BJx
User's Manual

Manual M96105
December 1997

Revision 3

BJx
User's Manual

Manual M96105
Kollmorgen Motion Technologies Group
201 Rock Road
Radford, VA 24141

December 1997

Printed in the United States of America.

WARNING

Dangerous voltages, currents, temperatures, and energy levels exist in this product and in the associated servo motor(s). Extreme caution should be exercised in the application of this equipment. Only qualified individuals should attempt to install, set-up, and operate this equipment. Ensure that the motor, amplifier, and the end-user assembly are all properly grounded and current limited per NEC requirements.

European Community (EC) Declaration of Conformity

We, Kollmorgen Corporation Industrial Drives Division, 201 Rock Road, Radford, Virginia USA; declare under sole responsibility that this equipment is exclusively designed for incorporation in another machine. The operation of this equipment is submitted to the conformity of the machine in which it is incorporated, following the provisions of the EC Electro-Magnetic Compatibility (EMC) directive 89/392/EEC.

Table of Contents

List of Figures

LIST OF TABLES

Chapter 1
System Description

INTRODUCTION
This chapter presents an overview of the BJx: its functions, features, and options.

PRODUCT DESCRIPTION
The BJx is a family of positioners, positioner/ amplifiers, and smart amplifiers. Together, these products represent some of the smallest, most cost-effective intelligent motion controllers available. The BJx line includes three products:

BJR The BJR is a programmable positioner/ amplifier for brush and brushless motors which operates from 40 VDC and below.

BJRL The BJRL is a programmable positioner/ amplifier for brush and brushless motors which operates from 115 AC.

BJP The BJP is a positioner that can be used with any servo amplifier, such as the KXA from Kollmorgen PMI.

All members of the BJx line execute the BJx language, which has set standards for motion control with its simple, BASIC-like command structure and sophisticated, decision-making capability. The BJx

provides the servo performance you have come to expect from Kollmorgen. By incorporating a high-performance microprocessor, Kollmorgen has designed the BJx without compromising on either positioner software or servo performance. This single microprocessor closes all servo loops, resulting in a truly integrated positioning system. The BJx has the features and performance you need in your next positioning application.

FEATURES
The BJx offers a wide feature set to accommodate real world positioning requirements:

- **LOW COST**

The BJx is affordable, even with its wide variety of advanced features.

- **INTEGRATED PACKAGE**

The BJR and BJRL are easy to install because the servo amplifier and the positioner are integrated into one package.

- **SIMPLE PROGRAMMING LANGUAGE**

The BJx uses simple, BASIC-like commands such as RUN, GOTO (for branching), and GOSUB/RETURN (for subroutines). Advanced IF/ELIF/ELSE/END IF statements result in programs that are easier to read. In addition, you can comment every line in your program.

- **ADVANCED MOTION CONTROL MOVES**

The simple language does not prevent you from solving complex problems. The BJx has separate acceleration and deceleration rates, as well as multiple S-curve acceleration profiles. Profiles can be changed in response to real-time events.

- **MASTER/SLAVE - ELECTRONIC GEARBOX**

The electronic gearbox is used to link two motors together so that the velocity of the slave is proportional to the velocity of the master. The ratio can be from 32,767:1 to 1:32,767. Also, the "index-on-gearing" feature permits real-time phase adjustments.

- **MASTER/ SLAVE - PROFILE REGULATION**

With profile regulation you can control the slave's motion profile according to an external master motor or frequency. Profile regulation modifies the velocity and acceleration of the slave axis without affecting the final position of the move. You can use profile regulation to implement "feed rate override."

- **MOTION GATING**

The BJx can precalculate moves to begin after a transition on the GATE input. This provides rapid and repeatable motion initiation.

- **REGISTRATION**

The BJx has the ability to capture the current position within 25 microseconds after a real-time event, resulting in accurate registration sequences.

- **MATHEMATICS**

Algebraic math is provided for commands such as:

$$X1 = 2 \times (X2 + X3)$$

The BJx has 100 program labels, 750 user-definable variables, and 50 user-definable switches. It also has 15 mathematical/logical operations and over 100 system variables.

- **USER UNITS**

Quantities such as position, velocity, and acceleration are automatically scaled into user-defined units. This allows you to program the BJx in convenient units, such as feet, inches, RPM, or degrees.

- **SUPERIOR SERVO LOOP CONTROL**

The BJx offers smooth, high-resolution motion control and a 32-bit position word. Long-term speed stability is 0.01%.

- **SELF-TUNING**

The BJx can tune itself. You do not need to be a servo expert to set up a system quickly. Just specify the desired bandwidth and let the BJx do the rest.

- **DIGITAL SERVO LOOPS**

Both the position and velocity loops are totally digital. The digital loops give the BJx features not available in standard velocity drives, such as self-tuning, zero velocity offset, and digitally-adjustable servo tuning parameters.

- **FEED-FORWARD GAIN**

Digital feed-forward gain reduces following error for more reliable position control at speed.

- **DIAGNOSTICS**

The BJx offers a complete set of error diagnostics, including English error messages. It remembers 20 errors even through power loss. In addition, the BJx lets you write your own error handler, so you can shut down your process smoothly. The BJx offers trace and single-step modes for debugging. Also included is complete fault monitoring, including travel limit switches and software position limits, as well as hardware safety circuits and checksums for safer and more reliable operation.

- **DIGITAL I/O**

The BJx has up to 20 optically isolated I/O sections. These I/O sections operate on 5 VDC, 12 VDC, or 24 VDC. Inputs and outputs can be connected as sourcing or sinking.

- **ANALOG I/O**

The BJx provides three 10-bit, single-ended, 0-5 VDC analog inputs. Analog power is provided to simplify connection to potentiometers. Also provided is one 8-bit, ±10 VDC analog output for meters.

- **SERIAL COMMUNICATIONS**

BJx serial communications provide a powerful link to other popular factory automation devices, such as PLCs, process control computers, and smart terminals. All BJx units support both RS-232 for terminals and PCs and RS-422/RS-485 for multidrop communications. With multidrop you can put up to 32 axes on one serial line. The BJx can autobaud from 300 baud to 19.2k baud, simplifying installation.

- **MOTION LINK**

Kollmorgen also offers MOTION LINK, a powerful, menu-driven communications package for your IBM-PC (c) compatible computer. With this package, the BJx programs and variables can be retrieved from or saved to a disk drive. Also, on-screen help and a full screen editor are built into MOTION LINK.

- **MENU-DRIVEN SOFTWARE**

The BJx programming language allows you to write operator-friendly, menu-driven software. By incorporating a Kollmorgen Hand Held Terminal (HHT-02) or other serial communications device, the operator can be prompted for specific process data.

- **MONITOR MODE**

The BJx provides interactive communications and permits all system variables and parameters to be examined and modified at any time--even during program execution or while the motor is running.

CHAPTER 2
GETTING STARTED

INTRODUCTION

This chapter presents Motion Link, an IBM-PC based communications package for the BJx. Basic modes of operation and their effect on communication are also discussed.

Motion Link is Kollmorgen's complete IBM-PC-based communications package for the BJx. Motion Link allows you to communicate directly with your BJx, edit and transmit programs to and from your IBM-PC, initialize variables, and record and display real-time variables in PC-Scope.

MOTION LINK INSTALLATION

Computer Requirements

Motion Link requires an IBM-PC or compatible computer with the following features:

- IBM-PC, XT, AT, PS/2, or compatible workstation.
- 512 K RAM.
- PC-DOS or MS-DOS Version 2.5 or later.
- Standard Video Adapter (CGA, MDA, EGA, MCGA, or VGA).
- Serial Port (for communication link with BJx). The serial communications port may be COM1 or COM2. These are the normal configurations:

  COM1: (PC Address 3F8h, IRQ #4)
  COM2: (PC Address 2F8h, IRQ #3)

Software Installation

This section explains backing up and copying files from the Motion Link disk to your computer's hard disk.

Backing Up the Disk

The Motion Link disk is located in the disk holder in the back of this manual. Make a back-up copy and store the master disk in a safe place.

To make a back-up disk, type:

```
DISKCOPY A: A:
```

1. Press enter and follow the DOS prompts on screen concerning source (Motion Link) and destination (blank disk) disks.

Software Installation

Follow these steps to install Motion Link on a hard disk:

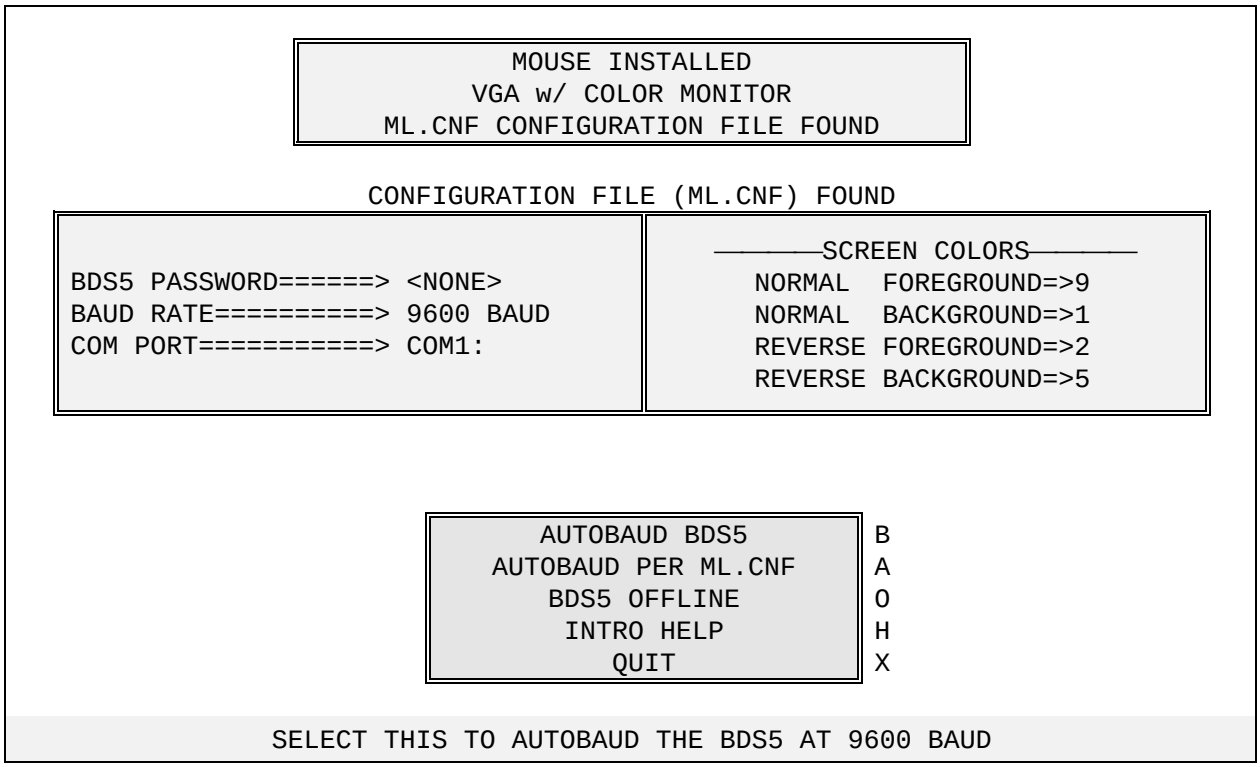


Figure 2.1 BJx Introduction Screen

1. Enter the root directory.
2. Make a subdirectory named ML5 on your hard disk.
3. Change to subdirectory ML5.
4. Insert the Motion Link disk into the A-drive.
5. Copy all files from the Motion Link disk onto the hard disk.

```
C:
MD\ML5
CD\ML5
COPY A:*.*
```

Establishing Communications

WARNING

Do not proceed unless you have completed the check-out procedure from the BJx *Installation* manual.

