Peter Smid

FANUC CNC Custom Macros

Programming resources for Fanuc Custom Macro B users

Library of Congress Cataloging-in-Publication Data

Smid, Peter.
Fanuc custom macros: programming resources for Fanuc Custom Macro B Users / Peter Simd.
p. cm
Includes index
ISBN 0-8311-3157-8
1. Machine-tools--Numerical control--Programming--Handbooks, manual, etc. 2. Macro instructions (Electronic computers)--Handbooks, manuals, etc. I. Title.

TJ1189.S5925 2004
621.9'023--dc22

2004056987

Industrial Press Inc.
200 Madison Avenue
New York, New York 10016

Cover Design: Janet Romano
Managing Editor: John Carleo

10 9 8 7 6 5 4 3 2 1

TABLE OF CONTENTS

1 FANUC MACROS

This handbook has been developed as a resource material for CNC programming at its highest level, using Fanuc and compatible Computer Numerical Control systems (CNC systems). Techniques described in the handbook are still part of the manual programming process, in the sense that no external CAD/CAM software or hardware is required. Although the main topic of this handbook is application of *Fanuc Custom Macros* in CNC programming (known as *Fanuc Custom Macro B*), several related topics have been added, mainly for coherence and comparison, but mainly as a refresher of some basic CNC programming skills required as a prerequisite.

The subject matters deal with several major topics, and the handbook is organized in the suggested order of learning. More experienced users can start at any section within the handbook:

- **General Introduction**
- **Review of G-codes and M-codes**
- **Review of subprograms**
- **System parameters**
- **Data setting**
- **Custom macros**
- **Probing applications**

Numerous examples and sample programs are used throughout the handbook. Their purpose is to serve not only as practical applications of the techniques explained, but - for many of them - as the basis for ready-to-run macro programs.

Although all the topics covered in the handbook are critical, they are discussed here for the single purpose of learning one subject, commonly known as *Custom Macros*, *User Macros*, *Fanuc Macros*, *Macro B*, or - just *Macros*. Several non-Fanuc controls also offer their version of macros, for example Fadal and Okuma, but only Fanuc macros are covered in this handbook.

General Introduction

This is the general introduction to the subject of macros. Its purpose is to make you aware of what macros are, what related subjects are important, and to identify several other helpful items to get you started in this important, exiting and often underestimated, field of CNC programming.

Knowledge of macros is becoming more and more essential, as companies large and small look towards more efficient ways of CNC program development, particularly for certain type of parts. Although CAD/CAM programming systems have become very popular and are on the rise, they do not and cannot always replace macro programming, for various reasons. Macros often serve as a special solution to special requirements.

The following brief descriptions provide some ideas of major subjects covered in the handbook.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

Typical Features

Typical features found in Fanuc macros are:

- Arithmetic and algebraic calculations
- Trigonometric calculations
- Variable data storage
- Logical operations
- Branching
- Looping
- Error detection
- Alarm generation
- Input and Output
- ... and many other features

A macro program resembles a standard CNC program to a certain extent, but includes many features not found in regular programming. Essentially, a macro program is structured as a regular subprogram. It is stored under its own program number (O-), and it is called by the main program or by another macro, using a G-code (typically G65). However, in a very simple form, macro features can be used in a single program as well, without the macro call command.

Main Program with Macro Features

Here is a simple example of a normal part program that cuts four slots (roughing cuts only):

```
N1 G21
N2 G17 G40 G80
N3 G90 G00 G54 X25.0 Y30.0 S1200 M03
N4 G43 Z2.0 H01 M08
N5 G01 Z-5.0 F100.0
N6 Y80.0 F200.0                    (SLOT 1)
N7 G00 Z2.0
N8 X36.0
N9 G01 Z-5.0 F100.0
N10 Y30.0 F200.0                   (SLOT 2)
N11 G00 Z2.0
N12 X47.0
N13 G01 Z-5.0 F100.0
N14 Y80.0 F200.0                   (SLOT 3)
N15 G00 Z2.0
N16 X58.0
N17 G01 Z-5.0 F100.0
N18 Y30.0 F200.0                   (SLOT 4)
N19 G00 Z2.0 M09
N20 G28 Z2.0 M05
N21 M30
%
```

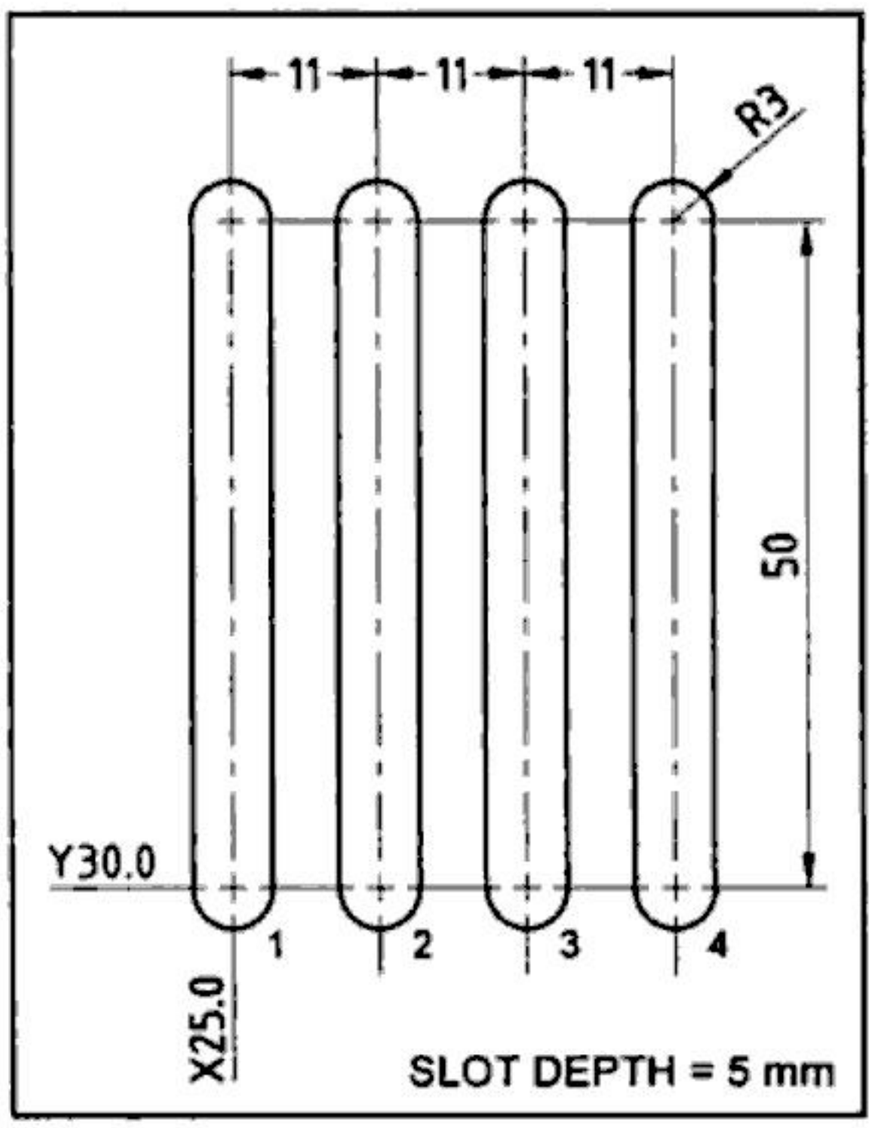


Figure 1

Simple job to illustrate feedrate as a macro function

Note the repetitive use of the two feedrates: F100.0 for plunging and F200.0 for slot cutting.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

Replacing Control Options

Fanuc controls offer many special features that are only available as an option. Typical optional features are *Scaling Function, Coordinate System Rotation, Polar Coordinates, Additional Offsets*, etc. With macros, you can develop a program performing exactly the same function what these options offer, without the extra cost, which often is quite high.

Hiding and Protecting Macro Programs

There will be macro programs that you create, than use them over and over again. After all, that is the main reason for their development in the first place. If something goes wrong with the macro in the control system, for example, an accidental deletion, its loss would cause a significant problem to the production process at that point. Macros can be *protected* within the control software, so they cannot be accidentally deleted or changed without forced additional steps. Macros with sensitive contents can also be *hidden* from the directory display.

Probing and Gauging

Probing and gauging are a very important areas of using custom macros. A section on probing, with examples, is also a significant part of this handbook. Using probes and similar devices, custom macros can be utilized as an 'on-machine inspection station', using a method commonly known as *in-process gauging*. Measured values (actual values) can be compared with the expected values (drawing values), and various offsets can be automatically adjusted.

Custom macros used in probing can be applied to different types of drawing specifications, such as corner locations, center locations, angles, diameters, depths, widths, automatic centering, boring measurements, and many others.

Various Shortcuts and Utilities

Many small utility programs can also be written into a macro form, to make the programming job (and the operator's job) easier and safer. Utilities are usually small programs that do not actually machine a part, but are used for certain common operations. Typical applications may include safe tool call, table or pallet indexing, management of tool life for unmanned operations, detection of worn out or broken tools, redefining the program zero (origin) for uneven castings, boring jaws on a lathe, counting the parts already machined, automating part-off operations on a CNC lathe, automatic tool changing, and many other possibilities. All these programs share a common feature - they are very effective shortcuts for repeatable activities that occur in CNC programming.

The field of macro applications is very extensive. In addition to the many possibilities already described, macros can be used to check spindle speeds, feedrates, and tool numbers, they can control the I/O (input and output) of data, check and register the active G-code command in a particular group, and disable the feedhold feature, spindle and feedrate overrides and a single block operation. The applications are virtually endless, and depend not only on your particular needs but your skills as well.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

Reference Tables

The following tables list the typical *preparatory commands* (G-codes) and *miscellaneous functions* (M-codes). Both milling and turning applications are included and the typical default preparatory commands are marked with the ♦ symbol (subject to change by the vendor or the user).

In case of discrepancy between the included tables and the CNC machine tool manual, always use the codes listed by the machine tool manufacturer

G-codes for Milling

The following table is a fairly comprehensive reference listing of all standard as well as the most common G-codes (preparatory commands) used for CNC milling programs (CNC milling machines and machining centers). All inter-dependent G-codes belong to the same group number and are modal, unless they belong to the Group 00, which identifies all non-modal commands:

G-code	Group	Description
G00	01	Rapid positioning mode
G01	01	Linear interpolation mode ♦
G02	01	Circular interpolation mode - clockwise direction
G03	01	Circular interpolation mode - counterclockwise direction
G04	00	Dwell function (programmed as a separate block)
G07	00	Hypothetical axis interpolation
G09	00	Exact stop check for one block
G10	00	Data setting mode (programmable data input)
G11	00	Data setting mode cancel
G15	17	Polar coordinate mode cancel ♦
G16	17	Polar coordinate mode
G17	02	*XY* plane designation ♦
G18	02	*ZX* plane designation
G19	02	*YZ* plane designation
G20	06	English units of input
G21	06	Metric units of input
G22	04	Stored stroke check *ON* ♦
G23	04	Stored stroke check *OFF*
G25	25	Spindle speed fluctuation detection *ON*
G26	25	Spindle speed fluctuation detection *OFF* ♦

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

G-code types cannot be mixed !

G-code Type			Group	Description
Type A	Type B	Type C		
G00	G00	G00	01	Rapid positioning mode
G01	G01	G01	01	Linear interpolation mode ♦
G02	G02	G02	01	Circular interpolation mode - clockwise direction
G03	G02	G03	01	Circular interpolation mode - counterclockwise direction
G04	G04	G04	00	Dwell function (programmed as a separate block)
G09	G09	G09	00	
G10	G10	G10	00	Data setting mode (programmable data input)
G11	G11	G11	00	Data setting mode cancel
G18	G18	G18	16	*ZX* plane designation ♦
G20	G20	G70	06	English units of input
G21	G21	G71	06	Metric units of input
G22	G22	G22	09	Stored stroke check *ON* ♦
G23	G23	G23	09	Stored stroke check *OFF*
G25	G25	G25	08	Spindle speed fluctuation detection *ON*
G26	G26	G26	08	Spindle speed fluctuation detection *OFF* ♦
G27	G27	G27	00	Machine zero return position check
G28	G28	G28	00	Machine zero return - primary reference point
G29	G29	G29	00	Return from machine zero
G30	G30	G30	00	Machine zero return - secondary reference point
G31	G31	G31	00	Skip function
G32	G33	G33	01	Threading function - constant lead thread
G34	G34	G34	01	Threading function - variable lead thread
G35	G35	G35	01	Circular threading CW
G36	G36	G36	01	Circular threading CCW *or:*
G36	G36	G36	00	Automatic tool compensation for the X-axis
G37	G37	G37	00	Automatic tool compensation for the Z-axis
G40	G40	G40	07	Tool nose radius compensation mode cancel ♦
G41	G41	G41	07	Tool nose radius compensation mode to the left
G42	G42	G42	07	Tool nose radius compensation mode to the right
G50	G92	G92	00	Coordinate system setting (tool position register) *and / or:*

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

3 REVIEW OF SUBPROGRAMS

To review the subject of subprograms, you have to understand first what a subprogram is, what it can be used for and what are its benefits. Comprehensive knowledge of subprograms is essential for macro program development.

In CNC programming, a subprogram is very similar in structure to a conventional program. What makes it different is its *content*. Typically, a subprogram is a separate program containing only unique repetitive tasks, such as a common contouring toolpath, a hole pattern or similar machining operations. For example, the task is to program a certain pattern of holes, in which the holes have to be spot-drilled, drilled and tapped. In standard part programming, the *XY* point coordinates for each hole will have to be calculated and repeated for each tool, using the appropriate fixed cycle. In a subprogram, the hole locations can be calculated only once, then stored in a separate program (subprogram) and retrieved many times, as needed, for different operations using different fixed cycles.

A subprogram is always called by another program (main program or another subprogram).

Subprogram must only contain data common to all parts or operations

Subprogram Example - Mill

To illustrate the concept of subprograms with a practical example, a very simple pattern of five holes is shown in the following illustration - *Figure 2*:

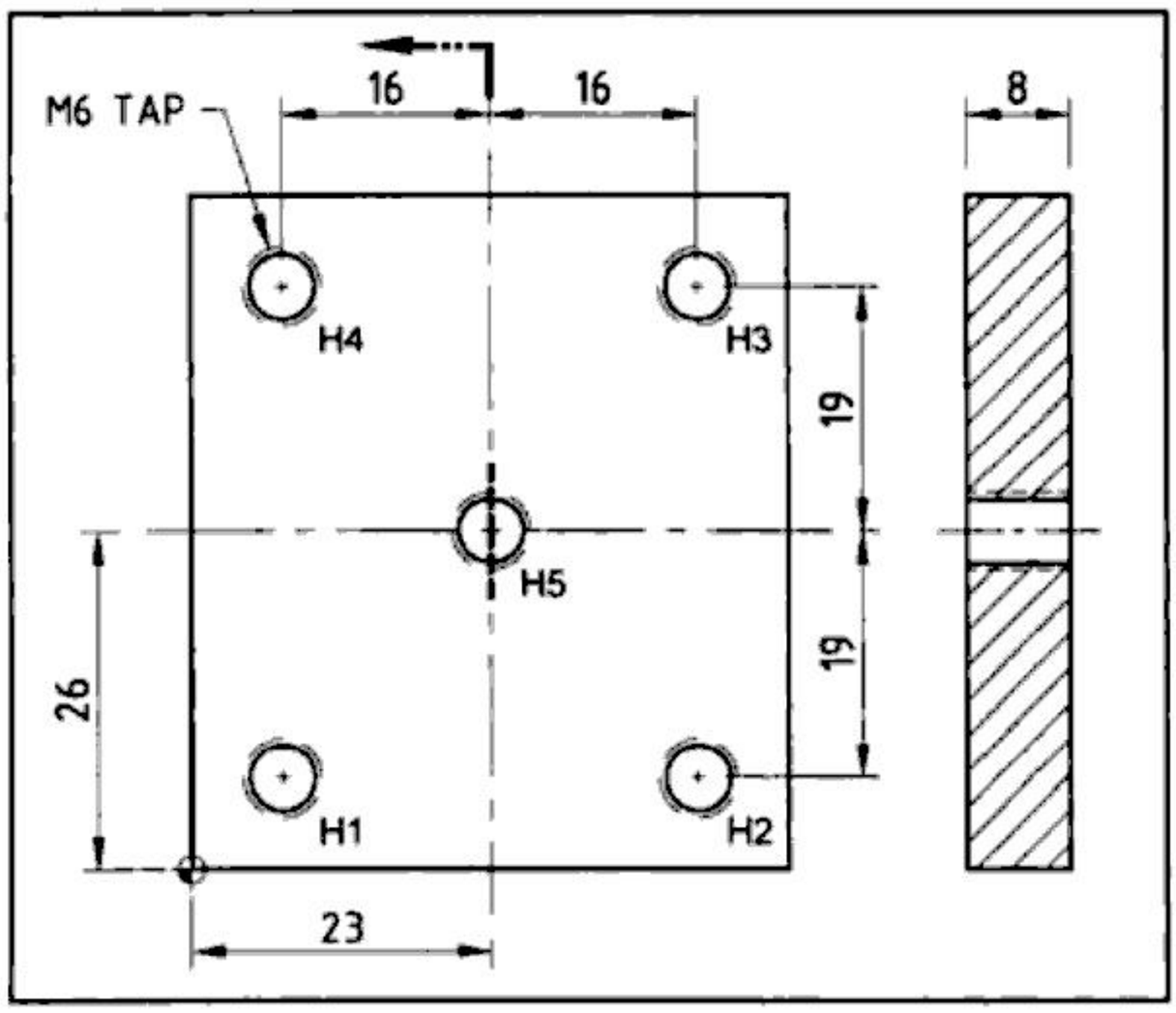


Figure 2

Sample drawing for subprogram example - mill application

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

```
O1002 (5 HOLE LOCATIONS SUBPROGRAM - VERSION 2)
N101 X7.0 Y7.0                                    (H1)
N102 X39.0                                        (H2)
N103 Y45.0                                        (H3)
N104 X7.0                                         (H4)
N105 X23.0 Y26.0                                  (H5)
N106 G80 G00 Z25.0 M09                            (CANCEL CYCLE AND CLEAR)
N107 G28 Z25.0 M05                                (Z-AXIS HOME RETURN)
N108 M99                                          (SUBPROGRAM END)
%
```

Do you like the program better than the previous version? In a strict technical definition, there is nothing wrong with the program - it will work well as is. Yet, there is a problem of different kind - the program uses a structure that many experienced programmers should and will avoid. Although the program itself is somewhat shorter, it is also much harder to interpret. Look at the reasons. When the subprogram is completed, the processing returns to the main program. Study the main program and you will see that it is impossible to tell whether the fixed cycle had been canceled or not. Also difficult is to see what other data may have been passed to the main program from the subprogram. You have look deep into the subprogram to find out these important details, which may be many printed pages away from the main program. In conclusion, while the shown *Example 3* is correct, it is definitely *not* recommended to be used, because of its poor structure.

