot Mill Cycle G108 Program O9018

The slot mill command machines slots parallel to a line through the diameter of the part. The slots can be at any given start angle and any number of slots can be cut. The information needed to use the command is listing in the following table:

Z	Depth to mill the slot.
R	Reference point to rapid to.
D	Diameter to start milling.
B	Diameter to finish milling.
H	Number of slots to cut around the part.
C	Feed rate to plunge cut with.
F	Feed rate to be used during slotting.
A	The start angle of the first slot.

Decimal points must be used with all addresses. If no decimal point is used the value will be taken as the least increment of the address. For example, if you use A15 the control will calculate based on .0015 degrees. The least increment in the inch system is .0001.

ALARM GENERATION

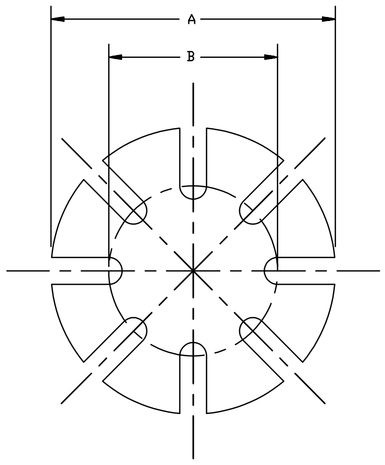
If a required address if not given an alarm will occur when the command is executed. See the alarm list for a complete explanation of the alarm.

HOW THE COMMAND WORKS.

The next figure shows a typical slotted part. The tool will rapid to the start diameter "D". The tool will then rapid to the reference point. Using the feed rate "C" it will feed to a depth "Z". The feed rate will then change to "F" and cut to the finish diameter "B" at the angle "A". The tool will then feed to the reference point. It will then rapid to the initial point and position to the start diameter of the next slot.

The complete command to cut 8 slots, .25" deep is listed next. The starting diameter is 2.0 inches and the finish diameter is 1.0 inches. The feed rate to plunge is 5.0 ipm, the feed rate to cut is 10.0. The reference point is .1" above the part and the start angle is 0.0 degrees.

G108 D2.0 B1.0 S0.0 C5.0 F8.0 R.1 Z-.25 H8.0;



Slot Milling Cycle

The following program will machine the slots described in the previous example:

```
O0001;
N1 T1;
N2 M6;
N3 M3 S1000;
N4 G0 G43 H1 X2.0 Y0.0 Z1.0;
N5 G108 D2. B1. S0. C5. F8. R.1 Z-.25 H8.;
N6 G0 G28 Z1.0;
N7 M30;
```

The tool does not have to be positioned to the proper start point. The command will position to the beginning and cut the slots in the proper location. The diameter the slots are cut on must be centered at X0 Y0. The program calculates the angular move based on a center of X0 Y0.

Macro Alarm List

Alarms used by Programming Unlimited

MC001 No Diameter (D)

A diameter value "D" is required to machine the pattern. The address used is "D".

MC002 No Hole Value (H)

The number of holes in the pattern must be given. The address used is "H".

MC003 No Feed Rate (F)

A feed rate "F" in inches per minute must be given. The address used is "F".

MC004 No Drill Cycle (C)

A "G" cycle must be given at address "C". C73, C74, C81, C82, C83, C84 which correspond to G73, G74, G81, G82, G83, G84 respectively.

MC005 No Width of Cut in a Pocket Cycle (W)

In a pocketing cycle no width of cut was given. The address used is "W".

MC006 Number of Slots Missing (H)

The number of slots to cut was not given. The address used is "H".

MC007 No Drill Depth (Z)

No "Z" depth to drill or mill was given. The address used is "Z".

MC008 No Plunge Feed Rate (C)

No feed rate "C" to use while plunging was given. The address used is "C".

MC009 No Start Angle in Slotting (A)

The start angle of the first the slot was not given. The address used is "S".

MC010 No Angle in Line at Angle (A)

No angle from the "X" axis was given in the line at angle command. The address used is "A".

MC011 Number of Holes in the Y axis (A)

The number of holes in the "Y" axis is missing in the array cycle. The address used is "A".

MC012 Number of Holes in the X axis (B)

The number of holes in the "X" axis is missing in the array cycle. The address used is "B".

MC013 Distance between holes in X axis (I)

The distance between holes in the "X" axis is missing in the array cycle. The address used is "I".

MC014 Distance between holes in Y axis (J)

The distance between holes in the "Y" axis is missing in the array cycle. The address used is "J".

MC015 Start point in the X axis (X)

The start point in the "X" axis is missing in the array cycle. The address used is "X".

MC016 Start point in the Y axis (Y)

The start point in the "Y" axis is missing in the array cycle. The address used is "Y".

MC017 Distance between Holes in Line Cycle (I)

The distance between holes was not given in the line at angle cycle. The address used is "I".