To begin using Motion Link,

1. Apply power to your BJx as described in the *Installation* manual. Connect the serial cable from your computer to J15.
2. Type:

```
CD\ML5
ML
```

3. When Motion Link responds, the BJx should respond on your PC monitor with an introductory screen similar to that shown in Figure 2.1.

This screen displays the current BJx configuration. The small box near the bottom of the screen provides five choices for the operator: Autobaud BDS5, Autobaud Per ML.CNF, BDS5 Offline, Intro. Help, and Quit. Normally, you will choose Autobaud BDS5 (B).

NOTE

Motion Link will Autobaud only on the COM1 port of your IBM-PC

Choosing to autobaud with the BJx allows direct interactive communication with the BJx. The BJx should respond with a sign-on message and the interactive prompt, "-->." This means the BJx is ready and waiting for a command. When you type,

you are communicating with the BJx just as you would with a terminal. For example, type:

```
P "HELLO, WORLD"
```

and the BJx should respond by printing:

```
HELLO, WORLD
```

You can enter any BJx command from Motion Link just as if your IBM-PC compatible computer were a terminal.

MOTION LINK OVERVIEW

Menus and Windows

Motion Link's special features are accessed through a menu bar printed at the top of your PC screen. When you select an entry from the menu bar, a pull-down window appears, allowing you to select an item. Press the F10 key, the right arrow key, or the left arrow key to display the menu bar. You can leave a window or the menu bar by pressing the escape key. The menu bar offers six choices:

PROGRAM	Modify BJx programs
VARIABLE	Modify BJx variable set
CAPTURE	Record communications
SCOPE	Display PC-Scope
OPTIONS	Set up Motion Link
HELP	On-screen help
UTILITIES	General function

Program

The PROGRAM pull-down window allows you to retrieve, edit, transmit, and save BJx programs. Normally, you will use the Motion Link Editor to write programs for the BJx. Motion Link will save the programs to a file with a .BDS extension. These are referred to as BDS5 files, although the format is compatible with the BJx. Note, however, that there are minor differences between the BDS5 and BJx command languages, so that programs written for one unit may not be executed properly by the other.

- EDIT - Calls the Motion Link Editor and assumes that you want to re-edit the last program. Note that if you have selected an item from either the VARIABLES or CAPTURE menu since you last edited a program, this selection is invalid.
- FROM DISK - Retrieves a program from your computer.
- FROM BJx - Retrieves the program currently stored in the BJx.
- NEW PROGRAM - Calls the Motion Link Editor, allowing you to enter a new program.

Upon exiting the Motion Link Editor, you can store the program to your computer disk and/or transmit it to the BJx.

Variables

The VARIABLES pull-down window allows you to retrieve, edit, transmit, and save BJx variable files. A BJx variable file contains a list of some or all of the BJx variables with initial values. This includes user variables and control variables. Together, these variables configure a BJx for an application.

Note that there are minor differences between the variables in the BDS5 and the BJx. For example, variables taken from the BDS5 should not be transmitted to the BJx. However, Motion Link gives variable files from either unit the file extension .VAR.

- EDIT - Calls the Motion Link Editor and assumes that you want to edit the last variable file edited. Note that if you have selected an item from either the PROGRAM or CAPTURE menu since you last edited a variable file, this selection is invalid.
- FROM DISK - Retrieves a variable file from your computer.
- FROM BJx - Retrieves all of the variables currently stored in the BJx.
- NEW VARIABLES - Calls the Motion Link Editor, allowing you to enter a new set of variables.

Capture

NOTE

This is a communications capture and is unrelated to the BJx variables CAP and CAPDIR, which are for position capture.

- EDIT - Allows you to examine the communications that have been captured. Note that if you selected an item from either the PROGRAM or VARIABLES menu since you

last captured communications or loaded a communications capture file, this selection is invalid.

- FROM DISK - Allows you to retrieve a capture file from disk and examine it with the Motion Link Editor.

- START CAPTURE - Starts (or re-starts) capturing communications from the BJx. This selection always clears the capture storage area before beginning to capture new communications.

- STOP CAPTURE - Terminates the communications capture. If you want to examine the communications that were captured, select "EDIT" in this menu.

Scope

- VIEW AGAIN - Lets you view playback data that was previously retrieved from the BJx.

- FROM DISK - Retrieves recorded data from your computer disk. Motion Link will display all of the playback files currently on your disk and allow you to choose the file you want. Playback files have the file type .CSV for "comma separated variables." This format is compatible with most spreadsheets.

- FROM BJX - Retrieves playback data stored in the BJx. After the playback data is retrieved, it is plotted and stored on disk.

- VIEW DATA - View the data in numerical (rather than graphical) format.

- PRINT PLOT - Print the plot on a line printer.

Options

- SELECT AXIS - Allows you to select options that are available to systems using RS-485 communications.

- BDS5 PASSWORD - Allows you to enter the password that you set with the BJx command "PASSWORD." If you set such a password in the BJx, Motion Link needs the password to transmit new programs to the BJx. If you use this selection to change the password, then you should use the UPDATE CONFIGURATION function described below to write a new configuration file.

  To set the password on the BJx, from the interactive mode (-->) enter

  PASSWORD

  and follow the instructions printed by the BJx.

- COMMUNICATIONS - Allows you to set up your communications port. After you have set up this port, Motion Link will initiate an autobaud sequence to re-establish communications. Remember to power-down the BJx so that it will autobaud. Autobauding is initiated when the BJx is powered up with the front panel switch SW1-2 turned on (See Chapter 3 of the Installation Manual). If you want Motion Link to use the new communications setup in the future, you must use the UPDATE CONFIGURATION function described below to write a new configuration file on your computer disk.

- SCREEN COLORS - Allows you to change the colors displayed on your computer monitor. If you want Motion Link to use the new colors in the future, you must use the UPDATE CONFIGURATION function described below to write a new configuration file.

- CABLE DISCONNECT - Recommended method of disconnecting the communication cable from a powered-up BJx. After you have reconnected the cable, press the space bar and Motion Link will restart communications. This procedure prevents generation of characters when reconnecting serial cables.

- UPDATE CONFIGURATION - Allows you to examine and write the Motion Link configuration file. This file contains information about your computer, such as the communications port you are using, the baud rate at which your computer is transmitting, and your screen colors.

  After you make these changes, you should update the configuration file (ML.CNF) with this selection.

- TL FROM DISK - This is an internal function.

- TL FROM BJX- This is an internal function.

Help

- BDS5 HELP <F1> - Displays several help screens for the BDS5. It lists BDS5 commands and variables with brief descriptions. These commands are generally compatible with the BJx.

- INTRO HELP - Displays introductory information about Motion Link.

- THIS HELP SCREEN <F10> - Displays a help screen.

- LAST COMMAND <F3> - Recalls your last command. You can also press F3.

- VARIABLE INPUT ^V - If you have included a variable input routine in your BJx program (VARIABLE$) and your program is running, this selection will initiate that routine. You can also press ^V (hold down the control key and press V).

- STOP MOTION ^X - Breaks your BJx program and stops motion. You can also press ^X.

Utilities

- RUN DEP01 SIMULATOR - Allows the computer to simulate Kollmorgen's DEP (Data Entry Panel).

- RUN BDS5 SETUP PROGRAM - Provides utilities to test I/O, drive feedback, communication, and dedicated switches.

- EXIT TO DOS Alt-X - Terminates Motion Link and returns to DOS. You can also press Alt-X (hold down the alternate key and press X) for this function.

- SHELL TO DOS - Allows you to temporarily exit (or "shell") to DOS so that you can execute a DOS command. Type "EXIT" to return to Motion Link.

Editor

The Motion Link Editor is a full-featured screen editor. Use this editor to examine or edit programs and variable files, or to capture data. All of the editor commands can be accessed from a menu bar and pull-down windows. Press the F10 key to display the menu bar, then use the left and right arrow keys to select a pull-down window. Each editor command can be accessed with a "control key" or "hot-key" sequence. You can use the control key as a shortcut in place of selecting from the window. The control-key sequence is listed beside each command here and in Motion Link. For example, the FILE-PRINT selection can be accessed with ^P (hold down the control key and press P). Many selections require two control keys, such as FILE-FILE MERGE ^K^R. In this case, hold down the control key, press and release K, then press R. The rest of this section will discuss each of the editor pull-down windows.

File

- SAVE FILE ^K^S - Copy a file in the editor to the disk.

- MERGE FILE ^K^R - Copy a file into the editor starting at the cursor.

- PRINT... ^P - Print the contents of the editor.

- EXIT <Esc> - Exit the Motion Link Editor. You can also use the escape key for this function.

Edit

- MARK START OF BLOCK ^K^B - Marks the beginning of a block. If you want to move or eliminate a block of text, use this command to mark the top and the bottom of the block you want to manipulate.

- COPY MARKED BLOCK ^K^C - After you have marked a block, use this command to copy the marked block into the Motion Link cut/paste memory.

- CUT MARKED BLOCK ^K^V - After you have marked a block, this command copies the marked block into the Motion Link cut/paste memory and deletes it from the editor.

- PASTE CUT/COPIED BLOCK ^K^P - After you have either copied or cut a block to the cut/paste memory, this command copies the cut/paste memory into the editor, starting at the cursor.

- SAVE MARKED BLOCK ^K^W - After you have marked a block, this command saves the marked block to a file on your disk.

GOTO

- FIND A STRING ^Q^F - Finds a string in the editor. Motion Link will prompt you to enter the string.

- REPEAT LAST FIND ^L - Repeats the last FIND A STRING.

- GOTO A LINE NUMBER ^Q^I - Moves the cursor to the specified line. Note that you can transmit your program to the BJx without comments. Since comment lines can be ignored by Motion Link when your program is transmitted, the line numbers in the editor may not agree with your program line numbers in the BJx. Because of this, Motion Link will ask you if you want to count comments. If you are trying to find a line number from a BJx error message and you transmitted your

program without comments, specify that you DO NOT want Motion Link to count comment lines.

- SHOW SIZE OF EDITOR ^Q^O - Displays space remaining in the Motion Link Editor. Use this selection if you are concerned that your program is filling up the editor. The Motion Link Editor can hold up to 2000 lines and up to about 24,000 bytes.

- SHOW FREE MEMORY ^K^F - Displays space remaining for your BJx program. Use this command when you are concerned that your program will fill up the BJx program memory.