Rules of Subprograms

From the last two examples for the five holes, you can see how a subprogram is defined, how it is ended and how it is called from another program. In a summary, there are two miscellaneous functions associated with subprograms:

M98	Subprogram call (followed by the subprogram number)
M99	Subprogram end

The **M98** function must always be followed by the subprogram number, for example,

```
M98 P1001
```

The subprogram must be stored in the control system under the assigned number, for example, as *O1001*. The miscellaneous function **M99** is usually programmed as a separate block - and also as the last block in the subprogram. This function will cause the transfer of the processing from the subprogram back to the program it *originated from*. That may be the main program or another subprogram.

The *end of record* symbol (the **%** sign) follows the **M99** function, the same way it follows the **M30** function in the main program. The **%** symbol represents a flag to stop transmission of the program, typically in DNC mode. When the processing returns to the program of origin, it will always be to the block immediately following the program call. For instance, look at the earlier *Example 2:*

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

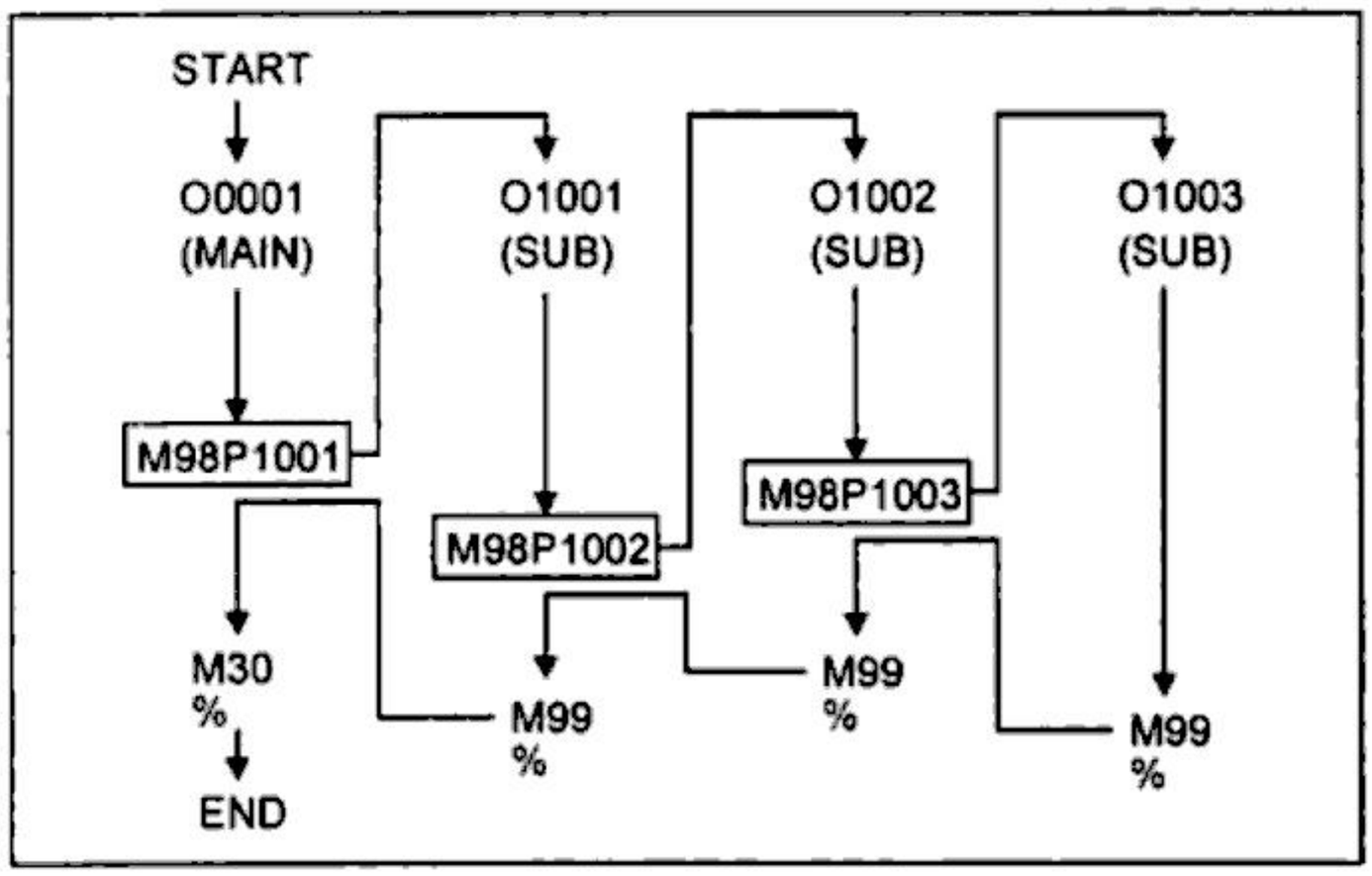


Figure 6

Three-level subprogram nesting

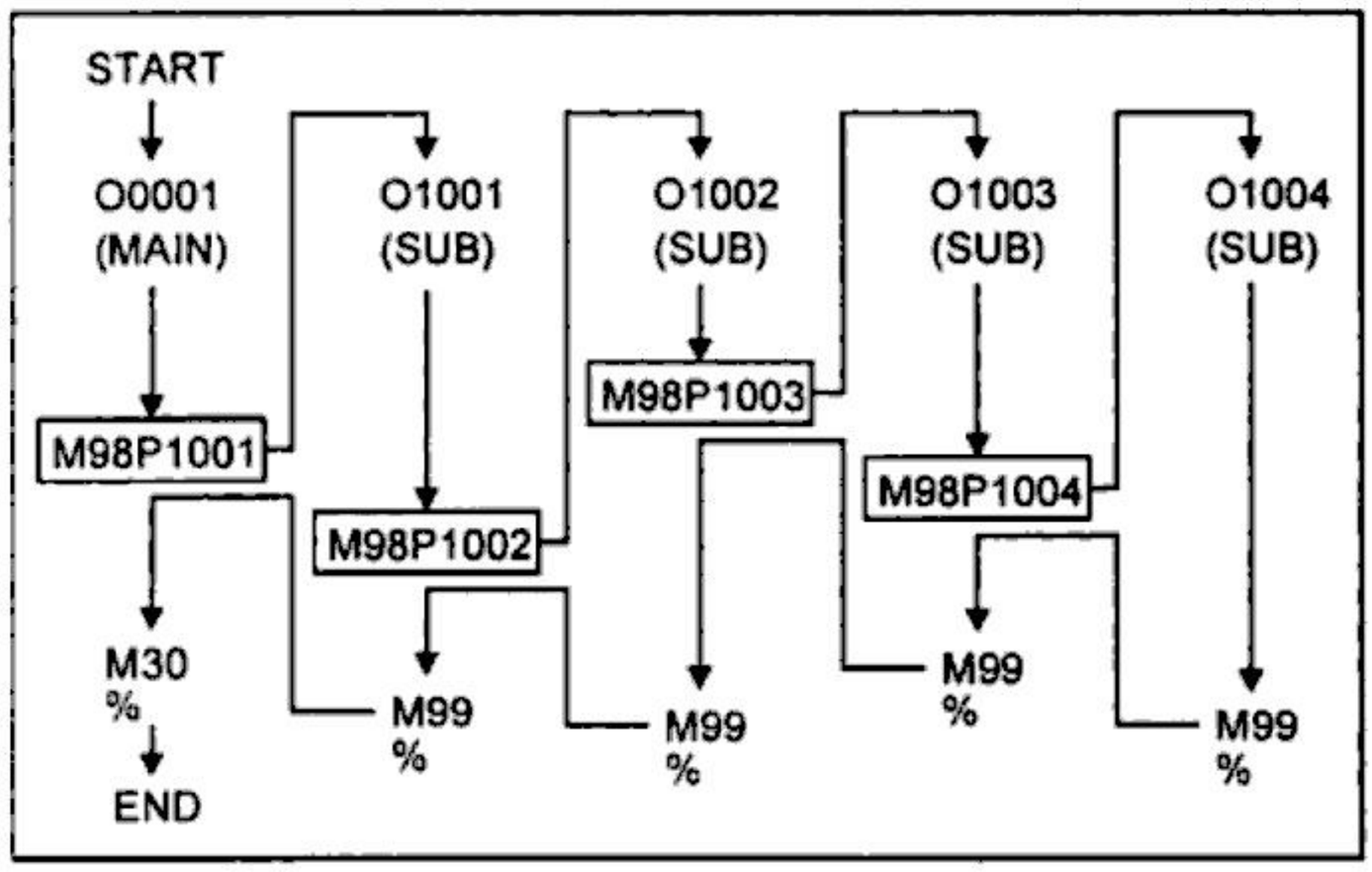


Figure 7

Four-level subprogram nesting

Subprogram Documentation

Any complex part program (subprograms and macros included) should always be well documented. Documenting CNC programs has been largely ignored by many users, often because of the perceived need to do a job fast. Although somewhat forgivable for simple and easy programs, the practice of *not* documenting programs is definitely not acceptable for subprograms and is also not acceptable for macros. Good program documentation is the key part of any CNC program development. Look at the schematic drawings of the four levels of subprogram nesting and you will see how complex the program can become with each increasing level of nesting. A good documentation will help the user in orientation and program 'decoding', therefore becomes a mandatory part of the programming process.

For both, the subprograms and macros, the program documentation should be internal as much as possible. This can be achieved by including important comments in the program body (main program, subprogram, or a macro). Program comments are typically enclosed in parentheses, for example, as **(DRILLING HOLE NUMBER 5)** . Provide only those comments that are relevant.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

4 SYSTEM PARAMETERS

The machine related information that establishes the connection between the CNC system and the machine tool, is stored as special data in internal control system registers, called the *system parameters,* or *control parameters*, or just *CNC parameters*. As an English word, its meaning is oriented towards mathematics, and is defined in a rather fancy sentence - *'A parameter is a quantity which may have various values, each fixed within the limits of a stated case'*. The sentence shows that the dictionary definition is right on for the purpose of defining parameters for a CNC system. Do not confuse parameters of the control system with the method of programming called *parametric programming* - except linguistically, they are *not* related. If you are a part programmer with limited experience, you should not be concerned about system parameters too much. Their original factory settings are generally quite sufficient for most work.

For specialized work like macro development, with all its related activities, such as probing and gauging, automatic offset changes, special methods of input and output, etc., a good in-depth knowledge of the system parameters is extremely important. There are hundreds of parameters available for any control system, and the majority of them you will never use.

Parameters are critical to the CNC machine operation - be careful when working with them !

What are Parameters ?

When the machine tool manufacturers design a CNC machine, they have to connect it with the CNC system, mostly designed by a *different* manufacturer. For example, a *Makino™* CNC machining center should be connected with a *Fanuc* CNC system - two independent manufacturers of two different products are involved in the process. The process of connection and configuration is often known as *interfacing*. The control system of Fanuc units has been designed with great internal flexibility and many parameters have to be set before the CNC machine tool is operational. Normally, it is the machine tool manufacturer (often called the *vendor*) who supplies the end user with all settings - *the interfacing* - a typical user seldom makes any changes. Even when a system parameter is changed by the program (standard or macro), the change is often only a temporary one, designed for a particular purpose. When the purpose is achieved, the program (or macro) normally resets the parameter to its original contents. Parameters are often changed intentionally, in order to optimize the performance of the machine tool.

The majority of control parameters relate to the specifics of a particular CNC machine tool, known to the machine tool manufacturer or vendor as various *defaults*. They include such items as all machine tool specifications, functions and characteristics that are in the exclusive domain of the manufacturer. Typical examples include rapid traverse rate, spindle speed ranges, length of axis motions, rapid or cutting feedrate ranges, clearances, various timers, data transfer baud rates, and many others. These parameters do not change and any attempt to make any changes severally endangers smooth machine operation.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

Parameter Display Screen

A brief look at a typical display of parameter screen on a CNC unit (using the *SYSTEM/PARAM* menus of the keyboard), many display screen pages will be available. The screen pages can be quickly scrolled through, forward or backward. A particular parameter can be called by its number (see *Parameter Manual*), in order to speed up the search. The screen cursor (display indicator) will be positioned at the parameter number and the parameter data will be displayed in reverse colors (highlighted). For the purpose of this handbook, the actual procedures of the keyboarding is not important - the control system manual describes all necessary steps.

Parameter Data Types

The classification of parameters in the brief section listed on the previous page has only shown the description of the parameters by *function*. In a control system, the actual parameter values are entered for each parameter number, as needed. Depending on the individual parameter application, the parameters are also classified by their *data type*. Each data type group uses a different range of valid parameter data entry.

On all Fanuc controls, there are four data type groups available. They are listed here with appropriate data ranges for each group:

Parameter Data Type	Available Data Range
Bit *or* Bit axis	0 *or* 1
Byte *or* Byte axis	0 *to* ±127 (byte) *or* 0 *to* 255 (byte axis)
Word *or* Word axis	0 *to* ±32767
Two-word *or* Two-word axis	0 *to* ±99999999

Bit-Type Data Type

Bits and *bytes* are common computer terms and should not be confused with each other. A *bit* is the smallest unit of a parameter input. Only *two* input values are allowed - the digit zero (0) and the digit one (1). The word *bit* is an abbreviation, derived from the full version of the words *binary digit*, as in *binary digit*. The English word *binary* has its origin in the Latin word *binarius*, meaning *something consisting of two parts*. Based on this definition, the two possible input values `0` and `1` represent a certain option, selected from no more than two conditions. Such a condition can be either *true* or *false*. *True* or *False* conditions can also be interpreted as *Yes* and *No*, *On* or *Off*, *Done* or *Not Done*, and so on. In the bit type entry, the selection represents one of only two possible alternatives.

A *byte* (described later) is a sequence of several adjacent bits (typically eight), that represents one alphanumerical character of data that is processed as a single unit of instruction.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

Fanuc controls allow the use of *any* increment of the block numbers, in *any* order, so the possibilities are quite extensive. From the point of view of the software designer, no high end software should contain *assumed* (or preset) values. Such an approach would limit the user and make the software weak as a result. Fanuc engineers respect that view and allow settings of the block number increment by individual users. In fact, it is not enough just to say *'Set automatic insertion of sequence numbers'* - we also have to say *'Set automatic insertion of sequence numbers with a particular increment'*.

Parameter **#0000** and bit **#5**, do not allow this setting at all. A *related parameter*, a parameter that is related to the **#0000** *and* its bit **#5** has to be used - and this parameter must contain the block increment amount. In the case of Fanuc 16/18/21 controls, the parameter is **#3216**, and is described in the Fanuc *Parameter Manual* as:

#3216

Increment in sequence numbers inserted automatically

In the above example, the block number increment can be set within the following limitations:

Data Type: **Word**
Valid Data Range: **0 *to* 9999**

The value of this parameter is the actual increment for the automatic block sequencing, within the range of 0 to 9999, defined as a *word type entry*. The word type entry will be discussed shortly. The selected amount will only be applied, if the bit **#5** of parameter **#0000** is set to 1, otherwise it will be ignored. This is a typical example of one parameter setting that is related to the setting of another system parameter. For different control systems (controls that claim Fanuc compatibility), the principle of setting will be the same, but the parameter numbers and the bit register numbers may be different. Always check the instructions for your control system.

Byte Data Type

The computer terms *'bit'* and *'byte'* have been already described somewhat briefly. These two words used in computing can be easily confused because the look similar. True, these words *are* similar, but they are *not* the same - they are unique words defined as *bit* and *byte*. Relating to the system parameters, the *bit type* parameter has already been described. The other type, the byte type system parameter accepts a range of values - from -127 to +127 for entries that require a signed value (plus and minus values), and the integer range from 0 to 255 for entries that do not require signed numbers. These ranges cover all eight bit entries, where each *byte digit* is the *bit*.

For example, many modern CNC machining centers are capable of the so called rigid tapping, rather than tapping using the floating tap tool holder. Most control systems will require a specific M-code for this function. This specific rigid tapping M-code is normally provided by the machine tool builder and must be interfaced to the control system - yes, you guessed it - *via a parameter setting*. The machine tool builder does all that. On Fanuc 16/18/21 controls for milling, the M-code for rigid tapping is specified by the parameter **#5210**:

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

Binary Numbers

In the preceding topics, the application of 0 and 1 entries was explained as a very common method of setting system parameter values. These are called the *bit settings* and are based on the system of *binary numbers*. Although many programmers have heard this term, not all understand its concept. In any CNC training program, binary numbers are not exactly the most appealing subject and are not generally covered. Strictly speaking, there is no need to know the binary numbers and how they work, but the knowledge does help in several special applications. Also, the subject of binary numbers may be interesting to those who would like to know more about it. The detailed description of binary numbers is beyond the scope of this handbook, but there are many excellent computer books describing the subject in detail. This section will only cover their main essence.

In everyday life, we use the decimal number system, which means we have ten digits available, from 0 to 9. The base of the decimal system is 10. In the decimal system, a number can be expanded, using the base of 10. For example, number 2763 can be represented as:

$$2 \times 10^3 + 7 \times 10^2 + 6 \times 10^1 + 3 \times 10^0 = 2763$$

The base of binary system is not 10 but 2. For the purposes of a macro beginner, the binary system uses only the digits *0* and *1* (one choice out of two). The prefix *bi-* means *'of two'*. Each symbol is known as a *bit* (***B***inary dig***IT***). Many CNC system parameters are of the binary type. In computing, various components can only have two states - *ON* or *OFF*, *Open* or *Closed*, *Active* or *Inactive*, and several others. These *ON-OFF* states are represented by the digits *0* and *1*, in the system parameters of the bit type. In one of the earlier examples, a typical setting of such a parameter was shown. For example, in the following parameter, the 8-bit value is

```
00001010
```

which can be represented as:

0	0	0	0	1	0	1	0
#7	#6	#5	#4	#3	#2	#1	#0
2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
128	64	32	16	8	4	2	1

Note how each bit in the above example is represented by its own number (#0 to #7), its own exponential representation and its bit value. The sum of the bits is simple to calculate:

$$0 \times 2^7 + 0 \times 2^6 + 0 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 = 10$$

In the chapter *'Automatic Operations'*, an actual application of the binary numbers is shown in detail, relating to the subject of *mirror image status check*.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

5 DATA SETTING

In smaller machine shops, job shops or any other environment where stand alone CNC machines are used, the machine operator typically sets all offset values that have to be input into the control during the job setup, manually by simply typing them in the proper registers. This common method is very useful when the programmer *does not know* the setting values - in fact, this is the normal situation in most shops that the programmer or machine operator does not know the *actual* values of various offsets at the time of programming or machine setup.

