CHAPTER 10
VARIABLES, MATH CAPABILITY, PROGRAM FLOW WORD COMMANDS AND REPORT GENERATION

This chapter discusses variables, math capability, program flow word commands and report generation, all programming procedures that can be used within programs and subprograms. These same procedures are also used in writing custom macros which will be discussed in chapter 11 "Custom Macros".

Variables are symbolic representations of specific numerical values that are assigned to them. Variables are used to input data into programs and subprograms in order to manipulate data and machine control commands within a program level or levels.

Math capability is the execution of the mathematical functions addition, subtraction, multiplication, division and equals to within programs/subprograms.

Program Flow Word Commands are word commands that can be used to manipulate the order of data processing within a program/subprogram.

Report Generation is the output of data such as comments, text, numbers and variables from programs/subprograms to external devices, such as printers and tape punch units.

10.1 VARIABLES

Variable is the term given to the concept of symbolic representation of a specific numerical value. Variables have numerical values assigned to them. Variables are used to input data into programs/subprograms and to manipulate data and machine control commands within a program level or levels.

Call level has a significant impact on variable use and function when used in custom macros. Different types of variables exist to facilitate communication and operation among various macro call command levels. SEE Chapter 11 for more information on variable use in custom macros.

Precision

The precision of variables used in custom macro is limited to an eight (8) place decimal floating point number. In this system, the magnitude of a number effects its precision. Precision is the number of digits to the right of the decimal. The limits of magnitude and precision are as follows.

Magnitude

Maximum: 99999999.
Minimum: -99999999.

Magnitude/Precision

Maximum: +.99999999
Minimum: +99999999.

A floating point system allows the control to deal with large magnitude, low precision decimals as well as with small magnitude, high precision decimals with a minimum of memory use. If the use of two or more variables in combination is encountered and they are not of equal precision, calculation uncertainty can result; but the amount the calculation is off .5 is minute. Remember, precision is limited to an eight place (total) decimal system.

The machine control system consists of numerical control (NC), program control (PC) and interface. Variables have different precision among PC and NC. The PC can access variables with limited precision. The PC effectively reads a fix (section 10.2.2) of a number or truncates the decimal portion, leaving only the integer portion. Therefore, if a program command must pass a decimal quantity to the PC, it will be multiplied by 10,000. This is done in the Monarch probe cycles and it is not likely to appear in a standard program.

Variable Types

With the Fanuc 11 control, there are three types of variables.

- Local Variables. There are 33 local variables (#1 through #33). These 33 variables are common to all levels when used in a program/subprogram, but limited to each level when used in a custom macro. SEE Chapter 11.

- Common Variables. There are 100 variables in an 11MA control (#100 through #149, #500 through #549) that are common to all levels. In an 11MF control, there are 200 variables (#100 through #199, #500 through #599) that are common to all levels.

- System Variables. There are numerous variables common to all levels.

10.1.1 Local Variables

Local variables are used to assign arguments in programs or subprograms. Up to 33 variables may be used. See Chapter 11 for use of local variables with a custom macro. All local variables are reset to vacant (or empty) on an M2 or M30 command, or on a Clear, E-Stop Reset or Retract pushbutton operation.

NOTE: Not all local variables need to be programmed. If a local variable is not set within the program, it is said to be "vacant" (or empty). If a variable is "vacant", it has no value. This does not mean the variable is zero. Zero is a value. Vacant means empty.

The variable #0 is always vacant and can not be set to any other value. A local variable is set equal to vacant if it is not set equal to a value within a given program. If the programmer wishes to test a variable for vacancy, it can be tested with an IF statement to see if it is equal to #0. This process would be useful for error checking in certain types of programs.

When local variables are used within the program itself, they are designated by #N where N ranges from 1 through 33. Local variables in a program can be assigned a value in a number of ways.

NOTE: When referring to the # sign, it is called 'pound'. #1 is referred to as 'pound one'.

1. A variable can consist of the code # and a number, i.e. #1, #2, #3, etc. up to #33.
2. A variable can consist of a formula, i.e. #1 = [#1 + 1.]. In this example, one is added to the current value in variable #1 and resets variable #1 to this amount.

When local variables are used in the macro call block, they are assigned by letter address. SEE Chapter 11 for more information.

10.1.2 Common Variables

Common variables can be used for data manipulation within the main program or subprogram without regard to level. They may be used common to or among subprogram or program levels.

There are100 common variables on the 11MA control and 200 common variables on the 11MF control that are common to all program/subprogram levels. #100 through #149 (11MA control) or #100 through #199 (11MF control) are cleared at power down. #500 through #549 (11MA control) or #500 through #599 (11MF control) are not cleared at power down..

