Software Reference

Part III
Software Reference

- *Summary of Changes*
- *The IMS Terminal Software*
- *Introduction to LYNX/MicroLYNX Programming*
- *Functional Groups*
- *Language Reference*
- *ASCII Table*
- *Error Table*
- *Factory Defaults*
- *Establishing Communication Using Hyperterminal*

IMS™ INTELLIGENT MOTION SYSTEMS, INC.
Excellence in Motion™

370 North Main St., P.O. Box 457, Marlborough, CT 06447 U.S.A.
Phone: 860/295-6102, Fax: 860/295-6107
Internet: www.imshome.com, E-Mail: info@imshome.com

This Page Intentionally Left Blank

Table Of Contents

List of Figures

SUMMARY OF CHANGES

The items listed below have been added or modified since the last revision.

New or Modified Items

05/19/04
Modified the description of the command POSCAP on Page 110.
Added POSCAP Syntax Example on Page 110.

03/10/05
Modified the description and *Notes* of the command DN on page 73
Modified the description of the command DRVTP on page 74.

12/21/05
Changed Baud Rate Variable to “38.4:38,400” on Page 59.

This Page Intentionally Left Blank

SECTION 1

The IMS Terminal Software

Section Overview

This section covers the usage of the IMS Terminal software, which is included with your MicroLYNX/LYNX product. There are two main benefits to be gained by using this software: First and most importantly, it includes the upgrade utility which allows you to upgrade your MicroLYNX/LYNX product. The MicroLYNX/LYNX Firmware cannot be upgraded without this utility! Second, it features a Program Editor Window for writing programs, and a Terminal Window for communicating with your MicroLYNX/LYNX system. Both the Program Editor and Terminal Window can be open at the same time. Each Window can have its preferences configured independantly in case you have more than one MicroLYNX/LYNX product connected to different COMM ports on your PC. This program also eliminates the need to use two separate programs such as Notepad and HyperTerminal, to program your system. Covered in this section are:

- Installing the IMS Terminal Software
- Using the IMS Terminal Software
- Upgrading the MicroLYNX/LYNX Firmware

Installing IMS Terminal

System Requirements

- IBM Compatible PC.
- Windows 9x (95/98) or Windows NT (Windows NT4.0 SP6, Windows 2000 SP1, Windows XP)
- 10 MB hard drive space.
- A free serial communications port.

Installation

The IMS Terminal software is a programming/communications interface. This software was created by IMS to simplify programming and upgrading the MicroLYNX/LYNX Systems. The IMS Terminal is also necessary to upgrade the software in your MicroLYNX/LYNX Systems. These updates will be posted to the IMS web site at www.imshome.com as they are made available.

To install the IMS Terminal to your hard drive, insert the IMS Product CD into your CD-ROM Drive. The CD should autostart to the IMS Main Index Page. If the CD does not autostart, click “Start > Run” and type “x:\IMS.exe” in the “Open” box and click OK.

NOTE: “x” is your CD ROM drive letter.

1) The IMS CD Main Index Page will be displayed.

Figure 1.1: IMS CD Main Index Page

2) Place your mouse pointer over the MicroLYNX Icon. The text message "LYNX Family Product" will be displayed. This verifies you have selected the correct software.
3) Click the MicroLYNX Icon. This opens the LYNX Product Family Page.

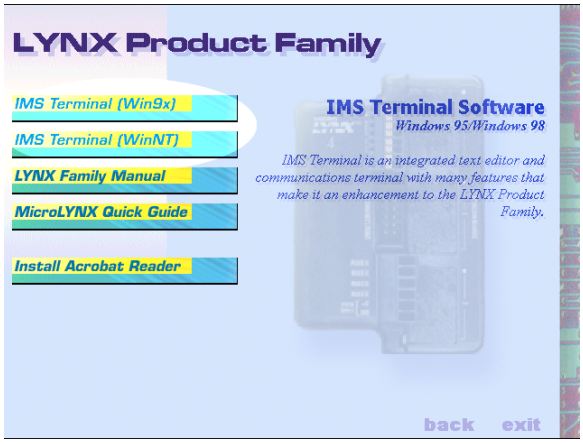


Figure 1.2: IMS CD LYNX Product Family Page

4) Place the mouse pointer over the menu and select IMS Terminal (Win9x) or IMS Terminal (WinNT). The displayed text will again verify your selection. Click your selection and the "Setup" dialog box will be displayed.

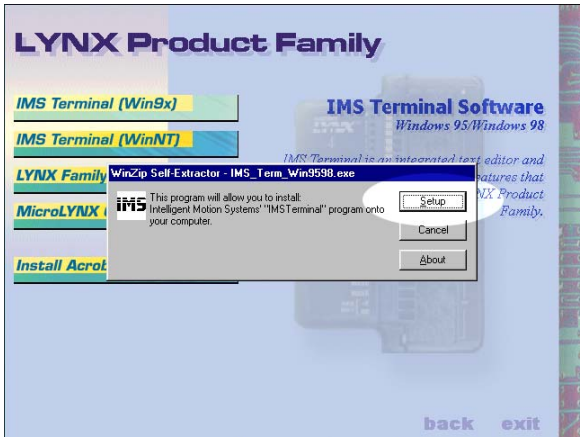


Figure 1.3: IMS CD Software Setup

5) Click SETUP in the Setup dialog box and follow the on-screen instructions.

Once IMS Terminal is installed the Communications Settings can be checked and/or set.

The communications settings are configured by means of the “Preferences Dialog Box”. The preferences dialog gives the user the ability to set the format for text size, font and color, as well as general communications settings. The optimum communications settings for the MicroLYNX/LYNX are set by default. After the IMS Terminal Software is installed you may start it and perform the configuration.

1) Open the IMS Terminal by clicking Start>Programs>IMS Terminal>IMS Term.
The following screen will be displayed.

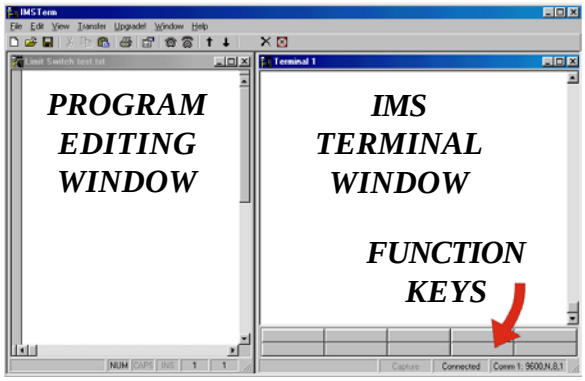


Figure 1.4: Main IMS Terminal Page

The left window is the Program Editing Window. The right window is the IMS Terminal Window. Resident programs and immediate commands can be executed, stopped and tracked from the Terminal Window.

2) You must select or verify the Communications Port that you will using.
   a) On the Menu Bar: click <Edit> <Preferences> or right click in the Terminal Window and click Preferences, or click the Preferences Button  on the main Tool Bar to display the Preferences Dialog Box.

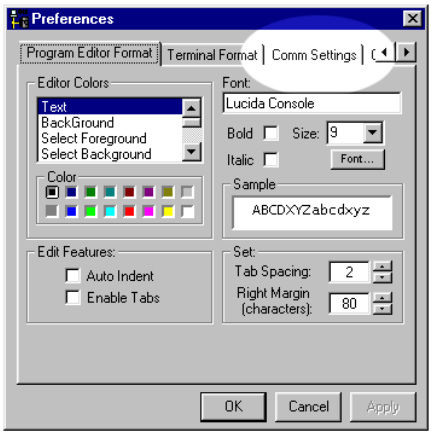


Figure 1.5: IMS Terminal Preferences Dialog Box

The Preferences Dialog Box allows you to select window colors and fonts for the Text Editing Window and Terminal Window as well as Communications Setup.

   d) Click the “Comm Settings” tab at the top of the dialog box. The Comm settings page will be displayed.

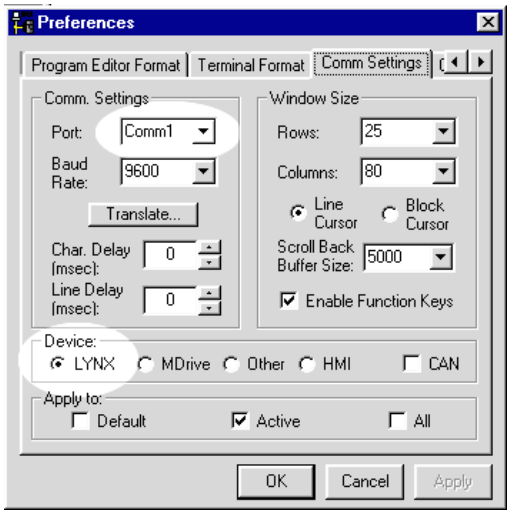


Figure 1.6: IMS Terminal Comm Settings Dialog Box

e) Under "Device" near the bottom of the box verify that "LYNX" is selected. The BAUD rate is already set to the MicroLYNX/LYNX default. Do not change this setting until you have established communications with the MicroLYNX/LYNX.

If you change the BAUD rate setting, power will have to be cycled for the change to take effect. Ensure that the IMS Terminal preferences are adjusted for the new BAUD settings.

f) Verify the Comm Port you are using.

g) The "Window Size" settings are strictly optional. You may set these to whatever size is comfortable to you.

h) Click "APPLY" and "OK". The settings will be saved automatically.

3) Verify all connections are made and apply power to the system. The following sign-on message should appear in the Terminal window:

```
Program Copyright 2001-2003 by:
Intelligent Motion Systems, Inc.
Marlborough, CT 06447
VER = xxxxxx       SER = Axxxxxx
```

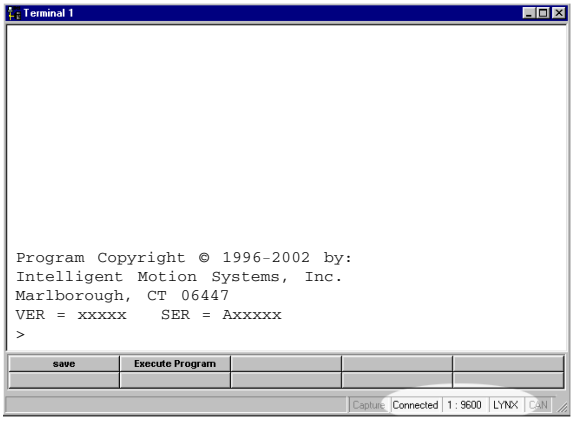


Figure 1.7: IMS Copyright Statement in Terminal Window

If you can see this sign-on message then you are up and running! If the sign-on message does not appear, try using a software reset. Hold down the “Ctrl” key and press “C” (^C). If the sign-on message still does not appear then there may be a problem with either the connections, hardware or software configuration of the MicroLYNX/LYNX or Host PC.

There are also indicators at the bottom of the page (See the previous figure) that show whether you are Connected or Disconnected, the current Baud Rate and the type of device (LYNX) for which the IMS Terminal is configured. These three items may be changed directly by double clicking on them.

Double Click on “Connected” and the system will disconnect. Double Click on “Disconnect” and the system will connect.
Double Click on the Baud Rate and the preferences page will open so you can change it.
Double Click on the “LYNX” and the preferences page will open with the option to change the drive. (For this MicroLYNX/LYNX application you would not change the “LYNX”.)

N Many of the commands you will be using work in both the Program Editor Window and the Terminal Window. You must have the proper window selected before activating the command.

Using the IMS Terminal Software

The IMS Terminal software is an easy to setup and use interface for MicroLYNX/LYNX programming. It is also required to upgrade the firmware in the MicroLYNX/LYNX System.

IMS Terminal Tool Bar

The IMS Terminal Tool Bar is configured with all the necessary functions to operate IMS Terminal.

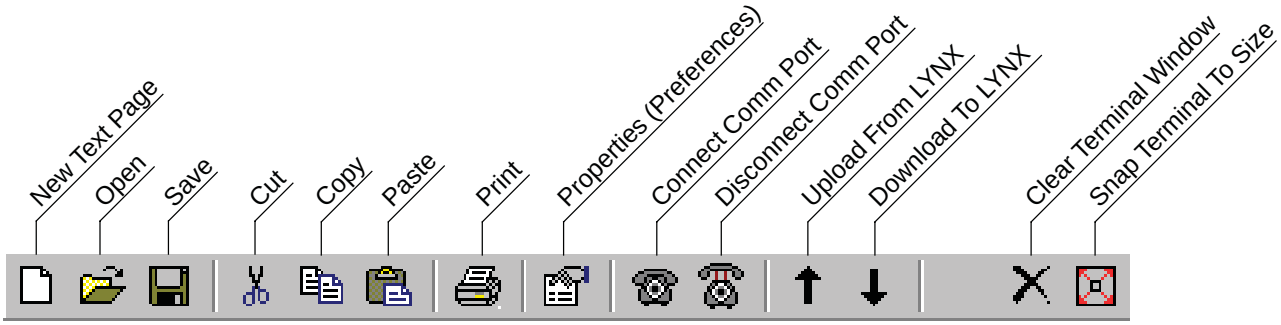


Figure 1.8: IMS Terminal Tool Bar

Creating, Downloading and Uploading Programs

Existing programs may be edited in the Program Editor Window from a file on a disk, a file on the hard drive or a file uploaded from the MicroLYNX/LYNX. You may also create a new program in the Program Editor Window.

NOTE: It would be beneficial to have your system connected and running and perform these steps as they are outlined.

Creating a New Program

Before you create a program you must have a new Program Editor Window open. Follow these steps:

1) Click on the Drop-Down Menu "View". The following dialog box will be displayed:

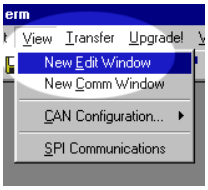


Figure 1.9: Drop-Down Menu for New Edit Window

2) Click on "New Edit Window". The following dialog box will be displayed:

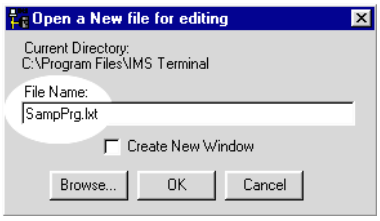


Figure 1.10: Dialog Box for Naming New Editor Window

3) There must be a file name in order to open the new window. If there is no file name the "OK" button will not be highlighted. Name this file <**SampPrg.lxt**>. The <lxt> extension designates programs for the MicroLYNX/LYNX.

4) Click "OK" and the new Program Editor Window will be displayed.

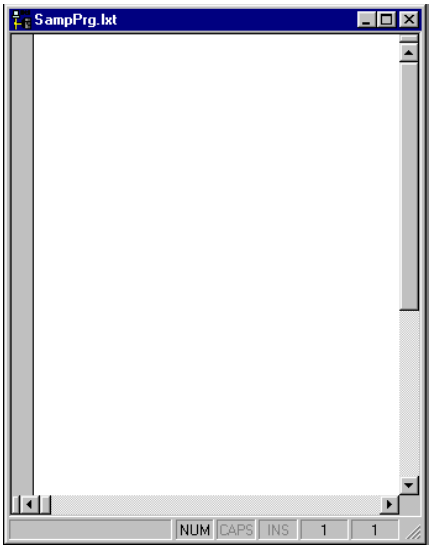


Figure 1.11: New Program Editor Window Named "analog.mxt"

Naming the program with the <lxt> extension automatically formats the text color and makes most of the characters appear in upper case. When you type a program the text will be color coded. In complex programs it may be difficult to read the text easily. By formatting indents, the overall appearance and readability will be greatly improved.

Formatting the Program Text

To format the text for indents you need to call up the “Preferences” dialog box. Click the “Program Editor Format” tab at the top of the box. The screen below will be displayed. In the “Edit Features” block (1) click on the small box to the left of “Auto Indent” and verify there is a check mark (✔) in the box. This will enable Automatic Indents. Once you indent your text with the “Tab” key, all subsequent lines will adopt the same indent. Simply backspace to return to the left margin. There is also an “Enable Tabs” option. If this box is checked, tabs will be inserted into your text. If the “Tabs” option is disabled, character spaces will be inserted. For this example the “Enable Tabs” will be turned off. In the “Set” block (2) you may also set the tab spacing. The default is 2 characters. When completed, click “Apply” and then click “OK”.

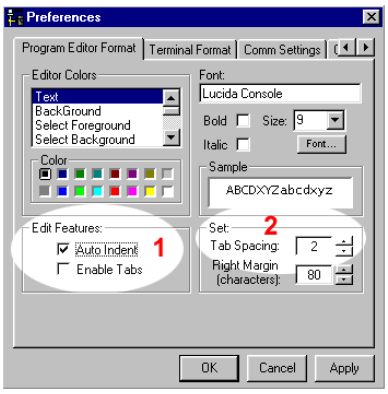


Figure 1.12: Program Editor Preferences Page

Individual preference will govern how you set up your indents. The format illustrated below is most commonly used. All of the set Variables and Program Modes are left aligned. All the Labels are indented 2 characters or 1 tab. The remaining commands are indented 4 characters or 2 tabs. Indent your text by pressing the “Tab” key.

A program can now be typed into the new Program Editor Window. For this example we will use the sample program shown here. Type the program as it is shown. You can type upper or lower case. Be sure to put all spaces in as they are shown. It is not necessary to put in the comments but they are allowed in the program provided they begin with an apostrophie (‘).

```
PGM 1                       'Enter program mode at line #1
POS=0                       'Set present position to zero
MUNIT=51200                 'Set Motor Units to 51,200 Steps/User Unit
MSEL=256                    'Motor resolution = 256 uSteps/Full Step
VM=1                        'Set Velocity Max. to 1 rev/sec.
ACCL=50                     'Set Acceleration to 50 revs/sec
DECL=50                     'Set Deceleration to 50 revs/sec
LBL TstPgm                  'Label the program TstPgm
MOVR 3                      'Move Relative 3 Revs from current pos.
HOLD 2                      'Hold prog. exec. until motion complete
DELAY 250                   'Delay 1/4 second
PRINT "position = ", POS    'Print the present position
MOVA 0                      'Move absolute to the zero position
HOLD 2                      'Hold prog. exec. until motion complete
PRINT "position = ", POS    'Print the present position
END                         'End the program
PGM                         'Exit the program mode
```

As you type, the text will be automatically formatted and color coded for the MicroLYNX/LYNX. When you edit or type new commands they will appear black and will then be automatically changed to the proper color and case when you press “Enter”. If you type in all lower case characters, upon pressing “Enter” part or all of it will be changed to upper case characters. This is an indicator that the syntax was correct and accepted by the IMS Terminal. If the command line is changed to red with no uppercase characters it may be a bad command. Add tabs where they are desired. When complete, your program should resemble the figure on the following page. Be sure to SAVE YOUR PROGRAM!

In the illustration below the default color coding is Dark Blue, Light Blue, Red, Green, Olive and Brown. Their designations are:

Dark Blue = Key Words

One Upper/One Lower Case = IMS Variables or Flags

All Upper Case = IMS Commands

Light Blue = Numerical Signs

Red = User Defined Data (Note that a line beginning with RED text is usually a bad command)

Green = Remarks (Not Shown)

Olive = Numerical Values

Brown = Text Strings in Quotes

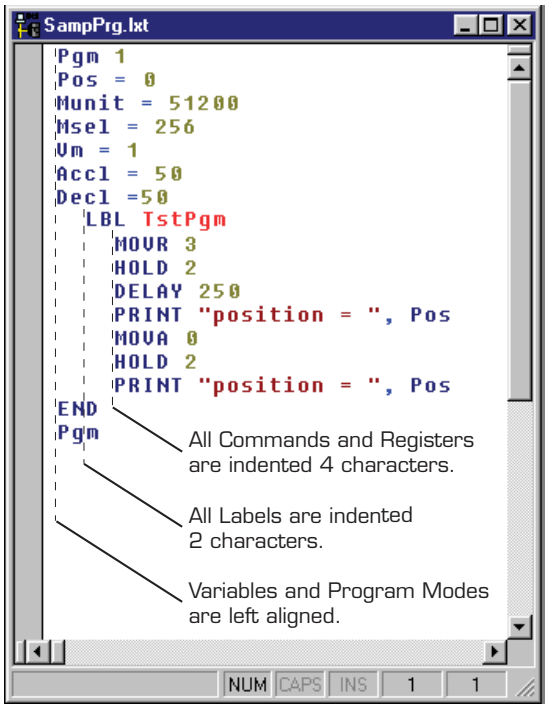


Figure 1.13: Formatted and Color Coded Program Text

NOTE: The indicator lines and labels are not part of the program. They have been added for illustration purposes only.

The colors may be changed to suit the user's preference. To change the colors call up the “Preferences” page. Click on the “Program Editor Format” tab at the top of the page. In the “Edit Colors” block you can set up your preferential colors for the different parts of your program. These changes will become the defaults after clicking “Apply” and re-saving your program.

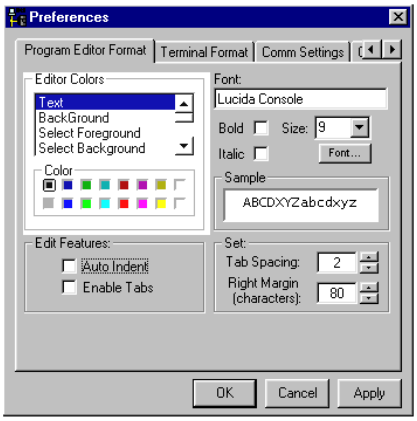


Figure 1.14: Dialog Box for Changing Text Colors in the Program Editor Window

Downloading a Program to the MicroLYNX/LYNX

There are two ways to download programs to the MicroLYNX/LYNX:

1) Directly from the Program Editor Window of the IMS Terminal.
2) From a file folder located on a hard drive or removable disk.

There are also two ways to enable the download dialog box.

1) Click the menu item “Transfer > Download”. The Download Dialog Box will open.

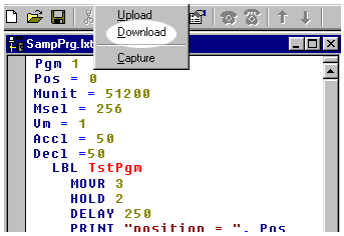


Figure 1.15: Program Download Drop-Down Menu

2) Click the Download Button on the Main Tool Bar. The Download Dialog Box will open.

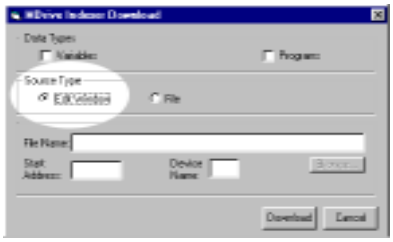


Figure 1.16: Program Download Dialog Box

Select the “Source Type > Edit Window” option, click download. The program will transfer to the MicroLYNX/LYNX.

If a Program has been previously created and stored, it may be downloaded by selecting “Source Type > File” on the dialog box and typing in a drive location:\file name in the “File Name” box in the dialog, or by browsing to the file location. Ensure the programs have been saved with the <**lxt**> extension.

NOTE: The program is not downloaded to the Terminal Window. It is downloaded directly to the MicroLYNX. What is shown in the Terminal Window is an echo of the downloaded program.

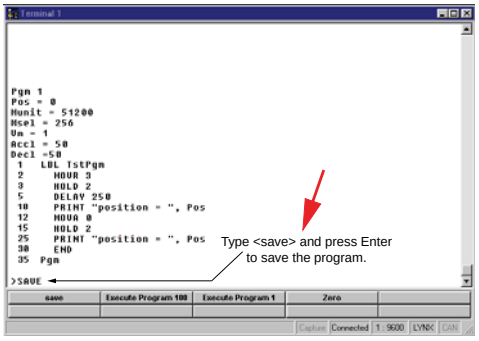


Figure 1.17: IMS Terminal Window Displaying Downloaded Program

NOTE: The program is not downloaded to the Terminal Window. It is downloaded directly to the MicroLYNX/LYNX. What is shown in the Window is an echo of the downloaded program.

NOTE: Because the program is downloaded directly, the system must be powered up and the sign-on message must be displayed (communicating).

NOTE: When the program is downloaded, the color of all characters will be changed to black and line numbers will be added.

NOTE: After the program is downloaded it must be saved. Type <save> next to the cursor and press Enter to save the program.

Uploading a Program From the MicroLYNX/LYNX

There are two ways to upload programs from the MicroLYNX/LYNX:

1) Directly to the Program Editor Window of the IMS Terminal.
2) To a file folder located on a hard drive or removable disk.

There are also two ways to enable the upload dialog box.

1) Click the menu item “Transfer > Upload”. The Upload Dialog Box will open.
2) Click the Upload Button  on the Main Tool Bar. The Upload Dialog Box will open.

With the Upload Dialog Box open. Select the “Destination Type > Edit Window” option, click “Upload”. The program will transfer from the MicroLYNX/LYNX.
Programs may be uploaded from the MicroLYNX/LYNX to a text file by selecting “Destination Type > File” on the dialog and typing in a drive location:\file name in the “File Name” box on the dialog.

Setting the Programmable Function Keys

The IMS Terminal has the capability of programming up to 10 Function Keys, a feature typically found only in more advanced terminal programs. The Function Keys can be set to provide quick access to commonly used MicroLYNX/LYNX Immediate Mode commands, execute programs, or even hold entire programs up to 2048 characters.

To access the function key setup dialog box, right-click the function key area at the bottom of the Terminal Window. The window below will be displayed.

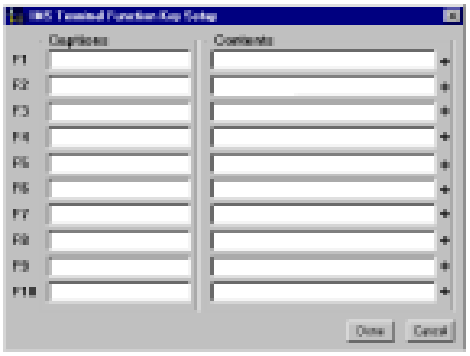


Figure 1.18: Function Key(s) Configuration Page

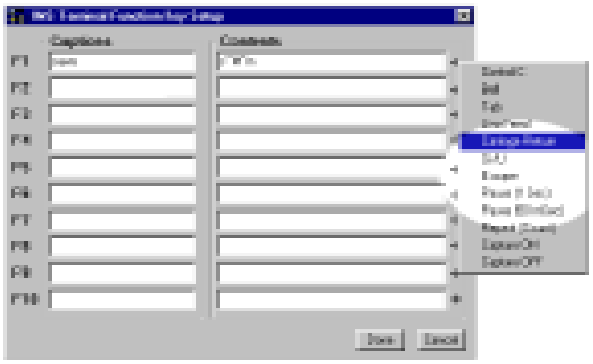


Figure 1.19: Entering Data for the Function Key(s)

To setup the function keys:

1) In this example the “Save” command is used. Enter “Save” in the Captions text field, this will be displayed on the function button.
2) Enter the text string in the Contents field consisting of MicroLYNX commands and ASCII codes. The command “Save” is entered.

Each command must be terminated with a Carriage Return (^M) and a pause time. Typically 50 msec (^m) is sufficient.

A fly-out dialog can be brought up by clicking the arrow on the right of the function key “Contents” field. This enables the programmer to embed common ASCII control codes in the function key text string.

3) Click “Done” to set the function.

To activate Function 1, Click the F1 Function Key or press the 

key on your keyboard.

Note: Holding the mouse pointer over the function key will display a small identification box which shows the Function Key number and the data it contains.

The Function Keys are numbered left-to-right: F1..F5 and F6..F10.

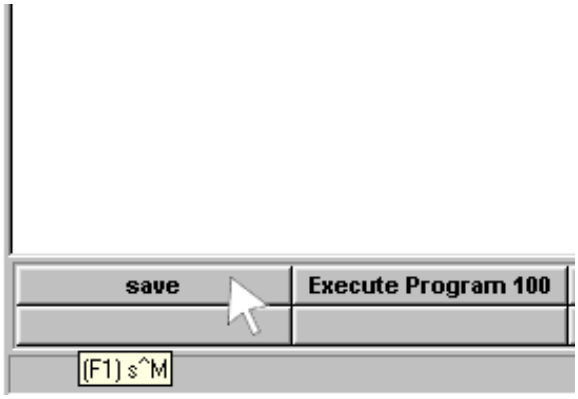


Figure 1.20: Activating a Function Key

Program Troubleshooting

The IMS Terminal offers several tools to help you troubleshoot and analyze programs. They are:

- ❒ Execute in Single Step Mode
- ❒ Execute in Trace Mode
- ❒ The Scroll Back Function
- ❒ The Capture Function

Single Step Mode

The Single Step Mode allows the user to execute a program in the Immediate Mode one line at a time. This will help the user to define problem areas by process of elimination. To use Single Step Mode, do the following:

It is recommended that you list (List) the program in the Terminal Window and either print it on paper or cut and paste it to another Program Edit Window. This will allow you to look ahead and see what line is coming up next.

1) Have the system and the program ready to run.
2) To run in Single Step Mode add a comma and the number two (2) to the execute command.
   Example: The Program Label is <aa>. Type `<EXEC aa,2>` and the program will run one line at a time.
3) Each line will be executed and listed in the Terminal Window and the Program will stop.
4) To execute and list the next line, press the Space Bar.
5) Press the Space Bar for each successive line until the program has completed.

While the program is executing, it will stop after each line is listed. At this time you may enter immediate commands such as velocity variables or actual moves as tests within the program. After entering immediate commands you may continue running in Single Step Mode by pressing the Space Bar again.

If you decide to cancel the Single Step Mode press the "Enter" key and the program will run in normal mode and finish, or press Escape (Esc) to abort the program.

Trace Mode

The Trace Mode allows the user to run a program and list each line as it is executed. Running Trace Mode in conjunction with the Scroll Back Function or the Capture Function will enhance your program troubleshooting tasks. To run Trace Mode:

1) Have the system and the program ready to run.
2) To run in Trace Mode add a comma and the number one (1) to the execute command.
   Example: The Program Label is <aa>. Type `<EXEC aa,1>` and the program will run in Trace Mode with each line executed and listed in the Terminal Window.
3) Each line can now be analyzed.

On very large programs all of the lines may not be displayed if the "Scroll Back Buffer" value is set too low. The Scroll Back Buffer can be set to a higher value allowing you to Scroll Back farther in the program .

The Scroll Back Buffer

The “Scroll Back Buffer” function for the IMS Terminal Window can be set to different line values. It allows you to scroll back in the program that has already been displayed in the Terminal Window. It can be very useful when troubleshooting a long program.
To set the Scroll Back Buffer:

1) Open the Preferences Page for the IMS Terminal Window.
2) Click on the “Comm Settings” tab at the top of the page. The following screen will be displayed.
3) In the highlighted area in the Figure below you will see a dialog box for “Scroll Back”.
4) To the left of the current value there is a small arrow to drop down the list. The list covers up to 2000 lines. You can select a value up to 2000 lines from the list.
5) If you wish to set the value higher, DO NOT open the drop down list. Simply click on the displayed value to highlight it and type in the new value up to a maximum of 32,000 lines.

NOTE: The Scroll Back Buffer utilizes RAM to store the data. The greater you set the Scroll Back Buffer capacity the greater the amount of RAM used.

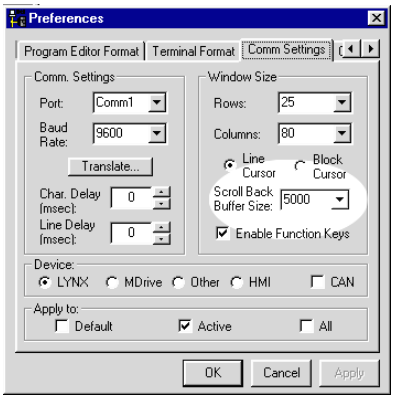


Figure 1.21: Setting the Scroll Back Buffer

The Capture Function allows you to capture Terminal Communications into a text file for the purpose of troubleshooting. You may have a program that fails after running a number of times. It may be from an accumulation of position errors or other factors. By enabling the Capture Function you can store an entire text file of the received communications to your hard drive for analysis.

Enable the Capture Function

The Capture function may be enabled through the drop-down menu under “Transfer”. When you click on “Capture” the “Capture Save” dialog box will be displayed.

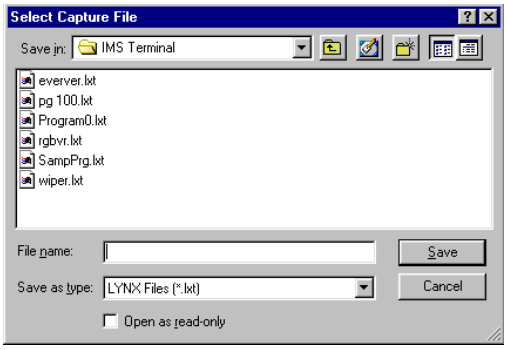


Figure 1.22: The Save Capture Dialog Box

Give the file you will be capturing a name and be certain to save it as a [.txt] file and click “Save”.

NOTE: The Capture Function may also be enabled through the Fly-Out menu on the Function Key configuration page by inserting it into the command string in the “Contents” line. However, the Capture Function can not be programmed with the Repeat command.

Upon clicking Save, the faded (disabled) Capture title below the Function Keys will change to “Capture ON” and to black letters.

Figure 1.23: Capture Off Indicator

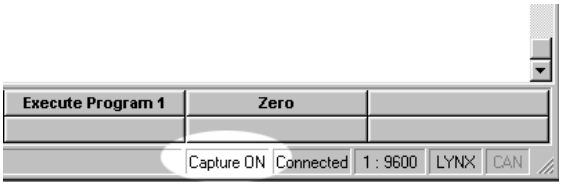


Figure 1.24: Capture ON Indicator

When the program is run, the data will scroll up the Terminal Window while a copy of the data is captured into the text file simultaneously.

Once the program stops, return to the “Transfer” Drop-Down menu and click on “Stop Capture”. The data that is currently in the Terminal Window is now also saved as the prenamed text file on your hard drive.

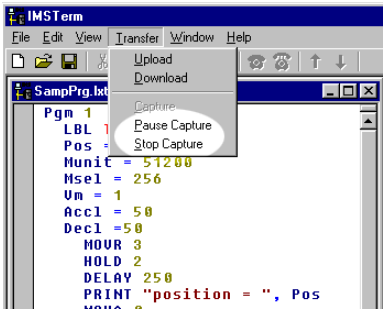


Figure 1.25: Stop Capture Command in Transfer Drop-Down Menu

Upgrading the MicroLYNX/LYNX Firmware

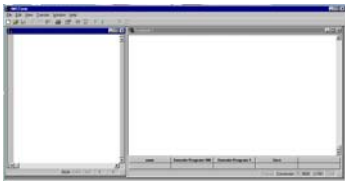
Before Upgrading

First download the version of firmware you wish to use for the upgrade. (www.imshome.com)

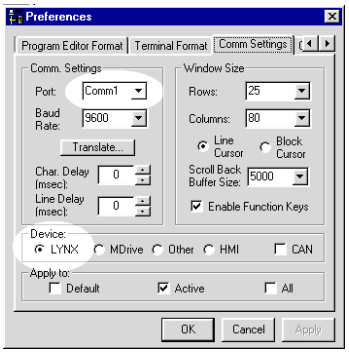
An isolated communications system free of electrical noise and interference is essential for trouble free communication.

