

## Section 6. COMMUNICATING WITH THE POSITIONER

---

### Chapter Objectives

The information contained in this chapter will enable you to set up communications with the positioner using an RS232C serial link. If more than one positioner is present in the system, details are provided on the connection of multiple positioners using the RS-232 Daisy Chain.

#### Command Interface

The interface is a three wire implementation (Tx, Rx, Ground) of RS232C. The Tx and Rx lines requires a minimum voltage swing of  $\pm 3$  volts.

Hardware handshaking is not supported in any form. The computer or terminal sending characters to the positioner should have its handshaking disabled by either hardware or software.

#### Communication Parameters

The positioner communications protocol is fixed and as follows:-

9600 baud, 8 data bits, no parity, 1 start bit, 1 stop bit

Device address: 1-32

Echo function: all characters received are immediately re-transmitted. This function can be enabled using the SSA0 command or disabled using the SSA1 command.

Once entered, a parameter will be remembered until power down. It is not necessary to keep re-stating the same parameter value.

When you power up the positioner, all parameters are assigned default values which the 1DR command will show you. You can change any of these and then make them the new default value by typing the SV (Save command).

(see also commands RFS, RIFS).

XON, XOFF software handshaking is supported.

#### Installing the RS232C

RS232C connections from the controller to the positioner are as shown in Figure 6-2. Note that the Tx and Rx lines are cross-connected so that transmit output is connected to receive input.

**Address Selection**

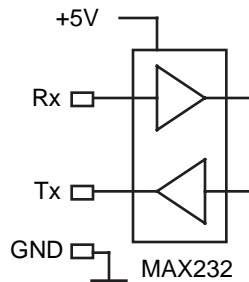
A unique address must be assigned to each positioner in the chain. Normally positioner 1 will be the first in the serial communications link. The positioner's address is assigned by configuring the DIL switch on top of the drive in a binary pattern as shown in Table 6-1.

The DIL switch can be accessed through the ventilation slots in the top of the drive using a small screwdriver or similar tool. Make sure that the AC power is disconnected before attempting to change the switch settings. SW1 is nearest to the front of the drive.

<b>Switch settings for addresses 1 to 16 (switch 5 off)</b>				
<b>Address</b>	<b>Sw 1</b>	<b>Sw 2</b>	<b>SW 3</b>	<b>Sw 4</b>
1 (17)	ON	OFF	OFF	OFF
2 (18)	OFF	ON	OFF	OFF
3 (19)	ON	ON	OFF	OFF
4 (20)	OFF	OFF	ON	OFF
5 (21)	ON	OFF	ON	OFF
6 (22)	OFF	ON	ON	OFF
7 (23)	ON	ON	ON	OFF
8 (24)	OFF	OFF	OFF	ON
9 (25)	ON	OFF	OFF	ON
10 (26)	OFF	ON	OFF	ON
11 (27)	ON	ON	OFF	ON
12 (28)	OFF	OFF	ON	ON
13 (29)	ON	OFF	ON	ON
14 (30)	OFF	ON	ON	ON
15 (31)	ON	ON	ON	ON
16 (32)	OFF	OFF	OFF	OFF

For addresses in brackets (17 to 32), turn on Switch 5.

**Table 6-1. Positioner Address Selection**



**Figure 6-1. RS232 Input**

**Controller Daisy  
Chain Wiring**

You may daisy chain up to 32 axes. Individual drive addresses are set with the Controller DIP switches (see Table 6-1). When daisy chained, the units may be addressed individually or simultaneously. You should establish a unique address for each axis. Refer to Figure 6-2 for Controller daisy chain wiring configuration.

Commands prefixed with a device address command only the unit specified. Commands without an address command all units on the daisy chain. The general rule is: *Any command that causes the drive to transmit information from the RS-232C port (such as a status or report command), must be prefixed with a device address.* This prevents daisy chained units from all transmitting at the same time.

Attach device identifiers to the front of the command. The Go (G) command instructs all units on the daisy chain to go, while 1G tells only unit #1 to go.

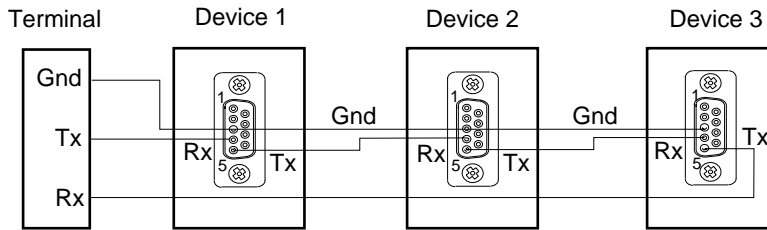
When you use a single communications port to control more than one Controller, all units in the daisy chain receive and echo the same commands. Each device executes these commands, unless this command is preceded with an address that differs from the units on the daisy chain. This becomes critical if you instruct any indexer to transmit information. To prevent all of the units on the line from responding to a command, you must precede the command with the device address of the designated unit.

No Controller executes a device-specific command unless the unit number specified with the command matches the Controller unit number. Device-specific commands include both buffered and immediate commands.

You must use status-request commands in an orderly fashion. Commands should only be issued when the host is ready to read the response. You should not send new commands until you receive a response from the previous status-request command. In particular, you should not issue an immediate-status command until the host receives a buffered command status response. If this is not followed, the command responses will be intertwined, rendering the data useless.

If you enable the Interactive mode (EX1), the Controller that is currently echoing will respond with a prompt (>). This prevents all the Controllers from sending out > in a daisy chain. Typically, you should disable the Interactive mode when you use a host computer with the Controller. The default for the EX command is enabled (EX1).

Figure 6-2 shows a multiple-drive configuration (daisy-chain) of RS-232C ports from one controlling terminal or computer.



**Figure 6-2. RS-232C Daisy Chain Configuration**

**Example**

Three Controllers are connected to an RS-232C daisy chain. Send the following commands:

<u>Command</u>	<u>Description</u>
<b>MN</b>	Sets unit to Preset mode for all three controllers
<b>A5</b>	Sets acceleration to 5 rps <sup>2</sup> for all three controllers
<b>V10</b>	Sets velocity to 10 rps for all three controllers
<b>LD3</b>	Disables limits (if they are not connected)
<b>1D25000</b>	Sets axis 1 distance to 25,000 steps
<b>2D50000</b>	Sets axis 2 distance to 50,000 steps
<b>3D100000</b>	Sets axis 3 distance to 100,000 steps
<b>G</b>	Moves all axes

Unit 1 moves 25,000 steps, unit 2 moves 50,000 steps and unit 3 moves 100,000 steps. All three units use the same acceleration and velocity rates.

**Interactive  
operation  
with a  
PC/PLC**

To create a reliable, trouble free interactive RS232 communications control program between a PC/PLC and the serial interface you should note the following points :

- Use the X-WARE package for terminal emulation on an IBM PC or equivalent computer system.
- The interface should only be operated in computer mode to suppress error messages and extended text output.
- The program should operate the interface with a response check during critical processes, such as sequence download. The response check will need to detect :
  - No response at all

- Incorrect characters
- More characters than expected
- Less characters than expected
- Control characters such as EOF caused by errors
- Correct response

The response expected should be either echoback, or character returned by interrogation commands, or the sequence of the two.

- If there is no check for echoback on each character, you should include a delay of up to 1ms between each character sent . This may require characters to be sent individually rather than as strings.
- You should not use the simple BASIC INPUT statement as this 'hangs' the link without time out if an error causes no carriage-return character to be received from the interface.
- Interpreted BASIC should not be used, as it is too slow for most applications.
- Interrupt error handling, such as the BASIC 'ON ERROR' statement, should be enabled.
- Commands such as SV, Z, RFS and RIFS take several seconds to execute. This also applies to status commands like DR and HELP. It is recommended you insert a suitable delay after these commands to allow them to fully execute their intended function without corrupting immediately following commands, although the use of these commands in an interactive situation is unlikely.
- If problems arise the communications error responses actioned by the R command will help diagnosis. In some cases a recovery can be attempted by sending the command again (such as the R command itself). But this may not always be appropriate, for example LD3 altered to L3 would not be fully corrected by simply sending LD3 again.
- A routine that breaks some of these rules may appear to work satisfactorily, but is not necessarily safe. The worst case communications response condition may involve a number of coincidences, such as sending a status request just as the interface hits a limit, just as it starts decelerating etc. Any motion control requirement explicitly takes priority over communications under these circumstances, and can therefore cause a communications problem unless you wait for echoback. Also different issues of software may have different execution time characteristics which can make a routine based on delays work at one software issue, and not another.

*Accurate defensive programming is required if safety constraints are to be met and possible damage to hardware prevented.*

### **Interactive Programming**

To help you achieve reliable, trouble free interactive programming between a Controller Positioner and a PC/PLC the following guidelines will be useful.

- The interface should be operated in terse mode (EX0) to suppress unsolicited error messages and extended text output. This will also allow for extra communication time.
- The program should include an interface response check, especially during critical processes such as sequence download. Checks should be made for:

- \*No response at all
- \*Incorrect characters
- \*More or less characters than expected
- \*Control characters such as EOF formed by errors
- \*The correct response

Where the correct response is the echoback of the transmitted characters or characters returned by interrogation commands or a combination of the two.

- The command buffer is 256 characters long. When it has filled up, the interface will echo a <CTRL>G instead of the correct character. Subsequent characters received will 'overwrite' the last character in the buffer until a 'free' space occurs.
- Given the above, it is preferable to transmit one character and wait for correct echoback before transmitting the next character. This enables an error to be rectified immediately.
- When monitoring the echoback characters in verbose mode (EX1), some are echoed as a combination of characters, for example:

<CR> is echoed as <CR> <LF>.

This is not the case in terse mode (EX0).

- If there is no check for echoback on each individual character, an explicit delay of at least 1ms between each character must be included. This may mean sending characters individually rather than in strings.
- Certain interface commands, notably ON, Z etc. take some time to execute and consequently momentarily interfere with communications. An explicit delay (several seconds is typical) should be inserted after these commands before another is sent.