Insert/Delete

- DELETE A WORD ^T - Deletes the next word after the cursor.

- DELETE TO END OF LINE ^Q^Y - Deletes from the cursor to the end of the line.

- DELETE A LINE ^Y - Deletes the entire line that the cursor is on.

- UNDELETE A LINE ^U - Inserts the last deleted line in the editor, starting at the cursor.

- INSERT A NEW LINE ^N - Inserts a blank line in the editor.

- DELETE ENTIRE EDITOR ^K^Y - Clears the entire Motion Link Editor.

Cursor

Table 2.1 shows the cursor control keys. Special keys are shown between less than and greater than symbols; for example, the Home key is shown as <Home>.

Table 2.1 Cursor Control Keys

TOP OF EDITOR	^<PageUp>
END OF EDITOR	^<PageDn>
BEGINNING OF LINE	<Home>
END OF LINE	<End>
LEFT ONE WORD	^<Left> or ^A
RIGHT ONE WORD	^<Right> or ^F

Types Of Data Files

Motion Link stores, retrieves, displays, and edits three types of data files. Each type has a different *file extension* or *file type*. *File extension* refers to the characters in the file name that follow the period. For example, the file TEST.BDS has the file extension "BDS." The three types of files are:

BDS User Programs for the BJx.

VAR Variable sets for the BJx. Variable files may include some or all of the BJx variables. For example, your Motion Link disk has the file "STANDARD.VAR." This variable file includes all of the standard variable settings. Variable files are normally transmitted to the BJx to initialize variables before programs are run.

CAP Capture files contain captured communications from the BJx. The capture features of the BJx allow you to collect and store up to 16,000 bytes of transmissions from the BJx.

Motion Link normally determines the proper file extension.

MOTION LINK SETUP PROGRAM

Access the Motion Link Setup Program through the Utilities Menu. Setup provides the following test capabilities:

- Communicate with the BJx
- Tune Drive
- Drive Test
- Drive Feedback
- Input Test
- Output Test
- Machine Setup - Units
- Machine Setup - Limits
- Motor Setup
- BJx Modes
- Communications
- Other
- Send Variables
- Reset Variables

This test program provides the operator with user-friendly methods for testing most BJx functions.

PROCESSOR MODES

Prompts

The BJx provides several modes of operation. Each mode is distinguished by a unique prompt, the short series of characters that the BJx writes to the screen requesting input. For example, the interactive prompt is "-->."

The BJx is designed to receive commands from a terminal or a computer through a serial port. In order to support computer communications, the BJx observes the following conventions.

Table 2.2 BJx Rules For Prompts

1. Prompts are 3 or 4 characters long.
2. Prompts end with a greater than symbol (>).
3. Each mode has a unique prompt.
4. Once the BJx displays a prompt, it stops transmitting until a new instruction and/or a carriage return are received.

The last rule ensures that there is never a question about which device is transmitting. If a ">" has been issued from the BJx, then the BJx will not transmit anything until a command has been entered. The only exception is if you program the BJx to print a ">" from a PRINT or INPUT command. The BJx will allow ">" in print statements, though this is considered a poor practice if you are using a computer to communicate with the BJx.

Similarly, the BJx will not accept input unless a ">" has been issued. The INPUT command is the only exception to this rule. This rule can be awkward if you are using the BJx from a terminal; if an error occurs during the interactive or monitor modes after the ">" has been displayed, the BJx will not print the error message until a carriage return or escape has been entered.

The prompt for each mode is listed below. The only exception is the Run mode. This mode does not have a prompt since input is not normally accepted from the serial port. Notice that the trace prompt does not end with ">." This is because the trace prompt does not indicate that the BJx is waiting for input. If the BJx is communicating within a multidrop communication line, then the prompt is modified to include a prefix that indicates the axis address. Table 2.3 shows the prompts in both the single and multidrop configurations. Note that the multidrop address is assumed 65 (ASCII "A") for this table.

Table 2.3 BJx Prompts

Mode	Non-multidrop (ADDR = 0)	Multidrop (ADDR = 65)
Interactive	`-->`	`A->`
Monitor	`==>`	`A=>`
Single-step	`s-->`	`As->`
Trace	`t...`	`At..`
Load	`1->`	`Al>`

Mode Descriptions

The following section describes each of the modes of operation. Figure 2.2 is a diagram of each mode and how it interacts with the other modes.

Interactive Mode

The BJx normally powers-up in the Interactive mode. This mode allows you to start programs, display and change variables, and enter motion commands for immediate execution. The prompt (-->) is written to the screen, and the BJx awaits a new command.

Refer to Figure 2.2. There are many ways to enter the Interactive mode. First, if the power-up label (POWER-UP$) is not present, the BJx will power-up in the Interactive mode. The BREAK (B) command and errors that break program execution cause the BJx to exit the Run mode and enter the Interactive mode.

Run Mode

The BJx is normally in the Run mode when a program is executing. There is no prompt because input is not accepted from the terminal. The program can display errors and print to the terminal.

Refer to Figure 2.2. After autobauding, the Run mode is normally entered from either the Interactive mode, the RUN command, or from multi-tasking. If the power-up label (POWER-UP$) is present, the BJx will start running your program at that label on power-up.

Errors can also cause the BJx to change modes. Some errors are serious enough to cause the BJx to break program execution. Usually, this has the identical effect of issuing a BREAK (B) command.

As an option, you can write an error handling routine beginning at label ERROR$. This routine should be short and should end with a BREAK (B) command. The error handler is intended for graceful error recovery. For example, you can set outputs or print a message. However, the program may issue a RUN command after handling the error condition.

Monitor Mode

The BJx Monitor mode is a unique mode for positioners. In this mode, the user program is running, but commands are accepted from the terminal for immediate execution. The Monitor mode allows you to display and change variables during program execution, including tuning variables.
You can print and modify any variables. The commands that are allowed from the Monitor mode are a subset of the commands allowed from the Interactive mode as shown in Table 2.4.

Table 2.4 Monitor Mode Commands

?	;	B	DIS	EN
ERR	K	P	PS	R
RS	S	ZPE		

In the Monitor mode, all print commands from the user program are suppressed, and the monitor prompt (==>) is displayed. Print commands typed in from the Monitor mode are executed immediately.

To enter the Monitor mode, press the escape key while a program is running. Pressing the escape key again will change back to the Run mode. STOP, BREAK, and KILL all return the BJx to the Interactive mode.

Single-Step Mode

The Single-Step mode is provided for debugging, and it allows you to execute a program one step at a time. The single-step prompt (s->) prints out, followed by the line that is about to be executed (the *next* command). Any command allowed from the terminal in the Monitor mode is also allowed from the terminal in the Single-Step mode. These commands allow you to probe the BJx variables while debugging your program. If you press the enter key without a command entered, then the next command in the user program is executed. To stop the program, enter the *S*, *B*, or *K* command. To turn off the Single-Step mode and allow the program to execute normally, press the escape key twice (once to get into the Monitor mode and again to get into the Run mode), or type *SS OFF*.

Single-Step mode is enabled by turning SS on, either from the program, from the Interactive mode before running the program, or from the Monitor mode. After SS is on, the BJx will enter the Single-Step mode when the user program is executed. SS can also be turned on and off from the program. This is useful if you want to single step through certain sections. Turning SS off from the program returns the BJx to the Run mode.

Trace Mode

The Trace mode is provided for debugging user programs. When in trace, the BJx prints statements before they are executed. The trace prompt (t...) is printed out, followed by the line that is about to be executed, and the line is then executed. This process is repeated for each command.

Trace is enabled by turning TRC on. When TRC is on, the BJx will enter the Trace mode when the user program is executed. TRC can be turned on and off from the Interactive mode before executing the program or from the program itself. It can be turned on from the Monitor mode. Pressing the escape key from the Trace mode will cause the BJx to exit to the Monitor mode and turn TRC off. If both TRC and SS are on, then the BJx will be in Single-Step mode.

Other Modes

Other modes shown in Figure 2.2 include the communication modes (Program Load, Program Dump, and System Dump). These modes are covered in later chapters.

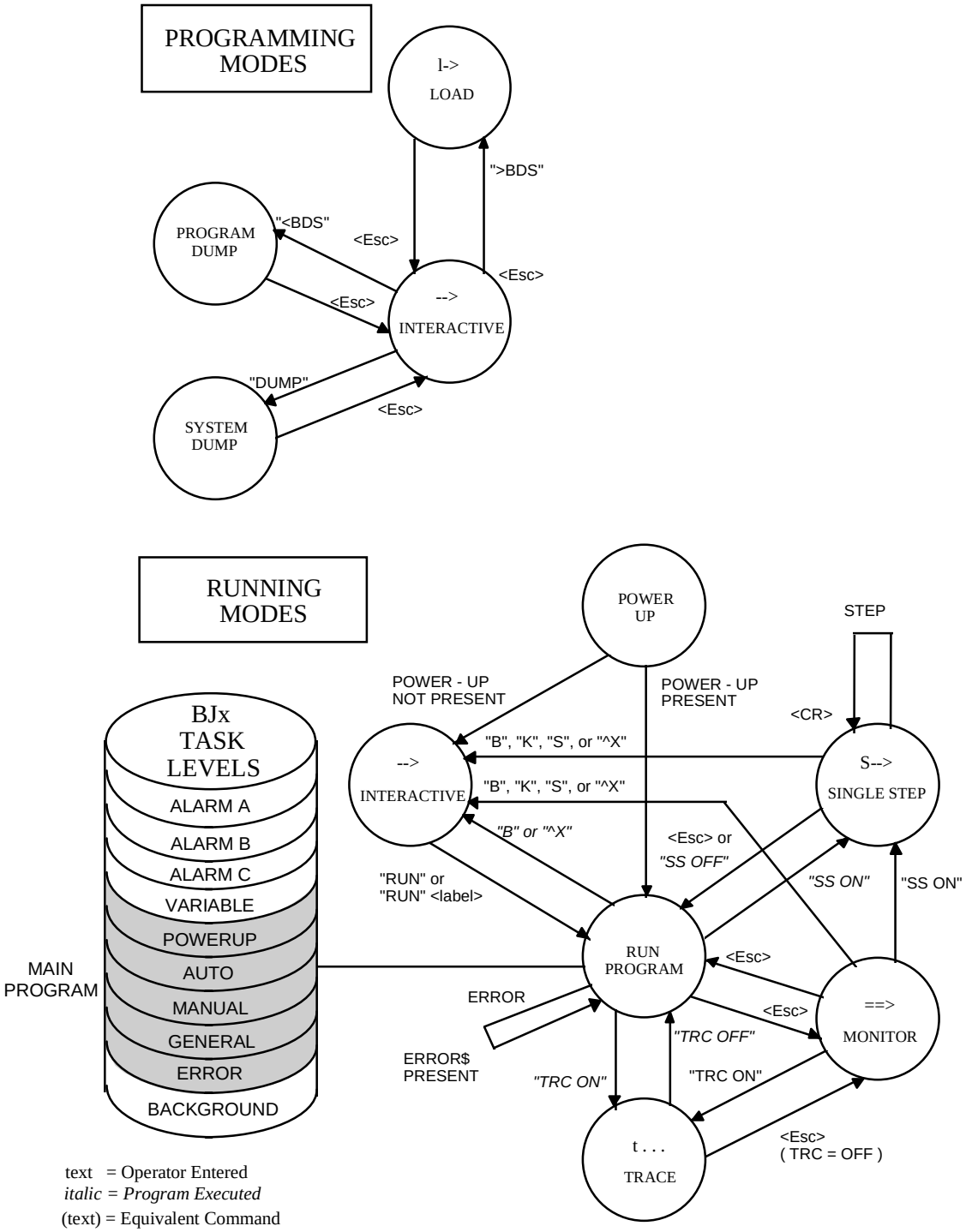


Figure 2.2 BJx State Table

CHAPTER 3
COMMAND LANGUAGE

INTRODUCTION

This chapter discusses the basics of the BJx programming language. Your BJx system should be mounted and wired as described in the *Installation* manual.