MC018 The Angle from Center is missing (A)

The angle from center of the slot was not given. The address used is "A".

MC020 Reference Point is Missing (R)

The reference point above the part is missing. The address used is "R".

MC021 Reference Point is Missing in Pocket Cycle (A)

The reference point above the part was not given in the pocketing cycle G103. The address used is "A".

MC022 Diameter to Mill in a Slotting Cycle (D)

The diameter to start milling the slots was not given. The address used is "D".

MC023 Diameter to Stop Milling (B)

The diameter to stop milling the slots was not given. The address used is "B".

Program Listings

System 10-15 Programs

Bolt Circle Macro

```
O9010(BOLT HOLE MACRO);
IF[#7EQ#0]THEN #3000=1(NO DIAMETER GIVEN);
IF[#11EQ#0]THEN #3000=2(NO HOLES GIVEN);
IF[#9LT#0]THEN #3000=3(NO FEEDRATE "F" GIVEN);
IF[#3EQ#0]THEN #3000=4(NO DRILL CYCLE GIVEN);
IF[#18EQ#0]THEN #3000=20(NO REFERENCE POINT GIVEN);
#100=1;
#27=#7/2.0;
G0X[[COS[#1]*#27]+#24]Y[[SIN[#1]*#27]+#25];
G98G[#3*10000]Z#26R#18F#9Q#17P#20;
#101=360/#11;
N1WHILE[#100LT ABS[#11]]DO1;
X[[[COS[[#101*#100]+#1]*#27]+#24]]Y[[[SIN[[#101*#100]+#1]*#27]+#25]];
#100=#100+1;
END1;
M99;
(X=X AXIS CENTER);
(Y=Y AXIS CENTER);
(Z=Z AXIS DEPTH);
(R=REFERENCE POINT ABOVE PART);
(T=DWELL VALUE FOR G82);
(Q=PECK INTERVAL FOR G73,G83);
(H=NUMBER OF HOLES TO DRILL);
(A=START ANGLE OF FIRST HOLE);
(D=DIAMETER OF BOLT CIRCLE);
(C=DRILL CYCLE TO USE);
M30;
```

Bolt Circle Skip Macro

```
O9011(BLT-HOL-SKIP);
IF[#7EQ#0]THEN#3000=1(NO DIAMETER GIVEN);
IF[#11EQ#0]THEN#3000=2(NO "H" GIVEN);
IF[#9EQ#0]THEN#3000=3(NO "F" GIVEN);
IF[#3EQ#0]THEN#3000=4(NO DRILL CYCLE C);
#100=1;
#19=1;
#27=#7/2.0;
G0X[[COS[#1]*#27]+#24]Y[[SIN[#1]*#27]+#25];
G[#3*10000]Z#26R#18F#9Q#17P#20;
#101=360/#11;
N1WHILE[#100LT ABS[#11]]DO1;
N2WHILE[#19LT#2]DO2;
X[[[COS[#101*#100+#1]*#7]+#24]]Y[[[SIN[[#101*#100]+#1]*#7]+#25]];
#19=#19+1;
#100=#100+1;
END2;
#19=0;
#100=#100+#21;
END1;
M99;
(X=CTR IN X AXIS);
(Y=CTR IN Y AXIS);
(Z=DEPTH OF HOLE);
(R=REFERENCE PT ABOVE PART);
(D=DIAMETER OF CIRCLE);
(A=START ANGLE OF FIRST HOLE);
(H=NUMBER OF HOLES IN PATTERN);
(S=NUMBER OF HOLES TO DRILL);
(B=NUMBER OF HOLES IN GROUP);
(C=CANNED CYCLE TO USE);
(T=DWELL TO USE EX. T500);
(Q=PECK INTERVAL EX .5 IN);
(U=NUMBER OF HOLES TO SKIP);
M99;
```

Circle Pocket Macro

```
O9012(CIRCLE-POCKET);
IF[#3EQ#0]THEN#3000=8(NO PLUNGE FEEDRATE GIVEN);
IF[#9EQ#0]THEN#3000=3(NO MILLING FEEDRATE GIVEN);
IF[#23EQ#0]THEN#3000=5(NO WIDTH OF CUT GIVEN);
IF[#1EQ#0]THEN#3000=21(NO REFERENCE POINT);
N1G0X#24Y#25Z#1;
#100=0;
#106=0;
#140=#5003;
#141=#5001;
#142=#5002;
#144=#[#4107+2000]; (D+2000)
#10=#18-#144; (POCKET RADIUS - ENDMILL RADIUS)
#143=#142-[#10-#4];
#145=[#10-#4];
#146=2*#145;
#12=#23*[#144*2]; (WIDTH-OF-CUT)
#104=[#12*.01];
#27=#26+#5; (Z DEPTH - FINISH)
IF[#7EQ2.0]#104=-#104;
IF[#6EQ#0]GOTO3000;
#100=ABS[FIX[[#26-#1+#5]/#6]];
#101=#100;
N2G90G0Z#1;
N3G0G90X#141Y#142;
N500IF[#106GE#100]GOTO3000;
#14=0;
G91G1Z-#6F#3;
WHILE[[#12+#14]LT[#145]]DO2;
N4G1G91Y-#12F#3;
#14=#14+#12;
N5G#7X0Y[2*#14]R#14F#9;
N6G#7X0Y-[2*#14]R#14;
N31G0Y0.0;
N32END2;
N2000#106=#106+1;
N19G90G1Y[#143];
G#7G91X0.0Y#146R145F#9;
G#7X0.0Y-#146R145;
G0Y0;
N8G90G0X#141Y#142;
N201GOTO500;
N3000IF[#5EQ#0]GOTO4000;
N9G90G0X#141Y#142;
#14=0;
```