A. Reserved Common Variables

#100 through #199 and #554 through #569 for Conversational Automatic Programming (11MF control)

#500 through #533 for Monarch's probe cycles

#534 through #549 for Monarch's MCL, used on all machines. **NOTE**: Some of the variables reserved by the Monach MCL can be utilized by the end user.

#535 is always equal to the absolute pocket of the tool in the spindle. This is usually used in conjuction with the M12 command. SEE chapter 6.

#536 is the value to adjust all tool length offsets, scaled by 10,000 when running in inch, or scaled by 1000 when running in metric. This is used with the M950 command. SEE chapter 6, M950 command and the Monarch Probe Manual, 'Delta Z' cycle for more information.

#537 is the last value the tool offsets were adjusted by the M950 command.

B. Labeled Variables

A name up to eight (8) characters in length (with no spaces) may be assigned to each of twenty variables (#500 through #519). These names are only significant for display on the CRT screen under 'SETTING, Common Variables'. These variables must be indexed by numerals for our program use. A name may be assigned to a single variable or to a string of variables using the following format.

Twenty (20) non-volatile variables #500 to #519 are reserved for Monarch Cortland's probe cycles. However, if you do not have Monarch's probing cycles, feel free to use these.

```
SETVN500[BOREDIA,RPLANE,ZDEPTH,OFFSET]
```

Where SETVN500 means set variables starting with #500. #500 is BOREDIA, #501 is RPLANE, #502 is ZDEPTH. The commas between labels tell the control to move to the next variable.

10.1.3 System Variables

System variables serve as a means of communication of NC or machine data to programs and subprograms. These variables are shared between the user and Monarch Cortland.

There are numerous variables common to all levels. Some system variables are given in the following charts:

- Chart 10-1: Miscellaneous System Variables
- Chart 10-2: System Variables for Work Offsets
- Chart 10-3: System Variables for Tool Offsets

If needed, additional system variables are given in Fanuc's Operator Manual.

Reserved System Variables

#2090 through #2099, #2290 through #2299, #2490 through #2499 and #2690 through #2699 for Monarch's probe cycles.

All variables related to the G54 work coordinate system.

Special System Variables

Operator Messages

System variable #3006 can be used to activate operator messages that force the message screen and stop the machine.

```
N10 #3006 = 1(message)
```

The above line demonstrates how a message can be put on the screen to guide the operator through a sequence, such as automatic setup. The message is placed between the parenthesis. It can be up to 26 characters long.

Alarm Messages

The control has the capability to put an alarm number and message on the screen with a flashing alarm signal on the CRT. This will also stop the machine.

```
N10 #3000 = xx(alarm message)
```

The above demonstrates how an alarm message can be forced.

xx is the alarm number, It can range from 10 to 1000. The alarm message is put between the parenthesis and can be up to 26 characters long.

Single Block Suppression

Single block suppression will allow you to make a portion of a program, subprogram or a macro execute like one block with system variable #3003.

The following subprogram will run in a single block:

```
O10(HOLE)
N10 #3003 = 1
N20
N30          NC statements
N40
N50 #3003 = 0
N60 M99
```

#3003 = 1 turns on single block suppression
#3003 = - turns off single block suppression

In other words, all the blocks between #3003 = 1 and #3003 = 0 will be executed as one block.

The above message and single block suppression commands are used by Monarch's probe cycles.

CHART 10-1: MISCELLANEOUS SYSTEM VARIABLES

These values could be used to determine the current position of an axis, or the value of the active tool offset.

SYSTEM VARIABLE	POSITIONAL INFORMATION		READING IN DURING MOVEMENT
#5001 #5002 #5003 #5004 #5005	X axis block end coordinate Y axis block end coordinate Z axis block end coordinate W axis block end coordinate B axis block end coordinate		Possible
#5021 #5022 #5023 #5024 #5025	X axis present coordinate Y axis present coordinate Z axis present coordinate W axis present coordinate B axis present coordinate	FROM MACHINE ZERO	Impossible
#5041 #5042 #5043 #5044 #5045	X axis present coordinate Y axis present coordinate Z axis present coordinate W axis present coordinate B axis present coordinate	WORK OFFSET EFFECTIVE	Impossible
#5061 #5062 #5063 #5064 #5065	X axis skip signal position Y axis skip signal position Z axis skip signal position W axis skip signal position B axis skip signal position		Possible
#5081 #5082 #5083 #5084 #5085	X axis tool offset value Y axis tool offset value Z axis tool offset value W axis tool offset value B axis tool offset value		Impossible
#5101 #5102 #5103 #5104 #5105	X axis servo position deviation Y axis servo position deviation Z axis servo position deviation W axis servo position deviation B axis servo position deviation		Impossible

CHART 10-2: SYSTEM VARIABLES FOR WORK OFFSETS (#5221 - #5325)

These values could be used to change or look at the values in a particular workpiece offset.

