Anilam Electronics Corporation

ADVANCED VARIABLE (PARAMETRIC) PROGRAMMING

ADVANCED VARIABLE (PARAMETRIC) PROGRAMMING or...

How to Write Your Own Custom Canned Cycles!

It is possible to use the Crusader to develop custom canned cycles that are applicable to your specific needs. This section will show you how.

The Crusader control, with Advanced Software System I, has the ability to store, calculate and act upon the results of variables. This section of the manual is to be used as a guideline to help explain the more advanced capabilities of the Crusader. Several programming examples using variables are given.

Do not attempt the advanced variable programming methods explained here until you are thoroughly familiar with all aspects of the Crusader already discussed. Anilam recommends that customers should try advanced variable programming only after using the Crusader control for at least one year.

TABLE OF CONTENTS

SECTION 1:
TOOL OFFSETS USING VARIABLES

When cutter diameter compensation is used to rough and finish a part by using the same tool with two or more diameters (or tool tip radius) offsets, the program is structured as follows:

MAIN PROGRAM	SUBROUTINE
1. T1001	100. Sub 1
2. X.520 Z-1.345	101. F10.
3. T 0	101. Z-.2 FA
4. Z0 RA	103. G41
5. X0 Y0 RA	104. Y1. FA
6. T1	105. X-1. FA
7. Z.1 RA	106. Y-1. FA
8. CALL 1	107. X1. FA
9. T1001	108. Y1.0 FA
10. X.510 Z-1.345	109. XO FA
11. T1	110. G40
12. CALL 1	111. YO
13. T1001	112. END
14. X.500 Z-1.345	
15. T1	
16. CALL 1	
17. T0	
18. Z0 RA	
19. X0 Y0 RA	
20. END	

* (T1001 is an offset definition block for diameter and/or length compensation. All examples can be used on lathe or mill Crusaders.)

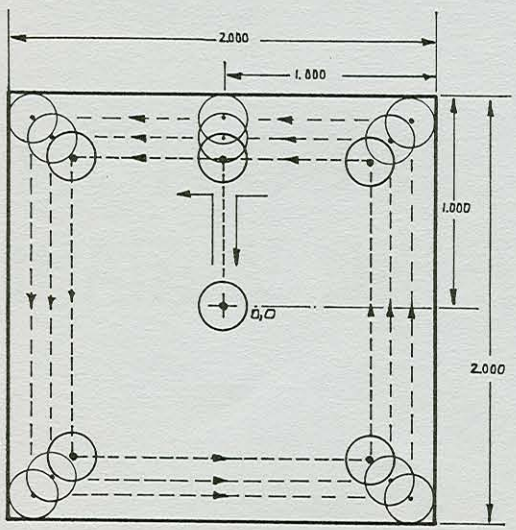


FIGURE 1-1

The 2 inch square window would be machined — first leaving .010 stock, then .005 stock, then making a finish cut using the program.

The program will work as written, but the operator will have to enter the Z axis tool length offset in events 2, 10, and 14. In a longer program with several tools, this procedure can be clumsy and can lead to operator errors.

A more convenient and safer way to program this type of operation would be to enter a variable which is equal to the tool length offset and, each time the offset is needed, state the variable. For example:

MAIN PROGRAM	SUBROUTINE
1. V80 = -1.345	100. Sub 1 (Subroutine remains the same as previous program)
2. T1001	112. End
3. X.520 Z V80	
4. TO	
5. Z0 RA	
6. X0 Y0 RA	
7. T1	
8. Z.1 RA	
9. CALL 1	
10. T1001	
11. X.510 Z V80	
12. T1	
13. CALL 1	
14. T1001	
15. X.500 Z V80	
16. T1	
17. CALL 1	
18. T0	
19. Z0 RA	
20. X0 Y0 RA	
21. END	

In this example Event 1 sets V80 equal to the tool length offset. In the places where the tool length offset value is needed, V80 is used and the control will use the value of V80 (-1.345) as the tool length offset. This eliminates the operator having to enter the tool length offset into the program in several places. This is especially desirable if the job is a repeat order and the tool length is different or if a new tool with a new offset is used. The new variable only needs to be entered once and the tool length offset is effectively changed 3 times in the program.