Circle Pocket Macro

```
N10G1Z#27F#3;
N40WHILE[[#12+#14]LT[#145]]DO3;
N11G1G91Y-#12F#3;
#14=#14+#12;
N12G#7X0Y[2*#14]R#14F#9;
N13G#7X0Y-[2*#14]R#14;
N14G0Y0;
END3;
N19G90G1Y#143;
G#7G91X0.0Y#146R#145F#9;
G#7X0.0Y-#146R#145;
G0Y0;
N4000G0G90X#141Y#142;
N14G1Z#26F#3;
#14=0;
WHILE[[#12+#14]LT[#145]]DO1;
G1G91Y-#12F#3;
#14=#12+#14;
G#7X0Y[2*#14]R#14F#9;
G#7X0Y-[2*#14]R#14;
G0Y0;
END1;
N1000IF[#4EQ#0]GOTO23; (IF NO FIN ALLOW)
N19G90G1Y#143;
G#7G91X0.0Y#146R#145F#9;
G#7X0.0Y-#146R#145;
G0Y0;
IF[#5003NE#26]GOTO4000;
N23G90G1Y[#142-#10];
N24G#7G91X0Y[2*#10]R#10F#9;
N25G#7X0Y-[2*#10]R#10;
N26G#7X#104Y[ABS[#104]]R[ABS[#104]];
N67G0G90Z#140;
G0X#141Y#142;
M99;
```

Radial Slotting Macro

```
O9013(RADIAL SLOT);
IF[#7EQ#0]THEN#3000=1(NO DIAMETER GIVEN);
IF[#1EQ#0]THEN #3000=9(NO START ANGLE GIVEN);
IF[#11EQ#0]THEN #3000=6(NUMBER OF SLOTS MISSING);
#27=#7/2;
#100=0;
#101=360/#11;
G0X[#27*COS[#19-#1]]Y[#27*SIN[#19-#1]];
WHILE[#100LT#11]DO1;
N1G0Z#18;
G1Z#26F#3;
G3X[#27*COS[#100*#101]]Y[#27*SIN[#19-#1]]R#27Z#2F#9;
G0Z#23;
G0X[#27*COS[[#100*#101]+[#19+#1]](CONTINUED ON NEXT LINE)
Y[#27*SIN[[#100*#101]+[#19-#1]];
END1;
M99;
```

Radial Slot With Ramp Macro

```
O9014(SLOT W/ RAMP);
IF[#7EQ#0]THEN #3000=1(NO DIAMETER GIVEN);
IF[#1EQ#0]THEN #3000=18(NO SLOT ANGLE GIVEN);
IF[#11EQ#0]THEN #3000=6(NUMBER OF SLOTS NOT GIVEN);
IF[#3EQ#0]THEN #3000=8(NO PLUNGE FEED GIVEN);
IF[#9EQ30]THEN #3000=3(NO FEEDRATE GIVEN);
IF[#18EQ#0]THEN #3000=20(NO REFERENCE POINT);
IF[#26EQ30]THEN #3000=7(NO "Z" DEPTH GIVEN);
#140=#5003;
#27=#7/2;
#100=0;
#101=360/#11;
#102=#11;
G0X[#27*COS[#19-#1]]Y[#27*SIN[#19-#1]];
WHILE[#100LT#102]DO1;
N1G0Z#18;
G1Z#26F#3;
G3X[#27*COS[[#100*#101]+#19]](CONTINUED ON NEXT LINE)
Y[#27*SIN[[#100*#101]+#19]]R#27Z#5;
G3X[#27*COS[[#100*#101]+[#19+#1]]](CONTINUED ON NEXT LINE)
Y[#27*SIN[[#100*#101]+[#19+#1]]]R#27Z#26;
G0Z#140;
#100=#100+1;
G0X[#27*COS[[#100*#101]+[#19-#1]]](CONTINUED ON NEXT LINE)
Y[#27*SIN[[#100*#101]+[#19-#1]]];
END1;
G0Z#5003;
M99;
```