> Understanding the subject of offset data setting is important for many macro programs

Input of Offsets

In a manufacturing environment that has to be very tightly controlled during production, for example in agile and automatic manufacturing, or manufacturing of the same parts in very large volumes, the manual offset data entry during setup is very costly and inefficient. Also, this method does not provide and efficient means of adjusting the offsets values, for example, for tool wear. An agile or large volume manufacturing uses modern tools such as CAD/CAM systems for design and toolpath development, concept of multiple machine cells, robots, preset tools, automatic tool changing and tool life management, tool breakage detection, pallets, programmable auxiliary equipment, machine automation, and so on. Unknown elements cannot exist in such environment - relationships of all reference positions *must always be known*, and the need for offsets to be established and set at each individual machine is eliminated. All the initial offset values are - *and must be* - always known to the CNC programmer, well before the actual setup takes place on the CNC machine.

There is a great advantage in such information being known and used properly - the initial offset data can be included in the part program and be channeled into appropriate offset registers through the program flow. There is no operator's interference and the machining process is fully automated, including the maintenance of tools and related offsets. All offset settings are under the program control, *including their updates* required for position change, or a tool length change, or a radius change and other similar changes. The offsets are adjusted and updated from the information provided by an *in-process gauging system* that must be installed on the machine and interfaced with the control system.

All this automation is possible with several programming aids, often available as an optional feature of the control system. The main aid is the use of user macros and a feature called *Data Setting*. Before you can handle macros for the purpose of offset setting and adjustment, you have to understand the concept of the data setting. As a matter of fact, the control unit may have the data setting feature without even having the macro option. Don't get discouraged without trying it first. Even a small machine shop with a single stand alone CNC machine can benefit from this feature, if it is available. Fanuc control systems use a special preparatory command for data setting - **G10**.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

This program entry will place X-10.0 into the *external* work coordinate offset, while retaining all other settings (the Y-axis, the Z-axis, and any additional axes as well). As a result of this update, each work coordinate system used in the active CNC program will be shifted by 10 mm into the X-negative direction.

The other two offset groups, one for the tool length, the other for the cutter radius, can also be set by the **G10** command. However, there is an additional subject you have to fully understand, before using the **G10** command for those two offset groups. The subject covers the *memory types*.

Offset Memory Types - Milling

During the development of more advanced CNC technology, Fanuc controls have introduced three, progressively more advanced, types of memory to store tool length and tool radius offsets. The three stages are known as the *Memory Type A*, *Memory Type B*, and *Memory Type C*, referring to the *type of offset memory*.

They have these characteristics on the display screen of the control system:

Memory Type	Characteristics
Type A	One display column, shared by the length and radius offsets
Type B	Two display columns, one for the length offset and one for the radius offset
Type C	Four display columns, two for the length offset and two for the radius offset

In offsets are applied in macro programs, we deal with two important criteria - the size of the tool (length or radius) and the amount of wear on the tool (length change or a radius change). The terms *Geometry Offset* and *Wear Offset* are used frequently in this application, and may require some clarification.

Geometry Offset

For the *tool length*, the geometry offset stores the actual preset length of the tool, or the actual amount measured during setup. In a program, the length offset is called with the address H.

For the *tool radius*, the geometry offset stores the known (nominal) radius of the cutter (typically one half of the specified diameter). In a program, the radius offset is called with the address D but can also be called with the address H.

Wear Offset

As the name suggests, the wear offset is the *amount of deviation* from the measured value (*i.e.*, from the geometry offset). Do not take the name literally. Wear means tool wear, yes, but it also means an amount of deviation for *any other reason*, for example, a deviation due to resharpening of the tool or due to cutting pressures.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

Memory Type C

The latest (and the greatest) type of offset memory is the *Type C*. It is a much improved input method, based on *Type B* - it offers extreme programming control and flexibility during machine operation. *Type C* offset memory contains geometry and wear offset registers that are *independent* of each other - they are *different* for the tool length offset *and* the tool radius offset data. In normal practice, that means the total of *four* display columns on the control system screen - *Figure 12*.

OFFSET NUMBER	H		D	
	GEOMETRY	WEAR	GEOMETRY	WEAR
01	0.000	0.000	0.000	0.000
02	0.000	0.000	0.000	0.000
03	0.000	0.000	0.000	0.000
04	0.000	0.000	0.000	0.000
05	0.000	0.000	0.000	0.000
...	...	...	...	...

Figure 12

Offset memory *Type C*

Geometry and *Wear* offsets are separated
into two columns for each offset type.
Tool length and tool radius may use the same offset numbers.

➨ Example:

Tool T04 uses H04 for tool length and D04 for the tool radius offset (both H and D can be used):

```
G43 Z2.0 H04                    Offset number 04 used for tool length - H address
...
G01 G41 X123.0 D04 F275.0       Offset number 04 used for tool radius - D address
```

This is the most advanced method of CNC programming of length and radius offsets, because the control systems that support it - offer convenience and flexibility to both, the CNC programmer and the CNC machine tool operator. There is no need to *shift* one offset number by 25 or 50 numbers - they are both the same, both using *its own* offset register. There is no need to worry whether to use the H-address or the D-address - in *Type C* offset memory, the H-address will always be used for the tool length offset, the D-address will always be used for the radius offset - and can both have the *same* offset number.

Memory Type and Macros

It is very important to understand the offset memory types, because when writing a custom macro program that uses either the tool length or the tool radius offset (or both), the macro will have to reflect the differences of each offset type. That also means a particular macro will not be transferrable to a different machine control system, unless it also includes a multiple choice that covers the available offset memory types.

Offset memory type determines the macro structure for length and radius offsets

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

G10 Offset Data Settings - Milling Examples

This section illustrates some of the common examples of **G10** offset data settings used in a program (standard or macro) for CNC machining centers. Block numbers are used for convenience.

➧ Example 1:

Block N50 will input the amount of negative 468.0 mm into the tool length offset register 5:

```
N50 G90 G10 L10 P5 R-468.0
```

➧ Example 2:

If this offset needs an adjustment to cut *0.5 mm less depth*, using the tool length offset 5, the **G10** block will have to be changed to incremental mode:

```
N60 G91 G10 L10 P5 R0.5
```

Note the **G91** incremental mode - if the blocks N50 and N60 are used in the listed order, then the registered amount of offset number 5 will be -467.5 mm.

➧ Example 3:

For memory *Type C*, the cutter radius value D may be passed to the selected offset register from the CNC program, using the **G10** command with *L12 (geometry)* and *L13 (wear)* offset groups:

```
N70 G90 G10 L12 P7 R5.0      ... inputs 5.0 mm radius amount into the geometry offset register 7

N80 G90 G10 L13 P7 R-0.03    ... inputs -0.03 mm radius amount into the wear offset register 7
```

The combined effect of the two entries will be the equivalent of cutter radius 4.97 mm.

➧ Example 4:

To increase or decrease a stored offset amount, use the incremental programming mode **G91**. The example in block N80 will be updated, by adding 0.01 mm to the current wear offset amount:

```
N90 G91 G10 L13 P7 R0.01
```

The new setting in the *wear* offset register number 7 will be 0.02 mm and the combined effect of both offsets number 7 will be the equivalent of cutter radius 4.98 mm (after blocks N70, N80 and N90 were processed). Always be careful with the **G90** or **G91** modes - it is recommended to reinstate the proper mode immediately after use, for any subsequent sections of the program.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

Note that the tool tip number (programmed in the G10 application as the Q entry) will always change the geometry offset and the wear offset simultaneously, whatever the value or the offset type is. There is a simple and very logical reason for it - it is a control *built-in safety* that attempts to eliminate data entry error (manually or automatically). It is impossible to have a different geometry *and* wear tool tip numbers for the same physical tool. Data related to axes or the tool nose radius *may* have different geometry and wear offset values, because they relate to dimensions.

Data Setting Check in MDI

Programming offset values through the standard or macro program requires full understanding of the data input format for a particular control system. It is too late when an incorrect setting causes a damage to the machine or the part. One way to make sure the offset data setting is correct is *to test it*. Such a test is very easy to perform in the MDI (Manual Data Input) mode of the control system. An single word or the whole block can be entered in the MDI mode to test the input, before committing the data into the program. Select the *Program* mode and the MDI mode at the control unit, then insert the input data to check - for example:

```
G90 G10 L10 P12 R-106.475
```

- ❑ **Press INSERT**
- ❑ **Press CYCLE START**

To verify the accuracy of the input, check the tool length offset H12 - it should contain the stored amount of -106.475. Still in the MDI mode, *update* the preset data - for example:

```
G91 G10 L10 P12 R-1.0
```

- ❑ **Press INSERT**
- ❑ **Press CYCLE START**

To verify the accuracy of the input, check the tool length offset H12 again -it should contain the stored amount of -107.475.

Other tests should follow the same process. Always select the test data and offset numbers carefully, so they cannot cause any damage.

Programmable Parameter Entry

This section covers yet another aspect of programming the G10 data setting command - this time as a *modal* command. It is used to change a *system parameter* through the standard or macro program. This application is sometimes called the *'Write to parameter function'*, and is not very common in everyday programming, even in macros. Before you attempt to use this method, make sure you fully understand the concept of control system parameters section, described earlier in this handbook - see previous chapter for details.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

➨ Example 3 :

Another example of a system parameter change is for the entry of a *two-word* parameter type (long integer). It will change the work coordinate offset **G54** to X-250.000:

```
G90
G10 L50
N1221 P1 R-250000
G11
```

This is another method, one that differs from the one described earlier. Parameter **#1221** controls the **G54**, **#1222** controls the **G55**, and so on. P1 refers to the X-axis, P2 refers to the Y-axis, and so on, up to 8 axes. Because the valid range of a long integer (two-word type) is required, a decimal point cannot be used. Since the setting is in metric system, and one micron (0.001 mm) is the least increment, the value of -250.000 will be entered as -250000. Be careful with the input of zeros - one zero too many or one zero too few could cause a major problem. Speaking from experience, this type of error is not always easy to discover. The following version of the example is *NOT* correct, and *will result in an error*:

```
G90
G10 L50
N1221 P1 R-250.0          Decimal point is not allowed in the R-address
G11
```

Correct input is *without* the decimal point, as R-250000. An error condition (control alarm) will also be generated if the P-address is not specified. For example,

```
G90
G10 L50
N1221 R-250000
G11
```

will generate an error condition (alarm) - the parameter P is missing.

➨ Example 4 :

The last example is similar to the previous one, but modified for two axes values:

```
G90
G10 L50
N1221 P1 R-250000
N1221 P2 R-175000
G11
```

If this example is used on a lathe control, the address P1 is the X-axis, the address P2 is the Z-axis. On a machining center, the address P1 is the X-axis, the address P2 is the Y-axis, and the address P3 will be the Z-axis, if required. In either case, the first two axes of the **G54** setting will be -250.0 (X) and -175.0 (Y) respectively.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

6 MACRO STRUCTURE

Developing macro programs is not much different from development of standard CNC programs, at least not in the general approach. Before macro programs can developed, study carefully the many 'tools of the trade' and ask a question - what features do we work with? Macros have the potential of being extremely powerful and flexible. Macros can also shorten the programming time by many hours, literally. Yet, in spite of their great possibilities, macros are often the 'forgotten gems' available for CNC programming. Many companies do have macro capabilities, but avoiding them, considering them too difficult and time consuming.

Macro tools include many functions, techniques and procedures. Custom macro cannot be classified as a true programming language, but macros do share many elements with languages such as *Visual Basic™*, *C++™*, *Lisp™*, *and* many others, including the derivatives of the 'early' languages, such as *Pascal*. The most important tool for the start is to know the format of the macro, and its contents. When these two features are considered together, in the proper sequential order, we are talking about the *macro structure*.

Basic Tools

Every CNC programming technique that a typical part programmer has already learned can be - and are - used in macros and macro development. An in-depth knowledge of CNC programming, combined with a good practical experience (even machining helps), is an essential requirement to learning macros and learning them right from the beginning. Many programming aids not found in standard CNC programming are also available in macros, but they *enhance* and *extend* the traditional programming methods - *they do not replace them.*

There are three basic areas to understand for successful macro development:

- **Variables** ... ***three types of data***
- **Functions and Constants** ... ***mathematical calculations***
- **Logical Functions** ... ***loops and branches***

These three feature areas offer many powerful special functions that are used *within* the body of a macro, which is very similar to a body of a subprogram, except standard subprograms cannot use variable data, whereby macros can (and do so very extensively).

Just like a subprogram, a macro by itself is not much of a use - it has to be interwoven (interfaced) with another program, *called from another program*, by a previously assigned program number. The address (letter) O is used to store the macro programs, the address (letter) P is used to call it, applying the same logic as for subprograms.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

Arguments

The data defined with the macro call, that is with the **G65 P-** command, are called *arguments*. Arguments contain the actual program values required for a particular macro application only. They are always *passed to* the macro itself. Variable data in the macro are replaced with the supplied arguments and the toolpath or other activity is based on the current definitions (arguments) passed to the macro.

A typical program sample of a **G65** macro using three arguments will have the following schematic format:

```
G65 P- L- <ARGUMENTS>
```

☞ where ...

G65	Macro call command
P-	Program number containing the macro (stored as O----)
L-	Number of macro repetitions (L1 is assumed as a default)
ARGUMENTS	Definition of local variables to be passed to the macro

An actual sample program macro call may be defined as:

```
G65 P8003 H6 A30.0 F150.0
```

☞ where ...

G65	Macro call command	
P1234	Program number containing the macro	(stored as O8003)
H6	Assignment of local variable H (#11)	argument to be passed to the macro O8003
A30.0	Assignment of local variable A (#1)	argument to be passed to the macro O8003
F150.0	Assignment of local variable F (#9)	argument to be passed to the macro O8003

Assignments of variables is a separate subject covered in a separate chapter. An assignment simply means *giving the variable a value required at the time of call*. From the example, it is evident that custom macro call **G65** is only *similar to*, but definitely not the same as, the subprogram call **M98**. When two different calls (**M98** and **G65**) of a previously stored repetitive program are compared, there are several very important differences:

- **In the G65 command, argument is passed to the macro in the form of variable data. In M98 only the subprogram can be called. No data passing is possible**
- **In a subprogram call M98, the block may include another data (*i.e.*, a motion to a tool location). In this case, the processing can be stopped in a single block mode. This is not possible in the G65 mode**
- **In a subprogram call M98, the block may include another data (*i.e.*, a motion to a tool location). In this case, the processing of the macro starts only after the 'other data' is completed. The G65 command calls a macro unconditionally**
- **Local variables are not changed with M98 but they are changed with G65**

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

Parameters related to DISPLAY - O8000-O8999 program range				
Control System	Parameter	Bit	Bit ID	Setting
Fanuc 0	n/a	n/a	n/a	0 = n/a 1 = n/a (not available)
Fanuc 10/11/15	#0011	#1	ND8	0 = Program display during execution is ALLOWED
				1 = Program display during execution is NOT ALLOWED
Fanuc 16/18/21	#3202	#0	NE8	0 = Program editing and display is ALLOWED*
				1 = Program editing and display is NOT ALLOWED*

Program Numbers - Range O9000 to O9999

The second group is named *Group 2*. It covers the range of program numbers O9000 to O9999 only. Programs using numbers from Group 2 cannot be edited, registered, or deleted, without a parameter setting. Again, the parameter access number depends on the control system:

Parameters related to EDITING - O9000-O9999 program range				
Control System	Parameter	Bit	Bit ID	Setting
Fanuc 0	#0010	#4	PRG9	0 = Program editing is ALLOWED
				1 = Program editing is NOT ALLOWED
Fanuc 10/11/15	#2201	#0	NE9	0 = Program editing is ALLOWED
				1 = Program editing is NOT ALLOWED
Fanuc 16/18/21	#3202	#4	NE9	0 = Program editing and display is ALLOWED**
				1 = Program editing and display is NOT ALLOWED**

Parameters related to DISPLAY - O9000-O9999 program range				
Control System	Parameter	Bit	Bit ID	Setting
Fanuc 0	n/a	n/a	n/a	0 = n/a 1 = n/a (not available)
Fanuc 10/11/15	#2201	#1	ND9	0 = Program display during execution is ALLOWED
				1 = Program display during execution is NOT ALLOWED
Fanuc 16/18/21	#3202	#4	NE9	0 = Program editing and display is ALLOWED**
				1 = Program editing and display is NOT ALLOWED**

NOTE: *Display = Display during execution, * and ** identify the same settings for editing and display*

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

Quite likely, the only difference between the three programs will be the S-address for spindle speed in rev/min and the F-address for feedrate value in mm/min (in/min). With a macro, both addresses S and F can be defined as variables (because they will change for each of the three materials), then supply the suitable speed and feedrate values for different materials, as needed. By changing only those two values, the program can be used for many more different materials, not just three. The main programming benefit is that the body of the macro program does not change at all, once it is verified.

Variable Declaration

Before they can be used, variables have to be *defined* - macro expression refers to this activity as *declaration* of variables - variables have to be *declared*. Just like the data entry into the memory of a calculator, the basic rules governing the declaration of variables is that a variable *must be defined first*, and only then it can be used in a program or a macro. In the program that uses the variable, the form of definition is represented by the # symbol (commonly called the *pound sign* or the *sharp sign* or the *number sign*). This number sign will be used in all macros. The definition of a variable can take several forms, the first of them is the variable *value*:

```
#i = assigned current value
```

☞ ... where the letter 'i' represents the variable number - for example:

`#19 = 1200`	*Value of 1200 is assigned to variable number 19* *it can be spindle speed (rev/min)*
`#9 = 150.0`	*Value of 150.0 is assigned to variable number 9* *it can be feedrate (mm/min, m/min, ft/min, in/min, etc.)*

These two macro statements store values - the value of 1200 is stored into the variable **#19** and the value of 150.0 into the variable **#9**. Both values shown in the example are *numbers*, but they are two different *types* of a number.