- This interference of communications can involve temporarily not servicing the RS-232 interrupt on the Controller, allowing subsequent characters received by the UART, during this condition, to cause an overrun error within the UART.
- The status request command (R) can be used to provide helpful diagnostics. The responses generated by the interface are as follows:

\*R<CR> - ready for a command with no errors

\*S<CR> - ready for a command with function error

\*T<CR> - ready for a command with previous comms error

\*U<CR> - ready for a command with function error and previous comms error

\*B<CR> - busy performing a move with no errors

\*C<CR> - busy performing a move with function error

\*D<CR> - busy performing a move with previous comms error

\*E<CR> - busy performing a move with function error and previous comms error

- The average execution time of commands is approximately 1.5ms. The execution time is dependent upon the activity of the indexer. When performing a move, especially during acceleration and deceleration, there is generally less time available for command execution. This time will delay the execution of subsequent commands, and hence reduce the rate at which the command input buffer is cleared.
- The interface will correctly respond to XON/XOFF protocol, but will not be able to generate the XON/XOFF signals when the command input buffer is full.
- If the control program does not use the echo back facility provided by the interface it may be disabled by sending the command SSA1. This will speed up the character handling routine a little, but will stop any commands from being transmitted to axes further down the chain, so should be used with care.

The fact that a routine that breaks some of these rules 'works' is irrelevant. Worst case communication response conditions may involve a number of coincidences such as sending a status request just as the interface hits a limit, just as it starts to decelerate, just as the servo is responding to a load glitch etc. Any motion control requirement explicitly takes priority over communications under these circumstances and can therefore cause a communication problem unless you wait for echoback.

If the Controller echoback is turned off by accident, causing an apparent loss of communication (hanging), it may be re-enabled by typing the following sequence:

CTRL-Q<space>E<space>1R<carriage return>

Note:

[1] This sequence is case sensitive - make sure you are in upper case.

[2] CTRL-Q means hold the 'control' key down and press the Q key at the same time. This sends the X-ON ASCII character. The Controller obeys the X-ON X-OFF communications protocol which is used to hold up communications if an interface is sending too much data too quickly. This means that if X-OFF is sent down the chain (the character CTRL-S) all communications from the axes stop until CTRL-Q (X-ON) is sent down the chain. This includes echoback of commands. If CTRL-Q is never sent the system will 'hang'.

[3] You may type CTRL-S by mistake. This condition can be cleared by cycling the power.

[4] The E command is the enable RS-232 command which can easily be miss-typed as the F command which will lock out the keyboard. Cycling the power will not cure this problem since SV cannot be typed following a keyboard lock out.

[5] The <space>1 in <space>1R is important. It sets axis 1 as the echoback device (as would <carriage return>1). If you make a typing error, such as TRD<space>3000 instead of TRD3000 the <space>3 sets axis 3 as the echoback device, where R can be any command.

Certain 'tidy up' commands can be useful. For example:

- If you are not getting the typed characters echoed back you may need to use the SSA0 command.
- To get a prompt (\$ or >) you may need to use the EX1 command.
- To get out of a \$ prompt (indicating a sequence definition) type XT.
- If buffered commands are being ignored, type S in case the indexer is busy. You should avoid typing Z or cycling the power.

In summary:

- Don't type CTRL-S
- Avoid accidental use of the command F
- Don't address a non-existent axis
- Don't rubout a device address
- Type carefully

None of the above conditions is an error so the status LED on the Controller will not indicate a fault and the axis may well be energised or even moving.

## Programming With the RP240

### Prompting an Operator or Displaying Information

In many motion control applications, the most important requirement is the operator interface. Presenting information to an operator in a desired format is often difficult. The RP 240 has two visual indicators to help present information to the operator. The simplest indicator is the 8 LEDs on the panel. These LEDs can be turned on or off with the DLED command. The LEDs can be used in conjunction with the outputs to show the state of an output, or they can be used to show status, such as motor moving, specific sequences in progress, etc.

If a DLED10001011 command is issued, the LEDs shown below would be illuminated. These eight LEDs can be labelled, using the slide in card provided, to represent cycle status, output status, etc.

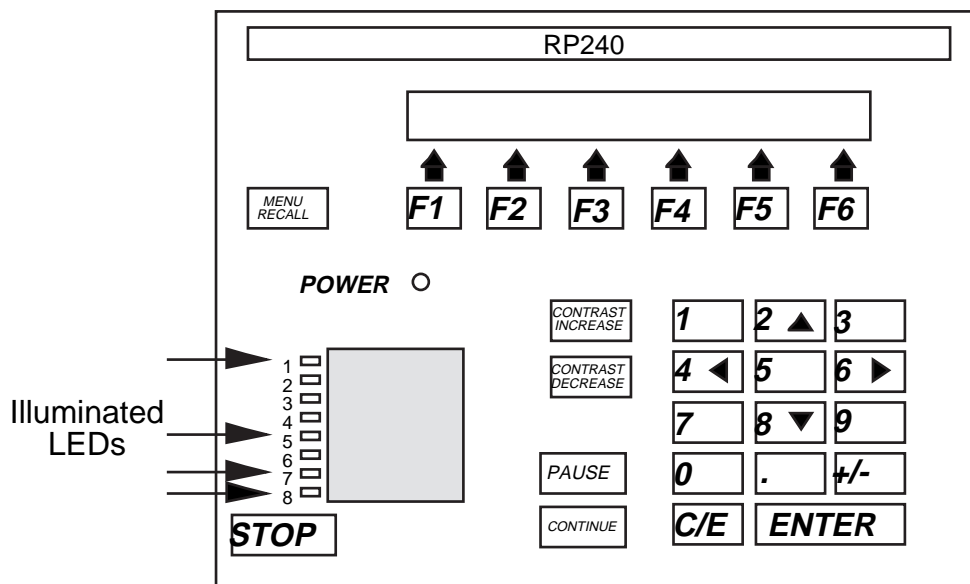


Figure 6-3. Status LEDs

## 58 PDHX-E SERIES DRIVE USER GUIDE

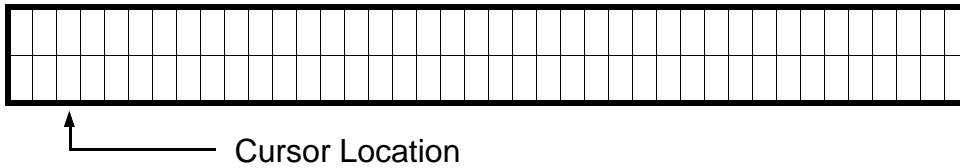
---

The other indicator is the two line, 40 character LCD display. This display can be controlled with specific Extended X Language commands. The Position Cursor (DPC) command allows the user to program the location of the cursor on the LCD display. The 'Display Text Data on RP240 Display' (DXTX) command allows the user to place text, beginning at the current cursor location, on the LCD display. For example: A user wishes their operator to see the message ENTER THE CYCLE COUNT. The user wants this message placed on line two, starting after two spaces in from the left. Following the message by three spaces the cycle count is to be entered, and stored in variable 1. Below are the steps required to accomplish this.

The cursor does not appear on the display. The cursor is displayed when the VARn=NUM or DPC commands are used.

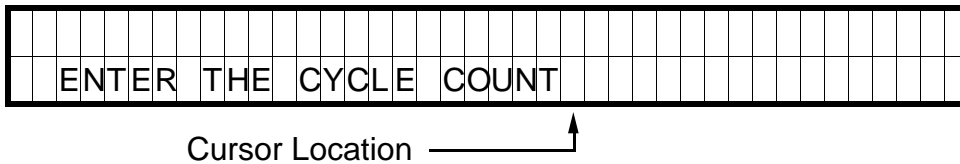
### Step 1

Issue the DPC202 command (line 2, position 02)



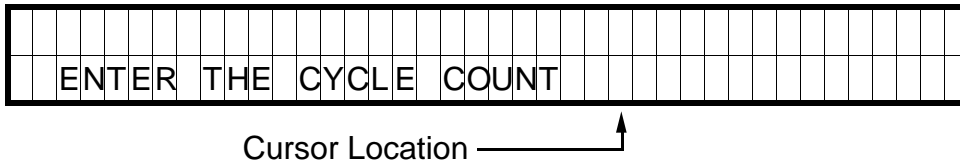
### Step 2

Issue the DXTX"ENTER THE CYCLE COUNT" command.



### Step 3

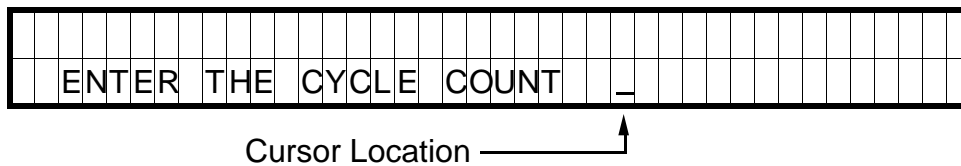
Issue the DPC225 command (line 2, position 25)



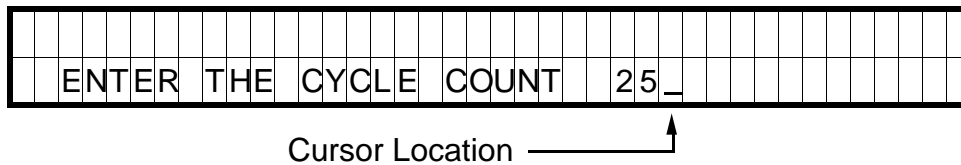
Once the operator has been provided with the prompting message, the actual cycle count data is provided using the Read and Enable Numeric Keypad (VARn=NUM) or Read and Enable Function Keys (VARn=FUN). The VARn=NUM command will enable the numeric keypad and allow the operator to enter information. The numbers, as entered, will be displayed at the current cursor location. Once the ENTER key is pressed, the number will be accepted by the Controller.

**Step 4**

Issue the VAR1=NUM command.

