

Chapter 8 Linear And Circular Interpolation

This chapter describes the linear and circular interpolation features of the ACR1000 motion control system. Linear and circular interpolation is possible for up to 16-axes. This feature is included in versions 2.7.x and 4.x only. In addition to the correct software, the ACR1000 card needs a cable to go between all the interpolating cards. This cable provides the necessary synchronization signals that make sure all the cards are moving as they are supposed to.

Linear Interpolation

Linear interpolation can be mixed with independent axis commands as desired by the programmer. To make an axis do interpolation, the axis must be told two things: 1) the destination and 2) the longest move (in pulses) any axis will move during the coordinated move.

In a multiaxis system, the cards may either run a stored program or execute one move at a time. First, let us look at an example on a 2-axis system. When the program is run, it is desired that the two axes make a 22.5 degree move with x-axis ending at 10 inches and y-axis ending at 2.5 inches. The velocity is to be 100 inches per minute. Assume that there are 10000 pulses in an inch.

X Axis Program	Y Axis Program
10 ACC 0	10 ACC 0
20 DEC 0	20 DEC 0
30 STP 0	30 STP 0
40 VEL 10000	40 VEL 10000
50 P28=100000	50 P28=100000
60 MOV 100000	60 MOV 25000

In the above example, the Y-axis will move at 1/4th the speed of X-axis. Note that this method results in a 32-bit precision velocity range to both cards.

P28 holds the number of pulses in the longest move to be made by any axis. This parameter is used to control how much the slower cards (the ones with less distance to travel) will scale down their velocities. If P28 is set to zero, then all moves will take place in a non-interpolation manner—thus this method yields the choice of either interpolation or non-interpolation movements.

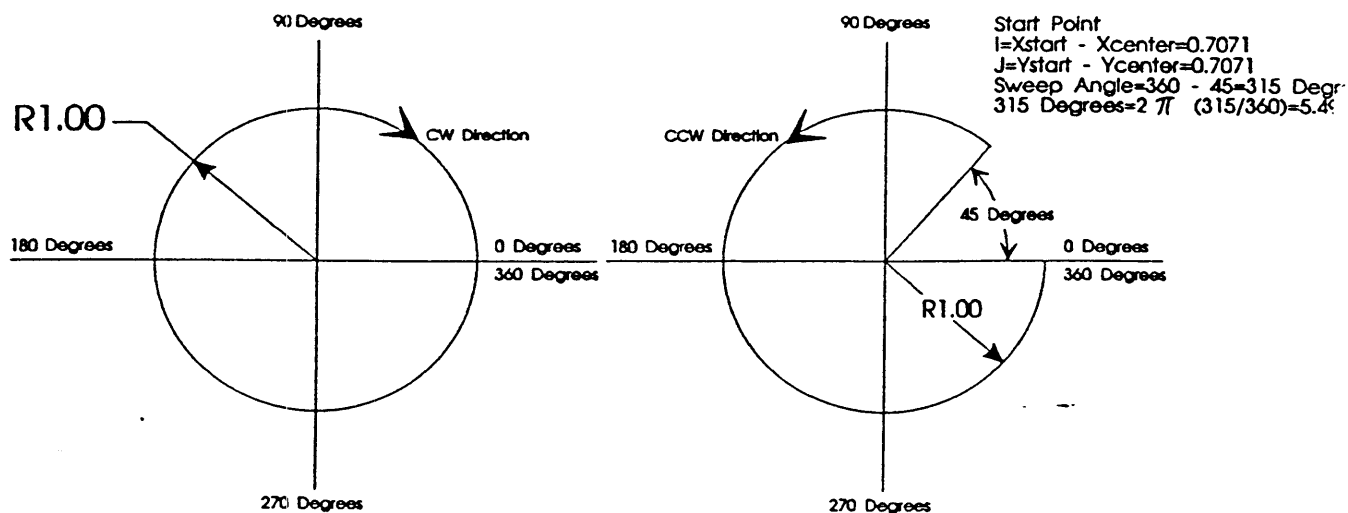
Circular Interpolation

This section could be called sinusoidal motion since that would be a true representation of the motion of each axis. The ACR1000 card has built-in provisions for allowing an axis to generate sinusoidal motion starting at any point and traveling any desired number of pulses. On an X-Y table, if one axis is commanded to perform a sine and another a cosine, the result is a perfect circle. Similarly, if the x-axis is commanded to perform a sinusoidal move and the y-axis is given a linear move, the result will be a perfect sine wave.