Real Numbers and Integers

There are two basic types of numerical values used in macros - a number can be either:

- **REAL number** ... ***real*** **number always requires a decimal point**
- **INTEGER number** ... ***integer*** **numbers cannot use decimal point**

When performing mathematical calculations, the *type* of every numerical value is important. In simple terms, real numbers are typically used for calculations, whereby integer numbers are used for counting and other applications that do not require a decimal point. When a variable number is used in the macro program, its value can be changed as required at any time, two or more variables may be used for mathematical calculations, etc.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

Positive and Negative Variables

A variable definition that is not equal to zero is called a *non-zero* variable. Non-zero variables may be expressed as either positive or negative variable values. For example,

```
#24 = 13.7          ... this is a positive value variable definition
#25 = -5.2          ... this is a negative value variable definition
```

Why is this very simple and common fact so important? The reason is that in a macro, the call of the variable may *also* be positive or negative, which means tow signs are in effect. When the variable is referenced in the macro, the sign can be intentionally *reversed*, in order to achieve the *opposite effect* of the definition, for example:

```
G00 X-#24           ... will be equivalent to G00 X-13.7

G00 Y-#25           ... will be equivalent to G00 Y5.2

G00 X#24            ... will be equivalent to G00 X13.7

G00 Y#25            ... will be equivalent to G00 Y-5.2
```

The sign in the declaration is always used together with the sign in the actual execution - the same declaration used as above. Look at one of the above examples:

```
G00 Y-#25           ... will be equivalent to G00 Y5.2
```

The reason is strictly mathematical and relates to the use of a double sign in a calculation. In many instances, a negative number will have to be added or subtracted, and so on.

The following examples show all four possibilities:

Calculation	Result	Format	Example
Positive + Positive	Positive	a + (+b) = a + b	3 + (+5) = 3 + 5 = 8
Positive + Negative	Negative	a + (-b) = a - b	3 + (-5) = 3 - 5 = -2
Negative - Positive	Negative	a - (+b) = a - b	3 - (+5) = 3 - 5 = -2
Negative - Negative	Positive	a - (-b) = a + b	3 - (-5) = 3 + 5 = 8

This simplified method may be even easier to understand:

+ + = +	+ - = -	- + = -	- - = +

Note that the actual order of the plus and minus symbols within a calculation, such as +- or -+ makes no difference to the result. However, the standard mathematical hierarchy of calculating order is and must always be maintained.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

8 ASSIGNING VARIABLES

In the general introduction to variables earlier, four groups of variables were identified that are used in macro programs:

- **Local variables**
- **Common variables**
- **System variables**
- **Null variables (same as *empty* or *vacant* variables)**

It is very important to understand these variables well, particularly their differences. This chapter explains how to specify a value of a variable - how to *assign* a value to a variable. The first two of the groups listed - the *local* variables and the *common* variables are covered by this topic.

Local Variables

Local variables transfer the user supplied data to the macro body. Up to 33 variables can be defined as local. Naming this group of variables local means their stored values are only applicable to the macro they have been defined in, they are not transferable between macros. In macro programs, each local variable is associated with an assigned letter of the English alphabet. There are two options available for the so called assignment lists, *Assignment List 1*, which has 21 local variables available, and *Assignment List 2*, which has 33 local variables available. Both assignment lists are described here in detail.

Defining Variables

Variables that are defined in the **G65** macro call, can be within the range of **#1** to **#33**. They are called the *local variables*, or *arguments*. They are available only to the macro that calls them and processes them. Once the processing of the macro is completed, each local variable is reset to a null value, which means it becomes empty and has no value - it becomes *vacant*.

In practical terms, the local variables are used to pass data definitions from the source program (such as a main program) to a macro. Once transferred, they have served their purpose and are no longer needed. These variables were local to the program that called them. We use local variables to assign values to macro program arguments. Local variables are also used for a temporary storage within the macro body, during calculations of formulas and other expressions.

In addition to the **G65** command, there are also preparatory commands **G66**, **G66.1**, and **G67**, all related to macros. The **G65** command is most significant of them and is covered here in depth.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

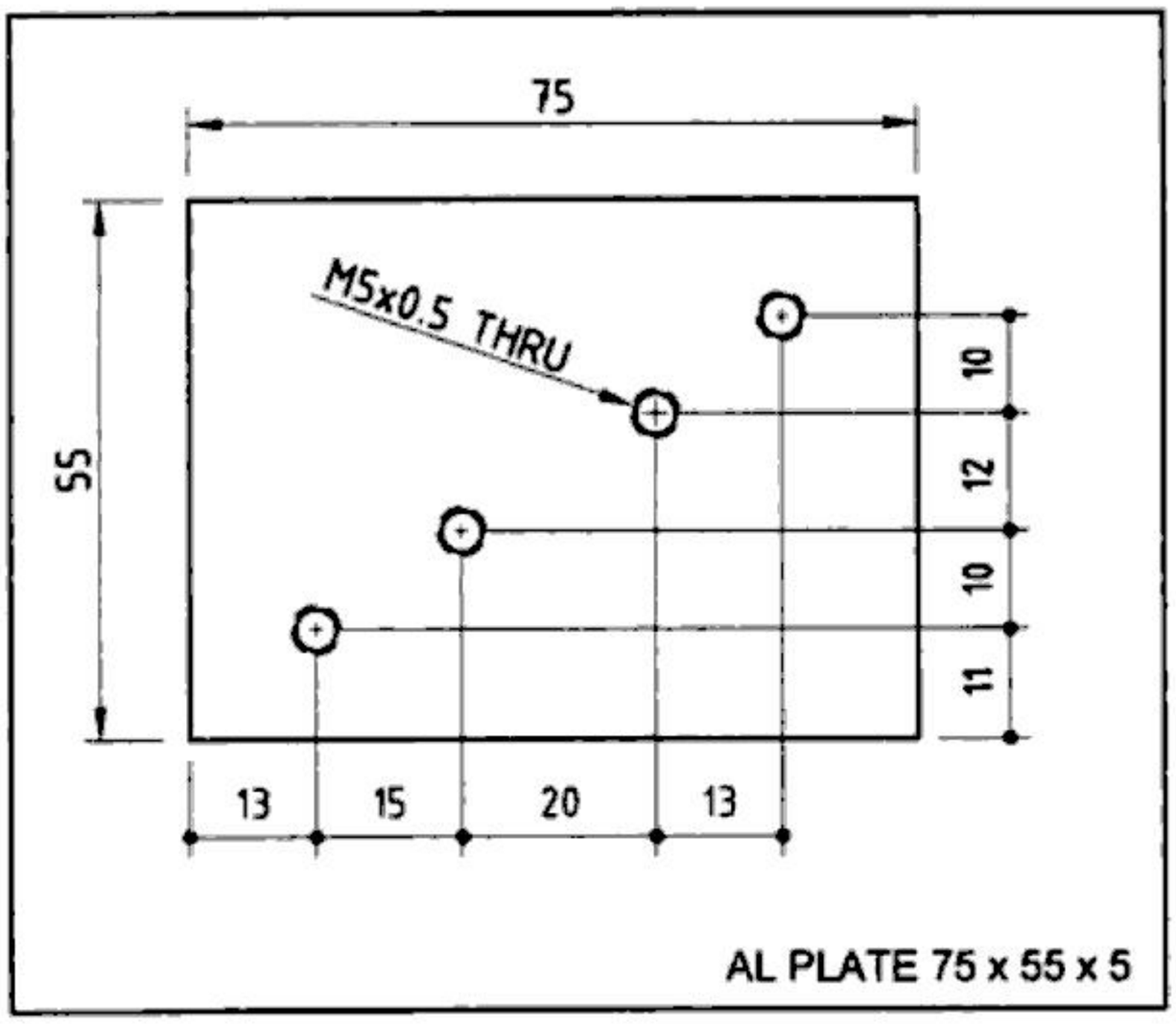


Figure 17

Drawing example for a modal macro call

X0Y0 is at the lower left corner,
Z0 is at the top of the 5mm plate

The simple example uses a part drawing in *Figure 17*, where four holes have to be tapped (drilling operation is omitted in the example). The macro will be designed for a special tapping operation only and **G84** tapping cycles *cannot* be used. This is also a good example of summing up the subjects covered so far.

The main objective of the macro is to program a lower feedrate when the tap moves into the material and a higher feedrate when the tap moves out. This tapping technique is useful for very fine threads in soft materials, to prevent thread stripping. These are the programming objectives:

- **Spindle speed** — **850 r/min**
- **Nominal feedrate** — **425 mm/min (850 r/min x 0.5 pitch)**
- **Feedrate in** — **80% of the nominal feedrate cutting in**
- **Feedrate out** — **120% of the nominal feedrate cutting out**
- **Retract clearance** — **3 mm**
- **Cutting depth** — **6.5 mm (1.5 mm below the bottom of part)**

Selection of Variables

Any assignment address can be used in the **G65** macro call, providing it meets the criteria of macros. Since letters will be used as assignments, the macro programmer has 21 of these letters to choose from. It makes sense to select letters that provide some relationship to their meaning in the macro. From the list above, selecting argument F for feedrate, S for spindle speed, Z for tapping depth, R for the initial and retract clearance, etc., makes it easier to fill in the assignments. This is only a teaching macro that does not have all the 'bells and whistles' incorporated into it. In this handbook, there are several version listed.

For the example at this stage (using modal macro call), only the following assignments will be provided - the clearance R-value as 3 mm (**#18**), Z-depth as -6.5 (**#26**), and the feedrate as 425.0 (**#9**). Development of the macro O8004 is quite simple:

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

To correct the accumulative error, or if you want to be absolutely certain, you have to round the motion in one direction to equal to the motion in the opposite direction:

```
N1 G20                                  English units input
N2 #100 = 3 + 19/64                     Input value of 3.296875   (motion A)
N3 #101 = 2 + 5/64                      Input value of 2.078125   (motion B)
N4 G91 G00 X-#100                       Incremental motion A to the left  X-3.2969
N5 G01 X-#101 F20.0                     Incremental motion B to the left  X-2.0781
N6 G00 X[ROUND[#100]+ROUND[#101]]       Incremental motion A+B to the right will be rounded
N7 M00                                  End of example
```

Rounding to a Fixed Number of Decimal Places

There are times when a fractional value has to be rounded to a *specific* (fixed) number of decimal places. Typically, *three* decimal places are required for the metric system, *four* decimal places are required for the English system, and perhaps *one* decimal place is required for cutting feedrate, regardless of the units selected.

In the following two examples, two given values will use a few techniques, providing the results of different rounding methods:

➨ Example 1 - Given fractional value is over 0.5 :

```
#1 = 1.638719                   Value to be rounded to a specific number of decimal places
```

If the **ROUND** function is applied to this defined value, it will return the *next whole* number:

```
ROUND[#1]                       Returns 2.0
```

In order to round the given value to a certain number of decimal places, the total of *three* steps will be necessary.

STEP 1 - The first step requires the given value to be *multiplied by the factor* of:

10	**... to round off to *one* decimal place**	
100	**... to round off to *two* decimal places**	
1000	**... to round off to *three* decimal places**	*typical for metric system*
10000	**... to round off to *four* decimal places**	*typical for English system*

... and so on

For example:

```
#2 = #1 * 1000                  Returns 1638.719    (Metric example)
#3 = #1 * 10000                 Returns 16387.19    (English example)
```

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

10 SYSTEM VARIABLES

The last group of variables is called the *System Variables*. The word *'system'* in the description of *System Variables* means the *Control System variables*. This group of variables is rather a special group and cannot be compared to the variable types already discussed (local and common). It is equally important in macros, but stands on its own.

In a macro program, this group is used to address the *registers* of the control memory (also called *addressable memory locations*). In certain situations (not normally), some system variables can also be used to change some *internal* data (also called *system* data) stored within the CNC system. For example, a work coordinate system (work offset) can be changed by manipulating the system variables (changing one or more system variables). In a similar way, items like the tool length compensation, macro alarms, parameter settings, parts count, modal values of the G-codes (plus several additional codes), and many others, can be changed as well. System variables are extremely important for automated environment, such as probing, unmanned and agile manufacturing, transfer systems, etc. There are many system variables available for each control system, and there significant differences between various control systems (even within the various Fanuc models available). It is very unlikely that any programmer will ever need them all. The control reference manual will come very handy as a reference resource.

Identifying System Variables

When working with system variables, there are two very important features to be aware of right from the beginning. Both relate to the way the system variables are identified by the control:

- **System variables are numbered from #1000 and up (four or five digit numbers)**
- **System variables are *not* displayed on the control display screen**

The numbering is fixed by Fanuc and cannot be changed. In this arbitrarily numbered system, a reference book or manual is required for each control model in the shop. Fanuc provides such a manual with the purchase of a particular control system. A great number of system variables are identified in this handbook as well.

Since the system variables cannot be directly displayed on the screen (applies to a large number of controls), there must be another way of finding what their current values are. The method used is called *a value transfer*. In the program, or in MDI mode, the value of a system variable can be *transferred* into a local or common variable. This chapter deals with this subject as well.

By organizing work, a tremendous step forward can be made. In the case of system variables, the first significant step to their better organization is by *grouping* them.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

#5201 to #5206 . . . . . Work offset value (shift or common) or up to #5215 [*]
#5221 to #5226 . . . . . Work offset value G54 or up to #5235 . . . [*]
#5241 to #5246 . . . . . Work offset value G55 or up to #5255 . . . [*]
#5261 to #5266 . . . . . Work offset value G56 or up to #5275 . . . [*]
#5281 to #5286 . . . . . Work offset value G57 or up to #5295 . . . [*]
#5301 to #5306 . . . . . Work offset value G58 or up to #5315 . . . [*]
#5321 to #5326 . . . . . Work offset value G59 or up to #5335 . . . [*]

[*] marks system variables of the *Read and Write* type

System Variables for Fanuc Series 16/18/21

#1000 through #1015. . . . . Data In DI Sending 16-bit signal from PMC to macro (reading bit by bit)

#1032 . . . . . Used for reading all 16-bits of a signal at one time

#1100 through #1115. . . . . Data Out DO Sending 16-bit signal from macro to PMC (writing bit by bit)

#1132 . . . . . Used for writing all 16-bits of a signal at one time to the PMC

#1133 . . . . . Used for writing all 32-bits of a signal at one time to the PMC - Values of -99999999 to +99999999 may be used for #1133

#2001 through #2200. . . . . Tool compensation values . (offsets 1-200) Memory Type A - Milling

#10001 through #10999. . . . Tool compensation values . (offsets 1-999) Memory Type A - Milling

#2001 through #2200. . . . . Wear offset values . . . . (offsets 1-200) Memory Type B - Milling

#2201 through #2400. . . . . Geometry offset values . . (offsets 1-200) Memory Type B - Milling

#10001 through #10999. . . . Wear offset values . . . . (offsets 1-999) Memory Type B - Milling

#11001 through #11999. . . . Geometry offset values . . (offsets 1-999) Memory Type B - Milling

#2001 through #2200. . . . . Wear offset values of H-code (offsets 1-200) Memory Type C - Milling

#2201 through #2400. . . . . Geometry offset values of H-code . (offsets 1-200) Memory Type C - Milling

#10001 through #10999. . . . Wear offset values of H-code (offsets 1-999) Memory Type C - Milling

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

Axis	External	G54	G55	G56	G57	G58	G59
1st = X	#5201	#5221	#5241	#5261	#5281	#5301	#5321
2nd = Y	#5202	#5222	#5242	#5262	#5282	#5302	#5322
3rd = Z	#5203	#5223	#5243	#5263	#5283	#5303	#5323
4th	#5204	#5224	#5244	#5264	#5284	#5304	#5324
5th	#5205	#5225	#5245	#5265	#5285	#5305	#5325
6th	#5206	#5226	#5246	#5266	#5286	#5306	#5326
7th	#5207	#5227	#5247	#5267	#5287	#5307	#5327
8th	#5208	#5228	#5248	#5268	#5288	#5308	#5328

No doubt, the table looks better organized than a plain list; it also is longer and does not contain any descriptions. It does not matter which representation is better, this methodical numbering system offers numerous benefits. It is not the cosmetics of the numbering system, it is a practically oriented numbering system that just happens to look appealing as well. This numbering system is suitable to use formulas in the macros, with variables, and allows calculation of the required address number based on the number of another address.

Take, for example, the following situation. If the calculations are based on the system variable `#5201`, all that is needed is a simple multiplication to get another coordinate system:

- Through the macro, add 20 times 1 to get the X-value for G54
- Through the macro, add 20 times 2 to get the X-value for G55
- Through the macro, add 20 times 3 to get the X-value for G56
- Through the macro, add 20 times 4 to get the X-value for G57
- Through the macro, add 20 times 5 to get the X-value for G58
- Through the macro, add 20 times 6 to get the X-value for G59

The value of *20* in this case is called the *shift* value. Of course, *any other* variable can be used as the base variable for the calculations. The logic of this approach can be used with many calculations, using the built-in numbering method. Going a little step further, think about how to handle the jump from one axis to another. Take it as a small challenge, but the next section will reveal the process and explanation.

Resetting Program Zero

At least a small but quite practical application example is in place here. Its purpose is to illustrate the application of system variables for a desirable, yet simple process. The system variable will be used in an actual macro program. Macro program development will be covered later, so a small preview now may be useful then. The project is quite simple, and the small macro program code can be very useful in everyday CNC machining. The macro example will only do one thing - it will reset the *current work offset setting to program zero at the current tool location*, using the topics discussed previously. This is known as *zero shift* or *datum shift*.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

Turning Control - FS-0T

Typical number of available tool offsets is 32, and the input of tool offset related system variables reflect that number.

Offset Registry	Tool Offset Number	Tool Wear Offset Value	Tool Geometry Offset Value
X-axis	1	#2001	#2701
	2	#2002	#2702
	3	#2003	#2703
	...	...	...
	32	#2032	#2732
Z-axis	1	#2101	#2801
	2	#2102	#2802
	3	#2103	#2803
	...	...	...
	32	#2132	#2832
Radius	1	#2201	#2901
	2	#2202	#2902
	3	#2203	#2903
	...	...	...
	32	#2232	#2932
Tool Tip	1	#2301	#2301
	2	#2302	#2302
	3	#2303	#2303
	...	...	...
	32	#2332	#2332

In the common *Type B* offset memory, four columns of system variables are required, 32 variables available for each column. Note that the *Geometry* and the *Wear* related variables are the same for the *Tool Tip* setting value, because they cannot be different for each mode. If a tool tip number 3, for example, is set to the *Geometry* offset, it will also appear as 3 in the *Wear* offset. Change of one will force the change of the other.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

Assignments for 64 Offsets or Less - Memory Type A

The offset memory *Type A* is not found very often in machine shops anymore. The following reference table lists system variables for 64 or fewer offsets, in memory *Type A*.