**Step 5**

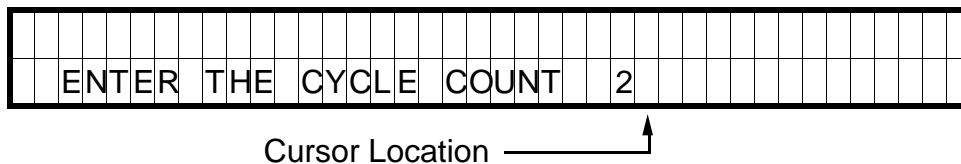
Press a 2, followed by a 5.



If the wrong value is entered press the C/E key and re-enter the value.

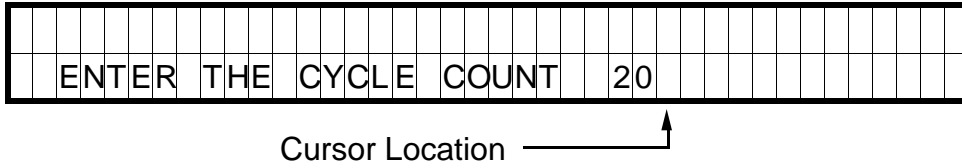
**Step 6**

Press the C/E key.



**Step 7**

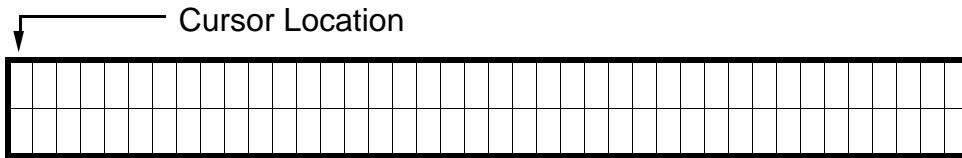
Press a 0 followed by an ENTER.



Once the cycle count has been entered, clear the display and enter text which allows an operator to choose a particular function (in this case a choice of parts).

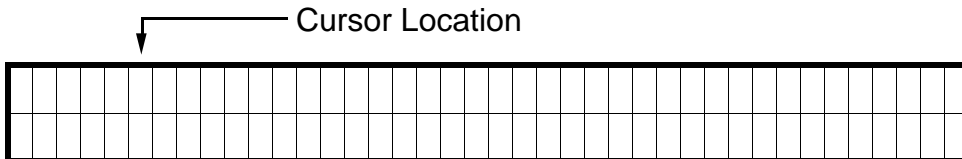
**Step 8**

DCLR0 command is issued.



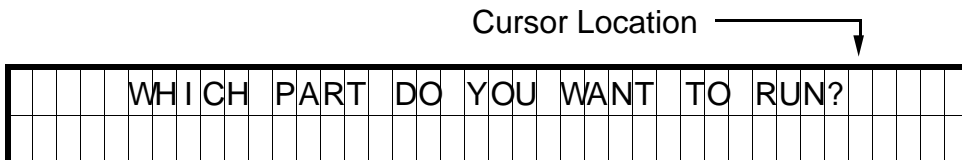
**Step 9**

Issue the DPC105 command (line 1, position 05)



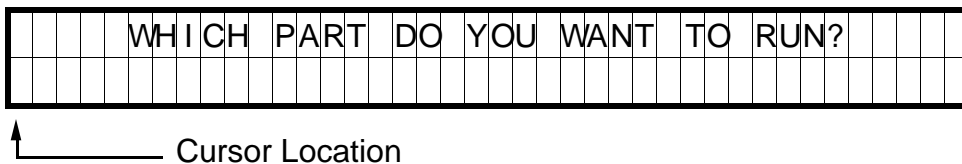
**Step 10**

Issue the DTXT"WHICH PART DO YOU WANT TO RUN?" command.



**Step 11**

Issue the DPC200 command (line 2, position 0)





In step 14 above, the operator pressed the F1 key. This value was stored in variable 2, and was used to select PART1. The following is a command example:

<u>Command</u>	<u>Description</u>
REPEAT	loop
IF(VAR2=1) XR11 NIF	Branch to sequence 11 if F1 is pressed
IF(VAR2=2) XR12 NIF	Branch to sequence 12 if F2 is pressed
IF(VAR2=3) XR13 NIF	Branch to sequence 13 if F3 is pressed
IF(VAR2=4) XR14 NIF	Branch to sequence 14 if F4 is pressed
IF(VAR2=5) XR15 NIF	Branch to sequence 15 if F5 is pressed
IF(VAR2=6) XR16 NIF	Branch to sequence 16 if F6 is pressed
UNTIL(VAR2=0)	End loop on menu recall pressed

For further information on the commands used in the example above, refer to the software reference guide for the Controller.

### **Enabling STOP and PAUSE Keys**

In addition to the function keys and numeric keypad, there are three other keys. The STOP key, and the PAUSE and CONTINUE keys are enabled by default, and can be disabled by user commands.

Typically, if an application uses the STOP key, the key will be enabled (DSTP1) in the power-up sequence. However, the STOP key can be enabled and disabled in any sequence, and at any time.

The PAUSE and CONTINUE keys can be enabled in any sequence. The DCNT command enables the PAUSE and CONTINUE keys.

If SSH is enabled (SSH1), the STOP key will function as a controlled stop. After motion is halted, the Controller will resume command processing with the command directly following the command that was stopped with the STOP key.

## Sample Program

This section provides an example of an RP240 application program. Refer to the Software Reference Guide for a more detailed description of the commands.

### *Power-up Sequence*

A configuration sequence can be used to initialise the Controller to a state compatible with the RP240 and used to initialise variables.

```
XE63
XD63
LD3 1DSTP1
VAR20=5
VAR21=1
XR21
XT
```

### *Sequence #21*

In this example sequence #21 provides the main menu for a demonstration program. Control is transferred to another sequence based on a function key input.

```
XE21
XD21
1DCLR0
1DPC106
1DTXT"DIGIPLAN'S RP240 DEMO PROGRAM"
1DPC200
1DTXT"ACCESS"
1DPC235
1DTXT"EXIT"
VAR1=FUN
IF(VAR1=1)
    XG22
ELSE
    IF(VAR1=6)
        1DCLR0
        HALT
    ELSE
        1DPC220
        1DTXT"WRONG BUTTON!"
        T1
        XG21
    NIF
NIF
XT
```

### **Sequence #22**

Sequence #22 prompts you for a password, and then transfers control to sequence #23. If the password is incorrect, control is passed to sequence #21.

```
XE22
XD22
1DCLR0
1DPC100
1DTXT"ENTER CODE NUMBER xxxx"
1DPC118
VAR1=NUM
IF(VAR1<1234 OR VAR1>1234)
    1DPC220
    1DTXT"WRONG NUMBER"
    T1
    XG21
ELSE
    XG23
NIF
XT
```

### **Sequence #23**

Sequence #23 asks you to make a selection via the function keys. Control is passed to a sequence based upon the function key pressed.

```
XE23
XD23
1DCLR0
1DPC101
1DTXT"JOG I/O TEACH MAKE FEED EXIT"
1DPC201
1DTXT"AXIS TEST MODE MOVE TO LENGTH"
VAR1=FUN
IF(VAR1=1)
    XR24
ELSE
    IF(VAR1=2)
        XR25
    ELSE
        IF(VAR1=3)
            XR26
        ELSE
            IF(VAR1=4)
                XR27
            ELSE
                IF(VAR1=5)
                    XG28
                ELSE
```

```

                                IF(VAR1=6)
                                  XG21
                                NIF
                              NIF
                            NIF
                          NIF
                        NIF
                      NIF
                    NIF
                  NIF
                NIF
              NIF
            NIF
          NIF
        NIF
      NIF
    NIF
  NIF
NIF
XG23
XT

```

**Sequence #24**

Sequence #24 simulates jogging. You can select CW or CCW jog options of either high or low velocity. The default jog velocities are stored in variables 20 and 21, which were assigned in sequence #63. The MENU RECALL key is used to exit Jog mode.

```

XE24
XD24
A75
VAR1=0
1DCLR0
1DPC101
1DTXT"CW CW CCW CCW STOP"
1DPC201
1DTXT"LOW HIGH LOW HIGH MOTION"
MC
VAR1=FUN
REPEAT
  VAR3=0
  WHILE(VAR3=0)
    IF(VAR1=1) H+ V(VAR21) G NIF
    IF(VAR1=2) H+ V(VAR20) G NIF
    IF(VAR1=3) H- V(VAR21) G NIF
    IF(VAR1=4) H- V(VAR20) G NIF
    IF(VAR1=5) 1DCLR0 1DPC212 1DTXT"INVALID SELECTION" T1 XG23 NIF {all on one line}
    VAR3=1
  NWHILE
  REPEAT
    VAR1=FUN
  UNTIL(VAR1=6)
  SB
  T0.3
  VAR1=FUN
UNTIL(VAR1=0)
V0
MN
XT

```

**Sequence #25**

Sequence #25 provides input status information and changes the state of LEDs.

```
XE25
XD25
1DCLR0
1DPC100
1DTXT"INPUT STATUS"
1DPC200
1DTXT"UPDATES"
1DPC233
1DTXT"OUTPUTS"
REPEAT
    VAR1=FUN
    1DPC113
    IF(IN_1) 1DTXT"1" ELSE 1DTXT"0" NIF
    IF(IN_X1) 1DTXT"1" ELSE 1DTXT"0" NIF
    IF(IN_XX1) 1DTXT"1" ELSE 1DTXT"0" NIF
    IF(IN_XXX1) 1DTXT"1" ELSE 1DTXT"0" NIF
    IF(IN_XXXX1) 1DTXT"1" ELSE 1DTXT"0" NIF
    IF(IN_XXXXX1) 1DTXT"1" ELSE 1DTXT"0" NIF
    IF(IN_XXXXXX1) 1DTXT"1" ELSE 1DTXT"0" NIF
    IF(IN_XXXXXXX1) 1DTXT"1" ELSE 1DTXT"0" NIF
    IF(IN_XXXXXXXX1) 1DTXT"1" ELSE 1DTXT"0" NIF
    IF(IN_XXXXXXXXX1) 1DTXT"1" ELSE 1DTXT"0" NIF
UNTIL(VAR1=6)
1DCLR0
1DPC100
1DTXT"THE LEDS BELOW WILL CHANGE STATE"
1DLED00000001 T1
1DLED00000011 T1
1DLED00000111 T1
1DLED00001111 T1
1DLED00011111 T1
1DLED00111111 T1
1DLED01111111 T1
1DLED11111111 T1
1DLED00000000 T1
VAR=0
XT
```

The 10 input bits in sequence #25 correspond to the total number of Controller user inputs.