Work offsets consist of five (5) available work coordinate systems G55 through G59 (2 through 6). A sixth work coordinate system G54 is restricted.

VARIABLE NO.	VALUE	WORK COORDINATE SYSTEM
#5221	X work zero point offset value of the 1st axis	G54
#5222	Y work zero point offset value of the 2nd axis	
#5223	Z work zero point offset value of the 3rd axis	
#5224	W work zero point offset value of the 4th axis	
#5225	B work zero point offset value of the 5th axis	
#5241	X work zero point offset value of the 1st axis	G55
#5242	Y work zero point offset value of the 2nd axis	
#5243	Z work zero point offset value of the 3rd axis	
#5244	W work zero point offset value of the 4th axis	
#5245	B work zero point offset value of the 5th axis	
#5261	X work zero point offset value of the 1st axis	G56
#5262	Y work zero point offset value of the 2nd axis	
#5263	Z work zero point offset value of the 3rd axis	
#5264	W work zero point offset value of the 4th axis	
#5265	B work zero point offset value of the 5th axis	
#5281	X work zero point offset value of the 1st axis	G57
#5282	Y work zero point offset value of the 2nd axis	
#5283	Z work zero point offset value of the 3rd axis	
#5284	W work zero point offset value of the 4th axis	
#5285	B work zero point offset value of the 5th axis	
#5301	X work zero point offset value of the 1st axis	G58
#5302	Y work zero point offset value of the 2nd axis	
#5303	Z work zero point offset value of the 3rd axis	
#5304	W work zero point offset value of the 4th axis	
#5305	B work zero point offset value of the 5th axis	
#5321	X work zero point offset value of the 1st axis	G59
#5322	Y work zero point offset value of the 2nd axis	
#5323	Z work zero point offset value of the 3rd axis	
#5324	W work zero point offset value of the 4th axis	
#5325	B work zero point offset value of the 5th axis	

CHART 10-3: TOOL OFFSETS

A programmer may want to adjust tool offsets to compensate for tool length or diameter wear using a result from a probe cycle, i.e. contour mill a counterbore, probe a diameter, adjust cutter radius compensation or re-cut a counterbore.

Offset Number	LENGTH (Hxx)		RADIUS (Dxx)	
	Geometry (TLG) Comp	Wear (TLW) Comp	Geometry (TRG) Comp	Wear (TRW) Comp
1	#2001	#2201	#2401	#2601
2	#2002	#2202	#2402	#2602
3				
●				
●				
99	#2099	#2299	#2499	#2699

NOTE: The 11MA control only has variables #2001 through #2099. Machines with the probe option or 11MF control have all four columns of variables as shown.

10.2 MATHEMATICAL CAPABILITY

The Fanuc 11 control is capable of executing mathematical commands. These math commands are included within the enhanced programming language user macro. Algebraic expressions or expressions consisting of ordered operations with variables and/or constants, math operators and functions are the means of numerical data calculation available to the user.

The mathematical capability consists of:

. Math Operations
. Mathematical Functions
. Algebraic Expressions

10.2.1 Mathematical Operations (+ – * / =)

Addition, subtraction, multiplication and division can be done by the control during the execution of a program. Listed below are these mathematical operations and the appropriate programming symbol/character for each.

SYMBOL/ CHARACTER	MATHEMATICAL OPERATION
+	Addition
–	Subtraction
*	Multiplication
/	Division
=	Equal To

Expressions using mathematical operations, variables, constants and functions may be used together with letter addresses as program commands.

Example: N10 G0 X[#1 + #2]

In block N10, (G0) selects positioning mode, (X[#1 + #2]) positions the X axis to the position equal to the value in variable 1 plus the value in variable 2.

Given: #1 = 1, #2 = 10 therefore X will move to 11 in the selected units system.

Operations may also be used in expressions or formula which equate (=) one left side of an expression to a right side expression. Constants, variables, functions and operators may be used.

Example: #1 = #2 + #1

10.2.2 Mathematical Functions

A function is a multi-letter word which does a given mathematical routine on a specified argument and returns the numeric value to the program. Functions are in the form, ABC[#j].

Where: ABC is a multi-letter word for a specified function, such as SIN, TAN, ABS, etc.

[#j] is argument or object of function enclosed within open and closed brackets []. Functions may be included in algebraic expressions or word address commands.

Sixteen mathematical functions which are summarized in Figure 10-4 are available to the programmer.