The listing is equivalent to the *Wear* offset listing only:

Offset Registry	Tool Offset Number	Tool Offset Value
X-axis	1	#2001
	2	#2002
	3	#2003
	. . .	. . .
	64	#2064
Z-axis	1	#2101
	2	#2102
	3	#2103
	. . .	. . .
	64	#2164
Radius	1	#2201
	2	#2202
	3	#2203
	. . .	. . .
	64	#2264
Tool Tip	1	#2301
	2	#2302
	3	#2303
	. . .	. . .
	64	#2364

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

Fanuc 10/11/15

Typical listing of G-codes (preparatory commands) modal information for the *higher* level CNC control systems:

System Variable Number		G-code Group	G-code Commands
Preceding Block	Executing Block		
#4001	#4201	01	G00 G01 G02 G03 G33 Note: G31 belongs to *Group 00*
#4002	#4202	02	G17 G18 G19
#4003	#4203	03	G90 G91
#4004	#4204	04	G22 G23
#4005	#4205	05	G93 G94 G95
#4006	#4206	06	G20 G21
#4007	#4207	07	G40 G41 G42
#4008	#4208	08	G43 G44 G45
#4009	#4209	09	G73 G74 G76 G80 G81 G82 G83 G84 G85 G86 G87 G88 G89
#4010	#4210	10	G98 G99
#4011	#4211	11	G50 G51
#4012	#4212	12	G65 G66 G67
#4013	#4213	13	G96 G97
#4014	#4214	14	G54 G55 G56 G57 G58 G59
#4015	#4215	15	G61 G62 G63 G64
#4016	#4216	16	G68 G69
#4017	#4217	17	G15 G16
#4018	#4218	18	G50.1 G51.1
#4019	#4219	19	G40.1 G41.1 G42.1
#4020	#4220	20	N/A to FS-M and FS-T controls
#4021	#4221	21	N/A
#4022	#4222	22	N/A
...	...	...	...

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

IF Function

IF

The **IF** function has several names - it is called the *decision* function, the *divergence* function, or most commonly, the *conditional* function. The format of th **IF** function is:

```
IF [ CONDITION IS TRUE ] GOTOn
```

where *n* is the block number *to branch to*, but *only if* the evaluated condition (the returned value) is *TRUE*. If the condition *is* true, all statements between the *IF-block* and *GOTOn-block* will be bypassed. If the evaluated condition is not *TRUE*, it is *FALSE*, and the program will continue processing the next block following the block containing the **IF** function. We can schematically represent the last example in a simple flow chart in *Figure 19*.

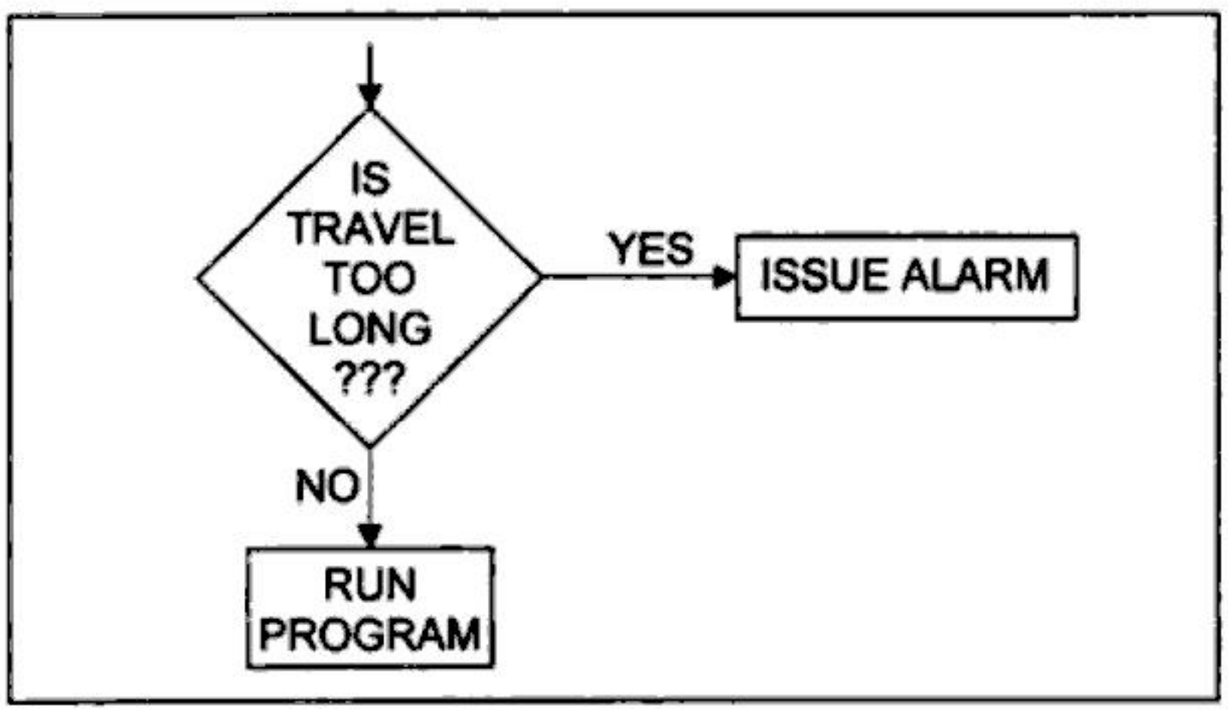


Figure19

Schematic flowchart representation of the **IF** conditional branching

The flowchart only shows the decision making and the results, *not* any complete program. The diagonal box identifies the condition to be evaluated (for example, machine travel limits), and the two rectangular boxes identify two - *and only two* - possible outcomes (YES or NO). Each outcome results in an action to be taken:

- **If the travel *IS* too long, generate alarm condition and stop processing**
- **If the travel *IS NOT* too long, run the rest of the program normally**

The **IF** function is one of the macro statements that control the order of program processing.

Conditional Branching

Branching from one block of the program to another block of the same program is unique to macros - it always means bypassing one or more program blocks. The bypass has to be done in a selective and controlled way, otherwise all kinds of problems will take over. The conditional function **IF** serves as a decision maker between at two options. The main statement in a macro is:

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

WHILE Loop Structure

In Fanuc type macro programming, the **WHILE** function is used to program loops. The format of the looping function **WHILE** consists of the function, condition and action:

```
WHILE [condition] DOn
```

In a plain language terms, think of the **WHILE** function as the *'as-long-as'* function. The looping function **WHILE [condition[DOn** in a Fanuc macro language means *'process the body of the loop as long as the specified condition is true'*. The **DOn** action establishes the connection with the end of the loop, where the **n** is replaced with a number of the matching **ENDn** statement. The loop is programmed with the **ENDn** function that corresponds to the **DOn** call, for example, **DO1** with **END1**, **DO2** with **END2**, and **DO3** with **END3**. Only *three* loop depths can be programmed.

The three allowed loop depths - often known in programming as the *levels of nesting* - have three similar forms:

- ***Single* level nesting**
- ***Double* level nesting**
- ***Triple* level nesting**

As the number of nesting levels increases, so does the programming complexity. The majority of loops for most macro applications are single level, double levels are not too unusual either. Triple level has a lot of power, but it does need a suitable application to employ it.

Single Level Nesting Loop

Programming only a single **WHILE** loop function between the **WHILE-DOn** loop call and its matching **ENDn**, defines the single level loop. This is the simplest and most commonly used looping function used in macro programs. The single level loop processes and controls one event at the time - *Figure 22*:

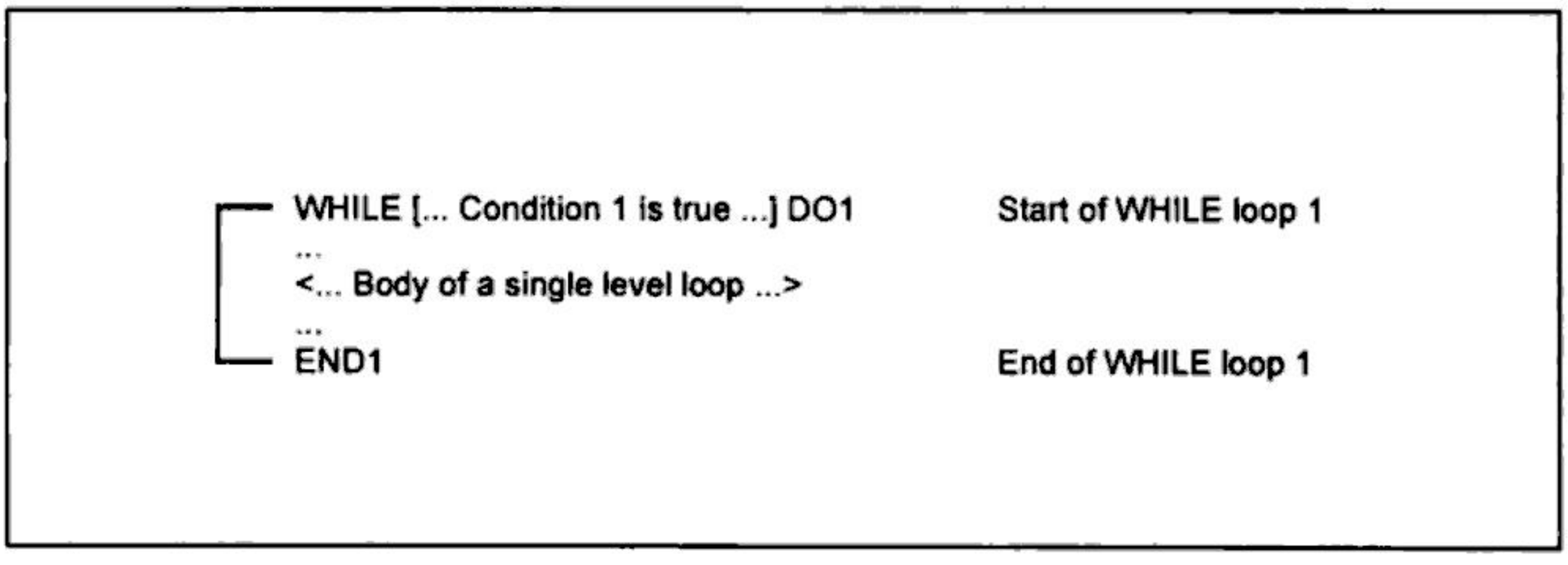


Figure 22

Single level of macro looping - controls one event at the time

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

Formula Based Macro - Sine Curve

When it comes to machining unique contours based on specific machining definitions (typically mathematical formulas), most CNC controls do not offer direct support. Developing a contour cutting program for a parabola, hyperbola, ellipse, sine curve, cycloid, and many other curves, may be not possible in standard CNC programs, but presents no problem in macros. This section illustrates the development of a sine curve as the actual cutting toolpath macro example. Since the control system does not directly support sine curve interpolation (or parabolic or hyperbolic interpolation, etc.), the toolpath will be simulated by many small linear motions, in `G01` mode.

Sine curve is one of several mathematical curves that may come handy in certain applications. Since it is based on a formula, it becomes a very fitting subject for macro development, the main reason for using this example. It is a simple formula that will be adapted to generate cutting tool motion. The *Figure 26* illustrates a typical sine curve along with the related terminology. The sine curve is, in effect, a flat representation of a full circle, from 0° to 360°. The distance between the start and end angles is called *Period*. The height of the curve is called *Amplitude* and is always the same above and below the X-axis:

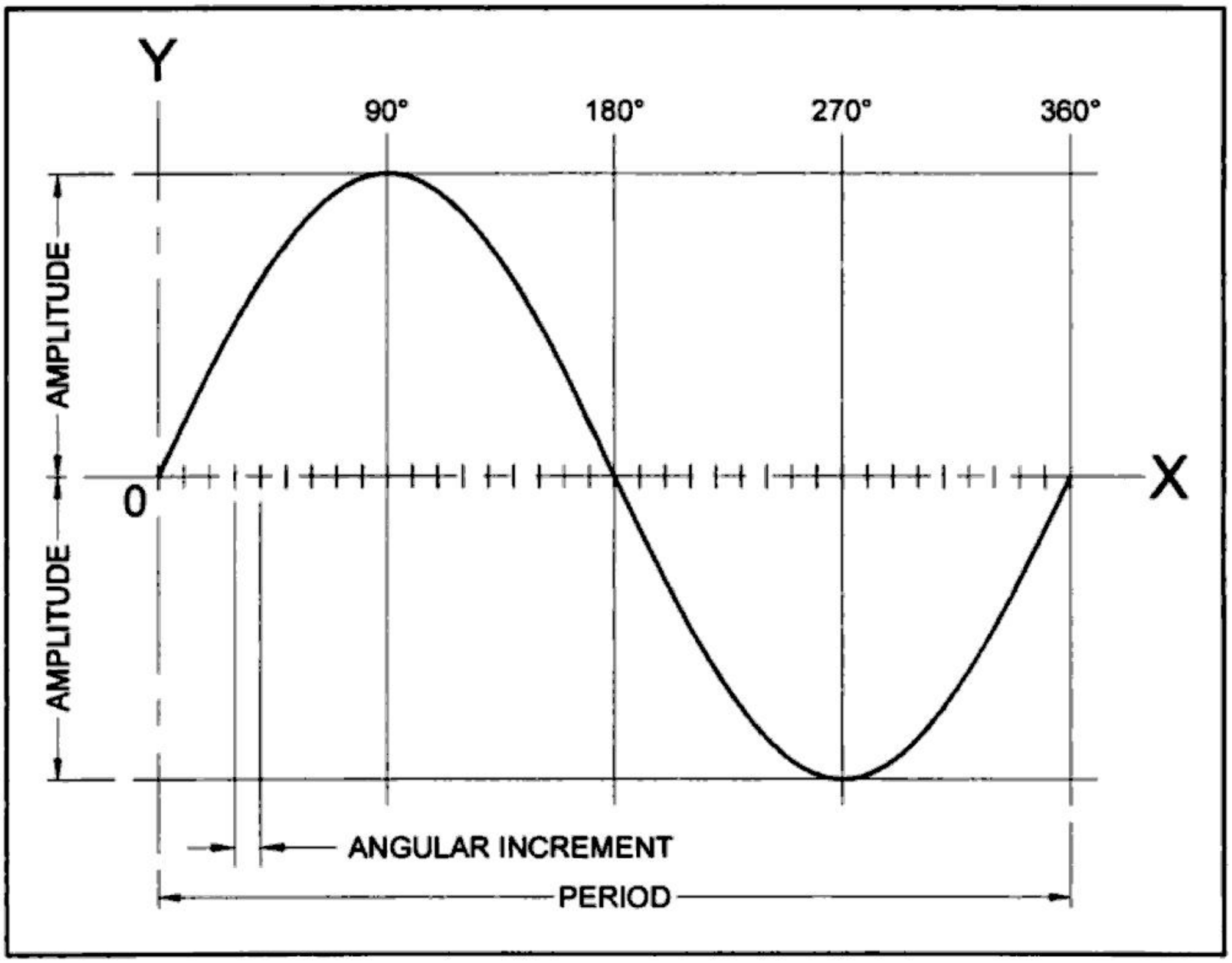


Figure 26

Sine curve - graphical layout

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

Alarm Format

The macro O8012 illustrates the actual application of a macro alarm that checks the input of an assigned variable (*i.e.*, argument R, assignment **#18**). Macro will check if the input radius is greater than 2.5 mm:

```
G65 P9000 R2.5                          Macro call with one argument (radius amount)

O8012                                   Macro start
...
IF[#18 GT 0.25] GOTO1001                Check condition for alarm - true or false ?
...                                     Process all blocks if condition is false
N1001 #3000 = 118 (RADIUS TOO LARGE)    Force alarm if condition is true
```

The selected alarm number and message to the operator is displayed on the screen as either

```
118 RADIUS TOO LARGE        or        3118 RADIUS TOO LARGE
```

Slight variations may be expected. This is a typical application of a programmed alarm - a controlled generation of an alarm by a macro, for a predictable possibility of an error.

Embedding Alarm in a Macro

Regardless of which alarm conditions are used in the macro, the transfer between the processed and the unprocessed portions of the program must be smooth, regardless of the returned value (*true* or *false*). For example, a macro may contain the following three alarms (to 'go-to'):

```
N1001 #3000 = 101 (HOLE SPACING IS TOO SMALL)
N1002 #3000 = 102 (TWO HOLES MINIMUM REQUIRED)
N1003 #3000 = 103 (DECIMAL POINT NOT ALLOWED)
```

In the macro O8013, these alarms will most likely be located towards the macro end. However, the macro program that *precedes* the alarms, using **G65 P8013 H8 I12.0 X75.0 Y100.0** macro call, will have to be processed without interruption, if the conditions are *false* (that means running good program, with no alarms). For example, this macro structure is *NOT* correct:

```
O8013                                   INCORRECT way to program alarms
IF[#4 LE 0] GOTO1001                    I=#4 variable stores the hole spacing
IF[#11 LT 2] GOTO1002                   H=#11 variable stores the number of holes
IF[#11 NE FUP[#11]] GOTO1003            Check if #11 contains the decimal point
G90 X#24 Y#25                           Previously defined tool location XY
   < ... macro body processing ... >
N1001 #3000 = 101 (HOLE SPACING IS TOO SMALL)
N1002 #3000 = 102 (TWO HOLES MINIMUM REQUIRED)
N1003 #3000 = 103 (DECIMAL POINT NOT ALLOWED)
M99
%
```

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

Evaluate the enclosed comments or try at the control to see how the timer works exactly:

```
O8017 (TIMING AN EVENT)
(PART --ONE-- USING #3001)
#3001 = 0                                  Reset to zero (start counting from zero)
G91 G01 X-100.0 F200.0                     Duration of this motion is 30 seconds
X100.0 F400.0                              Duration of this motion is 15 seconds
N999 (THIS MUST BE AN EMPTY BLOCK)         An empty block to prevent look-ahead !!!
#101 = #3001                               Returns calculation of 45632.000 (milliseconds)
#102 = #3001/1000                          Returns calculation of 00045.632 (seconds)
M00                                        Temporary stop to check variables

(PART --TWO-- USING #3002)
#103 = #3002                               Reset to zero (start counting from zero)
G91 G01 X-100.0 F200.0                     Duration of this motion is 30 seconds
X100.0 F400.0                              Duration of this motion is 15 seconds
N999 (THIS MUST BE AN EMPTY BLOCK)         An empty block to prevent look-ahead !!!
#104 = [#3002-#100]*3600                   Returns calculation of 45.631993 (seconds)
M00                                        Temporary stop to check variables
M30                                        End of program
%
```

Note the blocks N999 and the attached comment. Since the control is in the look-ahead mode, it calculates the final value prematurely. The empty block guarantees accurate calculated value.