During upgrades, the communication baud rate is switched from 9600 to 19,200 and is more susceptible to electrical noise. Your communications cable should be kept to a length of 6 feet.

1) Open “IMS Terminal ” software. The following screen will be displayed. The left panel is the Program Edit Window and the right panel is the Terminal Window. The Firmware Upgrade will superimpose several dialog boxes and instructions over these two windows.



2) Confirm that the terminal window is set for MicroLYNX/LYNX communication.
   - Right click in the Terminal Window.
   - Click “Preferences” near the bottom of the pop-up menu.
   - A “Preferences” dialog box will be displayed.
   - Click on the “Comm Settings” tab at the top of the box. The following page will be displayed.
   - Confirm that LYNX is selected in the “Devices” block.



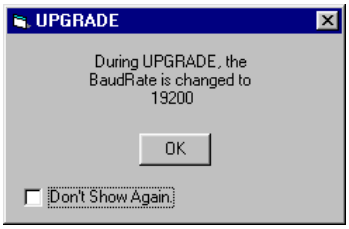
3) You may power up the MicroLYNX at this time but it is not necessary. You will have to cycle the power later on in this procedure. If you do power up at this time:
   - The sign-on message will appear

```
Copyright 2001-2003 by:
Intelligent Motion Systems, Inc.
Marlborough, CT 06447
VER = xxxxx SER = Axxxx
```

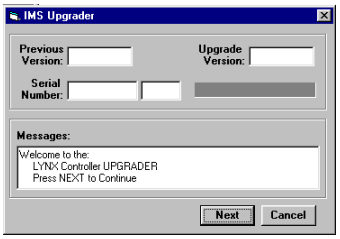
4) Check and/or reestablish communications if the sign-on message does not appear.

5) Click in the IMS Terminal Window to activate it and then click the “Upgrade” menu item on the IMS Terminal menu bar.

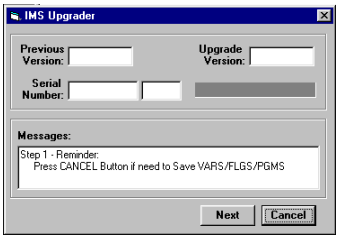
6) Message appears: “During upgrade, the baud rate is changed to 19,200.”
   - Click “OK”



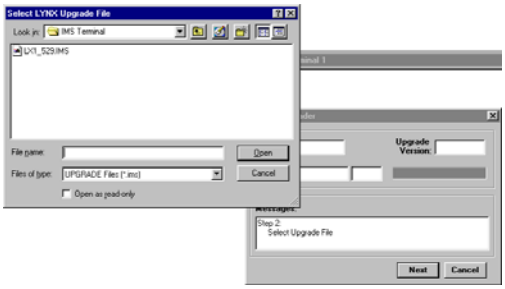
7) Message: “Welcome to the LYNX Controller Upgrader. Press next to continue.
   - You do not need to enter data in the windows. This will fill in automatically as you progress.
   - Click “Next”



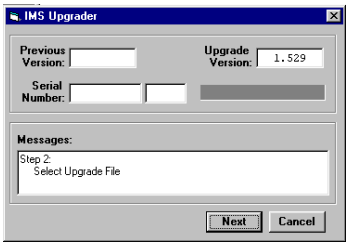
8) Message: Step 1 Reminder - Press Cancel if you need to save VARS/FLGS/PGMS.
   - Any Variables, Flags and Programs stored in the MicroLYNX will be erased during the Upgrade. If you do not have backup files, click Cancel and save them now.
   - After saving, re-enter the Upgrade Mode to this point.
   - Click “Next”.



9) The Windows Explorer page “Select LYNX Upgrade File” opens.
   - Browse and select the desired version of the upgrade file.
   - Click “Open” or double click the file.

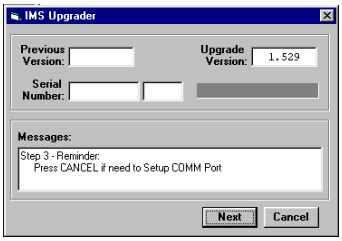


10) Message appears: Step 2 Select upgrade file.
   - The Upgrade Version will now appear in the Upgrade Version window.
   - Click “Next”



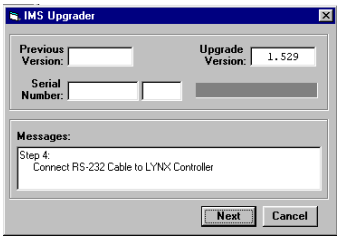
11) Message appears: Step 3 Reminder Press cancel if you need to setup Comm port.

- If the Comm port has not been setup previously, click Cancel and connect it now.
- Re-enter the Upgrade Mode to this point.
- Click “Next”



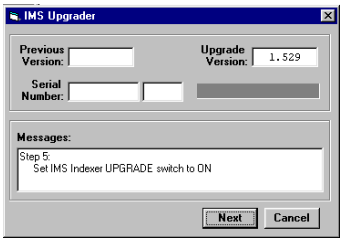
12) Message appears: Step 4 Connect RS-232 cable to the MicroLYNX/LYNX System.

- If the RS-232 has been connected previously, ignore this step. This is just a reminder.
- Click “Next”



13) Message appears: Step 5 Set the Indexer Upgrade Switch to “ON”.

- Click “Next”



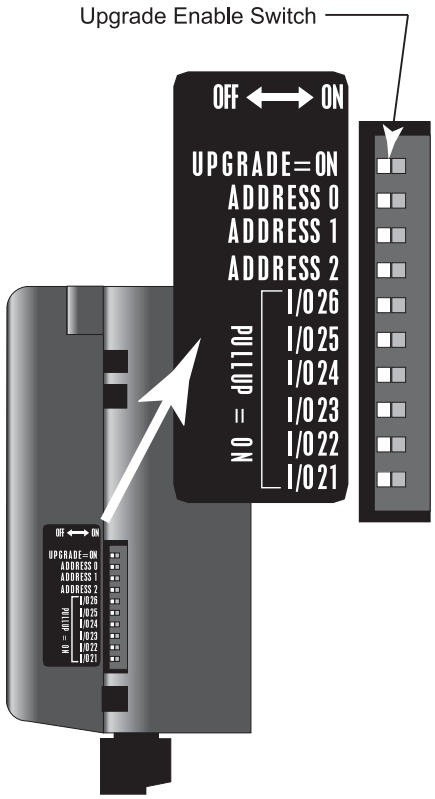
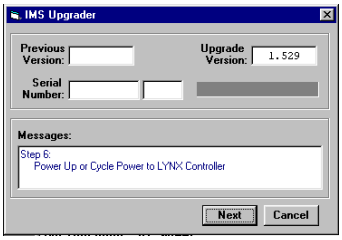


Figure 1.26: MicroLYNX Upgrade Enable Switch

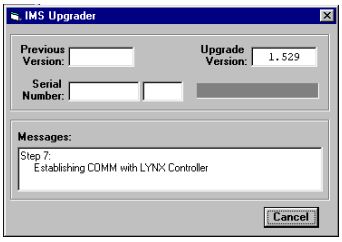
14) Message: Step 6 Power up or cycle power to MicroLYNX.

- Even if the unit has been previously powered up, you must cycle power in order for the Upgrade Switch to be recognized in the “ON” position.
- Click “Next”



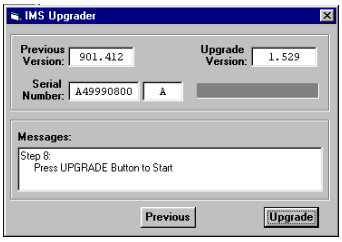
15) Message: Step 7 Establishing Comm with MicroLYNX/LYNX.

- Wait for step 8 to appear.



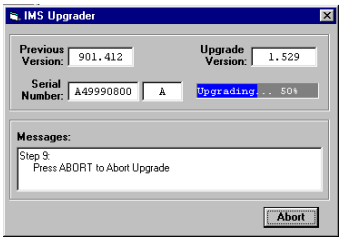
16) Message: Step 8 Press upgrade button to start.

- Click the upgrade button.



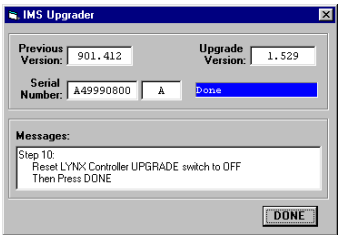
17) Message: Step 9 Press ABORT to abort upgrade.

- Monitor the progress in the “Upgrading…%” window.
- Step 10 will appear when DONE.



18) Message: Step 10 Resetting MicroLYNX/LYNX. Then Press DONE.

- Click “DONE”
- Upgrade window will close.



19) Type **`PRINT VER`** and the Version Number you upgraded with will be displayed.

- The > cursor will appear.
- The Upgrade is complete. The MicroLYNX is ready to run.

SECTION 2

Introduction to LYNX/MicroLYNX Programming

Section Overview

This section will cover the tools required to effectively program the LYNX/MicroLYNX product, the basic components of the LYNX/MicroLYNX Software, and the most commonly used commands and variables. The LYNX/MicroLYNX instruction set features a large arsenal of commands which allow it to be very flexible in its use in numerous applications. However, the basic commands will apply to most programs. *Section 4: LYNX/MicroLYNX Programming Language Reference* contains detailed descriptions of each instruction, variable, flag and keyword, as well as real-world usage examples for each.

Throughout this section, there a few things for you to note:

The word "True" and the number "1" are used interchangeably, as are "False" and "0". These refer to digital logic states. True will ALWAYS be equal to 1, False will ALWAYS equal 0.

The apostrophe character (') is recognized by the LYNX/MicroLYNX as a comment character. Any text in a program that follows an apostrophe will not be loaded into user memory space. It is a good practice to comment your programs as you are learning the LYNX/MicroLYNX Programming Language. This will be valuable in debugging your program as it will provide a step-by-step description of each program step. Below is a sample line of commented LYNX/MicroLYNX code:

```
ACCL=360          'Set the acceleration variable to 360 munits per second²
```

Tools Required:

Terminal

The terminal can be at a minimum a hand held terminal or a DOS driven terminal such as Pro Comm Plus. IMS recommends that the IMS Terminal software produced by IMS be used, however, either Terminal (Windows 3.1x) or HyperTerminal (Windows 95/98) can be used if you are unable to use the IMS Terminal. Terminal can be located in *Program Manager/Accessories/Terminal* for Window 3.1x. HyperTerminal for Windows 95/98 can be found in *Programs/Accessories/HyperTerminal*. The settings (whichever terminal is used), will be: ANSI Terminal, Direct connect to COM port, BAUD Rate = 9600, Data Bits = 8, Parity = None, Stop Bits = 1, Flow Control = NONE.

TIP: The Terminal that is included with Windows 3.1x features programmable function keys which can be configured for the commands that you commonly will use (i.e. CP 1,1, IP, DVF). If you are using the upgrade version of Windows 95/98 and have upgraded from Windows 3.1x, the executable file should still be located at c:\windows\terminal.exe.

NOTE: Here is a known bug with HyperTerminal: If the horizontal scroll bar is not set all the way to the bottom left of the window, the commands issued to the LYNX/MicroLYNX may appear garbled. This is corrected by dragging the scroll bar all the way to the left.

Text Editor

A text editor is recommended for writing and editing the programs. The program then can be simply saved then uploaded as a text transfer with the *Transfer-Send Text file*. The *Terminal Setup* under *Tools Required* on the previous page illustrates the most effective screen setup for using HyperTerminal together with the Windows 95/98 text editor Notepad. Notepad is located at *Start-Programs-Accessories-Notepad* for Windows 95/98, and in the program group *Accessories* in the Windows 3.1x program manager.

Basic Components of LYNX/MicroLYNX Software

Instructions

An instruction results in an action, there are three types:

Motion

Motion instructions are those that result in the movement of a motor. The syntax of these commands are as such: first type the command followed by a space, and then the velocity or position data. For example, *MOVA 2000* will move the motor to position 2000.

I/O

An I/O instruction results in the change of parameters or the state of an Input or Output. The syntax of these commands are as such: first type the command followed by a space, then the I/O #, then an equal sign, then the data. Example: *IO 21=1* will set I/O 21 true.

Program

A program instruction allows program manipulation. The syntax of these vary due to the nature of the command. Some examples would be as such: *PGM 100*, this command toggles the system into program mode starting at address 100. *BR Loop, IO 21=1*, this command will Branch to a program labeled Loop if I/O 21 is true.

System

A system instruction is an instruction that can only be used in immediate mode to perform a system operation such as program execution (EXEC) or listing the contents of program memory (LIST). For example: EXEC 2000 will execute a program located at line 2000 of program memory space.

Variables

Variables are labeled data that allow the user to define or manipulate data. These can also be used with the built-in math functions to manipulate data. There are two classes of variables: factory defined and user defined. The syntax for each variable may differ. See *Section 4: LYNX/MicroLYNX Programming Language Reference* for usage instructions and examples.

Factory Defined Variables

These variables are predefined at the factory. They cannot be deleted. When a DVF (Delete Variables and Flags) or IP (Initialize Parameters) instruction is given, these variables will be reset to their factory default value. There are two types of factory defined variables. They are:

- Read/Writable: These factory defined variables can have their value altered by the user to effect events inside or outside of a program. For example, ACCL (Acceleration Variable) can be used to set the Acceleration, or POS (Position Variable) can be used to set a position reference point.
- Read Only: These factory defined variables cannot be manipulated by the user, but contain data that can be viewed or used to effect events inside a program. For example, VEL (velocity variable) registers the current velocity of the motor in MUNITs per second. (MUNITs will be explained later in this section.)

User Defined Variables

One of the powerful features of the LYNX/MicroLYNX is that it allows the user to define variables using the VAR (Variable) Instruction. It is important to note that when a DVF (Delete Variables and Flags) or IP (Initialize Parameters) instruction is given, these variables will be deleted! This class of variable must also be saved to memory using the SAVE instruction or when power is removed or a software reset (^C) occurs they will be lost. There are two types of user defined variables:

- Global Variables: Global variables are variables that are defined outside of a program. The benefit to using a global variable is that no user memory is required. For example, the user can define a variable called SPEED by entering VAR SPEED into the terminal. The user can then set that variable to equal the value of the read only variable VEL (velocity) by entering SPEED = VEL into the terminal.
- Local Variables: This type of user defined variable is defined within a program and can only effect events within that program. It is stored in user memory with the program. Examples of this type of variable will be given later in the section. It is worthy of note that a local variable is not static, but is erased and declared again each time a program is executed.

Flags

Flags show the status of an event or condition. A flag will only have one of two possible states: either 1=true/on/enabled or 0=false/off/disabled. As with variables, there are two classes of flags: factory and user defined.

Factory Defined Flags

Factory defined flags are predefined at the factory and cannot be deleted. When a DVF (Delete Variables and Flags) or IP (Initialize Parameters) instruction is given, these flags will be returned to their factory default state. There are two types of factory defined flags:

- Read/Writable: This type of flag is user alterable. They are typically used to set a condition or mode of operation for the LYNX/MicroLYNX. For example: RATIOE = 1 would enable ratio mode operation, or EE = 0 would disable the encoder functions.
- Read Only: Read Only flags cannot be modified by the user. They only give an indication of an event or condition. Typically this type of flag would be used in a program in conjunction with the BR (branch instruction to generate an if/then event based upon a condition. For Example: the following line of code in a program BR STOPPROG, ACL = 0 would cause a program to branch to a subroutine named “STOPPROG” when the ACL, the read only acceleration flag, is false.

User Defined Flags

This class of flag is defined by the user by using the instruction FLG. This class of flag can be either contained in a program or defined in immediate mode. There are two types of user defined flags:

- Global Flags: Global flags are flags that are defined outside of a program. The benefit to using a global flag is that no user memory is required. For example, the user can define a flag called IN_POS by entering FLG IN_POS into the terminal.
- Local Flags: This type of user defined flag is defined within a program and can only effect events within that program. It is stored in user memory with the program. It is worthy of note that a local variable is not static, but is erased and declared again each time a program is executed.

Keywords

Keywords are used in conjunction with the PRINT, GET and IP instructions to indicate or control variables and flags. For instance, PRINT UVARS would print the state of all the user-defined variables to the screen. IP FLAGS would restore all the flags to their factory default state.

Variables

MUNIT

MUNIT, or motor units, is the scaling function used to put steps into user units.
For example, here is a possible scenario: a ball screw has a 3/8 pitch = .375 inch travel per revolution using a 1.8 degree step motor being stepped by a Half/Full stepper, in half step there are 400 steps per revolution. If the user wants to operate in inches, the munit scaler would be:

(1 Rev/.375) X (400 steps/Rev) = 400/.375 = 1066.667

type MUNIT=400/.375 *then hit enter*

It is recommended that you allow the LYNX/MicroLYNX math functions to perform all the calculations for you. As in the example, were you to round the result of that calculation to 3 decimal places and enter 1066.667 as the MUNIT, it would lead to positional inaccuracy.

POS

POS indicates the position in munits.

- POS takes its reading from CTR1, which is the counter for Clock 1
- *To read the position, type* PRINT POS *or* PRINT CTR1 *then hit enter*
- *To zero the position, type* POS=0 *then hit enter*

VI

Initial velocity in munits per second.

- *To read the initial velocity, type* PRINT VI *then hit enter*
- *To write to the Initial velocity, type* VI=.25 *then hit enter*

VM

Maximum or final velocity.

- *To read the final velocity, key-in* PRINT VM *then hit enter*
- *To write to the final velocity, key-in* VM=5 *then hit enter*

ACCL

Acceleration in munits per second^2.

- *To read the acceleration, key-in* PRINT ACCL *then hit enter*
- *To write to the acceleration, key-in* ACCL=75 *then hit enter*

DECL

Deceleration in munits per second^2 .

- *To read the deceleration, key-in* PRINT DECL *then hit enter*
- *To write to the deceleration, key-in* DECL=ACCL *then hit enter*

Math Functions

Another powerful feature of the LYNX/MicroLYNX is its ability to perform common math functions and to use these to manipulate data.

Addition ..... NEW_POS*=POS+CTR3
Subtraction ..... DELTA*=CTR2-POS
Multiplication ..... ACCL=ACCL*2
Division ..... ACCL=ACCL/2
Absolute value ..... WAIT=Abs CTR3
*User-defined variable used as an example.

Motion Commands

MOVA

Move to an absolute position relative to a defined zero position.
For example, type the following commands followed by hitting enter:

```
POS=0
MOVA 200
PRINT POS
```

The terminal screen will read 200

```
MOVA 300
PRINT POS
```

The screen will echo back 300.

MOVR

Move number of steps indicated relative to current position.
For example, type the following commands followed by hitting enter:

```
POS=0
MOVR 200
PRINT POS
```

The terminal screen will read 200

```
MOVR 300
PRINT POS
```

Notice the position echoed is 500 and not 300.

SLEW

Move at a constant velocity.

```
SLEW 2000
```

The motor will move at a constant velocity 2000 munits per second.

HOLD

A HOLD 2 should typically follow any MOVA or MOVR commands in a program so that program execution is suspended until motion is complete.
(Note: There are circumstances where you may not want to hold up program execution.)
Below is a usage example.

```
PGM 1
MOVR 200
HOLD 2
END
PGM
```

I/O Commands

I/O Grouping

Group 10
Differential High speed I/O
- I/O Lines 11 – 18
  Predefined as differential Step/Direction outputs

Group 20-50
Isolated 5/24vdc I/O
Control Module
- Group 20 = I/O Lines 21 – 26
- Group 30 = I/O Lines 31 – 36

Isolated I/O Module
- Group 40 = I/O Lines 41 – 46
- Group 50 = I/O Lines 51 – 56

IOS

Sets the parameters of the I/O, this command configures the I/O.

Using the PRINT command to read IO parameters
Read all I/O parameters – "PRINT IOS"
Read I/O group 20 parameters – "PRINT IOS 20"
Read I/O 21 parameters – "PRINT IOS 21"

Setting the I/O parameters
Set group 20 I/O parameters – "IOS 20=#,#,#,#,#,#"
Set I/O 25 parameters – "IOS 25=#,#,#,#,#,#"

For example: To set I/O 25 as a Jog+ input/Low True/Level triggered the following would be entered:
IOS 25 = 16,0,0,0

IO

Used to read/write the binary state of an I/O group or read/write of an individual output. (Note: I/O must be configured as Outputs to set the state of outputs.)

Each I/O Group has 6 weighted bits:
Least Significant Bit is 1, and Most Significant Bit is 6.
Weight of the LSB=1 and MSB=32

Using the PRINT command to read the state of I/O group 20 - "PRINT IO 20"

To determine the decimal equivalent of the binary state of the whole group, you would add together the decimal weight of each set bit.
Decimal equivalent = 44, because 32 + 8 + 4 = 44

To set the state of I/O group 30 – "IO 30=39"
This will set all 6 I/O lines in group 30 to, 100111 the binary equivalent of 39
Decimal equivalent 32 + 4 +2 + 1 = 39.

To set the state of individual I/O 31 – "IO 31= 0"
The binary equivalent of group 30 is now = 38 = 100110.
Decimal equivalent = 38, because 32 + 4 + 2 = 38
To read state of individual I/O 31 - "PRINT IO 31"
A "1" or "0" will appear (1=true, 0=false)

System Instructions

The following System instructions will be used frequently.

CP

The CP Instruction is used

Program Instructions

PGM

This instruction toggles the LYNX/MicroLYNX into or out of program mode.

Switch to program mode at address 200	PGM 200
	xxxx
Program starting at address 200	xxxx
	xxxx
Switch out of program mode	PGM

LBL

Assigns a label or name to a program or subroutine.

Switch to program mode at address 200	PGM 200
Label command will name the program	LBL Program1
	xxxx
Program named by lbl command	xxxx
	xxxx
Switch out of program mode	PGM

BR

Used to branch conditionally or unconditionally to a routine.

Switch to program mode at address 200	PGM 200
Label command will name the program	LBL Program1
	xxxx
Program named by LBL command	xxxx
	xxxx
Unconditional branch to Program1	BR Program1
Switch out of program mode	PGM

END

Designates the end of a program.

Switches to program mode at address 200	PGM 200
Label command will name the program	LBL Program1
	xxxx
Program named by LBL command	xxxx
	xxxx
Unconditional branch to Program1	BR Program1
Designates the end of the program	END
Switches out of program mode	PGM

DELAY

Delays program execution in milliseconds.

Switches to program mode at address 200	PGM 200
Label command will name the program	LBL Program1
	xxxx
Program named by LBL command	xxxx
	xxxx
Delay 2 seconds between re-execution of program	DELAY 2000
Unconditional branch to program1	BR Program1
Designates the end of the program	END
Switches out of program mode	PGM

PRINT

Outputs specified text and parameter values to a terminal or terminal software on a Host PC.

Switches to program mode at address 200	PGM 200
Label command will name the program	LBL Program1
	xxxx
Program named by LBL command	xxxx
	xxxx
Prints text in quotes and then POS	PRINT “Position = ” POS
Delay 2 seconds between re-execution of program	DELAY 2000
Unconditional branch to program1	BR Program1
Designates the end of the program	END
Switches out of program mode	PGM

VAR

Command used to define a variable with 8 alphanumeric characters.

Switches to program mode at address 200	PGM 200
Define a variable named Count	VAR Count
Label command will name the program	LBL Program1
	xxxx
Program named by LBL command	xxxx
	xxxx
Prints text in quotes and then POS	PRINT “Position = ” POS
Delay 2 seconds between re-execution of program	DELAY 2000
Unconditional branch to program1	BR Program1
Designates the end of the program	END
Switches out of program mode	PGM

Program mode is the mode that the LYNX/MicroLYNX must be in to enter programs. This is done by simply typing PGM and then an address between 1 and 8000. After the program has been entered, type PGM to toggle out of program mode.

Check proper hook up of system components to the LYNX/MicroLYNX Product.

When ready to write a program, it is a good rule of thumb to Clear Program memory with the CP 1,1 command. Delete user-defined Variables and Flags with the DVF command, and Initialize Parameters with the IP command. With the LYNX/MicroLYNX Product now at factory default, there are no parameters that will throw you off track when and if you need to debug your program.

Solve I/O configuration: Configuring the I/O is done using the IOS command. The I/O can be configured as a clock input or output, a user input or output, and a dedicated I/O. The I/O can also be configured as a low true or high true.

Compute Scaling factor that scales pulses or steps into user units of degrees, rpm, inches, etc. This is using the MUNIT variable and, if an encoder is installed and enabled, the EUNIT variable also.

Using the text editor, notepad or wordpad, start writing the program. It is often easier to start with the basic motion you want. After verifying that it works, then edit the text file and add the loops and branches as needed.

There are three ways to program the LYNX/MicroLYNX Product: The first is in immediate program mode where you program as you type. This is not recommended. We recommend the use of a text editor, using the Copy and Paste functions to simply paste the program onto the IMS Terminal, or using the Send Text file function to transfer the file to the IMS Terminal.

After the final version of the program has been entered, a SAVE should be issued to save the program from Flash Memory to Non-Volatile Memory.

Program Samples

System Characteristics of Sample Programs

1) The 1.8 degree stepper motor is being driven by an IM483 in 1/256 resolution.
Therefore 1 rev. of the motor is 360/1.8=200; 200 X 256=51200 micro-steps.
The normally open dry contact switch will be between ground and the inputs.
The internal pull-up resistor to 5 VDC for the inputs has been selected by the dip switches.
Therefore when the switch is pressed the input will be grounded or low, and when not pressed it will be 5VDC or high.

2) The 1.8 degree stepper motor is being driven by an IM483 in 1/256 resolution.
One revolution of the motor gives 25 mm of deflection. The normally open dry contact switch will be between ground and the Inputs. The internal pull-up resistor to 5VDC for the inputs has been selected by the dip switches.
Therefore when the switch is pressed the input will be grounded or low, and when not pressed it will be 5VDC or high.

Sample Program 1

1A) This first program will set I/O 21 as an Input to interface a switch. When the Input is pulled low through the switch, the motor will move one revolution. The switch will, essentially, initiate the program (G0 Switch).

```
IOS 21=9,0,0,0,0,0   'Set I/O 21 to be a G0 input
PGM 1                'Enter program mode at address 1
LBL InitProg         'Name the following program InitProg
POS=0                'Set position to zero
LBL TurnOnce         'Name the following program TurnOnce
MOVR 51200           'Move relative 51200 steps
HOLD 2               'Suspend program execution until motion has stopped
END                  'Designate the end of the program
PGM                  'Exit program mode
```

1B) The second program will set I/O 21 as an Input to interface a switch. When the Input is pulled low through the switch, the motor will move one revolution. The switch will essentially initiate the program (G0 Switch). Then it will wait 3 seconds, return to zero, and wait for I/O 25 to become true before repeating the cycle. After each cycle it activates one of the 6 LED's until the sixth one is reached then it resets the LED's all off and ends the program.

```
IOS 21=9,0,0,0,0,0   'Set I/O 21 to be a G0 input
IOS 25=0,0,0,0,0,0   'Set I/O 25 to be a User Input, Low true.
PGM 1                'Enter program mode at address 1
LBL InitProg         'Name the following program InitProg
POS=0                'Set position to zero
VAR Lights=1         'Define the variable Lights set it equal to 1
IOS 30=0,1,0,0,0,0   'Set group 30 to all be User Outputs, Low True
IO 30=1              'Set IO group 30 to 1, IO 31 true, Low active
LBL TurnOnce         'Name the following program TurnOnce
IO 30=Lights         'Set IO group 30 all false, Low true, so all high
Lights=Lights*2      'Double the value of Lights( 1,2,4,8,16,32,64)
BR Done, Lights>33   'Conditional Branch to Done if Lights greater than 33
MOVR 51200           'Move relative 51200 steps
HOLD 2               'Suspend program execution until motion has stopped
DELAY 3000           'Delay three seconds
MOVA 0               'Move absolute to zero
HOLD 2               'Suspend program execution until motion has stopped
BR TurnOnce          'Unconditional Branch to TurnOnce
LBL Done             'Name the following program Done
IO 30=0              'Set IO group 30 all false
END                  'Designate the end of the program
PGM                  'Exit program mode
```

Program Sample 2

2A) This program will set I/O 21 & 25 as inputs to interface the switches. When input 21 is true it starts the program which moves the motor at a constant velocity until I/O 25 is true, then it prints its position and returns to zero.

```
IOS 21=9,0,0,0,0,0   'Set I/O 21 to be a Go input
IOS 25=0,0,0,0,0,0   'Set I/O 25 to be a user input
PGM 1                'Enter program mode at address 1
LBL ProgInit         'label the following program ProgInit
POS=0                'Set position to zero
MUNIT=51200/25       'Scale micro-steps into Millimeters
LBL TurnOnce         'label the following program TurnOnce
SLEW 500             'Move at constant velocity of 500 mm per second
LBL Loop1            'label below program Loop1
```

```
DELAY 2              'delay 2 milliseconds
BR Loop1, IO 21=0    'Conditional Branch to Loop1 if io 25 is high
SLEW 0               'Move at constant velocity of 0 mm per second
HOLD 1               'Suspend program execution until motion has stopped
PRINT POS            'Prints position
END                  'Designate the end of the program
PGM                  'Exit program mode
```

2B) This program will run upon power up “Start-up”, provided it is saved to NVM prior to power down. The program will first ask for the Speed in mm-per-second at which to slew. Once entered, it will slew at that speed until input 21 is true and then print the position where it stopped, return to zero and ask for another speed.

```
PGM 200              'Enter program mode at address 200
LBL start-up         'label the following program Start-up
POS=0                'Set position to zero
MUNIT=51200/25       'Scale micro-steps into Millimeters
IOS 21=0,0,0,0,0,0   'Set I/O 21 to be a Go input
VAR Speed            'define the variable speed
LBL MoveMe           'label the following program MoveMe
PRINT "Enter Speed:" 'Print to terminal Enter Speed:
INPUT Speed          'Allow user to enter data
SLEW Speed           'Move at constant velocity equal to speed
LBL Loop1            'label below program Loop1
DELAY 2              'delay 2 milliseconds
BR Loop1, IO 21=0    'Conditional Branch to Loop1 if io 21 is high
PRINT POS            'Prints position
MOVA 0               'Move back to zero position
HOLD 2               'Suspend program execution until motion has stopped
BR Moveme            'Unconditional Branch to MoveMe
END                  'Designate the end of the program
PGM                  'Exit program mode
```

Cut to Length Application

This program asks for several variables using the Print and Input commands. It then will start feeding the material in the cutsize increments with a delay adjustable by the encoder input on Counter 3. When the material leftover is less than the cutsize, the user has the option to modify the cutsize. When there is no material left it will exit the program.

```
PGM 1                                 'Start program mode at address 1
LBL Cutstuff                          'Name the program "Cutstuff"
MUNIT=51200/25                        'Scale steps into user units
CTR3=100                              'Set Counter 3 (Clock 3 counter) to 100
POS=0                                 'Set Position Register (Clock 1 counter) to 0
VAR Feedrate=0                        'Define the Variable "Feedrate" and set it to 0
VAR Cutsize=0                         'Define the Variable "Cutsize" and set it to 0
VAR Length=0                          'Define the Variable "Length" and set it to 0
VAR Leftover=0                        'Define the Variable "Leftover" and set it to 0
VAR Enter=0                           'Define the Variable "Enter" and set it to 0
VAR Time=100                          'Define the Variable "Time" and set it to 100
FLG Answer1=0                         'Define the Flag  "Answer1" and set it to 0
PRINT "Enter Feed Rate in inches/sec- "; 'Prompts user for Feed Rate speed
INPUT Feedrate                        'Enters the Data entered by the user into Feedrate
                                      'variable
PRINT "Enter length of raw material in inches -";      'Prompts user for Length
INPUT Length                          'Enters the Data entered by user into
                                      'Length variable
LBL Cutting                           'Name the program "Cutting"
PRINT "Enter in inches, length of Cut -";'Prompts user for length of Cut
INPUT Cutsize                         'Enters the Data entered by user into Cutsize variable
LBL Go_now                                  'Name the program "go_now"
Leftover=Length-POS                         'Set Leftover equal to material length less
                                            'already cut
BR Toosmall, Leftover<Cutsize               'Branch to "toosmall" if leftover is less than
                                            'cutsize
Time=abs CTR3                               'Set time equal to the absolute value of ctr3
DELAY Time                                  'Delay for (time * .001) seconds
VI=Feedrate/100                             'Set Initial Velocity to Feed rate divided by 100
VM=Feedrate                                 'Set Max Velocity to Feed rate entered
MOVR Cutsize                                'Move number of inches entered for Cutsize
HOLD 2                                      'Suspend the Program execution until Motion stops
BR Done, POS=Length                         'Branch to "Done" if amount cut is equal to
                                            'Length
BR Done, POS>Length                         'Branch to "Done" if amount cut is less than
                                            'Length
BR Go_now                                   'Branch to "Go_now"
LBL Toosmall                                'Name the program "Toosmall"
BR Done, Leftover=0                         'Branch to "Done" if remaining material is equal
                                            'to 0
PRINT " Remaining material is smaller or equal to cutsize!!"
PRINT "You have " leftover " inches remaining  "
PRINT  "======================================"
PRINT " Do you wish to modify cutsize: Yes=1, No=0 "
INPUT Answer1                               'Enters the Data entered by user into answer1
                                            'Flag
BR Cutting, Answer1=1                       'Branch to "Cutting" if user wants to modify
                                            'cutsize
LBL Done                                    'Name the program "Done"
PRINT " Program has ended. Remove all debris !!"
PRINT  "======================================"
PRINT "To run program again type cutstuff"
END                                         'End of Program
PGM                                         'Ends Program Mode
```

SECTION 3

Functional Grouping of the Instruction Set

Section Overview

This section covers contains a logical grouping of the LYNX/MicroLYNX product family instruction set. Each subsection contains a table summarizing a description, usage example and default setting for each instruction, variable, flag or keyword. In the case where a command can logically be placed in more than one group, it is duplicated in each group. The following functional groups are presented:

- Acceleration and Deceleration
- Position
- Encoder
- Miscellaneous Motion
- Event
- Instructions (Immediate Mode)
- Miscellaneous and Setup Flags
- Velocity
- Drive and Motor
- I/O
- Data
- Instructions (Program Mode)
- Miscellaneous and Setup Variables
- Mathematical and Logical Functions

Using the Tables

The instruction set summary tables are set up in the manner illustrated in the following example:

Example Table			
Command	**Usage Example**	**Description**	**Default**
INST	INST	This instruction causes that event.	
VAR	VAR=<num>,<mode>	Variable contains some data. <num>= Some number (range) or unit of measure. <mode> = 0: Does this. <mode> = 1: Does that.	<num> = 1024 <mode> = 0
FLAG	FLAG=<flg>	Flag enables/disables some function. <flg> = 0: Disabled. <flg> = 1: Enabled.	0

Command

The command is given in the left hand column.