**Sequence #26**

Sequence #26 sets up the learning mode for moves.

```
XE26
XD26
MPA
MN
VAR30=1
VAR29=1
VAR28=33
REPEAT
    1DCLR0
    1DTXT"HOW MANY MOVES TO TEACH?_x"
    1DPC200
    1DTXT"(MAXIMUM OF THREE MOVES)"
    1DPC125
    VAR30=NUM
UNTIL(VAR30>0 AND VAR30<4)
L(VAR30)
PZ
1DCLR0
1DTXT"PLEASE MAKE MOVE# "
1DVO29,2,0,0
1DPC200
T1
XR24
XR(VAR28)
VAR29=VAR29+1
VAR28=VAR28+1
N
VAR1=0
XT
```

**Sequence #27**

Sequence #27 runs the pre-programmed moves defined by sequence #28.

```
XE27
XD27
A100
MPI
PZ
V10
VAR29=40
L(VAR30)
XR(VAR29)
G
VAR29=VAR29+1
N
VAR1=0
XT
```

**Sequence #28**

Sequence #28 prompts the operator to select the feed length, maximum speed and to count the number of items to be cut.

```
XE28 XD28
1DCLR0 1DPC100
1DTXT"SELECT FEED LENGTH, MAX SPEED, AND COUNT"
T2 1DCLR1
REPEAT
  VAR30=0
  1DPC102
  1DTXT"ENTER FEED LENGTH IN INCHES: (0-99.9)"
  1DPC212
  1DTXT"FEED LENGTH=xx.x"
  1DPC224
  VAR2=NUM
  IF(VAR2>99.9 OR VAR2<0)
    1DCLR0
    1DPC215
    1DTXT"OUT OF RANGE"
    T2
    1DCLR2
    VAR30=1
  NIF
  IF(VAR30=0)
    1DCLR0
    1DPC112
    1DTXT"FEED LENGTH="
    1DPC125
    1DVO2,2,1,0
    1DPC202
    1DTXT"YES"
    1DPC236
    1DTXT"NO"
    VAR22=FUN
  NIF
  IF(VAR22>1)
    1DCLR0
    1VAR30=1
  NIF
  UNTIL(VAR30=0)
  1DCLR0
  REPEAT
    VAR30=0
    1DPC101
    1DTXT"ENTER MAX SPEED (RPM): (0-2400RPM)"
    1DPC215
    1DTXT"MAX SPEED=xxxx"
    1DPC225
    VAR3=NUM
    IF(VAR3>2400 OR VAR3<0)
      1DCLR0
      1DPC215
```

```
1DTXT"OUT OF RANGE"  
T2  
1DCLR2  
VAR30=1  
NIF  
IF(VAR30=0)  
1DCLR0  
1DPC114  
1DTXT"MAX SPEED"  
1DPC125  
1DVO3,4,2,0  
1DPC202  
1DTXT"YES"  
1DPC236  
1DTXT"NO"  
VAR22=FUN  
NIF  
IF(VAR22>1)  
1DCLR0  
VAR30=1  
NIF  
UNTIL(VAR30=0)  
1DCLR0  
REPEAT  
VAR30=0  
1DPC103  
1DTXT"ENTER TOTAL NUMBER OF CUTS: (1-100)"  
1DPC210  
1DTXT"TOTAL # OF CUTS=xxx"  
1DPC226  
VAR4=NUM  
IF(VAR>4100 OR VAR4<1)  
1DCLR2  
1DPC215  
1DTXT"OUT OF RANGE"  
T2  
1DCLR2  
VAR30=1  
NIF  
IF(VAR30=0)  
1DCLR0  
1DPC113  
1DTXT"# OF CUTS="  
1DPC124  
1DVO4,3,0,0  
1DPC202  
1DTXT"YES"  
1DPC236  
1DTXT"NO"  
VAR22=FUN  
NIF  
IF(VAR22>1)  
1DCLR0  
VAR30=1
```

```

NIF
UNTIL(VAR30=0)
1DCLR0
1DPC201
1DTXT"START"
1DPC235
1DTXT"EXIT"
REPEAT
  VAR22=FUN
  IF(VAR22<6 AND VAR22>1)
    1DCLR0
      1DPC212
      1DTXT"INVALID SELECTION"
      T2
      1DCLR0
  NIF
UNTIL(VAR22=6 OR VAR22=1)
IF(VAR22=6)
  XG23
NIF
1DCLR0
VAR5=VAR2*25000
VAR6=VAR3/60
VAR7=VAR4
LD3 MN A100 V(VAR6) D(VAR5) 1DPC211 1DTXT"# LEFT TO CUT:"
L(VAR4) 1DPC225 1DVO7,3,0,0
G VAR7=VAR7-1
T.1 N
1DCLR0 1DPC213 1DTXT"JOB COMPLETED!" T1.5 1DCLR0 XG23
XT

```

Short service sequences used to set up distances in the self learn mode.

```

XE33 XD33
VAR6=POS XT
XE34 XD34
VAR7=POS XT
XE35 XD35
VAR8=POS XT
XE40 XD40
D(VAR6) XT
XE41 XD41
D(VAR7) XT
XE42 XD42
D(VAR8) XT

```

## Section 7. PROGRAMMING

---

### Commands

The complete command list for the X150/X150E controller is contained in a separate document entitled **X150/X150E Software Reference User Guide** ( No. 1600.221.XX).

### Communicating with the Positioner

The positioner is designed to work in one of two modes:

- a) The human interactive mode
- b) The computer host mode

The command EX1 enables the first of these and the command EX0 the second. In the human interactive mode, the positioner will respond to a dumb terminal in an intelligent, user friendly manner and provide the user with help facilities and error messages if errors occur. In the host computer mode all unsolicited output and help information is suppressed so that communication may be handled by software resident in the host computer.

If a computer is to be used for the terminal function, terminal emulation software will be required. The X-Ware program available from Digiplan for use with IBM PC's and compatibles is suitable for this purpose. X-Ware is supplied with the product.

### Individual Commands

An individual positioner command controls a single parameter, function, or action such as acceleration, velocity, position, time delay, pause, loop, go, etc. There are two classes of individual commands, Immediate and Buffered.

Individual commands are variable in length. They can consist of one or more letters with a delimiter (carriage return or space character) and one or more letters and numbers with a delimiter. Each command is entered as a character/delimiter combination. Some commands include a sign ( $\pm$ ) to denote direction of motion. The number of characters used depends on the type of command entered.

Typical commands have the form:

**1E**  
**S**  
**A10**  
**V3**  
**D46000**

When two or more individual commands are entered on the same line, they are separated by spaces, and multiple command entries will be displayed on a single line of your terminal screen.

The example below shows a set of individual command entries with space delimiters on the same line:

**MN A10 V2 D25000 L10 G N**

If spaces are used as the delimiter and a large number of multiple command entries are made, you could exceed the ability of your terminal to display characters as a single line (80 characters per line is a typical value). When the carriage return is used as a delimiter, the cursor returns to the beginning of the next line.

**NOTE:** The Positioner will normally echo carriage return with carriage return line feed in terminal mode. If you get double line spacing you need to turn off the auto line feed facility on your terminal. In computer mode (EX0) only the characters received are echoed.

The example that follows shows the effect of using carriage return as the delimiter when the terminal is not in auto line feed.

**MN  
A10  
V2  
D25000  
L10  
G  
N**

### **Immediate Commands**

Commands that are identified as "Immediate" are executed immediately on receipt and will take priority over whatever operation is in progress. These commands include various Stop commands that clear the command buffer, and various Status Request commands that have no effect on the command buffer.

### **Buffered Commands**

Commands that are identified as "Buffered" are received by the positioner and stored (in the command buffer) if the positioner is not free to execute them. Stored commands are executed as soon as the positioner gets to them, in the order that they were received.

### **Buffer Capacity**

Any combination of individual commands and command groups can be entered in the buffer until the total number of characters currently stored (including the delimiters) equals 2000. The

positioner uses a first-in-first-out serial buffer. As the commands are read from the buffer, additional commands can be entered to replace them. Therefore, the possibility exists that a command set could actually consist of more than 2000 characters. Examples of command sets that might be used for different applications are presented later.

The sequence buffer is different from the command buffer in that it can be copied to battery backed up RAM using the SV command, while the command buffer is not battery backed up, so its contents are lost on power down.

### **Multiple Positioner Commands**

When multiple positioners are on the communications line, commands like the previous example are executed by all positioners on the line. To send commands to a single positioner, the device address of that positioner should be put in front of the instruction. The device address is set using the DIP switch on the individual positioner.

Example:

Three positioners are on an RS232C chain. They are sent the following commands:

**MN A10 V10 1D25000 2D50000 3D100000 G**

Unit 1 moves 25000 steps, unit 2 moves 50000 steps, and unit 3 moves 100000 steps. All three use the same move profile rates.

During sequence download to one axis in a chain, all of the other axes must have their communication switched off by use of the F command. The alternative is to make each command in the sequence device specific by adding the device address.

