

## How do I send a set point change to a slave instrument using a recipe on a Dualpro or Multipro?

---

---

There are two ways to do this. You re-direct the recipe set point (H op code) to any attached slave by selecting the PROG(ram) menu and scrolling down to the H OP (redirect) parameter. If this parameter is set to NORM(al) the temperature set point is sent to the loop in the controller that is configured to control temperature. If this parameter is set to DIR(ected), the set point is sent to the temperature slave that is the is configured in the next two parameters.

The next two parameters in the PROG menu following the H OP parameter re-direct the temperature set point if H OP is set to DIR(ected). The first parameter is REDIR CHAN or H CH. This is the address of a slave Dualpro or Multipro that is connected to the controlling master unit running the recipe. If the slave is connected to the master instrument this parameter is set to 0. The next parameter is the REDIR SLAVE # or H SL. This is the address of the attached temperature slave. The valid address range is H1 to H16. This setting corresponds to the address of the slave instrument that should receive the set point.

It is important to remember that if the H opcode temperature set point is re-directed to a slave, then any loop of the Dualpro or Multipro that is configured to control temperature cannot be changed through the recipe.

If there is a requirement to change a set point of a slave and the set point of the master instrument's temperature loop through a recipe then it necessary to use the second method of control. Set the H OP parameter to NORM and load the following recipe and background (logic) program. These programs are samples that demonstrate the use of program flags in the Version 3.5 recipe language. If you want to use this feature it will probably be necessary to modify this code to fit your application.

The following sample recipe uses the Z opcode and F opcode of the recipe language to send set points to different temperature slaves. This is done by storing the desired set point in the Z register and then setting a flag that corresponds to the slave address that should accept the set point. The slave address and corresponding flags are shown in the following table:

Slave Name	Slave Address	Flag
Slave 1	1	16
Slave 2	2	17
Slave 3	3	18
Slave 4	4	19
Slave 5	5	20
Slave 6	6	21
Slave 7	7	22
Slave 8	8	23

## Flag Recipe Program

The sample recipe shown below sets two set points. The temperature set point of the Dualpro/Multipro loop is set first and then the slave temperature is set. This condition would be similar to a batch furnace application, where the Dualpro is controlling the furnace temperature and a temperature slave is controlling the quench temperature.

First we set the furnace temperature on the Dualpro with the H opcode. Then we store the set point of the quench temperature with the Z opcode. When Flag 16 (Slave 1) is set the flag is sensed by a background (logic) program that is running on the Dualpro. Once this flag has been seen, the background program takes the value in the Z register and sends it to the correct slave. The background program then sets a flag that tells the recipe that the change has occurred. This condition is sensed by the W (wait) opcode in the recipe. Following the W opcode is a L (limit) statement. If this limit is exceeded then a PAL 93 "LIMIT TIME OUT" alarm will appear on the display of the Dualpro.

The sample recipe changes the set point of the Dualpro and Slave 1 twice, 1000 and 100 and then 1100 and 200 respectively. The program then loops to start over. The following sample shows the program line number of each recipe command, the recipe opcode and data, and a comment for each line.

## Sample Recipe

1	H,1000	Change temp of Dualpro loop
2	Z,0100	Change temp of 10pro slave
3	F,016.1	Set Slave 1 setpoint (Flag 16 ON)
4	W,016.1	Wait for confirm (Flag 16 ON)
5	L,00:01	Limit 1 min
6	i,0010	Delay to view temp changes
7	H,1100	Change temp of Dualpro loop
8	Z,0200	Change temp of 10pro slave
9	F,016.1	Set Slave 1 setpoint (Flag 16 ON)
10	W,016.1	Wait for confirm (Flag 16 ON)
11	L,00:01	Limit 1 min
12	i,0010	Delay to view temp changes
13	J,0000	Jump back to beginning of recipe

## Background Program

The following background program listing is used in conjunction with Process Master Control's PM7 PCOMPILE program. This program allows the user to use a program text file to generate and download the background program to a Dualpro or Multipro.

The first part of the program assigns names to the various memory registers in the Dualpro that are used in conjunction with the recipe flags. The comments in the program explain which registers are assigned to each flag. These assignments are fixed in the instrument firmware and should not be changed.