Dwell as a Macro

Although the dwell function **G04** can be used much more efficiently in the majority of programs, the dwell may also be programmed with a macro, using the system variable **#3001**. For example, **G04 P5000** (a five second dwell) is equivalent to the following macro (and its call):

- **Macro call:**

```
...
G65 P8018 T5000                            T=#20 can be any other local variable (T is in ms)
...
```

- **Macro definition:**

```
O8018 (TIMER AS A DWELL)
#3001 = 0                                  Set system variable for timer to zero
WHILE[#3001 LE #20] DO1                    Loop until #3001 reaches the set delay
END1                                       Loop end
M99                                        Macro end
%
```

Note that even with the **WHILE** loop in effect, there is no need to program a counter, since the system variable **#3001** is always counting.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

Most CNC lathes may benefit from these functions a little more than machining centers.

Regardless of the application (milling or turning), an important reminder:

> **Be careful when activating the state of the M-S-T functions !**

Feedhold, Feedrate, and Exact Check Control

System variable **#3004** is similar to the **#3003**, but is used for automatic operation control of the *feedhold* switch, the *feedrate override* switch, and the *exact stop check* control. This variable can have up to eight settings, with the 0 (zero) setting as the default for all three functions, when the machine and control power is turned on. Zero setting means the function is active. Pressing the *RESET* button or *Power Off* will clear both system variables **#3004** and **#3003**.

System Variable #3004	Feedhold	Feedrate Override	Exact Stop Check
0	Enabled	Enabled	Enabled
1	Disabled	Enabled	Enabled
2	Enabled	Disabled	Enabled
3	Disabled	Disabled	Enabled
4	Enabled	Enabled	Disabled
5	Disabled	Enabled	Disabled
6	Enabled	Disabled	Disabled
7	Disabled	Disabled	Disabled

The #3004 **variable** controls three *states* of operation:

- ❑ **Feedhold**
- ❑ **Feedrate *override***
- ❑ **Exact *stop check mode***

Make sure to understand how these operations work before attempting to use them in macros.

◆ *Operation State 1 - FEEDHOLD*

When the ***feedhold*** is disabled in a macro, using the variable **#3004**, and the feedhold button on the control operation panel is pressed, the machine stops in the single block mode. If the system variable **#3003** (described earlier) has disabled the single block mode, there will be no single block mode operation available at all.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

Controlling the Number of Machined Parts

There are two more system variables that relate to auto mode operation. Two system variables, **#3901** and **#3902** control the counting of machined parts during an automatic operation.

They are:

#3901	*Number of parts completed (machined)*
#3902	*Number of parts required*

System variable **#3901** is used for the number of parts *machined*. The value of this variable stores the number of completed parts.

System variable **#3902** is used for the number of parts *required*. The value of this variable stores the number of required parts (the target number).

Both variables can be used to write to or to read from (read/write type). A negative value should not be used with these variables in macros (consider using the **ABS** function).

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

Benefits of Parametric Programming

Fast turnaround in production is the most significant benefit of family of parts macros. More time is often needed to develop a macro than a standard program, but this time is an excellent investment, especially if the macro will be used often. Knowing the benefits parametric programming offers, contributes to better decision when to develop a parametric program and when a standard program is more suitable. Parametric programming benefits in these improvements:

- **Overall benefits**
  - Quick turnaround between parts
  - Reduced time for program checking
  - Product quality improvement
  - Decrease of overall production costs

Individually, the benefits may be further identified in the *production* and *programming* areas:

- **Benefits in the production area:**
  - Reduction of scrap parts
  - Increased quality of the machined part
  - Tooling cost down due to standardized tooling
  - Increased productivity of the CNC machine
  - Lower maintenance costs

- **Benefits in the programming area:**
  - Drastic reduction is programming time
  - Programming errors reduced or eliminated
  - Consistency for all similar parts
  - Easier workload transition

In order to benefit from the parametric approach to programming, the first step is to identify suitable parts. Not every programming job is suitable for the additional investment in time.

When to Program Parametrically

The several areas already mentioned are also very important in determination whether the parametric programming will bring benefits or not:

- Large number of parts that are same in shape but different in dimensions
- Large number of parts that are similar in shape
- Parts that repeat fairly frequently
- Parts that contain repetitive tool path
- Various machining patterns

Parametric of programming is never a replacement for other methods - it only enhances them. There could be a significant investment in time spent on parametric macro program development. The resulting benefits must be tangible and measurable, in order to be economically efficient.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

Identify Variable Data

The purpose of finding the data values that change from part to part means finding *variable data*. Data that changes will help establish variables for the macro, either as a direct input or for further calculations. In the next listing, the same standard program is presented, this time with all variable data underlined:

```
(PIN-001 STANDARD PROGRAM)                    VARIABLE DATA IS UNDERLINED
(XOZO - CENTERLINE AND FRONT FINISHED FACE)
(BAR PROJECTION FROM CHUCK FACE = PART LG + 5 MM)
N1 G21 T0100                                  Metric units and Tool 1 - no wear offset
N2 G96 S100 M03                               CSS at 100 m/min - CW spindle rotation
N3 G00 X53.0 Z0 T0101 M08                     Start position for face cut + wear offset + coolant
N4 G01 X-1.8 F0.1                             Face just below centerline at 0.1 mm/rev feedrate
N5 G00 Z3.0                                   Clear-off face - Z-axis only - by 3 mm
N6 G42 X51.0                                  X-start for G71 cycle and tool radius offset
N7 G71 U2.5 R1.0                              G71 - 2.5 mm cutting depth, 1.0 retract
N8 G71 P9 Q14 U1.5 W0.125 F0.3                G71 - N9 to N14 contour - XZ stock - 0.3 mm/rev
N9 G00 X16.0                                  Calculated X-diameter for chamfer - '1'
N10 G01 X24.0 Z-1.0 F0.1                      Cut front chamfer at 0.1 mm/rev - '2'
N11 Z-23.0 R3.0 F0.15                         Cut small dia + inner radius at 0.15 mm/rev - '3'
N12 X46.0 R-2.0                               Cut face and outer radius - '4'
N13 Z-47.0                                    Cut large diameter 3 mm past part length - '5'
N14 X54.0 F0.3                                Clear-off stock diameter - X-axis only by 2 mm - '6'
N15 G70 P9 Q14 S125                           G70 finish contour at 125 m/min
N16 G00 G40 X100.0 Z50.0 T0100 M09            Rapid to tool change position + cancellations
N17 M01                                       Program stop (optionally - next tool expected)
```

This is a simple example - six program entries have been identified (underlined). Study them individually and very carefully - these values will become variables in the macro. Block by block evaluation yields some insight into the selected data:

```
N9 G00 X16.0                                  Calculated X-diameter for chamfer - '1'
```

Block N9 represents the first point of the contour *'1'* (P9 in the cycle block). It is the X-position for the chamfer cutting that follows. This diameter is not on the drawing, it has to be calculated, based on the chamfer size (1 mm at 45°) at the small diameter and the current Z-clearance (3 mm as per block N5). Working with a 45° chamfer is always easy and no trigonometry is required. The small diameter is 24 mm, chamfer is 1 mm, and the Z-clearance is 3 mm.

To calculate the corresponding X-diameter is easy - just watch the process carefully and make sure all values are calculated for a diameter, not per side (radius):

$$X = 24 - 2 \times 1 - 2 \times 3 = 16 \text{ mm} = X16.0$$

This calculation will be part of the macro, using other variables, still to be defined.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

Note the change in block numbers in the following programs:

```
(PIN-001 - MAIN PROGRAM)
(X0Z0 - CENTERLINE AND FRONT FINISHED FACE)
(BAR PROJECTION FROM CHUCK FACE = PART LG + 5 MM)
N1 G21 T0100                                  Metric units and Tool 1 - no wear offset
N2 G96 S100 M03                               CSS at 100 m/min - CW spindle rotation
N3 G00 X53.0 Z0 T0101 M08                     Start position for face cut + wear offset + coolant
N4 G01 X-1.8 F0.1                             Face just below centerline at 0.1 mm/rev feedrate
N5 G00 Z3.0                                   Clear-off face - Z-axis only - by 3 mm
N6 G42 X51.0                                  X-start for G71 cycle and tool radius offset
N7 G65 P8021 A23.0 B44.0 C24.0 D46.0 R3.0 (PIN-001 MACRO ARGUMENTS)
N8 G00 G40 X100.0 Z50.0 T0100 M09             Rapid to tool change position + cancellations
N9 M01                                        Program stop (optionally - next tool expected)
...                                           Other tool(-s) may follow

O8021 (PIN-XXX MACRO PROGRAM)                 Four pins in the family are covered by this macro
N101 G71 U2.5 R1.0                            G71 - 2.5 mm cutting depth, 1.0 retract
N102 G71 P103 Q108 U1.5 W0.125 F0.3           G71 - N103 to N108 contour - XZ stock - 0.3 mm/rev
N103 G00 X[#3-2*1-2*3]                        Calculated X-diameter for chamfer - 'A'
N104 G01 X#3 Z-1.0 F0.1                       Cut front chamfer at 0.1 mm/rev - 'B'
N105 Z-#1 R#18 F0.15                          Cut small dia + inner radius at 0.15 mm/rev - 'C'
N106 X#7 R-2.0                                Cut face and outer radius - 'D'
N107 Z-[#2+3.0]                               Cut large diameter 3 mm past part length - 'E'
N108 X54.0 F0.3                               Clear-off stock dia - X-axis only - by 2 mm - 'F'
N109 G70 P103 Q108 S125                       G70 finish contour at 125 m/min
N110 M99                                      End of macro
%
```

Any legitimate block numbers are allowed, as long as there are no duplicates in the same program (subprograms and macros included). Keep in mind that the P and Q addresses in the cycles indicate block numbers actually used in the contour program.

Final Version

The purpose of this presentation was to develop a basic macro program for a family of similar parts. Without a doubt, many additions to the macro can easily be made, depending on the exact nature of the existing job. Tolerances and surface finish may play a great role in the program development, as may some specific requests by the customer. These are all easy to implement. The main objective was to introduce a skilled CNC programmer into the world of macros.

One significant change that can be made - and it can also show how flexible macros are - is to make it easier to change machining from one part to another.

In the macro O8021, the only way to change the assignments of variables for different pins is in the block N7 - the **G65** block. This is quite a common method, but not the best method. Much better method is to include all four definitions into a single main program, and change just one variable number (at the program top) to select the required part (pin type). This objective is easy to achieve by including the four definitions along with the **IF** function in the main program:

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

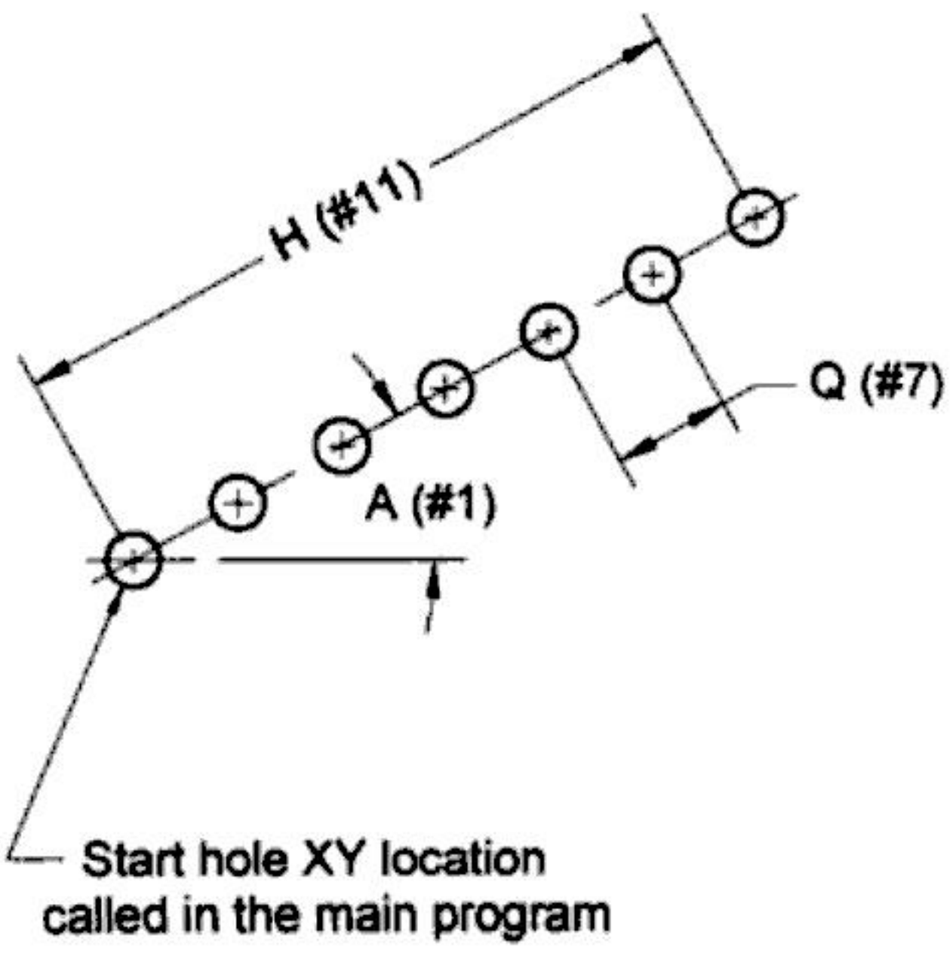
You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

When considered with similar drawings, this example provides all data needed to set up macro framework. It covers the conditions and restrictions, in which the macro application will be valid. For the educational purposes, this first application will be relatively simple (but definitely useful). Many programming techniques used in this macro will repeat in subsequent examples, with added features. First, evaluate the various self-imposed conditions and constrains towards the set goal:

❑ **All holes are spaced equally within the pattern**	*EQSP holes*
❑ **Any number of holes is acceptable - minimum of two holes**	*within machine capabilities*
❑ **The distance between the holes must be known**	*pitch of holes*
❑ **Any pitch between holes is acceptable**	*within machine capabilities*
❑ **The location of the first hole must be known**	*as XY coordinates*
❑ **Any angle between the first hole and the last hole must be known**	*establishes direction*

Once the conditions have been established and applied to an example, like the one in *Figure 31*, the most important first step has been completed. When evaluating a single drawing, always think of all other possibilities that may exist in similar drawings. For example, is the pattern of holes horizontal or vertical, is it rotated in the opposite direction, should the macro still be able to handle this pattern? Logically, there is no fundamental difference between one orientation and another. Always consider *all* features, including the angle, even if the angle is zero. Zero degree angle will define the horizontal orientation to the right of the first hole (east direction). A one-hundred-eighty degree angle defines the horizontal orientation to the *left* of the first hole (west direction). In the macro, the defined angle controls the orientation of the linear hole pattern. Based on all these considerations, including the self-imposed restrictions and other decisions, a common macro specific drawing will be necessary, applicable to *all* similar patterns - *Figure 32:*

MACRO O8101



Angular hole pattern macro - Version 1

VAR	#	Description
H	11	Number of EQSP holes
Q	17	Spacing between holes
A	1	Angular direction of cutting

G65 P8101 H- Q- A-

Figure 32

Variable data for angular pattern of holes - Version 1 - Macro O8101

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

Frame Hole Pattern

Frame hole pattern is quite common in many machine shops, and consists of a series of equally spaced holes, forming a rectangular pattern. In effect, this pattern consists of four sets of holes in an angular arrangement, so the first macro (O8101) can be used - four times. However, a frame pattern is more efficient and - if developed correctly - prevents double cutting of the corner holes, which can easily happen using other methods. The macro to develop is a macro that defines such a pattern of holes, starting at the lower left hole of the rectangle, then continuing around the frame in the CW or CCW direction. Again, certain decisions, conditions and restrictions have to be imposed first, based on the type of work.

Figure 35 represents a typical drawing for a frame hole pattern.