ARRAY OF HOLES MACRO

```
O9015(ARRAY);
IF[#1EQ#0]THEN #3000=11(HOLE IN "Y" NOT GIVEN);
IF[#2EQ#0]THEN #3000=12(HOLES IN "X" NOT GIVEN);
IF[#4EQ#0]THEN #3000=13(NO DISTANCE IN "X");
IF[#5EQ#0]THEN #3000=14(NO DISTANCE IN "Y");
IF[#24EQ#0]THEN #3000=15(START IN "X" NO GIVEN);
IF[#25EQ#0]THEN #3000=16(START IN "Y" NOT GIVEN);
G0X#24Y#25;
#140=#5003;
G[#3*10000]Z#26R#18F#9P#7Q#17;
#27=#1*#2(TOTAL HOLES);
#30=#1-1(HOLES IN Y-1);
#28=1;
N1DO1;
N2WHILE[#29LT#30]DO2;
G91Y#5;
#29=#29+1;
#28=#28+1;
END2;
IF[#27-#28EQ0]GOTO4;
IF[#27-#28LT0]GOTO4;
#29=0;
G91X#4Y#7;
#28=#28+1;
#31=0;
N3WHILE[#31LT#30]DO3;
Y-#5;
#31=#31+1;
#28=#28+1;
END3;
#28=#28+1;
IF[#27-#28EQ0]GOTO4;
IF[#27-#28LT0]GOTO4;
X#4Y-#7;
END1;
N4G90G0Z#140;
M99;
(X,Y START POINT IN LOWER LEFT);
(A=NUMBER OF HOLES IN Y);
(B=NUMBER OF HOLES IN X);
(I=DISTANCE IN X);
(J=DISTANCE IN Y);
M30;
```

COUNTERBORING MACRO

```
O9016(COUNTERBORING CYCLE);
N1IF[#7EQ#0]THEN #3000=1(NO DIAMETER GIVEN);
N2IF[#9EQ#0]THEN #3000=3(NO FEEDRATE GIVEN);
N3IF[#18EQ#0]THEN #3000=20(NO REFERENCE POINT);
IF[#26EQ#0]THEN #3000=7(NO "Z" DEPTH GIVEN);
N4#140=#5003(STORE INITIAL Z PT.);
N7#143=#4107+2000(H + 2000);
N8#144=#[#143](RADIUS OF END MILL);
N9#27=#7/2;
N10#100=#27-#144(RAD OF CTRBORE - RAD OF EM);
#101=#100*2;
G90G0Z#18(RAPID TO REF PT.);
G1Z#26F#9(FEED TO DEPTH);
G1G91Y-#100;
G3G91X0.0 Y#101R#100;
G3G91X0.0Y-#101R#100;
G1Y#100F20.;
G0G90Z#140;
M99;
(Z=DEPTH OF COUNTERBORE);
(D=DIAMETER OF COUNTERBORE);
(R=REFERENCE POINT ABOVE PART);
(F=FEEDRATE);
M30;
```

Line At Angle Macro

```
O9017(LINE-ANGLE);
IF[#11EQ#0]THEN #3000=2(NO H VALUE GIVEN);
IF[#9EQ#0]THEN #3000=3(NO FEEDRATE GIVEN);
IF[#3EQ#0]THEN #3000=4(NO DRILL CYCLE "C" GIVEN);
IF[#26EQ#0]THEN #3000=7(NO Z DEPTH GIVEN);
IF[#24EQ#0]THEN #3000=15(NO "X" START PT. GIVEN);
IF[#25EQ#0]THEN #3000=16(NO "Y" START PT. GIVEN);
IF[#1EQ#0]THEN #3000=10(NO ANGLE GIVEN);
IF[#4EQ#0]THEN#3000=17(NO I VALUE GIVEN);
#100=1;
G0X#24Y#25;
G98G[#3*10000]Z#26R#18F#9Q#17P#20;
WHILE[#100LT#11]DO1;
N1G91X[COS[#1]*#4]Y[SIN[#1]*#4];
#100=#100+1;
END1;
M99;
(I DISTANCE BETWEEN HOLES);
(A ANGLE FROM X AXIS CCW);
M30;
```

Slotting Macro

```
O9018(SLOT MACRO);
IF[#11EQ#0]THEN #3000=6(NUMBER OF SLOTS NOT GIVEN);
IF[#7EQ#0]THEN #3000=22(NO START DIAMETER GIVEN);
IF[#2EQ#0]THEN #3000=23(NO END DIAMETER GIVEN);
IF[#18EQ#0]THEN #3000=20(N0 REFERENCE POINT GIVEN);
IF[#9EQ#0]THEN #3000=3(NO FEEDRATE GIVEN);
IF[#3EQ#0]THEN #3000=8(NO PLUNGE FEEDRATE GIVEN);
#140=#5003;
#31=1;
#101=360/#11;
G0X[COS[[#101*#31]+#1]*#7]Y[SIN[[#101*#31]+#1]*#7];
WHILE[#31LT#11]DO2;
G0X[COS[[#101*#31]+#1]*#7]Y[SIN[[#101*#31]+#1]*#7];
G0Z#18;
G1Z#26F#3;
G1X[COS[[#101*#31]+#1]*#2]Y[SIN[[#101*#31]+#1]*#2]F#9;
G1Z#18;
G0Z#140;
#31=#31+1;
N2;
M99;
```

Controls Without IF-Then Capability

System 6M Style Programs

Controls like the Fanuc 6M and Mazak M32 don't support the IF-THEN statement unfortunately. For these types of controls use the following Programs with GOTO statements.