FUNCTION	DESCRIPTION	EXAMPLE	INPUT #J #J/#K	ANSWERS #I
#i = SIN[#J]	Sine (degrees)	#i = SIN[#j]	45	.707
#i = COS[#j]	Cosine (degrees)	#i = COS[#j]	45	.707
#i = TAN[#j]	Tangent (degrees)	#i = TAN[#j]	45	1
#i = ATAN[#j/#k]	Arc Tangent (degrees)	#i = ATAN[#i/#k]	1	45°
#i = SQRT[#j]	Square Root	#i = SQRT[#j]	4	2
#i = ABS[#j]	Absolute Value	#i = ABS[#j]	−4	4

#i = BIN[#j]	Conversion from BCD to BIN
	
	BCD = Binary Coded Decimal
	BIN = Binary
		#i = BIN[#j]	99	153

#i = BCD[#j]	Conversion from BIN to BCD	#i = BCD[#j]	153	99
#i = ROUND[#j]	Rounding (word address rounded to least input increment)	#i = ROUND[#j] #I = ROUND[#j]	4.6 4.61151	5.0 4.6115
#I = FIX[#j]	Truncate (disregard decimal portion of #j)	#i = FIX[#j]	4.6	4.0
#i = FUP[#j]	Round up to nearest integer	#i = FUP[#j]	4.4	5.0
#i = LN[#j]	Natural Log	#i = LN[#j]	1	0
#i = EXP[#j]	Exponent with base	#i = EXP[#j]	1	2.718-2.818

FIGURE 10-4
MATHEMATICAL FUNCTIONS AVAILABLE TO THE PROGRAMMER

10.2.3 Algebraic Expressions

In the 11M control, constants, variables, mathematical operations and functions may be combined to form algebraic expressions in programs.

Algebraic expressions may contain:

1. A real or integer constant. Example: real = 10.1; integer = 10

   An integer has an understood decimal point.

2. A variable index. Example: #1

3. Mathematical Operation. Example #1 = 1. + 1.

4. Functions. Example: #1 = SIN[45]

5. Delimiters which are open and close brackets [ ].
   Example: #1 = #1*[#2 + #3]

These delimiters assign a priority or order of operation to the algebraic expression. Operation is from inner most bracket to outer most bracket. The number of left brackets must equal the number of right brackets in an expression. Up to five (5) levels of brackets including the brackets used in functions may be included.

Order of operation is further controlled by execution sequence of various mathematical operations contained within the expression.

. Functions
. Multiplication
. Division
. Addition
. Subtraction

In the absence of brackets, the above operations control the order of processing.

Order of operation affects both sides of an algebraic expression. Left side can be either a singular variable or an index variable, i.e. #1 = or #[#1 + #2] =

Right side can be either singular or multiple expressions consisting of constants, variables and math operators, i.e. #1 = SIN[[[#2 + #3]*#4 + #5]*#6]

Where:
1. [#2 + #3]
2. Result of 1, multiplied by #4
   [#2 + #3]*#4]
3. Result of 2, plus #5
   [[#2 + #3]*#4 + #5]
4. Result of 3, multiplied by #6
   [[[#2 + #3]*#4 + #5]*#6]
5. Result of 4 as argument for function SIN
   SIN[[[#2 + #3]*#4 + #5]*#6]
6. #1 equated to result of 5

Below is an example of three fold nesting.

```
#[#31*#18 + #30*[1-#18]] = #[#30*[1-#18]*#33] + #[#30*#18] + #[#32*[1-#18]]
```

Note: Number of math operators should be one less than the number of constants and variables.

Further Considerations

The Fanuc 11m control processes math statements, and conditional branches very fast, almost too fast. If you are trying to single step through a program in which several lines in a row contain math statements, conditional branches, variable assignments, macro or subprogram call blocks, etc, one push of cycle start will probably cause several blocks to execute. This is usually not a problem unless you want to be able to stop after each to correct answers.

Example:

```
O10(M80 example 1)
N10 G90 G0 G54

NC Statements

N80 G28 Z0
N90 #1 = SIN[45]
N100 #2 = [[15. + 2.]*#1]
N110 G90 G0 X#2

Program Continues
```

In The above example if you were running in single and were sitting at N80 and pressed cycle start, the control would exectue blocks N80 through N100 all at once. Again this is not a problem unless you want to stop after each block. To do that you would have to insert an M80 command after each math statement as shown below.

```
O10(M80 example 1)
N10 G90 G0 G54

NC Statements

N80 G28 Z0
N85 M80
N90 #1 = SIN[45]
N95 M80
N100 #2 = [[15. + 2.]*#1]
N105 M80
N110 G90 G0 X#2

Program Continues
```

Then once the program is proven out you can take out the M80 commands.