The command to do sinusoidal or circular motion is a little more detailed than the linear case. The axis still requires knowing how many pulses to go, but also needs to know the radius, direction of travel (clockwise or counter clockwise), precisely where the start point is in relation to the center of the arc and the absolute position at the end of the arc or circle. To use the sine-wave terminology, we need the following:

1. Peak Amplitude
2. Direction
3. Start Amplitude
4. End Position

This information is required for both the sine and the cosine wave axes. The reason the start amplitude of the sine as well as cosine is needed is that, in general, each sine or cosine amplitude repeats twice for each 2π period—but each *pair* of amplitudes happens only once. Thus to completely define the start point, both the amplitudes must be given. In machine tool language, these amplitudes are known as I and J.



CIRCLE COMMAND FORMAT

The format of the CIRCLE command is:

CIRCLE *rot_dir*, *radius*, *I*, *J*, *distance*

Parameters may be used for any circle command arguments. The *rot_dir* argument indicates the direction of rotation. Use +1 for CW, -1 for CCW. The *radius* is the radius (or amplitude of the sine wave) of the circle in pulses. The *distance* parameter is the circumferential distance from the start point to the end point. If the circle is a 360 degree arc then the distance in pulses is $2\pi \cdot \text{radius}(\text{units}) \cdot \text{pulses/unit}$. A simple way to calculate the distance is to determine the sweep angle between the start point and the end point in radians. Then multiply by the radius expressed in pulses. To convert degrees to radians multiply by π and divide by 180 ($2\pi/360$). *I* & *J* indicate where on the circumference of the circle or arc the current position of the axis represents, relative to the center of the circle (0,0). To generate elliptical motion, use different values for the primary and secondary radii. (See section on elliptical interpolation for more details.) If (Xstart,Ystart) is the start coordinate of the circle/arc, then the following can be used.

X axis

$I = (X_{\text{start}} - X_{\text{center}})$

$J = (Y_{\text{start}} - Y_{\text{center}})$

-or-

$I = \text{Radius} \cdot \sin(\text{Angle})$

$J = \text{Radius} \cdot \cos(\text{Angle})$

Y axis

$I = (Y_{\text{start}} - Y_{\text{center}})$

$J = -(X_{\text{start}} - X_{\text{center}})$

$I = \text{Radius} \cdot \cos(\text{Angle})$

$J = -\text{Radius} \cdot \sin(\text{Angle})$

Once the *I*,*J* values for the X-axis are calculated, the Y-axis *I* & *J* are related like this: $YI = XJ$; $YJ = -XI$.

In addition to the command arguments, the following parameters must be loaded with correct values for each CIRCLE move:

P28: Maximum move distance of all axes moving together. This value must be the same for all axes.

P29: Position the axis will be at when the move is completed. This number, unlike *I* & *J* is absolute and referenced to system zero. To determine this value, use the following.

X-axis: $P29 = \text{start position} + (\text{rot_dir} \cdot (\text{lend} - \text{lstart}))$

Y-axis: $P29 = \text{start position} + (\text{rot_dir} \cdot (\text{Jend} - \text{Jstart}))$

where

$\text{lstart} = \text{radius} \cdot \cos(\text{StartAngle})$, $\text{Jstart} = \text{radius} \cdot \sin(\text{StartAngle})$

$\text{lend} = \text{radius} \cdot \cos(\text{EndAngle})$, $\text{Jend} = \text{radius} \cdot \sin(\text{EndAngle})$

All positions are in pulses.

Circular Interpolation Example

As an example, we will see how to generate a 360 degree SINE wave starting at 0 degrees and ending at 360 degrees. The amplitude (*radius*) is 1 inch (10000 pulses). Note the value that *I* and *J* can take is such that $0 \leq I \leq R$. and $0 \leq J \leq R$, where *R* is the *radius*. Assume 1 inch=10000 pulses.