The actual program starts with the .ORG 1 statement. This assigns the starting location of the program as Background program 1. The program initializes parameters in the instrument and then starts a loop that continually looks for the recipe flags. If a flag is set

the program branches to that flag handling routine, sends the set point data to the correct slave, sets the response flag, and then resets all flags.

Since this program handles the flags for eight temperature slaves, most of the listing includes duplicate routines that manage the flags for the other slave flags. Following this listing is a listing of the actual background programs that are generated by the Source Code. If PCOMPILE is not available, the actual program listing can be used to enter these programs directly into the instruments background program space.

### Background Program Source Code

```
.DEFINE MAS_TBL          x0000  ;MASTER DATA TABLE
.DEFINE (_FG_FLAGS_1)    (A8)   ;FLAGS SET WORD #1 (FLAGS 0-15) MASTER TABLE
.DEFINE (_FG_FLAGS_2)    (#AD)  ;FLAGS SET WORD #2 (FLAGS 16-31)
.DEFINE (_FG_FLAGS_3)    (#AE)  ;FLAGS SET WORD #3 (FLAGS 32-47)
;BG FLAG WORD DEFINITIONS
.DEFINE (_BG_FLAGS_1)    (B2)   ;FLAGS WAIT WORD #1 (FLAGS 0-15) MASTER TABLE
.DEFINE (_BG_FLAGS_2)    (#B7)  ;FLAGS WAIT WORD #2 (FLAGS 16-31)
.DEFINE (_BG_FLAGS_3)    (#B8)  ;FLAGS WAIT WORD #3 (FLAGS 32-47)

.DEFINE (OPTIONS)        (E0)   ;[62] SELECTABLE OPTIONS
;bit 0 =
;bit 1 =
;bit 2 =
;bit 3 =
;bit 4 =
;bit 5 =
;bit 6 =
;bit 7 =
;bit 8 =
;bit 9 =
;bit A =
;bit B =
;bit C =
;bit D =
;bit E =
;bit F = HELPER BIT FOR FLAG 16 PROCESSING
```

```
*****
```