To enter events 3, 11, and 15 press: X.500 ZV80 EVENT ENTER. When Z is pressed, the Z light will come on and zeros will be displayed in the Z axis. When V is pressed, the V light will come on and only 2 digits in the Z axis will stay on indicating a 2 digit variable is to be entered. As 80 is pressed, it will automatically be entered in Z (which is the selected axis).

Use V80 through V94 when working with this feature, as these are not presently used by built-in canned cycles, which could alter their value.

The value of a variable can be looked at any time by pressing SINGLE STEP, DISPLAY POSITION (turn it off), V80 (or the variable number which you want to see).

SECTION 2a:
ZERO SHIFT (AUX 1101) USING VARIABLES

When a multiple zero shift feature is used with several tools, then program is constructed as follows:

MAIN PROGRAM

```
1.  T1001
2.  Z-1.345
3.  T1002
4.  Z-2.345
5.  T1003
6.  Z-3.345
7.  A1101
8.  X0 Y0 Z0
9.  T0
10. Z0 RA
11. X0 Y0 RA
12. T1
13. CALL 1
14. A1101
15. X2. Y0 Z-1.345
16. CALL1
17. A1101
18. X4. Y0 Z-1.345
19. CALL1
20. A1101
21. X6. Y0 Z-1.345
22. CALL1
23. A1101
24. X8. Y0 Z-1.345
25. CALL1
26. A1101
27. X8. Y0 Z-1.345
28. CALL1
29. T0
30. Z0 RA
31. X0 Y0 RA
32. T2
33. CALL2
34. A1101
35. X2. Y0 Z-2.345
36. CALL2
37. A1101
38. X4. Y0 Z-2.345
39. CALL2
40. A1101
41. X6. Y0 Z-2.345
```

MAIN PROGRAM

```
42. CALL2
43. A1101
44. X8. Y0 Z-2.345
45. CALL2
46. T0
47. Z0 RA
48. X0 Y0 RA
49. T3
50. CALL3
51. A1101
52. X2. Y0 Z-3.345
53. CALL3
54. A1101
55. X4. Y0 Z-3.345
56. CALL3
57. A1101
58. X6. Y0 Z-3.345
59. CALL3
60. A1101
61. X8. Y0 Z-3.345
62. CALL3
63. T0
64. Z0 RA
65. X0 Y0 RA
66. END
```

SUBROUTINES

SUB 1: END (Motion to cut part with Tool 1)

SUB 2: END (Motion to cut part with Tool 2

SUB 3: END (Motion to cut part with Tool 3)

The above program will produce the parts shown. The total length offsets must be entered in several places in the program for use each time the zero shift featured is activated.

A more efficient program can be written using variables. Two examples are given below:

MAIN PROGRAM

```
1.  V81 = –1.345
2.  V82 = –2.345
3.  V83 = –3.345
4.  T1001
5.  Z V81
6.  T1002
7.  Z V82
8.  T1003
9.  Z V83
10. A1101
11. X0 Y0 Z0
12. T0
13. Z0 RA
14. X0 Y0 RA
15. V0 = 80
16. A5041
17. T1
18. CALL1
19. CALL4
20. CALL1
21. CALL5
22. CALL1
23. CALL6
24. CAALL1
25. CALL7
26. CALL1
27. T0
28. Z0 RA
29. X0 Y0 RA
30. V0 = 80
31. A5042
32. T2
33. CALL2
34. CALL4
35. CALL2
36. CALL5
37. CALL2
38. CALL6
39. CALL2
40. CALL7
```

MAIN PROGRAM

```
41. CALL2
42. T0
43. Z0 RA
44. X0 Y0 RA
45. V0 = 80
46. A5043
47. T3
48. CALL3
49. CALL4
50. CALL3
51. CALL5
52. CALL3
53. CALL6
54. CALL3
55. CALL7
56. CALL3
57. T0
58. Z0 RA
59. X0 Y0 RA
60. END
```