BOLT CIRCLE MACRO

```
O9010(BOLT HOLE MACRO) ;
IF[#7EQ#0]GOTO2001;
IF[#11EQ#0]GOTO2002;
IF[#9LT#0]GOTO2003;
IF[#3EQ#0]GOTO2004;
IF[#18EQ#0]GOTO2005;
#100=1;
#27=#7/2.0;
G0X[[COS[#1]*#27]+#24]Y[[SIN[#1]*#27]+#25];
G[#3*10000]Z#26R#18F#9Q#17P#20;
#101=360/#11;
N1WHILE[#100 LT ABS[#11]]DO1;
X[[[COS[[#101*#100]+#1]*#27]+#24]]Y[[[SIN[[#101*#100]+#1]*#27]+#2 5]];
#100=#100+1;
END1;
M99;
N2001#3000=1(NO B.C. DIAMETER GIVEN);
N2002#3000=2(NO HOLES GIVEN);
N2003#3000=3(NO FEEDRATE "F" GIVEN);
N2004#3000=4(NO DRILL CYCLE GIVEN);
N2005#3000=20(NO REFERENCE POINT GIVEN);
(X=X AXIS CENTER);
(Y=Y AXIS CENTER);
(Z=Z AXIS DEPTH);
(R=REFERENCE POINT ABOVE PART);
(T=DWELL VALUE FOR G82);
(Q=PECK INTERVAL FOR G73,G83);
(H=NUMBER OF HOLES TO DRILL);
(A=START ANGLE OF FIRST HOLE);
(D=DIAMETER OF BOLT CIRCLE);
(C=DRILL CYCLE TO USE);
M30;
```

Note: On some 6M controls this program generates an error. If this happens, change the G[#3*10000] to G#3. See the line in bold text.

BOLT CIRCLE SKIP MACRO

```
O9011(BLT-HOL-SKIP)
IF[#7EQ#0]GOTO2001
IF[#11EQ#0]GOTO2002
IF[#9EQ#0]GOTO2003
IF[#3EQ#0]GOTO2004
IF[#18EQ#0]GOTO2005
#100=1
#19=1
#27=#7/2.0
G0X[[COS[#1]*#27]+#24]Y[[SIN[#1]*#27]+#25]
G[#3*10000]Z#26R#18F#9Q#17P#20
#101=360/#11
N1WHILE[#100LTABS[#11]]DO1
N2WHILE[#19LT#2]DO2
X[[[COS[#101*#100+#1]*#27]+#24]]Y[[[SIN[[#101*#100]+#1]*#27]+#25]]
#19=#19+1
#100=#100+1
END2
#19=0
#100=#100+#21
END1
M99
N2001#3000=1(NO DIAMETER GIVEN)
N2002#3000=2(NO H GIVEN)
N2003#3000=3(NO FEEDRATE GIVEN)
N2004#3000=4(NO DRILL CYCLE GIVEN)
N2005#3000=20(NO REFERENCE POINT GIVEN)
(X=CTR IN X AXIS)
(Y=CTR IN Y AXIS)
(Z=DEPTH OF HOLE)
(R=REFERENCE PT ABOVE PART)
(D=DIAMETER OF CIRCLE)
(A=START ANGLE OF FIRST HOLE)
(H=NUMBER OF HOLES IN PATTERN)
(B=NUMBER OF HOLES IN GROUP)
(C=CANNED CYCLE TO USE)
(T=DWELL TO USE EX. .5 SEC.)
(Q=PECK INTERVAL EX .5 IN)
(U=NUMBER OF HOLES TO SKIP)
M30
```

Note: On some 6M controls this program generates an error. If this happens, change the G[#3*10000] to G#3. See the line in bold text.

