AXIMA Software Generic Instructions Reference Manual

P/N 400264-00
Revision: A3
Date: August 14, 2000
© EMERSON Motion Control, Inc. 1997, 2000.

AXIMA Software Generic Instructions Reference Manual

Information furnished by EMERSON Motion Control is believed to be accurate and reliable. However, no responsibility is assumed by EMERSON Motion Control for its use. EMERSON Motion Control reserves the right to change the design or operation of the equipment described herein and any associated motion products without notice. EMERSON Motion Control also assumes no responsibility for any errors that may appear in this document. Information in document is subject to change without notice.

P/N 400264-00
Rev. A3
Date: August 14, 2000
© EMERSON Motion Control, Inc. 1997, 2000.

© EMERSON Motion Control, Inc. 1997, 2000.

Document Number: 400264-00

Revision A3

Date: August 2000

Printed in United States of America

Information in this document is subject to change without notice. Companies, names, and data used in examples herein are fictitious unless otherwise noted. No part of this manual may be reproduced by any means without the written permission of EMERSON Motion Control.

The following are trademarks of EMERSON Motion Control and may not be reproduced in any fashion without written approval of EMERSON Motion Control.

EMERSON Motion Control

Commercial names of products from other manufacturers or developers that appear in this manual are registered or unregistered trademarks of those respective manufacturers or developers which have expressed neither approval nor disapproval of EMERSON Motion Control.

This document has been prepared to conform to the current released version of hardware and software system. Because of our extensive development efforts and our desire to further improve and enhance the product, inconsistencies may exist between the product and documentation in some instances. Call your customer support representative if you encounter an inconsistency.

Table of Contents

Motion Related Generic Instructions

BKL Set backlash compensation

Format: BKL {axis {value}} {axis {value}} ...

Units: units

This instruction displays or sets the backlash compensation of an axis. Backlash is primarily used to compensate for error introduced by hysteresis in mechanical gearboxes. Backlash is added to the secondary setpoint when the primary setpoint moves in the positive direction. If the primary setpoint is not changing, the backlash stays in it's previous state. The backlash offset is used during the summation of the secondary setpoint.

Issuing a BKL command to an axis without an argument will display the current setting for that axis. The default backlash is 0.0 for all axes.

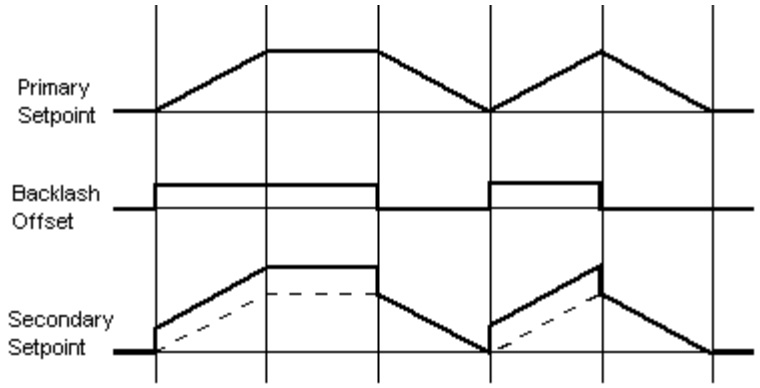

Figure 1 Backlash Compensation Graph

NOTE: The primary setpoint is the summation of the current position and the total cam, gear, and jog offsets. The secondary setpoint is the summation of the primary setpoint and the total ballscrew and backlash offsets. The secondary setpoint is the one that is actually used by the servo loop.

Usage example: This example sets backlash compensation for the axis to 0.0025 units.

```
BKL A 0.0025
```

BSC Ballscrew compensation

Format: BSC command { axis { data } } { axis { data } } ...

This instruction is used along with a second command to initialize and control ballscrew compensation for an axis. Ballscrew compensation is primarily used to compensate for nonlinear position error introduced by mechanical ballscrews. Ballscrew commands are identical to cam commands. Both ballscrews and cams can be active at the same time, each with different settings and offset tables.

The following is a list of valid ballscrew command combinations. See the corresponding cam command description for details.

BSC DIM	Allocate ballscrew segments
BSC SEG	Define ballscrew segment
BSC SRC	Redefine ballscrew source
BSC ON	Enable ballscrew output
BSC OFF	Disable ballscrew output
BSC SCALE	Set ballscrew output scaling
BSC OFFSET	Set ballscrew output offset
BSC FLZ	Set ballscrew input offset
BSC SHIFT	Set incremental ballscrew shift
BSC RES	Transfer ballscrew offset

The main difference between ballscrew and electronic cam is that the default source for a ballscrew points to the primary setpoint, therefore the BSC SRC command is normally not required. The primary setpoint is used so that the ballscrew offset is not fed into the calculation of the ballscrew index, causing an unstable condition.

NOTE: The primary setpoint is the summation of the current position and the total cam, gear, and jog offsets. The secondary setpoint is the summation of the primary setpoint and the total ballscrew and backlash offsets.

The secondary setpoint is the one that is actually used by the servo loop.

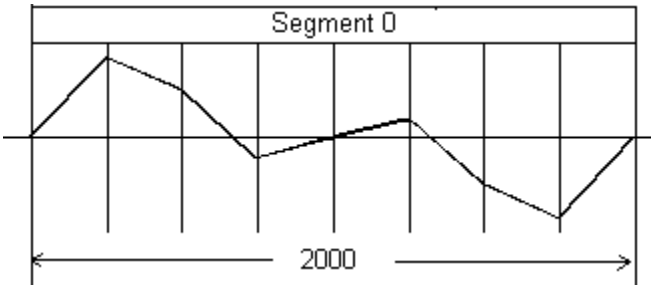

Figure 2 Ballscrew Compensation Graph

The following example enables the X axis to use the above ballscrew table:

Usage example:

```
dim la(2)
dim la0(9)
la0(00)=0
la0(01)=853
la0(02)=500
la0(03)=-146
la0(04)=0
la0(05)=146
la0(06)=-500
la0(07)=-853
la0(08)=0
BSC DIM X1
BSC seg x(0,2000,la0)
BSC on x
```

BSC DIM Allocate ballscrew segments

Format: BSC DIM {axis segments} {axis segments} ...

This instruction allocates working space for ballscrew compensation. A ballscrew must be dimensioned before it can be initialized. This is in addition to the dimensioning done for the actual arrays attached to the ballscrew segments. Dimensioning a ballscrew, unlike dimensioning a cam, automatically sets the source to the primary

setpoint of the given axis. The BSC SRC command does not usually need to be used.

A ballscrew can be composed of more than one segment with each segment having different distances between table entries. This allows some parts of the table to be defined coarsely and others to be defined in more detail.

The memory allocated by the BSC DIM command is a base of 48 bytes of working space plus an additional 20 bytes per defined segment.

The following example allocates two ballscrew segments for the B axis and a single segment for the B axis:

Usage example:

```
BSC DIM A2 B1
```

BSC SEG Define ballscrew segment

Format: BSC SEG { axis (segment, length, array_name) } ...

Units:

```
segment= none
length =  input units
array_name =  none
```

This instruction defines the segments that were allocated with the BSC DIM command. The “segment” is a number from 0 to segments-1 and indicates which segment is being defined. The “length” parameter defines the total length of the given segment. The "array_name" is the name of the longint array where the data points are to be stored.

An error will occur if the ballscrew has not been allocated with the BSC DIM command.