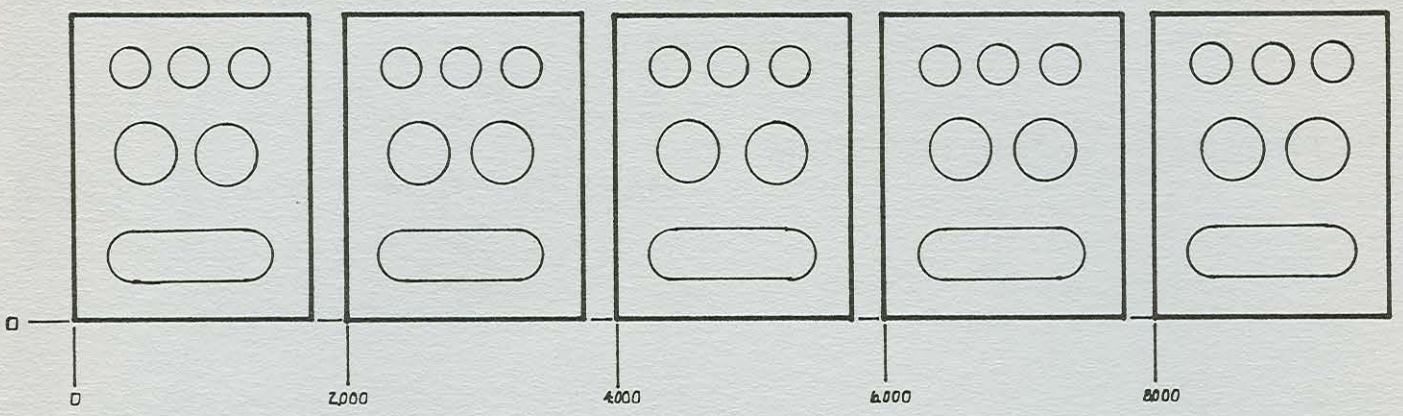
SUBROUTINES

```
400. SUB 4
401. A1101
402. X2. Y0 Z V84
403. END

500. SUB 5
501. A1101
502. X4. Y0 Z V84
503. END

600. SUB 6
601. A1101
602. X6. Y0 Z V84
603. END

700. SUB 7
701. A1101
702. X8. Y0 Z V84
703. END
```



In this example, variables 81-83 are set to the tool length offsets of tools 1-3 respectively. Subroutines 1-3 remain the same as the previous program (tool motion for tools 1-3).

The zero shifts (to move the zero to each part) are put in subroutines with the Z axis equal to V84. Before each tool is called, the appropriate variable for that tool's offset is transferred to V84. For example, in event 15, V0 is set equal to 80. This is so we can perform operations on the 80 decade of variables (V80-V89).

Event 16 is AUX 5041. This is a special AUX code which does a direct transfer of a variable's value. There are several special AUX codes (discussed later) which are used in conjunction with variables. These AUX codes all have 4 digits. The first two digits determine the type of operation to be performed. In the case of AUX 5041, the digits 50 indicate the operation is a direct transfer. The last two digits (41) represent the variables on which the operation is being performed. Since V0 equals 80, the variables being used are **84** and **81** (from **5041**). The direct transfer is made from variable 81 into variable 84 or V84 is made equal to the value of V81. In this case V81 equals – 1.345. After the AUX 5041 is executed, V84 will equal – 1.345. Before Tool 2 is activated, an AUX 5042 is programmed. Similarly, this sets V84 as equal to the value of V82 (which is tool 2's offset). After this AUX code is executed, V84 will equal – 2.345. Before Tool 3 is activated AUX 5043 makes V84 equal to – 3.345. By using this method, the same subroutine can be used to shift zero for all 3 tools and if an offset is changed, only one event will need to be edited.

SECTION 2b:
CALLING SUBROUTINES USING VARIABLES

The next example will use a new AUX code to further shorten the same program. Events 17-30 and 32-45 and 47-59 are similar. Instead of programming CALL 1, CALL 2, CALL 3, and TOOL 1, TOOL 2, TOOL 3, a variable will be used instead of the numbers 1, 2, 3. This is so a subroutine can be created.