Initially the axis is sitting at 0. The *radius* or amplitude is 10000. At 0 degrees $I = 10000$ and $J = 0$. Since we wish to go from 0 degrees to 360 degrees, the direction is CW (+1).

```

rotdir is +1
radius is 10000
I is 10000
J is 0
distance is 62832 (this is  $2\pi \cdot 10000$ )
    
```

Plugging in the values, the final command is:

```

P28=0
P29=0
CIRCLE +1,10000,10000,0,62832
    
```

Note that prior to the CIRCLE command, P28 and P29 must be set. P28 is loaded up with the maximum number of pulses any axis is moving. In this case since only one axis is moving P28 should be set to 0. P29 must be loaded with the end point of the axis. In this example, since the axis was at 0, it will end up at 0 after going a complete 360 degrees.

In an x-y table application, if two axis were required to do the complete circle, the following programs would be sufficient.

X-AXIS PROGRAM

```

10 P28=62832
20 P29=0
30 CIRCLE 1,10000,10000,0,62832
    
```

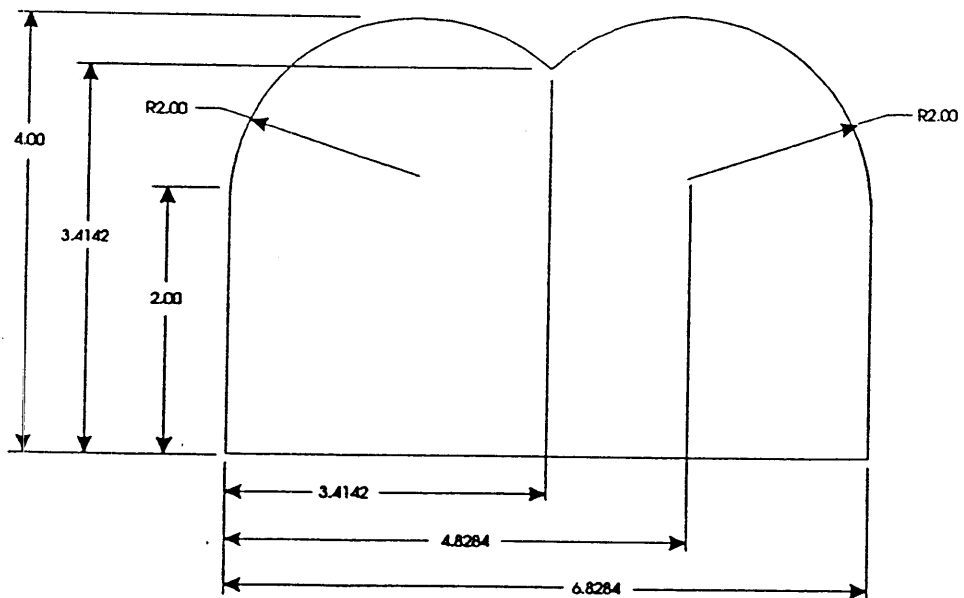
Y-AXIS PROGRAM

```

10 P28=62832
20 P29=0
30 CIRCLE 1,10000,0,-10000,62832
    
```

2-Axis Example

As a further illustration, the programming for a 2-axis path will be illustrated.



XY-Contour Example

Starting at 0,0, we have

a line ending at (0,2).

a clockwise ARC with radius 2 inches, center at (2,2), starting at 180° and ending at 45°.

an ARC of radius 2 inches, center at (4.8284,2) start angle of 135° and end angle of 0 degrees.

a line to (2,0).

a line to (0,0).

The ARCS require the following math:
Assuming 1"=10000 pulses, for the first ARC:

X-AXIS
rotdir = +1
radius = 20000
I = -20000
J = 0
distance = $2\pi \cdot (135/360) \cdot 20000 = 47124$
P28 = 47124
P29 = 34142

Y-AXIS
rotdir = +1
radius = 20000
I = 0
J = 20000
distance = $2\pi \cdot (135/360) \cdot 20000 = 47124$
P28 = 47124
P29 = 34142

For the second ARC

X-AXIS
rotdir = +1
radius = 20000
I = -14142
J = 14142
distance = $2\pi \cdot (135/360) \cdot 20000 = 47124$
P28 = 47124
P29 = 68284

Y-AXIS
rotdir = +1
radius = 20000
I = 14142
J = 14142
distance = $2\pi \cdot (135/360) \cdot 20000 = 47124$
P28 = 47124
P29 = 20000

The finished program will look as follows. Note that this program is shown without line numbers. If the axes are commanded in the immediate mode, line numbers are not needed. If the axes are run in the program mode, line numbers must be included. Blank lines are added for readability only.