A ballscrew can be composed of more than one segment with each segment having different distances between table entries. This allows some parts of the table to be defined coarsely and others to be defined in more detail.

The following internal formulas are modified by the BSC SEG information:

distance between entries = segment length / (number of table entries - 1)

total ballscrew length = sum of segment lengths

Note that this information can be altered while the ballscrew is enabled, allowing the replacement of segments or the changing segment lengths on the fly.

The following example defines the segment 1 of the A axis ballscrew as being 10000 units long and pointing to longint array LA0 for its data:

Usage example: `BSC SEG A(1,10000,LA0)`

BSC SRC Redefine ballscrew source

Format 1: BSC SRC { axis encoder_number } ...

Format 2: BSC SRC { axis var_name } ...

This instruction sets a pointer to a memory area that is used to generate an index into the ballscrew table. Dimensioning a ballscrew automatically points the source at the primary setpoint of the given axis. Therefore, the BSC SRC command is not usually required.

An error will occur if the ballscrew has not been allocated with the BSC DIM command.

Normally, "Format1" is used to define a source with an encoder number. The optional "Format2" allows any system or user parameter to be used as a source. The data type of the source is automatically determined by the card. It is possible to use a floating point variable as the source but the index update will become course after the number becomes larger than six digits to the left of the decimal point.

The following example sets the source of the X ballscrew to encoder 3 and the source of the Y ballscrew to the current position of AXIS1. (Note that the parameter P12544 is not enclosed in parentheses.)

Usage example: `BSC SRC X3 Y P12544`

BSC ON Enable ballscrew comp

Format: BSC ON {axis} {axis} ...

This instruction enables ballscrew compensation for the designated axes. An error will be returned if the ballscrew has not been allocated with the BSC DIM command.

The following example enables ballscrew compensation for A, B, and C axes:

Usage example: `BSC ON A B C`

BSC OFF Disable ballscrew comp

Format: BSC OFF {axis} {axis} ...

This instruction disables ballscrew compensation for the designated axes. An error will be returned if the ballscrew has not been allocated with the BSC DIM command.

The following example disables ballscrew compensation for X and Y axes:

Usage example: `BSC OFF A B`

BSC SCALE Set ballscrew output scaling

Format: BSC SCALE {axis {scale}} {axis {scale}} ...

Units: none

This instruction sets or displays the ballscrew output scaling of an axis. After the ballscrew table and index are used to interpolate an initial offset value, the value is multiplied by the ballscrew output scaling factor and then shifted by the ballscrew output offset. This number is then multiplied by the PPU of the given axis.

Issuing a BSC SCALE command to an axis without an argument will display the current setting for that axis. An error will be returned if the ballscrew has not been allocated with the BSC DIM command. The default ballscrew output scaling is 1.0 for all axes.

The following example scales the A axis ballscrew offset by 50 percent:

Usage example: `BSC SCALE A0.5`

BSC OFFSET Set ballscrew output offset

Format: BSC OFFSET {axis {scale}} {axis {scale}} ...

Units: output units

This instruction sets or displays the ballscrew output offset of an axis. After the ballscrew table and index are used to interpolate an initial offset value, the value is multiplied by the ballscrew output scaling factor and then shifted by the ballscrew output offset. This number is then multiplied by the PPU of the given axis.

Issuing a BSC OFFSET command to an axis without an argument will display the current setting for that axis. An error will be returned if the ballscrew has not been allocated with the BSC DIM command. The default ballscrew output offset is 0.0 for all axes.

The following example shifts the A axis ballscrew table output 500 units:

Usage example: `BSC OFFSET A500`

BSC FLZ Set ballscrew input offset

Format: BSC FLZ {axis {offset}} {axis {offset}} ...

Units: input units

This instruction sets or displays the ballscrew input offset of an axis. The ballscrew input offset is added to the ballscrew table index before it is used to calculate the actual table index. This is used to shift the zero of the table to the location of the input offset.

Issuing a BSC FLZ command to an axis without an argument will display the current setting for that axis. An error will be returned if the ballscrew has not been allocated with the BSC DIM command. The default ballscrew input offset is 0.0 for all axes.

The following example shifts the A axis ballscrew table index by 250 units:

Usage example: `BSC FLZ A250`

CAM Electronic CAM

Format: CAM command {axis {data}} {axis {data}} ...

This instruction is used along with a second command to initialize and control an electronic CAM for an axis. An electronic CAM is primarily used as a replacement for a mechanical CAM. Internally, CAM is identical to ballscrew with the exception of the default source

initialization. Both CAM and ballscrew can be active at the same time.

The following is a list of valid CAM command combinations:

CAM DIM	Allocate CAM segments
CAM SEG	Define CAM segment
CAM SRC	Redefine CAM source
CAM ON	Enable CAM output
CAM OFF	Disable CAM output
CAM SCALE	Set CAM output scaling
CAM OFFSET	Set CAM output offset
CAM FLZ	Set CAM input offset

CAM uses an arbitrary encoder position to generate an index into a table of offset values. If this index falls between two table entries, the CAM offset is linearly interpolated between the entries. This offset is then scaled, shifted by the output offset, and then multiplied by the PPU for the given axis.

A CAM table can be composed of more than one segment with each segment having different distances between table entries. The data for each segment of the table resides in separate longint arrays, possibly of different sizes. This allows some parts of the table to be defined coarsely and others to be defined in more detail.

The table index automatically tracks which segment it is in and where it is within that segment. It also wraps around if it goes off either end of the table. The wraparound point is determined by the total length of the table equal to the summation of the individual segment lengths.

NOTE: The primary setpoint is the summation of the current position and the total cam, gear, and jog offsets. The secondary setpoint is the summation of the primary setpoint and the total ballscrew and backlash offsets. The secondary setpoint is the one that is actually used by the servo loop

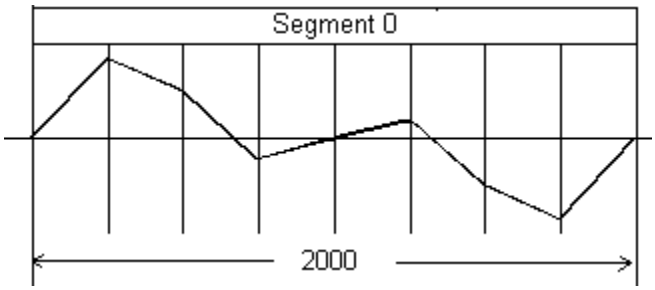

Figure 3 Electronic CAM Graph

The following example enables the X axis to use the above CAM table with encoder number 4 as the input source:

Usage example:

```
dim la(2)
dim la0(9)
la0(00)=0
la0(01)=853
la0(02)=500
la0(03)=-146
la0(04)=0
la0(05)=146
la0(06)=-500
la0(07)=-853
la0(08)=0
CAM DIM A1
CAM SRC A4
CAM seg A(0,2000,la0)
CAM on A
```

CAM DIM Allocate CAM segments

Format: CAM DIM {axis segments} {axis segments} ...

This instruction allocates working space for a CAM. The CAM must be dimensioned before it can be initialized. This is in addition to the dimensioning done for the actual arrays attached to the CAM segments. A newly dimensioned CAM has no source defined for it. The CAM SRC command must be used.

A CAM can be composed of more than one segment with each segment having different distances between table entries. This

allows some parts of the table to be defined coarsely and others to be defined in more detail.

The memory allocated by the CAM DIM command is a base of 48 bytes of working space plus an additional 20 bytes per defined segment.