MAIN PROGRAM

1. V81 = −1.345
2. V82 = −2.345
3. V83 = −3.345
4. T1001
5. Z V81
6. T1002
7. Z V82
8. T1003
9. Z V83
10. A1101
11. X0 Y0 Z0
12. T0
13. Z0 RA
14. X0 Y0 RA
15. V0 = 80
16. A5141
17. V85 = .0001
18. CALL8
19. A5751 (V85 = .0002)
20. A5142 (V84 = V82)
21. CALL8
22. A5751
23. A5143
24. CALL8
25. END

SUBROUTINE

SUB 1—SUB 7 remains the same.

800. SUB 8
801. TOOL V85
802. CALL V85
803. CALL4
804. CALL V85
805. CALL5
806. CALL V85
807. CALL6
808. CALL V85
809. CALL7
810. CALL V85
811. T0
812. Z0 RA
813. X0 Y0 RA
814. END

Up through event 16 the program is the same. Event 17 sets the Variable 85 equal to .0001 (internally the decimal point is not seen so V85 = 001). When SUB 8 is called, V85 equals 1 so Tool V85 means Tool 1 and Call V85 means Call 1.

Then a new AUX code (5751) is programmed. The first 2 digits (57) mean "add to a variable". The last two digits (**51**) mean "add to **V85** the number **1** (**51**)". Since V85 equals 1, now V85 equals 1 + 1 = 2. Any digit (1-9) can be added to a variable in this fashion. Since V85 now equals 2, Tool V85 will mean Tool 2 and Call V85 will mean Call 2. At Event 22, Aux 5751 will mean V85 equals V85 + 1.

Since V85 equals 2, after this AUX code is executed, V85 will equal 2 + 1 = 3. Now Tool V85 will mean Tool 3 and Call V85 will mean Call 3.

SECTION 3:
AUX CODE DEFINITIONS FOR MANIPULATING VARIABLES

Listed below are AUX codes which perform mathematical operations on variables. Examples are given for each AUX code.

Aux 50—: Direct Transfer

This AUX code transfers the value of one variable directly to another variable. For example:

1. V0 = 80
2. V81 = – .437
3. A5061

Step 1 makes V0 equal to 80 so that functions will be performed on the 80 decade of variables (V80—V89). Step 2 makes V81 equal to – .437 so we can say the value of V81 is – .437. Step 3 transfers the value of V81 (5061) into V86 (506l).

After executing this AUX code, V86 will also equal – .437. The first two digits of the AUX code (5061) define the operation to take place. This is called a direct transfer operation. The third and fourth digits (61) represent the two variables which are being acted on. This AUX code 5061 should be understood as "make V86 equal to the value of V81".

Another example would be:

1. V0 = 30
2. V33 = 16.25
3. AUX 5073

In this example, V37 is made equal to the value of V33. After this AUX code is executed, V37 will equal 16.25.

AUX 51—: Indirect Transfer

This AUX code enables the user to transfer the value of a variable into a variable which has the value of another variable. For example:

1. V0 = 80
2. V81 = .0037
3. V37 = -1.150
4. AUX 5121

In this example, V81 is set to the number 37. Variable 37 is equal to – 1.150.

The AUX code 5121 means Variable 82 (5121) is now equal to the value of the variable in V81 (5121). The value of V81 is 37. The value of V37 is – 1.150. So after this AUX code is performed, V82 will equal – 1.150. This enables the user to transfer the value of a variable from one decade to another. In this example we transferred the values of V37 to V82. Another example is as follows:

1. V0 = 60
2. V66 = 0015
3. V15 = .220
4. A5176

This AUX code means "make V67 (5176) equal to the value of the variable in V66 (5176)". Since V66 equals 15 and the value of V15 equals .220, after this AUX code is executed V67 will equal .220.