INSTRUCTIONS

The BJx can respond to instructions entered from the terminal. The format of the instructions is usually a command followed by one or more parameters. For example, the jog instruction (J) followed by one parameter, the desired speed, would cause the motor to jog at that speed. The command and parameter must be separated by at least one space.

```
J 10
```

Comments

Instructions can be commented. A semicolon marks the beginning of a comment, and the BJx ignores everything on the line after the semicolon. For example,

```
J 10 ;THIS IS A GOOD COMMENT
```

is valid. Note that a space must separate the last parameter from the semicolon:

```
J 10;BAD COMMENT-";" MUST BE
    ;PRECEDED BY A SPACE

;GOOD LINE. SPACE NOT REQUIRED-
;WHOLE LINE IS A COMMENT
```

VARIABLES

The BJx uses a wide range of variables to monitor and control its processes. Many of these variables have units--for example, all variables related to velocity have velocity units.

Introduction to Units

Many BJx variables are set and printed in variable units. These include variables that control or display

- Current
- Position
- Velocity
- Acceleration

Later in this chapter, we will discuss units in detail. For now, set units as follows:

Current Units:
Use percent-of-full current. Set INUM = 4095 and IDEN = 100.

Position Units:
Use counts. Set PNUM = 1 and PDEN = 1.

Velocity Units:
Use RPM. Set VDEN = 10. Set VNUM = feedback-encoder lines * 43.691. For example, for a 1000-line encoder, set VNUM to 43,691. If the encoder is connected indirectly to the motor (e.g. through a belt or gear), use the number of counts per motor revolution.

Acceleration Units:
Use RPM/second. Set ANUM = VNUM and ADEN = 10,000.

Three Types of Variables

The BJx has many variables, all of which are listed in Appendix E. These variables can be divided into three groups: monitor, control, and user.

- MONITOR VARIABLES
  Monitor variables are read-only. They can be displayed and used in calculations but cannot be changed directly. The BJx updates these variables to reflect status. Position feedback, PFB, is an example of a monitor variable.

- CONTROL VARIABLES
  Control variables allow you to change or limit some process in the BJx. An example of a control variable is current limit, ILIM. ILIM limits the current command.

- USER VARIABLES
  User variables allow you to store information for later use or hold intermediate results of calculations.

Variable Limits

All variables have limits. Attempting to set a variable to a value outside its limits generates an error. For example, ILIM must be between 0 and 100. The limits of each variable are listed in Appendix E.

Switches

Switches are variables that can be set to either 0 or 1.

Printing Variables

All variables can be displayed using the Print (P) command. For example, type:

```
P ILIM
```

Since the standard setting of ILIM on most systems is 100, the terminal should display:

```
100
```

Suppose you want to display PFB, the position feedback. Type:

```
P PFB
```

The position feedback should now be displayed. By hand, rotate the motor shaft about half a revolution. Print PFB again. Notice that it has changed to reflect the new position.

Changing a Variable

NOTE

The following example changes ILIM. Reset ILIM to its original value.

Variables are changed with assignment instructions. An assignment begins with the name of the variable, followed by "=" and ending with the new value. This example assigns ILIM a new value of 10:

```
P ILIM
ILIM=10
P ILIM
```

Restore ILIM to its original value:

```
ILIM=100        ;USE ORIGINAL ILIM VALUE
```

Changing Variables

Most variables can be changed but often under limited conditions. For example, the maximum acceleration level, AMAX, can be changed only when the BJx is disabled. Attempting to change AMAX with the BJx enabled will generate an error.
The conditions under which a variable can be changed are called programming conditions. The programming conditions of all variables are listed in Appendix E.

NOTE

Limits and programming conditions for all variables are shown in Appendix E.

User Variables

User variables are like memory on a hand-held calculator. They can be used as application-specific variables or for storing intermediate results of complex calculations. There are 250 user variables: X1, X2, . . . X250. (On some models, user variables can be extended to 750 by setting EXTDX=1). They can be displayed and assigned new values in the same way as other variables. They can store numbers that range from -2^{31} (-2,147,473,648) to $2^{31}-1$ (2,147,473,647)negatives in right place?}. For example, if you want to store PFB, the position feedback, at a particular time and use it later in a calculation, you can assign PFB to a user variable. Type the following line on the terminal:

```
X1=PFB
```

X1 will store the current value of PFB indefinitely.

Indirect User Variables

An advanced method of accessing the values stored in user variables is called *indirect*. With indirect user variables, the specified user variable "points" at another user variable. Indirect references to variables have the format: X(Xn) where n is between 1 and 250. The value stored in the variable Xn specifies the variable that X(Xn) refers to. For example, suppose you want to look at either X1 or X2 when X10 is either 1 or 2. Type the following example:

```
X1=100
X2=1000
X10=1          ;USE X10 TO POINT TO X1
P  X(X10)      ;PRINT WHAT X10 POINTS
               ;AT
```

The BJx responds:

```
100
```

since X(X10) = X1 = 100.

Now type:

```
X10=2          ;USE X10 TO POINT TO X2
P  X(X10)      ;PRINT WHAT X10 POINTS
               ;AT
```

The BJx responds with the value of X2:

```
1000
```

Indirect user variables are often used to look up data in tables. For example, they are often used in programs that remember a large number of positions taught by the operator. In this case, many user variables are used to remember positions, and one variable is used to point at the group. Use indirect references with caution since it is easy to make programming errors with them.

User Switches

User switches are similar to user variables, except that they can only take on values of 0 or 1. A user switch can be used in place of a user variable if you only need to store 0 or 1. An example of a good place for a user switch would be to store information for go/no-go decisions. This saves user variables for other places. There are 50 user switches: XS1-XS50. For example, type:

```
XS33=1
P XS33
```

and the BJx prints 1.

Power-up and Variables

All control variables, except switches, are stored through power down. All user variables (X1-X250) are set to zero on power-up. If extended user variables are enabled (EXTDX ON), then they (X251-X750) are also cleared.

NOTE

All user variables are set to zero on Power-up.

Most switches are reset on power-up. The only exceptions are switches which, if not remembered, are likely to destabilize the system. Table 3.1 shows the condition of all BJx programmable switches on power-up.

Special Constants

The examples above have used decimal numbers in most of the assignments. There are four special constants that make the BJx easier to use: ON, OFF, Y, and N. ON is the same as 1 and OFF is the same as 0. Similarly, Y is 1 and N is 0. These constants are normally used for switches. Compare the two statements:

```
O1=1
O1 ON
```

These statements are equivalent, although the effect of the second is more intuitive. When you write programs, the use of ON and OFF and Y and N can make the program easier to understand. Note, however, that the P command normally prints numbers, not ON, OFF, Y, or N. For example:

```
O1=ON
P OUT
```

will result in "1" being printed, not "ON." Another point to recognize is that the equal sign (=) is optional. The following two statements produce identical results.

```
O1=ON
O1 ON
```

Table 3.1 Power-Up State of Programmable Switches

OFF	ON	REMEMBER FROM LAST POWER-UP
CAM	CAPDIR	ENCDIR
CAP	DIR	LPF
CLAMP	ECHO	MENCDIR
DEP	MSG	
EXTDX	MULTI	
FAULT	PL	
GATEMODE	PLIM	
GEAR	PROMPT	
O1 - O5	STATSEN	
PROP		
RAMP		
REG		
ROTARY		
SS		
TQ		
TRC		
TRIP		
WATCH		
XS1-XS50		

MATH

Hexadecimal

The BJx allows constants to be entered in hexadecimal, or hex. Hex, or base 16 representation, is often used when programming computers. BJx hex constants begin with a number followed by an "h." For example: 16h, 0Fh and 0FFh are all hex numbers. Appendix B shows the hex conversion of 0 through 255. From the appendix, you can see that 25 hex is equal to 37 decimal so that the two following instructions are equivalent.

```
X9=37
X9=25H
```

Sometimes, the first digit of a hex number can be a letter. In this case, the number must be preceded with a zero. For example:

```
X9=FFH      ;ERROR-HEX NUMBER
            ;MUST BEGIN WITH A
            ;NUMBER.
X9=0FFH     ;VALID STATEMENT
```

Hex is useful when trying to use general purpose inputs to control the user program. For more information about applying these inputs, see the "General Purpose Input/Output" section later in this chapter.

Algebraic Functions

The BJx provides four standard algebraic functions: multiplication, division, addition, and subtraction. The usual algebraic operators (*, /, +, -) are used. Standard algebraic hierarchy is observed: all multiplications and divisions are done before any additions or subtractions. Parentheses are provided to override this precedence. Type in the following examples:

```
P 1+2*3     ;THIS PRINTS 7, NOT 9--* IS
            ;DONE BEFORE +
P (1+2)*3   ;THIS PRINTS 9
```

Math expressions must obey the rules listed below in Table 3.2.

Table 3.2 Rules for Math Expressions

1. No spaces are allowed.
2. Any valid variables can be used.
3. Any valid constants can be used.
4. Indirect user variables can be used.
5. Any math operator can be used.
6. Parentheses can be nested to 2 levels.
7. Integer math is used for all operations.
8. Expressions are evaluated left to right.

Valid math expressions can be substituted for numbers in most instructions. A few examples of math expressions in assignment instructions follow.

```
X1=500
X1=5*100
X1=5000/10
X1=(7+3)*(28+22)
```

All set X1 to 500. Furthermore, variables can be used in the expression:

```
X1=20
X2=30
X3=X1*X2  ;FILL X3 WITH 600
```

Fractional results from division are rounded to the nearest integer. Also, expressions are evaluated from left to right. These two conditions can cause unexpected results. Consider the following expressions:

```
P 53/100*280    ;THIS PRINTS 280
P 280/100*53    ;THIS PRINTS 159
P 280*53/100    ;THIS PRINTS 148
```

In exact math, these three expressions are equivalent; they calculate 53% of 280, which is exactly 148.4. However, with integer math, the first expression is evaluated as 280. This is because 53/100 is evaluated first. The result, 0.53, is rounded to the nearest integer, 1, which is multiplied by 280. Likewise, in the second expression, the 280/100 is evaluated as 3, which is multiplied by 53 to get the result 159. Only the third expression gives the expected result, 148. In this example, round-off error is minimized by performing the multiplication first.