The following example allocates two CAM segments for the A axis and a single segment for the B axis:

Usage example: `CAM DIM A2 B1`

CAM FLZ Set CAM input offset

Format: CAM FLZ {axis {offset}} {axis {offset}} ...

Units: input units

This instruction sets or displays the CAM input offset of an axis. The CAM input offset is added to the CAM table index before it is used to calculate the actual table index. This is used to shift the zero of the table to the location of the input offset.

Issuing a CAM FLZ command to an axis without an argument will display the current setting for that axis. An error will be returned if the CAM has not been allocated with the CAM DIM command. The default CAM input offset is 0.0 for all axes.

The following example shifts the A axis CAM table index by 250 units:

Usage example: `CAM FLZ A250`

CAM ON Enable CAM output

Format: CAM ON {axis} {axis} ...

This instruction enables CAM output for the designated axes. An error will be returned if the CAM has not been allocated with the CAM DIM command. The following example enables CAM for A, B, and C axes:

Usage example: `CAM ON A B C`

CAM OFF Disable CAM output

Format: CAM OFF {axis} {axis} ...

This instruction disables CAM output for the designated axes. An error will be returned if the CAM has not been allocated with the CAM DIM command. The following example disables CAM for A and B axes:

Usage example:

```
CAM OFF A B
```

CAM OFFSET Set CAM output offset

Format: CAM OFFSET {axis {scale}} {axis {scale}} ...

Units: output units

This instruction sets or displays the CAM output offset of an axis. After the CAM table and index are used to interpolate an initial offset value, the value is multiplied by the CAM output scaling factor and then shifted by the CAM output offset. This number is then multiplied by the PPU of the given axis.

Issuing a CAM OFFSET command to an axis without an argument will display the current setting for that axis. An error will be returned if the CAM has not been allocated with the CAM DIM command. The default CAM output offset is 0.0 for all axes.

The following example shifts the A axis CAM table output 500 units:

Usage example:

```
CAM OFFSET A500
```

CAM RES Transfer cam offset

Format: CAM RES { axis { offset } } { axis { offset } } ...

Units: units

This instruction either clears or preloads the cam offset of a given axis and adds the difference to the current position. This instruction will also clear out any cam shift that may have been built up by an incremental cam. The default "offset" argument is zero.

The current position and cam offset are adjusted according to the following formula:

current_position = current_position + cam_offset - offset

cam_offset = offset

When a cam is turned off, the offset remains in the cam offset parameter. The CAM RES command can be used to transfer the offset into the current position where it can be used as part of a normal move.

The following example transfers the A axis cam offset into the current position:

Usage example: `CAM RES A`

CAM SCALE Set CAM output scaling

Format: CAM SCALE {axis {scale}} {axis {scale}} ...

Units: none

This instruction sets or displays the CAM output scaling of an axis. After the CAM table and index are used to interpolate an initial offset value, the value is multiplied by the CAM output scaling factor and then shifted by the CAM output offset. This number is then multiplied by the PPU of the given axis.

Issuing a CAM SCALE command to an axis without an argument will display the current setting for that axis. An error will be returned if the CAM has not been allocated with the CAM DIM command. The default CAM output scaling is 1.0 for all axes.

The following example scales the A axis CAM offset by 50 percent:

Usage example: `CAM SCALE A0.5`

CAM SEG Define CAM segment

Format: CAM SEG { axis (segment, length, array_name) } ...

Units: segment = none

length = input units

array_name = none

This instruction defines the segments that were allocated with the CAM DIM command. The "segment" is a number from 0 to segments-1 and indicates which segment is being defined. The "length" parameter defines the total length of the given segment. The "array_name" is the name of the longint array where the data points are to be stored.

An error will occur if the CAM has not been allocated with the CAM DIM command.

A CAM can be composed of more than one segment with each segment having different distances between table entries. This allows some parts of the table to be defined coarsely and others to be defined in more detail.

The following internal formulas are modified by the CAM SEG information:

distance between entries = segment length / (number of table entries - 1)

total length of the CAM = sum of segment lengths.

Note that this information can be altered while the CAM is enabled, allowing the replacement of segments or the changing segment lengths on the fly.

The following example defines the segment 1 of the A axis CAM as being 10000 units long and pointing to longint array LA0 for its data:

Usage example:

```
CAM SEG A(1,10000,LA0)
```

CAM SHIFT Set incremental cam shift

Format: CAM SHIFT { axis { offset } } { axis { offset } } ...

Units: units

This instruction sets the incremental cam shift. The first entry of one cam segment is normally equal to the last entry of the previous segment. In cases where this is not true, the cam is considered to be incremental. The starting "shift" for all cams is 0.0. Issuing a CAM SHIFT with no argument will display the current reading.

Whenever an incremental cam crosses a segment boundary, the difference between the two entries is used to adjust the cam shift.

The cam shift is added to the interpolated offset to generate the actual cam offset. If the total of all segment boundary shifts is not equal to zero, the overall pattern will be offset by that amount each cycle. Crossing cam segment boundaries backwards will also adjust the cam shift.

The following example clears the A axis cam shift:

Usage example: `CAM SHIFT A0`

CAM SRC Redefine CAM source

Format 1: CAM SRC { axis encoder_number } ...

Format 2: CAM SRC { axis var_name } ...

Units: none

This instruction sets a pointer to a memory area that is used to generate an index into the CAM table. This instruction is required since a new CAM will not have a default source assigned to it. An error will occur if the CAM has not been allocated with the CAM DIM command.

Normally, "Format1" is used to define a source with an encoder number. The optional "Format2" allows any system or user parameter to be used as a source. The data type of the source is automatically determined by the card. It is possible to use a floating point variable as the source but the index update will become course after the number becomes larger than six digits to the left of the decimal point.

The following example sets the source of the A axis to encoder 3 and the source of the B axis to the current position of AXIS1 (note that the parameter P12544 is not enclosed in parentheses) :

Usage example: `CAM SRC A3 B P12544`

FVEL Set final velocity

Format: FVEL {rate}

Units: units / second

This instruction sets the final velocity value for a master move profile. Final velocity is used as a target velocity when the STP ramp

is active. The value is used to slow down, but not stop, between moves.

A move will not ramp up to this value, it will only ramp down. The final velocity is only used when STP is non-zero and the current velocity is greater than the final velocity.

Issuing an FVEL command without an argument will display the current setting. The default final velocity is zero. Regardless of the setting, the master bits "FVEL Zero Pending" and "FVEL Zero Active" can be used to temporarily override the final velocity to zero. An error will be returned if no master is attached.

Usage example: This example generates a path using different combinations of velocity, final velocity, and stop ramps. Note that the velocity profile between moves 3 and 4 does not ramp down even though STP is set to 1000. This is because the final velocity of 2000 is greater than the current velocity at that point in the profile.