AUX 52—: Indirect Transfer

This AUX code works similar to AUX 51— except the transfer is made from the fourth digit into the variable which is the value of the third digit. Example:

1. V27 = .125
2. V23 = 88
3. V0 = 20
4. A5237

In this example, V23 is set equal to the number 88. The AUX code 5237 means "make the variable which is the value of V23 (5237) equal to the value of V27 (5237)" or "V88 (which is the value of V23) equal to the value of V27" (or V88 = .125). Another example:

1. V42 = 0016
2. V43 = 2.625
3. V0 = 40
4. A5223

This program means "make the variable whose value is in V42 equal to the value of V43" (or make V16 = 2.625).

AUX 53—: Addition

This AUX code simply means "add the value of the variable in the third digit to the value of the variable in the forth digit and store the sum in the variable of the third digit". For example:

1. V86 = .375
2. V87 = .125
3. V0 = 80
4. A5367

This would perform the addition of V86 and V87 and store the results in V86. This means V86 equals V86 + V87 or V86 = .500.

Another example would be:

1. V62 = – 2.5
2. V63 = 7.0
3. V0 = 60
4. A5323

In this example, after the AUX code is executed, V62 will equal (– 2.5) + (7.0) = 4.5.

In the next example we will use several AUX codes to perform a desired operation. Let's say we want to add the value of V68 and V83 and store the results in V79. The program could be written:

1. V68 = 6.
2. V83 = 5.
3. V84 = 0068
4. V0 = 80
5. A5154 (V85 is equal to the variable whose value is in V84; V85 equals V68 = 6.000)
6. A5353 (V85 equals V85 + V83 = 6 + 5 = 11)
7. V84 = 0079
8. A5245 (Variable whose value is in V84 is equal to V85, V79 = 11)

AUX 54—: Subtraction

This AUX code subtracts the value of the variable in the fourth digit from the value of the variable in the third digit. Example:

1. V0 = 80
2. V84 = 66
3. V83 = 30.
4. A5443

After this AUX code is executed, C84 will equal 36. (AUX 5443 means V84 is equal to V84 – V83 or V84 = 66 – 30 = 36). Another example is:

1. V0 = 10
2. V11 = – 6.
3. V12 = – .5
4. AUX 5412

This program will make V11 (5412) equal to V11 – V12; or V11 equals (– 6.) – (– .5) = – 6.5.

AUX 55—: MULTIPLICATION

This AUX code multiplies the value of the variables in the third and fourth digit and stores it as the value of the variable of the third digit. Example:

1. VO = 80
2. V81 = 2.562
3. V82 = .0003
4. A 5512

This program will make V81 equal to V81 times V82, or V81 = 7.686. Caution: the control does **not** recognize the decimal point. Multiplication or division can cause a digit over flow. In the example above the control multiplies the numbers 25620 × 3 = 76860.

AUX 56—: DIVISION

This AUX code divides the value of the variable in the third digit by the value of the variable in the fourth digit and stores the result as the value of the third digit. Example:

1. VO = 80
2. V85 = 3.
3. V86 = 0002
4. A5656

After this program is executed V85 will be = to 1.500. **Caution**: the control does integer division only. The decimal point is ignored and if there is a remainder the quotient is **truncated** and **not rounded**. For example if you divide .0008 by .0003 the result will be .0002 because the answer .0002666 is truncated and not rounded.

AUX 57—: ADD IMMEDIATE

This AUX code will add the number in the fourth digit to the value of the variable in the third digit. Example:

1. VO = 80
2. V81 = 0
3. DO 5
4. Aux 5713
5. End

After this program is executed V81 will equal 0015 because the digit 3 has been added to the value of V81 5 times. V81 equals 0 + 3 + 3 + 3 + 3 + 3 = 15.

This AUX code can be used to keep a counter on the number of times a loop has been completed. For example:

1. V80 = 0
2. V82 = 0
3. DO 85
4. Call 3
5. Aux 5721
6. End

While this program is running the value of V82 can be viewed by pressing display position (turn it off), then press V82 and its value will be displayed.

AUX 58—: SUBTRACT IMMEDIATE

This AUX code works similar to AUX 57— except the number in the fourth digit will be subtracted from the value of the variable in the third digit. Example:

1. VO = 10
2. V12 = 0020
3. A5821

After this program is executed, V12 will equal 0019. Another example:

1. VO = 30
2. V35 = – 0018
3. A5856

After this program is executed, V35 will equal – 24 or (– 18) – 6 = – 24.