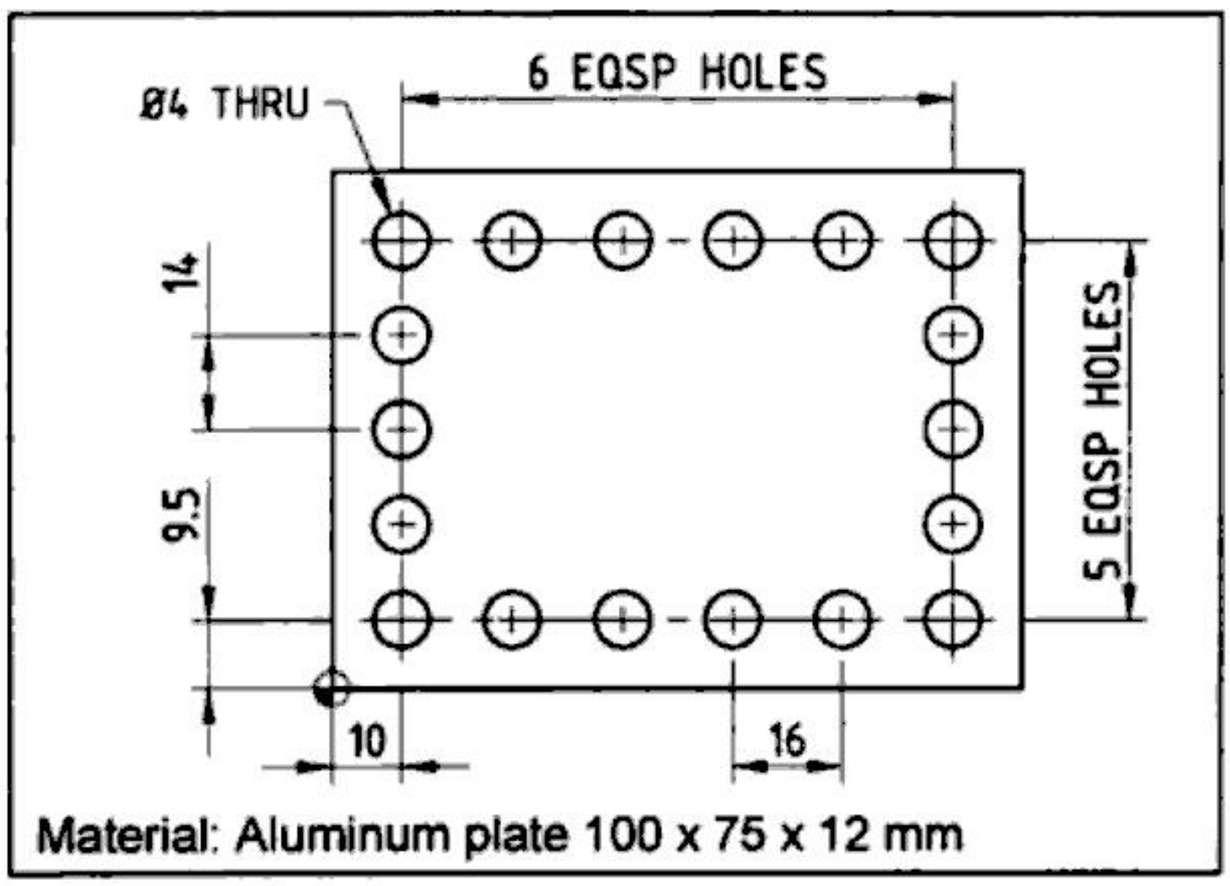


Figure 35

Frame hole pattern - definitions

Based on the typical pattern shown, study carefully what features have been considered. Note that the equal spacing between holes is (or could be) *different* for the X-axis and the Y-axis. This reality has to be taken into consideration. Based on the example drawing, as well as on the result of the necessary thinking process, the following features have been employed in the macro presented here:

❑ **All holes are spaced equally within the pattern**	*pitch in X can be different from the pitch in Y*
❑ **Any number of holes is acceptable**	*machine permitting - 2 min per row or column*
❑ **Distance between holes must be known**	*pitch along X and Y (both positive)*
❑ **Any pitch between holes is acceptable**	*within machine capabilities*
❑ **Location of the first hole must be known**	*XY coordinates*
❑ **First hole is the lower left corner of the pattern**	*must be known*
❑ **Machining direction is CCW**	*X+ Y+ X- Y-*

In the macro, the key element will be to prevent cutting any corner hole twice. That can be achieved by programming L0 or K0 in the fixed cycle called. From these selected conditions, the assignments in the **G65** macro call block can now be defined.

Figure 36 shows the visual definition of all required variables.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

Although the concept of this macro may need some effort to understand it thoroughly, a few points may help. First, note the input of the BCD - *the Bolt Circle Diameter* - internally, in the macro, the diameter is irrelevant - it is the *radius* that is needed within the macro for calculations. However, it is the diameter value that is the normal way of dimensioning the drawing, so it should be the *diameter value* that is input into the **G65** variable definitions (variable R in the above figure is actually not used - instead, the variable W is redefined, to save memory resources of the control system. Another variable that is necessary for calculations (but not defined as an assignment) is identified as B in the illustration only, and nested internally in the macro.

For final reference, finished top of the part is program zero for the Z-axis (Z0). X0Y0 is located at the lower left corner (but could be anywhere). The following main program reflects the bolt circle drawing illustrated in *Figure 37:*

```
O0024 (MAIN PROGRAM)
N1 G21                                          Metric mode
N2 G90 G00 G54 X0 Y0 S1200 M03                  First motion block + spindle speed
N3 G43 Z10.0 H01 M08                            Tool length offset + clearance above
N4 G99 G82 R1.0 Z-15.9 P300 F225.0 L0           Fixed cycle call data - no machining (or K0)
N5 G65 P8104 X50.0 Y37.5 W49.0 H6 A1.0 S1       Macro call with assignments - full circle
N6 G80 Z10.0 M09                                Retract above work
N7 G28 Z10.0 M05                                Return to machine zero
N8 M01                                          End of current tool
...
%
```

```
O8104 (BOLT HOLE CIRCLE MACRO)                  Macro number and description
#10 = #4003                                     Store current setting of G90 or G91
IF[#23 LE 0] GOTO9101                           Bolt circle diameter to be greater than zero
IF[#11 NE FUP[#11]] GOTO9102                    No fractions allowed for number of holes
IF[#11 LE 0] GOTO9103                           Minimum number of holes is one
IF[#19 EQ #0] THEN #19 = 1                      Start hole number = 1 (one) if not specified
IF[#19 NE FUP[#19]] GOTO9102                    No fractions allowed for start hole number
IF[#19 LT 1] GOTO9104                           Start hole number must be one or higher
IF[#19 GT #11] GOTO9105                         Start hole number must be less than all holes
#23 = #23/2                                     Change diameter of bolt circle to radius
WHILE[#19 LE #11] DO1                           Start loop for holes
#30 = [#19-1]*360/#11+#1                        Calculate current hole angle
X[COS[#30]*#23+#24] Y[SIN[#30]*#23+#25]         Calculate current X and Y hole location
#19 = #19+1                                     Update counter for the loop
END1                                            End of loop
GOTO9999                                        Bypass alarm messages
N9101 #3000=101 (DIA MUST BE GT 0)                  Alarm number 101 or 3101
N9102 #3000=102 (HOLES DATA MUST BE INTEGER)        Alarm number 102 or 3102
N9103 #3000=103 (ONLY POSITIVE NUM OF HOLES)        Alarm number 103 or 3103
N9104 #3000=104 (START HOLE MUST BE INTEGER)        Alarm number 104 or 3104
N9105 #3000=105 (START HOLE NUMBER TOO HIGH)        Alarm number 105 or 3105
N9999 G#10                                      Restore modal G-code
M99                                             End of macro
%
```

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

Variable Data for Circular Pocket Roughing

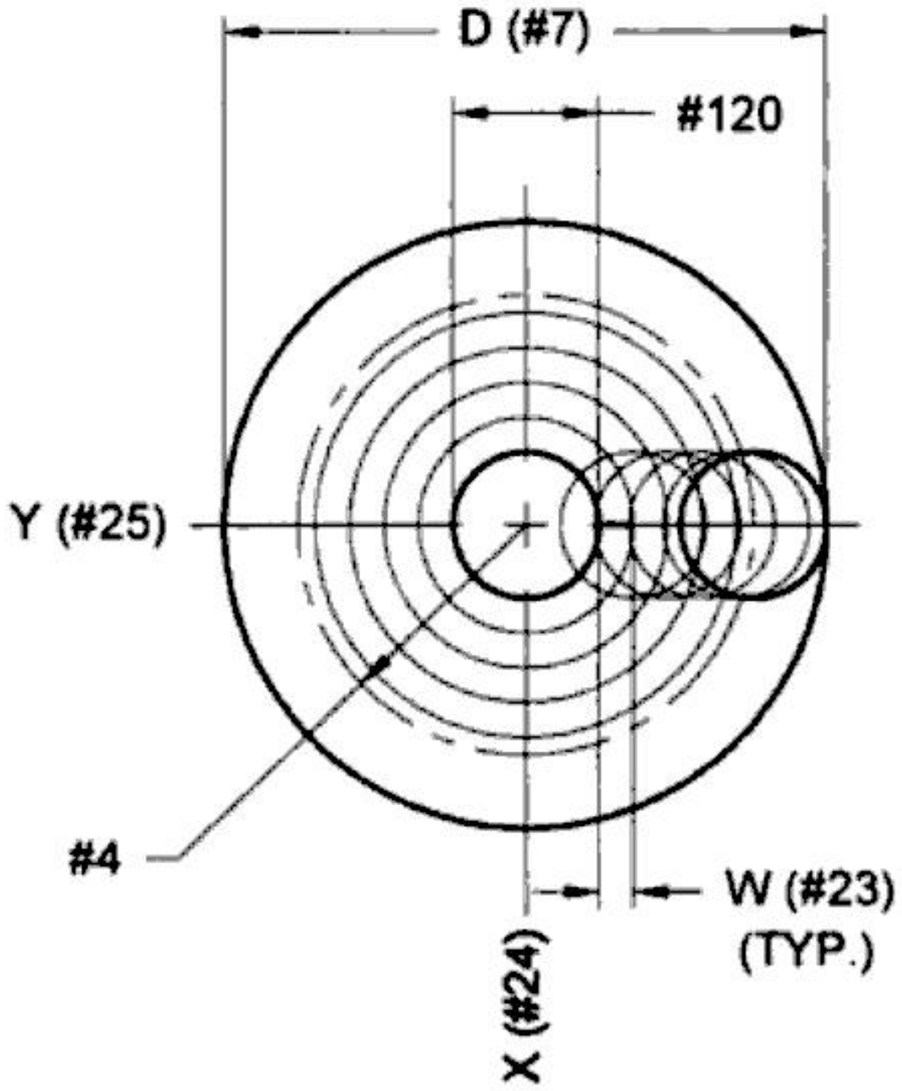
Circular pocket roughing includes many settings that change from job to job. Good planning is important and when completed, the following features will be defined to develop a macro toolpath for a circular pocket roughing:

- ❑ **Pocket center as absolute X-location** ... *assigned letter X (variable #24)*
- ❑ **Pocket center as absolute Y-location** ... *assigned letter Y (variable #25)*
- ❑ **Pocket final depth** ... *assigned letter Z (variable #26)*
- ❑ **Pocket final diameter** ... *assigned letter D (variable #7)*
- ❑ **Depth of each cut** ... *assigned letter K (variable #6)*
- ❑ **Width of each cut** ... *assigned letter W (variable #23)*
- ❑ **Tool offset number** ... *assigned letter T (variable #20)*
- ❑ **Cutting feedrate** ... *assigned letter F (variable #9)*

If the K (**#6**) is omitted, macro will cut to the full depth, as specified in assignment Z (**#26**). This macro is also a good example of using two loops simultaneously - *a loop within a loop*.

The *Figure 42* shows a graphical representation of the variables used for circular pocket roughing macro. Note that the common variable **#120** is defined as a *calculated* value within the macro, based on the tool radius stored in the offset number identified by variable T (**#20**).

MACRO O8106



Circular pocket - roughing macro

VAR	#	Description
X	24	Absolute center location in X
Y	25	Absolute center location in Y
Z	26	Pocket depth
D	7	Pocket diameter (initial)
D	7	Pocket radius #7/2 - applied
K	6	Depth of each cut (Z-axis)
W	23	Width of cut (typical)
n/a	4	Calculated final radius to cut
T	20	Tool radius offset number
F	9	Cutting feedrate

G65 P8106 X- Y- Z- D- K- W- T- F-

Figure 42

Assignment of variables for a typical circle pocket roughing - Macro O8106

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

- Z-clearance above the work is 2 mm or 0.1 inches (automatically selected)
- Feedrate for Z-axis infeed is one half of the programmed feedrate
- Direction of machining is from the center in climb milling mode (G41 D..) at M03
- The tool offset numbers must be within the range of 1-33
- Tool offsets used are for *Memory Offset Type C* - less than 200 offsets - as per Fanuc control designation
- Tool diameter must be greater than 0 and less than the pocket diameter
- *LEAD-IN* and *LEAD-OUT* arcs are identical and calculated by the macro as:

```
(POCKET DIA - TOOL DIA) / 2
```

Additional conditions can be applied only for more advanced approach:

- If cleanup of the bottom is required during finishing, the tool diameter must be *POCKET DIA / 3 or greater*
- Stepped Z-depth may be added to the macro, if desired

```
O0027 (MAIN PROGRAM)
N1 G21                                          Metric mode
N2 G90 G00 G54 X0 Y0 S800 M03                   Any first motion location with spindle speed
N3 G43 Z25.0 H04 M08                            Tool length offset with clearance above part
N4 G65 P8107 X50.0 Y37.5 Z7.5 W49.0 T4 F150.0   Macro call with assignments
N5 Z25.0 M09                                    Retract above work
N6 G28 Z25.0 M05                                Return to machine zero
N7 M01                                          End of current tool
%
```

```
O8107 (CIRCULAR POCKET FINISHING MACRO)
IF[#26 EQ 0] GOTO9101                           Depth of pocket must not be a zero
IF[#23 LE 0] GOTO9102                           Pocket diameter must be a positive value
IF[#20 LE 0] GOTO9103                           Offset number required for cutter radius offset
IF[#20 GT 33] GOTO9104                          Maximum number of offsets is 33
IF[#9 EQ #0] GOTO9105                           Cutting feedrate must be defined
#120 = #[2400+#20]+#[2600+#20]                  Retrieve the stored value of selected tool offset
IF[#120 LE 0] GOTO9106                          Offset value radius must be positive
#23 = #23/2                                     Change diameter of pocket to radius
IF[#23 LE #120] GOTO9107                        Pocket radius must be larger than offset radius
#101 = [#23+#120]/2                             Calculate Lead-in/Lead-out arc radius
#10 = #4003                                     Store current setting of G90 or G91
#26 = ABS[#26]                                  Guarantee that the Z-depth is a positive value
#126 = #4006                                    Check current units (English G20 or Metric G21)
IF[#126 EQ 20.0] THEN #126 = 0.1                Clearance above work is 0.1 inch for G20
IF[#126 EQ 21.0] THEN #126 = 2.0                Clearance above work is 2 mm for G21
G90 G00 X#24 Y#25                               Rapid to start position X and Y
Z#126                                           Rapid to start position Z (above pocket center)
G01 Z-#26 F[#9/2]                               Feed to depth at one half of the feedrate
G91 G41 X#101 Y[#23-#101] D#20 F#9              Motion from center + cutter radius offset
G03 X-#101 Y#101 I-#101                         Lead-in arc tool motion
J-#23                                           Full circle tool motion
X-#101 Y-#101 J-#101                            Lead-out arc tool motion
G01 G40 X#101 Y-[#23-#101]                      Return to the center + cancel cutter radius offset
G90 G00 Z#126                                   Retract above the finished pocket
```

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

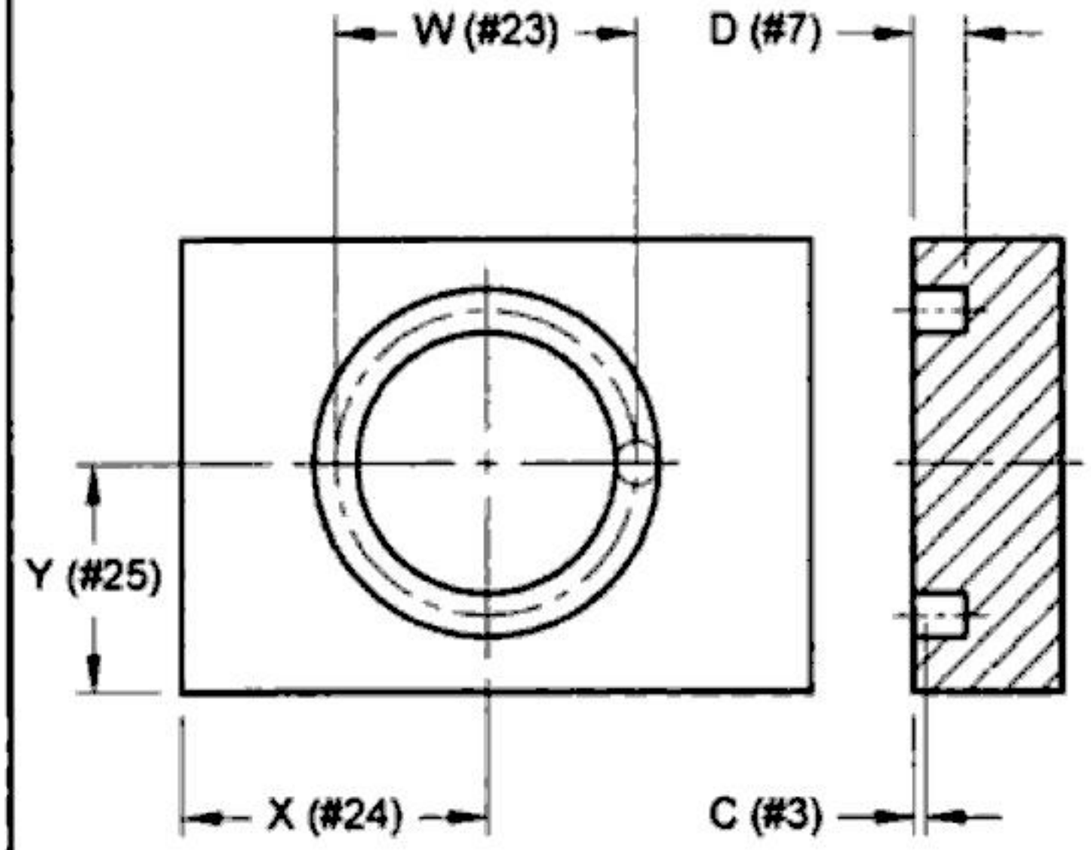
You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

Macro Version Development

Normally, custom macros are developed from scratch - there is no need to create a subprogram first and then 'translate' it. Based on the provided drawing in *Figure 48* (and all similar drawings), a general master template of all required variables can be made *(Figure 49)*:

MACRO O8109



Multiple depth circular groove macro

VAR	#	Description
X	24	Groove center X-coordinate
Y	25	Groove center Y-coordinate
W	23	Groove center diameter
D	7	Distance between centers
C	3	Maximum depth of cut
F	9	Cutting feedrate

G65 P8109 X- Y- W- D- C- F-

Figure 49

Assignment of variables for a typical circular groove cutting macro - Macro O8109

Based on the drawing specifications, the macro call will be developed as expected:

```
G65 P8109 X- Y- W- D- C- F-
```

☞ where ...

X = (#24) Center location of the circular groove in X-axis
Y = (#25) Center location of the circular groove in Y-axis
W = (#23) Groove diameter on the centerline (groove pitch diameter)!
D = (#7) Total depth of the groove
C = (#3) Cutting depth of the groove (maximum depth of each cut)
F = (#9) Cutting feedrate for the circle machining

Although this macro only covers the most important general concepts of controlling the groove depth, it should not be very difficult to develop another macro, one that completes both walls of the groove, once the final depth has been reached (finishing cuts). Regardless of the macro complexity (easy or difficult), thorough pre-planning is absolutely essential. Good planning can save many valuable hours of work and yield excellent results.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

Options Available

Although the most common programming codes that are used as special cycles are the G-codes, M-functions are often used to allow a hardware function call by the special M-code, selected by the CNC macro programmer (often at the manufacturer's level).