### **Programming Modes**

#### **Normal Mode**

The command MN selects normal mode. In this mode moves are set by the D command. Within normal mode there are two options, absolute mode and incremental mode:-

### Absolute Mode

The command MPA puts the positioner into absolute position mode. Distances entered using the D command are interpreted as absolute positions relative to absolute zero

Example:

<b>MPA</b>	Sets absolute mode
<b>PZ</b>	Sets current position to be absolute zero
<b>D4000</b>	Sets move to absolute position 4000
<b>G</b>	Execute move
<b>D2500</b>	Sets move to absolute position 2500
<b>G</b>	Executes move of 1500 steps CCW

### Incremental Mode

The command MPI puts the positioner into incremental positioning mode. D command distances are interpreted as the number of steps to move from the current position.

Example:

<b>1MPI</b>	Sets incremental mode
<b>1PR</b>	Response=1500 (current position)
<b>1D4000</b>	Sets move to be 4000 steps CW
<b>1G</b>	Executes move to position 5500
<b>1D-3000</b>	Sets move to 3000 steps CCW
<b>1G</b>	Executes move to position 2500

### Continuous Mode

The command MC selects continuous mode in which the motor runs continuously at the specified velocity until stopped or a new velocity is programmed. Distance data is ignored.

Example:

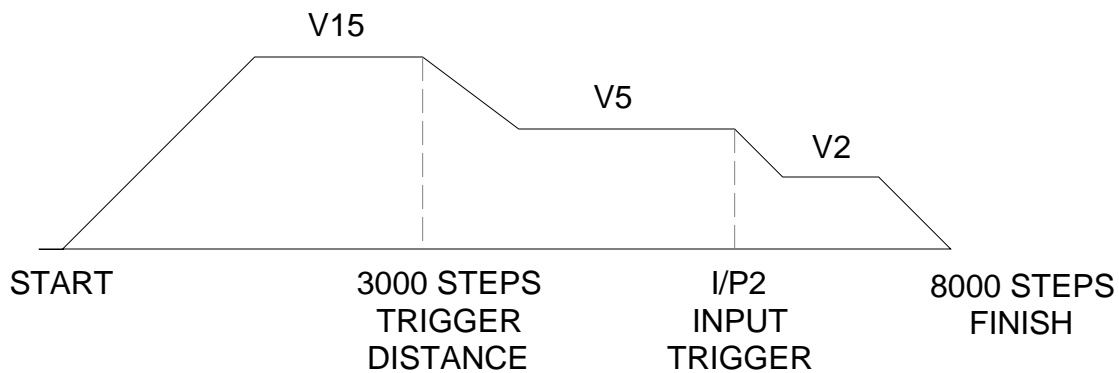
<b>MC</b>	Sets continuous mode
<b>D4000</b>	Ignored in continuous mode
<b>A50</b>	Sets acceleration to 50 revs/sec <sup>2</sup>
<b>V20</b>	Sets velocity to 20 revs/sec
<b>G</b>	Motor runs at 20 revs/sec continuously
.	
.	
<b>V10</b>	Motor speed changes to 10 revs/sec
.	
.	
<b>S</b>	Motor stops

### Speed Change Mode

The MQ command allows speed changes to be made during a preset move. The speed changes can be triggered at a set distance using the TRD command or by an input using the TRE command.

Example:

<b>3MQ</b>	Set MQ mode for axis 3
<b>3A50</b>	Set axis 3 acceleration to 50 revs/sec <sup>2</sup>
<b>V15</b>	Set velocity to 15 revs/sec
<b>D8000</b>	Set move distance to 8000 steps
<b>G</b>	Start move
<b>TRD3000</b>	Go to next command at D = 3000 steps
<b>V5</b>	Change velocity to 5 revs/sec
<b>TRE_X1</b>	Go to next command when Input 2 goes high
<b>V2</b>	Change velocity to 2 revs/sec



**Figure 7-1. Speed Change Mode**

### Command Types

#### Start Move Commands

The positioner reads and stores individual commands in the order they are entered. Each command is read and executed before the next command is read. The GO command "G" is the principal buffered command that initiates shaft motion. When the buffer reads a GO command, the motor will then move in accordance with the move parameters that reside in the positioner at the time the Go command is read.

When the Go command is read, the positioner will incorporate any motion parameter command that was encountered prior to the Go command. For example, the "A" command would be combined with the Go command to change the acceleration value as follows:

### **A123 G**

These commands instructs the Positioner to change the acceleration value to 123 rev/sec/sec and then execute the move using the last mode, velocity, and distance values that were previously entered.

When the RS232C positioner is initially activated, a series of commands of the form " MN An Vn Dn G " is required. These command sets up the initial operating conditions for the positioner. After the positioner has been initialised, single move parameter commands can be used.

If only "G " is entered, the positioner will repeat the previous motion pattern. For example, a string of Go commands:

**G G G G**

would instruct the positioner to move the motor the same way four times.

There is no need to re-enter all the shaft motion commands to change one of the variables. If one or more of the motion parameters is changed by a command entry and the Go command is then entered, the prior pattern will be repeated using the new motion parameter(s) that were entered prior to the latest GO command.

For example:

**A14 G V2.6 G D-27634 G**

would change acceleration to 14 and move, then change velocity to 2.6 and move with 14 as the acceleration value, then change position and move again using 14 and 2.6 as the acceleration and velocity values.

The CONTINUE command "C " is used following execution of a PAUSE command, "PS " or "U ". CONTINUE will allow execution of the next command waiting in the command buffer (if any command is stored).

#### **Loop Commands**

The LOOP command allows a cycle to be repeated continuously "L" or a given number of times "Ln" up to 65535. The END-OF-LOOP command "N" indicates where the loop ends. The END-OF-LOOP command can be used to indicate that the positioner should proceed with further commands after the designated numbers of loops have been executed, or in combination with the "Y" command to indicate where execution is to stop. The "U" command may be used to temporarily halt Loop execution, the C command will then cause the loop to resume execution.

**Stop Move  
Commands**

The STOP MOVE commands are presented in order of severity of response. SOFTWARE RESET "K" command is the most abrupt and severe stop command you can use. The STOP AT END OF CURRENT LOOP command "Y" is the least abrupt and severe stop command

***Stop at End of  
Current Loop Y***

Also the 'stop sequence' command.

This command will not halt processing until command processing reaches the character N at the end of the command loop. At that time, the Positioner will execute the next command in the buffer after N, if any. The command loop cannot be restarted without re-entering the commands.

***Controlled Stop  
command S***

In the PRESET or CONTINUOUS mode this command will decelerate the motor to a stop at the last used acceleration rate.

An "S " command will always cause a deceleration to velocity zero at the last used acceleration.

The "S " command clears any remaining commands in the command buffer unless prevented from doing so via the SSH command.

**NOTE:** Normally, the motor is decelerated to a stop at the same rate as it was accelerated.

***Controlled Stop  
command LS***

The LS or 'Limit Stop' command will decelerate the motor to a stop at the deceleration rate defined by the LA or 'Limit Acceleration' command.

***Kill Command K***

This command stops positioner commands to the motor. In addition it terminates a loop, ends a time delay, and aborts a sequence download icommand (XD command). The command buffer is also cleared.

### **Software Power-on Reset Z**

The "Z " command is equivalent to cycling the AC power to the Positioner; that is, it disables the communications interface and returns all internal settings to their power-on values. The command buffer is cleared. Like the "K " command, "Z " causes an immediate cessation of torque signal to the motor. Z also de-energises and resets the drive ready for re-initialisation at power up.

**NOTE:** When the "Z " command is used, the Positioner is busy for up to 2 sec and will ignore any commands. The status indicator will be blank for this period.

### **Pause Commands**

There are four types of pause commands available as follows:-

Pause and wait for continue command C to continue. A common use of the Buffered Pause PS is to hold execution of individual commands in a Command Loop until the entire loop has been loaded.

### **Tnnn**

Pause in processing commands for a given number of seconds and then continue.

### **U**

Hold (pause) now and wait for continue. This is an immediate command.

### **TRE**

Pause in processing commands until the designated TRIGGER condition is met.

Triggers are used to synchronise positioner operations with external events. They can be used to implement a "handshaking" function with other devices. See command details for exact syntax.

### **Status Request Commands**

All status request commands result in data being returned to the controller from the positioner. To prevent multiple positioners from all responding at once, the status request commands are given the classification "Device Specific" meaning that the device address of the responding positioner must be placed in front of the command. No positioner will execute a Device Specific command without its device number.

The use of status request commands must be conducted in an orderly fashion. Commands should only be issued when the host is ready to read the response. New commands should not be sent until the response is received. In particular, after a buffered status command, an immediate status command should not be sent until the response has been received by the

host. If this procedure is not followed, the command responses will be intertwined, rendering the information useless.

There are two status commands that can be used to request position.

The DPA Command will report how many steps the motor has moved relative to the absolute zero position. This position is calculated by summing the total number of moves commanded since the positioner was at the absolute zero position. If position reports are needed that are relative to the beginning of each individual move, the PR command should be used.

### Using the Evaluation Commands

The IF, WHILE and UNTIL commands contain fields that are evaluated to provide conditional program flow. The evaluation statement is contained in parentheses after the IF, WHILE and UNTIL commands. A space or underscore (\_) indicates each comparison.

```
IF(condition)..commands..ELSE..commands..NIF
REPEAT..commands..UNTIL(condition)
WHILE(conditional)..commands..NWHILE
```

The conditions take the form of a set of evaluation statements:

1. INnnnnn\_nnnnnnnnnn: compare to input status

```
LMT+ LMT- HOME AUX_IN STOP _
IN1 IN2 IN3 IN4 IN5 IN6 IN7 IN8 IN9 IN10
```

2. Variable compares:

```
VARn>VARn
VARn=VARn
VARn<VARn
```

A constant, POS (the present value of the position counter), ABS (the present value of the absolute encoder) or FEP (follower encoder position) can be substituted for the variable.