X-AXIS
 VEL 10000
 ACC 0
 DEC 0
 STP 0
 P28 = 20000
 MOV 0

Y-AXIS
 VEL 10000
 ACC 0
 DEC 0
 STP 0
 P28 = 20000
 MOV 20000

P28=47124
 P29=34142
 CIRCLE 1,20000,-20000,0,47124

P28=47124
 P29=34142
 CIRCLE 1,20000,0,20000,47124

P28=47124
 P29=68284
 CIRCLE 1,20000,-14142,14142,47124

P28=47124
 P29=20000
 CIRCLE 1,20000,14142,14142,47124

P28=20000
 MOV 68284

P28=20000
 MOV 0

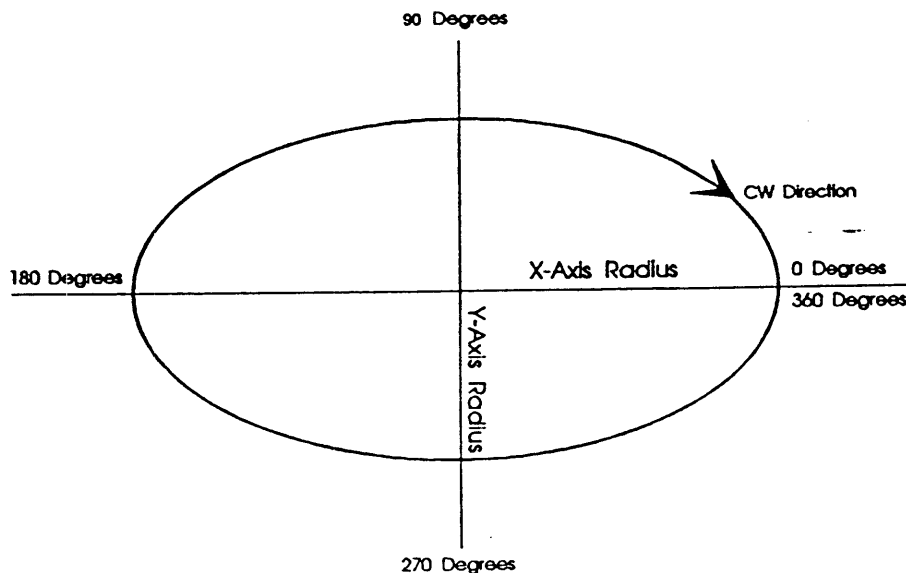
P28=68284
 MOV 0

P28=68284
 MOV 0

Note that in some cases, the axis is being commanded to move to zero when it is already at zero. This is necessary to keep the moves synchronized; all axes have to be programmed to move even if only one is moving. If the user chooses not to do this, then P28 should be set to 0 on all axes and they will not move in a interpolated manner.

Elliptical Interpolation

Elliptical interpolation is very similar to circular interpolation in form. For circular interpolation, both the axes cards are creating the same amplitude (radius) profiles. For an Ellipse, each axis runs with a different amplitude. With elliptical interpolation the following diagram applies:



Ellipse Example
Fig. 8.3

ELLIPSE FORMAT

As an example, we shall see how to generate an ellipse starting at 0 degrees and going 360 degrees. Since the major axis radius is 2", this is 20000 pulses in a 10000 pulses per inch system. The minor axis is 1" (10000 pulses). Each axis will be programmed as if it were doing a circle. The X-axis will be programmed as if it were doing a 20000 pulse radius and moving a complete 360 degrees. The Y-axis will be programmed as if it were doing a 10000 pulse circle and moving a complete 360 degrees. The parameter P28, which determines the feed rate along with the VEL command, has to be set such that it is the maximum of the *distance* calculation of the two axis. Assuming both axes start at 0:

X-AXIS PROGRAM

```
10 P28=125664  
20 P29=0  
30 CIRCLE +,20000,20000,0,125664
```

Y-AXIS PROGRAM

```
10 P28=125664  
20 P29=0  
30 CIRCLE +,10000,0,-10000,62832
```

Note that *I* and *J* must be calculated individually for each axis, based on the radius of each axis. Also note that, as with circles, the order of *I* & *J* is reversed for the Y axis, with *J* negated.