Usage Example

The usage example column illustrates how the instruction, variable or flag would be used in a program or in immediate mode. In the case of the expressions bracketed by the <> symbol only the contents would be typed not the symbols themselves. For example: VAR=<num>,<mode> would be entered VAR=23, 1 *(arbitrary numbers used in example)*. The following codes are mostly self explanatory and are used to identify the various settings:

<num>	=	Some number.
<param>	=	Parameter.
<time>	=	Time.
<flg>	=	Flag, this will be 1 or 0.
<percent>	=	Percentage.
<lbl/addr>	=	Program label or address.
<mode>	=	Mode.
<chan>	=	Channel.
<func>	=	Function.
<cond>	=	Condition.
<state>	=	Logic state.

Description

The description column contains a brief description of the command and an elaboration of the expression bracketed by the <> symbols.

Factory Default

This column contains the factory default setting of the variable or flag discussed.

Acceleration and Deceleration

Acceleration and Deceleration Related Variables and Flags			
Command	**Usage Example**	**Description**	**Default**
ACCL	ACCL=<num>	Peak acceleration value. <num>= User units/sec^2.	1000000.000
ACL	BR <lbl/addr>, ACL=<flg> PRINT ACL	Read-only acceleration flag. <lbl/addr> = Program label or address. <flg>=1: Accelerating. <flg>=0: Not accelerating.	0
ACLT	ACLT=<param>	Acceleration type variable. <param>=0: User Defined. <param>=1: Linear. <param>=2: Triangle S-Curve. <param>=3: Parabolic. <param>=4: Sinusoidal S-Curve.	1
ACLTBL	ACLTBL=<num>, <val>	User-defined acceleration profile table. <num>=0 - 256 <val>=0.00-1.00	Empty
DCL	BR<lbl/addr>, DCL=<flg> PRINT DCL	Read-only deceleration flag. <lbl/addr> = Program label or address. <flg>=1: decelerating. <flg>=0: not decelerating.	0
DCLT	DCLT=<param>	Deceleration type variable. <param>=0: User Defined. <param>=1: Linear. <param>=2: Triangle S-Curve. <param>=3: Parabolic. <param>=4: Sinusoidal S-Curve.	1
DECL	DECL=<num>	Peak Deceleration Value <num>= User units/sec^2.	1000000.000
LDCLT	LDCLT=<param>	Specifies deceleration type used when a limit is reached. <param>=0: User Defined. <param>=1: Linear. <param>=2: Triangle S-Curve. <param>=3: Parabolic. <param>=4: Sinusoidal S-Curve.	1
LDECL	LDECL=<num>	Peak deceleration value when stopping due to a limit. <num>= user units/sec^2.	1000000.000

Velocity

Velocity Related Variables, Flags and Instructions			
Command	**Usage Example**	**Description**	**Default**
JOGS	JOGS=<num>	Jog speed variable. <num>= user units/sec.	256000.000
PMV	PMV=<num>	Position maintenance velocity variable. <num>= user units/sec.	10240.000
SLEW	SLEW <num>=<mode>	Slew the motor at a constant velocity instruction. <vel> = user units/sec. <mode> = 0: Use acceleration ramp. <mode> = 1: Do not use acceleration ramp.	Mode 0 used if <mode> not specified.
SSTP	SSTP<mode>	Stop the current motion using the specified deceleration profile and optionally stop the program. <mode> = 0: Stop motion only. <mode> = 1: Stop motion and program.	Mode 0 used if <mode> not specified.
VAE	VAE=<flg>	Velocity to Analog Enable <flg> = 1 Enabled <flg> = 0 Disabled	0
VCHG	BR<lbl/addr>, VCHG=<flg> PRINT VCHG	Read-only flag indicates when velocity is changing. <lbl/addr> = Label or address of program. <flg> = 0: Velocity is not changing. <flg> = 1: Velocity is changing	0
VEL	BR<lbl/addr>, VEL=<num> PRINT VEL	Register that contains the actual velocity of the axis. In user units per second. Read-only. <lbl/addr> = Label or address of program. <num> = user unit/sec.	0.000
VI	VI=<num>	Initial velocity of the axis during a point-to-point motion. <num> = user units/sec	102400.00
VM	VM=<num>	Maximum velocity reached by the axis during a point-to-point motion. <num> = user units/sec.	768000.000

Position

Position Related Variables, Flags and Instructions			
Command	**Usage Example**	**Description**	**Default**
CMOVR	CMOVR <0-127> (linear) CMOVR <0, 1> (S-Curve)	Call Saved Relative Move	Empty
FIOS	FIOS<num1>,<num2>,<line>	Find I/O switch instruction. Parameters are optional. <num1> = ± speed in user units/sec. <num2> = ± creep in user units/sec. <line> = I/O line.	If not specified: <num> = VM <param> = VI
MOVA	MOVA <num>, <mode>	Perform point-to-point move or index to an absolute position instruction. Use of <mode> is optional. <num> = Absolute position. <mode> = 0: Motion ceases when position is reached. <mode> = 1: Motion part of a profile, does not decelerate.	Mode 0 is used when mode not specified.
MOVR	MOVR <num>, <mode>	Perform point-to-point move or index to a relative position instruction. <num> = Absolute position. <mode> = 0: Motion ceases when position is reached. <mode> = 1: Motion part of a profile, does not decelerate.	Mode 0 is used when mode not specified.

Position Related Variables, Flags and Instructions cont'd			
Command	**Usage Example**	**Description**	**Default**
MVG	BR <lbl/addr>, MVG=<flg> PRINT MVG	Read-only flag indicates when motion is taking place. <lbl/addr> = Program label or address. <flg> = 0: Axis is stationary. <flg> = 1: Axis is in motion.	0
PCHG	BR <lbl/addr>, PCHG=<flg> PRINT PCHG	Read-only flag indicates when the axis is trying to reach a specified relative or absolute position. <lbl/addr> = Program label or address. <flg> = 0: Axis is moving in a "jog" or "slew". <flg> = 1: Axis isindexing to a position.	0
POS	POS=<num> PRINT POS	Register which contains the axis position in user units. <num> = ± Position	0.000
POSCAP	PRINT POSCAP	Read-only variable: position at time of TRIP.	0.000

Drive and Motor

Drive and Motor Related Variables, Flags and Instructions			
Command	**Usage Example**	**Description**	**Default**
DRVEN	DRVEN=<flg>	Drive enable flag: enables/disables drive output. <flg> = 0: Drive output disabled. <flg> = 1: Drive output enabled.	1
DRVRS	DRVRS=<flg>	Drive reset flag: resets drive output. <flg> = 0: Drive not reset. <flg> = 1: resets the drive to phase B on fullstep.	0
DRVTP	PRINT DRVTP	Read-only drive type variable. Provides a means to interrogate system to determine the type of drive. Response = 2: IM483H Response = 4: IM805H	
HCDT	HCDT=<time>	Holding current delay time variable. <time> = Time in milliseconds.	0
MAC	MAC=<percent>	Motor acceleration current variable. Used when velocity is changing. <percent> = 0 - 100	25
MHC	MHC=<percent>	Motor holding current variable. Used when axis is stationary. <percent> = 0 - 100	5
MRC	MRC=<percent>	Motor run current variable. Used when axis is at max velocity. <percent> = 0 - 100	25
MSEL	MSEL=<param>	Microstep resolution variable. Valid <param> settings are: 2, 4, 8, 16, 32, 64, 128, 256, 5, 10, 25, 50,125, 250.	256
MSDT	MSDT=<time>	Motor settling delay time variable. <time> = 0 - 65,535 milliseconds.	0
MUNIT	MUNIT=<num>	Motor units variable specifies the number of clock pulses per user unit. <num> = Clock pulses/user unit	1.000
PMHCC	PMHCC=<percent>	Variable specifes the position manitenance hold current change. <percent> = 0 to MHC	0

Encoder

Encoder Related Variables, Flags and Instructions			
Command	**Usage Example**	**Description**	**Default**
EDB	EDB=<num>	Encoder deadband variable specifies the ± length of the deadband for position maintenance. <num> = user units.	2.000
EE	EE=<flg>	Master enable flag for all encoder functions. <flg> = 0: Encoder functions disabled. <flg> = 1: Encoder functions enabled.	0
EUNIT	EUNIT=<num>	Conversion variable for converting motor steps or user units to encoder counts. <num> = encoder counts per user unit.	1.000
PMHCC	PMHCC=<percent>	Position Maintenance Holding Current Change Variable. (Range = 0 to 100)	0
PME	PME=<flg>	Position maintenance enable flag. <flg> = 0: Position maintenance disabled. <flg> = 1: Position maintenance enabled.	0
PMV	PMV=<num>	Position maintenance velocity variable. <num> = user units/sec.	10240.000
STALL	BR<lbl/addr>, STALL=<flg> PRINT STALL	Flag which indicates if the motor has stalled. <lbl/addr> = Program label or address. <flg> = 0: Axis not stalled. <flg> = 1: Axis stalled.	0
STLDE	STLDE=<flg>	Flag enables stall detection/ <flg> = 0: Stall detection disabled. <flg> = 1: Stall detection enabled.	0
STLDM	STLDM=<mode>	Stall detect mode setting determins whether motor stops when a stall is detected. <mode> = 0: Stop motor. <mode> = 1: Do not stop motor.	0
STLF	STLF=<num>	Stall factor variable. <num> = User units.	10

I/O

I/0 Related Variables, Flags and Instructions			
Command	**Usage Example**	**Description**	**Default**
ADS	ADS<chan>=<aunit>,<func>,<law>	Setup variable for the Analog Input/Joystick module. <chan> = Channel # (1 or 2). <aunit> = User Unit = MUNIT * AUNIT. <func> = 1: Analog input. <func> = 2: Joystick interface. <law> = 1: Linear. <law> = 2: Square law. <law> = 3: Cube law. <law> adjusts the joystick position to motor velocity transformation.	1, 1, 1
AIN	<var>=AIN<chan>	Variable causes a read of analog input channel. <var> = Variable to which data is saved. <chan> = Analog input channel.	
AOUT	AOUT=<num>	Output Analog to Channel. <num> = 1 or 2, Analog Module is in Slot 2. <num> = 3 or 4, Analog Module is in Slot 3.	

I/0 Related Variables , Flags and Instructions cont'd			
Command	**Usage Example**	**Description**	**Default**
IJSC	IJSC	Instruction supports the Analog Input/Joystick Interface in joystick mode.	
IO	IO<line>=<lstate> IO<group>=<bstate>	Variable reads or writes the state of an I/O <line> or <group>. <lstate> = 0: I/O line inactive. <lstate> = 1: I/O line active. <bstate> = (0-63): Binary state of all lines in group.	
IOF	IOF<group>=<param>	Variable sets the level of digital filtering to be applied to a specified I/O Group. <group> = 10 - 50 <param> = 1 - 7	Group 10 = 0 Groups 20-50 = 7
IOS	See Language Reference for usage example.	Variable configures the I/O, also used as a keyword with the IP instruction.	
JSC	JSC=<num>	Joystick center position variable. Automatically updated by IJSC or manually using: <num> = 0 - 4095 (AUNIT = 1)	2048
JSDB	JSDB=<num>	Joystick deaband variable. Automatically updated by IJSC or manually using: <num> = 0 - 4095 (AUNIT = 1)	10
JSE	JSE=<flg>	Joystick Enable/Disable Flag. Enables velocity mode for the Analog Input/Joystick Module. <flg> = 0: Disabled <flg> = 1: Enabled	0
JSFS	JSFS=<num>	Joystick full scale variable. Automatically updated by IJSC or manually using: <num> = 0 - 4095 (AUNIT = 1)	2038
JSHCE	JSHCE=<flg>	Enables Hold Current when JoyStick and Jog are at zero velocity, else Run Current. <flg> = 0: Always Run Current even when Velocity = 0. <flg> = 1: Switch to Hold Current when Velocity = 0.	1
LIMSTP	LIMSTP=<flg>	Flag specifies whether or not motion will cease when a limit is reached. <flg> = 0: Motion will not stop. <flg> = 1: Motion will stop.	1

Miscellaneous Motion

Miscellaneous Motion Related Variables, Flags and Instructions			
Command	**Usage Example**	**Description**	**Default**
BLE	BLE=<flg>	Flag enables/disables backlash compensation. <flg> = 0: Disabled. <flg> = 1: Enabled.	0
BLM	BLM=<mode>	Variable specifies the mode for backlash compensation. <mode> = 0: Mathematical Compensation. <mode> = 1: Mechanical Compensation.	Mode 0
BLSH	BLSH=<num>	Backlash compensation amount. <num> = User Units.	0.000
CTR1		Counter which represents the raw counts sent to the primary motor.	0.000
CTR2		Counter which represents the raw counts received from the encoder.	0.000
CTR3		Counter which represents the raw counts of the clock seen on I/O 15 and 16.	0.000
FIOS	FIOS<num1>,<num2>,<line>	Find I/O switch instruction. Parameters are optional. <num1> = ± speed in user units/sec. <num2> = ± creep in user units/sec. <line> = I/O line.	If not specified: <num> = VM <param> = VI
HAE	HAE=<flg>	Flag which enables/disables half axis scaling mode. <flg> = 0: Disabled. <flg> = 1: Enabled.	0

Miscellaneous Motion Related Variables, Flags and Instructions cont'd			
Command	**Usage Example**	**Description**	**Default**
HAS	HAS=<param>	Variable defines the scaling factor for half axis mode. <param> = -1 to <1	0.000
RATIO	RATIO=<param>	Variable sets the ratio for a secondary drive to a specified value. <param> = -2 to <2	1.000
RATIOE	RATIOE=<flg>	Master flag enables/disables ratio functions. <flg> = 0: Disabled. <flg> = 1: Enabled.	0
RATIOW	RATIOW=<num>	Pulse width for the secondary channel(s) being used to drive the motor(s) in ratio mode. <num> = 0 - 254 (1 - 254 = 50ns increments)	0
STEPW	STEPW=<num>	Pulse width for the step clock of the primary axis. <num> = 0 - 254 (1 - 254 = 50ns increments)	0

Data

Data Related Instructions, Keywords and Flags			
Command	**Usage Example**	**Description**	**Default**
ALL	PRINT ALL IP ALL GET ALL	Keyword used with the GET, PRINT, and IP instructions to indicate the inclusion of all variables, flags and program space where applicable.	
CPL	CPL <var/flg>	Instruction that performs the two's complement of the specified variable or flag. <var/flg> = Variable or flag.	
DEC	DEC <var>	Instruction used to decrement the specified variable by 1. <var> = Variable.	
DVF	DVF <param1>,<param2>	Instruction that deletes user-defined variables and flags. <param1> = 0: All user vars and flags deleted. <param1> = 1: Only user vars deleted. <param1> = 2: Only user flags deleted. <param2> = 0: All global and local user vars and/or flags deleted. <param2> = 1: Only global user vars and/or flags deleted. <param2> = 2: Only local user vars and/or flags deleted.	If no parameter is specified, both will be 0
ERR	PRINT ERR	Read-only status flag indicates whether an error has occurred.	
ERRA	PRINT ERRA	Read-only variable displays the memory location that the error occurred.	
ERROR	PRINT ERROR	Read-only variable contains the error code for the most recent error. See Appendix B for a list of possible errors.	
FAULT	FAULT=<flg>	Flag allows the user to enable/disable the fault indicator LED. <flg> = 0: Disabled. <flg> = 1: Enabled.	1
FLAGS	PRINT FLAGS IP FLAGS GET FLAGS	Keyword used with the GET, PRINT, and IP instructions to indicate the inclusion of only flags.	

Data Related Instructions, Keywords and Flags cont'd			
Command	**Usage Example**	**Description**	**Default**
FLG	FLG <name>=<state>	Instruction to define a user flag that can be TRUE or FALSE. <name> = Identifier for flag, up to 8 characters. <state> = Logic state, 1 or 0.	
GET	GET <param>	Instruction that retrieves the specified information from non-volatile memory (NVM). <param> = ALL: All vars, flags, and program space. <param> = VARS: Variables only. <param> = FLAGS: Flags only. <param> = PGM: Program space. <param> = IOS: I/O settings.	If <param> is not specified then <param> = ALL
INC	INC <var>	Instruction increments the specified variable by 1.	
INPUT	INPUT <var>, <param>	Instruction used to request input from the user. <param> is an optional nowait parameter. <var> = Variable. <param> = 0: Suspend prog. execution. <param> = 1: Do not suspend prog. execution.	If <param> is not specified then <param> = 0
INPUT1	INPUT1 <var>, <param>	Enhancement to the INPUT instruction which will only accept input from LYNX/MicroLYNX COMM 1. <param> is an optional nowait parameter. <var> = Variable. <param> = 0: Suspend prog. execution. <param> = 1: Do not suspend prog. execution.	If <param> is not specified then <param> = 0
INPUT2	INPUT2 <var>, <param>	Enhancement to the INPUT instruction which will only accept input from LYNX/MicroLYNX COMM 2. <param> is an optional nowait parameter. <var> = Variable. <param> = 0: Suspend prog. execution. <param> = 1: Do not suspend prog. execution.	If <param> is not specified then <param> = 0
IP	IP <param>	Initializes specified parameters to the factory default state. <param> = ALL: All vars, flags and I/O settings. <param> = VARS: Variables only. <param> = FLAGS: Flags only. <param> = IOS: I/O settings.	If <param> is not specified then <param> = ALL
PGM	GET PGM	Keyword used with the GET instruction to retrieve the contents of program space from NVM.	
PGM	PGM	Instruction used to place the LYNX/MicroLYNX in program mode.	
PRINT	PRINT <text/param>	Instruction used to output text and parameter value(s) to the host PC. See Language Reference for usage details.	
PRINT1	PRINT1 <text/param>	Enhancement to the PRINT instruction that will allow the user to only output to LYNX/MicroLYNX COMM 1.	
PRINT2	PRINT2 <text/param>	Enhancement to the PRINT instruction that will allow the user to only output to LYNX/MicroLYNX COMM 2.	
SAVE	SAVE	Saves all variables, flags and programs currently in working memory to NVM.	
SER	PRINT SER	Read-only variable that contains the serial number of the LYNX product.	
SET	SET <var>=<num> SET <flg>=<state>	Instruction used to set a variable or a flag to a specified value. NOTE: The SET instruction does not need to be entered to take effect. When entering <var>=<num> or <flg>=<state> SET is assumed.	

Data Related Instructions, Keywords and Flags cont'd			
Command	**Usage Example**	**Description**	**Default**
STATS	PRINT STATS	Keyword used with the PRINT instruction to output the values of only status flags.	
UFLGS	PRINT UFLGS	Keyword used with the PRINT instruction to output the values of only user defined flags.	
ULBLS	PRINT ULBLS	Keyword used with the PRINT instruction to output the values of only user defined labels.	
UVARS	PRINT UVARS	Keyword used with the PRINT instruction to output the values of only user defined variables.	
VAR	VAR = <name>	Instruction used to define a user variable to hold numeric data. <name> = 1 to 8 alpha-numeric characters.	
VARS	PRINT VARS GET VARS IP VARS	Keyword used with the GET, PRINT and IP instructions to indicate the inclusion of only variables.	
VER	PRINT VER	Read-only variable that contains the version information of the LYNX product.	

Event (Trip)

Event (Trip) Related Instructions, Keywords and Flags cont'd			
Command	**Usage Example**	**Description**	**Default**
TI1, TI2, TI3, TI4	TI<x>=<input>,<lbl/addr>,<output>	Trip on input variables which setup an input event (trip) for the specified input. This variable was formerly IT<x>. <x> = 1 - 4 <input> = Input used for trip. <lbl/addr> = Subroutine label or address to be activated on trip. <output> = I/O # to be set true on input trip.	0, 0, 0
TIE1, TIE2, TIE3, TIE4	TIE<x>=<flg>	Flag enables/disables the corresponding event trip. Flag was formely ITE<x>. <x> = 1 - 4 <flg> = 0: Disabled <flg> = 1: Enabled	0
TP1, TP2, TP3, TP4	TP<x>=<pos>,<lbl/addr>,<output>	Trip on position variables which setup an input event (trip) for the specified input. <x> = 1 - 4 <pos> = Position used for trip. <lbl/addr> = Subroutine label or address to be activated on trip. <output> = I/O # to be set true on input trip.	0.000, 0, 0
TPE1, TPE2, TPE3, TPE4	TPE<x>=<flg>	Flag enables/disables the corresponding event trip. <x> = 1 - 4 <flg> = 0: Disabled <flg> = 1: Enabled	0

Event (Trip) Related Instructions, Keywords and Flags cont'd			
Command	**Usage Example**	**Description**	**Default**
TT1, TT2, TT3, TT4	TTx><time>,<lbl/addr>,<output>	Timer trip variables which setup an input event (trip) for the specified input. This variable was formerly TI<x>. <x> = 1 - 4 <time> = Time used for trip. <lbl/addr> = Subroutine label or address to be activated on trip. <output> = I/O # to be set true on input trip.	0, 0, 0
TTE1, TTE2, TTE3, TTE4	TTE<x>=<flg>	Trip on timer enable, enables the corresponding timer event trip (TT). Formerly TIE<x>. <x> = 1 - 4 <flg> = 0: Disabled <flg> = 1: Enabled	0
TTR1, TTR2, TTR3, TTR4	TTR<x>=<flg>	Trip on timer repeat, specifies whether or not the corresponding timer event trip (TT) will repeat each time the specified period expires. Formerly TIR<x> <x> = 1 - 4 <flg> = 0: Do not repeat. <flg> = 1: Repeat.	0
TV	TV=<velocity>,<lbl/addr>,<output>	Trip on velocity. This variable was formerly VT. <velocity> = Velocity used for trip. <lbl/addr> = Subroutine label or address to be activated on trip. <output> = I/O # to be set true on velocity trip.	0.000, 0, 0
TVE	TVE=<flg>	Trip on velocity enable. Enables the corresponding velocity trip. Formerly VTE. <flg> = 0: Disabled <flg> = 1: Enabled	0

Instructions That Can Be Used in a LYNX/MicroLYNX Program

Instructions That Can Be Contained in a LYNX Program			
Command	**Usage Example**	**Description**	**Default**
BR	BR <lbl/addr>, <param>	Performs conditional or unconditional branch to a routine in a LYNX program. <lbl/addr> = Subroutine label or address. <param> = Condition parameter: sets the condition for the branch. If blank, branch will be unconditional.	
CALL	CALL <lbl/addr>, <param>	Allows the user to invoke a subroutine within a LYNX program. <lbl/addr> = Subroutine label or address to be invoked if <param> = TRUE. <param> = Condition parameter: Flags or logical functions to be evaluated.	
CPL	CPL <var/flg>	Instruction that performs the two's complement of the specified variable or flag. <var/flg> = Variable or flag.	
DEC	DEC <var>	Instruction used to decrement the specified variable by 1. <var> = Variable.	
DELAY	DELAY=<time>	Delay program execution for specified <time>. <time> = Time in milliseconds.	

Instructions That Can Be Contained in a LYNX Program cont'd			
Command	**Usage Example**	**Description**	**Default**
DVF	DVF <param1>,<param2>	Instruction that deletes user defined variables and flags. <param1> = 0: All user vars and flags deleted. <param1> = 1: Only user vars deleted. <param1> = 2: Only user flags deleted. <param2> = 0: All global and local user vars and/or flags deleted. <param2> = 1: Only global user vars and/or flags deleted. <param2> = 2: Only local user vars and/or flags deleted.	If no parameter is specified, both will be 0
END	END	Stops the execution of a LYNX program.	
FLG	FLG <name>=<state>	Instruction to define a user flag that can be TRUE or FALSE. <name> = Identifier for flag, up to 8 characters <state> = Logic state, 1 or 0	
GET	GET <param>	Instruction that retrieves the specified information from non-volatile memory (NVM). <param> = ALL: All vars, flags and program space. <param> = VARS: Variables only. <param> = FLAGS: Flags only. <param> = PGM: Program space. <param> = IOS: I/O settings.	If <param> is not specified then <param> = ALL
HOLD	HOLD <mode>	Hold program execution until the specified motion phase completes. <mode> = 0: Program suspended until position change completes. <mode> = 1: Program suspended until velocity change completes. <mode> = 2: Program suspended until motion ceases.	0
INC	INC <var>	Instruction increments the specified variable by 1.	
INP	INP=<flg>	Input Pending Status Flag	0
INPUT	INPUT <var>, <param>	Instruction used to request input from the user. <param> is an optional nowait parameter. <var> = Variable. <param> = 0: Suspend prog. execution. <param> = 1: Do not suspend prog. execution.	If <param> is not specified then <param> = 0
INPUT1	INPUT1 <var>, <param>	Enhancement to the INPUT instruction which will only accept input from LYNX/MicroLYNX COMM 1. <param> is an optional nowait parameter. <var> = Variable. <param> = 0: Suspend prog. execution. <param> = 1: Do not suspend prog. execution.	If <param> is not specified then <param> = 0
INPUT2	INPUT2 <var>, <param>	Enhancement to the INPUT instruction which will only accept input from LYNX/MicroLYNX COMM 2. <param> is an optional nowait parameter. <var> = Variable. <param> = 0: Suspend prog. execution. <param> = 1: Do not suspend prog. execution.	If <param> is not specified then <param> = 0
IP	IP <param>	Initializes specified parameters to the values stored in Non-Volitile Memory.. <param> = ALL: All vars, flags and I/O settings. <param> = VARS: Variables only. <param> = FLAGS: Flags only. <param> = IOS: I/O settings.	If <param> is not specified then <param> = ALL

Instructions That Can Be Contained in a LYNX Program cont'd			
Command	**Usage Example**	**Description**	**Default**
LBL	LBL=<name>	This instruction will label the address of a program or a subroutine within a program. <name> = 1 to 8 alphanumeric charaters including "_" underscore.	
MOVA	MOVA <num>, <mode>	Perform point-to-point move or index to an absolute position instruction. Use of <mode> is optional. <num> = Absolute position. <mode> = 0: Motion ceases when position is reached. <mode> = 1: Motion part of a profile, does not decelerate.	Mode 0 is used when mode not specified
MOVR	MOVR <num>, <mode>	Perform point-to-point move or index to a relative position instruction. <num> = Absolute position. <mode> = 0: Motion ceases when position is reached. <mode> = 1: Motion part of a profile, does not decelerate.	Mode 0 is used when mode not specified
NOP	NOP	No operation instruction, used to fill one byte of program space.	
ONER	ONER <lbl/addr>	On error, go to the specified label or address <lbl/addr>.	
PRINT	PRINT <text/param>	Instruction used to output text and parameter value(s) to the host PC. See Language Reference for usage details.	
PRINT1	PRINT1 <text/param>	Enhancement to the PRINT instruction that will allow the user to only output to LYNX/MicroLYNX COMM 1.	
PRINT2	PRINT2 <text/param>	Enhancement to the PRINT instruction that will allow the user to only output to LYNX/MicroLYNX COMM 2.	
RET	RET	Return statement RET is required at the end of a subroutine invoked by a CALL instruction.	
RUN	RUN <lbl/addr>	The RUN instruction executes a background taskto be run at the specified label or address <lbl/addr>.	
SAVE	SAVE	Saves all variables, flags and programs currently in working memory to NVM.	
SET	SET <var>=<num> SET <flg>=<state>	Instruction used to set a variable or a flag to a specified value. NOTE: The SET instruction does not need to be entered to take effect. When entering <var>=<num> or <flg>=<state> SET is assumed.	
SLEW	SLEW <num>=<mode>	Slew the motor at a constant velocity instruction. <vel> = User units/sec. <mode> = 0: Use acceleration ramp. <mode> = 1: Do not use acceleration ramp.	Mode 0 used if <mode> not specified
SSTP	SSTP<mode>	Stop the current motion using the specified deceleration profile and optionally stop the program. <mode> = 0: Stop motion only. <mode> = 1: Stop motion and program.	Mode 0 used if <mode> not specified
VARS	PRINT VARS GET VARS IP VARS	Keyword used with the GET, PRINT and IP instructions to indicate the inclusion of only variables.	

Instructions That Can Be Used in Immediate Mode

Instructions That Can Be Issued in Immediate Mode			
Command	**Usage Example**	**Description**	**Default**
CP	CP <lbl/address>, <flg>	Clear program instruction clears the program space in working memory beginning with the label or address specified by <lbl/addr>. <flg> = 0: Clear specified program only. <flg> = 1: Clear entire program space beginning with secified <lbl/addr>.	If <flg> not specified <flg>=0
CPL	CPL <var/flg>	Instruction that performs the two's complement of the specified variable or flag. <var/flg> = Variable or flag.	
DEC	DEC <var>	Instruction used to decrement the specified variable by 1. <var> = Variable.	
DVF	DVF <param1>,<param2>	Instruction that deletes user defined variables and flags. <param1> = 0: All user vars and flags deleted. <param1> = 1: Only user vars deleted. <param1> = 2: Only user flags deleted. <param2> = 0: All global and local user vars and/or flags deleted. <param2> = 1: Only global user vars and/or flags deleted. <param2> = 2: Only local user vars and/or flags deleted.	If no parameter is specified, both will be 0
END	END	Stops the execution of a LYNX program.	
EXEC	EXEC <lbl/addr>, <mode>	Execute the program label or located at address specified by <lbl/addr>. <mode> = 0: Normal execution. <mode> = 1: Trace mode. <mode> = 2: Single step mode.	If <mode> not specified <mode>=0
FIOS	FIOS<num1>,<num2>,<line>	Find I/O switch instruction. Parameters are optional. <num1> = ± Speed in user units/sec. <num2> = ± Creep in user units/sec. <line> = I/O line.	If not specified: <num> = VM <param> = VI
FLG	FLG <name>=<state>	Instruction to define a user flag that can be TRUE or FALSE. <name> = Identifier for flag, up to 8 characters. <state> = Logic state, 1 or 0.	
GET	GET <param>	Instruction that retrieves the specified information from non-volatile memory (NVM). <param> = ALL: All vars, flags and program space. <param> = VARS: Variables only. <param> = FLAGS: Flags only. <param> = PGM: Program space. <param> = IOS: I/O settings.	If <param> is not specified then <param> = ALL
INC	INC <var>	Instruction increments the specified variable by 1.	
IP	IP <param>	Initializes specified parameters to the factory default state. <param> = ALL: All vars, flags, and I/O settings. <param> = VARS: Variables only. <param> = FLAGS: Flags only. <param> = IOS: I/O settings.	If <param> is not specified then <param> = ALL

Instructions That Can Be Issued in Immediate Mode cont'd			
Command	**Usage Example**	**Description**	**Default**
LIST	LIST <lbl/addr>, <flg>	List stored program space beginning with label or address specified by <lbl/addr>. <flg> = 0: List through end of program space. <flg> = 1: List through first END.	If <lbl/addr> not specified will list all program space, If <flg> not specified <flg> = 0
MOVA	MOVA <num>, <mode>	Perform point-to-point move or index to an absolute position instruction. Use of <mode> is optional. <num> = Absolute position. <mode> = 0: Motion ceases when position is reached. <mode> = 1: Motion part of a profile, does not decelerate.	Mode 0 is used when mode not specified.
MOVR	MOVR <num>, <mode>	Perform point-to-point move or index to a relative position instruction. <num> = Absolute position. <mode> = 0: Motion ceases when position is reached. <mode> = 1: Motion part of a profile, does not decelerate.	Mode 0 is used when mode not specified
ONER	ONER <lbl/addr>	On error, go to the specified label or address <lbl/addr>.	
PAUS	PAUS	Suspends the executing program as well as any motion that is in progress.	
RES	RES	Resumes program and motion suspended by the PAUS instruction.	
SAVE	SAVE	Saves all variables, flags and programs currently in working memory to NVM.	
SET	SET <var>=<num> SET <flg>=<state>	Instruction used to set a variable or a flag to a specified value. NOTE: The SET instruction does not need to be entered to take effect. When entering <var>=<num> or <flg>=<state> SET is assumed.	
SLEW	SLEW <num>=<mode>	Slew the motor at a constant velocity instruction. <vel> = User units/sec. <mode> = 0: Use acceleration ramp. <mode> = 1: Do not use acceleration ramp.	Mode 0 used if <mode> not specified.
SSTP	SSTP<mode>	Stop the current motion using the specified deceleration profile and optionally stop the program. <mode> = 0: Stop motion only. <mode> = 1: Stop motion and program.	Mode 0 used if <mode> not specified
VARS	PRINT VARS GET VARS IP VARS	Keyword used with the GET, PRINT and IP instructions to indicate the inclusion of only variables.	
<esc>	Escape Key	Terminates all active operations and all running programs.	
^C	CTRL+C Keys	Terminates all active operations and all running programs, forces a partial reset of the LYNX or MicroLYNX.	