Logical Functions: AND, OR

Two logical math functions, AND and OR, can also be used in math expressions. ANDing is indicated by "&" operator and ORing is indicated by "!" operator. When evaluating an expression, AND has the same level of precedence as multiplication, and OR has the same level as addition.

Like hex, logical math is often used when programming computers. With logical functions, two numbers are converted to binary representation and compared bit by bit. When the numbers are ORed, if either bit is set, the resulting bit is set. With ANDing, both bits must be set for the result to be set. Type in the following examples:

```
P  1!2      ;THIS IS 3
```

The BJx responds with 3 since

```
            00000001 (Binary 1)
     OR     00000010 (Binary 2)
            00000011 (Binary 3)
```

```
P  1&2          ;THIS IS 0
```

The BJx responds with 0 since

```
                00000001 (Binary 1)
        AND     00000010 (Binary 2)

                00000000 (Binary 0)
```

Logical math is generally used with hex constants.

Logical math is also useful when trying to use general purpose inputs to control the user program.

GENERAL PURPOSE INPUT/ OUTPUT

Digital I/O

The BJx provides 8 general purpose inputs and 5 general purpose outputs. Inputs can be referred to individually as I1, I2, . . . I8 or collectively as IN. Similary, outputs can be referred to as O1, O2, . . . O5 or as OUT. On power-up, all outputs are turned off.

As examples, you can turn the third output on and the fifth off by typing:

```
O3  ON          ;TURN ON THE THIRD
                ;OUTPUT BIT
O5  OFF         ;TURN OFF THE FIFTH
                ;OUTPUT BIT
```

You can display the fifth input by typing:

```
P  I5
```

and either 1 or 0 will be displayed.

Whole Word I/O

Inputs and outputs can also be referred to collectively. In order to do this, the individual inputs or outputs are referenced as the bits of a digital word. Whole Word references are especially useful when you are trying to set or clear many output bits at once. If you are unfamiliar with logical/binary math or you plan to use I/O one bit at a time, you may not be interested in Whole Word I/O. However, it can save space and execution time when properly used.

Whole Word I/O is done using the variables OUT and IN. OUT is a 5-bit digital word representing all of the outputs, with O1 as the least significant bit (LSB). IN is an 8-bit digital word representing all of the inputs, with I1 as the LSB. Each bit has a value which depends on its position within the word. The value in OUT or IN is the sum of the values for each bit that is turned on. The value for each bit is listed in Table 3.3.

Table 3.3 Output 1-5 DecimalValues

Out Bits	O5	O4	O3	O2	O1
Value	16	8	4	2	1

For example, if O5 and O4 are on and all other outputs are off, then:

```
OUT  =  16 (value of O5) + 8 (value of O4)
     =  24.
```

Many bits can be set or cleared with one instruction. For example,

```
OUT=7
```

turns on O1, O2, and O3 while turning all other outputs off. One logical math statement can be used to set some bits without affecting others. For example:

```
O1  ON
O2  ON
O3  ON
```

can be replaced with:

```
OUT=OUT!7   ;SET 3 BITS WITH
            ;LOGICAL OR
```

which turns on O1, O2, and O3 without affecting O4 and O5. The logical AND can be used to turn off several bits:

```
OUT=OUT&7   ;CLEAR 2 BITS WITH
            ;LOGICAL AND
```

turns off O4 and O5 and does not affect O1-O3. Note that the hex representation can be especially useful when setting the higher bits:

```
O4  ON
O5  ON
```

is the same as:

```
OUT=OUT!024
```

IN is formed with I1-I8 in the same way OUT is formed with O1-05:

Table 3.4 Input 1-8 Decimal Values

In Bits	I8	I7	I6	I5
Value	128	64	32	16
In Bits	I4	I3	I2	I1
Value	8	4	2	1

For example, if IN were equal to 130, that would mean I2 and I8 were on and all others were off, because 130 is the sum of those bits:

130 = 2 + 128

Analog I/O

The BJx provides the general purpose analog inputs, AIN1, AIN2, and AIN3. These inputs resolve 0-5 VDC to 10 bits. AIN1-3 range from 0 to 1023. One analog output is provided: AOUT outputs ±10 VDC resolved to 8 bits. AOUT ranges from -128 to +127.

FAULT LOGIC

This section covers how to enable the BJx and how faults affect the operation. This discussion will center around Figure 3.1. Note that this drawing is a functional diagram; it does not directly represent the actual hardware and software used to implement these functions.

Severity-4 Fault Latch

The Severity-4 Fault latch is diagrammed at the top of Figure 3.1. As indicated, any Severity-4 Error disables the amplifier, stops communications, and blinks the FAULT LED, located on the BJx front panel. Severity 4 errors are the most severe errors; they indicate that the unit may not be functioning at the most basic level. Communication is limited to blinking the FAULT LED a fixed number of times, followed by a pause. The number of blinks between pauses indicates the error number. Do not confuse this condition with autobauding, where the FAULT LED blinks at a constant rate.

Please refer to Appendix D for a list of all errors with the corresponding severity.

Severity-3 Fault Latch

The Severity-3 Fault Latch is set by any Severity-3 error. This latch turns on the FAULT LED and the FAULT software switch. The latch remains set until the next Enable (EN) command.

Ready Latch

The Ready Latch is set by the EN command. This turns on the READY Switch. The Ready Latch remains on until a disable command (DIS or K) or a Severity 3 Error. The Ready Latch, combined with the REMOTE Input, controls ACTIVE. When ACTIVE is high, the amplifier is enabled. Note that REMOTE can be disabled with the command

```
NOREMOTE
```

In this case, ACTIVE and READY always have the same value. To reinstate REMOTE, type

```
NOREMOTE OFF
```

OK Latch

The OK Latch is set by Power-up, and the RUN and Enable Commands. This latch controls the OK (Green) LED on the BJx front panel, the OK Output on J11, and the OK Software Switch. Any error that breaks the program or disables the amplifier turns off the OK Latch.

DRIVE CONTROL

This section discusses several variables which control the basic functions of the drive.

Direction Control

The BJx has three switches that allow you to control encoder direction:

DIR

DIR is the direction control for printing and setting velocity and position variables, and for issuing jog and move commands. Changing DIR does not invert the feedback encoder direction. Essentially, DIR is a convenience that allows you to reverse the direction of rotation with a single command.

ENCDIR

ENCDIR sets the direction of the feedback encoder. ENCDIR is normally set to 1. If you are installing a SILVERLINE motor and a BJR(L), leave ENCDIR = 1. However, if you are installing a BJP or using a non-SILVERLINE motor or encoder, be advised that it is common to inadvertently invert the sign of the feedback encoder. This causes the motor to run away when enabled. ENCDIR allows you to reverse the feedback direction without rewiring your system. ENCDIR is remembered on power-up.

IF ENCDIR IS SET INCORRECTLY, THE MOTOR WILL RUN AWAY.

MENCDIR

MENCDIR is similar to ENCDIR. It inverts the direction of the master encoder. Changing MENCDIR inverts the sign of PEXT and VEXT as well as reversing electronic gearing and camming. See Camming and Electronic Gearbox Sections in Chapter 4.

Position

Position Command and Feedback, PCMD & PFB

PCMD is the commanded position. It is generated internally from motion commands such as Jog. PCMD is in position units. The standard position units are encoder counts. PCMD is set to PFB when the BJx is disabled.

PFB, the position feedback, is the actual position of the motor, and it is updated every millisecond. PFB is in position units. It is always active, even when the BJx is disabled. PFB is reset to zero when the BJx is powered-up.

Position Error, PE & PEMAX

PE is position error, sometimes referred to as following error. It is the difference between PCMD and PFB. PE is zero when the BJx is disabled. PE is in position units.

When the magnitude of the position error exceeds the value stored in PEMAX, a Position Error Overflow error is generated. This disables the BJx. Normally, you want to set PEMAX to the lowest possible level that will still allow the system to run reliably. However, setting PEMAX too low can generate nuisance errors since the position error has some variation during motion. PEMAX is in position units.

Position error is limited to protect the system. Excessive position error can indicate a fault condition. For instance, bearings wear out over the life of a motor. The increased load from worn bearings can increase the position error during motion. In many cases, position error is the first indication of wear.

Sampling PFB, PCMD and PEXT

When PFB and PCMD are used on the same line, they are always sampled during the same sampling interval (millisecond). This allows you to use PCMD, PFB, and a third variable, PEXT (see Chapter 4), without concern that the variables might be sampled at different times. For example:

```
P  PCMD "-" PFB " = " PCMD-PFB
```

This command would print the expected results. This is because the BJx stores PCMD and PFB at the beginning of every command, then uses those stored values when the command is executed.

Velocity

VCMD, VFB, VE, & VAVG

VCMD is the commanded velocity. Like PCMD, VCMD is generated internally from motion commands. VCMD is zero when the BJx is disabled. VCMD is in velocity units.

VFB is the feedback velocity, and it is updated every millisecond. VFB is always active, even when the BJx is disabled; if you turn the motor shaft by hand and print VFB on the terminal, you can see the velocity changing. Because VFB is updated rapidly, the speed can appear to vary, even when the motor is rotating at a fairly constant speed. This is because the VFB shows the speed averaged over only 1 millisecond. The speed from one millisecond to the next normally varies a few RPM. The long term speed (that is, measured over a few seconds) normally varies much less (about 0.01%). VFB is in velocity units.

VE is velocity error. VE is the difference between VCMD and VFB in velocity units.

VAVG is the average of VFB over the previous 16 milliseconds. Occasionally, the normal sample-to-sample variation of VFB is undesirable. In these cases, use VAVG.