Binary Method Of Specifying Interpolated Moves

In v2.7.x series software, there are provisions to allow the ACR1000 card to accept data at high speed. This is needed in applications that have a DNC (Direct Numerical Control) link. If the point data in a 3-axis system is generated by a CAD/CAM system, the points may be spaced very close together and come very quickly. With this method of specifying moves, the ACR1000 is able to keep up with the shortest distance between moves and very high velocities. This mode is only available in the PC BUSS (ISA bus) mode.

Fast Move Block Format - ASCII

The command must have following format.

```
ctrl-D wwww  
xxxxxxx  
yyyyyyy
```

wwww is the VEL followed by a carriage return. If the velocity has been set and is not changing it can be omitted and just a carriage return is needed. xxxxxxx is the Master Move Distance. (Normally P28.) This is the maximum number of pulses that any axis will move. Again this number is terminated by a carriage return. yyyyyyy is the target position of the axis that is being sent this command. This is also terminated by a carriage return.

As soon as the ACR1000 board sees the ctrl-D it disables the echo. After the entire command is received, the previous echo status is restored. Also, all these parameters can either be specified in decimal or hexadecimal format. Using the hexadecimal format will speed up throughput considerably.

Fast Move Block Format - Binary

DNC applications that require high data throughput with short length lines, arcs and circles should see a considerable improvement in performance using this interface. This new interface is only available for PC BUSS (ISA bus) application of the ACR-1000.

For Lines & Circles

BYTE 0: #4 (ctrl-D)

BYTE 1: MOVE TYPE

Bit Number	Meaning
7	0=ASCII, 1=RAW
6	0=LIN, 1=CIRC
5	0=ABS, 1=INC
4	0=16 Bit Velocity, 1=24 Bit Velocity.
3-0	Reserved, set to 0000

NOTE: 24 Bit Velocities apply to v 4.x Only

1. 16 bit velocities (Byte 1:4 is 0)

Byte	Meaning
2-3	Velocity (Unsigned bit integer, LS byte 1st)
4-7	Master Move (P28-Unsigned 32 bit integer, LS byte 1st)
8-11	Move Distance (Signed 32 bit integer, LS byte 1st) For circles, this is a positive number if arc is "cw" and negative if arc is "ccw".

The following apply only to arc/circle moves

12-15	RADIUS (Unsigned 32 bit integer, LS byte 1st)
16-19	IVECTOR (Signed 32 bit integer, LS byte 1st)
20-23	JVECTOR (Signed 32 bit integer, LS byte 1st)
24-27	ENDPOINT (P29-Signed 32 bit integer, LS byte 1st)

2. 24 bit velocities (Byte 1:4 is 1. v4.x only)

Byte	Meaning
2-4	Velocity (Unsigned 24 bit integer, LS byte 1st)
5-8	Master Move (P28-Unsigned 32 bit integer, LS byte 1st)
9-12	Move Distance (Signed 32 bit integer, LS byte 1st). For circles, this is a positive number if arc is "cw" and negative if arc is "ccw".

The following byte assignments only pertain to arcs.