AUX 59—: NEGATE

This AUX function multiplies the value of the variable in the fourth digit by – 1 and stores it as the value for the variable in the third digit. Example:

1. VO = 80
2. V85 = 2.375
3. A5965

After this program is executed, V86 will equal – 2.375 and V85 will remain equal to 2.375. Another example:

1. VO = 80
2. V89 = – .0625
3. A5999

After this program is executed, V89 will equal + .0625 or – .0625 × – 1 = + .0625).

SECTION 4: LOGIC TESTING PERFORMED ON VARIABLES

Six different tests can be performed on the contents of a variable and, depending on if the test is true or false, another variable is set equal to 1 or 0 respectively. This is usually used as a conditional test to do something or not depending on the value of a variable. Examples are given after each of the six tests are defined.

AUX 60AB — CHECK IF B = 0

This AUX code will check the value of Variable B and if it is equal to 0, this code will make variable A equal to 1. If Variable B does not equal 0, then Variable A will be set equal to 0. Example:

1. VO = 80
2. V81 = 0006
3. V82 = 0002
4. A5412
5. A6031

Event 4 will subtract V82 from V81 and store the results in V81. Event 5 will check the value of V81 and if it is equal to zero, V83 will be set equal to 1. Since V81 – V82 will equal 4, V81 will not be equal to 0 so V83 will be set equal to 0. Another example:

1. VO = 80
2. V82 = 6
3. V83 = 0
4. A5523
5. A6042

Event 4 will multiply V82 × V83 and store the results in V82. Event 5 checks the value of V82 and if it equals 0, V84 will be set equal to 1. Since V82 × V83 = 0, V82 will equal 0 and V84 will be set equal to 1.

AUX 61AB — CHECK IF B Not = 0

This AUX code checks the value of Variable B and if it does not equal 0, Variable A is set equal to 1. If Variable B does equal 0, Variable A will be set equal to 0. Example:

1. VO = 80
2. V88 = .500
3. V87 = .251
4. A5487
5. A5487
6. A6168

Event 4 will subtract V87 from V88 and store the results in V88. Event 5 will again subtract V87 from the new value of V88 and store the results in V88. Event 6 will check the value of V88 and since V88 will not equal 0, V86 will be set equal to 1.

AUX 62AB — CHECK IF B > 0

This AUX code checks the value of Variable B and if it is greater than 0, Variable A is set equal to 1. If Variable B is less than or equal to 0, Variable A will be set equal to 0. Example:

1. VO = 80
2. V83 = – .375
3. A5933
4. A6243

Event 3 will negate (multiply by – 1) the value of V83 and then store the result in V83. Event 4 checks the contents of V83 and if it is greater than 0, V84 will be set equal to 1. Since – 1 × – .375 = .375 (which is greater than 0), V84 will be set equal to 1. If the results were equal to zero, or less than zero, V84 would be set equal to zero.

The remaining AUX codes for logic testing all work similarly to these examples. Their definitions are:

AUX 63AB — CHECK IF B < 0

If the value of Variable B is less than 0, Variable A is set equal to 1, otherwise Variable A is set equal to 0.

AUX 64AB — CHECK IF B > = 0

If the value of Variable B is greater than or equal to 0, Variable A is set equal to 1, otherwise Variable A is set equal to 0.

AUX 65AB — CHECK IF B < = 0

If the value of Variable B is less than or equal to 0, Variable A is set equal to 1, otherwise Variable A is set equal to 0.

SECTION 5:
CONDITIONAL LOOPS AND BRANCHING (GO TO STATEMENTS)

The Crusader has the ability to execute or not execute sections of a program dependent on the value of a variable. The example used is one where patterns of holes will be center drilled, however, some of the holes will be drilled and some will be counter bored.