Miscellaneous and Setup Variables

Miscellaneous and Setup Variables			
Command	**Usage Example**	**Description**	**Default**
BAUD	BAUD = <param>	Sets the BAUD rate for serial communications with the LYNX/MicroLYNX. <param> = 48: 4800 bps <param> = 96: 9600 bps <param> = 19: 19,200 bps <param> = 38: 38,000 bps	9600 bps
BKGDA	BKGDA = <num>	Variable holds the present instruction address for the background task. <num> = 7 - 8175	
BREAK	BREAK =<num>, <lbl/addr>	Variable allows user to set up to 10 break points within a LYNX program. <num> = 0: Function disabled. <num> = 1-10: Program addresses specified by <lbl/addr> where execution will break.	
COMDT	COMDT=<var>=	Communication Delay Time. Resets comm if time completes <var> = 0, disabled <var> = 1, 1 to 65,000 sec	0
DISP	DISP = <lines>, <chars>, <wrap>	Specifies the display format for the PRINT instruction. <lines> = Number of lines (0-255, 0=no limit). <chars> = Number of characters (0-255, 0=no limit). <wrap> = 0: Do not wrap line. <wrap> = 1: Wrap long lines to next line.	0, 0, 0
DN	DN = <char>	Variable stores the device name to be used when in PARTY mode of operation. <char> = A - Z, a - z, 0 - 9	!
ECHO	ECHO = <mode>	Specifies whether or not the LYNX or MicroLYNX will echo commands received via communications port back over the line. <mode> = 0: Full duplex. <mode> = 1: Half duplex. <mode> = 2: Only respond to PRINT and LIST commands.	0
ESC	ESC=<flg>	Selects whether ESC or CTRL^E is used to stop Motion and User Program.	1
PAUSM	PAUSM=<mode>	Determines how motion is stopped in response to the PAUS instruction and whether or not it is restarted in response to the RES instruction. <mode> = 0: Normal DECL, resume with RES. <mode> = 1: LDECL deceleration, resume with RES. <mode> = 2: Complete motion, normal DECL. <mode> = 3: Complete motion, LDECL deceleration. <mode> = 4: Normal DECL, no resume. <mode> = 5: LDECL deceleration, no resume	0
PFMT	PFMT=<num1>, <num2>, <param>	Specifies the print format for numeric values. <num1> = # of digits before decimal (0-16). <num2> = # of digits after decimal (0-16). <param> = 0: Spaces as placeholders. <param> = 1: Zeros as placeholders. <param> = 2: No padding.	10, 3, 2
PRMPT	PRMPT=<char>	Specifies the character to be used by the LYNX or MicroLYNX as a prompt character. <char> = Character or ASCII decimal value (32-254).	> (ASCII 62)
LOCK	LOCK=<flg>	Lock User Program so that it cannot be viewed with LIST - cleared by CP 1,1 or RFTD.	0

Miscellaneous and Setup Flags

Miscellaneous and Setup Flags			
Command	**Usage Example**	**Description**	**Default**
BIO	BIO=<flg>	Sets communication mode. <flg> = 0: ASCII. <flg> = 1: Binary.	0
BKGD	BR<lbl/addr>, BKGD=<flg> PRINT BKGD	Read only status flag indicates whether or not a background program is running. <flg> = 0: Background program not running. <flg> = 1: Background program running.	0
BSY	PRINT BSY	Read only status flag indicates whether or not a program is running. Response = 0: Program not running. Response = 1: Program running.	0
CSE	CSE=<flg>	Enables/disables use of checksum when binary communications are used. <flg> = 0: Disabled. <flg> = 1: Enabled.	0
GECHE	GECHE=<flg>	Enables/disables the echo of global commands for use in party mode. <flg> = 0: Disabled. <flg> = 1: Enabled.	0
HELD	PRINT HELD	Read only status flag indicates whether or not a program is waiting for a position change, velocity change or motion to complete. Response = 0: Program not held. Response = 1: Program held.	0
HOST	HOST=<flg>	Enables/disables the status of a LYNX or MicroLYNX as the host module in a multi-drop system. <flg> = 0: Disabled (not host). <flg> = 1: Enabled (host).	0
LOGO	LOGO =<flg>	Enables/disables the sign-on banner. <flg> = 0: Disabled. <flg> = 1: Enabled.	1
PARTY	PARTY=<flg>	Enables/disables party mode. <flg> = 0: Disabled. <flg> = 1: Enabled.	0
PAUSD	BR <lbl/addr>, PAUSD = <flg> PRINT PAUSD	Read only status flag indicates whether or not a program has been paused. <flg> = 0: Program not paused. <flg> = 1: Program paused.	0
QUED	QUED=<flg>	Enables/disables the ability of LYNX/MicroLYNX modules in a multi-drop system to receive broadcast commands. <flg> = 0: Disabled. <flg> = 1: Enabled.	0
STK	BR <lbl/addr>, STK = <flg> PRINT STK	Read only status flag indicates a program subroutine stack fault. <flg> = 0: No fault. <flg> = 1: Stack overflow or underflow.	

Mathematical and Logical Functions

All mathematical and logical functions are evaluated sequentially, there is no hierarchy of functions. Therefore, the equation 1 + 2 * 3 evaluates to 9, and not 7. All functions can be evaluated in immediate mode, although their real usefulness is in a control module program.

Mathematical and Logical Functions			
Function	**Symbol**	**Example**	**Binary Mode Opcode Hex (Decimal)**
Addition	+	VAR3 = VAR1 + VAR2	10h (16)
Subtraction	-	VAR3 = VAR1 - VAR2	11h (17)
Multiplication	*	VAR3 = VAR1 * VAR2	12h (18)
Division	/	VAR3 = VAR2 / VAR1	13h (19)
AND (bitwise)	&	VAR3 = VAR1 & VAR2 (bitwise) FLG3 = FLG1 & FLG2 (logical)	14h (20)
OR (bitwise)	|	VAR3 = VAR1 | VAR2 (bitwise) FLG3 = FLG1 | FLG2 (logical)	15h (21)
XOR (bitwise)	^	VAR3 = VAR1 ^ VAR2 (bitwise) FLG3 = FLG1 ^ FLG2 (logical)	16h (22)
NOT	!	FLG3 = !FLG1 (Usable for flags only).	17h (23)
Equal To	=	VAR = <num>, FLG = <0/1>, FLG1 = VAR1 = VAR2	18h (24)
Less Than	<	FLG1 =<0/1>, VAR1 < VAR2	19h (25)
Greater Than	>	FLG1 = <0/1>, VAR1 > VAR2	1Ah (26)
Less Than or Equal To	<=	FLG1 = <0/1> ,VAR1 <= VAR2	1Bh (27)
Greater Than or Equal To	>=	FLG1 = <0/1>, VAR1 >= VAR2	1Ch (28)
Not Equal	<>	FLG1 = <0/1>, VAR1<> VAR2	1Dh (29)
Sine	SIN	VAR1 = SIN VAR2	1Eh (30)
Cosine	COS	VAR1 = COS VAR2	1Fh (31)
Tangent	TAN	VAR1 = TAN VAR2	20h (32)
Arc Sine	ASIN	VAR1 = ASIN VAR2	21h (33)
Arc Cosine	ACOS	VAR1 = ACOS VAR2	22h (34)
Arc Tangent	ATAN	VAR1 = ATAN VAR2	23h (35)
Square Root	SQR	VAR1 = SQR VAR2	24h (36)
Absolute	ABS	VAR1 = ABS VAR2	25h (37)
Integer Part	INT	VAR1 = INT VAR2	26h (38)
Fractional Part	FRC	VAR1 = FRC VAR2	27h (39)

Syntax Rules

These examples are to familiarize the user with the syntax rules for programming. Details on the Variables, Flags and Instructions follow.

1) Use of the < > characters only emphasize a parameter. They are not used in any program format.

2) The ∧ character indicates a space.

3) When setting Variables or Flags, there must be an equal sign <=> after the keyword and before the value. Optional spaces or tabs may be used before and/or after the <=> sign.
   Examples:
   Variables: ACCL=500 VM=100000 [Variables are usually a number value.]
   Flags: BLE=0 GECHE=1 [Flags are usually binary with zero (0) FALSE or one (1) TRUE.]

4) Any Instruction must be followed by a minimum of one (1) space before any parameter.
   Examples:
   LBL∧<name> HOLD∧1 MOVR∧<position>,<mode>

5) Any Instruction that requires more than one (1) parameter entry must have a comma <,> separating the parameters. Spaces or tabs between the comma and the next entry are optional.
   Examples:
   FIOS∧<speed>,<creep>,<line> or FIOS∧<speed>,∧<creep>,∧<line>
   DVF∧<param1>,<param2>,<param3> or DVF∧<param1>,∧<param2>,∧<param3>

6) In the Syntax example below, the first digit is the value of <param1>, the second digit is the value of <param2> and the third digit is the value of <param3>.
   DVF∧2,1,1

7) Some parameters use a zero (0). A zero (0) does not have to be typed but it must be represented by a comma <,>.
   Example:
   DVF∧1,0,2 could be written as DVF∧1,,2
   As shown, no value precedes the second comma. A zero (0) will automatically be assumed for this parameter. (Spaces and tabs are optional.)

8) Some Instructions such as FIOS (**F**ind **I/O S**witch Instruction) require parameters such as <±speed>, <±creep> and <line>. The <±speed> parameter is Maximum Velocity (VM), the <±creep> parameter is Initial Velocity (VI) and the <line> parameter is the I/O line number for the Input. The + or – sign preceding the <speed> and <creep> parameters instruct the unit to move in the positive (+) or negative (–) direction. In the FIOS Instruction these parameters will temporarily override any previously set VM and VI parameters. The Instruction would be written:
   FIOS∧<±speed>,<±creep>,<line> (Spaces and tabs are optional.)

9) In this example, Initial Velocity (VI) and Maximum Velocity (VM) were previously stated as:
   VI=500
   VM=750000
   If the previously programmed Initial Velocity (VI) is to be used, the Instruction is written as:
   FIOS∧±350000,,21
   The Maximum Velocity (VM) is overridden and set to ±350000. Since nothing was entered for the <creep> parameter, the previous VI of +500 would be used. (The positive direction is the default.) If a zero (0) was entered for the <creep> parameter, no VI is used and the entire cycle is executed at the <speed> parameter.

ACCL Variable	Peak Acceleration Variable			
Usage Example	**Unit**	**Range**	**Default**	**Binary Mode Opcode Hex (Decimal)**
ACCL=<num>	User units per second^2	± .0000000000000001 to ± 9,999,999,999,999,999	1000000.000	60h (96)

Description

The ACCL Variable sets the peak acceleration that will be reached by the Control Module in user units per second^2, based upon the value of MUNIT. If the user units have not been set, then the value is in Microsteps per second^2. The actual acceleration profile is maintained by the ACLT variable. The value given by ACCL sets the maximum acceleration that the Control Module will reach.

Related Commands

MUNIT, ACLT, ACL, PFMT

ACL Read Only Status Flag	Acceleration Flag		
Usage Example	**Function**	**Default**	**Binary Mode Opcode Hex (Decimal)**
BR <lbl/addr>, ACL BR <lbl/addr>, !ACL PRINT ACL	<flg> = FALSE (0): Not accelerating. <flg> = TRUE (1) Axis is accelerating.	FALSE (0)	B8h (96)

Description

The ACL Flag is a read only flag. The flag will be in a logic TRUE or “1” state when the axis is accelerating. It will be logic FALSE “0” at all other times. It can be used to branch to a program subroutine for actions such as toggling an output while the axis is accelerating, for example: to power an LED indicator.

Related Commands

ACCL

ACLT Variable	Acceleration Type Variable		
Usage Example	**Parameters**	**Default**	**Binary Mode Opcode Hex (Decimal)**
ACLT=<param>	<param>=0: User Defined. <param>=1: Linear. <param>=2: Triangle S-Curve. <param>=3: Parabolic. <param>=4: Sinusoidal S-Curve.	1 - Linear	61h (97)

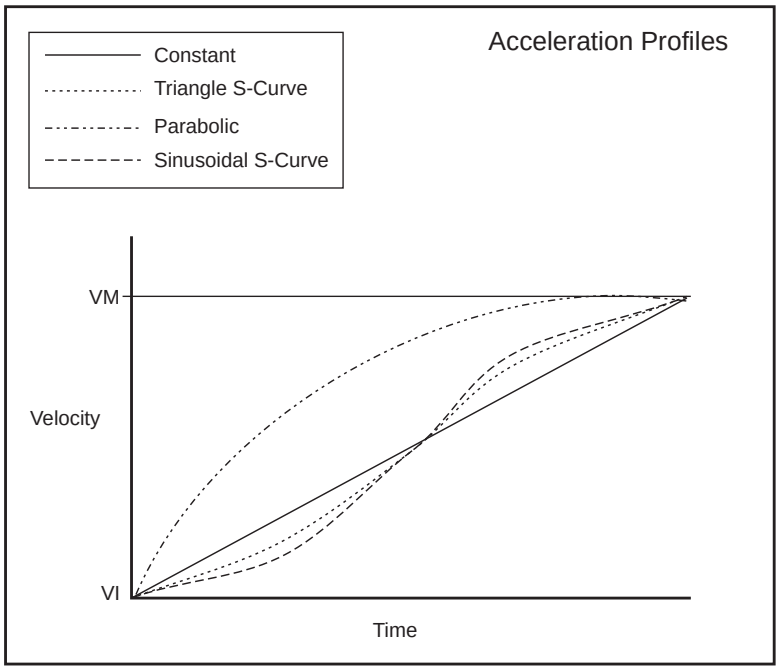
Description

The ACLT Variable defines the type of curve that will be used to build acceleration. The acceleration profiles are defined as follows:

0 – User-defined acceleration profile. This will follow the user-defined points in the ACLTBL (acceleration table) for the acceleration profile.
1 – Constant (linear) acceleration.
2 – Triangle S-Curve profile.
3 – Parabolic profile.
4 – Sinusoidal S-Curve profile.

Comparison of Acceleration Types:

1 – Constant smooth (linear) acceleration from initial to max velocity.
2 – Triangle S-Curve profile.
3 – The parabolic profile best utilizes the speed torque characteristics of a stepper motor since the highest acceleration takes place at low speeds. It will, however, be the profile that results in the maximum jerk, and is not recommended for applications requiring smooth starting and stopping. Such applications would include those that pull a material or move liquid.
4 – The Sinusoidal S-Curve profile is very similar to #2, the triangle S-Curve. The main difference is that it has less jerk when starting or stopping.



Related Commands ACCL, ACLTBL

ACLTBL Variable	Acceleration Table Variable		
Usage Example	**Range**	**Default**	**Binary Mode Opcode Hex (Decimal)**
ACLTBL<num> = <val>	<num> = 0 - 256 <val> = 0.00 - 1.00	Empty	62h (98)

Description

The acceleration table is a table of 256 points that can be used to define a user acceleration profile. The value specified in num 0 is the scale factor for the table. It is used to normalize the acceleration time in relation to constant acceleration (TYPE 1). A point in the table can be specified by setting ACLTBL <num> = <val> as shown in the example below. To use this, all 256 points must be defined.

If ACLTBL num 0 is set to 0 then the table is considered empty. In order for the table to be used, the ACLT, DCLT or LDCLT variable must be set to 0.

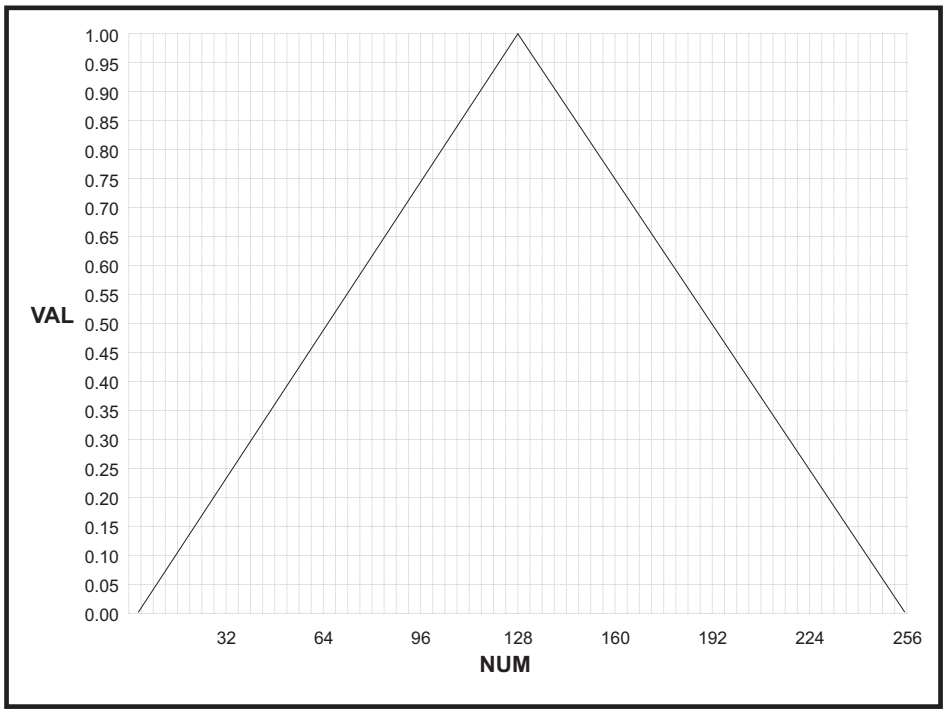
The routine below illustrates how the ACLT variable in all its types, effects the acceleration profile.

```
ACLTBL 0 = ACLFACTOR ≅ ((sum of 1-256)/256)
ACLTBL 1 = 0
ACLTBL 2 = 0.110
ACLTBL 3 = 0.220
```

```
ACLTBL 256 = 0.110
```

Related Commands

ACCL, ACLT, DCLT, LDCLT



ADS Variable	Analog Input Setup Variable		
Usage Example	**Range**	**Default**	**Binary Mode Opcode Hex (Decimal)**
ADS<chan>=<aunit>,<func>, <law>	<chan> = 1 - 2 <aunit> = value <func> = 1 - 2 <law> = 1 - 4	1, 1, 1	B3h (179)

Description

The ADS variable is used to set up the analog input functions of the analog input/joystick module. The following parameters are used:

<chan> = Channel # (1 - 4)
<aunit> = User Unit = MUNIT * AUNIT

<func> = 1: Analog input
<func> = 2: Joystick interface

<law> = 1: Linear
<law> = 2: Square law
<law> = 3: Cube law
<law> adjusts the joystick position to motor velocity transformation.

Related Commands

AIN, JSC, JSDB, JSFS, IJSC

AIN Read Only Variable	Read Analog Input Channel		
Usage Example	**Range**	**Default**	**Binary Mode Opcode Hex (Decimal)**
<var> = AIN <chan>	<var> = Variable to which data is saved. <chan> = 1 - 2		71h (113)

Description

Read only variable causes a read of the analog input channel <chan>. Data is saved to the variable <var>.

Related Commands

ADS, JSC, JSDB, JSFS, IJSC

ALL Keyword	Retrieve All Parameters	
Usage Example		**Binary Mode Opcode Hex (Decimal)**
PRINT ALL IP ALL GET ALL		63h (99)

Description

The ALL keyword is used with GET, IP and PRINT instructions to signify that all types of parameters should be retrieved from nonvolatile memory (NVM), initialized to factory default values, or printed to the serial port.

When used with the GET instruction, all values of variables and flags are retrieved from NVM into working memory (RAM). In addition, the program space in working memory (RAM) is also refreshed from NVM. When used with the IP instruction, all system variables and flags in working memory (RAM) are restored to their factory default settings - user flags and variables are not affected. When used with the PRINT instruction, all variable and flag values are echoed to the host computer.

In order to save the changes made to working memory when ALL is used with the IP instruction, the SAVE instruction must be executed.

Related Commands

PRINT, IP, GET

AOUT Variable	Output Analog Voltage to Channel		
Usage Example	**Range**	**Default**	**Binary Mode Opcode Hex (Decimal)**
AOUT<chan>=<val>	0 - 4095 counts, User Units	0	7Ch (124)

Description

Sets the selected Analog Output Channel to <val>.

Related Commands

DAS

BAUD Setup Variable	Baud Rate Variable			
Usage Example	**Unit**	**Parameters**	**Default**	**Binary Mode Opcode Hex (Decimal)**
BAUD=<param>	bps	<param> = 48: 4800 <param> = 96: 9600 <param> = 19: 19,200 <param> = 38.4: 38,400	9600 bps	64th (100)

Notes

This variable sets the baud rate for serial communications with the control module. It sets the rate for both the RS-232 and RS-485 interfaces. The baud rate is set by indicating the first two digits of the desired rate as shown in the range section below.

In order for the new BAUD rate to take effect, the user must issue the SAVE instruction and then reset the Control Module. When the Control Module is reset, it will communicate at the new BAUD rate. NOTE: You will have to reset your terminal to the default setting of 9600 following any IP (Initialize Parameters) instruction to reestablish communications with the LYNX/MicroLYNX.

Related Commands SAVE

BIO Setup Flag	Binary Mode of Operation Flag		
Usage Example	**Function**	**Default**	**Binary Mode Opcode Hex (Decimal)**
BIO=<flg>	<flg> = FALSE (0):ASCII. <flg> = TRUE (1) Binary.	FALSE (0)	B9h 185)

Description

This flag, when set to TRUE (1), sets the communications mode to binary. When the flag is FALSE (0), the communications mode is ASCII.

Related Commands CSE

BKGD Read Only Status Flag	Background Program Running		
Usage Example	**Status**	**Default**	**Binary Mode Opcode Hex (Decimal)**
BR<lbl/addr>, BKGD BR<lbl/addr>, !BKGD PRINT BKGD	BKGD = FALSE (0): Background program not running. BKGD = TRUE (1): Background Program running.	FALSE (0)	BAh (186)

Description

This Read Only Status Flag indicates whether or not a background program is running. A background program is started by the RUN instruction. The result is two tasks: a foreground task and a background task running at the same time.

Related Commands RUN, BKGDA

BKGDA Read Only Status Variable	Background Program Address Variable		
Usage Example	**Range**	**Default**	**Binary Mode Opcode Hex (Decimal)**
PRINT BKGDA	1 - 8175		65h (101)

Description This variable holds the present instruction address for the background task.

Related Commands BKGD, RUN

BLE Setup Flag	Backlash Compensation Enable Flag		
Usage Example	**Function**	**Default**	**Binary Mode Opcode Hex (Decimal)**
BLE=<flg>	<flg> = FALSE (0): Disabled. <flg> = TRUE (1): Enabled.	FALSE (0)	BBh (187)

Notes Backlash could be described as the amount of mechanical variance in a system. For example, the nut on a leadscrew may not engage until several steps into the move. Again, during a direction change it would also take several steps for the actual motion in the opposite direction to commence. The LYNX/MicroLYNX Control Module is able to compensate for that amount using this feature with the BLM (Backlash Compensation Mode) and BLSH (Backlash Compensation Amount) Variables.

In order to use backlash compensation the function must be enabled. This flag will be used in conjunction with the BLM and BLSH variable to establish the type and amount of backlash compensation employed.

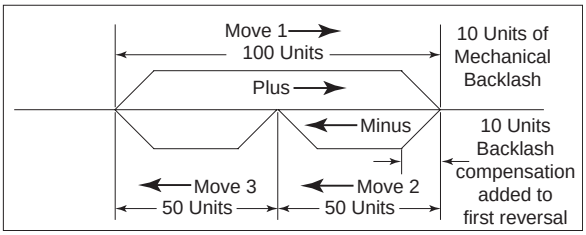
Related Commands BLM, BLSH

BLM Setup Variable	Backlash Compensation Mode		
Usage Example	**Modes**	**Default**	**Binary Mode Opcode Hex (Decimal)**
BLM=<mode>	<mode> = 0: Mathematical compensation. <mode> = 1: Mechanical compensation.	Mode 0	66h (102)

Notes

Mode 0: Mathematical Compensation

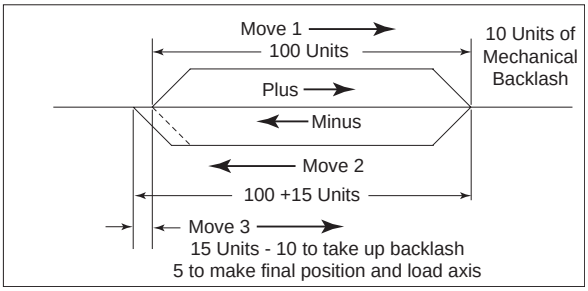
When mathematical backlash compensation is employed the value of BLSH is added to each change of direction. On each reversal move the Control will output the programmed user units plus the backlash units to the driver. This takes up the backlash resulting from the change in direction and completes the move to the correct position.



1. Assuming backlash has already been taken up, Move 1 is plus 100 user units.
2. The axis has 10 user units of backlash which is entered in the program (BLSH=10).
3. Move 2 is 50 user units. The 10 user units of compensation will be added to this move. The control will output 60 user units, 50 to make the move and 10 to take up the backlash.
4. Since the backlash has been taken up, Move 3 is a normal move with no compensation.
5. In the next plus move (reversal) the 10 user units will again be added to take up the backlash.

Mode 1: Mechanical Compensation

Mechanical backlash compensation always "loads" the axis in the direction of the sign (±) of the BLSH. A move in the direction opposite to that indicated by the sign (±) of BLSH will have the value specified by BLSH added to it. A separate move will then be made relative to the sign (±) of BLSH to take up the backlash amount and "load" the axis. Whenever possible, program more backlash than there actually is.



1. Assuming backlash has been removed and the axis is "loaded" in the plus direction, Move 1 is plus 100 user units.

   NOTE: *Whenever possible, always enter a larger compensation value than the actual. This will ensure proper backlash removal and proper axis "loading".*
2. Since the axis has 10 user units of backlash, BLSH=+15 or some value greater than 10 would be programmed.
3. Move 2 is a programmed move of minus 100 user units but because of 10 units backlash, the physical movement of the axis would only be 90 user units. Since Move 2 is opposite the sign of the compensation, 15 user units of compensation will be added for a total of 115 units. Because of the physical backlash the result would be a 5 unit overshoot. The axis will then move back in the plus direction (Move 3) 15 user units – 10 to take up backlash and 5 to to go to the correct position and "load" the axis again.

Related Commands BLE, BLSH

BLSH Setup Variable	Backlash Compensation Amount Variable			
Usage Example	**Unit**	**Range**	**Default**	**Binary Mode Opcode Hex (Decimal)**
BLSH=<num>	Userunits	± .0000000000000001 to ± 9,999,999,999,999,999	0.000	67h (103)

Description

This variable represents the amount of backlash compensation employed in user units (or Microsteps if MUNIT or EUNIT not specified).

The sign indicates direction and is only required when using Backlash Compensation Mode 1 (BLM=1).

If no sign is given i.e. BLSH=15, a plus value is assumed.

A minus sign (-) must always be programmed.

Related Commands

BLE, BLM, MUNIT, EUNIT

BR Program Mode Instruction	Branch Instruction	
Usage Example	**Condition**	**Binary Mode Opcode Hex (Decimal)**
BR <lbl/addr>, <cond>	<lbl/addr> = Program or subroutine label or address. <cond> = Specified: Conditional branch. <cond> = Blank or unspecified: Unconditional branch.	30h (48)

Notes

The branch instruction can be used to perform a conditional or unconditional branch to a routine in a LYNX/ MicroLYNX program. It can also be used to perform loops and IF THEN logic within a program.

There are two parameters to a branch instruction. These are used to perform two types of branches:

Conditional Branch
This type of branch first specifies a label or address where program execution should continue if the second parameter, the condition, is true. The condition parameter may include flags as well as logical functions that are to be evaluated.

Unconditional Branch
In this type of branch the second parameter is not specified, then the execution will continue at the address specified by the first parameter.

Syntax Examples

This section of code will use the branch instruction to execute a segment of code 10 times. In this case we will move a motor 10 user units 10 times. This usage is similar to a loop instruction in a higher level language.

```
VAR LOOPCNT = 0              'Create variable LOOPCNT, set to 0
PGM 100                      'Start program at address 100
LBL LOOPLBL                  'Label program address LOOPLBL
MOVR 10                      'Move the motor 10 user units
HOLD 2                       'Halt prog. execution until motion stops
DELAY 1000                   '1 second delay after motion stops
INC LOOPCNT                  'Increment the variable LOOPCNT
BR LOOPLBL, LOOPCNT<10       'Branch to LOOPLBL if LOOPCNT value is
                             'less than or equal to 10
PRINT "Done!"
END                          'End the program
PGM                          'Return to immediate mode
```

The following section of code will illustrate how a user could use the branch instruction to perform the equivalent of a DO-WHILE loop in a higher language. In this example, while the motor is accelerating, the velocity will be reported to the host terminal or terminal program running on a PC.

```
PGM 200               'Start program at address 200
LBL CNTVEL            'Label address location CNTVEL
MUNIT = 51200/25      'Set the user units to Millimeters (arbitrary)
ACCL = 25             'Set acceleration to 25 mm/sec²
DECL = ACCL           'Set deceleration to 25 mm/sec²
VM = 200              'Set max velocity to 200 mm/sec
VI = VM/100           'Set initial velocity to 20 mm/sec
MOVR 2500             'Perform a relative move of 2500 mm
LBL DOWHILE           'Create subroutine DOWHILE
BR ENDWHILE,ACL = 0   'Conditional branch to routine ENDWHILE when the
                      'acceleration flag is equal to 0.
PRINT "Velocity = ",VEL, " millimeters per second"
BR DOWHILE            'Unconditional branch to routine DOWHILE
LBL ENDWHILE          'Create routine ENDWHILE
PRINT "Motor is at constant velocity =",VEL, " millimeters per second"
END                   'End the Program
PGM                   'Return to Immediate Mode
ENDENPGM PRINT "Done"
```

BREAK Variable	Break Point Variable		
Usage Example	**Function**	**Default**	**Binary Mode Opcode Hex (Decimal)**
BREAK=<num>, <lbl/addr>	<num> = 0: Function disabled. <num> = 1 - 10: Break points. <lbl/addr> = Program label or address where execution will break.	<num> = 0	68h (104)

Notes

Break allows the user to set break points within a LYNX/MicroLYNX program for help in debugging the program. When the program is executed while there are break points set, the program executes continuously until the address or label specified by the break point is encountered. The user can then step through the program by pressing the space bar to execute a single line. If the user wishes to continue execution to another break point or to the end of the program, this can be done by pressing the enter key.

There are 11 entries in the break point table. The first entry (break 0) enables or disables the function. If it is set to 0 the function is disabled, any nonzero value enables the function. The remaining ten entries (break 1 – break 10) hold program addresses at which execution should break awaiting a command to continue from the user. The program address may be entered numerically or by label.

BSY Read Only Status Flag	Busy Flag		
Usage Example	**Status**	**Default**	**Binary Mode Opcode Hex (Decimal)**
PRINT BSY	BSY = FALSE (0): No program running. BSY = TRUE (1): Program running.	FALSE (0)	BCh (188)

Notes

The BSY flag is a read only status flag which will read TRUE (1) when a program is executing. It will be in a FALSE (0) state at all other times.

By setting an output to I/O Type 21, the LYNX/MicroLYNX Product will activate that output whenever the BSY Flag is TRUE.

Related Commands PRINT, EXEC, IOS

CALL Program Mode Instruction	Call Subroutine Instruction	
Usage Example	**Condition**	**Binary Mode Opcode Hex (Decimal)**
CALL <lbl/addr>, <cond>	<lbl/addr> = Subroutine label or address to be invoked if <cond> = TRUE. <cond> = Flag or logical function.	31h (49)

Notes

This function can be used to invoke a subroutine within a program. This allows the user to segment code and call a subroutine from a number of places rather than repeating code within a program.

There are two parameters to the CALL instruction. The first specifies the label or program address of the subroutine to be invoked if the second parameter, the condition, is true. If the second parameter is not specified, the subroutine specified by the first parameter is always invoked. The condition parameter can include flags as well as logical functions that are to be evaluated.

The subroutine should end with a RET instruction. The RET instruction will cause program execution to return to the line following the CALL instruction.

Syntax Example

```
IOS 20=0,1                  'Set IO group 20 to Input, HIGH TRUE
PGM 100                     'Start program at address 100
LBL MAINPGM                 'Label program "MAINPGM"
MOVR 51200                  'Index 51,200 msteps relative to current pos.
HOLD 2                      'Suspend program until motion stops
DELAY 500                   'Delay 500 milliseconds
CALL WAITIN21, IO 21=1      'Invoke subroutine "WAITIN21" when IO line 21=TRUE
BR MAINPGM                  'Loop back to "MAINPGM"
       LBL WAITIN21         'Declare program subroutine "WAITIN21"
       MOVR 51200*5         'Index relative 256,000 msteps
       HOLD 2               'Suspend program until motion stops
       DELAY 500            'Delay 500 milliseconds
       BR WAITIN21, IO 21=1 'Loop to beginning of subroutine while IO 21 is TRUE
       RET                  'Return to main program
END                         'End program
PGM                         'Return to immediate mode
```

Related Commands

RET

CMOVR Program Mode Instruction	Call Saved Relative Move	
Usage Example	**Condition**	**Binary Mode Opcode Hex (Decimal)**
CMOVR <0-127> (linear) CMOVR <0, 1> (S-Curve)	Execute Selected Relative Move.	5Ch (92)

Description

Executes a previously stored relative move, SMOVR.

Related Commands

SMOVR

COMDT Variable	Communication Delay Time		
Usage Example	**Range**	**Default**	**Binary Mode Opcode Hex (Decimal)**
COMDT<val>	0 - 65000 milliseconds	0	90h (144)

Descrption Sets the maximum allowed time between communication characters.

Notes If the time <val> is exceeded then the ERR flag is set, ERROR is set to 4030 and the communication receive buffer is reset. If COMDT is set to ZERO, then this function is disabled.

CP Immediate Mode Instruction	Clear Program Instruction	
Usage Example	**Modes**	**Binary Mode Opcode Hex (Decimal)**
CP <lbl/addr>, <mode>	<lbl/addr> = Subroutine label or address to be cleared. <mode> = 0: Clear only specified program. <mode> = 1: Clear to end of working memory.	32h (50)

Notes This instruction will clear the program space in working memory (RAM) as specified by the instruction parameters.

There are two parameters to the CP instruction. The first specifies the label or program address of the location at which the clear command should begin. The second indicates whether only the specified program or subroutine (0) or the entire program space beginning with the specified address or label (1) should be cleared. If the second parameter is omitted or is specified as 0, the program space is cleared only until the first END or RET is reached. However, if it is specified as 1, the program space is cleared to the end of the program space.

Remember that this instruction operates on working memory (RAM). In order to remove the programs from the program space for the next power up, a SAVE instruction must be executed to save the contents of working memory in permanent memory (NVM).

Syntax Examples

```
CP 1,1          'This will clear all of working memory
CP TSTPRG,0     'This will clear the program labeled TSTPRG only.
CP 2000,1       'Clear from line 2000 to the end of working memory space
CP 2000,0       'Clear from line 2000 to the first END or RET
```

Related Commands SAVE

CPL Immediate/Program Instruction	Twos Complement Instruction	
Usage Example	**Parameter**	**Binary Mode Opcode Hex (Decimal)**
CPL <var/flag>	<var/flag> = Variable or flag.	33h (51)

Notes

This instruction will perform the twos complement of the specified variable or flag.

Has the effect of negating a numerical value. For instance, a variable named TESTVAR has a value of 2. CPL TESTVAR will cause the value of TESTVAR to equal -2. In the case of flags it will also be negated. For example a flag named TESTFLAG = TRUE (1), then CPL TESTFLAG will cause TESTFLAG to be FALSE (0)

Syntax Examples

```
VAR TESTVAR = 2        'Declare user variable "TESTVAR", set value to 2
PGM 100                'Start program at address 100
  LBL TEST             'Label the program "TEST"
  PRINT TESTVAR        'Print the value of TESTVAR
  CPL TESTVAR          'Twos complement TESTVAR
  PRINT TESTVAR        'Print the value of TESTVAR
  END                  'End the program
  PGM                  'Return to immediate mode
```


CSE Setup Flag	Check Sum Enable Flag		
Usage Example	**Function**	**Default**	**Binary Mode Opcode Hex (Decimal)**
CSE=<flg>	<flg> = FALSE (0): Disabled. <flg> = TRUE (1): Enabled.	FALSE (0)	BDh (189)

Notes

When this flag is enabled and binary mode communications is being used, each command sent to the LYNX/MicroLYNX requires a checksum to be included as the last byte of the command. The checksum is only used in binary mode and is the low 8 bits of the 16 bit sum of the address field, character count field, command field, data fields and separators included in the message. Refer to the section Modes of Operation for more information about the format of commands in binary and ASCII modes.

Related Commands

BIO

CTR1 Register Variable	Clock #1 Counter Variable			
Usage Examples	**Unit**	**Range**	**Default**	**Binary Mode Opcode Hex (Decimal)**
<var> = CTR1 <math> CTR<x> <var> = CTR1 <math> <var> PRINT CTR1 CTR1 = <num> BR <lbl/addr>, CTR1=<num>	User Units	±2,147,000,000	0	69h (105)

Notes

This variable contains the raw count representation of the clock pulses sent to the motor drive. If there is no encoder in use (EE = 0), then this value scaled using MUNITS will match the value in the POS variable. If there is an encoder in use (EE = 1), this value scaled using MUNITS can be compared to the POS value to determine the position error for the axis (in this case POS is based on CTR2).