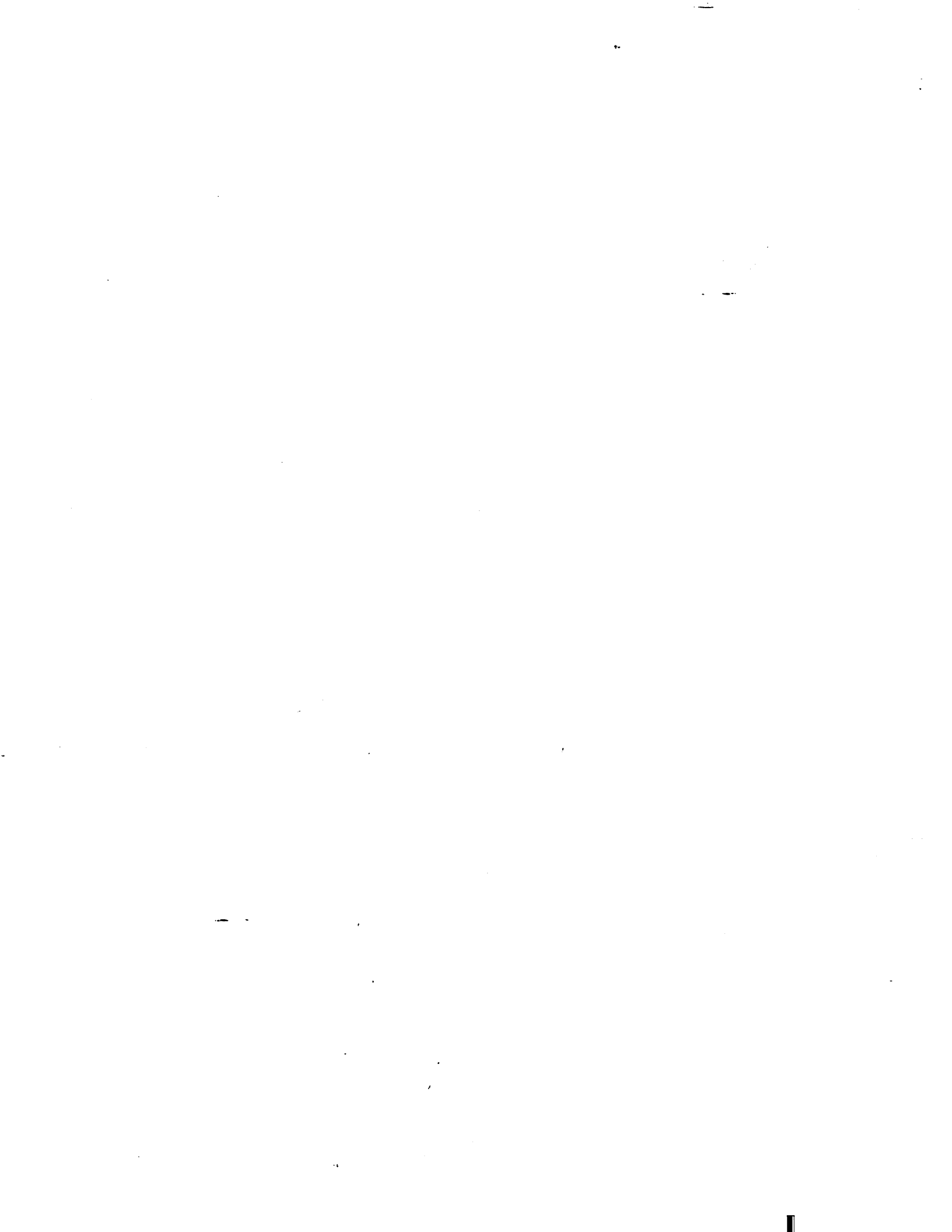
13-16	RADIUS (Unsigned 32 bit integer, LS byte 1st)
17-20	IVECTOR (Signed 32 bit integer, LS byte 1st)
21-24	JVECTOR (Signed 32 bit integer, LS byte 1st)
25-28	ENDPOINT (P29-Signed 32 bit integer, LS byte 1st)

Move Counter

After interrupting a series of moves, it is now possible to determine which move in the series was last executed. A 32 bit "move counter" located at addresses 401DH-402H has been added for this purpose. Prior to initially starting a series of moves, the counter should be zeroed by commanding POKE 401D=0,0,0,0. Of course this counter may also be preset by POKEing in a non-zero value, but if this is done, note that the values must be in hexadecimal format. As each move command begins motion, the counter is incremented. Thus the value will reflect which actual move is in progress at the time of interruption.

This counter may be read two ways: 1) Sending a ctrl-L (12 dec/0Ch) will return a 32 bit ASCII-hex string followed by a carriage return in the same fashion as other status commands. This allows reading the counter at any time and is the preferred method of the two. 2) If the Acroloop is not busy running a program or waiting for a move to complete, the counter may be read using PEEK 401D,4. This will return a string of 4 ASCII-hex bytes separated by spaces and terminated with CR/LF. Of course, if a 16 bit counter is all that is needed, simply PEEK 401F,2.

JOG ing the axis will have no affect on this counter.



Chapter 9 Programming the ACR1000

There are two distinct parts to the ACR1000 board, the Axis Motion Controller and the Programmable I/O Controller (PLC). This chapter describes how to program and control the operation of the ACR1000. Chapter 10 describes the PLC.

Command and Program Modes

The easiest way to describe the format of program entry is to relate it to BASIC as used in most personal computers. Like the BASIC language, the ACR1000 is either in the command mode or the program mode.