### ;MASTER TABLE PARAMETERS

```
*****
```

```
.DEFINE (SLAVE1_SP)      (40)
.DEFINE (SLAVE2_SP)      (41)
.DEFINE (SLAVE3_SP)      (42)
.DEFINE (SLAVE4_SP)      (43)
.DEFINE (SLAVE5_SP)      (44)
.DEFINE (SLAVE6_SP)      (45)
.DEFINE (SLAVE7_SP)      (46)
```

```
.DEFINE (SLAVE8_SP) (47)
.DEFINE (SLAVE1_ACK) (48)
.DEFINE (SLAVE2_ACK) (49)
.DEFINE (SLAVE3_ACK) (4A)
.DEFINE (SLAVE4_ACK) (4B)
.DEFINE (SLAVE5_ACK) (4C)
.DEFINE (SLAVE6_ACK) (4D)
.DEFINE (SLAVE7_ACK) (4E)
.DEFINE (SLAVE8_ACK) (4F)
```

```
.INCLUDE SYSUTL.INC ;std MMI system
```

```
*****
;START OF CODE
```

```
*****
.ORG 1
@BG_START
ST (C3),0 ;clear BG program stack
ST (C7),0 ;clear DATA stack
GS UN @_SYS_INIT
@MAIN_LOOP
GS UN @_V35_RECIPES ;process v3.5 foreground recipes
BR UN @MAIN_LOOP
```

```
*****
@_V35_RECIPES
```

```
*****
;FLAG_16 SET SLAVE 1 SETPOINT
;FLAG_17 SET SLAVE 2 SETPOINT
;FLAG_18 SET SLAVE 3 SETPOINT
;FLAG_19 SET SLAVE 4 SETPOINT
;FLAG_20 SET SLAVE 5 SETPOINT
;FLAG_21 SET SLAVE 6 SETPOINT
;FLAG_22 SET SLAVE 7 SETPOINT
;FLAG_23 SET SLAVE 8 SETPOINT
*****
SE (_FG_FLAGS_1),15 ;KEEP FLAG 11 ON ALWAYS
ST (ZF),(_FG_FLAGS_1) ;SO Z CODE WORKS
@CHK_FLAGS
BI (_FG_FLAGS_2),0
GN ZE @DO_FLAG_16
BI (_FG_FLAGS_2),1
GN ZE @DO_FLAG_17
BI (_FG_FLAGS_2),2
GN ZE @DO_FLAG_18
BI (_FG_FLAGS_2),3
GN ZE @DO_FLAG_19
BI (_FG_FLAGS_2),4
```

```

GN ZE @DO_FLAG_20
BI (_FG_FLAGS_2),5
GN ZE @DO_FLAG_21
BI (_FG_FLAGS_2),6
GN ZE @DO_FLAG_22
BI (_FG_FLAGS_2),7
GN ZE @DO_FLAG_23
RET
@DO_FLAG_16
; SET THE SLAVE 1 SETPOINT
ST (Z2),MAS_TBL ;MASTER TABLE x0000
AD (Z2),&(SLAVE1_SP) ;SLAVE OFFSET
WR (Z2),(Z1) ;PASS SETPOINT TO SLAVE WITH Z1
ST (Z2),MAS_TBL ;MASTER TABLE x0000
AD (Z2),&(SLAVE1_ACK) ;SLAVE OFFSET
RD (Z3),(Z2) ;PUT ACTUAL INTO Z3
CP (Z1),(Z3) ;MAKE SURE CHANGE HAPPENED
BN ZE @FINISH_FLAGS ;NOT CHANGED EXIT TO TRY AGAIN
@FLAG_16_END
ST (Z1),0 ;RESET Z1
SE (_BG_FLAGS_2),0 ;SET BG FLAG TO 0 FOR RECIPE WAIT
DL UN 2 ;HOLD RESET
RE (_FG_FLAGS_2),0 ;RESET FG FLAGS BIT 0
RE (_BG_FLAGS_2),0 ;RESET BG FLAGS BIT 0
RET
@DO_FLAG_17
; SET THE SLAVE 2 SETPOINT
ST (Z2),MAS_TBL ;MASTER TABLE x0000
AD (Z2),&(SLAVE2_SP) ;SLAVE OFFSET
WR (Z2),(Z1) ;PASS SETPOINT TO SLAVE WITH Z1
ST (Z2),MAS_TBL ;MASTER TABLE x0000
AD (Z2),&(SLAVE2_ACK) ;SLAVE OFFSET
RD (Z3),(Z2) ;PUT ACTUAL INTO Z3
CP (Z1),(Z3) ;MAKE SURE CHANGE HAPPENED
BN ZE @FINISH_FLAGS ;NOT CHANGED EXIT TO TRY AGAIN
@FLAG_17_END
ST (Z1),0
SE (_BG_FLAGS_2),1
DL UN 2
RE (_BG_FLAGS_2),1
RE (_FG_FLAGS_2),1
RET
@DO_FLAG_18
; SET THE SLAVE 3 SETPOINT
ST (Z2),MAS_TBL ;MASTER TABLE x0000
AD (Z2),&(SLAVE3_SP) ;SLAVE OFFSET
WR (Z2),(Z1) ;PASS SETPOINT TO SLAVE WITH Z1
ST (Z2),MAS_TBL ;MASTER TABLE x0000

```

```

AD (Z2),&(SLAVE3_ACK) ;SLAVE OFFSET
RD (Z3),(Z2) ;PUT ACTUAL INTO Z3
CP (Z1),(Z3) ;MAKE SURE CHANGE HAPPENED
BN ZE @FINISH_FLAGS ;NOT CHANGED EXIT TO TRY AGAIN
@FLAG_18_END
ST (Z1),0
SE (_BG_FLAGS_2),2
DL UN 2
RE (_BG_FLAGS_2),2
RE (_FG_FLAGS_2),2
RET
@DO_FLAG_19
; SET THE SLAVE 4 SETPOINT
ST (Z2),MAS_TBL ;MASTER TABLE x0000
AD (Z2),&(SLAVE4_SP) ;SLAVE OFFSET
WR (Z2),(Z1) ;PASS SETPOINT TO SLAVE WITH Z1
ST (Z2),MAS_TBL ;MASTER TABLE x0000
AD (Z2),&(SLAVE4_ACK) ;SLAVE OFFSET
RD (Z3),(Z2) ;PUT ACTUAL INTO Z3
CP (Z1),(Z3) ;MAKE SURE CHANGE HAPPENED
BN ZE @FINISH_FLAGS ;NOT CHANGED EXIT TO TRY AGAIN
@FLAG_19_END
ST (Z1),0
SE (_BG_FLAGS_2),3
DL UN 2
RE (_BG_FLAGS_2),3
RE (_FG_FLAGS_2),3
RET
@DO_FLAG_20
; SET THE SLAVE 5 SETPOINT
ST (Z2),MAS_TBL ;MASTER TABLE x0000
AD (Z2),&(SLAVE5_SP) ;SLAVE OFFSET
WR (Z2),(Z1) ;PASS SETPOINT TO SLAVE WITH Z1
ST (Z2),MAS_TBL ;MASTER TABLE x0000
AD (Z2),&(SLAVE5_ACK) ;SLAVE OFFSET
RD (Z3),(Z2) ;PUT ACTUAL INTO Z3
CP (Z1),(Z3) ;MAKE SURE CHANGE HAPPENED
BN ZE @FINISH_FLAGS ;NOT CHANGED EXIT TO TRY AGAIN
@FLAG_20_END
ST (Z1),0
SE (_BG_FLAGS_2),4
DL UN 2
RE (_BG_FLAGS_2),4
RE (_FG_FLAGS_2),4
RET
@DO_FLAG_21
; SET THE SLAVE 6 SETPOINT
ST (Z2),MAS_TBL ;MASTER TABLE x0000