CTR1 is associated with Clock 1 (Step Clock/Direction-Defaulted to Differential I/O channels 11 and 12). Refer to the IOS variable for information on how these channels are set up by default and how they can be changed for your system.

Although the value of CTR1 can be set by the user, it is probably not necessary for the user to set this value directly. The value is automatically updated by the LYNX/MicroLYNX software when the POS value is set. The value of CTR1 is effected when POS is changed regardless of whether an encoder is being used in the system or not (EE = 0 or 1).

The example below will use the value of CTR1 to calculate the position error when working with the encoder functions enabled. Note that the position error is in raw counts and not user units in this case.

```
VAR POSERR            'Define variable POSERR
EE = 1                'Enable the encoder function
MOVR 100              'Perform a relative move of 100 counts
HOLD 2                'Suspend program execution until move completes
POSERR = CTR1-CTR2    'Calculate position Error
PRINT POSERR          'Display position error
```

Related Commands POS, CTR2, MUNIT, EE, IOS,

CTR2 Register Variable	Clock #2 Counter Variable			
Usage Examples	**Unit**	**Range**	**Default**	**Binary Mode Opcode Hex (Decimal)**
<var> = CTR2 <math> CTR<x> <var> = CTR2 <math> <var> PRINT CTR2 CTR2 = <num> BR <lbl/addr>, CTR2=<num>	User Units	±2,147,000,000	0	6Ah (106)

Notes

This variable contains the raw counts representation of the clock edges received from the encoder if one is connected to the LYNX/MicroLYNX Product. If the encoder is in use (EE = 1), then this value scaled by EUNITS is given in POS and the encoder feedback is registered. If the encoder is not being used (EE = 0), the value of CTR2 can be used by the user to manually verify the position of the axis (in this case POS is based on CTR1).

CTR2 is associated with Clock 2 (Default Differential I/O channels 13 and 14). Refer to the IOS variable for information on how these channels are set up by default and how they can be changed for your system. It should be noted that the clock type could effect the clock rate here. For instance, if a quadrature clock type is chosen, the actual count will be four times the number of lines. A 1000 line encoder would produce 4000 counts per revolution of the motor.

If the encoder is in use by the LYNX/MicroLYNX Product (EE = 1), then the value of CTR2 probably need not be set directly as the value will be modified by a change to POS. If, however, the encoder is not in use (EE = 0) but an encoder is connected to the system, the user may directly modify the value of CTR2 in order to set the reference with respect to the motor.

Related Commands POS, CTR1, EUNIT, EE, IOS, HAS, HAE,

CTR3 Register Variable	Clock #3 Counter Variable			
Usage Examples	**Unit**	**Range**	**Default**	**Binary Mode Opcode Hex (Decimal)**
PRINT CTR3 CTR3 = <num>	Clock Pulses (IO15 and 16)	±2,147,000,000	0	6Bh (107)

Notes

This variable contains the raw counts representation of the clock seen on Differential I/O channels 15 and 16. This channel will typically be used to drive a second stepper drive as an event type input for a second encoder, or as the master clock input for the half axis mode.

Again, refer to the IOS variable for information on how these channels are set up by default and how they can be changed for your system. It should be noted that the clock type could effect the clock rate here. For instance, if a quadrature clock type is chosen, the actual count will be four times the number of lines. A 1000 line encoder would produce 4000 counts per revolution of the motor.

Related Commands IOS, RATIO, MUNIT, RATIOE, HAS, HAE,

DAS Setup Variable	Digital to Analog Output Setup Variable		
Usage Example	**Parameters**	**Default**	**Binary Mode Opcode Hex (Decimal)**
DAS <chan> = <AUNIT>,<type>	<chan>= 1-4 (1 & 2, Exp. Slot 1 or 2, 3 & 4, Exp. Slot 3) <AUNIT>= 4095/User Unit - absolute value <AUNIT>= 4095/User Unit x 2 - plus or minus value <type>= 1 - Volts, absolute value <type>= 2 - Volts, ± centered around 2.5v <type>= 3 - Velocity, absolute value <type>= 4 - Velocity, ± centered around 2.5V <type>= 5 - Position, absolute value <type>= 6 - Position, ± centered around 2.5V	1,1	B4h (180)

Description Used to setup the selected ANALOG OUT channel.

Related Commands AOUT

DCL Read Only Status Flag	Deceleration Flag		
Usage Example	**Function**	**Default**	**Binary Mode Opcode Hex (Decimal)**
BR <lbl/addr>, DCL BR <lbl/addr>, !DCL PRINT DCL	<flg> = FALSE (0): Not decelerating. <flg> = TRUE (1) Axis is decelerating.	FALSE (0)	BEh (190)

Notes The Deceleration Flag is a read only status flag which will be TRUE (1) when the Control Module is decelerating the Axis. It will be FALSE (0) at all other times.

Related Commands DECL, DCLT

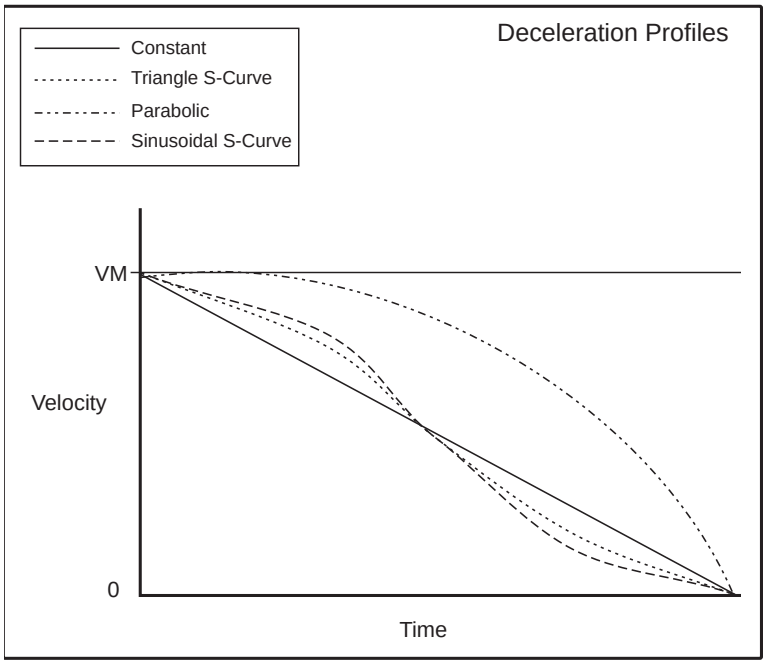
DCLT Setup Variable	Deceleration Type Variable		
Usage Example	**Parameters**	**Default**	**Binary Mode Opcode Hex (Decimal)**
DCLT=<param>	<param>=0: User Defined. <param>=1: Linear. <param>=2: Triangle S-Curve. <param>=3: Parabolic. <param>=4: Sinusoidal S-Curve.	1 - Linear	6Ch (108)

Notes

The DCLT Variable defines the type of curve that will be used to build deceleration.

Comparison of Deceleration Types:

1 – Constant smooth (linear) deceleration from initial to max. velocity.
2 – Triangle S-Curve profile.
3 – The Parabolic profile best utilizes the speed torque characteristics of a stepper motor since the highest acceleration takes place at low speeds. It will, however, be the profile that results in the maximum jerk, and is not recommended for applications requiring smooth starting and stopping. Such applications would include those that pull a material or move liquid.
4 – The Sinusoidal S-Curve profile is very similar to #3, the triangle S-Curve. The main difference is that it has less jerk when starting or stopping.



Related Commands

DECL, ACLTBL

DEC Immediate/Program Instruction	Decrement Variable Instruction	
Usage Example	**Parameter**	**Binary Mode Opcode Hex (Decimal)**
DEC <var>	<var> = User or factory defined variable.	34h (52)

Description The Decrement Variable instruction will decrement the specified variable by one.

Syntax Example In the following example we will write a routine that will perform an operation in a loop 10 times.

```
VAR LOOPCTR =10              'Declare variable "LOOPCTR"
PGM 100                      'Start program at address 100
  LBL LOOP10                 'Label program "LOOP10"
  DEC LOOPCTR                'Decrement LOOPCTR variable
  PRINT "LOOPCTR=", LOOPCTR 'Print value of LOOPCTR variable
  HOLD 2                     'Suspend execution
  DELAY 1000                 'Delay 1 second
  BR LOOP10, LOOPCTR>0       'Loop to beginning of program while LOOPCTR >0
PRINT "DONE"
END
PGM
```

DECL Setup Variable	Peak Deceleration Variable			
Usage Example	**Unit**	**Range**	**Default**	**Binary Mode Opcode Hex (Decimal)**
DECL=<num>	User Units per second^2	±.0000000000000001 to ±9,999,999,999,999,999	1.000000.000	6Dh (109)

Notes

The DECL Variable sets the peak deceleration that will be reached by the Control Module in user units per second^2. If the user units have not been set then the value is in Microsteps per second^2.

The actual deceleration profile is maintained by the DCLT variable. The value given by DECL sets the maximum deceleration that the Control Module will reach.

Related Commands MUNIT, DCLT, DCL

DELAY Program Mode Instruction	Delay Program Execution Instruction		
Usage Example	**Parameter**	**Range**	**Binary Mode Opcode Hex (Decimal)**
DELAY <time>	<time> = Time in milliseconds.	0 - 65535	35h (53)

Notes

The Delay Instruction will delay program execution for a specified number of milliseconds before continuing.

The maximum delay time is 65535 milliseconds or 65.535 seconds.

Syntax Example

In the following example we will set an output, leave it set for 500 milliseconds and then clear it.

```
PGM 100                 'Start program at address 100
LBL SAMPLE              'Label the program "SAMPLE"
IOS 21 = 0, 1           'Define I/O line 41 as a user defined output
IO 21 = 1               'Set I/O line 41 to TRUE (1)
DELAY 500               'Hold I/O line 41 in a TRUE (1) state for 500ms
IO 21 = 0               'Set I/O line 41 to FALSE (0)
END                     'End program
PGM                     'Return to immediate mode
```

DISP Setup Variable	Format Display Variable		
Usage Example	**Range**	**Default**	**Binary Mode Opcode Hex (Decimal)**
DISP=<lines>, <chars>, <wrap>	<lines> = 0 - 255 <chars> = 0 - 255 <wrap> = 0: Do not wrap lines. <wrap> = 1: Wrap lines.	0, 0, 0	6Eh (110)

Notes

Specifies the display format for the print command. There are three parameters for this variable. The first, lines, gives the number of lines per screen. The second, chars, gives the number of characters per line. And the third, wrap, specifies whether or not to wrap long lines to the next line.

Related Commands

PRINT

DN Setup Variable	Device Name Variable		
Usage Example	**Range**	**Default**	**Binary Mode Opcode Hex (Decimal)**
DN=<"char">	ASCII Character: a-z, A-Z, 0-9	Exclamation Mark (!)	6Fh (111)

Notes

The DN Variable stores the device name to be used when the LYNX/MicroLYNX Product is to be addressed in party mode operation.

The name is only used when party mode communications is being used (PARTY = 1). If the QUED flag is set, the LYNX/MicroLYNX Product will respond if addressed by its own name, or by the QUEUE or broadcast name "^".

All LYNX/MicroLYNX system nodes in party mode will respond if the name in a command is given as "*".

When the name is changed it must be saved into the nonvolatile memory if it is to be used in later sessions without being changed again.

Related Commands PARTY, QUED, SAVE

DRVEN Setup Flag	Drive Enable/Disable Flag		
Usage Example	**Function**	**Default**	**Binary Mode Opcode Hex (Decimal)**
DRVEN=<flg>	<flg> = FALSE (0): Disabled. <flg> = TRUE (1): Enabled.	TRUE (1)	BFh (191)

Notes

The DRVEN flag enables or disables the drive module attached to the LYNX or MicroLYNX. This Flag is only relavent to drive modules, external drives are not affected by this flag.

Related Commands DRVTP, DRVRS

DRVRS Flag	Drive Reset Flag		
Usage Example	**Function**	**Default**	**Binary Mode Opcode Hex (Decimal)**
DRVRS=<flg>	<flg> = FALSE (0): Drive not reset. <flg> = TRUE (1): Reset drive.	FALSE (0)	C1h (195)

Notes

The drive reset flag is a momentary flag which, when TRUE (1), will remain so for 10 µs before returning to its default (FALSE) state.

Related Commands DRVEN, DRVTP

DRVTP Read Only Variable	Drive Type Variable		
Usage Example	**Range**	**Default**	**Binary Mode Opcode Hex (Decimal)**
PRINT DRV TP	Response = 2: IM483H Response = 3: IM805H		70h (112)

Notes The DRVTP variable provides a means to interogate the MicroLYNX to determine system configuration.

Related Commands DRVEN, DRVRS

DVF Immediate/Program Instruction	Delete User Defined Variables And Flags Instruction		
Usage Example	**Parameter**	**Default**	**Binary Mode Opcode Hex (Decimal)**
DVF <param1>, <param2>, <param3>	<param1> = 0: All User Variables and Flags deleted. <param1> = 1: Only User Variables deleted. <param1> = 2: Only User Flags deleted. <param2> = 0: All Global and Local User Variables and/or Flags deleted. <param2> = 1: Only Global User Variables and/or Flags deleted. <param2> = 2: Only Local User Variables and/or Flags deleted. <param3> = 0: Delete Global and/or Local User Variables and/or Flags. <param3> = 1: Delete Global and/or Local User Variables and/or Flags and Factory User Variables and/or Flags.	0, 0, 0	37h (55)

Notes This instruction deletes User-defined and/or Factory Variables and Flags.

<param1> defines what is to be deleted. Variables, Flags or both.
<param2> defines the category of Variables and/or Flags to be deleted. Global, Local or both.
<param3> defines User Variables and/or Flags or User and Factory Variables and/or Flags.

Global Variables and Flags are defined in immediate mode, while local Variables and Flags are defined as part of a program.

In the Syntax examples below, the first digit is the value of <param1>, the second digit is the value of <param2> and the third digit is the value of <param3>.

It is not necessary to type the zero (0). A blank will be assumed as a zero (0). The second Syntax Example is written DVF 0,2. It can also be written as: DVF ,2. The fourth example could also be written as: DVF 0,0,0. If you chose to delete all Global and Local User Variables and Flags and all Factory Variables and Flags the last example could be written as: DVF ,,1.

Syntax Examples

```
DVF 1,2             'Delete only local User Variables
DVF 0,2             'Delete all local User Flags and Variables
DVF 2,2             'Delete only local User Flags
DVF                 'Delete all User Flags and Variables
DVF 2,1,1           'Delete all Global User and Factory User Flags
DVF 0,0,1           'Delete all Global, Local & Factory User Vars & Flgs
```

Related Commands VAR, FLG,

ECHO Setup Variable	Echo Mode Variable		
Usage Example	**Modes**	**Default**	**Binary Mode Opcode Hex (Decimal)**
ECHO=<mode>	<mode> = 0: Full Duplex. <mode> = 1: Half Duplex. <mode> = 2: No echo, only responds to PRINT and LIST. <mode> = 3: Echos Immediate Command after execution.	Full Duplex (0)	72h (114)

Notes

This variable specifies whether or not the Control Module should echo commands received via the communications port back over the line.

0 – Echo all information back over communications line. CR/LF Indicates Command Accepted (Full Duplex).
1 – Don't echo the information, only send back prompt. CR/LF Indicates Command Accepted (Half Duplex).
2 – Does not send back prompt, only responds to PRINT and LIST commands.
3 – Saves Echo in Print Queue then executes Command. Prints after execution.

EDB Setup Variable	Encoder Deadband Variable			
Usage Example	**Unit**	**Range**	**Default**	**Binary Mode Opcode Hex (Decimal)**
EDB=<distance>	User Units	0 - 65535	2.000	73h (115)

Notes

This variable defines the + and - length of the encoder deadband for position maintenance.

When position maintenance is enabled, a move is made to the specified encoder position and when the move is complete, the LYNX/MicroLYNX Product maintains position within the specified deadband so that the position remains within (desired position – EDB < actual position < desired position + EDB).

The deadband position is specified in user units if EUNIT has been set. Otherwise, it is specified in encoder counts.

Related Commands

EUNIT, PME, EE

EE Setup Flag	Master Encoder Enable/Disable Flag		
Usage Example	**Function**	**Default**	**Binary Mode Opcode Hex (Decimal)**
EE=<flg>	<flg> = FALSE (0): Disabled. <flg> = TRUE (1): Enabled.	FALSE (0)	C5h (197)

Notes

This is the master enable for all of the encoder functions. It specifies whether or not position maintenance and/or stall detection should be performed if their individual enable flags are set.

If EE is TRUE but STLDE is FALSE, a stall will be detected but not acted upon. In other words, the STALL flag will become TRUE if the encoder does not keep up with the motor, but the motor will not be stopped as a result of the stall. Encoder feedback requires the use of I/O 13 and I/O 14 as the feedback input.

Related Commands

PME, STLDE, EDB, STALL

END Immediate/Program Instruction	END Program Instruction	
Usage Example	**Usage Rule**	**Binary Mode Opcode Hex (Decimal)**
END	Both immediate mode and program.	38h (56)

Notes

Stops the execution of program. It should be the last line of a program written in memory.

If executed in immediate mode, the END instruction stops the execution of the current program as well as any background program that has been started by a RUN instruction.

Syntax Example

A program will probably be identified by a label and run to the END statement. The following example program simply moves the motor to absolute 0.

```
PGM 100
LBL ENDMOVE    'Label Program ENDMOVE
MOVA 0         'Perform absolute move to position 0
END            'End program
PGM
```

Related Commands

EXEC, RUN

ERR Read Only Status Flag	Error Flag		
Usage Example	**Status**	**Default**	**Binary Mode Opcode Hex (Decimal)**
BR <lbl/addr>, ERR BR <lbl/addr>, !ERR PRINT ERR	Response = FALSE (0): No error exists. Response = TRUE (1): Error exists.	FALSE (0)	C6h (198)

Notes

The ERR flag is automatically cleared when a new program is executed. The only way to manually clear the ERR flag is to read the value of the ERROR variable.

By setting the type of an output to 23, the user can specify that the control module should activate the output whenever an error has occurred.

There is an instruction, ONER, which allows the user to specify the execution of a subroutine in the program memory when an error occurs. The subroutine might contain instructions to read the ERROR variable which would clear the ERR flag.

Related Commands

ERROR, ONER, IOS

ERRA Read Only Variable	Error Address Variable		
Usage Example	**Response**	**Default**	**Binary Mode Opcode Hex (Decimal)**
PRINT ERRA Response = <addr1>, <addr2>	<addr1> = Foreground program address. <addr2> = Background program address.	0	B1h (177)

Notes

The ERRA variable allows the user to troubleshoot programs and is automatically set when the ERROR flag is set. It contains the ERROR type and program location. It will clear only when it is replaced by another error address. ERRA will return two numbers. The first number will be the address of the last error in the foreground program, the second will be the address of the last error in the background program.

Related Commands

ERROR, ONER, FAULT

ERROR Read Only Variable	Error Type Variable		
Usage Example	**Range**	**Default**	**Binary Mode Opcode Hex (Decimal)**
PRINT ERROR	See Error Table: Appendix B	0	74h (116)

Notes

This read only variable indicates the program error code for the most recent error that has occurred in the Control Module. The ERROR variable must be read in order to clear the ERR flag.

See Appendix B for a list of possible errors.

Related Commands

ERR, ONER, FAULT, ERRA

ESC Flag	Escape Character Select Flag		
Usage Example	**Function**	**Default**	**Binary Mode Opcode Hex (Decimal)**
ESC=<flg>	<val> = 0: ESC <val> = 1: CTRL^E	FALSE (0)	C2h (194)

Description Selects the ESC character <flg = 0> or CTRL^E <flg = 1> and causes all motion and program execution to stop.

EUNIT Setup Variable	Encoder UnitsVariable			
Usage Example	**Unit**	**Range**	**Default**	**Binary Mode Opcode Hex (Decimal)**
EUNIT=<num>	Encoder counts per user unit	±.0000000000000001 to ±9,999,999,999,999,999	1.000	75h (117)

Notes Although EUNIT is alphabetically before MUNIT in this manual it is recommended that you review MUNIT first to familiarize yourself with some of the factors used.

The EUNIT is the conversion factor for changing encoder counts to user units when an encoder is being used for position feedback.

When the encoder is enabled (EE = 1), POS will have the value of the scaled encoder counts. In other words, CTR2 / EUNIT will equal POS.

Note that if EUNIT is left at 1, the user will be programming in encoder counts and the MUNIT should be Microsteps / Encoder Counts.

Example A 1.8° Stepping Motor is being used and is set to resolution (MSEL) of 256. This means 51200 Microsteps will be generated for each revolution of the motor.

360° ÷ 1.8° = 200 Steps per revolution.
200 Steps × 256 = 51200 Microsteps per revolution.

The motor is equipped with a 500 line Quadrature Input Encoder. Quadrature Input means that there are 4 pulses for each Encoder line. Therefore, there are 2000 Encoder pulses for each motor revolution.

For each Encoder Pulse the motor will move 25.6 Microsteps.

51200 ÷ 2000 = 25.6 Microsteps per Encoder Pulse.

To program your moves in Encoder Pulses, the EUNIT and MUNIT would be set to:

```
EUNIT = 1            'Set EUNIT variable to use Encoder Pulses as the user unit
MUNIT = 51200/2000   'Set MUNIT variable to 25.6 Microsteps per Encoder Pulse
```

NOTE: Enter the MUNIT as a division problem and allow the MicroLYNX to calculate the Microsteps/Pulse. The stepping motor can not physically move in fractional Microsteps but will move to within 5% of a full step.

Make all moves in Absolute mode. This will eliminate any accumulative error that may occur.

In the example below the EUNIT and MUNIT variables will be set to measure position in degrees. In this example the stepper driver is set 256 resolution with a 1.8° step motor with a 500 line quadrature encoder input. Since it is a quadrature input, the encoder resolution is multiplied by 4 to get the base EUNIT of 2000. To illustrate the use of the LYNX/MicroLYNX Control Module's math functions the divide by (/) function will be used. Allowing the LYNX/ MicroLYNX to perform calculations will yield greater positional accuracy.

```
EUNIT = 2000/360      'Set EUNIT variable to use degrees as the user unit
MUNIT = 51200/360     'Set MUNIT variable to monitor position in degrees
```

NOTE: THE MUNIT MUST BE DIVIDED BY THE SAME SCALING FACTOR AS THE EUNIT!

NOTE: The ratio of Microsteps to Encoder Pulses must be a minimum of 3:1!

That is, if you have a 500 line Encoder which is 2000 pulses, the minimum Microsteps you can have is 6000. Looking at the MSEL table the lowest value you could use is 32 which would be 6400 Microsteps with a 1.8° motor.

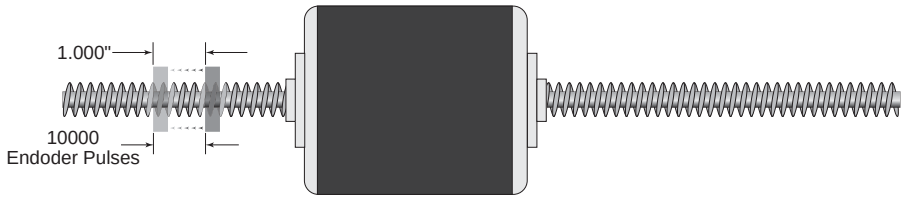
Linear Example

In the "Linear Example" under MUNIT you could program your inch moves in Encoder Pulses. Again a 500 Line Quadrature Input Encoder is used which is 2000 pulses per revolution.

As stated, the 5 Pitch Lead Screw must rotate one (1) revolution for 0.20" of linear travel or 5 revolutions to move one (1) inch. Using those values:

```
EUNIT = 2000/0.20     'Set EUNIT variable to use inches as the user unit
MUNIT = 51200/0.20    'Set MUNIT variable to monitor position in inches
```

Programming in EUNITS, a one (1) inch move would require 10000 Encoder Pulses.



You can make similar calculations for the Rotary Example and the Gearbox Example shown in the MUNIT command description. The only difference is that the scaling factor of your user units will be applied to the EUNIT as well as the MUNIT.

NOTE: THE MUNIT MUST BE DIVIDED BY THE SAME SCALING FACTOR AS THE EUNIT!

Related Commands MUNIT, POS, EE

EXEC Immediate Mode Instruction	Execute Program Instruction	
Usage Example	**Modes**	**Binary Mode Opcode Hex (Decimal)**
EXEC <lbl/addr>, <mode>	<mode> = 0: Normal execution. <mode> = 1: Trace mode. <mode> = 2: Single step mode.	39h (57)

Notes

If the program to be executed is specified by a label, the EXEC instruction can be omitted. For instance, if a program is specified by the label TSTPRG, the command EXEC TSTPRG is equivalent to simply typing TSTPRG.

There are three modes of program execution.

Mode 0 Normal execution, is specified by a mode of 0 (or simply leaving the mode blank).

Mode 1 Trace mode is specified by a mode of 1. This means that the program executes continuously until the program END is encountered, but the instructions are “traced” to the communications port so the user can see what instructions have been executed.

Mode 2 Single step mode is specified by a mode of 2. In this mode, the user can step through the program using the space bar to execute the next line of the program. The program can be resumed at normal speed in this mode by pressing the enter key.

Syntax Examples

```
EXEC TSTPRG, 2          'Execute TSTPRG in single step mode.
EXEC 2000               'Execute program at line 2000 in normal mode.
```

Related Commands PAUS, END

FAULT Read Only Status Flag	Fault Indicator LED Enable/Disable Flag		
Usage Example	**Function**	**Default**	**Binary Mode Opcode Hex (Decimal)**
FAULT=<flg>	<flg> = FALSE (0): Disabled. <flg> = TRUE (1) Enabled.	TRUE (1)	EFh (239)

Notes

This Flag allows the user to enable or disable the red fault indication LED on the Control Module. When TRUE (1) will display all ERROR conditions by illuminating the Fault indicator LED. When FALSE (0) the Fault LED will not illuminate.

In order to clear the FAULT LED you must issue a PRINT ERROR statement.

Related Commands ERROR, ONER, IOS, ERRA

FIOS Immediate/Program Instruction	Find I/O Switch Instruction		
Usage Example	**Parameter**	**Default**	**Binary Mode Opcode Hex (Decimal)**
FIOS <speed>, <creep>, <line>	<speed> = ± speed in user units/sec. <creep> = ± creep in user units/sec. <line> = I/O line number.	<speed> = VM <creep> = VI	3Ah (58)

Notes

This instruction will find the selected I/O switch.

There are three optional parameters for this command:

1) Speed: Specifies the direction and speed that the axis will move until the switch is activated.

2) Creep: Specifies the direction and speed that the axis will move off the switch until it becomes inactive again.

3) Line: Specifies the Input switch to be monitored.

When FIOS is executed, the axis moves in the direction specified by the sign of speed at the speed until the input specified by line becomes active. It then creeps off of the switch in the direction specified by the sign of creep at the creep speed. Motion is stopped as soon as the switch becomes deactivated.

If speed is not specified, the speed used to find the switch is –VM. If creep is not specified, the speed used to move off of the switch is +VI. If line is not specified, the input specified as the home switch (IOS type 12) is monitored for activation.

If a limit switch is encountered before the specified switch is seen, the direction will be reversed until the specified switch is seen. The homing sequence will then take place with the creep moving in the specified direction to the home position.

If both limits are encountered before the specified switch is seen, the motion is stopped and an error is flagged.

Syntax Example

In this example we will use the FIOS command to home the axis on initial power up. We will not specify the line parameter since we want to use the home switch. We will specify the speeds, however. Assume that the MUNIT and EUNIT variables have been set so that the user unit is inches, therefore speeds are specified in inches per second. We will search for the switch at 5 inches per second and come off of it at .1 inch per second.

```
PGM 1
IOS 21=12              'Set IO line 21 to a homing input
PGM100                 'Start program at address 100
LBL FINDIO             'Label program "FINDIO"
FIOS -5,+.1,21         'Find home switch at -5 in/sec, creep off at +0.1 in/sec
HOLD 2                 'Suspend Program execution until motion completes
END                    'End program
PGM                    'Return to immediate mode
```

Related Commands

VM, VI, IOS

FLAGS Keyword	Retrieve Flags Keyword	
Usage Example		**Binary Mode Opcode Hex (Decimal)**
PRINT FLAGS IP FLAGS GET FLAGS		78h (120)

Notes Used with the GET, IP and PRINT commands to specify that all flags should be retrieved from nonvolatile memory (NVM), set to their factory default values, or printed to the serial port, respectively. When used with the GET instruction, only flag values are retrieved from NVM. When used with the IP instruction, only system flag values are set to the factory default parameters. In this case, user-defined flags are not affected. When used with the PRINT instruction, only flag values are echoed to the host computer.

Related Commands PRINT, GET, IP

FLG Immediate/Program Mode Instruction	Define User Flag Instruction	
Usage Example	**Parameters**	**Binary Mode Opcode Hex (Decimal)**
FLG <name>	<name> = 1 to 8 Alpha-numeric Characters +Underscore (_)	3Bh (59)

Notes The name of the flag can be 1 to 8 alphanumeric characters in length. You may use the underscore (_) character in the name as well. The value of the flag can be initialized when it is defined. If it is not specifically initialized, it will have a value of FALSE until it is set.

Flags can be “global” or “local”. A local flag is one that has been defined in a program while a global flag is defined in immediate mode. It should be noted that a local flag is not static, but is erased and declared again whenever the program is executed.

GECHE Setup Flag	Global Echo Enable/Disable Flag		
Usage Example	**Function**	**Default**	**Binary Mode Opcode Hex (Decimal)**
GECHE=<flg>	<flg> = FALSE (0): Disabled. <flg> = TRUE (1) Enabled.	FALSE (0)	C8h (200)

Notes Enable (1) or disable (0) the echo of Global commands. For use in party mode communications only.

A global command is any command that specifies the LYNX/MicroLYNX Product name as the GLOBAL Control module character “*” instead of a specific LYNX/MicroLYNX system node name.

This flag should be TRUE for only one LYNX/MicroLYNX node on the common RS-422 line.

Related Commands PARTY

GET Immediate/Program Mode Instruction	Retrieve Variables and Flags Instruction	
Usage Example	**Parameters**	**Binary Mode Opcode Hex (Decimal)**
GET VARS GET FLAGS GET ALL		3Ch (60)

Notes

Retrieves the specified information from nonvolatile memory (NVM) into working memory (RAM).

There is one optional parameter to this instruction. If there is no value given for this parameter or it is ALL, then all variables, flags and the program space are refreshed in working memory. Alternately, if the parameter is specified as FLAGS only the values of system flags are refreshed, and if the parameter is specified as VARS only the values of the system variables are refreshed.

It should be noted that user-defined flags and variables (those defined using a FLG or VAR instruction) are not refreshed with a GET command.

Related Commands

ALL, FLAGS, VARS, PGM

HAE Setup Flag	Half Axis Mode Enable/Disable Flag		
Usage Example	**Function**	**Default**	**Binary Mode Opcode Hex (Decimal)**
HAE=<flg>	<flg> = FALSE (0): Disabled. <flg> = TRUE (1) Enabled.	FALSE (0)	C9h (201)

Notes

In half axis mode the master clock is taken from the clock input 2, 3 or 4 (line pairs 13-14, 15-16 or 17-18) which have been set for input, clock type and ratio enabled. The primary axis moves as a ratio of this clock based on the factor entered in HAS. This is an implementation of a master follower where the master is input into a clock input and the primary axis follows based on the specified factor.

Related Commands

HAS

HAS Setup Variable	Half Axis Mode Scaling Variable			
Usage Example	**Unit**	**Range**	**Default**	**Binary Mode Opcode Hex (Decimal)**
HAS=<num>	Scaling Factor	–1≤ num <1	1.000	79h (121)

Notes In half axis mode the master clock is taken from a clock input 2, 3 or 4 (line pairs 13-14, 15-16 or 17-18) which have been set for input, clock type and ratio enabled. This is the factor at which the count rate out to the primary drive will follow the external clock in half axis mode. This is an implementation of a master follower where the master is input into the clock input and the primary axis follows based on the specified factor.

HAE must be set to TRUE in order to enable the function.

Related Commands HAE, IOS

HCDT Setup Variable	Hold Current Delay Time Variable			
Usage Example	**Unit**	**Range**	**Default**	**Binary Mode Opcode Hex (Decimal)**
HCDT=<time>	milliseconds	0 - 32,765	500	7Ah (122)

Notes The HCDT variable sets the delay time in milliseconds between the cessation of motion and when the LYNX or MicroLYNX shifts to the holding current level specified by the MHC variable. The delay time is also effected by the MSDT (Motor Settling Delay Time) variable in that the total time from motion ceasing to current change is represented by the sum of MSDT + HCDT.

Related Commands MAC, MRC, MHC, MSDT

HELD Read Only Status Flag	Program Execution Held Flag		
Usage Example	**Status**	**Default**	**Binary Mode Opcode Hex (Decimal)**
BR <lbl/addr> HELD BR <lbl/addr> !HELD PRINT HELD	Status = FALSE (0): Program executing. Status = TRUE (1): Program suspended.	FALSE (0)	CAh (202)

Notes This flag is TRUE (1) when the program is waiting for the position change, velocity change or motion to complete.

Related Commands HOLD

HOLD Program Mode Instruction	Hold Program Execution During A Move Instruction	
Usage Example	**Modes**	**Binary Mode Opcode Hex (Decimal)**
HOLD <mode>	<mode> = 0: Suspend program until position change completes. <mode> = 1: Suspend program until velocity change completes. <mode> = 2: Suspend program until motion completes.	3Dh (61)

Notes

Hold program execution until the specified motion phase completes. There is one optional parameter to the HOLD instruction which specifies how long the program execution should be held. If the parameter is 0 or not specified, the program will suspend until the position change completes (PCHG becomes FALSE). If the parameter is 1, the program will suspend until the velocity change completes (VCHG becomes FALSE). If the parameter is 2, the program will suspend until the motion completes (MVG becomes FALSE).

Syntax Example

In this example we will start a motion and wait for the motion to complete before continuing with the program.

```
MOVR 10         'Perform a relative move of ten user units
HOLD 2          'Suspend program execution until motion completes
```

Related Commands

HELD, PCHG, VCHG, MVG

HOST Setup Flag	Host Interface Enable/Disable Flag		
Usage Example	**Status**	**Default**	**Binary Mode Opcode Hex (Decimal)**
HOST=<flg>	<flg> = FALSE (0): Disabled. <flg> = TRUE (1): Enabled.	FALSE (0)	CBh (203)

Notes

This is the Host Interface flag. It is only relevant in a system that contains several LYNX/MicroLYNX Product nodes in a multi-drop configuration. When this flag is set, the node that will serve as the interface between the Host PC and the rest of the system is connected via the RS-232 port. Other LYNX/MicroLYNX Product nodes in the system are connected together via RS-485 interface.