The ACR1000 defaults to the command mode when it powers up unless the very first command in memory is the PBOOT command. The PBOOT command forces the ACR1000 to power up in the run mode. In the command mode, as soon as a command followed by a <CR> (ENTER or RETURN key on most keyboards) is received, the ACR1000 will immediately execute the command. The executed command does not get stored into memory unless it is preceded by a line number.

When a command is preceded by a line number in the range of 0 to 65535 and followed by a <CR>, the ACR1000 stores the command into its program memory. This is the program mode. As each command is received, the ACR1000 builds a program by using the line number to tell it where the line should be placed within the program. No commands within the program will be executed until the RUN command is given. (See PBOOT in chapter 7 for power-up execution of a program). There is a run flag relay associated with the run mode. Whenever the system is running a program, this relay is set. This provides a method for a host PC to monitor whether the ACR1000 has completed its program. See the status commands in chapter 7 and the relay address tables in chapter 10 for specific information on the run mode flag.

Most commands can be executed in either the program mode or the command mode. However, there are some commands that are only allowed in the command mode such as the RENUMBER command described in Chapter 7.

Command Format

Most commands may be entered by using the first 3 letters of the command. This speeds up throughput somewhat when a host PC is used to communicate with the ACR1000. Lower-case letters may be used when entering any command, but parameters must still use a capital "P".

Command Line Format

The following program illustrates line format. The program is shown exactly as it would be keyed in the program mode.

```
10 VEL 1000
20 ACCEL 10
30 DECEL 10
40 STP 0
50 MOVE 100000
60 VEL 2000
70 STP 10
80 MOVE 200000
90 STP 2
100 MOV 0
110 GOTO 40
```

NOTE: Spaces entered in the program are ignored unless they are within quotes.

Multiple Commands On A Line

Starting with ACR1000 executive version 2.6.966, multiple commands are allowed on a single line. This has a few advantages: In the command mode, a block of instructions can be sent to the ACR1000 together without some of the overhead of many single instructions. In the program mode, programs become more readable and there is a slight increase in execution speed. The most powerful aspect of a multi-command line is the ability to have many commands conditional on a single test.

NOTE: A REM command will terminate the execution of a line. Any commands after the REM on the same line will be ignored.

Example program:

```
100 ACC 10: DEC10: STP10: VEL 1000
200 MOVE /12345: REM INDEX 12345 PULSES
299 REM IF FAULT, SET USER RELAY,EXIT
300 IF 1 THEN PRINT "**FAULT**": SET 117: CLR 116: GOTO 1000
400 IF 96 THEN 300
500 GOTO 200
```

Writing A Program Using Variables

The ACR1000 card has 50 variable parameters that can be used as arguments for most commands. This allows writing subroutines that have a fill-in-the-blank approach. The same subroutine may be called several times, filling different numbers into parameters, and running a different profile each time. These variables (or parameters) are described in chapter 6.

The following example illustrates the use of variables.

```
10 P30=1000
20 P31=10
30 P32=2000000
40 GSB 140
50 P32 = 0
60 GSB 140
70 GOTO 70
140 ACC P31
150 DEC P31
160 STP P31
170 VEL P30
180 MOV P32
190 RET
```

In the above program, the parameters are loaded up with values. Then the Acceleration, Deceleration, STOP and MOVE commands use these parameters to go through the profile in the subroutine starting at line 140. The first time the subroutine is called on line 40, the axis will move to position 2000000. The second time, it is called at line 60, it will move to position 0.

On v3.x and v5.x the Acroloop can address memory as an indexed array. You may wish to refer to the end of Chapter 6 for a more thorough discussion of the parameter array.

Number Entry In Hexadecimal Format

For specifying arguments, the programmer can either use decimal (base 10) numbers or hexadecimal (base 16) numbers. This will work for all commands with the exception of arguments for branch commands (i.e. GOTO, GOSUB). The hexadecimal numbers can be in the range of 0,1,2,3....E,F. In addition, the HEX number must be preceded by the 2 digit identifier '&H'. In other words, typing MOVE 1000 or MOVE &H3E8 will cause the axis to move to the same place (1000 base 10 = 3E8 base 16). Besides giving the user the choice of an alternate base arithmetic, the hexadecimal entry is more efficient as far as execution is concerned.