Here is an example where an M80 is needed to slow down the control to ensure that the correct tool offset is activated.

```
O8973(SAFE START)
N10 G0 G90 G54
N20 G49 Z0 H0
N30 G28 Z0
N40X10.Y10.T20M6
N50M80
N60S1000M3
N70G43H#535Z.1
```

Program Continues

In this example the #535 is used for the H offset number (tool length offset). #535 is always equal to the pocket number of the tool in the spindle. It is set by Monarch's interface. In this example, the M80 has to be between thetool change and the G43 block. If it is not, the control will read ahead and substitute the #535 value in block N70 before the tool change is made. If that happened the tool length offset that is activated when the machine acts on block N70 would be the one of the previous tools in the spindle. As you can visualize, this could be disastrous.

The rule about using an M80 in a part program is this; it should be used anytime you will be looking at a variable controlled by an external device (the machine is considered an external device). Examples would be an axis position, tool in the spindle, probe trip position, etc.

If you are not sure if you need an M80 command, put it in. It will not hurt anything.

10.3 PROGRAM FLOW CONTROL COMMANDS

Program flow within a given program level can be controlled by unconditional or conditional branch commands and/or iteration commands.

10.3.1 Unconditional Branch

An unconditional branch is a statement which forces a jump from the current active block (containing the unconditional branch) to the specified block within the program level.

Example: GOTO10

In this example, the program unconditionally jumps to block N10.

10.3.2 Conditional Branch

Conditional branch is a statement which if a condition is true, forces a jump from the current active block (containing the conditional branch) to the specified block within the program level. If the conditional branch is false, the program processes and acts upon the next block.

Example: IF[#1EQ#2]GOTO10

In this example, the program would jump to block N10 if #1 is equal to #2. If #1 is not equal to #2, the next block would be processed and acted upon.

IF[#1EQ#2] is considered a conditional expression. The evaluation of which results in either a true (success) or false (failure) answer.

Logical operators available to the programmer are:

EQ	equal to
NE	not equal to
GT	greater than
LT	less than
GE	greater than or equal to
LE	less than or equal to

So a conditional expression consists of a left term which is compared via logical operators to a right term. These terms may consist of constants, variables or functions.

10.3.3 Iteration (repetition) using IF and GOTO Statements

Iteration or repetition may be a requirement for many programs or subprograms. **Note:** Where operational speed is a concern, an unconditional and/or conditional branch, counter and counter increment may not be desired. Refer to the section further along on WHILE DO statements, they run faster.

Example 1:

```
N10 #32 = 10 (number of rep)
N20 #33 = 0 (initialize counter)
N30 IF[#33EQ#32]GOTO220
.
.
NC statements
.
.
N200 #33 = #33 + 1
N210 GOTO30
N220    program continues
```

Example 2: Iteration loop counts up with a conditional and an unconditional branch. In this example, the counter is initialized and incremented.

```
N10 #32 = 10 (number of reps)
N20 #33 = 0 (initialize counter)
N30 #33 = #33 + 1
.
NC statements
.
N210 IF[#33LT#32]GOTO230
N220 GOTO30
N230 Program continues
```

Example 3: Iteration loop count down with a conditional and an unconditional branch. In this example, the counter is initialized and decremented.

```
N10 #32 = 10 (number of reps)
N20 IF[#32EQ0]GOTO220
.
NC statements
.
N200 #32 = #32-1
N210 GOTO20
N220    program continues
```

Example 4: Iteration loop count down with a conditional branch. The counter is initialized and decremented.

```
N10 #32 = 10 (number of reps)
N20 #32 = #32-1
.
NC statements
.
```

```
N200 IF[#32GT0]GOTO20
N220    Program continues
```

Example 5: Sample program to drill a series of holes using IF and GOTO statements to create a loop. This cycle will drill 10 holes at 1.0 inch increments along the X axis, starting at X1.0 Y1.0.

```
O0015(IF STATMENT)
N10 G90 G0 G54
N20 G49 Z0 H0
N30 G28 Z0 M8
N40 G60 W-10.
N50 G55 X1. Y1. T1 M6
N60 S1000 M3
N70 G43 Z.1 H1
N80 G99
N90 G81 R.1 Z-1. F10.
N100 #24 = 10. (NUMBER OF HOLES)
N110 #32 = 1. (INITIALIZE COUNTER)
N120 IF #32 GE #24 GOTO 160
N130 G91 X1.
N140 #32 = #32 + 1.
N150 GOTO 120
N160 G90 G0 G54
N170 G49 H0 Z0
N180 G28 Z0
N190 M30
```

N10 through N90 are standard NC programming blocks.

N100 sets #24 equal to 10, the total number of holes to be drilled.

N110 sets #32 = 1 to initialize the counter. It is set to 1 because one hole has already been drilled in block N90.

N120 is the beginning of the loop. The control will repeat blocks N120 through N150 until the condition in block N120 is true (#32, the counter is greater than or equal to #24, the total number of holes.) If the condition is true, the control will execute the GOTO command in this block and jump to N160 to continue with the program.