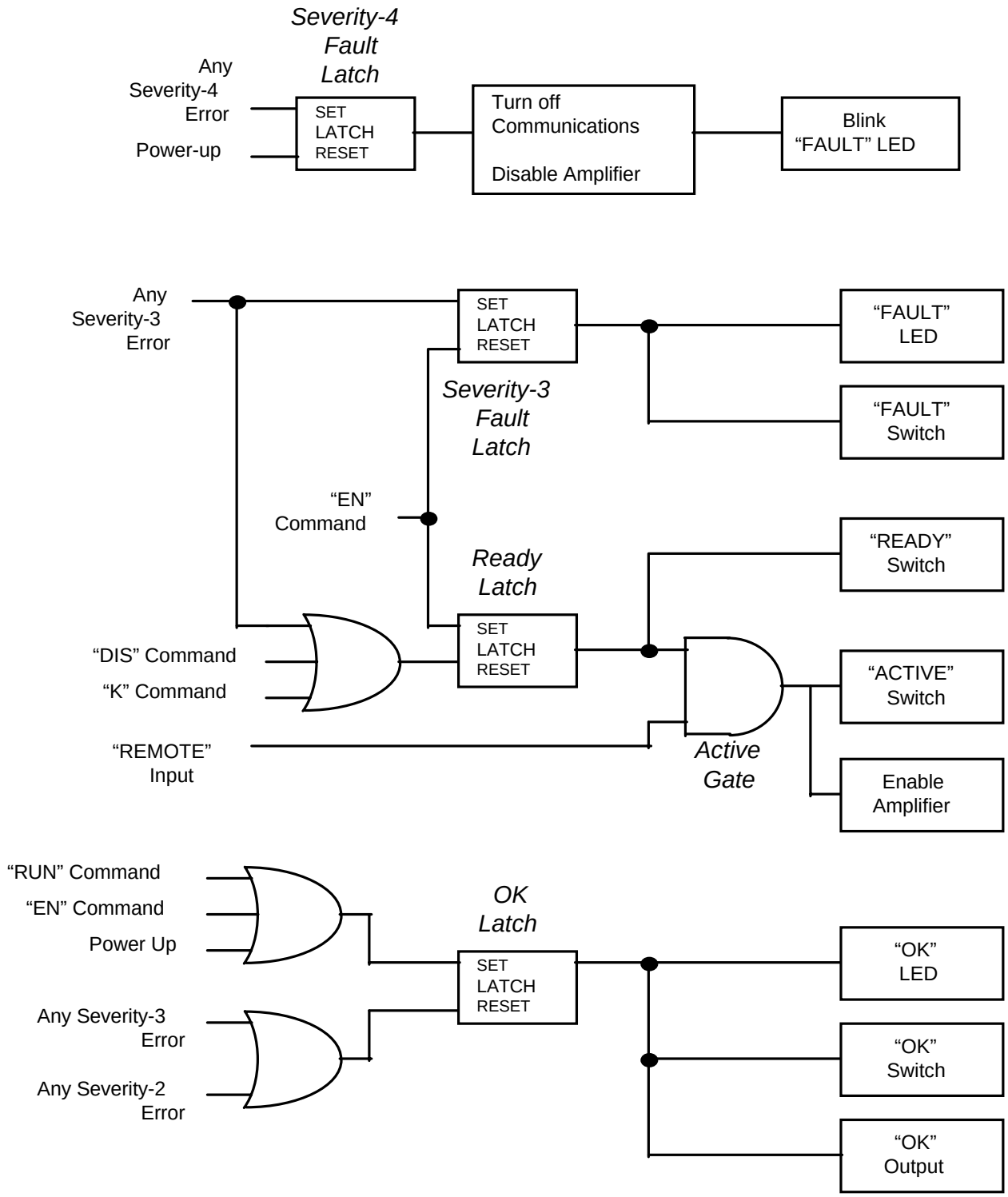


Figure 3.1 BJx Enable/Fault Logic Diagram

Velocity Limits, VMAX & VOSPD

VMAX is the BJx maximum velocity. VMAX is in velocity units.

VOSPD is the maximum velocity for your system. The BJx generates an overspeed fault if VFB is ever greater than VOSPD. You can set VOSPD to any level below 1.2_VMAX. When an overspeed occurs, the BJx is disabled.

You should set VOSPD to at least 10% or 15% above your system's maximum speed to avoid nuisance overspeed faults. You can change VOSPD only when the BJx is disabled. VOSPD is in velocity units.

Current

Motor Current, ICMD

ICMD is commanded motor current. ICMD, like PCMD and VCMD, is generated internally from motion commands. ICMD is in current units.

Current Limits, IMAX & ILIM

IMAX is the maximum level of current that the BJx can output, set at the factory as 100%.

ILIM limits the peak of ICMD, the commanded current. You can set ILIM to any level below IMAX. This allows you to limit the current below the maximum level that the BJx can output. You can set ILIM at any time, even during profile moves. ILIM is in current units.

Current Time Limit, SATTIME

SATTIME is that maximum length of time that the BJx will output ILIM current, set at the factory as 2000 milliseconds (2 seconds).

When ICMD is equal to ILIM, SAT is ON (1). If SAT stays ON for SATTIME milliseconds, the drive will fault and disable, generating ERROR 28, CURRENT SATURATION TIME-OUT. SATTIME may be set by the user, and its units are milliseconds.

Enabling the Position Loop with PL

PL is a switch that controls the position loop. If PL is on, then the position loop is enabled. If PL is off, then it is disabled, and the BJx is running as a velocity loop. Most positioning applications run with PL on. Position loops are discussed in more detail later in this chapter. PL turns on at power-up. You can change PL at any time.

Controlling the Velocity Loop with PROP

PROP is a switch that controls the integration section of the velocity loop. If PROP is on, then the velocity loop is proportional and the integral is disabled. If PROP is off, then the velocity loop is fully integrating. PROP is turned off at power-up, and most applications run with PROP off. You can change PROP at any time. Sometimes proportional velocity loops are used during set-up.

ENABLING THE BJX


WARNING

THE BJx WILL BE ENABLED AND THE MOTOR WILL TURN.

SECURE THE MOTOR.

At this point you should turn REMOTE, LIMIT, and MOTION on as described in the *Installation* manual. Type the following command to print the state of the REMOTE input:

```
P  REMOTE   ;REMOTE SHOULD BE 1
P  LIMIT    ;LIMIT SHOULD BE 1
P  MOTION   ;MOTION SHOULD BE 1
```

If you are not using any, or all, of REMOTE, LIMIT, and MOTION, you can individually disable each:

```
NOREMOTE ON     ;DISABLE REMOTE
NOLIMIT ON      ;    "    LIMIT
NOMOTION ON     ;    "    MOTION
```

You can re-enable any of these functions with the enable command. For example:

```
NOREMOTE OFF    ;ENABLE REMOTE
```


WARNING

THE MOTOR MAY RUN AWAY! BE PREPARED TO DISARM THE BJx.

The feedback encoder may be electrically or mechanically reversed. This is controlled by your components, wiring, and the variable ENCDIR. If the encoder is reversed, the motor will run away.


WARNING

SHOCK HAZARD!

Large voltages from the AC Line and the DC Bus can cause injury. Ensure that the wiring is correct. See the *Installation* manual.


WARNING

THE MOTOR MAY MOVE UNEXPECTEDLY!

BE PREPARED TO REMOVE POWER FROM THE BJx!

Complete "Initial Check-Out" in the *Installation* manual before continuing.

This section will enable the BJx. The system may be unstable. The motor may begin oscillating or run away. Be prepared to remove power quickly.

To enable the BJx, apply power as described in the BJx *Installation* manual and enter the enable command:

```
EN
```

The BJx should turn on. To verify that it did turn on, print ACTIVE. If ACTIVE is 1, then the BJx is enabled; otherwise, it is disabled.

To disable the BJx, enter the disable command:

```
DIS
```

As an alternative, you can disable the BJx with the one-letter kill command by typing:

```
K
```

ENABLE, DISABLE, and KILL are examples of BJx commands. All of the BJx commands are listed, with their formats and syntax, in Appendix C.


NOTE

Appendix C is a quick reference for all BJx commands.

MOTION COMMANDS

This section discusses how to control motion using the BJx. Basic motion commands are described first. Later sections discuss advanced motion control.

Basic Motion Commands

AMAX, ACC, & DEC

The BJx controls acceleration with three variables: AMAX, ACC, and DEC.

AMAX is the maximum acceleration allowed for almost all motion commands. The only exception is electronic gearbox. AMAX is the upper limit for the normal acceleration rates, ACC and DEC. AMAX should always be set below the acceleration level that can damage your machine. Errors that stop motion will decelerate the motor at AMAX; therefore, your machine is subject to deceleration rates of AMAX at any time. AMAX is in acceleration units, which are RPM/second as a default. AMAX can be changed only when the BJx is disabled.


CAUTION

Set AMAX below the maximum acceleration rate that your machine can experience without damage.

ACC is the acceleration rate for most moves and it is given in acceleration units. ACC can be changed at any time, although it must be less than AMAX. Attempting to set ACC to a value greater than AMAX will generate an error.

DEC is the deceleration rate for most moves. DEC is also in acceleration units, and it can be changed at any time. Attempting to set DEC to a value greater than AMAX will generate an error.

EN

Many times, the MOTION input is controlled by the normally closed contacts of a push button. This push button is often called "STOP," since pressing the button opens the MOTION input and forces the motor to stop. Emergency Stop should not be implemented with the MOTION input. Emergency should not be implemented with the MOTION input. Emergency Stop should be connected to a contactor that removes power from the system. This is because an emergency stop, which is for safety, should not depend on BJx functions to operate properly. Before any motion can take place, the BJx must be enabled. Type:

```
EN
```

The MOTION Input

MOTION is a hardware input that enables motion. If MOTION is on, motion is enabled; otherwise, it's inhibited. You can enable the BJx if MOTION is off, but commanding motion will generate an error. See the *Installation* manual for instructions on how to wire MOTION.

MOTION can be disabled as follows:

```
NOMOTION ON
```

This eliminates the requirement that MOTION be on. MOTION can then be used as a general purpose input.

Type the following command to print the state of the MOTION input:

```
P  MOTION    ;MOTION SHOULD BE 1
```


WARNING

Do not use MOTION or any other BJx input for Emergency Stop. When Emergency Stop is activated, it should directly remove power from the system.

STOP (S) Command

Any motion can be stopped using S, the STOP command. S has no parameters. S decelerates the motor at AMAX and terminates all motion commands. The S command does not disable the BJx.

Normally, the STOP command should only be given from the terminal or from the program in response to an error condition. A better method for stopping motion from the program under normal circumstances is to use the JOG (*J*) command (see JOG, later in this chapter). Type:

```
J  0    ;JOG TO 0 SPEED--STOP MOTION
        ;AT DEC, NOT AMAX
```

The *J 0* command also stops motion from any mode, much like STOP. Unlike STOP, *J 0* decelerates at the rate specified by DEC.


NOTE

The STOP (*S*) command should not be used as a part of normal program operation. Use *J 0*.

At any time, when motion is commanded, if the MOTION input turns off, an error is generated and all motion is stopped, as if the STOP command were given. Also, any errors with a severity of 2 or 3 will stop motion in a straight line deceleration at a rate of AMAX. Appendix D lists all errors and their severity.

STOP and BREAK with Control X (^X)

You can execute a stop and break command with the control-X (^X) character. Control-X or ^X means that you hold down the control key (Ctrl) on your terminal (or IBM-PC) and press the X key. This has the same effect as typing B, then S from your terminal.

Limiting Motion

The BJx allows you to limit motion with both Software and Hardware Travel Limits.

Hardware Travel Limits

Hardware Travel Limits limit the range of motion. If you have an application with boundaries that should never be crossed, you are encouraged to use the Hardware Travel Limits with limit switches.

Exceeding Hardware Travel Limits is a more severe error than exceeding Software Travel Limits. The BJx assumes that Software Travel Limits should catch normal overtravel conditions and that a Hardware Travel Limit indicates a serious problem. Hardware Travel Limits disable the BJx rather than just stopping motion.

The *Installation* manual discusses how to wire LIMIT. Usually, two limit switches are wired in series and connected to LIMIT; the contacts of these switches must be closed for the BJx to be enabled. If the contacts open, the BJx will be disabled, the motor will coast to a stop, and an error will be generated. This limit is a safety device and not part of normal program operation. Hardware Travel Limits are always enabled.