```

```

AD (Z2),&(SLAVE6_SP) ;SLAVE OFFSET
WR (Z2),(Z1) ;PASS SETPOINT TO SLAVE WITH Z1
ST (Z2),MAS_TBL ;MASTER TABLE x0000
AD (Z2),&(SLAVE6_ACK) ;SLAVE OFFSET
RD (Z3),(Z2) ;PUT ACTUAL INTO Z3
CP (Z1),(Z3) ;MAKE SURE CHANGE HAPPENED
BN ZE @FINISH_FLAGS ;NOT CHANGED EXIT TO TRY AGAIN
@FLAG_21_END
ST (Z1),0
SE (_BG_FLAGS_2),5
DL UN 2
RE (_BG_FLAGS_2),5
RE (_FG_FLAGS_2),5
RET
@DO_FLAG_22
; SET THE SLAVE 7 SETPOINT
ST (Z2),MAS_TBL ;MASTER TABLE x0000
AD (Z2),&(SLAVE7_SP) ;SLAVE OFFSET
WR (Z2),(Z1) ;PASS SETPOINT TO SLAVE WITH Z1
ST (Z2),MAS_TBL ;MASTER TABLE x0000
AD (Z2),&(SLAVE7_ACK) ;SLAVE OFFSET
RD (Z3),(Z2) ;PUT ACTUAL INTO Z3
CP (Z1),(Z3) ;MAKE SURE CHANGE HAPPENED
BN ZE @FINISH_FLAGS ;NOT CHANGED EXIT TO TRY AGAIN
@FLAG_22_END
ST (Z1),0
SE (_BG_FLAGS_2),6
DL UN 2
RE (_BG_FLAGS_2),6
RE (_FG_FLAGS_2),6
RET
@DO_FLAG_23
; SET THE SLAVE 8 SETPOINT
ST (Z2),MAS_TBL ;MASTER TABLE x0000
AD (Z2),&(SLAVE8_SP) ;SLAVE OFFSET
WR (Z2),(Z1) ;PASS SETPOINT TO SLAVE WITH Z1
ST (Z2),MAS_TBL ;MASTER TABLE x0000
AD (Z2),&(SLAVE8_ACK) ;SLAVE OFFSET
RD (Z3),(Z2) ;PUT ACTUAL INTO Z3
CP (Z1),(Z3) ;MAKE SURE CHANGE HAPPENED
BN ZE @FINISH_FLAGS ;NOT CHANGED EXIT TO TRY AGAIN
@FLAG_23_END
ST (Z1),0
SE (_BG_FLAGS_2),7
DL UN 2
RE (_BG_FLAGS_2),7
RE (_FG_FLAGS_2),7
RET

```

@FINISH\_FLAGS

RET

## Actual Background Program (Logic Program) Opcodes

The program shown above is a source code listing that was used to generate the background programs shown below. The Flag program has been compiled into six background programs. The line number, program opcode and data, and a reference to the source code line number and operation are shown in the following listings. This listing can be modified to use all or fewer of the flags available. Just remember that each background program has 24 steps. The last step of each program must jump to the next program. The last step of program six must jump back to the beginning of the background program since it must continually loop to detect a flag set.