N130 puts the control in incremental mode (G91) and moves the X axis 1.0 inch then drills the next hole.

N140 adds one to the counter #32

N150 GOTO 120 sends the control back to block N120 to see if the condition in block N120 is true yet.

N160 the program continues.

NOTE 1: Air tapping (G184) will not work in this example due to its use of common variables.

NOTE 2: The variables for this example and the remaining examples in this chapter were selected because of their letter representation when used as a custom macro. SEE the example in Chapter 11 "Custom Macro".

When executing GOTO statements, operational speed is reduced if a very long program is being run. This occurs because the control searches forward to the end of the program then reviews the program from the top. So in some cases an IF structure for a loop may be time intensive.

In cases where time is a consideration and loops are nested to three (3) or less levels a WHILE DO statement should be used.

10.3.4 Iteration (repetition) using WHILE DO Statements

WHILE DO statements should be used where time is a consideration and loops are nested to three (3) or less levels.

With the use of a WHILE DO command, a number of blocks may be repeated.

Example:

```
WHILE[conditional expression]DO1
.
.
NC statements
.
END1

WHILE[conditional expression]DO2
.
NC statements
.
END2

WHILE[conditional expression]DO3
.
NC and macro statements
.
.
END3
```

RULES

1. WHILE DO 1, 2 and 3 are the only indexes allowed.

2. WHILE DO index and END index must exist for each DO condition.

3. WHILE DO statements may be nested to 3 levels.

```
WHILE[conditional exp]DO1
    WHILE[conditional exp]DO2
        WHILE[conditional exp]DO3
        END3
     END2
END1
```

4. WHILE DO ranges cannot intersect

```
WHILE[    ]DO1
    WHILE[    ]DO2
END1
    END2
```

5. A branch cannot be made from outside to inside a WHILE DO range

```
GOTO10
WHILE[    ]DO1
N10
END1
```

6. A branch can be made from inside to outside a WHILE DO range

```
WHILE[    ]DO1
GOTO 10
END1
N10
```

7. Subprograms of programs can be called from within a WHILE DO range. WHILE DO statements can be nested up to three more levels in the called subprogram.

```
WHILE[    ]DO1
M98 P1000
G67
END1
```

EXAMPLES OF WHILE DO STATEMENTS

Example 1: Lets take the same example we used with the IF statement to drill a series of holes and re-write the program to use the WHILE DO statement. This cycle will drill 10 holes at 1.0 inch increments along the X axis, starting at X1.0 Y1.0.

```
O0015(WHILE-DO)
N10 G90 G0 G54
N20 G49 Z0 H0
N30 G28 Z0 M8
N40 G60 W-10.
N50 G55 X1. Y1. T1 M6
N60 S1000 M3
N70 G43 Z.1 H1
N80 G99
N90 G81 R.1 Z-1. F10.
N100 #24 = 10. (NUMBER OF HOLES)
N110 #32 = 1. (INITIALIZE COUNTER)
N120 WHILE [#32 LT #24] DO1
N130 G91 X1.
N140 #32 = #32 + 1.
N150 END1
N160 G90 G0 G54
N170 G49 H0 Z0
```

```
N180 G28 Z0
N190 M30
```

N10 through N90 are standard NC programming blocks.

N100 sets #24 equal to 10, the total number of holes to be drilled.

N110 sets #32 = 1 to initialize the counter. It is set to 1 because one hole has already been drilled in block N90.

N120 is the beginning of the WHILE DO loop. The control will repeat the blocks between the WHILE and END commands until the condition in the WHILE block is false. In this example blocks N130 and N140 will repeat as long as the counter #32 is less than #24, the total number of holes.

N130 puts the control in incremental mode (G91) and moves the X axis 1.0 inch then drills the next hole.

N140 adds one to the counter #32

N150 END1 marks the end of the WHILE DO loop. The control goes back to block N120 to see if the loop should run again. If the condition in N120 is false (#32 is no longer less than #24) the control will move on to block N160.

N160 the program continues

NOTE: Air tapping (G184) will not work in this example due to its use of local variables.