3. AND: Boolean AND operation
4. OR: Boolean OR operation

*Example*Command**VAR4=4000****IF(IN\_10 OR VAR1=3)****XR1****ELSE****XR2****NIF****REPEAT****VAR5=POS****XR5****T5.0****H****UNTIL(VAR5>VAR4 AND IN\_00)****VAR5=POS****VAR6=ABS****WHILE(VAR6>VAR5 AND IN\_00)****XR7****T2.0****NWHILE**Description

Set variable 4 to 4000

If input 1 is active and input 2 inactive or the user variable 1 equals 3 then execute sequence 1, else execute sequence 2

Execute sequence 1

ELSE command

Execute sequence 2

End of IF

Repeat command

Set variable 5 to current indexer position

Execute sequence 5

Delay 5 seconds

Change direction

If inputs 1 and 2 are inactive and the position counter is greater than VAR4, execute the next command. Else, jump to the repeat command

Set variable 5 to current indexer position

Set variable 6 to the absolute encoder position

If the absolute position encoder is greater than VAR5 and inputs 1 and 2 are inactive, execute the following commands, else execute the command following the NWHILE command

Execute sequence 7

Delay 2 seconds

End of WHILE - jump to WHILE command

**Drive Energise/De-energise Function**

Energising and de-energising the drive is subject to the commands given and error status as follows:

*The drive will energise if:*

The ST0 or ON command is given when the WATCHDOG signal is inactive.

*The drive will de-energise if :*

The ST1 or OFF command is given.

The WATCHDOG signal become operative.

The de-energising signal must be restored, then the ST0 or ON command must be given to re-energise the drive.

**Homing Function**

The GH command causes the motor to rotate in the direction and at the speed specified until the home position is reached. In practice, it continues a little beyond this point but you can locate the home position accurately by then using the commands:

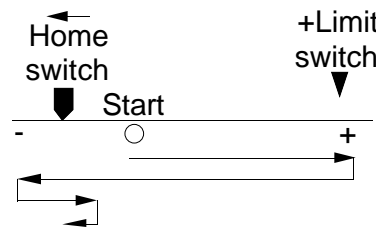
**MPA D0 G**

(The GH command always makes the home position zero)

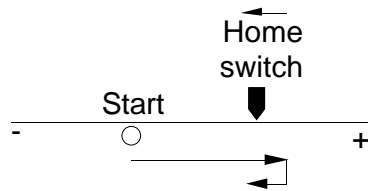
The SP command can be used after homing to change the home position from zero if required.

If the home position is not found in the direction given in the GH command, the limit switch position will be reached and the switch will be operated. The motor will then reverse and seek the home position in the opposite direction. For recognised home detection, the home switch must always be activated from the direction specified in the command (shown by the arrow in Figure 7-2), so as illustrated in this figure, a limit switch reversal will take place if the direction given in the GH command is away from the home position (GH+ in the illustration).

Positive homing from + side of home switch:-

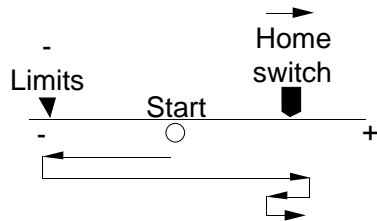


Positive homing from - side of home switch:-

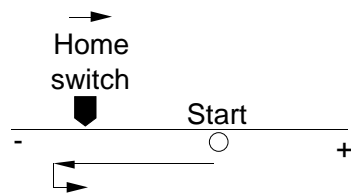


**Figure 7-2. Positive Homing**

Negative homing from - side of home switch:-



Negative homing from + side of home switch:-



**Figure 7-3. Negative Homing**

### Basic Programming Guide

This section uses examples of simple programs as an introduction to programming and explains the use of the command language.

The system configuration is a stepper motor. The address select switch has been set for unit 1. The motor starting position is 8000 steps CW from the home position.

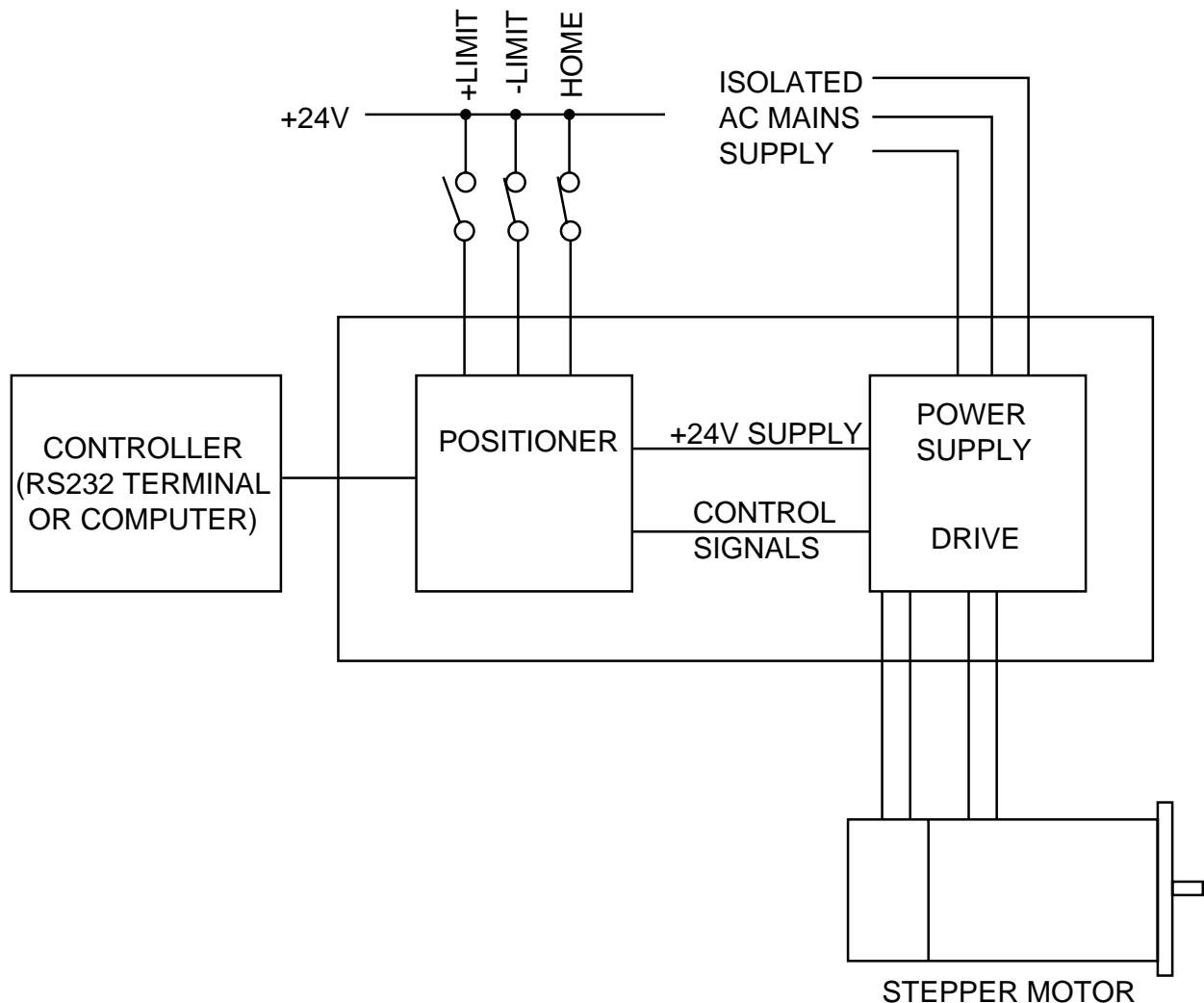


Figure 7-4. System Used in Example Programs

#### Example 1

In this simple example the requirement is to cause the motor to perform two revolutions in a CW direction and stop.

## 84 PDHX-E SERIES DRIVE USER GUIDE

---

The following commands in the order shown will fulfil the requirement:-

<u>Command</u>	<u>Purpose</u>
<b>MN</b>	Set move mode to normal
<b>1A10</b>	Set acceleration motor 1 = 10
<b>1V5</b>	Set velocity motor 1 = 5
<b>1D8000</b>	Set move distance motor 1 = CW 2 revs
<b>1G</b>	Start move - motor 1 moves 2 revs CW

### *Description*

Commands cause motor 1 to accelerate at 10 revs/sec<sup>2</sup> to a velocity of 5 rps, travelling CW for 8000 steps (2 revs) at the G command.

### **Example 2**

This example causes axis 1 to go to the home position, then turn 5 revolutions CCW at 10 revs/sec and set the resulting position as absolute zero. The motor finally turns 8 revolutions CW to finish at absolute position 1600.

<u>Command</u>	<u>Purpose</u>
<b>ST1</b>	Energise the drive
<b>A10</b>	Set acceleration to 10 revs/sec <sup>2</sup>
<b>V10</b>	Set velocity to 10 revs/sec
<b>GH-2</b>	Send axis 1 to go home (2 revs/sec CCW)
<b>1D</b>	Set axis 1 to go 5 revs CCW
<b>G -20000</b>	The axis moves to the required absolute position
<b>MPA</b>	Set absolute position mode
<b>PZ</b>	Set position to absolute zero
<b>1D32000</b>	Set move to abs. position 1600 (8 revs CW)
<b>G</b>	Move to position 1600 (1 rev CW from starting position)
<b>1PR</b>	Report absolute position

### *Description*

Commands 2 - 6 cause the motor to go to the home position and then turn 5 revolutions CCW. Command 7 sets the absolute positioning mode and Command 8 sets the resulting position as absolute zero. Commands 9 and 10 cause the motor to turn 8 revolutions CW to absolute position 32000.

## Programmable I/O Functions

### Programmable Inputs

You can program each input or define them to correspond to a function. There are 15 functions that can be defined by the user. To define an input to a particular function use the IN command. The user definable inputs are labeled IN1 to IN10 on the front panel. The remaining inputs are dedicated as Limit+, Limit-, Home, Stop and Aux-In.

All input functions have been assigned a letter - refer to the IN command in the software section.

To program an input to perform a function, use the IN command followed by the input number and the corresponding function letter.