### Logic Program 1

```
1 ST C3 x0000      ;flags.PRG 56 ST (C3) 0
2 ST C7 x0000      ;flags.PRG 57 ST (C7) 0
3 GS UN x0bca      ;flags.PRG 58 GS UN @_SYS_INIT
4 GS UN x0601      ;flags.PRG 60 GS UN @_V35_RECIPES
5 BR UN x0004      ;flags.PRG 61 BR UN @MAIN_LOOP
6 SE A8 x000f      ;flags.PRG 75 SE (A8) 15
7 ST ZF A8         ;flags.PRG 76 ST (ZF) (A8)
8 BI #AD x0000     ;flags.PRG 78 BI (#AD) 0
9 GN ZE x0202     ;flags.PRG 79 GN ZE @DO_FLAG_16
10 BI #AD x0001    ;flags.PRG 80 BI (#AD) 1
11 GN ZE x1002    ;flags.PRG 81 GN ZE @DO_FLAG_17
12 BI #AD x0002   ;flags.PRG 82 BI (#AD) 2
13 GN ZE x0703    ;flags.PRG 83 GN ZE @DO_FLAG_18
14 BI #AD x0003   ;flags.PRG 84 BI (#AD) 3
15 GN ZE x1503    ;flags.PRG 85 GN ZE @DO_FLAG_19
16 BI #AD x0004   ;flags.PRG 86 BI (#AD) 4
17 GN ZE x0c04    ;flags.PRG 87 GN ZE @DO_FLAG_20
18 BI #AD x0005   ;flags.PRG 88 BI (#AD) 5
19 GN ZE x0305    ;flags.PRG 89 GN ZE @DO_FLAG_21
20 BI #AD x0006   ;flags.PRG 90 BI (#AD) 6
21 GN ZE x1105    ;flags.PRG 91 GN ZE @DO_FLAG_22
22 BI #AD x0007   ;flags.PRG 92 BI (#AD) 7
23 GN ZE x0806    ;flags.PRG 93 GN ZE @DO_FLAG_23
24 JP UN x0002
```

### Logic Program 2

```
1 BR UN x0000      ;flags.PRG 94 BR UN 0
2 ST Z2 x0000      ;flags.PRG 97 ST (Z2) x0000
3 AD Z2 x0040      ;flags.PRG 98 AD (Z2) x40
4 WR Z2 Z1         ;flags.PRG 99 WR (Z2) (Z1)
5 ST Z2 x0000      ;flags.PRG 100 ST (Z2) x0000
6 AD Z2 x0048      ;flags.PRG 101 AD (Z2) x48
7 RD Z3 Z2         ;flags.PRG 102 RD (Z3) (Z2)
8 CP Z1 Z3         ;flags.PRG 103 CP (Z1) (Z3)
```

---

---

Marathon Sensors Inc.

3100 E. Kemper Road, Cincinnati, Ohio 45241  
Phone: 513-772-1000 Fax: 513-326-7090

9 JN ZE x1606	;flags.PRG 104 BN ZE @FINISH_FLAGS
10 ST Z1 x0000	;flags.PRG 106 ST (Z1) 0
11 SE #B7 x0000	;flags.PRG 107 SE (#B7) 0
12 DL UN x0002	;flags.PRG 108 DL UN 2
13 RE #AD x0000	;flags.PRG 109 RE (#AD) 0
14 RE #B7 x0000	;flags.PRG 110 RE (#B7) 0
15 BR UN x0000	;flags.PRG 111 BR UN 0
16 ST Z2 x0000	;flags.PRG 114 ST (Z2) x0000
17 AD Z2 x0041	;flags.PRG 115 AD (Z2) x41
18 WR Z2 Z1	;flags.PRG 116 WR (Z2) (Z1)
19 ST Z2 x0000	;flags.PRG 117 ST (Z2) x0000
20 AD Z2 x0049	;flags.PRG 118 AD (Z2) x49
21 RD Z3 Z2	;flags.PRG 119 RD (Z3) (Z2)
22 CP Z1 Z3	;flags.PRG 120 CP (Z1) (Z3)
23 JN ZE x1606	;flags.PRG 121 BN ZE @FINISH_FLAGS
24 JP UN x0003	