CIRCLE POCKET MACRO

```
O9012(CIRCLE-POCKET)
IF[#3EQ#0]GOTO2001;
IF[#9EQ#0]GOTO2002;
IF[#23EQ#0]GOTO2003;
IF[#1EQ#0]GOTO2004;
N1G0X#24Y#25Z#1;
#100=0;
#106=0;
#140=#5003;
#141=#5001;
#142=#5002;
#144=#[#4107+2000]; (D+2000)
#10=#18-#144; (POCKET RADIUS - END MILL RADIUS)
#143=#142-[#10-#4];
#145=[#10-#4];
#146=2*#145;
#12=#23*[#144*2]; (WIDTH OF CUT)
#104=[#12*.01];
#27=#26+#5; (Z DEPTH - FINISH)
IF[#7EQ2.0]#104=-#104;
IF[#6EQ#0]GOTO3000;
#100=-[FUP[[#26-#1]/#6]+2];
#100=ABS[FIX[[#26-#1+#5]/#6]];
#101=#100;
N2G90G0Z#1;
N3G0G90X#141Y#142;
N500IF[#106GE#100]GOTO3000;
#14=0;
G91G1Z-#6F#3;
WHILE[[#12+#14]LT[#145]]DO2;
N4G1G91Y-#12F#3;
#14=#14+#12;
N5G#7X0Y[2*#14]R#14F#9;
N6G#7X0Y-[2*#14]R#14;
N31G0Y0.0;
N32END2;
N2000#106=#106+1;
N19G90G1Y[#143];
G#7G91X0.0Y#146R145F#9;
G#7X0.0Y-#146R145;
G0Y0;
N8G90G0X#141Y#142;
N201GOTO500;
N3000IF[#5EQ#0]GOTO4000;
N9G90G0X#141Y#142;
```

```
#14=0;
N10G1Z#27F#3;
N40WHILE[[#12+#14]LT[#145]]DO3;
N11G1G91Y-#12F#3;
#14=#14+#12;
N12G#7X0Y[2*#14]R#14F#9;
N13G#7X0Y-[2*#14]R#14;
N14G0Y0;
END3;
N19G90G1Y#143;
G#7G91X0.0Y#146R#145F#9;
G#7X0.0Y-#146R#145;
G0Y0;
N4000G0G90X#141Y#142;
N14G1Z#26F#3;
#14=0;
WHILE[[#12+#14]LT[#145]]DO1;
G1G91Y-#12F#3;
#14=#12+#14;
G#7X0Y[2*#14]R#14F#9;
G#7X0Y-[2*#14]R#14;
G0Y0;
END1;
N1000IF[#4EQ#0]GOTO23; {IF NO FIN ALLOW)
N19G90G1Y#143;
G#7G91X0.0Y#146R#145F#9;
G#7X0.0Y-#146R#145;
G0Y0;
IF[#5003NE#26]GOTO4000;
N23G90G1Y[#142-#10];
N24G#7G91X0Y[2*#10]R#10F#9;
N25G#7X0Y-[2*#10]R#10;
N26G#7X#104Y[ABS[#104]]R[ABS[#104]];
N67G0G90Z#140;
G0X#141Y#142;
M99;
N2001#3000=8(NO PLUNGE FEEDRATE GIVEN);
N2002#3000=3(NO FEEDRATE GIVEN);
N2003#3000=5(NO WIDTH OF CUT GIVEN);
N2004#3000=21(NO REFERENCE POINT);
M30;
```

Arc Mill Macro

```
O9013(RADIAL ARC)
IF[#7EQ#0]GOTO2001
IF[#1EQ#0]GOTO2002
IF[#11EQ#0]GOTO2003
IF[#3EQ#0]GOTO2004
IF[#9EQ#0]GOTO2005
IF[#1EQ#0]GOTO2006
#140=#5003
#27=#7/2
#100=0
#101=360/#11
#102=#11
G0X[#27*COS[#19-#1]]Y[#27*SIN[#19-#1]]
WHILE[#100LT#102]DO1
N1G0Z#18
G1Z#26F#3
G3X[#27*COS[[#100*#101]+[#19+#1]]]Y[#27*SIN[[#100*#101]+[#19+#1]]]R#27Z#2F#9
G0Z#140
#100=#100+1
G0X[#27*COS[[#100*#101]+[#19-#1]]]Y[#27*SIN[[#100*#101]+[#19-#1]]]
END1
G0Z#5003
M99
N2001#3000=1(NO DIAMETER GIVEN)
N2002#3000=9(NO START ANGLE "S" GIVEN)
N2003#3000=2(NO HOLE VALUE "H" GIVEN)
N2004#3000=8(NO PLUNGE FEED "C" GIVEN)
N2005#3000=3(NO CUTTING FEED "F" GIVEN)
N2006#3000=18(NO HALF ANGLE "A" WAS GIVEN)
(Z=DEPTH TO START SLOTS)
(D=DIAMETER OF SLOTS)
(B=DEPTH TO END SLOT IF A RAMPED SLOT IS TO BE CUT)
(C=PLUNGE FEED)
(F=FEED TO CUT WITH)
(A=ANGLE OF SLOT FROM CENTERLINE)
(S=ANGLE OF CENTERLINE OF FIRST SLOT)
(H=NUMBER OF SLOTS TO CUT)
M30
```