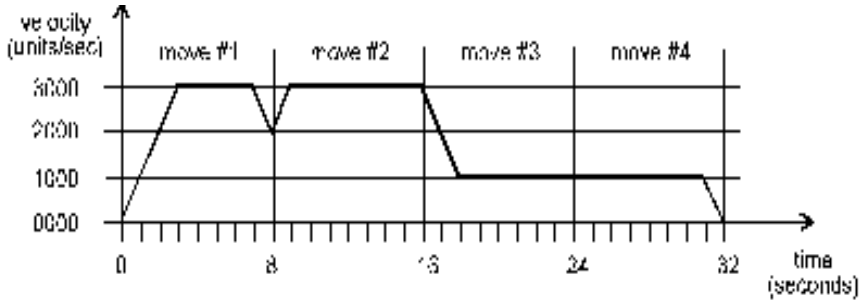

Figure 4 Set Final Velocity Graph

```
ACC1000 DEC1000
VEL3000 FVEL2000 STP1000 A/19000
VEL3000 FVEL2000 STP0 A/23500
VEL1000 FVEL2000 STP1000 A/10000
VEL1000 FVEL0 STP1000 A/7500
```

IVEL Set initial velocity

Format: IVEL {rate}

Units: units / second

This instruction sets the initial velocity value for a master move profile. If this value is zero (default) it is ignored. Otherwise, the move will start at this velocity regardless of the current acceleration ramp instruction and deceleration ramp instruction settings. Issuing an IVEL command without an argument will display the current setting. The default initial velocity is zero. An error will be returned if no master is attached. The following example sets the initial velocity to 1000 units / second:

Usage example: IVEL 1000

JLM Set jog limits

Format1: JLM { axis { limit } } ...

Format2: JLM { axis { (plus, minus) } } ...

Units: units

This instruction, available with A4 (or later) firmware, sets the jog limits for an axis. The jog limits are only checked when the "jog limit check" bit is set and the JOG FWD or JOG REV commands are in operation. The JOG ABS and JOG INC commands ignore jog limits even if the "jog limit check" bit is set. Jog limits only place limits on jog offset calculations. The primary and secondary setpoints are not part of the jog limits.

When the "jog limit check" bit is set, the JOG FWD command will jog to the positive jog limit and stop. If the current jog offset is greater than the positive jog limit, the JOG FWD command will do nothing. Likewise, the JOG REV command will jog to the negative jog limit and if the offset is less than the negative jog limit, JOG REV will do nothing.

Issuing a JLM command to an axis without an argument displays the current positive and negative limits for that axis. The first command format sets the positive limit to plus "limit" and the negative limit to minus "limit". The second format sets the positive limit to "plus" and the negative limit to "minus". The default for both is 0.0 for all axes.

Usage example: This example sets the A axis jog limits to +3.5 and -1.0 units:

```
JLM A(3.5,-1.0)
```

JOG SRC Set external timebase

Format: JOG SRC axis sourcedef { axis sourcedef } ...

Group: Setpoint Control

This instruction available with A4 (or later) firmware only, specifies the timebase for jogging. See the SRC command for the definition of the "sourcedef" argument. During each servo interrupt, the change in source pulses is multiplied by the servo period and the resulting delta time is fed into the jog mechanism. By default, jog is sourced off the CLOCK, feeding a single time unit per interrupt. Redirecting the jog source allows an external timebase to be used. The following example sets the A axis jog source to encoder 3:

Usage example:
```
JOG SRC A ENC3
```

PAUSE Activate pause mode

Format: PAUSE { PROG number | ALL }

This instruction pauses the currently selected program by setting the program's "pause mode enable" bit. When this bit is set, a feedhold is issued to the attached master and the program's "pause mode wait" bit is set, pausing the program.

A program can be resumed by issuing a RESUME command or by manually clearing the program's "pause mode enable" bit. When this bit is cleared, a cycle start is issued to the attached master and the "pause mode wait" bit is cleared, resuming the program.

The PAUSE PROG command will pause the corresponding program and the PAUSE ALL command will pause all programs. These commands can be issued from anywhere in the system, including programs. The following example pauses the current program:

Usage example:
```
PAUSE
```

RATCH Software Ratchet Commands

Format: RATCH index command { data }

Group: Global Objects

This instruction, available with A4 (or later) firmware only, is used along with a second command to setup software ratchets. The ratchet

"index" is a number from 0 to 7. Software ratchets are sources that can ignore, negate, or buffer both positive and negative pulses.

When a ratchet is set up for buffering, pulses in the buffering direction are added to an internal count instead of causing the ratchet output to change. Pulses in the normal direction are first used to unbuffer previously buffered pulses. When there are no more pulses to unbuffer, the ratchet tracks normally.

The following is a list of valid ratchet command combinations:

`RATCH SRC` Define ratchet source

`RATCH MODE` Set ratchet mode

RATCH SRC Define ratchet source

Format: RATCH index SRC sourcedef

Group: Global Objects

This instruction, available with A4 (or later) firmware only, sets the input source for a ratchet. The default ratchet source is NONE. See the SRC command for the definition of the "sourcedef" argument.

The following example sets the source of ratchet 2 to encoder 7:

Usage example:

```
RATCH2 SRC ENC7
```

RATCH MODE Set ratchet mode

Format: RATCH index MODE { mode }

Group: Global Objects

This instruction, available with A4 (or later) firmware only, sets the conversion mode for a ratchet. Issuing a mode command to a ratchet without an argument will display the current mode for that ratchet. The default ratchet mode is zero.

The following is a table of ratchet modes and their affect on incoming source pulses:

mode	positive pulses	negative pulses
0	normal	normal
1	normal	ignore
2	normal	negate
3	normal	buffer
4	ignore	normal
5	ignore	ignore
6	ignore	negate
7	ignore	buffer
8	negate	normal
9	negate	ignore
10	negate	negate
11	negate	buffer
12	buffer	normal
13	buffer	ignore
14	buffer	negate
15	buffer	buffer

The following example sets ratchet 7 to buffer negative pulses:

Usage example: `RATCH7 MODE 3`

RESUME Release pause mode

Format: RESUME { PROG number | ALL }

This instruction resumes the currently selected program by clearing the program's "pause mode enable" bit. When this bit is cleared, a cycle start is issued to the attached master and the program's "pause mode wait" bit is cleared, resuming the program.

A program can be paused by issuing a PAUSE command or by manually setting the program's "pause mode enable" bit. When this bit is set, a feedhold is issued to the attached master and the "pause mode wait" bit is set, pausing the program.

This instruction will resume the corresponding program and the Resume All command will resume all programs. These commands can be issued from anywhere in the system, including programs. The following example resumes the current program:

Usage example: `RESUME`

ROTARY Set rotary axis length

Format: ROTARY { axis { length } } { axis { length } } ...

Units: units

This instruction sets the rotary axis length used for the shortest distance calculations. Issuing a ROTARY command without an argument will display the current setting. The default rotary length is 0.0 for all axes, disabling shortest distance moves.

If the rotary length of an axis is non-zero, a MOD function is done on absolute moves and the result is run through a shortest distance calculation. The resulting move will never be longer than half the rotary axis length. Incremental moves are not affected by the rotary axis length.

This procedure actually converts absolute moves into incremental moves that are up to plus or minus half the rotary length. Current positions are normally generated that lie outside of the rotary length boundaries. The NORM command can be used to return the current position to within the bounds of the rotary length. The following example sets the rotary length of the A axis to 360 units:

Usage example:
```
ROTARY A360
```

SRC Set external timebase

Format: SRC sourcedef

Group: Velocity Profile

This instruction, available with A4 (or later) firmware only, specifies the timebase for coordinated motion. The source can be defined in any of the following formats:

sourcedef	description
NONE	Disconnect device from source
CLOCK	Connect to servo clock (1 pulse per period)
ENC *encoder*	Connect to encoder register
encoder	Connect to encoder register
RATCH *ratchet*	Connect to ratchet output
parameter	Connect to user or system parameter

During each servo interrupt, the change in source pulses is multiplied by the servo period and the resulting delta time is fed into the velocity profile mechanism. By default, the velocity profile is sourced off the CLOCK, feeding a single time unit per interrupt. Redirecting the source allows an external timebase to be used for coordinated motion.