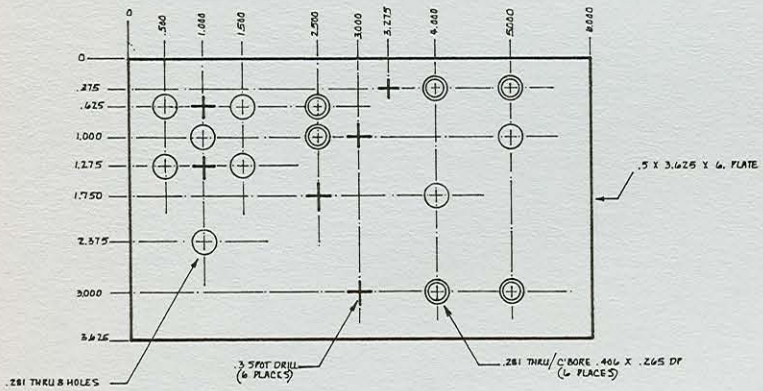


FIGURE 3

Instead of making several subroutines and calling each one separately when needed, one subroutine can be called and variables set to make the program branch differently for different tools. The program would be constructed as follows:

TOOL 1 = SPOT DRILL
TOOL 2 = .281 DRILL
TOOL 3 = .406 END MILL

MAIN PROGRAM

1. T1001
2. Z – 1.234
3. T1002
4. Z – .345
5. T1003
6. Z – 2.345
7. TO
8. ZO RA
9. X – 6. YO RA
10. T1
11. V81 = .0001
12. V82 = – .15
13. V83 = .0001
14. V20 = 15.
15. V21 .1
16. G81
17. Call 1
18. T2
19. V82 = – .6
20. V83 = 0
21. V20 10.
22. V21 .1
23. V23 .2
24. G83
25. Call 1
26. T3
27. V81 = 0
28. V82 = – .265
29. V20 6.
30. V21 .1
31. V22 1.
32. G82
33. Call 1
34. End

SUBROUTINE

100. Sub 1
101. Do V81
102. X.5Y – .625 Z V82RA
103. Y – 1.275 RA
104. X1.OY – 2.375 RA
105. X1.50Y – 1.275 RA
106. Y – .625 RA
107. X1.0 Y – 1.0 RA
108. DO V83
109. Y – .625 RA
110. Y – 1.275 RA
111. X 2.5Y – 1.75 RA
112. End
113. End
114. X 2.5Y – .625 ZV82RA
115. Y – 1.0 RA
116. DO V83
117. X3.Y – 1.0 RA
118. Y – 3.0 RA
119. X 3.275Y – .375RA
120. End.
121. Do V81
122. X 4.Y – 1.75 RA
123. Y 5.0 Y – 1.0 RA
124. End
125. X5.0 Y – .375 RA
126. X4. RA
127. Y – 3. RA
128. X5.0 RA
129. G80
130. TO
131. ZO RA
132. X – 6.YO RA
133. End

Subroutine 1 contains the X Y coordinates of all of the spot drill and hole locations. The spot drill locations are programmed inside a do loop that starts with DO V83. If V83 is set equal to 1 then the do loop will be executed once. If V83 is set equal to 0 then the do loop will be skipped and not executed.

Any portion of a program can be skipped by preceeding it with DO 0 and putting an END after the last event you want to skip. If a subroutine call is surrounded by a variable controlled do loop, then the subroutine can be skipped or called dependent on the value of the variable.

Note that after tool 1 spot drills all of the holes, V83 is set equal to 0 so that when Subroutine 1 is called, the Do loops that contain the locations for locations that are not to be drilled are skipped. After tool 2 is used, V83 is also set equal to 0 so that only the coordinates outside the conditional do loops are executed with tool 3. The Z depth is also controlled using V82 so that one subroutine can spot, drill, and counterbore.

SECTION 6:
USER PROGRAMMABLE MODAL SUBROUTINES (CUSTOM MODAL CANNED CYCLES)

If a canned cycle does not exist which you have an application for, a custom canned cycle can be written. If it is to be non-modal then a subroutine would be created and called when desired in the program. If it is to be modal, special programming techniques must be used.

As an example, the holes in the plate below will be bored with a special cycle which will rapid into the counter bore, feed through at 4 IPM and feed out at 8 IPM.