If you are not using limit switches, you can disable the LIMIT input by typing:

```
NOLIMIT ON
```

You can then use LIMIT as a general purpose input. You can re-enable LIMIT at any time by typing:

```
NOLIMIT OFF
```

Software Travel Limits, PMAX & PMIN

Software Travel Limits limit the range of motion of the motor. There are two software limits: maximum and minimum. If position feedback (PFB) moves outside the software limits, an error is generated and motion stops. Software Travel Limits are intended as a guard against motion that is out of range due to improper operation or programming errors.

PMAX is the maximum position allowed and PMIN is the minimum. If PFB is greater than PMAX, negative motion is allowed but positive motion is not. If PFB is less than PMIN, only positive motion is allowed. PMAX and PMIN are in position units and can be changed at any time.

Software Travel Limits are enabled with PLIM, which can also be changed at any time. If PLIM is on, software limits are active; otherwise, PMIN and PMAX are ignored. PLIM is turned on at power-up. If you have an application with boundaries that should not be crossed, you are encouraged to use Software Travel Limits.

Note that you should set DIR before setting the Software Travel Limits. This is because DIR relates PMAX and PMIN to clockwise and counter-clockwise motion limits. If you change DIR, you must reset PMAX and PMIN.

User Position Trip Points, PTRIP1 & PTRIP2

The BJx provides two user position trip points, which control a switch. You can use this switch to control your program.

The two trip points are PTRIP1 and PTRIP2. Both are in position units, and you can program either at any time. If the position feedback (PFB) is greater than or equal to PTRIP1, then the TRIP1 switch will be on. If PFB is less than PTRIP1, then TRIP1 will be off. Similarly, if PFB is greater than or equal to PTRIP2, then TRIP2 will be on; otherwise, TRIP2 will be off.

Trip points are not limits in the sense that they do not inhibit motion. They convert position feedback to an on-or-off signal. Trip points are particularly useful with alarms and the HOLD command, both of which are presented in Chapter 5.
Position trip points require a lot of calculations. As a result, they slow the execution of the user program by about 4%. If you are not using trip points, you can disable them by typing:

```
TRIP OFF
```

When the BJx is powered-up, trip points are enabled.

Profiles

When a positioner commands the motor to move from one point to another, it must control acceleration, deceleration, and traverse speed. The velocity of the motion versus time is called the profile. Simple profiles begin and end at zero speed and have three segments: acceleration, traverse, and deceleration. You must specify ACC, the acceleration rate, and DEC, the deceleration rate, before commanding the move. The traverse speed and the distance to move are specified in the move command itself.

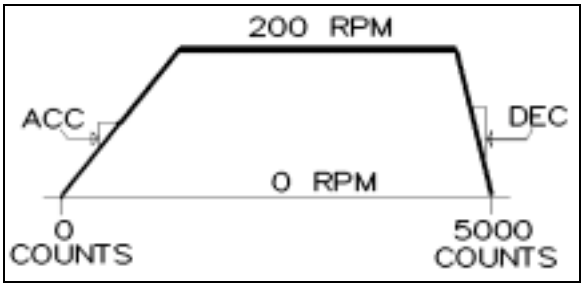


Figure 3.2. A Simple Profile

The graph in Figure 3.2 shows a simple profile. The move begins at position 0 and ends at position 5000. The traverse speed is 200 RPM. ACC and DEC are specified independently before the move is commanded.

S-Curves

The BJx also allows you to specify the type of acceleration you want. You can select S-curve accelerations for smoothness or straight-line accelerations for quickness. The graph in Figure 3.3 shows the profile from Figure 3.2 using S-curves instead of straight lines.

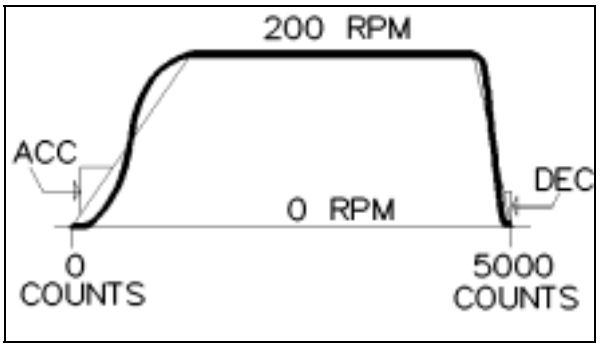


Figure 3.3. S-Curve Profile

Notice that ACC and DEC are still independent. Notice also that they specify the *average* acceleration, not the peak. Since S-curves reduce the acceleration rate at the endpoints of the acceleration, the acceleration rate in the middle must increase. Typically, when you switch to S-curves, you must reduce ACC and DEC to stay within the ratings of the motor. However, since S-curves reduce overshoot, you may find that you increase the overall acceleration rate when you use them.

NOTE

You may need to reduce ACC and DEC when using S-curves.

For some applications, S-curves can reduce the average acceleration too much; in others, straight line acceleration produces motion that jerks the motor excessively. The BJx provides different levels of S-curves, allowing you to make the trade-off. There are three levels that are selected by setting the variable SCRV to either 1, 2, or 3. For more information on S-curves, see Industrial Drives application note B101, "Acceleration Profiles."

Table 3.5 S-Curve Acceleration Chart

For this acceleration...	Set SCRV to...
Straight-line	1
Polynomial	2
Sinusoid	3

Move Absolute (MA) Command

There are two kinds of simple moves: absolute and incremental. With absolute moves, you specify the end position; with incremental moves, you specify the total distance of the move.

The MA command allows you to command absolute moves by specifying the end position. ACC, DEC, and SCRV are all in effect for MA moves. As an option, you can specify the traverse speed. For example,

```
MA  50000  1000
```

moves to position 50,000 at a peak speed of 1000 RPM.

NOTE

Not specifying the speed in MA commands reduces execution time.

Move Incremental (MI) Command

The MI command allows you to command incremental moves by specifying the total distance of the move. ACC, DEC, and SCRV are all in effect for MI moves. For example,

```
MI  5000  200
```

causes the motor to move 5000 counts at a peak speed of 200 RPM. The profiles that were shown earlier in Figures 3.2 and 3.3 could have been generated from this example.

Incremental Move Example

SHOCK HAZARD!

Large voltages from the AC line and the DC bus can cause injury. Wire the BJx as described in the *BJx Installation* manual.

THE MOTOR MAY MOVE UNEXPECTEDLY!

BE PREPARED TO REMOVE POWER FROM THE BJx!

Complete "Initial Check-Out" in the *BJx Installation* manual before continuing.

This section will enable the BJx. The system may be unstable. The motor may begin oscillating or run away. Be prepared to remove power quickly.

Turn on the AC line voltage. Type in the following example:

```
EN
ACC  1000
DEC  1000
MI  4000  100
```

Profile Limits

With both the MA and MI commands, if the traverse speed cannot be reached because ACC or DEC is too small for the specified move, then the BJx reduces the maximum speed so that the move, for all practical purposes, is triangular. Actually, there is a very short (less than 5 milliseconds) traverse segment so that the move still has three segments.

The maximum time for an entire move is not limited. However, the time for each acceleration or deceleration is limited to 30 seconds. If the acceleration rate is so low that this limit is exceeded, then the BJx generates an error explaining that either ACC or DEC is too low. This error is issued before the motion command begins. In this case, ACC or DEC must be increased, or the peak speed of the move must be decreased.

Multiple Profile Commands

The BJx allows one succeeding move to be calculated while the present move is being executed. This is called buffering. Buffering reduces inter-index delay, the delay between successive moves, almost to zero. When you are commanding motion from the Interactive mode (-->), be careful not to type in two move commands while another is executing. This generates an error. If you are commanding motion from your program, the BJx automatically pauses before calculating a third motion profile, thus stopping this error from occurring.

Profile Final Position, PFNL

If you want to keep track of the end position of the present move, the variable PFNL (Position Final) is provided. This variable contains the final position of a move. The variable can be used to compute the distance remaining by combining it with PFB (Position Feedback):

```
P  "DISTANCE TO GO "  PFNL-PFB
                ;PRINT THE AMOUNT OF
                ;POSITION TO GO TO
                ;FINISH THE MOVE
```

JOG (J) Command

This section describes J, the JOG command. Jogging is useful when you want to command motion without position endpoints. For example,

```
J  500
```

causes the motor to rotate at 500 RPM indefinitely. Jogs are useful for machine set up and testing.

ACC and DEC are in effect with Jogs, as is SCRV. Software and Hardware Travel Limits are also in effect. Jog is the only move command that can cause motion to change direction without stopping first. However, since changing directions involves both acceleration and deceleration, Jog commands that change direction of rotation use ACC or DEC, whichever is lower. Jog commands should be used with caution, since motion continues indefinitely.

NORMALIZE (NORM) Command

NORM, the NORMALIZE command, is required if you want to reset the BJx position feedback, PFB. Often, you may want to set the position feedback to some known value. For example, on power-up the position feedback is set to zero. After a homing sequence, you may need to reset the position register. This is done using NORM. For example,

```
NORM  10000
```

sets PFB (position feedback) as well as PCMD (POSITION command) to 10,000 in position units. As an alternative, you can enter:

```
PFB=10000
```

Setting PFB has the same effect as the NORM command. Use whichever you think makes your program easier to understand.

Now type in:

```
P  PFB
```

Normalize the position to 1000 with:

```
NORM  1000
```

Again, print PFB:

```
P  PFB
```

and see that it is now 1000. The NORM command cannot be used when GEAR is on or when motion is commanded from MA, MI, or any other motion command.

Gating Motion with GATE

The GATEMODE variable allows you to pre-calculate a profile and begin motion within 1.5 milliseconds of a switch closure. To enable GATE, turn on GATEMODE and follow it with one or two MA or MI commands.

When the hardware input GATE transitions from low to high, motion begins. GATE is on Connector J11. After motion is begun, GATEMODE is turned off. You must re-enable GATEMODE for each move you want gated. Also, you cannot turn GATEMODE on when motion is commanded from Jogs, MA, or MI commands. If you turn GATEMODE on and command motion, but turn GATEMODE off before the GATE input turns on (thus, allowing motion to begin), the commanded motion will be "forgotten" by the BJx.

In the following example, two MI commands are entered and precalculated with GATEMODE on.

```
GATEMODE  ON      ;ENABLE GATING
MI  1000  100     ;PRECALC MOVES.
                  ;MOTION
MI  -1000         ;DELAYED TIL GATE
                  ;IS HIGH
W  0              ;WAIT FOR MOTION
                  ;TO START
```

This means no motion will take place until the hardware input GATE is high. If the above lines were part of a program, the W command would delay program execution until the GATE switch turned on.

Zero Position Error (ZPE) Command

The ZPE command zeros position error by setting PCMD to PFB without changing PFB. There are occasions when this will be necessary. For example, if the BJx is run for some time as a velocity loop, then position error can accumulate well beyond PEMAX. If the position loop is turned on with this condition, a position error overflow error will occur. To prevent the error, you must first zero the position error, then turn the position loop on by entering:

```
ZPE
PL  ON
```

The ZPE command is also frequently used with clamping, which is discussed later in this chapter.