### Logic Program 3

1 ST Z1 x0000	;flags.PRG 123 ST (Z1) 0
2 SE #B7 x0001	;flags.PRG 124 SE (#B7) 1
3 DL UN x0002	;flags.PRG 125 DL UN 2
4 RE #B7 x0001	;flags.PRG 126 RE (#B7) 1
5 RE #AD x0001	;flags.PRG 127 RE (#AD) 1
6 BR UN x0000	;flags.PRG 128 BR UN 0
7 ST Z2 x0000	;flags.PRG 131 ST (Z2) x0000
8 AD Z2 x0042	;flags.PRG 132 AD (Z2) x42
9 WR Z2 Z1	;flags.PRG 133 WR (Z2) (Z1)
10 ST Z2 x0000	;flags.PRG 134 ST (Z2) x0000
11 AD Z2 x004a	;flags.PRG 135 AD (Z2) x4a
12 RD Z3 Z2	;flags.PRG 136 RD (Z3) (Z2)
13 CP Z1 Z3	;flags.PRG 137 CP (Z1) (Z3)
14 JN ZE x1606	;flags.PRG 138 BN ZE @FINISH_FLAGS
15 ST Z1 x0000	;flags.PRG 140 ST (Z1) 0
16 SE #B7 x0002	;flags.PRG 141 SE (#B7) 2
17 DL UN x0002	;flags.PRG 142 DL UN 2
18 RE #B7 x0002	;flags.PRG 143 RE (#B7) 2
19 RE #AD x0002	;flags.PRG 144 RE (#AD) 2
20 BR UN x0000	;flags.PRG 145 BR UN 0
21 ST Z2 x0000	;flags.PRG 148 ST (Z2) x0000
22 AD Z2 x0043	;flags.PRG 149 AD (Z2) x43
23 WR Z2 Z1	;flags.PRG 150 WR (Z2) (Z1)
24 JP UN x0004	

### Logic Program 4

1 ST Z2 x0000	;flags.PRG 151 ST (Z2) x0000
2 AD Z2 x004b	;flags.PRG 152 AD (Z2) x4b
3 RD Z3 Z2	;flags.PRG 153 RD (Z3) (Z2)
4 CP Z1 Z3	;flags.PRG 154 CP (Z1) (Z3)

5 JN ZE x1606	;flags.PRG 155 BN ZE @FINISH_FLAGS
6 ST Z1 x0000	;flags.PRG 157 ST (Z1) 0
7 SE #B7 x0003	;flags.PRG 158 SE (#B7) 3
8 DL UN x0002	;flags.PRG 159 DL UN 2
9 RE #B7 x0003	;flags.PRG 160 RE (#B7) 3
10 RE #AD x0003	;flags.PRG 161 RE (#AD) 3
11 BR UN x0000	;flags.PRG 162 BR UN 0
12 ST Z2 x0000	;flags.PRG 165 ST (Z2) x0000
13 AD Z2 x0044	;flags.PRG 166 AD (Z2) x44
14 WR Z2 Z1	;flags.PRG 167 WR (Z2) (Z1)
15 ST Z2 x0000	;flags.PRG 168 ST (Z2) x0000
16 AD Z2 x004c	;flags.PRG 169 AD (Z2) x4c
17 RD Z3 Z2	;flags.PRG 170 RD (Z3) (Z2)
18 CP Z1 Z3	;flags.PRG 171 CP (Z1) (Z3)
19 JN ZE x1606	;flags.PRG 172 BN ZE @FINISH_FLAGS
20 ST Z1 x0000	;flags.PRG 174 ST (Z1) 0
21 SE #B7 x0004	;flags.PRG 175 SE (#B7) 4
22 DL UN x0002	;flags.PRG 176 DL UN 2
23 RE #B7 x0004	;flags.PRG 177 RE (#B7) 4
24 JP UN x0005	