To properly configure the system, the host computer should be connected to the Host Interface via RS-232. The remaining nodes in the system should then have their RS-485 RX inputs connected to the Host Interface Control module's RS-485 TX output, and their RS-422 TX outputs connected to the Host Interface's RS-485 RX input. The HOST flag of the Host Interface should be set. Host PC communications are received by the Host Interface Control module and forwarded to all of the other control modules in the system via the RS-485 channel. Responses from the Host Interface module are routed to the Host PC via the RS-232 channel, but are not seen by the other system nodes on the RS-485 channel. The Host Interface module to the Host PC via the RS-232 channel routes responses from the other control modules.

Only the Host Interface should have the HOST flag set. All other system nodes should have the flag cleared which allows the control modules to operate on commands received via either the RS-485 or RS-232 ports. In addition, the LYNX/MicroLYNX Products's responses are output to both ports.

It should be noted that there is a switch which allows the user to set the host flag in hardware, but the software overrides the hardware. Therefore, if switch is set for Host in hardware and the user sets the host flag to FALSE (0) in software, the unit will not act as a host interface.

Related Commands PARTY

IJSC Immediate/Program Instruction	Calibrate Joystick Instruction	
Usage Example	**Parameter**	**Binary Mode Opcode Hex (Decimal)**
IJSC		84h (132)

Notes

The IJSC instruction is a new addition to the LYNX/MicroLYNX instruction set. It is added to support the Analog Input/Joystick interface module when operating in joystick mode.

Execution of this command followed by moving the joystick over its range of motion and back to center, then pressing the "ENTER" key or allowing it to time out in 30 seconds will calibrate the joystick.

INC Immediate/Program Instruction	Increment Variable Instruction	
Usage Example	**Parameter**	**Binary Mode Opcode Hex (Decimal)**
INC <var>	<var> = Any user or factory defined variable.	3Fh (63)

Notes

The Increment Variable instruction will increment the specified variable by one.

Syntax Example

In the following example we will write a routine that will perform an operation in a loop 10 times.

```
VAR LOOPCTR = 0              'Declare variable LOOPCTR, set value to 0
PGM 100
LBL LOOP10                   'Declare subroutine LOOP10
INC LOOPCTR                  'Increment the value of LOOPCTR
PRINT "LOOPCTR=", LOOPCTR    'Display the value of LOOPCTR
DELAY 1000                   'Delay Program execution for 1 sec.
BR LOOP10, LOOPCTR<10        'Cond. branch to LOOP10 while LOOPCTR < 10
PRINT "DONE"
END
PGM
```

INP Flag	Input Pending Status Flag		
Usage Example	**Function**	**Default**	**Binary Mode Opcode Hex (Decimal)**
INP=<flg>	<flg> = FALSE (0): <flg> = TRUE (1):	FALSE (0)	F0h (240)

Description

Indicates if User has responded to Input var/flg,1 command.

Related Commands

INPUT

INPUT Program Mode Instruction	User Input Request Instruction	
Usage Example	**Parameter**	**Binary Mode Opcode Hex (Decimal)**
INPUT <var>, <param>	<var> = Any user or factory defined variable. <param> = 0: Suspend program execution while waiting for user input. <param> = 1: Do not suspend program execution.	40h (64)

Notes

Command to request input from the user over the RS-232 or RS-485 channel. The input must be numeric and is input into the variable that is specified as a parameter to the command.

This instruction has been modified since the prior release with the inclusion of the "no wait" parameter <param>. This parameter allows the user to determine whether or not the program execution will suspend while awaiting input from the user. If <param> = 0 or is not specified, program execution will suspend until the input request is satisfied. If <param> = 1, then program execution will continue uninterrupted.

It is up to the programmer to use the PRINT command to request the information from the user, before using the INPUT statement to accept the information into the specified variable. In order to keep the cursor on the same line as the user instructions, the string should be followed by a semicolon as shown in the following example.

The variable used as the parameter for the INPUT instruction may be a system or USER variable. If a USER variable is being used, it must be declared prior to the INPUT instruction using the VAR instruction.

Syntax Example

In the following example we will write a routine that will request that the user input the velocity to be used for the next move

```
VAR  SPEED                    'Declare  "SPEED"  variable
PGM  100                      'Start  program  at  address  100
LBL  SAMPLE                   'Label  the  program  "SAMPLE"
PRINT  "Input  the  velocity  for  the  next  move:";
INPUT  SPEED                  'Input  velocity
SLEW  SPEED                   'Perform  a  relative  move  of  ten  user  units
END
PGM
```

INPUT1 Program Mode Instruction	User Input Request Instruction (LYNX COMM1)	
Usage Example	**Parameter**	**Binary Mode Opcode Hex (Decimal)**
INPUT1 <var>, <param>	<var> = Any user or factory defined variable. <param> = 0: Suspend program execution while waiting for user input. <param> = 1: Do not suspend program execution.	57h (87)

Notes

This is an enhancement of the INPUT instruction in that it will only accept input from LYNX/ MicroLYNX COMM 1, otherwise it operates the same as the INPUT instruction.

INPUT2 Program Mode Instruction	User Input Request Instruction (LYNX COMM2)	
Usage Example	**Parameter**	**Binary Mode Opcode Hex (Decimal)**
INPUT1 <var>, <param>	<var> = Any user or factory defined variable. <param> = 0: Suspend program execution while waiting for user input. <param> = 1: Do not suspend program execution.	58h (88)

Notes This is an enhancement of the INPUT instruction in that it will only accept input from LYNX/ MicroLYNX COMM 2, otherwise it operates the same as the INPUT instruction.

IO Variable	Read/Write IO Variable	
Usage Example	**Range**	**Binary Mode Opcode Hex (Decimal)**
PRINT IO <line/group> IO <line/group> = <0-1/0-63>	<line/group> = I/O lines (21-26, 31-36, 41-46, 51-56) or IO Group (20 - 50)	7Bh (123)

Notes

There are two types of I/O with the LYNX/MicroLYNX system. First, there can be up to eight (8) high speed differential I/O individually programmable as clock inputs or outputs or for general purpose use. If used as inputs, these are digitally filtered with a cutoff frequency that can be set by the user.

Second, there are up to twenty-four (24) general purpose I/O which can be used for special purpose inputs, such as limits or home, as well as general purpose inputs and outputs. As inputs, each is digitally filtered with a cutoff frequency that can be set by the user. For more details on I/O structure and availability by module see the section on Configuring the Digital IO, in the part of this document pertaining to the LYNX/MicroLYNX product purchased.

I/O is divided into the following groups.

Group 10 ..... Up to 8 High Speed Differential I/O line pairs.
Group 20 ..... General Purpose I/O lines 21 - 26
Group 30 ..... General Purpose I/O lines 31 - 36
Group 40 ..... General Purpose I/O lines 41 - 46
Group 50 ..... General Purpose I/O lines 51 - 56

Each digital I/O line can be programmed as Input or Output, as well as have its various functions such as triggering, High/Low TRUE, etc. using the IOS variable. The digital filtering for inputs can be set using the IOF variable.

You can report or change the state of individual inputs or outputs, or you can report or change the binary state of the entire group. In the former case, the response from the LYNX/MicroLYNX will be a 1 if the input or output is active, and a 0 if it is not. In the latter case, the response is a decimal equivalent of the byte that is a bitwise representation, or binary weight of the entire group.

If for some reason the I/O cannot be set (i.e. output shorted, held to True or 1) an error message will be generated. See: Appendix B: Error Table for more details.

Related Commands IOS, IOF

IOF Setup Variable	Digital Input Filtering Variable		
Usage Example	**Range**	**Default**	**Binary Mode Opcode Hex (Decimal)**
PRINT IOF <group> IOF <group> = <param>	<group> = 10 - 50 <param> = 1 - 7	IO Group 10 = 0 IO Groups 20 - 50 = 7	7Dh (125)

Notes

This variable sets the digital filtering to be applied to the specified I/O group.

When setting the digital filtering for the I/O, you must specify the group for which the filter should be applied. This can be group 1 (the high speed I/O) or groups 2 - 5 (the standard and optional I/O).

The filter values used for the high speed differential I/O are different than those used for the general purpose I/O.

IOF Settings for Differential IO (Group 10)

Filter Setting	Cutoff Frequency	Minimum Detectable Pulse Width
0 (default)	5.00 MHz	100 nanoseconds
1	2.50 MHz	200 nanoseconds
2	1.25 MHz	400 nanoseconds
3	625 kHz	800 nanoseconds
4	313 kHz	1.6 microseconds
5	156 kHz	3.2 microseconds
6	78.1 kHz	6.4 microseconds
7	39.1 kHz	12.8 microseconds

IOF Settings for General Purpose Isolated IO (Groups 20 - 50)

Filter Setting	Cutoff Frequency	Minimum Detectable Pulse Width
0	27.5 kHz	18 microseconds
1	13.7 kHz	36 microseconds
2	6.89 kHz	73 microseconds
3	3.44 kHz	145 microseconds
4	1.72 kHz	290 microseconds
5	860 Hz	581 microseconds
6	430 Hz	1.162 milliseconds
7 (default)	215 Hz	2.323 milliseconds

Related Commands IOS, IO

IOS Setup Variable	I/O Configuration Variable/Keyword		
Usage Example	**Range**	**Default**	**Binary Mode Opcode Hex (Decimal)**
See Below	See Below	See Below	7Eh (126)

Description Specifies the set up of the I/O. Is also used as a keyword for the IP instruction.

Usage IOS <Line/Group> = <type>, <i/o>, <h/l>, <l/e>, <clk type>, <ratio>

Default Settings

I/O Group 10

I/O	Function	IOS	Notes
11	CLK1A	1, 1, 1, 0, 2, 0	Direction Output*
12	CLK1B	2, 1, 1, 0, 2, 0	Step Clock Output*
13	CLK2A	3, 0, 1, 0, 1, 0	Quadrature Input CH A
14	CLK2B	4, 0, 1, 0, 1, 0	Quadrature Input CH B
15	CLK3A	5, 0, 1, 0, 1, 0	Quadrature Input CH A
16	CLK3B	6, 0, 1, 0, 1, 0	Quadrature Input CH B
17	CLK4A	7, 0, 1, 0, 1, 0	1 MHz (When as an Output)
18	CLK4B	8, 0, 1, 0, 1, 0	10MHz (When as an Output)

* Internal signal. No available pin assignment.

I/O Groups 20 - 50

I/O	Function	IOS	Notes
21-26	USER	0, 0, 1, 0, 0, 0	Standard “on board” I/O
31-36	USER	0, 0, 1, 0, 0, 0	Optional I/O Expansion
41-46	USER	0, 0, 1, 0, 0, 0	Optional I/O Expansion
51-56	USER	0, 0, 1, 0, 0, 0	Optional I/O Expansion

Notes You can specify the set up for individual I/O or for the entire group of I/O. To specify the group, you would specify 10 for group 10, 20 for group 20, etc. Otherwise, simply specify the I/O number. There are six settings that can be specified for each I/O. The first setting is the I/O type <type>. The type can be one of the following:

Type	Function	Input/Output	Type	Function	Input/Output
0:	USER	Input or Output	13:	LIMIT PLUS	Input
1:	CLK1A (DIR)	Output Only	14:	LIMIT MINUS	Input
2:	CLK1B (SCLK)	Output Only	15:	RESET DRIVE	Input
3:	CLK2A	Input or Output	16:	JOG PLUS	Input
4:	CLK2B	Input or Output	17:	JOG MINUS	Input
5:	CLK3A	Input or Output	18:	MVG	Output
6:	CLK3B	Input or Output	19:	PCHG	Output
7:	CLK4A	Input or Output	20:	VCHG	Output
8:	CLK4B	Input or Output	21:	BSY	Output
9:	GO (EXEC1)	Input	22:	STALL	Output
10:	STOP (SSTP1)	Input	23:	ERR	Output
11:	PAUSE/RES	Input	24:	PAUSD	Output
12:	HOME	Input			

The second setting is Input or Output <i/o>:
0 = Input
1 = Output

The third setting is High/Low True <h/l>:
0 = Low True
1 = High True

The fourth setting is Level/Edge Triggering <l/e>:
0 = Level Triggered
1 = Edge Triggered

The fifth setting is Clock Type <clk type>: (Differential I/O Only)
0 = No Clock
1 = Quadrature
2 = Step/Direction
3 = Up/Down

The sixth setting is Ratio <ratio>: (Differential I/O Only)
0 = No Ratio
1 = Ratio Mode

Syntax Examples

```
IOS 20 = 0          'Set all the inputs in Group 20 to user defined.
IOS 21 = 10,0,1,1   'Set I/O Line 21 to a Stop Input, High True, Edge Triggered.
```

A more detailed discussion on configuring the digital I/O using the IOS variable can be found in I/O configuration section of the part of this document pertaining to the LYNX/MicroLYNX product purchased.

Related Commands IOF, IO, IP, FIOS, LIMSTP, JOGS

IP Immediate/Program Instruction	Initialize Parameters Instruction	
Usage Example		**Binary Mode Opcode Hex (Decimal)**
IP ALL IP VARS IP FLAGS IP IOS		41h (65)

Notes

Initializes specified parameters to the factory defaults in working memory (RAM).

To specify which kind of parameters should be initialized, use the following keywords:

ALL (or blank)	All variables, flags, and I/O settings (IOS)
VARS	Variables only
FLAGS	Flags only
IOS	I/O only

If you want the factory default settings to permanently replace the contents of the specified parameter type in NVM, you must perform a SAVE after the IP instruction. Otherwise, the old values will be restored once power is cycled.

Syntax Example

```
PRINT IOS 20                'Show defualt settings
IOS 20=0,1,1,1,0,0          'Change ios settings
PRINT IOS 20                'Show changes
IP                          'Clear all
PRINT IOS 20                'Show cleared to default
```

Related Commands

ALL, VARS, FLAGS, IOS

JOGS Setup Variable	Jog Speed Variable			
Usage Example	**Unit**	**Range**	**Default**	**Binary Mode Opcode Hex (Decimal)**
JOGS=<speed>	User Units/sec	± .0000000000000001 to 9,999,999,999,999,999	256000.000	83h (131)

Notes

Speed at which the motor should move when a jog motion is performed.

The jog motion is performed in response to an input which is assigned the Jog Plus or Jog Minus type. When inputs have been designated with these types via IOS variables, the closure of the Jog Plus input causes the motor to move in the positive direction at the speed specified by JOGS. Similarly, the closure of the Jog Minus input causes the motor to move in the negative direction at the speed specified by JOGS.

Related Commands

MUNIT, IOS

JSC Setup Variable	Joystick Center Position Variable			
Usage Example	**Unit**	**Range**	**Default**	**Binary Mode Opcode Hex (Decimal)**
JCS=<num>	AUNIT	0 - 4095	2048 (AUNIT=1)	84h (132)

Notes The JSC variable supports the Analog Input/Joystick Interface module and is updated automatically by means of the IJSC instruction, or can be updated manually as shown above.

Related Commands IJSC, JSDB, JSFS, JSE

JSDB Setup Variable	Joystick Deadband Variable			
Usage Example	**Unit**	**Range**	**Default**	**Binary Mode Opcode Hex (Decimal)**
JSDB=<num>	AUNIT		10 (AUNIT=1)	85h (133)

Notes The JSDB variable supports the Analog Input/Joystick Interface module and is updated automatically by means of the IJSC instruction, or can be updated manually as shown above.

Related Commands IJSC, JSC, JSFS, JSE

JSE Setup Flag	Joystick Enable/Disable Flag		
Usage Example	**State**	**Default**	**Binary Mode Opcode Hex (Decimal)**
JSE = <flg>	<flg> = FALSE (0): Disabled. <flg> = TRUE (1): Enabled.	FALSE (0)	D0h (208)

Notes The JSE flag enables/disable joystick (velocity) mode for the MicroLYNX Analog Input/Joystick Module.

Related Commands IJSC, JSC, JSFS, JSDB

JSFS Setup Variable	Joystick Full Scale Variable			
Usage Example	**Unit**	**Range**	**Default**	**Binary Mode Opcode Hex (Decimal)**
JSFS=<num>	AUNIT		2038	B5h (133)

Notes The JSFS variable supports the Analog Input/Joystick Interface module and is updated automatically by means of the IJSC instruction, or can be updated manually as shown above.

Related Commands IJSC, JSC, JSDB, JSE

JSHCE Flag	Enable Hold Current when JoyStick & Jog at zero.		
Usage Example	**State**	**Default**	**Binary Mode Opcode Hex (Decimal)**
JSHCE = <flg>	<flg> = FALSE (0): Disabled. <flg> = TRUE (1): Enabled.	TRUE (1)	C4h (196)

Description Enables Hold Current when velocity = 0, flg = 1. Otherwise Run Current will be active all of the time.

Related Commands JSE

LBL Program Mode Instruction	Label Program/Subroutine Instruction	
Usage Example	**Parameter**	**Binary Mode Opcode Hex (Decimal)**
LBL <name>	<name> = 1 - 8 Alphanumeric characters and underscore (_).	42h (66)

Notes This instruction will label the address of a program or subroutine within a program.

The name of the label can be 1 to 8 alphanumeric characters in length. You may use the underscore (_) character in the name as well. The first character of a label cannot be a numeral.

Subroutine calls, branches, program execution, events (trip) and break points can refer to the label instead of the address.

Syntax Example

```
PGM 100                 'Begin program at address line 100 of memory
LBL MY_PGM              'Name the program MY_PGM
PRINT "This is my program"
END                     'End the program
```

Related Commands CALL, BR, EXEC, IT[1-4], TI[1-4], TP[1-4], VT, BREAK

LDCLT Setup Variable	Limit Deceleration Type Variable		
Usage Example	**Parameters**	**Default**	**Binary Mode Opcode Hex (Decimal)**
LDCLT=<param>	<param>=0: User Defined <param>=1: Linear <param>=2: Triangle S-Curve <param>=3: Parabolic <param>=4: Sinusoidal S-Cuve	1 - Linear	86h (134)

Notes

The LDCLT Variable defines the type of curve that will be used to build deceleration when a limit has been hit. The deceleration profiles are defined as follows:

0 – User defined deceleration profile. This will follow the user defined points in the ACLTBL (acceleration table) for the acceleration profile.
1 – Constant (linear) deceleration.
2 – Triangle S-Curve profile.
3 – Parabolic profile.
4 – Sinusoidal S-Curve profile.

See DCLT in this section for an graphic example of deceleration types.

Comparison of Deceleration Types:
1 – Constant smooth (linear) deceleration from initial to max velocity.
2 – Triangle S-Curve profile.
3 – The Parabolic profile best utilizes the speed torque characteristics of a stepper motor since the highest acceleration takes place at low speed. It will, however, be the profile that results in the maximum jerk and is not recommended for applications requiring smooth starting and stopping. Such applications would include those that pull a material or move liquid.
4 – The Sinusoidal S-Curve profile is very similar to #3, the triangle S-Curve. The main difference is that it has less jerk when starting or stopping.

Related Commands

DECL, ACLTBL

LDECL Setup Variable	Limit Deceleration Variable			
Usage Example	**Unit**	**Range**	**Default**	**Binary Mode Opcode Hex (Decimal)**
LDECL=<num>	User Units per second^2	± .0000000000000001 to ± 9,999,999,999,999,999	1.000000.000	87h (135)

Notes

The LDECL Variable sets the peak deceleration that will be reached by the LYNX or MicroLYNX when a limit is reached in user units per second^2. If the user units have not been set then the value is in Microsteps per second^2.

The actual deceleration profile is maintained by the LDCLT variable. The value given by LDECL sets the maximum deceleration that the Control Module will reach.

Related Commands

MUNIT, LDCLT

LIMSTP Setup Flag	Limit Stop Flag		
Usage Example	**Status**	**Default**	**Binary Mode Opcode Hex (Decimal)**
LIMSTP=<flg>	<flg> = FALSE (0): Do not stop program. <flg> = TRUE (1): Stop program.	FALSE (0)	D2h (210)

Notes The Limit Stop Flag specifies whether or not the program should be stopped automatically when a limit is reached. A TRUE (1) stops the program and a FALSE (0) does not.

Regardless of the state of LIMSTP, an error is generated when a limit is reached. If LIMSTP is FALSE (0) when a limit is reached, the program will continue to run. In this case, the user should write code to stop the axis in the routine that is executed the ONER command. This gives the user flexibility in how motion should be stopped when a limit is reached.

Related Commands ONER

LIST Immediate Mode Instruction	List Stored Program Space Instruction	
Usage Example	**Parameter**	**Binary Mode Opcode Hex (Decimal)**
LIST <lbl/addr>, <flg>	<lbl/addr> = Starting label or address <flg> = 0: List through first END. <flg> = 1: List through end of program space.	43h (67)

Notes If LIST is issued with no starting address specified, then the entire program space is reported to the host. If it is issued with a starting address and no stop flag or a stop flag of 0, then the program space is listed from the specified starting address to the first END that is encountered. Finally, if it is issued with a starting address and a stop flag of 1, then the program space is listed starting from the specified address and continuing until the end of the program space.

LOCK Flag	Lock Program Space		
Usage Example	**Status**	**Default**	**Binary Mode Opcode Hex (Decimal)**
LOCK=<flg>	<flg> = FALSE (0): Disabled. <flg> = TRUE (1): Enabled.	FALSE (0)	D1h (209)

Notes When LOCK = 1, the program cannot be viewed or modified. Lock can only be cleared by executing CP1,1 or RTFD. If LOCK was saved, CP and RTFD should be followed by a SAVE.

Related Commands PGM, CP, RTFD, LIST, SAVE

LOGO Setup Flag	Sign On Banner Enable/Disable Flag		
Usage Example	**Status**	**Default**	**Binary Mode Opcode Hex (Decimal)**
LOGO=<flg>	<flg> = FALSE (0): Disabled. <flg> = TRUE (1): Enabled.	TRUE (1)	D3h (211)

Notes This simply controls whether or not when the LYNX/MicroLYNX Product powers up a sign-on banner is echoed out the serial port. This banner consists of copyright and version information.

MAC Setup Variable	Motor Acceleration Current Setting Variable			
Usage Example	**Unit**	**Range**	**Default**	**Binary Mode Opcode Hex (Decimal)**
MAC=<percent>	Percent	0 - 100	25	88h (136)

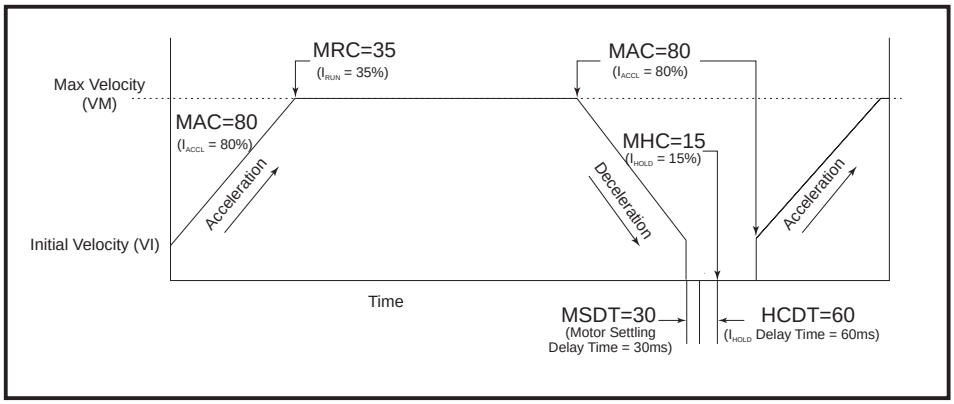
Notes This variable controls the percent of driver output current to be used when the axis is accelerating. See the section on current control in the part of this document pertaining to your product for more information. The figure below illustrates the relationship between the current control variables.

Related Commands MRC, MHC, PMHCC

MHC Setup Variable	Motor Holding Current Setting Variable			
Usage Example	**Unit**	**Range**	**Default**	**Binary Mode Opcode Hex (Decimal)**
MHC=<percent>	Percent	0 - 100	5	8Ah (138)

Notes This variable controls the percent of driver output current to be used when the axis is between moves. See the section on current control in the part of this document pertaining to your product for more information.

Related Commands MRC, MAC, PMHCC

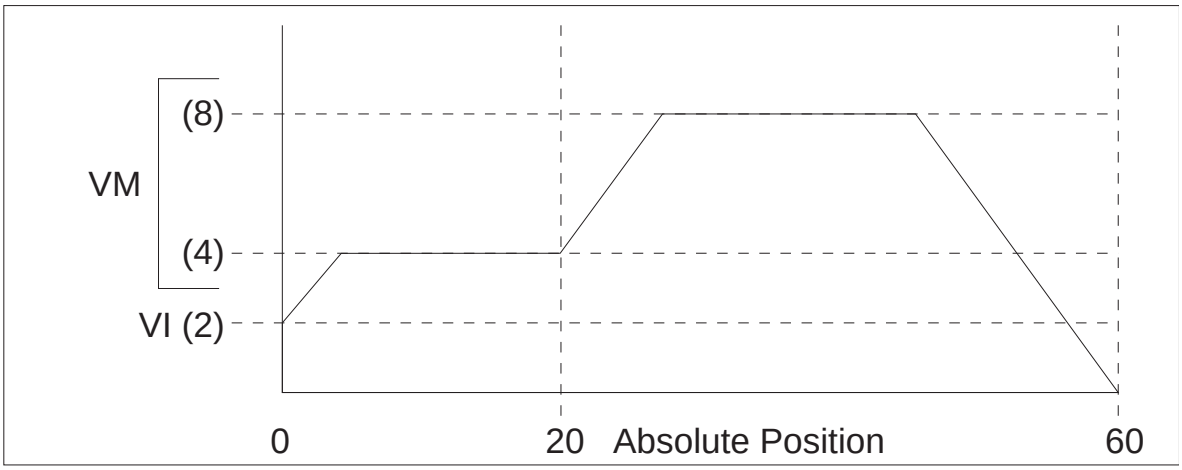


MOVA Immediate/Program Instruction	Move To Absolute Position Instruction		
Usage Example	**Parameters**	**Default**	**Binary Mode Opcode Hex (Decimal)**
MOVA <position>, <mode>	<position> = ± Absolute position. <mode> = 0: Decelerate to position and stop. <mode> = 1: Do not decelerate, move part of profile.	Mode 0	44h (68)

Notes

There are two parameters to the MOVA instruction. The first specifies the absolute position to which the axis should move. The second specifies the mode of the movement. If mode = 0 then the axis should just stop when the specified position is reached. If mode = 1 then the motion is part of a profile and the motor should not decelerate to the specified position. In this case, it is expected that a new motion will take place immediately after the position is reached, so the motion continues at the final speed. Note that if mode is not specified, it is the same as having specified a mode of 0.

If MUNIT has been specified, then the position should be given in user units. Otherwise, the position should be specified in Microsteps.



Syntax Example

This example will use the MOVA instruction to create the profile shown below. Ensure that your start position is set to absolute 0.

```
POS = 0                 'Set Position to 0
PGM 100                 'Start program at address 100
LBL MOVADEMO            'MOVADEMO program
VM = 4                  'Maximum velocity set to 4 user units/sec for move 1
MOVA 20,1               'Index to absolute position 20, do not decelerate
HOLD 0                  'Suspend program execution until completion of position change
VM = 8                  'Maximum velocity set to 8 user units/sec for move 2
MOVA 60                 'Index to absolute position 60, decelerate and stop
END                     'End program
PGM                     'Return to immediate mode
```

Related Commands

VI, VM, ACL, ACLT, DCL, DCLT

MOVR Immediate/Program Instruction	Move To Relative Position Instruction		
Usage Example	**Parameters**	**Default**	**Binary Mode Opcode Hex (Decimal)**
MOVR <position>, <mode>	<position> = ± Relative position. <mode> = 0: Decelerate to position and stop. <mode> = 1: Do not decelerate, move part of profile.	Mode 0	45h (69)

Notes

The primary difference between MOVA and MOVR is that where MOVA indexes to a position, MOVR will index a distance from the current position.

There are two parameters to the MOVR instruction. The first specifies the relative position to which the axis should move. The second specifies the mode of the movement. If mode = 0 then the axis should just stop when the specified position is reached. If mode = 1 then the motion is part of a profile and the motor should not decelerate to the specified position. In this case, it is expected that a new motion will take place immediately after the position is reached, so the motion continues at the final speed. Note that if mode is not specified, it is the same as having specified a mode of 0.

If MUNIT has been specified, then the position should be given in user units. Otherwise, the position should be specified in Microsteps.

Syntax Example

```
MOVR -10      'Specify a relative move of 10 user units in the -
direction
```

A profile within a program can be performed in the same fashion as the example given in the MOVA example. If MOVR is used, then the motion would start from the current location.

Related Commands VI, VM, ACL, ACLT, DCL, DCLT

MRC Setup Variable	Motor Run Current Setting Variable			
Usage Example	**Unit**	**Range**	**Default**	**Binary Mode Opcode Hex (Decimal)**
MRC=<percent>	Percent	0 - 100	25	8Ch (140)

Notes

This variable controls the percent of driver output current to be used when the axis is at velocity. See the section on current control in the part of this document pertaining to your product for more information. Figure 4:4 illustrates the relationship between the current control variables.

Related Commands MAC, MHC, PMHCC

MSDT Setup Variable	Motor Settling Delay Time Variable			
Usage Example	**Unit**	**Range**	**Default**	**Binary Mode Opcode Hex (Decimal)**
MSDT=<time>	Time in milliseconds.	0 - 32,765	0	8Eh (142)

Notes

Specifies the motor settling delay time. This is the time between moves if consecutive motions are executed. The PCHG and MVG flags are not cleared until the settling time has elapsed, so the settling time is included in the move time and will effect the HOLD command.

Related Commands PCHG, MVG, HOLD

MSEL Setup Variable	Motor Resolution Select Variable		
Usage Example	**Parameters**	**Default**	**Binary Mode Opcode Hex (Decimal)**
MSEL=<param>	See Table Below	256	91h (145)

Notes

The MSEL variable controls the microstep resolution of the MicroLYNX or driver module. There are 14 parameters that can be used with this variable, 8 binary and 6 decimal. The table below illustrates the parameter settings and their associated resolutions for a 1.8° stepper motor.

If using a motor with a step angle other than 1.8°, the microsteps/rev resolution will change with the step angle of the motor.

For example: a .45° step angle motor (800 full steps/rev) with MSEL variable set to MSEL=16, or 16 microsteps/step will have a resolution of 12,800 microsteps/rev.

The MSEL parameters given in the table below are the only valid parameters that will be accepted by the LYNX/MicroLYNX.

Microstep Resolution Settings	
MSEL Parameter (Microsteps/Step)	**Microsteps/Rev**
Binary Microstep Resolution Settings (1.8° Motor)	
2	400
4	800
8	1,600
16	3,200
32	6,400
64	12,800
128	25,600
256	51,200

Decimal Microstep Resolution Settings (1.8° Motor)	
5	1,000
10	2,000
25	5,000
50	10,000
125	25,000
250	50,000

MUNIT Setup Variable	Motor Units Variable			
Usage Example	**Unit**	**Range**	**Default**	**Binary Mode Opcode Hex (Decimal)**
MUNIT=<num>	Microsteps per User Unit	±.0000000000000001 to ±9,999,999,999,999,999	1.000	91h (145)

Notes

The MUNIT is the conversion factor for changing Microsteps to user units. The user units may be a linear measure such as inches or millimeters or a rotary measure such as degrees. There are several factors that are required to determine the MUNIT. They are:

- The user's desired units of measure.
- The programmed MSEL (resolution) value.
- Any mechanical devices that increase or decrease the mechanical movement of the motor such as the pitch of a lead screw, the ratio of a gearbox, the diameters of a belt and pulley system etc.

MUNITS are used when the encoder is not enabled (EE = 0). The Position (POS) will have the value of the scaled Microsteps. In other words, the value of Counter 1 (CTR1) which is in Microsteps will be divided by the MUNIT value. The result will equal the POS.

If the encoder is enabled (EE = 1), then the user units are entered as EUNITS. The conversion will be based on encoder counts to user units.

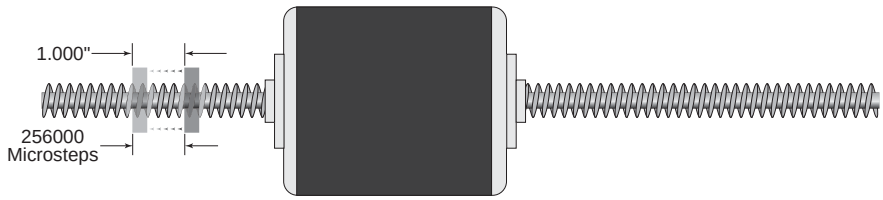
Related Commands

EUNIT, POS, EE

Linear Example

Calculate the MUNITS with the following factors.

- A 1.8° Stepping Motor
- A 5 Pitch Leadscrew
- An MSEL (resolution) Value of 256
- User Units are to be in Inches



A 1.8° Stepping Motor = 200 Steps/Revolution (360° ÷ 1.8° = 200 Steps).

With the MSEL resolution set at 256 the Microsteps will be 51200.
(200 Steps × 256 = 51200 Microsteps/Revolution.)

The travel of a 5 Pitch Leadscrew is 0.200 Inches/Revolution. (1" ÷ 5 = 0.20 inches or 5 revolutions to move 1 inch.)

The MUNITS to move 1" would be 256000. (1" = 5 Revolutions × 51200 Microsteps = 256000.)

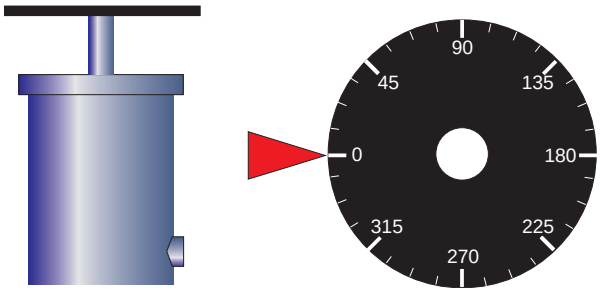
```
MUNIT = 256000
```

To move 1" you would program:

```
MOVR 1          'make a relative move of 1 which will equal 1" of travel
```

Rotary Example Calculate the MUNITS with the following factors.

- A 1.8° Stepping Motor
- A 360° indexing wheel
- An MSEL (resolution) Value of 64
- User Units are to be in Degrees



A 1.8° Stepping Motor = 200 Steps/Revolution (360° ÷ 1.8° = 200 Steps).

With the MSEL resolution set at 64 the Microsteps will be 12800.
(200 Steps × 64 = 12800 Microsteps/Revolution.)