The following example sets source of the current master to ratchet number 5:

Usage example:

```
SRC RATCH5
```

STP Set stop ramp

Format: STP {rate}

Units: units / second^2

The STP command sets the deceleration ramp to be used at the end of the next move. Issuing a STP command with no argument will display the current setting. The default stop ramp is 20000 units / second^2.

Setting STP to zero will end the move without ramping down. This allows back-to-back moves to be merged together. The final velocity of the first move will then be the initial velocity of the second move.

Setting STP to anything other than zero will cause the move to ramp down at the end of the move to the final velocity set by the FVEL command. This value is normally zero which brings the move to a complete stop. If the final velocity is greater than zero, the move will slow down and then ramp back up on the following move.

The following example sets up accel and stop ramps of 10000 units per second^2 and then tells the system to index the A axis. This will accelerate to speed using ACC and decelerate to a complete stop using the STP value.

Usage example:

```
ACC 10000
STP 10000
A 10
```

TRJ Start new trajectory

Format: TRJ {axis target} {axis target} ...

Units: units

This instruction allows changing the trajectory of the axes on the fly. Once the TRJ command is executed, the axes will continue to go along the specified vector until one of the following conditions are met

1. The move completes normally
2. The move is aborted. (HALT)
3. Another TRJ command is received.

The following example makes the axis go along a 45 degree vector and then after 10 seconds goes at a 90 degree vector

Usage example:

```
TRJ A100000 B100000
DWL 10
TRJ B100000
```

Other Generic Instructions

BRESET Disable Battery Backup

Format: BRESET

This instruction disables the battery backup the next time power is removed from the board. This allows controller boards to be stored on the shelf without needlessly draining power from the battery. The next time power is applied to the board, after shutting down with BRESET in effect, the battery will return to normal and will hold programs during consecutive power sequences.

NOTE: Once this command is issued, there is no way to return the battery to normal operation without removing and then restoring power. Stored programs will be lost.

Usage example: BRESET

CLOSE Close a device

Format: CLOSE #device

This instruction closes a device. The valid range for "device" is 0 to 3. Each program has it's own device #0 which is used as its default device. Devices #1 through #3 are board-wide system resources that can be closed from within any program or from any system or program prompt.

When a device is opened, the operating system attached to that device enters an idle state, allowing incoming characters to be used by a program instead of being interpreted as commands. When the device is closed, the device will enter its auto-detect mode as if it were starting from power-up.

Usage example: `CLOSE #1`

ECHO Control Character Ecoing

Format: ECHO {mode}

This instruction controls the prompt and echo on a communication channel. Issuing an ECHO command without an argument displays

the current setting. The default setting for echo control is 1 for all communication channels.

The following table lists the valid echo modes:

Echo Mode	Command Prompt	Error Messages	Character Echo
0	ON	ON	OFF
1	ON	ON	ON
2	ON	OFF	OFF
3	ON	OFF	ON
4	OFF	ON	OFF
5	OFF	ON	ON
6	OFF	OFF	OFF
7	OFF	OFF	ON

The following example turns off error message reporting:

Usage example: `ECHO 3`

INPUT Receive data from a device

Format: INPUT { ; } { #device ,} {"prompt string" separator} parameterlist

This instruction receives data from a device and places the data into the designated parameters. If no device number is given, device #0 is used. If the device is closed, or was never opened, the INPUT command will return an error.

The optional semicolon that follows the INPUT command controls the echo of characters as they are received. Characters are normally echoed. Placing a semicolon after the command will prevent the characters from being echoed.

If a "prompt string" is used, it will be printed out to the device before the parameters are read from the device. The separator after the prompt string can be either a comma or a semicolon. If the separator is a semicolon, the final carriage return / linefeed output sequence will be suppressed. Otherwise, a carriage return / linefeed will be output after all of the data has been read from the device.

The "parameterlist" is a list of parameters separated with commas. When the data is read from the device, either a comma or a carriage return will cause the current field to be registered and the next field

to begin. If the current field is the last parameter in the parameter list, the input command will end.

Characters less than CHR$(32) or greater than CHR$(126) will be ignored. In order to read these characters, the INKEY$ function must be used.

Usage example:

```
REM --- main program
DIM $V(1,80)
OPEN "COM1:9600,N,8,1" AS #1
PRINT #1,
PRINT #1, "Enter 'EXIT' to quit ..."
INPUT #1, "Command?", $V0
$V0 = UCASE$($V0)
PRINT #1, "["; $V0; "]"
IF ($V0 = "EXIT") GOTO 300
GOTO 200
REM --- program shutdown
PRINT #1, "Program terminated"
CLOSE #1
```

MODE Binary Data Formatting

Format: MODE { mode }

This instruction controls the encoding and decoding of the data fields in immediate mode commands. Issuing a MODE command without an argument displays the current setting. The default setting for the FIFO channel is 0 and the default for the COM1 and COM2 channels is 1.

Control character prefixing and high bit stripping follow Kermit communications protocol conventions. The escape code for control prefixing is the '#' character and the escape code for high bit stripping is the '&' character.

These sequences were added primarily for the serial communication channels. The control prefixing was added to prevent valid data within a binary packet from being confused with the XON / XOFF flow control codes. The high bit stripping was added for cases in which a 7-bit data path must be used. In general, the FIFO channel does not require these precautions.

The following table lists the valid data formatting modes. Note that it is not possible to activate high bit stripping without also activating the control character prefixing.

Mode Value	High Bit Stripping	Control Prefixing
0	OFF	OFF
1	OFF	ON
2	OFF	OFF
3	ON	ON

The following example turns on both control prefixing and high bit stripping:

Usage example: `MODE 3`

OPEN Open a device

Format: OPEN "device string" AS #device

This instruction opens a device. The valid range for "device" is 0 to 3. Each program has it's own device #0 which is used as its default device. Devices #1 through #3 are board-wide system resources that can be opened and used from within any program or from any system or program prompt.

The "device string" describes the device that is to be opened. Serial device strings contain information required to set up communications. Valid device strings are:

```
"FIFO:"
"COM1:baudrate,parity,databits,stopbits"
"COM2:baudrate,parity,databits,stopbits"
baudrate = 300,600,1200,2400,9600,19200,38400
parity = N,E,O
databits = 5,6,7,8
stopbits = 1,2
```

When a device is opened, the operating system attached to that device enters an idle state, allowing incoming characters to be used by a program instead of being interpreted as commands. When the device is closed, the device will enter its auto-detect mode as if it were starting from power-up.

Usage example:

```
OPEN "COM1:9600,N,8,1" AS #1
PRINT #1, "Hello world!"
CLOSE #1
```

PLS Programmable Limit Switch

Format: PLS index command {data}

This instruction is used with a second command to control the eight Programmable Limit Switch (PLS) objects that execute in the background. A PLS uses a source parameter to generate a table index. This table index is used to lookup an array entry which is then transferred into a destination parameter.

By pointing the source to an encoder position and the destination to the digital outputs, the PLS can sequence through a set of outputs based on a shaft position, similar to a mechanical cam operating a bank of switches.