## Logic Program 5

1 RE #AD x0004	;flags.PRG 178 RE (#AD) 4
2 BR UN x0000	;flags.PRG 179 BR UN 0
3 ST Z2 x0000	;flags.PRG 182 ST (Z2) x0000
4 AD Z2 x0045	;flags.PRG 183 AD (Z2) x45
5 WR Z2 Z1	;flags.PRG 184 WR (Z2) (Z1)
6 ST Z2 x0000	;flags.PRG 185 ST (Z2) x0000
7 AD Z2 x004d	;flags.PRG 186 AD (Z2) x4d
8 RD Z3 Z2	;flags.PRG 187 RD (Z3) (Z2)
9 CP Z1 Z3	;flags.PRG 188 CP (Z1) (Z3)
10 JN ZE x1606	;flags.PRG 189 BN ZE @FINISH_FLAGS
11 ST Z1 x0000	;flags.PRG 191 ST (Z1) 0
12 SE #B7 x0005	;flags.PRG 192 SE (#B7) 5
13 DL UN x0002	;flags.PRG 193 DL UN 2
14 RE #B7 x0005	;flags.PRG 194 RE (#B7) 5
15 RE #AD x0005	;flags.PRG 195 RE (#AD) 5
16 BR UN x0000	;flags.PRG 196 BR UN 0
17 ST Z2 x0000	;flags.PRG 199 ST (Z2) x0000
18 AD Z2 x0046	;flags.PRG 200 AD (Z2) x46
19 WR Z2 Z1	;flags.PRG 201 WR (Z2) (Z1)
20 ST Z2 x0000	;flags.PRG 202 ST (Z2) x0000
21 AD Z2 x004e	;flags.PRG 203 AD (Z2) x4e
22 RD Z3 Z2	;flags.PRG 204 RD (Z3) (Z2)
23 CP Z1 Z3	;flags.PRG 205 CP (Z1) (Z3)
24 JP UN x0006	

## Logic Program 6

1 BN ZE x0016	;flags.PRG 206 BN ZE @FINISH_FLAGS
2 ST Z1 x0000	;flags.PRG 208 ST (Z1) 0
3 SE #B7 x0006	;flags.PRG 209 SE (#B7) 6
4 DL UN x0002	;flags.PRG 210 DL UN 2
5 RE #B7 x0006	;flags.PRG 211 RE (#B7) 6
6 RE #AD x0006	;flags.PRG 212 RE (#AD) 6
7 BR UN x0000	;flags.PRG 213 BR UN 0
8 ST Z2 x0000	;flags.PRG 216 ST (Z2) x0000
9 AD Z2 x0047	;flags.PRG 217 AD (Z2) x47
10 WR Z2 Z1	;flags.PRG 218 WR (Z2) (Z1)
11 ST Z2 x0000	;flags.PRG 219 ST (Z2) x0000
12 AD Z2 x004f	;flags.PRG 220 AD (Z2) x4f
13 RD Z3 Z2	;flags.PRG 221 RD (Z3) (Z2)
14 CP Z1 Z3	;flags.PRG 222 CP (Z1) (Z3)
15 BN ZE x0016	;flags.PRG 223 BN ZE @FINISH_FLAGS
16 ST Z1 x0000	;flags.PRG 225 ST (Z1) 0
17 SE #B7 x0007	;flags.PRG 226 SE (#B7) 7
18 DL UN x0002	;flags.PRG 227 DL UN 2
19 RE #B7 x0007	;flags.PRG 228 RE (#B7) 7
20 RE #AD x0007	;flags.PRG 229 RE (#AD) 7
21 BR UN x0000	;flags.PRG 230 BR UN 0
22 BR UN x0000	;flags.PRG 232 BR UN 0

### Reasons Why the Limit Would be Exceeded

There are several reasons why a time limit alarm may occur will these programs are being used. The main ones to consider would be if;

#### The Background program stops

Check to make sure the background program is still running. If it has stopped due to a power lose make use the Auto Program Start (APS) is set to YES.

#### Communications to the slave stops.

Communication to the slave must be setup correctly. Verify that the slaves are talking to the master controller by checking the slave status. On the Dualpro select the SLVE INST display under the Page Disp key. Press the UP Arrow key until the ST 1 parameter is displayed. If “G 00” is displayed the communications with the slave is OK. If an “S XX” is displayed where XX is some number, it is possible the slave is set in local mode and not accepting remote set points. If the lower display is “B XX” where XX is probably 25 or less, then the communications link to the slave has failed. On the Multipro select the SLAVE menu. The ST(atus) column indicates the status with the appropriate letter similar to the Dualpro. Insure that the communications setup on the master Aux bus is the same as the slave buad rate and parity. Verify the slave baud rate, parity and address.

#### The slave controller is set to local mode.

If the slave is in local mode it will not accept a remote set point. The background program compares the actual slave set point to the value set. If these numbers do not match a time error occurs. See the above checks to verify communications status and insure that the slave controller is in remote mode.