Example 2: Example 1 used two local variables and one WHILE DO loop to make a row of holes in the X axis. Let's take that program and add a couple more variables and a second WHILE DO loop, nesting the first one, to make the machine make a grid of holes. This cycle will drill 50 holes at 1.0 inch increments; 10 rows of five holes along the Y axis, starting at X1.0 Y1.0.

```
O0015(GRID)
N10 G90 G0 G54
N20 G49 Z0 H0
N30 G28 Z0 M8
N40 G60 W-10.
N50 G55 X1. Y1. T1 M6
N60 S1000 M3
N70 G43 Z.1 H1
N80 G99
N90 G81 R.1 Z-1. F10.
N100 #24 = 10.(NUMBER OF HOLES IN X AXIS)
N110 #21 = 1.(INCREMENT BETWEEN HOLES IN X AXIS)
N120 #32 = 1.(INITIALIZE COUNTER FOR X AXIS)
N130 #25 = 5.(NUMBER OF HOLES IN Y AXIS)
N140 #22 = 1.(INCREMENT BETWEEN HOLES IN Y AXIS)
N150 #33 = 1.(INITIALIZE COUNTER FOR Y AXIS)
N160 WHILE [#32 LE #24] DO1
N170 IF #32 EQ 1 GOTO190
N180 G90 Y1. G91 X#21
```

```
N190 WHILE [#33 LT #25] DO2
N200 G91 Y#22
N210 #33 = #33 + 1
N220 END2
N230 #33 = 1.(RESET Y AXIS COUNTER)
N240 #32 = #32 + 1.
N250 END1
N260 G90 G0 G54
N270 G49 H0 Z0
N280 G28 Z0
N290 M30
```

N10 through N90 are standard NC programming blocks.

N100	sets #24 equal to 10, the total number of holes to be drilled in the X axis.
N110	sets #21 = 1. This is used for the increment between holes along the X axis.
N120	sets #32 = 1 to initialize the counter used for the X axis holes. It is set to 1 because one hole has already been drilled in block N90.
N130	sets #25 = 5, the total number of holes to be drilled in the Y axis.
N140	sets #22 = 1. This is used for the increment between holes along the Y axis.
N150	sets #33 = 1 to initialize the counter used for the Y axis holes.
N160	is the beginning of the first WHILE DO loop. This loop controls the number of rows machined along the X axis. The machine will repeat the blocks N170 through N240 as long as the counter #32 is less than or equal to #24, the total number of holes along the X axis.
N170	is a conditional branch that is needed to jump past the move in block N180 if the first row along the X axis is being machined.
N180	(G90 Y1. G91 X#24) will move the machine to an absolute Y axis position of 1. inch and an incremental X axis position of 1. inch.
N190	is the beginning of the second WHILE DO loop. The control will repeat the blocks N200 through N210 as long as the counter #33 is less than #25, the total number of holes in the Y axis.
N200	puts the control in incremental mode (G91) and moves the Y axis the value in #22 then drills the next hole.
N210	adds one to the Y axis counter #32.
N220	END2 marks the end of the second WHILE DO loop. The control goes back to block N190 to see if the loop should run again. If the condition in N190 is false (#33 is no longer less than #25) the control will move on to block N230.
N230	resets the Y axis counter back to 1.
N240	adds one to the X axis counter #33.

N250 END1 marks the end of the first WHILE DO loop. The control goes back to block N160 to see if the loop should run again. If the condition in N160 is false (#32 is no longer less than or equal to #24) the control will move on to block N260.

N260 the program continues

NOTE: Air tapping (G184) will not work in this example due to its use of local variables.

In Example 2 the WHILE DO statements are nested as follows:

```
WHILE DO1
    STEP 1
        WHILE DO2
            STEP 2
        END2
   STEP 3
END1
```

Example 3: Now lets take the previous example (example 2) and put the WHILE DO section into a subprogram so it can be used more than one place in the program.

```
O0015 (MAIN PROGRAM)
N10 G90 G0 G54
N20 G49 Z0 H0
N30 G28 Z0 M8
N40 G60 W-10.
N50 G55 X1. Y1. T1 M6
N60 S1000 M3
N70 G43 Z.1 H1
N80 G99
N90 G81 R.1 Z-1. F10.
N100 #24 = 10. (NUMBER OF HOLES IN X AXIS)
N110 #21 = 1. (INCREMENT BETWEEN HOLES IN X AXIS)
N120 #25 = 5. (NUMBER OF HOLES IN Y AXIS)
N130 #22 = 1. (INCREMENT BETWEEN HOLES IN Y AXIS)
N140 M98 P0016
N150 G90 G0 G54
N160 G49 H0 Z0
N170 G28 Z0
N180 G55 X1. Y1. T2 M6
N190 S1000 M3
N200 G43 Z.1 H2
N210 G99
N220 G86 R.1 Z-1. F0.
N230 #24 = 10. (NUMBER OF HOLES IN X AXIS)
N240 #21 = 1. (INCREMENT BETWEEN HOLES IN X AXIS)
N250 #25 = 5. (NUMBER OF HOLES IN Y AXIS)
N260 #22 = 1. (INCREMENT BETWEEN HOLES IN Y AXIS)
N270 M98 P0016
N280 G90 G0 G54
N290 G49 H0 Z0
```

```
N300 G28 Z0
N310 M30

O0016 (GRID SUBPROGRAM)
N10 #32 = 1.
N20 #33 = 1.
N30 WHILE [#32 LE #24] DO1
N40 IF #32 EQ 1GOTO60
N50 G90 Y1. G91 X#21
N60 WHILE [#33 LT #25] DO2
N70 G91 Y#22
N80 #33 = #33 + 1
N90 END2
N100 #33 = 1.(RESET Y AXIS COUNTER)
N110 #32 = #32 + 1.
N120 END1
N120 M99
```

NOTE: Air tapping (G184) will not work in this example due to its use of local variables.