The following is a list of valid PLS command combinations:

Command	Description
PLS SRC	Set PLS source pointer
PLS DST	Set PLS destination pointer
PLS BASE	Attach array to PLS
PLS RES	Reset or preload counter
PLS ROTARY Available with A2 (or later) firmware only	Set PLS rotary length
PLS FLZ	Set PLS index offset
PLS MASK	Set PLS output bit mask
PLS RATIO	Set PLS scaling ratio
PLS ON	Enable PLS update
PLS OFF	Disable PLS update

Since there are eight PLS objects, the "index" argument must be in the range of 0 - 7.

The following block diagram outlines PLS operation:

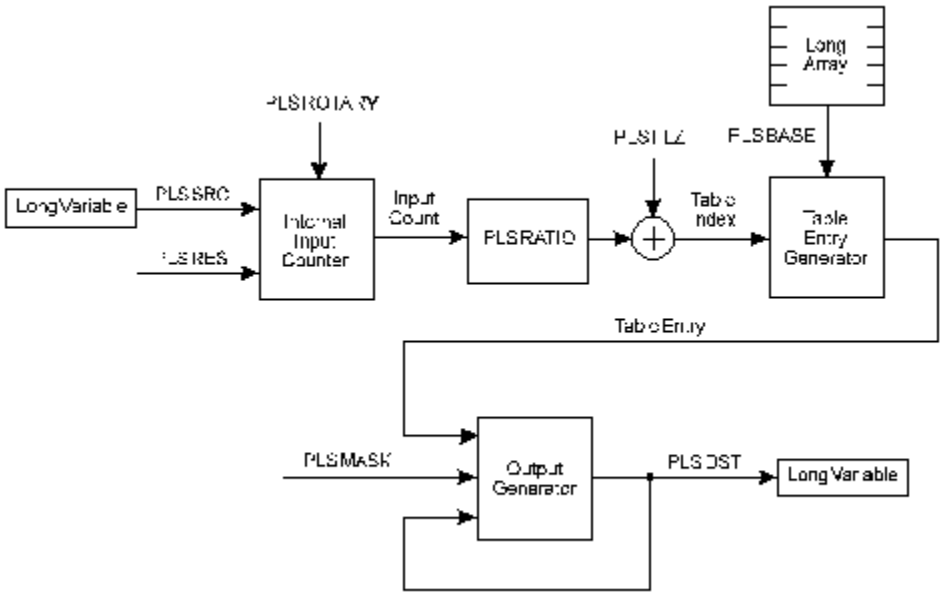

Figure 5 PLS Operation Diagram

A long integer pointed to by the PLS SRC command, typically an encoder position, is used as the PLS source. As the source parameter changes, the PLS generates an internal input count. If a rotary length is set with the PLS ROTARY command, the input count "wraps around" based on the rotary length. While in rotary mode, the input count can be reset or preloaded using the PLS RES command.

The internal input count is multiplied by the PLS RATIO and added to the PLS FLZ to generate a table index. The table index is used to fetch an entry from the long integer array pointed to with the PLS BASE command. If the table index is outside of the array boundaries, a zero is used instead of an array entry.

The table entry is merged with the parameter pointed to by the destination pointer. By default, the destination pointer points to P4097, the optoisolated digital outputs, but this can be changed by using the PLS DST command. The PLS can be set to modify any or all of the 32 bits in the destination parameter by using the PLS MASK command.

The following example sets PLS 0 to look at ENC 0 and use the long integer array LA0 for its table of values. The mask is set to 65535 (0x0000FFFF) so that only the lower 16 bits of P4097 (OUT32 - OUT43) will be modified. The bits will sequence through a binary pattern representing the encoder position from of 0 to 1999 pulses.

These examples assume that PROG0 has enough user memory allocated inside it to accommodate a 2000 element long integer array. This memory allocation can be done using the DIM command from the SYS level.

Usage example:

```
PROG0
100 DIM LA(1)
110 DIM LA0(2000)
120 DIM LV(5)
130 LV0=0
200 LA0(LV0) = LV0
210 LV0=LV0+1
220 IF (LV0 < 2000) GOTO 200
300 PLS0 SRC ENC0
310 PLS0 BASE LA0
320 PLS0 MASK 65535
330 PLS0 ON
RUN
```

PLS SRC Set PLS source pointer

Format1: PLS index SRC LV number

***Format2:** PLS index SRC P number

***Format3:** PLS index SRC ENC number

*Available with A4 (or later) firmware only

This instruction sets the PLS source pointer. The first two command formats allow any long integer parameter to be used as an input source. The ENC format sets the PLS source pointer to the address of the corresponding encoder position parameter. Since there is no default for the PLS source pointer, one must be defined before the PLS ON command is issued.

The following example sets the source of PLS 5 to encoder 3:

Usage example: `PLS5 SRC ENC3`

PLS DST Set PLS destination pointer

Format1: PLS index DST LV number

Format2: PLS index DST P number

This instruction, available with A4 (or later) firmware only, sets the PLS destination pointer. The two command formats allow any long integer parameter to be used as the PLS destination. The default PLS destination pointer is the address of P4097, the optoisolated digital output parameter.

The following example sets the destination of PLS 3 to P4109 (XIO-2 outputs):

Usage example: `PLS3 DST P4109`

PLS BASE Set PLS array pointer

Format: PLS index BASE LA number

This instruction sets the PLS array pointer. The "number" argument indicates which long integer array is to be used. The array must first be allocated using the DIM command. Since there is no default for the PLS array pointer, one must be defined before the PLS ON command is issued.

The following example attaches PLS 4 to the LA2 array:

Usage example: `PLS4 BASE LA2`

PLS RES Reset or preload internal counter

Format: PLS index RES { offset }

Units: input counts

This instruction resets or preloads the PLS internal input counter. If the "offset" argument is left out, the counter is set to zero. The internal counter is only used if the PLS is set to rotary operation by the PLS ROTARY command. In a linear PLS, the internal input count is always equal to the source parameter.

The following example resets the internal counter on PLS 7 to 1000 counts:

Usage example:
```
PLS7 RES 1000
```

PLS ROTARY Set PLS rotary length

Format: PLS index ROTARY { length }

Units: input counts

This instruction, available with A2 (or later) firmware only, sets the PLS rotary length. Issuing a PLS ROTARY command with no argument will display the current setting. The default rotary length is 0 counts.

If the rotary length is zero, the PLS is linear and the output will be zeroed if a table index is generated that lies outside the boundaries of the PLS array. The internal input count is always equal to the current value of the source parameter when a PLS is linear.

If the rotary length is non-zero, the source parameter is used to generate an input count that "wraps-around" by the given length (modulus) before it is used to generate a table index. Note that this only affects the internal PLS input count and that it is still possible to generate table indexes outside of the array boundaries. While in rotary mode, the count can be modified with the PLS RES command.

The following example sets PLS 6 rotary length to 2000 counts:

Usage example:
```
PLS6 ROTARY 2000
```

PLS FLZ Set PLS index offset

Format: PLS index FLZ { offset }

Units: array entries

This instruction sets the index offset. Issuing a PLS FLZ command with no argument will display the current setting. The default index offset is 0 array entries.

The PLS SRC is used to generate an input count that is multiplied by the PLS RATIO and then added to the PLS FLZ to generate a table index. The table index is then used to retrieve an array entry which is transferred into the PLS DST parameter.

The following example sets the offsets PLS 3 by 10 array entries:

Usage example: `PLS3 FLZ 10`

PLS MASK Set PLS output bit mask

Format: PLS index MASK { mask }

This instruction sets the PLS output bit mask. Issuing a PLS MASK command with no argument will display the current setting. The default mask setting is -1 (0xFFFFFFFF) allowing all bits to be transferred.