For example

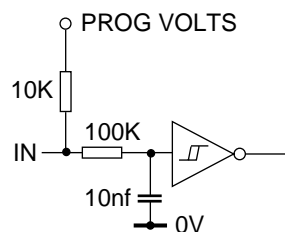
**1IN7H** Set input 7 as direction toggle.

The active input level and the type of input (pull-up or pull-down) can be programmed using the ICON command.

To view the current status of the user definable inputs type IN. All user definable inputs will be displayed.

### Programmable Input Levels

Inputs can be programmed to operate as pull-down (sink) inputs or pull-up (source) inputs. The pull-up voltage may also be programmed as 5V or 24V. This is achieved by using the circuit arrangement shown in Figure 7-5, where PROG VOLTS can be programmed to 0V (pull-down), 5V or 24V (pull-up).



**Figure 7-5 Input Circuit**

Inputs are configured in 3 banks of 5 inputs, allowing each bank to be programmed independently. The three banks are:

<u>BANK 1</u>	<u>BANK 2</u>	<u>BANK3</u>
IN1	LMT+	IN6
IN2	LMT-	IN7
IN3	HOME	IN8
IN4	STOP	IN9
IN5	AUX_IN	IN10

If you need further information refer to the ICON command.

### **Verifying Input Wiring**

To verify that you have wired the inputs correctly for your application use the IS command. This command reports the active state of all inputs regardless of the function assigned to them.

<u>Command</u>	<u>Response</u>
1IS	000000_0000000000

This indicates that no inputs are active.

The voltage level, sink or source operation for the inputs can be configured using the ICON command. However the IS command will indicate the status of any input regardless of the input configuration.

<u>Command</u>	<u>Response</u>
1IS	000000_1000000000

This indicates that input 1 is active.

### **Set Up Example**

The limit inputs are to be connected to PNP normally closed proximity switches. The Home switch is connected to a normally open PNP proximity switch. Input 4 is to be configured as a kill input active when 0V is applied to the input. Inputs 6,7 and 8 are sequence select lines and input 9 is a sequence strobe line which is connected to the PNP output stages of a PLC. The stop input is active when 0V is applied to the input. All inputs operate at 24V dc.

This is achieved using the ICON command, as follows:

**ICON 11111110**

All bank 1, 2 and 3 inputs are programmed to +24V DC sense, active low. Vs is set to +24V and a motor mounted encoder (if used) is connected to the 'Primary Encoder' connector.

Configure the inputs as follows :

1IN4C	Configure input 4 as kill input
1IN6N	Configure input 6 as a data input
1IN7N	Configure input 7 as a data input
1IN8N	Configure input 8 as a data input
1IN9N	Configure input 9 as a data input
1IN10W	Configure input 10 as a sign bit
1OUT3J	Configure output 3 as a strobe output
1OUT4J	Configure output 4 as a strobe output
1OUT5J	Configure output 5 as a strobe output

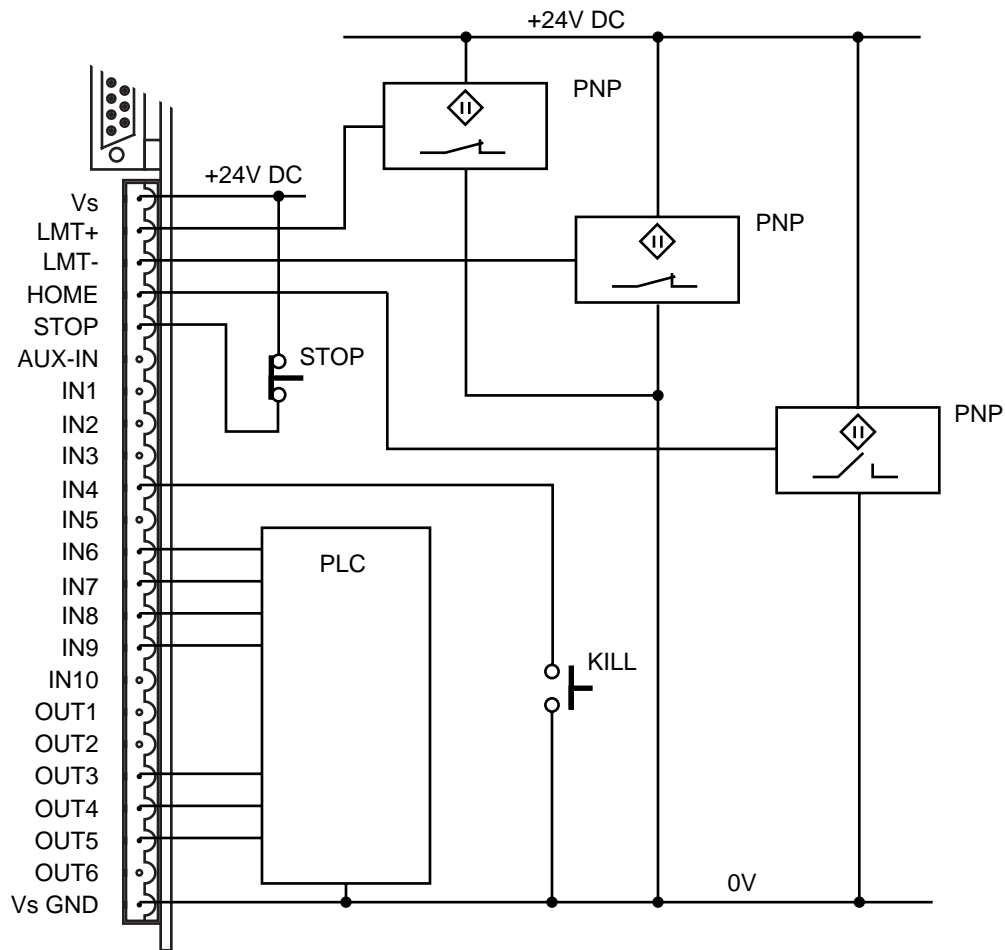


Figure 7-6 Input

### **Programmable Output**

You can program each output, or configure it, to perform to a particular function. There are 6 user definable outputs which can be programmed as either NPN or PNP open collector outputs.

To allocate an output to a particular function use the OUT command. The user definable outputs are labelled OUT1 to OUT6 on the front panel. All output functions have been assigned a letter, refer to the OUT command in the software section.

To program an output to perform a function, use the OUT command followed by the output number and the corresponding function letter.

For example

**1OUT3T**      Set output 3 as in position.

The choice of NPN or PNP output can be programmed using the OCON command.

To view the current status of the user definable output type 1OUT. All user definable outputs will be displayed.

The response of the output to an active level will depend upon the programmed state of that output.

Note: Although it has been stated that all output functions can be defined using the OUT command there is one exception to this:

OUTPUT 1 can be configured as a composite fault output using the SSD command. When SSD is set to 1 then output 1 is configured as a composite fault output and overrides the function defined by OUT. SSD must be set to 0 for output 1 to be configured using the OUT command.

### **Programmable Output Levels**

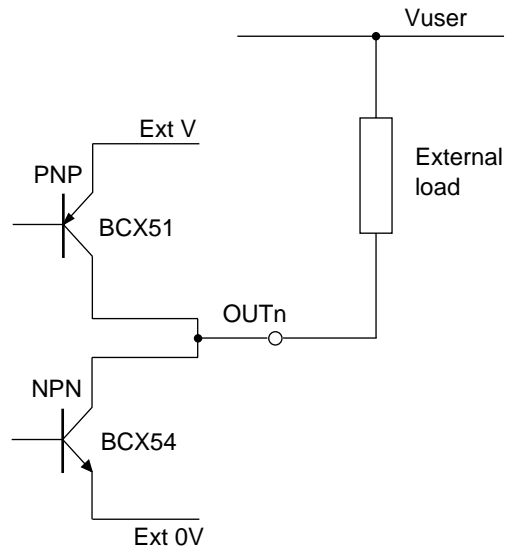
Outputs can be programmed as NPN or PNP open collector outputs. The circuit arrangement shown in Figure 7-7 is used. Output voltage levels will be determined by the value of the external voltage (Vs) and the value of the connected load.

The selection of NPN or PNP outputs is made using the OCON command. For example:

**1OCON000111**

Sets OUT1-3 to be NPN and OUT4-6 to be PNP. Only 6 bits can be set using this command.

Note: the output circuit uses a connected pair of output transistors, consequently when the NPN transistor is used, but is turned OFF, the external user voltage ( $V_{user}$ ) must be no greater than  $V_s$ .



**Figure 7-7 Output Example**

#### Verifying Output Operation

You can directly control each output using the output command 0. Using this command you can force any output to the desired state, in order to do this the output must be configured as a programmable output (function A).

	<u>Command</u>	<u>Description</u>
> 101X0		Turn output 1 on, ignore output 2 and turn off output 3.

#### Using Programmable Outputs

Programmable outputs can be used to interface with other parts of the system in order to indicate such things as in position, sequence in progress, fault or a user defined function.

The following steps demonstrate how you can configure these outputs.

### Example

Output 1 is to be configured as a composite fault output. Output 3 is to indicate when the motor is moving and Output 5 is to indicate when the motor is in position.

<b>1SSD1</b>	Set output 1 as composite fault
<b>1OUT3B</b>	Set output 3 as moving / not moving
<b>1OUT5T</b>	Set output 5 as in position output

Typing 1OUT will display

```
;Output 01 Function F  
;Output 02 Function A  
;Output 03 Function B  
;Output 04 Function A  
;Output 05 Function T  
;Output 06 Function A
```

## Sequences

### Introduction

A sequence is a series of commands that is executed in order whenever the sequence is run. Immediate commands cannot be stored in a sequence since they cannot be stored in the command buffer. *The only "saveable in sequence" commands are buffered commands.*

The positioner has a section of non-volatile RAM totalling 8000 characters allocated for storing up to 64 sequences. The sequence buffers are of variable length, so they can store long sequences of up to 8000 characters or several shorter ones, provided that the total length of all sequences does not exceed the allocated 8000 character space.

During normal operation a command can specify the execution of any sequence stored in memory.