Ramped Arc Mill Macro

```
O9014(SLOT W/ RAMP)
IF[#7EQ#0]GOTO2001
IF[#1EQ#0]GOTO2002
IF[#11EQ#0]GOTO2003
IF[#3EQ#0]GOTO2004
IF[#9EQ30]GOTO2005
IF[#18EQ#0]GOTO2006
IF[#26EQ30]GOTO2007
#140=#5003
#27=#7/2
#100=0
#101=360/#11
#102=#11
G0X[#27*COS[#19-#1]]Y[#27*SIN[#19-#1]]
WHILE[#100LT#102]DO1
N1G0Z#18
G1Z#26F#3
G3X[#27*COS[[#100*#101]+#19]]Y[#27*SIN[[#100*#101]+#19]]R#27Z#5
G3X[#27*COS[[#100*#101]+[#19+#1]]]Y[#27*SIN[[#100*#101]+[#19+#1]] ]R#27Z#26
G0Z#140
#100=#100+1
G0X[#27*COS[[#100*#101]+[#19-#1]]]Y[#27*SIN[[#100*#101]+[#19-#1]] ]
END1
G0Z#5003
M99
N2001#3000=1(NO DIAMETER GIVEN)
N2002#3000=18(NO SLOT ANGLE GIVEN)
N2003#3000=6(NUMBER OF SLOTS NOT GIVEN)
N2004#3000=8(NO PLUNGE FEED GIVEN)
N2005#3000=3(NO FEEDRATE GIVEN)
N2006#3000=20(NO REFERENCE POINT)
N2007#3000=7(NO "Z" DEPTH GIVEN)
M30
```

ARRAY OF HOLES MACRO

```
O9015(ARRAY)
IF[#1EQ#0]GOTO2001
IF[#2EQ#0]GOTO2002
IF[#4EQ#0]GOTO2003
IF[#5EQ#0]GOTO2004
IF[#24EQ#0]GOTO2005
IF[#25EQ#0]GOTO2006
G0X#24Y#25
#140=#5003
G[#3*10000]Z#26R#18F#9P#7Q#17
#27=#1*#2(TOTAL HOLES)
#30=#1-1(HOLES IN Y-1)
#28=1
N1DO1
N2WHILE[#29LT#30]DO2
G91Y#5
#29=#29+1
#28=#28+1
END2
IF[#27-#28EQ0]GOTO4
IF[#27-#28LT0]GOTO4
#29=0
G91X#4Y#7
#28=#28+1
#31=0
N3WHILE[#31LT#30]DO3
Y-#5
#31=#31+1
#28=#28+1
END3
#28=#28+1
IF[#27-#28EQ0]GOTO4
IF[#27-#28LT0]GOTO4
X#4Y-#7
END1
N4G90G0Z#140
M99
N2001#3000=11(HOLE IN "Y" NOT GIVEN)
N2002#3000=12(HOLES IN "X" NOT GIVEN)
N2003#3000=13(NO DISTANCE IN "X")
N2004#3000=14(NO DISTANCE IN "Y")
N2005#3000=15(START IN "X" NO GIVEN)
N2006#3000=16(START IN "Y" NOT GIVEN)
(X,Y START POINT IN LOWER LEFT)
(A=NUMBER OF HOLES IN Y)
```

```
(B=NUMBER OF HOLES IN X)
(I=DISTANCE IN X)
(J=DISTANCE IN Y)
M30
```

COUNTERBORE MACRO

```
O9016(COUNTERBORING CYCLE)
N1IF[#7EQ#0]GOTO2001
N2IF[#9EQ#0]GOTO2002
N3IF[#18EQ#0]GOTO2003
IF[#26EQ#0]GOTO2004
N4#140=#5003(STORE INITIAL Z PT.)
N7#143=#4107+2000(H + 2000)
N8#144=#[#143](RADIUS OF END MILL)
N9#27=#7/2
N10#100=#27-#144(RAD OF CTRBORE - RAD OF ENDMILL)
#101=#100*2
G0Z#18(RAPID TO REF PT.)
G1Z#26F#9(FEED TO DEPTH)
G1Y-#100
G3G91X0.0 Y#101R#100
G3G91X0.0Y-#101R#100
G1 Y#100F20.
G0Z#140
M99
N2001#3000=1(NO DIAMETER GIVEN)
N2002#3000=3(NO FEEDRATE GIVEN)
N2003#3000=20(NO REFERENCE POINT)
N2004#3000=7(NO "Z" DEPTH GIVEN)
(Z=DEPTH OF COUNTERBORE)
(D=DIAMETER OF COUNTERBORE)
(R=REFERENCE POINT ABOVE PART)
(F=FEEDRATE)
M30
```