When the PLS table entry is transferred it into the PLS DST parameter, the bit mask is used to determine which bits will be transferred. The bit transfer is done according to the following logic formula:

dest = (*dest* AND NOT *mask*) OR (*entry* AND *mask*)

The following example sets the mask for PLS 5 to 255, enabling the lower eight bits:

Usage example: `PLS5 MASK 255`

PLS RATIO Set PLS scaling ratio

Format: PLS index RATIO { ratio }

Units: array entries / input count

This instruction sets the scaling ratio. Issuing a PLS RATIO command with no argument will display the current setting. The default index offset is 1.0 entries / count.

The PLS SRC is used to generate an input count that is multiplied by the PLS RATIO and then added to the PLS FLZ to generate a table index. The table index is then used to retrieve an array entry which is transferred into the PLS DST parameter.

The following example sets the scaling ratio of PLS 2 to 0.25 entries / count:

Usage example: `PLS2 RATIO 0.25`

Format: PLS index ON

This instruction enables PLS update. When a PLS is active, a table index is calculated from the PLS SRC parameter. The table index retrieves an entry from an array and then transfers it into the PLS DST parameter. This sequence takes place every interrupt.

The following example enables the update of PLS 4:

Usage example: `PLS4 ON`

PLS ON Enable PLS update

Format:

PLS index ON

This instruction enables PLS update. When a PLS is active, a table index is calculated from the PLS SRC parameter. The table index retrieves an entry from an array and then transfers it into the PLS DST parameter. This sequence takes place every interrupt.

The following example enables the update of PLS 4:

Usage example: `PLS4 ON`

PLS OFF Disable PLS update

Format: PLS index OFF

This instruction disables PLS update. The PLS output will remain in the state that it was when the PLS was turned off.

The following example disables the update of PLS 3:

Usage example: `PLS3 OFF`

SAMP Data sampling

Format: SAMP { channel } command { data }

Data sampling allows the monitoring of system parameters at the servo interrupt rate. Optionally, data may be also sampled at fixed frequency (i.e. every 25 milliseconds) or on the rising or falling edge of a bit flag.

A total of eight channels can be simultaneously filled from different parameter sources. For example, one channel can monitor actual position while another is monitoring output voltage for a given axis. The resulting information can then be transferred to an offline system for graphical plotting or tuning analysis.

The sample command is combined with other commands to prepare the system for data sampling. The following is a list of valid sample command combinations:

SAMP SRC	Set sample source
SAMP BASE	Set sample base
SAMP CLEAR	Clear sample channels
SAMP TRG	Set sample trigger

The following is a list of system parameters related to data sampling:

P6912 Sample Array Index

P6913 Sample Trigger Index

P6914 Sample Timer Clock

P6915 Sample Timer Period

P6912 Sample Array Index

Indicates where the next samples are going to be put in the user defined sample arrays. During sampling, if the index is greater than or equal to the size of the array, that channel is tagged as being full. If all channels are full, the index is reset and the "trigger armed" and "in progress" flags are cleared. This allows different channels to have arrays of different lengths.

P6913 - Sample Trigger Index

Set with the SAMP TRG command and is stored as a one's complement number (to allow triggering on minus zero.) A number greater than or equal to zero will trigger on an active state or a rising edge depending on the setting of the sample mode flag. A value less than zero is bitwise inverted and triggers on an inactive state or a falling edge.

P6914 - Sample Timer Clock

Indicates the number of milliseconds remaining before a sample will be taken. This value is normally zero unless the sample timer period has been set. Whenever a sample is taken, this parameter is loaded with the value in sample timer period.

P6915 - Sample Timer Period

Loaded into the sample timer clock whenever a sample is taken. This parameter is normally zero, indicating that samples should be taken at the servo interrupt rate. For edge triggered sample operation, the period indicates the number of milliseconds that will pass after an edge before a sample is taken.

Related System Flags:

The following is a list of system flags related to data sampling:

BIT104 - Sample Trigger Armed

BIT105 - Sample In Progress

BIT106 - Sample Mode Select

BIT107 - Sample Trigger Latched

BIT104 - Sample Trigger Armed

Enables monitoring of the data sample trigger which will eventually set the sample in progress flag. This flag is cleared when all of the sample channels have been filled, indicating that the sample has completed.

BIT105 - Sample In Progress

Enables an actual sample to be taken and is normally set by a sample trigger condition but can also be set manually. The flag is cleared when all of the sample channels have been filled. It is also cleared after every sample if in the edge trigger mode. This is to prevent multiple samples from being taken on the edge trigger condition.

BIT106 - Sample Mode Select

Selects either the continuous or edge trigger mode of sampling. In the continuous mode, a trigger condition will set the sample in progress flag, causing a sample to be taken every servo interrupt (or sample period) until all of the sample channels have been filled. In the edge trigger mode, a trigger edge will set the sample in progress flag which is then cleared after the single sample has been taken.

BIT107 - Sample Trigger Latched

Tracks the previous state of the trigger condition for detecting trigger edges. If a trigger condition is detected and the previous trigger condition was false, an edge trigger will occur. Normally, this flag is not modified by user programs.

Code Execution Outline:

```
if  ( sample trigger armed )
if  ( trigger condition met )
if  ( ( level trigger ) or ( edge trigger
and not trigger latched ) )
set sample in progress
set trigger latched flag
else clear trigger latched flag

if  ( sample in progress )
if  ( sample clock > 0 )
update sample clock
if  ( sample clock = 0 )
sample clock = sample period
sample active channels
increment sample index
if  ( channels full )
clear sample armed flag
clear sample active flag
sample index = 0
if  ( edge trigger )
clear sample active flag
```

Usage example: The following example takes a 500 samples of axis 0 current position and output signal at the default servo interrupt rate of 2 kHz (250 milliseconds total sample time) :

```
PROG0
DIM SA1
DIM SA0(500)
SAMP CLEAR: REM reset sample defaults
```

```
SAMP0 SRC P12290: REM axis 0 current
position
SAMP0 BASE LA0: REM store data in LA0
SAMP1 SRC P12319: REM axis 0 output signal
SAMP1 BASE SA0: REM store data in SA0
SAMP TRG 516: REM master 0 in motion flag
```

SAMP SRC Set sample source

Format: SAMP channel SRC parameter

This instruction selects a source for the given sample channel. Sample channels are numbered 0 through 7. The source 'parameter' can be either a system parameter from Appendix B or any user defined parameter.

When sampling, the channel will transfer information from the source into the array set by the SAMP BASE command. The source and base should both be of the same type since no data conversion is done during the transfer.

The following example sets SAMP 0 source to AXIS 0 actual position (P12290) and SAMP 1 source to AXIS 0 output signal (P12319) :

Usage example:

```
SAMP0 SRC P12290
SAMP1 SRC P12319
```

SAMP BASE Set sample base

Format1: SAMP channel BASE LA index

Format2: SAMP channel BASE SA index

This instruction selects a storage array base for the given sample channel. Sample channels are numbered 0 through 7. The 'base' parameter can either a 32-bit long integer array (LA) or a 32-bit floating point array (SA).

When sampling, the channel will transfer information into this array from the source set by the SAMP SRC command. The source and base should both be of the same type since no data conversion is done during the transfer.

The following example ties sample channel 0 to the long integer array LA0 and then ties channel 1 to the 32-bit floating point array SA0:

Usage example:
```
SAMP0 BASE LA0
SAMP1 BASE SA0
```

SAMP CLEAR Clear sample channels

Format: SAMP CLEAR

This instruction clears out all of the system parameters and flags which are related to data sampling. It also clears out any the internal pointers which may have been set with the SAMP SRC and SAMP BASE commands.