Sequence identifiers: number from 1 - 64.

When a sequence is programmed, the positioner automatically calculates a checksum for the sequence and stores the length of the sequence and its checksum in an internal directory.

The command XSS can be used to verify the existence of a sequence.

If a power-on sequence is used, the sequence and its sequence number are stored in the non-volatile RAM.

### Programming Sequences

Sequences are programmed by first sending a sequence definition command XD, which places the commands following XD in the sequence buffer. This continues until an end of sequence command (XT) is received. Any condition which prevents proper recording of the sequence is saved and can be accessed with the status command XSD. The programmed sequence can be tested with the run sequence command XR, which executes any specified sequence.

Note: A sequence cannot be defined whilst there are buffered commands still waiting to be executed i.e., while the drive is in an active state.

The XU command can be used to read back the entered sequence.

To begin the programming of a sequence, the XD command immediately followed by sequence identifier number (1 to 64) and a delimiter must be entered. An XT command ends the sequence. The save command SV should be used to store the sequence into the positioner's non-volatile memory.

The commands entered after the XD command and before the XT command will be executed in the order in which they were entered when the sequence is run.

Example:

**NOTE:** The ">" prompt is replaced by the "\$" prompt when in sequence edit mode.

<b>XD1</b>	Begin definition of sequence No.1
<b>A10</b>	Set acceleration to 10 revs/sec/sec
<b>V10</b>	Set velocity to 10 revs/sec
<b>D4000</b>	Set move distance to 4000 steps
<b>G</b>	Go
<b>H</b>	Reverse next move
<b>G</b>	Go
<b>XT</b>	End of sequence

\* The definition of sequence No.1 is terminated by XT but the sequence is NOT stored in non-volatile memory until the save command SV is given.

**NOTE:** Turn off switch 8 (on top of the drive) to write-protect sequences.

Whenever the above sequence is run, the motor will turn 1 revolution CW and then 1 revolution CCW.

### Running Sequences

#### *By Command Input*

A sequence can be run by entering the XR command immediately followed by a sequence identifier number (1 - 64) and a delimiter.

Example:

XR3[space] runs sequence No.3

#### **Standalone Operation**

Stored sequences may be automatically executed when you power up the system or executed by remote switches, RP240, TM8 and thumbwheel.

A sequence selected as a binary pattern, set on inputs using the INnO command can be run in response to a sequence strobe set on an input using the INnB command.

#### **Example**

Inputs 1 to 4 are to be used as sequence select lines.

Input 5 is to be used as a sequence strobe line.

Programs have been stored in sequences 1 to 16.

1IN1O	Set input 1 as sequence data input
1IN2O	Set input 2 as sequence data input
1IN3O	Set input 3 as sequence data input
1IN4O	Set input 4 as sequence data input
1IN5B	Set input 5 as sequence strobe input.

A binary pattern is configured on the inputs corresponding to the decimal sequence number. On a logic 0 to 1 transition of the sequence strobe input, the sequence corresponding to the binary pattern will be executed.

If a sequence is already running when the sequence strobe is activated the sequence strobe will be ignored. A sequence in progress must be terminated before execution of the next sequence strobe.

#### **Power On Sequence**

One of the stored sequences can be designated to be executed when the device is powered up. The sequence to be executed can be selected by prior RS232C command (XP1-64). The single sequence specified is executed once; control then passes to the normal command processor loop. Alternatively, power-on execution can be disabled with either the XP0 or XZ commands.

It is possible to have the positioner execute a sequence on power up by defining the commands within a sequence using the XD, XT and XP commands. To define a sequence you will need to enter:

**XP1 XE1 XD1 MN A10 V5 D40000 G XT 1SV**

In this example the operator would not power the unit on until ready for the sequence to begin. The XP1 command enables a single run of sequence No.1 after power on. The XD1 command signifies the start of the definition of sequence No.1 and the XT command signifies the end of the sequence No.1 definition. When the operator applies power to the positioner, sequence number 1 will be executed (the motor will turn 10 revolutions in the positive direction). The sequence can be stopped using the S command or, if necessary, by using the emergency stop (K command).

***Changing a  
Programmed  
Sequence***

Once a sequence is defined it cannot be redefined until it has been deleted. A new sequence cannot be downloaded over an existing one.

To change a sequence, it must be deleted from memory and then redefined as a new sequence using the same number.

First, the sequence must be deleted from memory using the XE command. It will not be deleted from back-up RAM unless the SV command is used after the XE.

Example:

**XE1** erases sequence No.1

The changed sequence should then be entered. It will not be saved to non-volatile memory until the SV command is issued.

***Examining the  
Contents of a  
Sequence***

If the operator wishes to check the contents of the sequence, he would enter:

**1XU1**

This would cause the positioner to send the contents of sequence number 1 to the terminal's screen. The 1 preceding the XU command is the device address which must be present since the XU command is a "device specific" command. After issuing this command the following will be displayed on the terminal's screen:

**;Sequence 1 is: MN A10 V5 D40000 G**

This is the command string that is performed when sequence number 1 is executed.

If the operator does not wish to have the sequence execute on power up, it can be cancelled by issuing either the XP0 or XZ command.

Sequence no.1 can also be run manually using the XR command.

### ***Sequence Status Requests***

Sequence status commands include XC, XSD, XSP, XSR, XSS. The XC command can be used to verify that the memory in the back-up RAM has not been corrupted in any way since last power down. The XC command will cause the positioner to send a checksum in the form of three decimal digits (000 - 255) to the terminal. The checksum may be recorded when sequences have been defined, so that it may be used for comparison later, such as each time the positioner is powered up.

### **Sequence Debugging Tools**

After creating your sequences, you may need to debug them to ensure that they are performing properly. The Controller provides several debugging tools.

- In trace mode, you can trace a sequence as it is executing
- You can single step through a program
- Error messages can be enabled explaining why the Controller has stopped execution

### ***Trace Mode***

You can use the trace mode to debug a sequence or a program of sequences. The trace mode allows you to track, command by command, the entire sequence. It displays to your terminal, over the RS-232C serial link, all of the commands as they are executed. The following example demonstrates the trace mode.

### ***Step 1***

Create the following sequence:

<u>Command</u>	<u>Description</u>
<b>XE1</b>	Erases sequence #1
<b>XD1</b>	Begins definition of sequence #1

<b>A10</b>	Acceleration is 10 rps <sup>2</sup>
<b>V5</b>	Velocity is 5 rps
<b>L5</b>	Loop 5 times
<b>GOSUB3</b>	Jump to sequence #3
<b>N</b>	Ends the loop
<b>XT</b>	Ends the definition of sequence #1

**Step 2**

Define the sequence #3.

<u>Command</u>	<u>Description</u>
<b>XE3</b>	Erases sequence #3
<b>XD3</b>	Begins definition of sequence #3
<b>D50000</b>	Sets the distance to 50000 steps
<b>G</b>	Initiates motion
<b>XT</b>	Ends the definition of sequence #3

**Step 3**

Enter the following command to enable the Trace mode.

<u>Command</u>	<u>Description</u>
<b>1XTR1</b>	Enable the trace mode

**Step 4**

You will now execute the sequence. The commands will be displayed on the terminal as each command in the sequence is run. Enter the following command:

<u>Command</u>	<u>Description</u>
<b>XR1</b>	Run sequence #1 - response is:

```
*SEQUENCE 001 COMMAND A10
*SEQUENCE 001 COMMAND V5
*SEQUENCE 001 COMMAND L5
*SEQUENCE 001 COMMAND GOSUB3 LOOP COUNT 1
*SEQUENCE 003 COMMAND D50000 LOOP COUNT 1
*SEQUENCE 003 COMMAND G LOOP COUNT 1
*SEQUENCE 003 COMMAND XT LOOP COUNT 1
*SEQUENCE 001 COMMAND N LOOP COUNT 1
*SEQUENCE 001 COMMAND GOSUB3 LOOP COUNT 2
*SEQUENCE 003 COMMAND D50000 LOOP COUNT 2
*SEQUENCE 003 COMMAND G LOOP COUNT 2
*SEQUENCE 003 COMMAND XT LOOP COUNT 2
*SEQUENCE 001 COMMAND N LOOP COUNT 2
*SEQUENCE 001 COMMAND GOSUB3 LOOP COUNT 3
*SEQUENCE 003 COMMAND D50000 LOOP COUNT 3
.
.
.
```

```

.
.
*SEQUENCE 003 COMMAND G LOOP COUNT 5
*SEQUENCE 003 COMMAND XT LOOP COUNT 5
*SEQUENCE 001 COMMAND N LOOP COUNT 5
*SEQUENCE 001 COMMAND XT
    
```

The format for the trace mode display is:

Sequence Number	Command	Loop Count
-----------------	---------	------------

**Step 5**

To exit the trace mode, enter the following command:

<u>Command</u>	<u>Description</u>
<b>XTR0</b>	Exits Trace mode

**Single-Step Mode**

You can debug your program with another level of debugging using the Single-Step mode. Single-Step mode allows you to execute one command at a time when you want the command to be executed. Use the XST command to enable Single-Step mode. Once you are in the mode, you can execute a sequence one command at a time. To execute a command, you must use the # sign. By entering a # followed by a delimiter, you will execute the next command in the sequence. If you follow the # sign with a number (n) and a delimiter, you will execute the next n commands. The following steps illustrate the use of Single Step mode:

**Step 1**

Enter Single-Step mode

**XST1**

**Step 2**

Begin execution of sequence #1

**XR1**

**Step 3**

You will not execute any commands until you use the # command. Type the following:

<u>Command</u>	<u>Description</u>
<b>#</b>	Executes one command

The response will be:

```
*SEQUENCE 001 COMMAND A10
```

**Step 4**

To execute more than one command at a time, follow the # sign with the number of commands you want executed.

**Command****Description****#3**

Executes 3 commands, then pauses sequence execution

To complete the sequence, use the # sign until all the commands are completed. To exit Single-Step mode, type:

**XST0**