NOTE: Since 12800 is not evenly divisible by 360° it is recommended that you enter the MUNIT as a math function and allow the MicroLYNX to calculate the value. (51200 ÷ 360 = 25.6.)

```
MUNIT = 12800/360
```

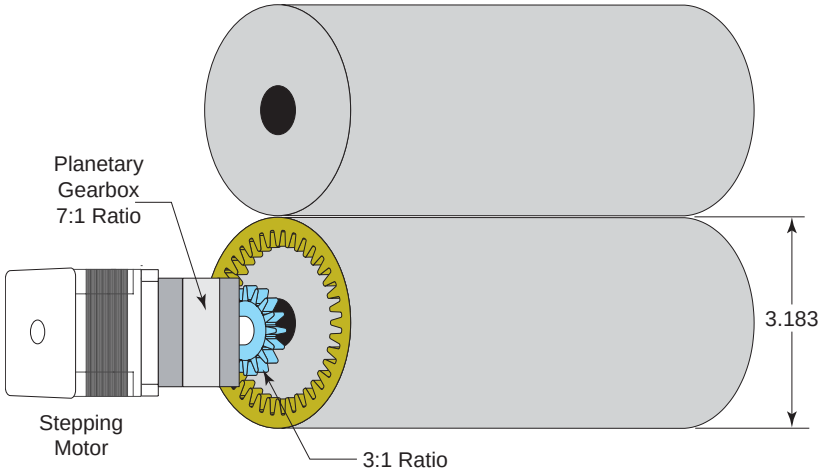
When operating with fractional values it is recommended that you program all moves in Absolute Mode. This will eliminate accumulative errors caused by rounding.

To move 15° you would program:

```
MOVA 15         'make an absolute move of 15 which will equal 15° of rotation
```

Gearbox Example Calculate the MUNITS with the following factors.

- A 1.8° Stepping Motor
- An MSEL (resolution) Value of 256
- A Planetary Gearbox with a 7:1 ratio
- A 3.183” diameter (10” circumference) pinch roller with a 3:1 drive gear
- User Units are to be in Inches of feed through the rollers



A 1.8° Stepping Motor = 200 Steps/Revolution (360° ÷ 1.8° = 200 Steps).

With the MSEL resolution set at 256 the Microsteps will be 51200.
(200 Steps × 256 = 51200 Microsteps/Revolution.)

To move 1” of feed at the pinch roller the roller must rotate 0.10 revolutions.
(1 revolution ÷ 10” circumference = 0.10 revolutions.)

At a 3:1 ratio, the pinch roller pinion gear will have to rotate 0.30 revolutions.
(0.10 × 3 = 0.30.) The output of the planetary gearbox will also have to turn 0.30 revolutions.

At a 7:1 ratio, the stepping motor will have to turn 2.10 revolutions.
(0.30 × 7 = 2.10.)

You could also multiply the drive ratios of 3:1 and 7:1 for an overall motor to roller ratio of 21:1. The roller will have to rotate 0.10 revolutions to feed 1” which means the motor will have to rotate 2.1 revolutions. (21 × 0.10 = 2.10.)

The stepping motor will have to move 107520 mictosteps for 2.10 revolutions.
(51200 × 2.10 = 107520.)

```
MUNIT = 107520
```

When operating fractional values it is recommended that you program all moves in Absolute Mode. This will eliminate accumulative errors caused by rounding.

To move 1” you would program:

```
MOVA 1          'make an absolute move of 1”
```


MVG Read Only Status Flag	Moving Flag		
Usage Example	**Status**	**Default**	**Binary Mode Opcode Hex (Decimal)**
BR <lbl/addr> MVG BR <lbl/addr> !MVG PRINT MVG	MVG = FALSE (0): Motor is stationary. MVG = TRUE (1): Motor is moving.	FALSE (0)	D5h (213)

Notes

Read only status flag which is TRUE (1) whenever the motor is moving.

This flag is TRUE (1) whenever the motor is moving regardless of the type of move, point-to-point, jog or slew. When a profiled move is taking place, this flag does not become FALSE (0) until the motion command with mode 0 has completed.

Related Commands PCHG, VCHG

NOP Program Mode Instruction	No Operation Instruction		
Usage Example	**Parameters**	**Default**	**Binary Mode Opcode Hex (Decimal)**
NOP			46h (70)

Notes

This instruction is used to fill up one byte of program space. It can be used if, in editing a program, there is a change in the line boundary that causes a gap in the program. It can also be used to leave space for future instructions. It is recommended, however, that programs are written to a file using a text editor and downloaded to the LYNX/MicroLYNX Product during debug. This will save a great deal of retyping during debug of the program.

Syntax Example

```
POS=0                 'Set position to 0
PGM 100               'Start program at address 100
      LBL NOPDEMO     'Label program "NOPDEMO"
      VM 4            'Max velocity 4 user units/sec
      NOP             'No operation
      MOVA 20,1       'Move absolute 20 user units, do not decelerate
      HOLD 0          'Suspend prog. until position change completes
      NOP             'No operation
      VM 8            'Max velocity 8 user units/sec
      MOVA 60         'Move absolute 60 user units, decelerate and halt
      NOP             'No operation
END                   'End Program
PGM                   'Return to immediate mode
```

ONER Immediate/Program Instruction	On Error Instruction		
Usage Example	**Parameters**	**Default**	**Binary Mode Opcode Hex (Decimal)**
ONER <lbl/addr>	<lbl/addr> = Subroutine to be called on error.		47h (71)

Notes

When an error occurs in a program or due to an immediate command, the specified subroutine is called. If a program was running when the fault occurs, once the error routine completes, program execution continues with the instruction after the one that caused the error. A program need not be running for the subroutine specified by ONER to run.

The error function is disabled by setting the address parameter of a subsequent ONER command to 0 or resetting the LYNX/MicroLYNX Product.

Syntax Example

Executing the following program will cause the above routine to be called when an error occurs, reporting the error to the host.

```
PGM 100                     'Start program at address 100
 LBL ERR_HND                'Label program "ERR_HND"
 PRINT "Error Number  ",ERROR,
 RET                        'Return from subroutine
 ONER ERR_HND               'On error, goto ERR_HND
END                         'End program
PGM                         'Return to immediate mode
```

If the error report is no longer desired it can be turned off as follows:

```
ONER 0
```

Related Commands

ERR, ERROR,

PARTY Setup Flag	Party Mode Enable/Disable Flag		
Usage Example	**Status**	**Default**	**Binary Mode Opcode Hex (Decimal)**
PARTY=<flg>	<flg> = FALSE (0): Disabled. <flg> = TRUE (1): Enabled.	FALSE (0)	D7h (215)

Notes

This flag should be set to TRUE (1) for LYNX/MicroLYNX systems that are used in a multidrop system (multiple LYNX/MicroLYNX Products connected on a common RS-485 channel.) It should be left as FALSE (0), the factory default, if a single unit is used.

While in PARTY mode, a LYNX/MicroLYNX system node will respond to commands that are addressed to its name (given in DN). In addition, it will respond to global commands which are specified by the "*" character in the name field. Also, if its QUED flag is TRUE, the system node will respond to commands which are specified by the "^" character in the name field. Also the controller will respond to ESC and ^C.

There is a hardware switch to enable party mode as well, but the software setting will override it.

Note: A delay time between command requests to the MicroLYNX must be considered to allow the MicroLYNX time to interpret a command and answer the host before a subsequent command can be sent. The time between requests is dependent on the command and the corresponding response from the MicroLYNX.

Related Commands HOST, QUED

PAUS Immediate Mode Instruction	Pause Program Execution Instruction		
Usage Example	**Parameters**	**Default**	**Binary Mode Opcode Hex (Decimal)**
PAUS			48h (72)

Notes

Suspends the executing program as well as any motion that is executing. The way the motion is suspended and resumed is determined by the value of PAUSM.

Immediate commands are allowed while the control module is paused.

To continue the program, use the RES instruction. To abort the program, use the END instruction.

Related Commands RES, END, PAUSD, PAUSM

PAUSD Read Only Status Flag	Paused Program Execution Flag		
Usage Example	**Status**	**Default**	**Binary Mode Opcode Hex (Decimal)**
BR <lbl/addr>, PAUSD BR <lbl/addr>, !PAUSD PRINT PAUSD	PAUSD = FALSE (0): Program not paused. PAUSD = TRUE (1): Program paused.	FALSE (0)	D8h (216)

Notes

This read only status flag will indicate whether or not a program has been paused.

Related Commands PAUS

PAUSM Setup Variable	Pause Mode Variable		
Usage Example	**Parameters**	**Default**	**Binary Mode Opcode Hex (Decimal)**
PAUSM=<mode>	<mode>=0: Normal deceleration, resume with RES. <mode>=1: LDECL deceleration, resume with RES. <mode>=2: Complete motion, stop with normal deceleration. <mode>=3: Complete motion, stop with LDECL deceleration. <mode>=4: Normal deceleration, no resume with RES. <mode>=5: LDECL deceleration, no resume with RES.	Mode 0	92h (146)

Notes

Determines how motion is stopped in response to the PAUS instruction and whether or not it is restarted in response to the RES instruction.

The following describes how motion is stopped and resumed for each value of PAUSM:

0 Interrupt motion with normal deceleration (DECL) and resume motion in response to a RES instruction.
1 Interrupt motion with the LDECL deceleration and resume motion in response to a RES instruction.
2 Complete the current motion stopping with the normal deceleration (DECL).
3 Complete the current motion stopping with the LDECL deceleration.
4 Interrupt motion with normal deceleration (DECL) but don't resume motion in response to a RES instruction.
5 Interrupt motion with the LDECL deceleration but don't resume motion in response to a RES instruction.

Related Commands PAUS, PAUSD, DECL, LDECL, RES

PCHG Read Only Status Flag	Position Change Flag		
Usage Example	**Status**	**Default**	**Binary Mode Opcode Hex (Decimal)**
BR <lbl/addr>, PCHG BR <lbl/addr>, !PCHG PRINT PCHG	PCHG = FALSE (0): Axis stationary. PCHG = TRUE (1): Axis is changing position.	FALSE (0)	D9h (217)

Notes

This read only status flag indicates whether or not the axis is trying to obtain a specified position.

This flag becomes TRUE when the axis is moving in a profile motion. It is FALSE when the axis is moving in a jog or slew motion and becomes FALSE after the specified position has been exceeded in a MOVA or MOVR instruction with mode = 1. When the motor is moving in jog or slew motion or after the position has been reached during a MOVA or MOVR instruction with mode = 1, MVG is TRUE.

See the example for MOVA where HOLD is used to wait until PCHG becomes FALSE before starting the second move in the profile.

PFMT Setup Variable	Print Format Variable		
Usage Example	**Parameters**	**Default**	**Binary Mode Opcode Hex (Decimal)**
PFMT=<num1>, <num2>, <param>	<num1>: Number of digits before the decimal (0 - 16). <num2>: Number of digits after the decimal (0 - 16). <param>=0: Spaces as placeholders. <param>=1: Zeros as placeholders. <param>=2: No padding.	10, 3, 2	93h (147)

Notes

The PFMT variable specifies the print format for numeric values.

There are three parameters with PFMT. The first specifies how many significant digits there will be before the decimal. The second specifies how many significant digits there will be after the decimal. And the third specifies the type of padding. Blank or 0 specifies padding with spaces, 1 specifies padding with zeros, and 2 specifies no padding.

There will be a total of 16 digits displayed so, if there are 10 digits specified to the left of the decimal, there can be at most 6 specified to the right.

Related Commands

PRINT, PRINT1, PRINT2

PGM Immediate Mode Instruction	Enter/Exit Program Mode Instruction
Usage Example	**Binary Mode Opcode Hex (Decimal)**
PGM <addr> (Enter program mode) PGM (Exit Program mode)	49h (73)

Notes

When starting program mode, you must specify at what address to enter the program instructions in the program space. Simply type “PGM” again when you have finished entering your program commands to go back to immediate mode.

While in program mode, blank lines are accepted as are tab characters. This allows the user to format a text file with a user for readability, and then download the program to the LYNX/MicroLYNX by transferring the text file in a program such as HyperTerminal. The example given below could be stored in a text file and downloaded. The lines preceded by an apostrophe (‘) are comments and will be ignored by the LYNX/MicroLYNX Product. When the program is listed, the tabs and blank lines will not show, but they are accepted by the control module for input.

PGM Keyword	Retrieve Program Keyword	
Usage Example		**Binary Mode Opcode Hex (Decimal)**
GET PGM		94h (148)

Notes Used with GET to signify that all the program space should be retrieved from nonvolatile memory (NVM).

Related Commands GET

PME Setup Flag	Position Maintenance Enable/Disable Flag		
Usage Example	**Status**	**Default**	**Binary Mode Opcode Hex (Decimal)**
PME=<flg>	<flg> = FALSE (0): Disabled. <flg> = TRUE (1): Enabled.	FALSE (0)	DAh (218)

Notes Specifies whether the position maintenance function, which maintains position within a specified deadband, is enabled (1) or disabled (0). The default setting is (0) disabled. In order for position maintenance to be performed, the Encoder enable flag (EE) must also be set to TRUE (1).

Related Commands EE, EDB

PMHCC Setup Variable	Position Maintenance Holding Current Change Variable			
Usage Example	**Unit**	**Range**	**Default**	**Binary Mode Opcode Hex (Decimal)**
PMHCC=<percent>	Percent	0 to 100	0	95h (149)

Notes This variable specifies the amount of current required to maintain position when position maintenance is enabled.

The value for PMHCC is a percentage ranging from 0% to 100%, limited by MRC. If Position Maintenance is active, the value of PMHCC will be added to MHC until MRC is reached. Thus, if MHC is set to 15%, and MRC is set to 50% then the effective range for PMHCC will be 0 - 35%.

Example:

PMHCC is active and a value of 2% is programmed. When motion stops, MHC which is set at 15% will be active and Position Maintenance will monitor any movement. Movement may be caused by force on the axis i.e. a vertical slide with improper counter balancing. If movement is detected, the position will be corrected and PMHCC will add 2% to the MHC changing it from 15% to 17%. If movement is still detected, PMHCC will add another 2% to MHC changing it to 19%. This will continue until MHC maintains position or until MRC is reached.

Related Commands EE, EDB, PME, PMV, MUNIT, EUNIT

PMV Setup Variable	Position Maintenance Velocity Variable			
Usage Example	**Unit**	**Range**	**Default**	**Binary Mode Opcode Hex (Decimal)**
PMV=<speed>	User Units per second	±.0000000000000001 to ±9,999,999,999,999,999	10240.000	96h (150)

Notes Velocity to be used during position maintenance repositioning. If EUNIT has been set, then the value of PMV should be specified in user units. Otherwise, the value is simply specified in Microsteps per second.

Related Commands EE, EDB, PME, MUNIT, EUNIT

POS Variable	Axis Position Variable			
Usage Example	**Unit**	**Range**	**Default**	**Binary Mode Opcode Hex (Decimal)**
POS=<±position> PRINT POS BR <lbl/addr>, POS=<±position>	User Units	±.0000000000000001 to ±9,999,999,999,999,999	0.000	97h (151)

Notes Contains the current position of the axis in user units. If the encoder is disabled, the POS register contains the scaled information that has been sent to the drive. In other words, POS = CTR1/MUNIT. In this case, if the user changes POS, CTR1 is also modified.
If the encoder is enabled, the POS register contains the scale information that has been seen at the encoder. In other words, POS = CTR2/EUNIT. In this case, if the user changes POS, CTR1 and CTR2 are both modified.

Modifying POS in essence changes the frame of reference for the axis. POS will probably be set once during system set up to reference or “home” the system.

Related Commands CTR1, CTR2, EE, MUNIT, EUNIT, POSCAP

POSCAP Read Only Variable	Axis Position At Time Of Trip Variable			
Usage Example	**Unit**	**Response**	**Default**	**Binary Mode Opcode Hex (Decimal)**
PRINT POSCAP	User Units	± Position	0.000	8Bh (139)

Notes

The POSCAP variable is a read only variable that captures the value of POS when a trip is encountered.

EE=0: POSACP is active to CTR1
EE=1: POSCAP is active to CTR2

Syntax Example

This example demonstrates how the POSCAP Variable captures the Position using a TRIP Input when the Registration or Index mark on the encoder is encountered during a SLEW command.

```
MUNIT = 51200                       'user units defined as revs
VM = 30                             'max velocity in revs per second
VI = VM/50                          'init. velocity
ACCL = 100                          'acceleration
DECL = ACCL                         'decelleration equals accelleration
MHC=MRC                             'motor hold current=runcurrent

IOS 21 = 0,1,1,0,0,0                'output definition for LED or relay
IOS 13 = 0,0,1,1,0,0                'gen purpose input defined
'***************************

VAR Label = 1                       'variable called Lable set to value of 1
VAR Speed = 3
'******** Program ********
PGM 1                               'start of program
       LBL STARTUP                  'label used for executing on power up
       TI1 = 13, Mark               'set input trip to go to lable called
                                    'Mark

       LBL Go
              POS = 0               'sets position equao to zero
              SLEW Speed            'slew at the value of Speed
              VM = Speed
              HOLD 1                'hold until max vel is reached
              TIE1 = 1              'enables input trip
       LBL Idle1                    'just a lable
              BR Idle1, MVG         'branch to idle1 if motor is moving
              IO 21 = 1             'sets output true
              DELAY 3000            'delay 3 seconds
              IO 21 = 0             'sets output off
              BR Go                 'branch to the label Go
END
       LBL Mark
              MOVA poscap + label   'move absolute to position captured on
                                    'trip plus 1
              HOLD 2                'hold prog execution till move completes
              RET                   'return to subsequent line from where
                                    'trip occurred
  PGM                               'end of program space
```

Related Commands POS, TIx, TIEx, TPx, TPEx, TTx, TTEx, TTRx, TVx, TVEx

PRINT Immediate/Program Instruction	Print Instruction
Usage Example	**Binary Mode Opcode Hex (Decimal)**
PRINT <"text"> PRINT <var/flg> PRINT <"text">,<var/flg>	4Ah (74)

Notes

This instruction is used to output text and parameter value(s) to the host PC. Text should be enclosed in quotation marks while parameters (variables and flags) should not. Text strings and parameters which are to be output by the same PRINT instruction should be separated by commas. The information being output is followed by a carriage return and line feed unless a semicolon (;) is included at the end of the PRINT instruction to indicate that the cursor should remain on the same line. This is useful when the PRINT instruction is being used to output instructions preceding an INPUT instruction.

The DISP instruction may effect how the data is printed. In addition, the PFMT variable will determine the representation of numerical data.

Note: A delay time between print commands to the MicroLYNX must be considered to allow the MicroLYNX time to interpret a command and answer the host before a subsequent command can be sent.

There are several control characters that can be embedded in the print text:

\b	Causes the cursor to backspace one character.
\c	Embeds a Ctrl-C into the text string.
\e	Embeds an ESC character into the text string to allow ANSI video escape sequences.
\g	Causes the terminal to sound the bell.
\n	Causes a line feed with no carriage return.
\r	Causes a carriage return with no line feed to allow overwriting of the same line.
\t	Embeds a Tab in the text string.

NOTE: These control characters MUST be lower case!

Syntax Example

This example will print the velocity and position information for the user's review.

```
PRINT "Velocity = ", VEL, "   Position = ", POS
```

The following example will request that the user input information into a variable. The cursor will remain on the same line for the user to input the data.

```
VAR TURNS                   'Declare user variable "TURNS"
PGM 100                     'Start program at address 100
       LBL SAMPLE           'Label program "SAMPLE"
       PRINT "Specify the number of turns: ";
       INPUT TURNS          'Request user input for TURNS
END                         'End program
PGM                         'Return to immediate mode
```

Related Commands

DISP, INPUT, INPUT1, INPUT2, PFMT, PRINT1, PRINT2

PRINT1 Immediate/Program Instruction	Print to LYNX COMM1 Instruction	
Usage Example		**Binary Mode Opcode Hex (Decimal)**
PRINT1 <"text"> PRINT1 <var/flg> PRINT1 <"text">,<var/flg>		59h (89)

Notes This is an enhancement of the PRINT instruction in that it will only output the print string to LYNX/MicroLYNX COMM 1, otherwise it operates the same as the PRINT instruction.

Related Commands DISP, INPUT, INPUT1, INPUT2, PFMT, PRINT, PRINT2

PRINT2 Immediate/Program Instruction	Print to LYNX COMM2 Instruction	
Usage Example		**Binary Mode Opcode Hex (Decimal)**
PRINT2 <"text"> PRINT2 <var/flg> PRINT2 <"text">,<var/flg>		5Ah (90)

Notes This is an enhancement of the PRINT instruction in that it will only output the print string to LYNX/MicroLYNX COMM 2, otherwise it operates the same as the PRINT instruction.

Related Commands DISP, INPUT, INPUT1, INPUT2, PFMT, PRINT, PRINT1

PRMPT Setup Variable	Specify Prompt Character Variable			
Usage Example	**Unit**	**Range**	**Default**	**Binary Mode Opcode Hex (Decimal)**
PRMPT=<char/ascii>	Character or ASCII decimal value	32 to 254	> (ASCII 62)	98h (152)

Notes Specifies the character that is used by the LYNX/MicroLYNX Product for a prompt. Valid characters are ASCII characters represented by decimal values 32 – 254. (See ASCII table in Appendix A)

QUED Setup Flag	Queue LYNX Controller Flag		
Usage Example	**Status**	**Default**	**Binary Mode Opcode Hex (Decimal)**
QUED=<flg>	<flg> = FALSE (0): Disabled. <flg> = TRUE (1): Enabled.	FALSE (0)	DBh (219)

Notes This flag, when TRUE (1), will enable LYNX/MicroLYNX nodes in a PARTY system to be able to receive broadcast commands. A queued node (one with QUED = 1) will respond to instructions addressed to "^". This in effect allows the host PC to broadcast instructions to multiple nodes in the system.

Related Commands PARTY

RATIO Setup Variable	Ratio Mode Variable		
Usage Example	**Range**	**Default**	**Binary Mode Opcode Hex (Decimal)**
RATIO=<num>	–2≤ num <2	1.000	99h (153)

Notes The RATIO variable is used when one or more secondary drives is following the primary drive. This is done by setting the ratio option of IOS for one or more high speed output pairs to TRUE (1) and then setting RATIOE to TRUE (1). The clock driving the secondary drive(s) will be ratioed to the one driving the primary drive by the RATIO specified.

I/O lines 11 and 12 typically will be used for the primary. I/O lines 13 and 14 can be used to ratio other external drives as well. This would be done by setting the lines up as clock outputs with the ratio option of the IOS set to TRUE (1).

Syntax Example In the following example we will set the secondary axis (in this case CLK3) to follow the primary axis (CLK1) at a ratio of ½. **NOTE: A Differential Digital I/O module is required to perform this function. (Or a Combination I/O module using I/O line pairs 13 and 14 to control the secondary axis.)**

```
IOS 15 = 5,1,1,0,2,1        'Set Diff I/O channel 15 to ratio
IOS 16 = 6,1,1,0,2,1        'Set Diff I/O channel 16 to ratio
RATIO = .5                  'Set ratio to one half
RATIOE = 1                  'Set ratio mode enable flag to true
```

Related Commands IOS, RATIOE, RATIOW

RATIOE Setup Flag	Ratio Mode Enable Flag		
Usage Example	**Status**	**Default**	**Binary Mode Opcode Hex (Decimal)**
RATIO=<flg>	<flg> = FALSE (0): Disabled <flg> = TRUE (1): Enabled	FALSE (0)	DCh (220)

Notes This flag, when TRUE (1), will enable ratio mode operation. Although setting a parameter of the IOS variable specifies ratio mode, this flag acts as a master enable of the mode. This allows the user to enable and disable the function without changing the I/O setup. In addition, if multiple drives are being ratioed, this allows them to be started simultaneously.

Related Commands IOS, RATIO, RATIOW

RATIOW Setup Variable	Ratio Mode Pulse Width Variable			
Usage Example	**Parameters**	**Range**	**Default**	**Binary Mode Opcode Hex (Decimal)**
RATIOW=<num>	<num> = 0: Square wave. <num> = 1 - 254: Pulses in increments of 50ns.	0 - 254	0	9Ah (154)

Notes Pulse width for the step clock of the secondary channel(s) being used to drive the motor(s) in ratio mode. It should be noted that if a square wave pulse is selected here, the ratio will be ½ that specified. For instance, if a ratio of 1 is specified and RATIOW is set to 0, the ratio will actually be ½. Thus, if a square wave pulse is desired, the true range of ratio is -1 <= RATIO < 1.

Related Commands RATIOE, RATIO

RES Immediate Mode Instruction	Resume Program Execution Instruction
Usage Example	**Binary Mode Opcode Hex (Decimal)**
RES	4Ch (76)

Notes Resume the program and, if necessary, motion that was suspended by a PAUS instruction. The program is always resumed, but the motion may or may not be resumed depending on the value of PAUSM at the time the PAUS instruction was issued.

Related Commands PAUS, PAUSM

RET Program Mode Instruction	Return From Subroutine Instruction	
Usage Example		**Binary Mode Opcode Hex (Decimal)**
RET		4Dh (77)

Notes A RET statement is required at the end of the subroutine executed by a CALL instruction.

Syntax Example

```
VAR VAL=0                    'Declare user variable "VAL", set to 0
PGM 100                      'Start Program at address 100
       LBL MAIN_PRG          'Label program "MAIN_PRG"
       MOVR 51200            'Move relative 51,200 user units
       HOLD 2                'Suspend program until motion completes
       CALL SUB_ROUT,VAL=1   'Call subroutine "SUB_ROUT" when VAL=1
       BR MAIN_PRG           'Unconditional branch to MAIN_PRG
       LBL SUB_ROUT          'Declare subroutine SUB_ROUT
       MOVR 51200*5          'Move relative 51,200 X 5 user units
       HOLD 2                'Suspend program until motion completes
       RET                   'Return from subroutine
END                          'End program
PGM                          'Return to immediate mode
```

Related Commands CALL

RTFD Program Mode Instruction	Return To Factory Defaults	
Usage Example		**Binary Mode Opcode Hex (Decimal)**
RTFD		5Bh (91)

Description Returns all variables and flags to their factory defaults. It also clears the User program space and removes all User declaired variables and flags.

Note The last saved settings will return after a power cycle or CTRL^C unless SAVE is executed before power down or CTRL^C.

RUN Program Mode Instruction	Run Background Task Instruction	
Usage Example		**Binary Mode Opcode Hex (Decimal)**
RUN <lbl/addr>		4Eh (78)

Notes

The RUN instruction starts a background task to be run at a specified address. When the background task is started, the foreground and background task both execute sharing the LYNX/MicroLYNX Product's processor. The background task runs until a RET or END instruction is reached or until the end of code space is reached. It is good practice to end the task using the RET or END instruction.

Note that only one background task may be executing at any one time. If you execute a second RUN instruction before the first one has completed, unexpected results will occur.

Syntax Example

The following code sample will run a background task that will enable or disable an output based on the position of the motor while a foreground task is indexing the motor. In this example assume a half/full step driver set to full step driving a 1.8° stepping motor. When executed, the motor will move 1 revolution, set the output 31, move back to position 0, clear the output, then repeat.

The Foreground Program:

```
PGM 10                      'Enter program at line 10
LBL TST_PGM                 'Name the program TST_PGM
       MUNIT = 200          'Set MUNIT so that 200 units = 1 Revolution
       IOS 21 = 0,1         'Set I/O line 21 to a user defined output
       POS = 0              'Set the position to 0
       RUN BACK             'Run the background program labeled BACK
LBL LOOP                    'Define Sub Loop
       MOVA 200             'Index to Absolute Position 200
       HOLD 2               'Suspend Prog. execution until move completes
       DELAY 2000           'Delay 2 seconds
       MOVA 0               'Index to Absolute Position 0
       HOLD 2               'Suspend Prog. execution until move completes
       DELAY 2000           'Delay 2 seconds
       BR LOOP              'Unconditional Branch to Sub LOOP
END
PGM
```

The Background Program:

```
PGM 200
LBL BACK                    'Define background task BACK
       IO 21 = 0            'Set I/O 21 to 0
LBL FULL                    'Declare subroutine FULL
       BR FULL, POS = 200   'Loop to sub FULL until POS = 200
       IO 21 = 1            'Set I/O 21 to 1
       DELAY 4              'Delay Prog. execution 4 msec
LBL ZERO                    'Declare subroutine ZERO
       BR ZERO, POS = 0     'Loop to sub ZERO until POS = 0
       IO 21 = 0            'Clear I/O 21
       DELAY 2              'Delay Prog. execution 4 msec
       BR BACK              'Unconditional branch to BACK
END
PGM
```

Related Commands RET, END, BKGD, BKGDA

SAVE Immediate/Program Instruction	Save Instruction	
Usage Example		**Binary Mode Opcode Hex (Decimal)**
SAVE		4Fh (79)

Notes

Saves all variables, flags and programs currently in working memory (RAM) to nonvolatile memory (NVM). The previous values in NVM are completely overwritten with the new values. If necessary, the user can get back to factory default values using the IP instruction.

When the user modifies variables, flags and program space, they are changed in working memory (RAM) only. If the SAVE instruction is not executed before power is removed from the control module, all modifications to variables, flags and programs since the last SAVE will be lost.

Related Commands IP, SET, PGM

SER Read Only Variable	Serial Number Variable	
Usage Example		**Binary Mode Opcode Hex (Decimal)**
PRINT SER		9Bh (155)

Notes

This read only variable can be used to display the LYNX/MicroLYNX Product's serial number. The value set is at the factory

SET Immediate/Program Instruction	Set Variable Or Flag Instruction	
Usage Example		**Binary Mode Opcode Hex (Decimal)**
SET <var/flg> =<val>		50h (80)

Notes

Sets a variable or flag to a specified value. SET is an optional command. It can be left off when assigning a value to a flag or variable. For instance, if the user wants to SET ACCL to 5, this can be done using the SET instruction (SET ACCL = 5) or the instruction can be implied (ACCL = 5).

Syntax Example

In the below syntax example you will notice that we did not type the SET command in front of the variable name. In the LYNX/MicroLYNX software, the SET is assumed when a variable or flag value is defined. Whenever a program is uploaded from the LYNX/MicroLYNX to a text file or LISTed to the terminal screen, the SET instruction will appear in front of the variables and/or flags that have been defined within the program.

```
RATIOW = 200  'Set ratio pulse width to 10µs
```

SLEW Immediate/Program Instruction	Slew Motor At Constant Velocity Instruction		
Usage Example	**Units**	**Modes**	**Binary Mode Opcode Hex (Decimal)**
SLEW <±speed>, <mode>	<±speed> = User Units/sec	<mode> = 0: Use acceleration ramp. <mode> = 1: Do not use acceleration ramp.	51h (81)

Notes When using the SLEW instruction, the user must at least give a velocity (sign indicates direction) at which the motor should run. The slew velocity will be based upon the value of MUNIT. In addition, the user can specify whether or not the acceleration ramp should be used to get to speed. If the second parameter is not specified or is given as 0, the acceleration ramp should be used to get to speed. If it is specified as 1, the slew rate should be reached by a step function without acceleration.

Syntax Example

```
SLEW .5, 1    'Slew the motor .5 user units/sec w/no acceleration ramp
```

Related Commands ACCL

SMOVR Variable	Save Relative Move			
Usage Example	**Parameters**		**Default**	**Binary Mode Opcode Hex (Decimal)**
SMOVR <loc> <dist>	<loc> = 0-127 (Linear) <loc> = 0-1 (S-Curve) <distance> = MOVR Values		Empty	8Dh (141)

Description Saves the settings for a relative move of distance <dist> at SMOVR location <loc>. It is possible to save 128 (ACLT=1 and DCLT=1) relative moves (MOVR). If either ALCT or DLCT is other than 1, the move is considered an “S-Curve” move and it will take up 41 linear move locations.

Note The SMOVR table takes up User program space.

SSTP Immediate/Program Instruction	Soft Stop Instruction		
Usage Example	**Modes**	**Default**	**Binary Mode Opcode Hex (Decimal)**
SSTP <mode>	<mode> = 0: Stop motion only, program continues to execute. <mode> = 1: Stop both motion and program.	Mode 0	52h (82)

Notes

Stop the current motion using the specified deceleration profile and optionally stop the program that is currently running. If SSTP is issued with no parameter or 0, only the motion is terminated. If, however, SSTP is issued with a parameter of 1, the motion and program are both terminated.

Syntax Example

The examples below illustrate the SSTP instruction being used in both modes:

MODE 0

```
PGM 100              'Start program at address 100
       LBL TST       'Label the program "TST"
       SLEW 100000   'Slew th motor at 100000 user units/sec
       DELAY 3000    'Delay 3 seconds
       SSTP 0        'Soft stop motion, continue executing program
       DELAY 2000    'Delay 2 seconds
       BR TST        'Unconditional branch to beginning of program
END
PGM
```

MODE 1

```
PGM 100              'Start program at address 100
       LBL TST       'Label the program "TST"
       SLEW 100000   'Slew th motor at 100000 user units/sec
       DELAY 3000    'Delay 3 seconds
       SSTP 1        'Soft stop motion, stop program
       DELAY 2000    'Delay 2 seconds
       BR TST        'Unconditional branch to beginning of program
END
PGM
```

STALL Status Flag	Axis Stalled Indicator Flag		
Usage Example	**Status**	**Default**	**Binary Mode Opcode Hex (Decimal)**
BR <lbl/addr>, STALL BR <lbl/addr>, !STALL PRINT STALL	STALL = FALSE (0): Not stalled. STALL = TRUE (1): Axis stalled.	FALSE (0)	DEh (222)

Notes

Status flag that indicates the motor has stalled. If the encoder is enabled (EE = 1) and the encoder “falls behind” the motor more than the specified factor, STLF, a STALL is indicated. If STLDE is also enabled (1), then the motor will be stopped when a STALL is detected. To clear the Stall Flag set Stall = Ø.

Related Commands

EE, STLDE, STLF

STATS Keyword	Retrieve Status Flags Keyword	
Usage Example		**Binary Mode Opcode Hex (Decimal)**
PRINT STATS		9Ch (156)

Notes Used with the PRINT instruction to print values of the status flags only. The status flags are ACL, BKGD, BSY, DCL, ERR, HELD, MVG, PAUSD, PCHG, STALL, STK, VCHG.

Related Commands PRINT

STEPW Setup Variable	Step Pulse Width Variable			
Usage Example	**Parameters**	**Range**	**Default**	**Binary Mode Opcode Hex (Decimal)**
STEPW=<num>	<num> = 0: Square wave. <num> = 1 - 254: Pulses in increments of 50ns.	0 - 254	0	9Dh (157)

Notes Step pulse width for the primary axis.

STK Read Only Status Flag	Subroutine Stack Fault Flag		
Usage Example	**Status**	**Default**	**Binary Mode Opcode Hex (Decimal)**
BR <lbl/addr>, STK BR <lbl/addr>, !STK PRINT STK	STK = FALSE (0): No fault. STK = TRUE (1): Stack overflow or underflow fault.	FALSE (0)	DFh (223)

Notes This is a read only flag that indicates a stack overflow or underflow.

STLDE Setup Flag	Stall Detect Enable/Disable Flag		
Usage Example	**Status**	**Default**	**Binary Mode Opcode Hex (Decimal)**
STLDE=<flg>	<flg> = FALSE (0): Disable. <flg> = TRUE (1): Enable.	FALSE (0)	E0h (224)

Notes If the encoder is enabled (EE = 1) and the encoder “falls behind” the motor more than the specified factor, STLF, a STALL is indicated. If STLDE is also enabled (1), then the motor will be stopped when a STALL is detected. EE is the master encoder enable - unless it is TRUE (1), nothing happens when STLDE becomes TRUE (1).