Typically, the following addresses can be used for either a macro call or a subprogram call:

- ❑ **G-code macro call** ... *common*
- ❑ **M-code macro call** ... *common*
- ❑ **M-code subprogram call** ... *less common*
- ❑ **S-code subprogram call** ... *less common*
- ❑ **T-code subprogram call** ... *less common*
- ❑ **B-code subprogram call** ... *less common*

G-code Macro Call

Total of 10 (that is ten) of G-codes can be defined as special custom macros that can be called by a G-code. Only the range between `G01` and `G255` is allowed, with the exception of `G65`, `G66`, and `G67` codes. Positive value is the same as `G65`, negative value is the same as `G66` (or `G66.1`).

Depending on the control system, the system parameters related to the *G-code Macro Call* are listed in the following tables (different control systems are shown):

FANUC SYSTEM 0	
G-code Macro Call - 10 options available - G65, G66 and G67 excluded	
Parameter Number	**Description <Valid data 1 - 255>**
220	G-code that calls the custom macro stored in program O9010
221	G-code that calls the custom macro stored in program O9011
222	G-code that calls the custom macro stored in program O9012
223	G-code that calls the custom macro stored in program O9013
224	G-code that calls the custom macro stored in program O9014
225	G-code that calls the custom macro stored in program O9015
226	G-code that calls the custom macro stored in program O9016
227	G-code that calls the custom macro stored in program O9017
228	G-code that calls the custom macro stored in program O9018
229	G-code that calls the custom macro stored in program O9019

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

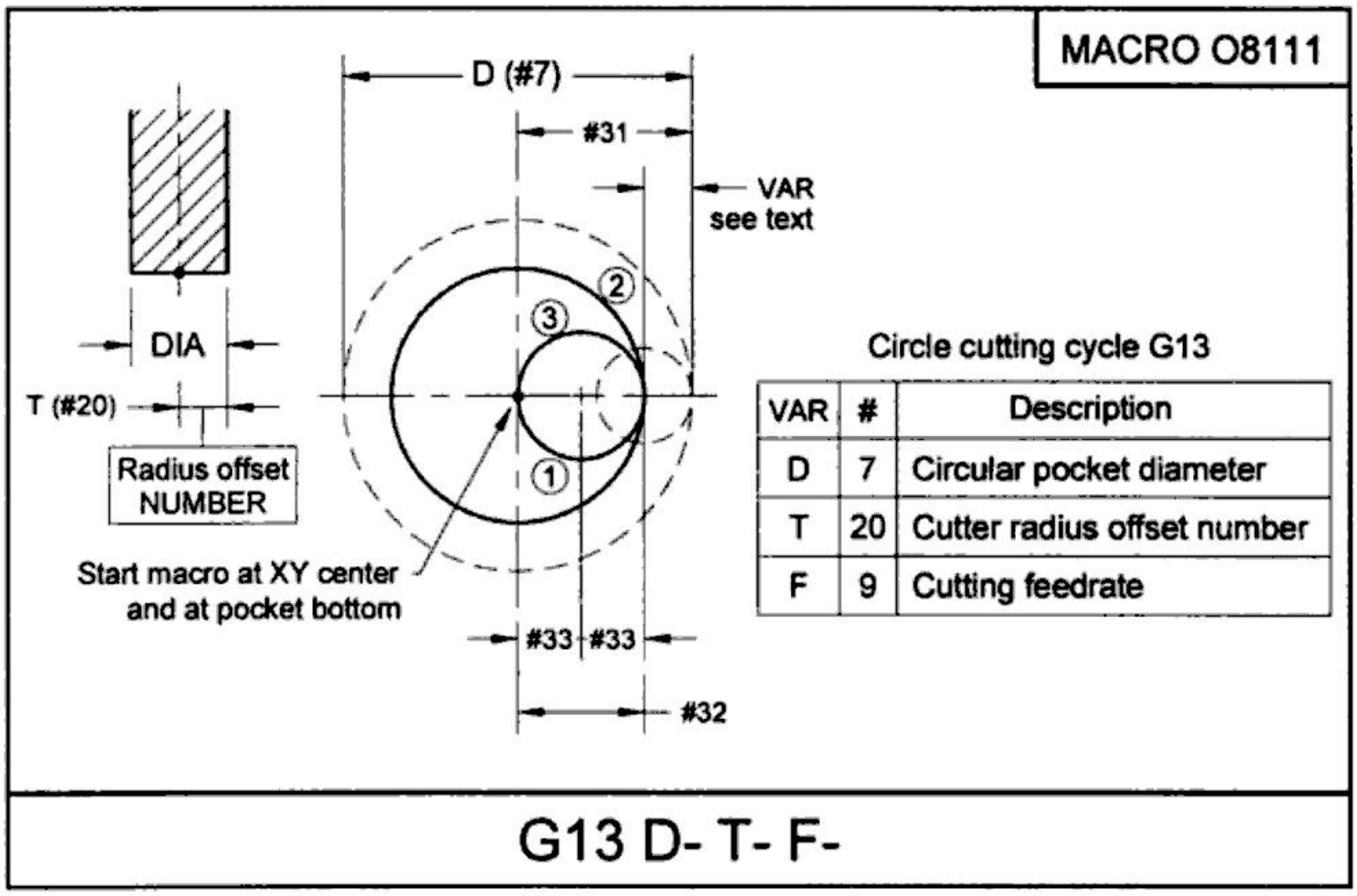


Figure 53

Illustration for the development of G13 circle cutting macro cycle (climb milling mode)

The cutter radius offset using the **G41** command will not be necessary, since the macro reads the radius offset value directly from the control register. In fact, it would be wrong to program **G41**. Without the **G41**, only arcs can be programmed, without linear lead-in and lead-out tool motions. The *Figure 53* also shows assignments of the three variables used in the macro.

The variable data is short for this macro - only three assignments are required:

- **The pocket size - normally given on the drawing as a diameter** — *Variable D (#7)*
- **The tool offset number where the cutter radius is stored** — *Variable T (#20)*
- **The cutting feedrate** — *Variable F (#9)*

There are macros similar to this one that require the pocket radius input rather than its diameter. Selecting the input of a diameter is a better choice, since circular holes or pockets are dimensioned as diameters. For the internal calculations, when the radius is needed, a simple calculation will store the radius as one half of the given diameter.

For most machining applications, the climb milling mode built into the macro (cycle **G13**) is the desirable way of metal removal. However, the macro is not suitable to cut in conventional mode, should such need arise. For that purpose, another macro (cycle **G12**) will have to be developed. Essentially, both macros will be the same, except the machining order of 1-2-3 for the climb milling mode will be reversed to 3-2-1 for the conventional mode (**G03** will change to **G02**). Both macros (cycles) are listed in this chapter.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

DPRNT Function Description

The **DPRNT** function writes a plain text output. The programming format is shown in *Figure 56*.

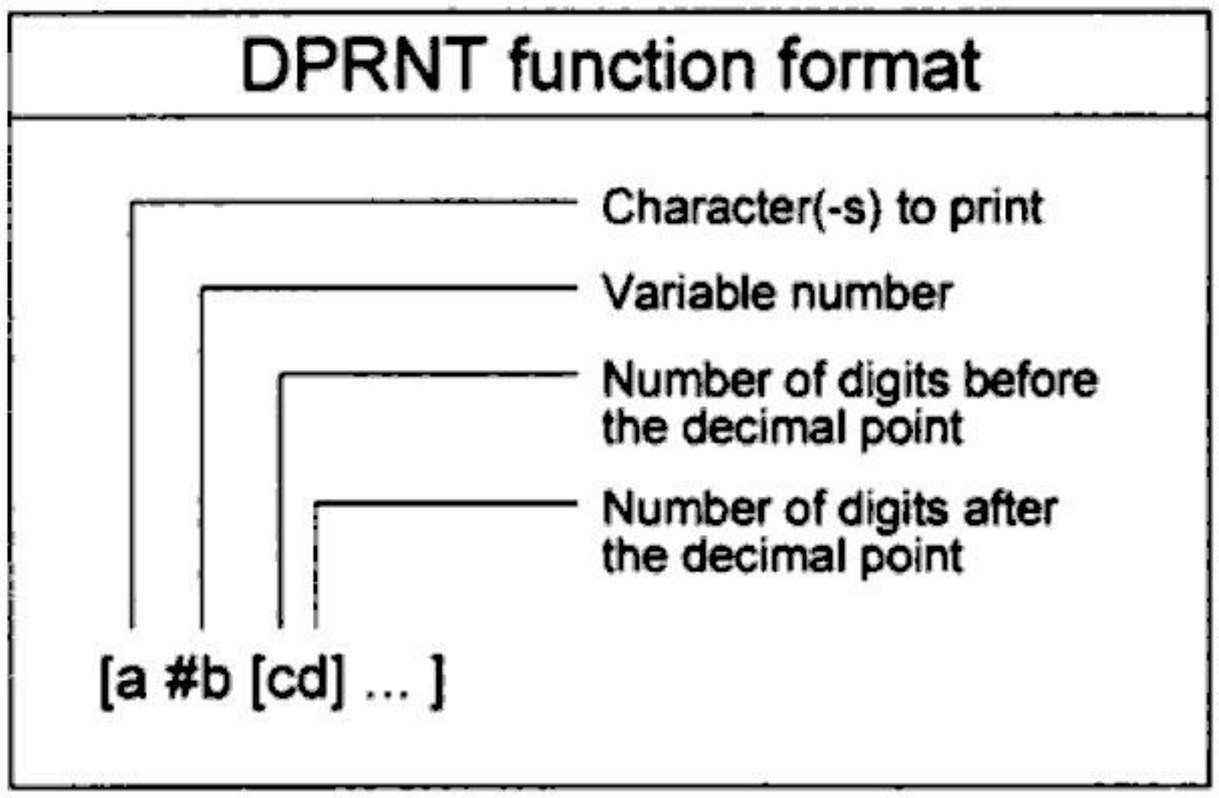


Figure 56

Format structure of the DPRNT function

In the **DPRNT** function, the characters can be the capital letters of the English alphabet (A to Z), all digits 0 to 9, and several special characters (+ - / * ...). Asterisk (*) will be output as the space code. The *End-of-Block* character (EOB) will be output according to the setting of the ISO code. Variables that are vacant (null variables) cannot be output on Fanuc 6 (alarm #114 will result), however, they will be output as *0* on Fanuc 10/11/15/16/18/21. Since the output format depends on the setting of some system parameters, let's look at the settings of the relevant parameters. There is a different between some controls.

Parameter Settings - Fanuc 10/11/12/15

In order to make the data transfer work correctly, some system related parameters have to be set accordingly. The following parameters have to be set for Fanuc controls 10/11/12/15:

Parameter Number	Setting value	Type
0021	Output device interface number for foreground	Byte

☞ where the setting value can be ...

1: Puncher to be connected with CD4A of BASE0 (RS-232C interface 1)
2: Puncher to be connected with CD4B of BASE0 (RS-232C interface 2)
3: Puncher to be connected with CD4 of serial port (RS-232C interface 3)
4: DNC1
13: Puncher to be connected with CD3 of serial port (RS-222 interface)
15: MMC DNC operational interface
16: MMC UPLOAD/DOWNLOAD interface

Note that PUNCHER can be any external RS-232 device.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

```
    . . .
BPRNT or DPRNT                    ... with variable specifications
    . . .
PCLOS
    . . .
```

Macro structure - Version 2

```
POPEN
    . . .
BPRNT or DPRNT                    ... with variable specifications
    . . .
BPRNT or DPRNT                    ... with variable specifications
    . . .
PCLOS
    . . .
```

Output Examples

The following macro example will download the current values of variables within the range of 100 to 149 to an external device, such as a computer disk file (in text format):

- Macro call:

```
G65 P8200 I100 J149               Example of macro call - range of variables specified
```

- Macro definition:

```
O8200 (VARIABLE SETTINGS PRINT-OUT)
POPEN                             Initialize the active communications port
#1 = 0                            Reset variable counter
WHILE[#1 LE [#5-#4]] DO1          Limit loop to selected range of variables
#2 = #[#4+[#1]]                   Current variable number - as a variable
#3 = #4+#1                        Current variable number - as a number (no # symbol)
DPRNT[VAR #3[5] ***DATA #2[57]]   Formatted output includes text, variable ID and value
#1 = #1+1                         Increase the counter of variables by one
END1                              End of loop
PCLOSE                            Close the active communications port
M99                               End - can be M30 if used as main program
%
```

Virtually any data that is stored in the control system can be output as a hard copy or displayed on the screen. Macro programs using the **DPRNT** function can be very useful in keeping records, creating a log of program flow, debugging a troublesome macro, and many other applications. Some common examples are shown in the next section.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

Probing Devices on CNC Machines

Any machine shop can benefit from measuring the part during a machining cycle. The probe unit is mounted in the tool magazine, with is own number and offset settings, just like any other tool. The major difference from other tools is that a probe does not rotate in the spindle when it is used. In-process gauging is closely tied to the macro program that controls the machining, measuring and adjustments of offsets in the control system. Macro features such as branching, conditional testing, looping, use of different variables and access to control features, are all important for programming the various probing devices.

In-Process Gauging Benefits

There are many benefits in measuring the various part features during the machining cycle - they are all relate to the increased productivity and overall accuracy. The most important benefits can be summed up into several items:

- **Part location, length, and diameter of the cutter used can be measured automatically**
- **All three offset memory groups can be calculated automatically and corrected as needed during machining, individually or collectively**
- **Reduction of machine idle time - setup of the part can be greatly simplified - there is no need to spend time on exact physical setup. The actual position of the part and its alignment with the machine axes and/or fixture datums can be corrected mathematically rather than physically**
- **Reduction in the level of scrap - since the actual machined dimensions are monitored by the macro program and the probing cycle, any required offset correction is made automatically**
- **Inspection of the first finished part does not require its removal from the machining area**
- **Broken tools can be detected and proper action followed as specified by the macro program**
- **Initial investment in the technology (equipment and skills) is returned much faster than other methods**
- **CNC operator's confidence level is increased and unattended machining can become a reality**

Types of Probes

Various manufacturers offer many different models of probes. When selecting probes, the main issue is, of course, the probe accuracy. However, accuracy of a probe is not usually an issue, which means the customer looks for additional features when selecting probes. In this context, it is very important to understand is that a probe in itself does not measure, so the question of probe accuracy is purely academic.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

The idea of making the first manufactured part exactly to specifications developed into the implementation of in-process gauging. In some cases, this method of measuring can totally eliminate off-line measuring systems (CMM), or at least complement them. By employing the full in-process gauging exclusively at the machine, CMM systems can be often eliminated (at least for certain applications) and the cost and downtime of the off-line measurement is also eliminated. There are several technological requirements for this technology to be successful.

- The probing system must be used in the toolholder, just as any other tool
- The probing system is stationary (non-rotating) and often locked in an oriented position
- Macro program option has to be available within the control system
- Proper interfaces between the probe and control system have to be established
- Special macro programs have to be developed and maintained for the measuring

In terms of economics, in-process gauging does extend the total cycle time, often quite significantly. Even if not every part in the batch is measured, the average cycle time must be considered. When the CNC machine tool is used as a measuring device, the equipment involved in the various stages of measurement has to be certified or calibrated. Calibrating of the probing device is done on a verified gauge (gage), using a macro program. The principles of calibration have been discussed earlier.

Many CNC machine tools incorporate a unique design that allows for the inclusion of a calibrated artifact within the working cube of the machine. Working cube is defined by the combined maximum amount of travel along the X-axis, Y-axis and Z-axis. If only two axes are considered, the term 'working cube' is changed into a working area or a working envelope.

Features to be Measured

In order to determine what features of the part can be measured on a CNC machine, a great deal depends on the type of probe used. In their basic applications, virtually every probe can measure the following features of a part or within a particular part:

- Center measurement
- External diameter
- Internal diameter
- External length - width
- Internal length - width
- Depth of a feature
- Angle of a feature

There are many other part features that can also be measured and some are more common than others. The most typical item in this group covers the part feature location, such as a center of a hole, distance between two points, diameter, and so on. Since the typical part feature is normally specified by at least the X and Y axes, it means a macro development that uses a combination of the various results and manipulating them mathematically. All this takes place within the macro body. In many cases, directly returned value are not used as such, but for calculations of a difference in values between two measurements. How this difference is handled depends on the particular application. Normally, the calculated value is used to adjust a particular offset setting.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

FANUC CNC Custom Macros

SPECIAL BONUS CD-ROM INCLUDED!

Programming resources for Fanuc Custom Macro B users

An invaluable companion to the author's best selling **CNC Programming Handbook**, this book is an extensive introduction to the subject of macros (known as Custom Macros or User Macros). Its purpose is to make you aware of what macros are, how to develop them, and how to use them effectively. It also explores important related subjects and identifies several other helpful topics in this increasingly important and exciting field of CNC programming.

Knowledge of macros is becoming more and more essential, as companies large and small look for more efficient ways of CNC program development. This approach does not replace but rather complements other programming methods. As an extension of manual programming, macros offer a much higher level of sophistication and often can serve as a special solution to special requirements.

This book offers many practical do's and don'ts while covering all the popular Fanuc control systems exclusively. Macros for different controls share a common approach and mainly differ in their syntax. Learning Fanuc custom macros will be a benefit if you should need to learn a macro for a different control at another time.

Numerous examples and sample programs are used throughout this book. Their purpose is to serve not only as practical applications of the techniques presented, but (for many of them) as the basis of ready-to-run macro programs. To help make your use of these programs as easy and as reliable as possible, all the sample programs have been reproduced on the enclosed CD.

CNC programmers and service technicians will find this book a very useful training and reference tool to use in a production environment. Also, it will provide the basis for exploring in great depth the extremely wide and rich field of programming tools that macros truly are.

ABOUT THE AUTHOR Peter Smid is a professional consultant, educator, and speaker, and has many years of practical, hands-on experience with CNC and CAD/CAM applications on all levels. He consults for both manufacturers and educational institutions on the practical use of CNC technology, part programming, CAD/CAM, advanced machining, tooling, setup, and many other related fields. Hundreds of organizations have used his services and benefited from his wide-ranging industrial background in CNC programming, machining, and company-oriented training. His enormously popular *CNC Programming Handbook* is also available through Industrial Press.

ISBN 0-8311-3157-8

INDUSTRIAL PRESS INC.
200 Madison Avenue, New York, NY 10016-4078
TEL: 212-889-6330 TOLL-FREE 1-888-528-7852 FAX: 212-545-8327
Web Address: www.industrialpress.com Email: info@industrialpress.com