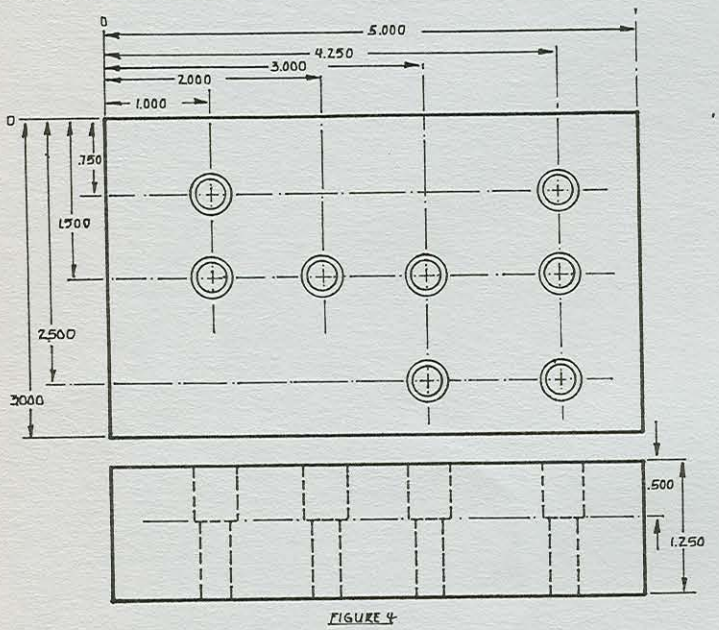


FIGURE 4

MAIN PROGRAM	SUBROUTINE
1. T1001	100. Sub 9088
2. Z – 1.234	101. Aux 9088
3. TO	102. End
4. Z0 RA	103. Sub 9588
5. X – 5. YO RA	104. XV27 YV28 RA
6. T1	105. Z – .4 RA
7. G88	106. F 4.
8. X1. Y – .75 RA	107. Z – 1.3 FA
9. Y – 1.5 RA	108. F 8.
10. X2. RA	109. Z – .4 FA
11. X3. RA	110. Z .1RA
12. Y – 2.5 RA	111. End
13. X4.25 RA	
14. Y – 1.5 RA	
15. Y – .75 RA	
16. G80	
17. TO	
18. ZO RA	
19. X – 5. YO RA	
20. End	

In this example the holes have been drilled and counterbored and the boring operation is all that is required. Event 7 activates G88. This is a user definable modal G code. There is no G88 code in the Crusader's internal software. A user programmable G88 code can be constructed as follows:

When a G88 event is read by the control in event 7, the control will react as if a call 9088 had been programmed. The programmer creates subroutine 9088 which contains the instruction AUX 9088. This special AUX code will make the control look ahead in the program for the next X Y Z command and store the X command in V27, the Y command in V28 and the Z command in V29. If the commands are incremental, they will be converted and stored in absolute.

This AUX code then calls subroutine 9588 which the programmer creates and which contains the motion desired in the canned cycle. Notice that the event 104 is: X, V27, YV28, RA. This will cause motion to the XY position that the control looked ahead to find. This sequence of looking ahead for the next XY coordinate, storing it in V27 and V28, then executing subroutine 9584 is repeated automatically until a G80 is reached in the program.

This modal subroutine programming can be used with features like Rotation, Scaling, and Bolt Circles.

This concludes the Advanced Variable Programming section. If you have any further questions or require more expert assistance, please feel free to contact our Applications Engineering department at Anilam Electronics Corp., 5625 N.W. 79th Ave., Miami, Florida 33166. (305)592-2727.

SHEET _____ OF _____

ANILAM CRUSADER PROGRAM SHEET

RT OR DRWG. # _____ PROGRAMMER _____ DATE _____

TOOL LIST	OFFSETS	SETUP INSTRUCTIONS

EVENT NO.				RAPID /FEED	INCR /ABS	EVENT NO.				RAPID /FEED	INC /AB
1						1					
2						2					
3						3					
4						4					
5						5					
6						6					
7						7					
8						8					
9						9					
0						0					
1						1					
2						2					
3						3					
4.						4.					
5						5					
6						6					
7						7					
8						8					
9						9					
0						0					