In this example (example 3) we took the WHILE DO loops and the counter initialization (#32 and #33) out of the part program and put them in a subprogram (O0016). The variables are set just before the call of the subprogram. In doing this the grid routine can be used many places in one or more programs.

This way of programming, setting variables and calling a subprogram can be even further simplified by using a custom macro call instead of a subprogram call. Refer to Chapter 11 for this program modified to work with a custom macro.

10.4 EXTERNAL DATA OUTPUT COMMANDS

The 11M control is capable of external data output of character strings (comments, text, numbers) and formatted numerical data (variables). This data is output via RS 232C interface to external devices, such as printers and tape punch units. With correct parameter settings and equipment, program generated data may be easily output to any external device.

Command words that control data output by programmed instruction are:

POPEN
DPRNT
BPRNT
PCLOS

10.4.1 Command Words

POPEN

This command opens the output process by connecting processing of output data to external IO units prior to executing data output commands.

A POPEN command should be specified prior to beginning a set or group of data output commands.

DPRNT

This command will output specified characters and numeric data in ISO code. Numeric values are assumed to consist of eight digit floating point decimal numbers.

The following formats apply:

DPRNT[argument]

DPRNT specifies decimal (ISO) code output.

[argument] specifies included characters and numeric characters.

A line feed carriage return is output by the data output command DPRNT[ ] after the last character in the included string is output.

Line length is determined by maximum line length of the target output device.

EXAMPLE 1: Output of Character String with DPRNT. The program would look like this. The * sign denotes a space code to be output.

```
DPRNT[This*is*a*example*of*report*generation*on* the*11M*control]
```

The output would look like this;

```
This is a example of report generation on the 11M control
```

EXAMPLE 2: Output of Character String and Variable Numerical Data

The program would look like this:

```
DPRNT[X#2[34]Y#5[34]]
DPRNT[BORE#2[34]]
```

Where:

X is a character

#N is a variable

[34] is a format specification

3 specifies number of digits to the left of the decimal.
4 specifies number of digits to the right of the decimal.

[ ] are required to delimit the expression.

The output would look like this.

```
X 1.0001Y 1.0003
BORE 1.0001
```

BPRNT

The BPRNT command will output specified characters on numerical data in binary code. This is not usual for most printer/output cases. SEE Fanuc's operation/programming manual for details.

PCLOS

This command closes the output process by releasing processing connection to external output units.

A PCLOS command shall be specified following the last data out put command.

NOTE: Without fail, specify POPEN and PCLOS as a pair within a given output sequence.

EXAMPLE:

```
POPEN[ ]
DPRNT[ ]
PCLOS
normal NC program
```

COMPOSITE EXAMPLE OF EXTERNAL DATA OUTPUT

(**NOTE**: Line numbers are for reference use only and are not part of the program)

```
 Line #
  1  POPEN
 2  DPRNT[ ]
 3  DPRNT[Part*SN*#9[40]***Month*#10[20]*day*#11[20]*year*#12[20]*hour*
 4  #13[20]* minute*#14[20]]
 5  DPRNT[ ]
 6  DPRNT[This*is*a*example*of*report*generation*on*the*11M*control]
 7  DPRNT[ ]
 8  DPRNT[X#2[34]Y#5[34]]
 9  DPRNT[Bore#2[34]]
 10 DPRNT[Bore#2[34]Bore#5[34]]
 11 DPRNT[This*space#2[34]This*space#5[34]]
 12 DPRNT[Bore*dia*X*axis*#2[34]*bore*dia*Y*axis*#5[34]]
 13 PCLOS
    M30
```

The output would look like this:

```
1
2
3  Part SN 9999  Month 2 Day 13 Year 86 Hour 11 minute
4
5
6  This is a example of report generation on the 11M control
7
8  X 1.0001Y 1.0003
9  Bore  1.0001
10  Bore  1.0001Bore  1.0003
11  This space  1.0001This space  1.0003
12  Bore dia X axis   1.0001 Bore dia Y axis   1.0003
13
```

10.4.2 Parameter Settings

For a more complete listing of parameter settings, refer to Monarch's Operator's Manual.