Related Commands EE, STALL, STLF, STLDM

STLDM Setup Variable	Stall Detection Mode Variable		
Usage Example	**Parameters**	**Default**	**Binary Mode Opcode Hex (Decimal)**
STLDM=<mode>	<mode> = 0: Stop motor when detecting a stall. <mode> = 1: Do not stop motor when detecting a stall.	Mode 0	89h (137)

Notes This variable sets the mode for stall detection.

Related Commands EE, STALL, STLF, STLDE

STLF Setup Variable	Stall Factor Variable			
Usage Example	**Unit**	**Range**	**Default**	**Binary Mode Opcode Hex (Decimal)**
STLF=<num>	User Units	0.0000000000000001 to 9,999,999,999,999,999	10.000	9Eh (158)

Notes If the encoder is enabled (EE = 1) and the encoder "falls behind" the motor more than the specified factor, a STALL is indicated. If STLDE is also enabled (1) and if STDLM = 0, then the motor will be stopped when a STALL is detected.

Related Commands EE, STALL, STLDE

TI1, TI2, TI3, TI4 Setup Variables	Trip On Input Variables **FORMERLY IT<x>**		
Usage Example	**Parameters**	**Default**	**Binary Mode Opcodes Hex (Decimal)**
TI<x>=<input>, <lbl/addr>, <output>	<x> = 1 - 4 <input> = Input line used for trip. <lbl/addr> = Subroutine invoked on trip. <output> = Output set TRUE on trip.	0, 0, 0	TI1 = 7Fh (127) TI2 = 80h (128) TI3 = 81h (129) TI4 = 82h (130)

Notes Sets up an input event (trip) for the specified input. There are three parameters for the TI variables. The first specifies which input line should cause the event. The second specifies the address of the subroutine that should be executed when the input is seen. The third optional parameter specifies the output line to be set TRUE when the input trip is seen.

The input used should be a user input or one of the limit or home inputs. Note that the GO input automatically looks for a subroutine at address 1 and if there is valid code there it starts execution from address 1.

The TIE flag for the appropriate event number must be enabled for the event to be recognized.

Related Commands TIE1, TIE2, TIE3, TIE4, IOS

TIE1, TIE2, TIE3, TIE4 Setup Flags	Trip On Input Enable/Disable Flags **FORMERLY ITE<x>**		
Usage Example	**Status**	**Default**	**Binary Mode Opcode Hex (Decimal)**
TIE<x>=<flg>	<x> = 1 - 4 <flg> = FALSE (0): Disable. <flg> = TRUE (1): Enable.	FALSE (0)	TIE1 = CCh (204) TIE2 = CDh (205) TIE3 = CEh (206) TIE4 = CFh (207)

Notes Enables/Disables the corresponding trip on input. Note the input trips are disabled when the LYNX/MicroLYNX encounters an END statement. Each trip is automatically disabled when it is detected.

NOTE: Initially, a trip may be pending.
To clear an edge I/O input trip, read the I/O.

Related Commands TI1, TI2, TI3, TI4

TP1, TP2, TP3, TP4 Setup Variables	Trip On Position Variables		
Usage Example	**Parameters**	**Default**	**Binary Mode Opcodes Hex (Decimal)**
TP<x>=<pos>, <lbl/addr>, <output>	<x> = 1 - 4 <pos> = ± Position in user units. <lbl/addr> = Subroutine invoked on trip. <output> = Output set TRUE on trip.	0.000, 0, 0	TP1 = A3h (163) TP2 = A4h (164) TP3 = A5h (165) TP4 = A6h (166)

Notes There are three parameters for the TPx variables. The first specifies the position at which the specified subroutine should be executed. The second specifies the address of the subroutine that should be executed when the position is reached. The third optional parameter specifies an output to be set TRUE when the trip is reached.

It should be noted that if EE is TRUE (1), in order to use TP3 and TP4 as encoder counts, the ENC input must be hard-wired to the EVENT input. The <pos> range is ±.0000000000000001 to ±9,999,999,999,999,999 user units based on the value of MUNIT.

Related Commands TPE1, TPE2, TPE3, TPE4, MUNIT

TPE1, TPE2, TPE3, TPE4 Setup Flags	Trip On Position Enable/Disable Flags		
Usage Example	**Status**	**Default**	**Binary Mode Opcode Hex (Decimal)**
TPE<x>=<flg>	<x> = 1 - 4 <flg> = FALSE (0): Disable. <flg> = TRUE (1): Enable.	FALSE (0)	TPE1 = E9h (233) TPE2 = EAh (234) TPE3 = EBh (235) TPE4 = ECh (236)

Notes These flags enable/disable the corresponding position event (trip).

Related Commands TP1, TP2, TP3, TP4

TT1, TT2, TT3, TT4 Setup Variables	Trip On Timer Variables FORMERLY TI<x>		
Usage Example	**Parameters**	**Default**	**Binary Mode Opcodes Hex (Decimal)**
TT<x>=<time>, <lbl/addr>, <output>	<x> = 1 - 4 <time> = Time in milliseconds (0 - 65,535). <lbl/addr> = Subroutine invoked on trip. <output> = Output set TRUE on trip.	0, 0, 0	TT1 = 9Fh (159) TT2 = A0h (160) TT3 = A1h (161) TT4 = A2h (162)

Notes There are three parameters for the TTx variables. The first specifies the period or time in milliseconds which should elapse before the event occurs. The second specifies the address of the subroutine that should be executed when the timer expires. The third optional parameter specifies an output to be set TRUE when the trip is reached.

TTRx specifies whether the associated event should be a one shot or repeated every time the specified period expires. TTEx must be enabled for the associated event to be recognized.

Related Commands TTE1, TTE2, TTE3, TTE4, TTR1, TTR2, TTR3, TTR4,

TTE1, TTE2, TTE3, TTE4 Setup Flags	Trip On Timer Enable/Disable Flags FORMERLY TIE<x>		
Usage Example	**Status**	**Default**	**Binary Mode Opcode Hex (Decimal)**
TTE<x>=<flg>	<x> = 1 - 4 <flg> = FALSE (0): Disable. <flg> = TRUE (1): Enable.	FALSE (0)	TTE1 = E1h (225) TTE2 = E2h (226) TTE3 = E3h (227) TTE4 = E4h (228)

Notes These flags enable the corresponding timer event (trip).

Related Commands TT1, TT2, TT3, TT4

TTR1, TTR2, TTR3, TTR4 Setup Flags	Trip On Timer Reload Flags FORMERLY TIR<x>		
Usage Example	**Status**	**Default**	**Binary Mode Opcode Hex (Decimal)**
TTR<x>=<flg>	<x> = 1 - 4 <flg> = FALSE (0): Do not repeat timer event. <flg> = TRUE (1): Repeat timer event.	FALSE (0)	TTR1 = E5h (229) TTR2 = E6h (230) TTR3 = E7h (231) TTR4 = E8h (232)

Notes TIRx specifies whether the associated event should be a one shot or repeated every time the specified period expires.

Related Commands TT1, TT2, TT3, TT4

TV Setup Variables	Trip On Velocity Variable FORMERLY VT		
Usage Example	**Parameters**	**Default**	**Binary Mode Opcodes Hex (Decimal)**
TV=<velocity>, <lbl/addr>, <output>	<velocity> = Velocity in user units/sec. <lbl/addr> = Subroutine invoked on trip. <output> = Output set TRUE on trip.	0.000, 0, 0	ACh (172)

Notes

There are three parameters for the VT variable. The first specifies the velocity at which the specified subroutine should be executed. The second specifies the address of the subroutine that should be executed when the velocity is reached. The optional third parameter specifies and output to be set TRUE when the trip is reached.

Once the trip has been set up, the specified subroutine is run when the velocity, VEL, passes through the velocity specified by vel. In other words, the subroutine will be called when the motor accelerates through the velocity and then again when it decelerates through it.

Note that the range of <velocity> is ±.0000000000000001 to ±9,999,999,999,999,999 user units based on the value of MUNIT.

Related Commands TVE, MUNIT

TVE Setup Flags	Trip On Velocity Enable/Disable Flag FORMERLY VTE		
Usage Example	**Status**	**Default**	**Binary Mode Opcode Hex (Decimal)**
TVE=<flg>	<flg> = FALSE (0): Disabled. <flg> = TRUE (1): Enabled.	FALSE (0)	EEh (238)

Notes This flags enables the corresponding velocity event (trip).

Related Commands TV

UFLGS Keyword	Report User Flags Keyword	
Usage Example		**Binary Mode Opcode Hex (Decimal)**
PRINT UFLGS		A7h (167)

Notes

This keyword is used with the PRINT instruction to report the state of all the user-defined flags which were created using the FLG instruction.
Returns:

G + Logic State = Global
L + Logic State = Local

Related Commands FLG

ULBLS Keyword	Report User Labels Keyword	
Usage Example		**Binary Mode Opcode Hex (Decimal)**
PRINT ULBLS		A8h (168)

Notes This keyword is is used with the PRINT instruction to report all the user-defined labels which were created using the LBL instruction.

Related Commands LBL

UVARS Keyword	Report User Variables Keyword	
Usage Example		**Binary Mode Opcode Hex (Decimal)**
PRINT UVARS		A9h (169)

Notes This keyword is used with the PRINT instruction to report the state of all the user defined variables which were created using the VAR instruction.
Returns:

G + Logic State = Global
L + Logic State = Local

Related Commands VAR

VAE Setup Flag	Velocity to Analog Enable Flag		
Usage Example	**Function**	**Default**	**Binary Mode Opcode Hex (Decimal)**
VAE=<flg>	<flg> = FALSE (0): Disabled. <flg> = TRUE (1): Enabled.	FALSE (0)	D4h (212)

Description Enables the velocity or position to be converted to an Analog Voltage output channel selected for this function.

Related Commands DAS

VAR Immediate/Program Instruction	Define User Variable Instruction	
Usage Example	**Parameters**	**Binary Mode Opcode Hex (Decimal)**
VAR <name> = <num>	<name> = 1 - 8 Alphanumeric characters and underscore (_). <num> = Some number.	54h (84)

Notes

Defines a user variable that can contain numeric data. The name of the variable can be 1 to 8 alphanumeric characters in length. You may use the underscore (_) character in the name as well. The value of the variable can be initialized when it is defined. If it is not specifically initialized, it will have a value of 0 until it is set.

Variables can be “global” or “local”. A local variable is one that has been defined within a control module program and can not be changed in immediate mode. A global variable is defined outside a control module program and can be changed in immediate mode. It should be noted that a local variable is not static, but is erased and declared again whenever the program is executed.

Syntax Examples

Local

```
PGM 100                 'Start program at address 100
        LBL TST         'Label program TST
        VAR MY_VAR=1000 'Declare user variable MY_VAR, set to 1000 user units
        SLEW MY-VAR     'Slew the amount specified by MY-VAR
        BR TST          'Unconditional Branch to TST
END
PGM
```

Global

```
VAR MY_VAR=1000         'Declare user variable MY_VAR, set to 1000 user units
PGM 100                 'Start program at address 100
        LBL TST         'Label program TST
        SLEW MY-VAR     'Slew the amount specified by MY-VAR
        BR TST          'Unconditional Branch to TST
END
PGM
```

Related Commands UVARS

VARS Keyword	Variables Keyword	
Usage Example		**Binary Mode Opcode Hex (Decimal)**
PRINT VARS GET VARS IP VARS		AAh (170)

Notes

Used with the GET, IP and PRINT commands to specify that all variables should be retrieved from nonvolatile memory (NVM), set to their factory default values, or printed to the serial port, respectively. When used with the GET instruction, only system variable values are retrieved from NVM. When used with the IP instruction, only system variable values are set to the factory default parameters. In these cases, user defined variables are not affected. When used with the PRINT instruction, only variable values are echoed to the host computer.

Related Commands PRINT, IP, GET

VCHG Read Only Status Flag	Velocity Changing Flag		
Usage Example	**Status**	**Default**	**Binary Mode Opcode Hex (Decimal)**
BR <lbl/addr>, VCHG BR <lbl/addr>, !VCHG PRINT VCHG	VCHG = FALSE (0): Velocity constant. VCHG = TRUE (1): Velocity changing.	FALSE (0)	EDh (237)

Notes Read Only status flag indicates whether or not the axis is changing velocity. Will be TRUE (1) whenever the axis is accelerating or decelerating.

VEL Read Only Register Variable	Velocity Variable			
Usage Example	**Unit**	**Range**	**Default**	**Binary Mode Opcode Hex (Decimal)**
PRINT VEL BR <lbl/addr>, VEL=<num> CALL <sub>, VEL =<num>	User Units/Sec.	± .0000000000000001 to ± 9,999,999,999,999,999	0.000	A8h (168)

Notes Register which contains the actual velocity of the axis in user units per second.

Related Commands EUNIT, MUNIT

VER Read Only Variable	Software Version Variable	
Usage Example	**Response**	**Binary Mode Opcode Hex (Decimal)**
PRINT VER	VER<chipsel#>=<board addr>, <version#>	A9h (169)

Notes This is a read only variable which will be changed only when the software is upgraded by using the upgrader program. It will print the software version of the LYNX/MicroLYNX Control Module, and the version of any add-on modules in the system. This list will not display when using the PRINT ALL, or PRINT VARS instruction, or if the communications mode selected is binary.

VI Setup Variable	Initial Velocity Variable			
Usage Example	**Unit**	**Range**	**Default**	**Binary Mode Opcode Hex (Decimal)**
VI=<num>	User Units/Sec.	± .0000000000000001 to ± 9,999,999,999,999,999	102400.000	ADh (173)

Notes Initial velocity for the axis during a point-to-point motion. The factory default value is 102,400 Microsteps per second with a minimum value of 12,000 Microsteps per second when MUNIT = 1.

The initial velocity for a stepper should be set to avoid the low speed resonance frequency and must be set lower than the pull in torque of the motor.

Related Commands EUNIT, MUNIT

VM Setup Variable	Maximum Velocity Variable			
Usage Example	**Unit**	**Range**	**Default**	**Binary Mode Opcode Hex (Decimal)**
VM=<num>	User Units/Sec.	± .0000000000000001 to ± 9,999,999,999,999,999	768000.000	AEh (174)

Notes Maximum velocity for the axis during a point-to-point motion. The maximum velocity is the velocity that will be reached for any MOVA or MOVR, provided of course that the move is long enough for the axis to reach the velocity. When a motion occurs, the axis starts at velocity VI and accelerates using the specified acceleration profile until the velocity VM is reached.

Related Commands EUNIT, MUNIT

DECIMAL TO HEX TO ASCII CONVERTER

DEC	HEX	ASCII	KEY	DEC	HEX	ASCII	DEC	HEX	ASCII	DEC	HEX	ASCII
0	00	NUL	ctrl @	32	20	SP	64	40	@	96	60	`
1	01	SOH	ctrl A	33	21	!	65	41	A	97	61	a
2	02	STX	ctrl B	34	22	"	66	42	B	98	62	b
3	03	ETX	ctrl C	35	23	#	67	43	C	99	63	c
4	04	EOT	ctrl D	36	24	$	68	44	D	100	64	d
5	05	ENQ	ctrl E	37	25	%	69	45	E	101	65	e
6	06	ACK	ctrl F	38	26	&	70	46	F	102	66	f
7	07	BEL	ctrl G	39	27	'	71	47	G	103	67	g
8	08	BS	ctrl H	40	28	(	72	48	H	104	68	h
9	09	HT	ctrl I	41	29	)	73	49	I	105	69	i
10	0A	LF	ctrl J	42	2A	*	74	4A	J	106	6A	j
11	0B	VT	ctrl K	43	2B	+	75	4B	K	107	6B	k
12	0C	FF	ctrl L	44	2C	,	76	4C	L	108	6C	l
13	0D	CR	ctrl M	45	2D	-	77	4D	M	109	6D	m
14	0E	SO	ctrl N	46	2E	.	78	4E	N	110	6E	n
15	0F	SI	ctrl O	47	2F	/	79	4F	O	111	6F	o
16	10	DLE	ctrl P	48	30	0	80	50	P	112	70	p
17	11	DC1	ctrl Q	49	31	1	81	51	Q	113	71	q
18	12	DC2	ctrl R	50	32	2	82	52	R	114	72	r
19	13	DC3	ctrl S	51	33	3	83	53	S	115	73	s
20	14	DC4	ctrl T	52	34	4	84	54	T	116	74	t
21	15	NAK	ctrl U	53	35	5	85	55	U	117	75	u
22	16	SYN	ctrl V	54	36	6	86	56	V	118	76	v
23	17	ETB	ctrl W	55	37	7	87	57	W	119	77	w
24	18	CAN	ctrl X	56	38	8	88	58	X	120	78	x
25	19	EM	ctrl Y	57	39	9	89	59	Y	121	79	y
26	1A	SUB	ctrl Z	58	3A	:	90	5A	Z	122	7A	z
27	1B	ESC	ctrl [	59	3B	;	91	5B	[	123	7B	{
28	1C	FS	ctrl \	60	3C	<	92	5C	\	124	7C	|
29	1D	GS	ctrl]	61	3D	=	93	5D	]	125	7D	}
30	1E	RS	ctrl ^	62	3E	>	94	5E	^	126	7E	~
31	1F	US	ctrl _	63	3F	?	95	5F	_	127	7F	DEL

APPENDIX B

Error Table

0 NO ERROR

Hardware Errors

1018 IO MODULE NOT INSTALLED.
1019 LYNX/MicroLYNX CHECK SUM INCORRECT.
1020 ONLY ALLOWED IN IMMEDIATE MODE.
1100 FAULT/LIMIT DETECTED IN A CONNECTED DRIVE. (Power OV/OC; Phase OC; Drive OT
1101 FAULT IN DRIVE 1.
1105 DRIVE 1 FAULT AND TYPE CHANGED.
1109 DRIVE 1 MSEL COULD NOT BE SET.
1113 DRIVE 1 TYPE CHANGED, MSEL COULD NOT BE SET.
1117 DRIVE 1 FAULT, MSEL COULD NOT BE SET.
1121 DRIVE 1 FAULT, TYPE CHANGED, MSEL COULD NOT BE SET.
1125 HOLD IGNORED, MOTOR DISABLED.
1126 DRIVE 1 NOT AVAILABLE.
1130 DRIVE 1 TYPE CHANGED.
1134 ILLEGAL DRIVE NUMBER.
1135 DRIVE 1 IN LIMIT.
1139 TRIED TO SET MAC OR MRC TO LESS THAN ONE.
1201 SELECTED ANALOG BOARD NOT INSTALLED.
1202 ANALOG CHANNEL NUMBER NOT AVAILABLE.
1204 ANALOG OPTION NOT INSTALLED.
1205 ANALOG VALUE OUT OF RANGE, POSSIBLY DEFECTIVE BOARD.

I/O Errors

2001 FIOS FOUND NO (HOME) SWITCH.
2002 NOT IN FACTORY MODE.
2020 OUTPUT FAULT AT DIGITAL IO GROUP 20.
2021 OUTPUT FAULT AT DIGITAL IO LINE 21.
2022 OUTPUT FAULT AT DIGITAL IO LINE 22.
2023 OUTPUT FAULT AT DIGITAL IO LINE 23.
2024 OUTPUT FAULT AT DIGITAL IO LINE 24.
2025 OUTPUT FAULT AT DIGITAL IO LINE 25.
2026 OUTPUT FAULT AT DIGITAL IO LINE 26.
2030 OUTPUT FAULT AT DIGITAL IO GROUP 30.
2031 OUTPUT FAULT AT DIGITAL IO LINE 31.
2032 OUTPUT FAULT AT DIGITAL IO LINE 32.
2033 OUTPUT FAULT AT DIGITAL IO LINE 33.
2034 OUTPUT FAULT AT DIGITAL IO LINE 34.
2035 OUTPUT FAULT AT DIGITAL IO LINE 35.
2036 OUTPUT FAULT AT DIGITAL IO LINE 36.
2040 OUTPUT FAULT AT DIGITAL IO GROUP 40.
2041 OUTPUT FAULT AT DIGITAL IO LINE 41.
2042 OUTPUT FAULT AT DIGITAL IO LINE 42.
2043 OUTPUT FAULT AT DIGITAL IO LINE 43.
2044 OUTPUT FAULT AT DIGITAL IO LINE 44.
2045 OUTPUT FAULT AT DIGITAL IO LINE 45.
2046 OUTPUT FAULT AT DIGITAL IO LINE 46.
2050 OUTPUT FAULT AT DIGITAL IO GROUP 50.
2051 OUTPUT FAULT AT DIGITAL IO LINE 51.
2052 OUTPUT FAULT AT DIGITAL IO LINE 52.
2053 OUTPUT FAULT AT DIGITAL IO LINE 53.
2054 OUTPUT FAULT AT DIGITAL IO LINE 54.
2055 OUTPUT FAULT AT DIGITAL IO LINE 55.
2056 OUTPUT FAULT AT DIGITAL IO LINE 56.
2101 ANALOG RANGE NOT ALLOWED.
2102 ANALOG DESTINATION/SOURCE NOT ALLOWED.
2103 ANALOG DESTINATION/SOURCE ALREADY USED.

2104 INVALID ANALOG CHANNEL NUMBER.
2105 ANALOG LAW NOT ALLOWED.
2106 CAN'T ENABLE JOYSTICK WHILE IN MOTION OR CAN'T EXEC MOTION CMD WITH JOYSTICK ENABLED.
2200 CAN ERRORS: 1-6,11-16,21-26,31-36.

Clock Errors

3001 TRIED TO SET CLK TO NON CLOCK LINE OR WRONG LINE.
3002 CAN'T HAVE CLOCK TYPE APPLIED TO IT.
3003 CAN'T HAVE RATIO AND NO_CLK.
3004 CLK IO CAN'T BE SET FOR RATIO MODE.
3005 IN HALF-AXIS MODE.
3006 TRIED TO SET TO INPUT WHEN DRIVE CONNECTED.
3007 NO IO SET FOR INPUT + RATIO.

Syntax Errors

4001 INVALID IO NUMBER.
4002 TRIED TO WRITE GROUP TO NONUSER.
4003 TRIED TO WRITE TO A NON-USER LINE.
4004 TRIED TO WRITE TO AN INPUT.
4005 TRIED TO SET AN OUTPUT ONLY TO INPUT.
4006 TRIED TO SET AN INPUT ONLY TO OUTPUT.
4007 TRIED TO SET LINE TYPE TO LINE THAT CAN'T BE SET THAT WAY.
4008 NOT A VALID IO TYPE.
4009 IO TYPE SW. PREVIOUSLY DEFINED.
4010 FIND SW MUST BE SET AS INPUT.
4011 MORE THAN ONE IO SET FOR RATIO INPUT.
4012 ILLEGAL RUN/EXEC MODE.
4013 RECEIVED UNACCEPTIBLE COMMAND.
4014 ILLEGAL PAR IN "INPUT PAR" COMMAND.
4015 LABEL HAS TO BE TEXT.
4016 ILLEGAL DATA ENTERED IN PRINT FORMAT.
4017 NO DATA ENTERED, COMMAND IGNORED.
4018 ILLEGAL DRIVE NAME.
4019 ADDRESS DOESN'T POINT TO VALID INSTRUCTION.
4020 TRIED TO EXECUTE A BAD USER PROGRAM INSTRUCTION.
4021 ILLEGAL LINE NUMBER.
4022 MULTI LINE PRINTS NOT ALLOWED IN BINARY MODE.
4023 ILLEGAL HOLD TYPE.
4024 NOT ALLOWED IN IMMEDIATE MODE.
4025 AN INPUT IS ALREADY PENDING.
4026 SELECTED COMM, PORT2, CANNOT BE SEPERATELY SELECTED.
4027 LINE NUMBER NOT NEEDED.
4028 INP CANNOT BE SET.
4029 ARRAY POINTER TOO LARGE (ACTBL OR SMOVR).
4030 COMM TIMED OUT.
4031 SMOVR ELEMENT NOT PROGRAMMED.
4032 BINARY COMM CHECKSUM FAILED.

Variable/Flag Errors

5001 ILLEGAL VARIABLE ENTERED.
5002 ILLEGAL FLAG ENTERED.
5003 ILLEGAL FLAG OR VARIABLE ENTERED.
5004 NO EQUAL IN: SET VARIABLE TO VALUE.
5005 ILLEGAL CHARACTER FOLLOWS DECLARATION OF VARIABLE OR FLAG.
5006 UNDEFINED USER VAR OR FLG.
5007 TRIED TO REDEFINE LBL/VAR/FLG.
5008 TRIED TO REDEFINE GBL/LCL LBL/VAR/FLG.
5009 INSTRUCTION/VARIABLE/FLAG NOT IMPLEMENTED IN THIS VERSION.

5010 VALUE OF LBL/VAR/FLG CHANGED - WARNING.
5011 FLAG IS READ ONLY.
5012 VARIABLE IS READ ONLY.
5013 CAN ONLY INIT ALL, VARS, FLAGS.
5016 CAN'T SET MULTI VARIABLES, READ ONLY.

Motion Errors

6001 REACHED PLUS LIMIT SW.
6002 REACHED MINUS LIMIT SW.
6003 TIME NEEDED TO MAKE MOVE LESS THAN 200USEC.
6004 NO DISTANCE FOR MOVE.

Encoder Errors

7001 STALL DETECTED.
7002 IMPROPER RATIO OF MUNIT TO EUNIT.
7003 MOVED OUT OF DEADBAND.
7004 CAN'T FIND POSITION AT END OF MOVE.
7005 STALLED WHILE DOING POSITION MAINTENANCE
7006 STALLED WHILE DOING EE POSITION CORRECTION AT END OF MOVE.

NVM Errors

8001 LABEL AREA FULL.
8002 SAVE FAILED.
8003 TRIED TO TAKE FROM EMPTY STACK.
8004 DATA NOT IN NVM.
8005 TRIED TO OVER FILL FOREGROUND STACK.
8006 TRIED TO SAVE WHILE MOTION IN PROGRESS.
8007 TRIED TO OVER FILL BACKGROUND STACK.
8008 BAD SECTOR IN PAGE 0 OF FLASH.
8009 BAD SECTOR IN PAGE 1 OF FLASH.
8010 BAD SECTOR IN PAGE 2 OF FLASH.
8011 BAD SECTOR IN PAGE 3 OF FLASH.

Out Of Range Errors

9001 IO FILTER OUT OF RANGE.
9002 IO GROUP OUT OF RANGE.
9003 PROGRAM ADDRESS OUT OF RANGE.
9004 RATIO OUT OF RANGE.
9005 DATA OUT OF RANGE FOR VARIABLE.
9006 PULSE WIDTH OUT OF RANGE.
9007 TOO MANY DIGITS SPECIFIED IN PRINT FORMAT.
9008 SUM OF ID AND FD EXCEEDS MAX NUMBER OF DIGITS.
9009 VALUE MUST BE POSITIVE ONLY.
9010 VM IS SET LESS THAN OR EQUAL TO VI.
9011 VI IS SET BELOW MIN_VELOCITY.
9012 MOVE DISTANCE TOO SHORT FOR PRESENT DECEL RATE.
9013 JOG SPEED LESS THAN MIN_VELOCITY.
9014 ANALOG INPUT NOT ALLOWED FOR DATA.
9015 COMM PORT OUT OF RANGE.
9016 PROGRAM LOCKED, CLEAR PROGRAM OR RETURN TO FACTORY DEFAULTS.

APPENDIX C

Factory Defaults

Variables

ACCL	= 1000000.000
ACLT	= 1
ADS 1	= 1.000, 1, 1
ADS 2	= 1.000, 1, 1
ADS 3	= 1.000, 1, 1
ADS 4	= 1.000, 1, 1
AIN	= 0.000
BAUD 1	= 96
BAUD 2	= 96
BKGDA	= 0
BLM	= 0
BLSH	= 0.000
COMDT	= 0
CTR1	= 0
CTR2	= 0
CTR3	= 0
DAS 1	= 1.000, 1
DAS 2	= 1.000, 1
DAS 3	= 1.000, 1
DAS 4	= 1.000, 1
DCLT	= 1
DECL	= 1000000.000
DISP	= 0, 0, 0
DN	= "!"
DRVTP	= 2
ECHO 1	= 0
ECHO 2	= 0
EDB	= 2.000
ERRA	= 0, 0
ERROR	= 5001
EUNIT	= 1.000
HAS	= 0.000
HCDT	= 500
IOF 10	= 0
IOF 20	= 7
IOF 30	= 7
IOF 40	= 7
IOF 50	= 7
IOS 11	= 1, 1, 1, 0, 2, 0
IOS 12	= 2, 1, 1, 0, 2, 0
IOS 13	= 3, 0, 1, 0, 1, 0
IOS 14	= 4, 0, 1, 0, 1, 0
IOS 15	= 5, 0, 1, 0, 1, 0
IOS 16	= 6, 0, 1, 0, 1, 0
IOS 17	= 0, 0, 1, 0, 0, 0
IOS 18	= 0, 0, 1, 0, 0, 0
IOS 21	= 0, 0, 1, 0, 0, 0
IOS 22	= 0, 0, 1, 0, 0, 0
IOS 23	= 0, 0, 1, 0, 0, 0
IOS 24	= 0, 0, 1, 0, 0, 0
IOS 25	= 0, 0, 1, 0, 0, 0
IOS 26	= 0, 0, 1, 0, 0, 0
IOS 31	= 0, 1, 1, 0, 0, 0
IOS 32	= 0, 1, 1, 0, 0, 0
IOS 33	= 0, 1, 1, 0, 0, 0
IOS 34	= 0, 1, 1, 0, 0, 0
IOS 35	= 0, 1, 1, 0, 0, 0
IOS 36	= 0, 1, 1, 0, 0, 0
IOS 41	= 0, 0, 1, 0, 0, 0
IOS 42	= 0, 0, 1, 0, 0, 0
IOS 43	= 0, 0, 1, 0, 0, 0
IOS 44	= 0, 0, 1, 0, 0, 0
IOS 45	= 0, 0, 1, 0, 0, 0
IOS 46	= 0, 0, 1, 0, 0, 0
IOS 51	= 0, 1, 1, 0, 0, 0
IOS 52	= 0, 1, 1, 0, 0, 0
IOS 53	= 0, 1, 1, 0, 0, 0
IOS 54	= 0, 1, 1, 0, 0, 0
IOS 55	= 0, 1, 1, 0, 0, 0
IOS 56	= 0, 1, 1, 0, 0, 0
IT1	= 0, 0
IT2	= 0, 0
IT3	= 0, 0
IT4	= 0, 0
JOGS	= 51200.000
JSC	= 2048.000
JSDB	= 10.000
JSFS	= 2038.000
LDCLT	= 1
LDECL	= 1000000.000
MAC	= 25
MHC	= 5
MRC	= 25
MSDT	= 0
MSEL	= 256
MUNIT	= 1.000
PAUSM	= 0
PFMT	= 10, 3, 2
PMV	= 10240.000
POS	= 0.000
POSCAP	= 0.000
PRMPT	= ">"
RATIO	= 1.000
RATIOW	= 5
SMOVR	Empty
STEPW	= 5
STLDM	= 0
STLF	= 10.000
TI1	= 0, 0, 0
TI2	= 0, 0, 0
TI3	= 0, 0, 0
TI4	= 0, 0, 0
TP1	= 0.000, 0, 0
TP2	= 0.000, 0, 0
TP3	= 0.000, 0, 0
TP4	= 0.000, 0, 0
TT1	= 0, 0, 0
TT2	= 0, 0, 0
TT3	= 0, 0, 0
TT4	= 0, 0, 0
TV	= 0.000, 0, 0
VI	= 1000.000
VM	= 768000.000
VT	= 0.000, 0

Flags

ACL	= FALSE
BIO	= FALSE
BKGD	= FALSE
BLE	= FALSE
BSY	= FALSE
CSE	= FALSE
DCL	= FALSE
DRIVEN	= TRUE
DRVFS	= FALSE
DRVRS	= FALSE
EE	= FALSE
ERR	= FALSE
ESC	= TRUE
GECHE	= FALSE
HAE	= FALSE
HELD	= FALSE
HOST	= TRUE
INP	= FALSE
JSE	= FALSE
JSHCE	= FALSE
LIMSTP	= FALSE
LOCK	= FALSE
LOGO	= TRUE
MVG	= FALSE
PARTY	= FALSE
PAUSD	= FALSE
PCHG	= FALSE
PME	= FALSE
QUED	= FALSE
RATIOE	= FALSE
STALL	= FALSE
STLDE	= TRUE
TIE1	= FALSE
TIE2	= FALSE
TIE3	= FALSE
TIE4	= FALSE
TPE1	= FALSE
TPE2	= FALSE
TPE3	= FALSE
TPE4	= FALSE
TTR1	= FALSE
TTR2	= FALSE
TTR3	= FALSE
TTR4	= FALSE
TVE	= FALSE
VAE	= FALSE
VCHG	= FALSE

Factory Defined User Flags

HIGH	= G TRUE
H	= G TRUE
LOW	= G FALSE
L	= G FALSE
TRUE	= G TRUE
T	= G TRUE
FALSE	= G FALSE
F	= G FALSE
YES	= G TRUE
Y	= G TRUE
NO	= G FALSE
N	= G FALSE
OFF	= G TRUE
ON	= G FALSE

APPENDIX D

Establishing Communications using Windows95 HyperTerminal

If your Host PC is equipped with the Windows9x or NT/2000 operating systems, you can create a new HyperTerminal Setup by following these steps:

1. Select "Start"
2. Select "Programs"
3. Select "Accessories"
4. Select "HyperTerminal"
5. Double Click on "Hypertrm.exe"
6. Then follow instructions:
   a) Enter name for this setup - "IMS Indexer" works well.
   And choose an Icon, then click "OK".
   b) Select Comm port to connect to. Should be a direct connection. Then click "OK".
   c) Now set the Comm Properties. (These are the factory default settings for the LYNX/MicroLYNX Product.
   These should be:

i)	Baud:	9600
ii)	Data Bits:	8
iii)	Parity:	None
iv)	Stop Bits:	1
v)	Flow Control:	None

Then click "OK".

You will now be in the HyperTerminal you created. You will need to adjust its "Properties". Do this by clicking on the "Properties" Icon on the tool bar or by going to the "File" Menu item. Then select "Properties".

The "Properties" Setup window will show.

i) Select ANSI terminal.
ii) Select ASCII Setup.
   a) Set "Line delay" to 10 msec.
   b) Set "Character delay" to 0 msec.
   (if transfers hang up, increase Char. Delay.)
   c) Click "OK".

BE SURE TO SAVE YOUR SETUP.

Once you have established communication, the following will appear in you terminal window either on power-up or when the system has been reset using ^C (CTRL-C).

```
Program Copyright © 1996-2002 by:
Intelligent Motion Systems, Inc.
Marlborough, CT 06447
VER = x.xxx  SER = Axxxxxx
```