Usage example:
```
SAMP CLEAR
```

SAMP TRG Set sample trigger

Format 1: SAMP TRG + index

Format 2: SAMP TRG - index

This instruction sets the trigger condition to be monitored when the sample trigger armed flag is set. A positive index will cause a trigger to occur on an active state or a rising edge, depending on the setting of the sample mode flag. A negative index will cause a trigger on an inactive state or a falling edge.

```
The following example will start sampling
when MASTER 0 starts moving ( BIT 516 ) :
```

Usage example:
```
SAMP TRG 516
```

Generic Instruction Expression Reference

CHR$ Character string

Format: CHR$(code)

This function returns a string of one character as defined by the given "code". The valid range for "code" is 0 to 255. If the value is outside that range, an error is returned.

Usage example:

```
PRINT CHR$(88)
Example output: X
```

GETCH Wait for a character

Format: GETCH(devicenumber)

This function returns a one character string from a device. If there is no character waiting to be read from the device, the function will wait until one becomes available.

The valid range for "devicenumber" is 0 to 3. Each program has it's own device #0 which is used as its default device. Devices #1 through #3 are board-wide system resources that can be opened and used from within any program or from any system or program prompt.

Usage example:

```
DIM $V(1,10)
OPEN "COM1:9600,N,8,1" AS #1
PRINT #1,
PRINT #1, "Press any key to continue"
$V0 = GETCH(1)
PRINT #1, "Program terminated"
CLOSE #1
```

INKEY$ Return a character

Format: INKEY$(devicenumber)

This function returns a one character string from a device. If there is no character waiting to be read from the device, the function will return a null string.

The valid range for "devicenumber" is 0 to 3. Each program has it's own device #0 which is used as its default device. Devices #1 through #3 are board-wide system resources that can be opened and used from within any program or from any system or program prompt.

Usage example:

```
DIM $V(1,10)
OPEN "COM1:9600,N,8,1" AS #1
PRINT #1,
$V0 = UCASE$(INKEY$(1))
IF ($V0 = "A") PRINT #1, "Apple"
IF ($V0 = "B") PRINT #1, "Banana"
IF ($V0 = "C") PRINT #1, "Coconut"
IF ($V0 = "X") GOTO 200
GOTO 130
PRINT #1, "Program terminated"
CLOSE #1
```

INSTR String search

Format: INSTR(stringexpression1, stringexpression2)

This function returns the position of "stringexpression2" within "stringexpression1". If the second string can not be located within the first, the function returns zero.

If the first string is a null string, the function returns a zero. If the second string is a null string and the first string has a length greater than zero, the function returns a one.

Usage example:

```
PRINT INSTR("ABCDEFG","CDE")
Example output:  3
```

KBHIT Check for waiting character

Format: KBHIT(devicenumber)

This function checks a device to see if a character is waiting to be read. If there is no character waiting to be read, the function will return a zero. Otherwise, the function will return a negative one (-1) indicating success.

The valid range for "devicenumber" is 0 to 3. Each program has it's own device #0 which is used as its default device. Devices #1 through #3 are board-wide system resources that can be opened and used from within any program or from any system or program prompt.

Usage example:

```
REM --- main program
DIM $V(1,80)
OPEN "COM1:9600,N,8,1" AS #1
PRINT #1,
PRINT #1, CHR$(65 + RND(26));
IF (KBHIT(1)) GOTO 170
GOTO 140
GOSUB 200 : REM fetch command
IF (UCASE$($V0) = "EXIT") GOTO 300
GOTO 140
REM --- command input
PRINT #1,
INPUT #1, "Command?", $V0
RETURN
REM --- program shutdown
PRINT #1, "Program terminated"
CLOSE #1
```

LCASE$ Convert to lower case

Format: LCASE$(stringexpression)

This function returns a string with all letters in lower case. This function is useful for making string comparisons that are not case sensitive.

Usage example:

```
PRINT LCASE$("AbCdEfG")
Example output: abcdefg
```

LEFT$ Left string

Format: LEFT$(stringexpression, n)

This function returns the leftmost "n" characters of the given string. If "n" is greater than the length of the string, the entire string is returned.

Usage example:

```
PRINT LEFT$("ABCDEFG", 3)
Example output:  ABC
```

LEN String length

Format: LEN(stringexpression)

This function returns the length of the given string expression.

Usage example:

```
PRINT LEN("ABCDEFG")
Example output:  7
```

MID$ Middle string

Format: MID$(stringexpression, start, length)

This function returns characters from the middle of the given string. If "start" is greater than the length of the string, the function returns a null string. If "length" would go beyond the end of the string, the function returns only the characters from "start" to the end of the string.

Usage example:

```
PRINT MID$("ABCDEFG", 2, 5)
Example output:  BCDEF
```

RIGHT$ Right string

Format: RIGHT$(stringexpression, length)

This function returns the rightmost "n" characters of the given string. If "n" is greater than the length of the string, the entire string is returned.

Usage example:

```
PRINT RIGHT$("ABCDEFG", 3)
Example output:  EFG
```

SPACE$ String of spaces

Format: SPACE$(n)

This function returns a string of "n" spaces.

Usage example:

```
PRINT "**********"
PRINT "*"; SPACE$(8); "*"
PRINT "**********"
LRUN
Example output: **********
                *        *
                **********
```

STR$ Convert numeric to string

Format: STR$(value)

This function converts "value" to a string and returns the string.

Usage example:

```
DIM $V(1, 10)
$V0 = STR$(1.234)
PRINT $V0
LRUN
Example output: 1.234
```

STRING$ String of characters

Format 1: STRING$(length, code)

Format 2: STRING$(length, stringexpression)

This function returns a string of characters either defined by the given "code" or the first character of a string expression.

Usage example:

```
PRINT STRING$(5, 88)
PRINT STRING$(10, "*")
LRUN
Example output: XXXXX
                **********
```

UCASE$ Convert to upper case

Format: UCASE$(stringexpression)

This function returns a string with all letters in upper case. This function is useful for making string comparisons that are not case sensitive.

Usage example:

```
PRINT UCASE$("AbCdEfG")
Example output:  ABCDEFG
```

VAL Convert string to numeric

Format: VAL(stringexpression)

This function converts the "stringexpression" to a numeric value and returns the value. Leading spaces and tab characters are ignored and the conversion continues until a character is reached that cannot be recognized as part of a number. If the conversion fails, the function returns a zero.

Usage example:

```
DIM DV(1)
DV0 = VAL("1.234")
PRINT DV0
LRUN
Example output:  1.234
```

Index

O

P

R

S

U

V

W

Since 1979, EMERSON Motion Control, a subsidiary of the Emerson Electric Company, has been a leader in the development and manufacturing of motion control equipment and software. EMERSON Motion Control continues to lead the industry in supplying motion control devices designed to improve production efficiency.

For more information about EMERSON Motion Control products and services, call (800) 39-SERVO or contact our website at www.emersondrivesolutions.com.

EMERSON Motion Control
Subsidiary of Emerson Electric Co.
12005 Technology Drive
Eden Prairie, Minnesota 55344
U.S.A.

Sales: (952) 995-8000 or (800) 39-SERVO
Service: (952) 995-8033
Fax: (952) 995-8011

Printed in U.S.A.

EMERSON
MOTION CONTROL