Line At Angle Macro

```
O9017(LINE-ANGLE)
IF[#11EQ#0]GOTO2001
IF[#9EQ#0]GOTO2002
IF[#3EQ30]GOTO2003
IF[#26EQ#0]GOTO2004
IF[#24EQ#0]GOTO2005
IF[#25EQ#0]GOTO2006
IF[#1EQ#0]GOTO2007
IF[#4EQ#0]GOTO2008
#100=1
G0X#24Y#25
G98G[#3*10000]Z#26R#18F#9Q#17P#20
WHILE[#100LT#11]DO1
N1G91X[COS[#1]*#4]Y[SIN[#1]*#4]
#100=#100+1
END1
M99
N2001#3000=2(NO H VALUE GIVEN)
N2002#3000=3(NO FEEDRATE GIVEN)
N2003#3000=4(NO DRILL CYCLE "C" GIVEN)
N2004#3000=7(NO Z DEPTH GIVEN)
N2005#3000=15(NO "X" START PT. GIVEN)
N2006#3000=16(NO "Y" START PT. GIVEN)
N2007#3000=10(NO ANGLE GIVEN)
N2008#3000=17(NO I VALUE GIVEN)
(I DISTANCE BETWEEN HOLES)
(A ANGLE FROM X AXIS CCW)
M30
```

SLOT MILLING MACRO

```
O9018(SLOT-MACRO)
IF[#11EQ#0]GOTO2001
IF[#7EQ#0]GOTO20002
IF[#2EQ#0]GOTO2003
IF[#18EQ#0]GOTO2004
IF[#9EQ#0]GOTO2005
IF[#3EQ#0]GOTO2006
#140=#5003
#31=1
#101=360/#11
G0X[COS[[#101*#31]+#1]*#7]Y[SIN[[#101*#31]+#1]*#7]
WHILE[#31LT#11]DO2
G0X[COS[[#101*#31]+#1]*#7]Y[SIN[[#101*#31]+#1]*#7]
G0Z#18
G1Z#26F#3
G1X[COS[[#101*#31]+#1]*#2]Y[SIN[[#101*#31]+#1]*#2]F#9
G0Z#140
#31=#31+1
END2
M99
N2001#3000=6(NUMBER OF SLOTS NOT GIVEN)
N2002#3000=22(NO START DIAMETER GIVEN)
M2003#3000=23(NO END DIAMETER GIVEN)
N2004#3000=20(N0 REFERENCE POINT GIVEN)
N2005#3000=3(NO FEEDRATE GIVEN)
N2006#3000=8(NO PLUNGE FEEDRATE GIVEN)
M30
```

Glossary of Terms

ASCII

American Standard Code for Information Interchange. A standard method of encoding alphanumeric characters into 7 or 8 bits.

BASIC language

An abbreviation for Beginners All Purpose Symbolic Instruction Code. One of the most popular languages for personal computers.

Baud

Unit of signaling speed. The speed in baud is the number of discrete conditions or events pre second. If each event represents only one bit condition, baud rate equals BPS.

Computer Aided Manufacturing

A software package that creates a CNC program with the aid of a computer. As the part complexity increases, so does the need for a CAM system.

Even Parity

A data verification method in which each character must have an even number of true bits.

Firmware

A computer program or software store in Read Only Memory.

FORTRAN

An abbreviation for FORmula TRANslation. Used for highly mathematical programs.

Input/Output

The process of transferring data from or to a computer system including communication channels, operator interface devices, or data acquisition and control channels.

Inspection Probes

Devices that record the position of the machine axis when contacting the part. Used to update offsets or record inspection data. Custom Macro is needed to use the probes.

Manual Data Input (MDI)

A method used to enter data into the control. MDI is used to set common variables and enter other data such as tool offsets. Most MDI operations can be eliminated through Custom Macro and the G10 (Load Parameter) function of the control.

Nonvolatile

A memory or data storage device that retains its information content when electrical power is removed. Ordinary RAM is volatile whereas ROM, bubble memory, battery backed CMOS RAM, floppy and hard disks are nonvolatile.

Odd Parity

A data verification method in which each character must have an odd number of true bits.

Parameter

A variable or an arbitrary constant appearing in a mathematical expression, each value of which restricts or determines the specific form of the expression. In the context of this manual, a setting that configures specific functions in the control.

Parity Check

The addition of a non-information bit that make up a transmission block to ensure that the total number of 1s is always either even (even parity) or odd (odd parity); used to detect transmission errors.

RAM

Random Access Memory. Semiconductor read/write volatile memory. Data stored is lost if power is removed.

RS232C

An Institute of Electrical and Electronics Engineers (IEEE) electrical standard for serial communications. This standard is used by the Fanuc control for communications to a Personal Computer.

SETVN

A command used to name the common variables #500-#519.

Vacant

If no value is defined for a local variable when the G65 call is made, the value for the variable is null, not zero. If a common variable has not been defined by your program it is also null. You can determine if a variable is null by comparing it to #0. For example, IF[#33EQ#0]GOTO100. See Custom Macro alarms for more information.

Index