GOHOME Command

GOHOME generates a home sequence that combines position capture with a two-step profile for fast, reliable homing. The format is:

GOHOME Vel1 [Vel2] [Source]

where Vel1 is the first (fast) speed;
Vel2 is the second (slow) speed;
Source selects HOME or INDEX.

Figure 3.4 shows a typical GOHOME sequence.:

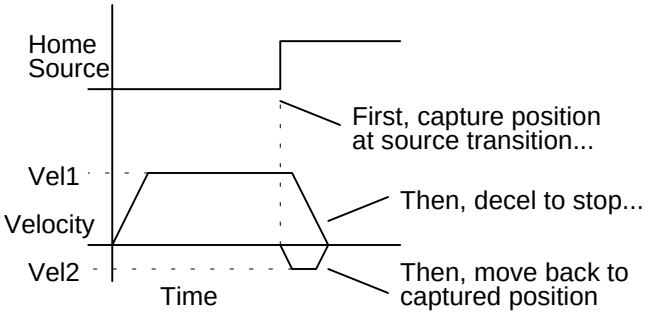


Figure 3.4 GOHOME Command

Homing Velocities: Vel1 and Vel2

GOHOME generates a two-step homing sequence. The first part of the sequence is normally at high speed; the profile accelerates to Vel1 and searches for the first transition on the home source. After capturing position, the profile reverses and moves back to the captured position. If Vel2 is not specified, it is assumed to be 10% of Vel1.

```
GOHOME 2000 150    ;HOME AT 2000 RPM
                   ;THEN REVERSE &
                   ;HOME AT -150 RPM

GOHOME 2000        ;HOME AT 2000 RPM
                   ;THEN REVERSE &
                   ;HOME AT -200 RPM
```

The sign of Vel1 determines the direction of the homing sequence. If Vel1 is positive, the initial direction of homing will be positive and vice versa. The second part of the profile is always opposite of the first; the sign of Vel2 is ignored.

Homing Source

GOHOME allows homing based on either of two inputs: the HOME switch (Connector J17) or the Feedback Encoder Index (J12). Specify either HOME or INDEX. If you do not specify either source, the HOME switch is assumed.

```
GOHOME 200  HOME  ;HOME TO
                  ;HOME SWITCH

GOHOME 200  INDE  ;HOME TO
                  ;INDEX SWITCH

GOHOME 200        ;HOME TO
                  ;HOME SWITCH
```

The home position is captured the first time the source switch changes state. If you want to home to the second change of state, you should jog past the first change prior to issuing the GOHOME command as shown in Figure 3.5.

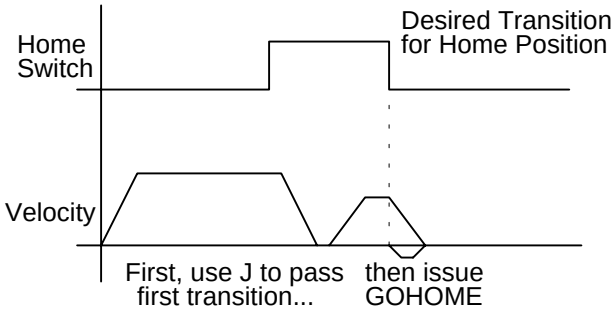


Figure 3.5 GOHOME to 2nd Transition

Capturing Position

Position capture is a feature where the position feedback (PFB) is *captured* when a hardware input transitions. The BJx position capture is accurate to ±40 microseconds. In other words, the position that is stored after a capture is equal to the actual position of the motor at the time of the capture, within 40 microseconds. Capture uses the HOME, INDEX, OR MINDEX hardware input as the capture trigger.

Enabling Capture, CAP, PCAP & PEXTCAP

The switch CAP controls capture. If CAP is on, then capturing is enabled. When capturing is enabled, the BJx will watch one of three inputs: HOME, the feedback encoder index, or the master encoder index. You select the source of the capture with the variable CAPSRC.

When the capture input changes to the state specified by CAPDIR, the BJx will store PFB in the variable PCAP and PEXT in the variable PEXTCAP. After the capture, the BJx turns CAP off. This tells you the capture is complete. You can then use PCAP and PEXTCAP as you would any other monitoring variable. PCAP is in position units. PEXTCAP is in external position units.

Capture Direction, CAPDIR

The capture is triggered when the HOME input changes from 0 to 1, or vice versa. If CAPDIR is 1, the capture occurs when the HOME input changes from 0 to 1. If CAPDIR is 0, the capture occurs when HOME changes from 1 to 0. CAPDIR can be changed at any time. Changing CAPDIR always turns CAP off.

Capture Source, CAPSRC

CAPSRC specifies which capture input to use as the capture trigger source. The possibilities are INDEX input, MINDEX input, and the HOME input.

For this capture source...	Set CAPSRC to...
HOME	**1**
INDEX	**2**
MINDEX	**3**

Clamping

Clamping stops BJx motion when the position error exceeds a set point. This is used to determine that the motor, usually through a lead screw, has run a part into a mechanical stop. The profile stops and the part is held with limited torque. This is sometimes referred to as "Feed to Positive Stop." The stop is detected by watching position error; when position error exceeds the variable PECLAMP, the part is assumed to have run into a stop. When a stop has been detected, the BJx will hold the current at ILIM, which should be set to the proper holding current. ILIM can be increased or decreased after the stop has been detected. To enable clamping, turn CLAMP on. PECLAMP can be changed at any time.

In general, clamping is done at low speeds with the current limited to some low level. After the clamp has occurred, the motor is assumed to be at zero speed. When the clamp has occurred, you can raise or lower ILIM to set the holding torque as desired. You can tell whether a clamp has occurred by looking at SEG, the present motion segment. If SEG is 0, then motion has stopped.

After the BJx stops motion, the position error stays at approximately PECLAMP. Before commanding any new motion, you should zero the position error with the ZPE command.

Clamping can be used with all move and jog commands. If jogs are used, the motion continues until the stop is found. If move commands are used, then motion does not continue past the specified endpoint, regardless of whether a part is found.

An example of clamping follows:

```
PECLAMP=1000        ;SET CLAMP  = 1000
                    ;POS UNITS
CLAMP  ON           ;ENABLE
                    ;CLAMPING MODE
MA  100000  400     ;MOVE AT MOST
                    ;100000 POS UNITS
                    ;IF THE MOTOR
                    ;GETS ALL THE
                    ;WAY TO 100000,
                    ;THEN THE STOP
                    ;WAS NOT
                    ;ENCOUNTERED.
                    ;ASSUMED THE
                    ;PART IS NOT
                    ;THERE.

W 0                 ;DELAY UNTIL
                    ;MOTION STOPS

IF  PCMD  EQ  100000  P  "PART NOT
FOUND"
                    ;IF PCMD = 100000 =
                    ;FINAL POSITION,
                    ;THEN THE PART
                    ;WAS NOT FOUND.
```

JOG TO (JT) & JOG FROM (JF)

In some applications, JOG commands need to be synchronized with position feedback. With J, the standard JOG command, the speed changes when the command is entered. Position dependent jogs (Jog To and Jog From) delay the speed change until a specified position is reached. You specify the position at which the change in speed begins with the Jog From (JF) command. Similarly, you specify the position at which the change in speed ends with the Jog To (JT) command.

With position dependent jogs, you must specify a position and the new speed. ACC, DEC, and SCRV are in effect. Position dependent jogs are always absolute moves (not incremental).

Figure 3.6 shows the effect of a JF command. This example assumes that the speed is already 2000 RPM when the JF command is executed.

```
;ASSUME PRESENT SPEED IS 2000 RPM
JF  50000  500
```

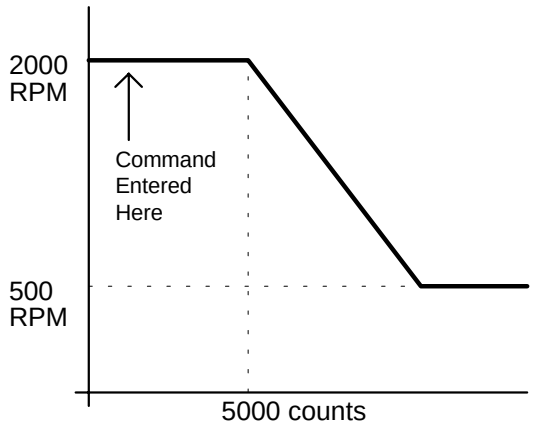


Figure 3.6 Jog From (JF) Command

Figure 3.7 shows the effect of the JT command. This example also assumes that the speed is 2000 RPM when the command is executed.

```
;ASSUME PRESENT SPEED IS 2000 RPM
JT  50000  500
```

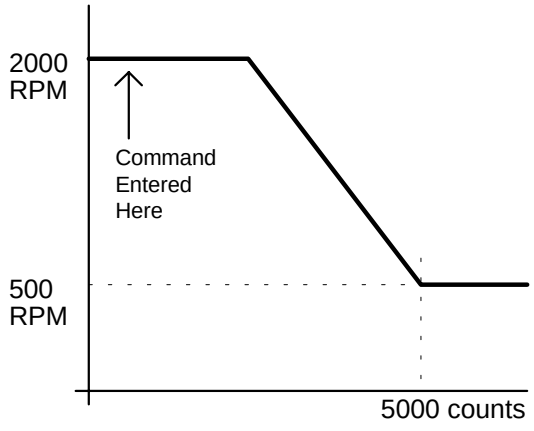


Figure 3.7 Jog To (JT) Command

Position dependent commands must be used with care. If you specify a position that has already passed, the BJx will generate ERROR 42, "MOVE W/O TIME." Also, if the Jog To command is given so that ACC or DEC prohibits the profile from reaching final speed before the specified position, the BJx will generate ERROR 42. ERROR 41, "MOVE NEEDS MOTION," is generated if Jog To or Jog From are commanded when the velocity is 0. Finally, a position dependent jog that attempts to change the direction of rotation will generate an error. All of these errors stop motion.

Registration

The BJx allows you to combine the position capture with the Jog To command to implement index-to-registration. One example of index-to-registration is a conveyor belt on which items are placed in random positions. An optical sensor detects the item upstream of the operation. The BJx, controlling the conveyor, continues at full speed and stops the item where the operation will take place. The high-speed position capture works at all velocities and during accelerations; it is accurate to 40 microseconds.

To implement index-to-registration, you usually jog the motor at a constant speed, capture the position (with the registration device connected to the HOME input), then use the Jog To command to stop the motor at an endpoint (normally a specified distance beyond the registration input).

Registration Example

The following example shows how to program the BJx for registration. The desired operation of the program is as follows:

1. Set CAPDIR (1 for low-to-high transition, 0 for high-to-low transition).
2. Enable capturing.
3. Begin move.
4. Wait for the BJx to capture.
5. Use the captured position to set the endpoint of the move.

For example, the following code segment jogs at 2000 RPM and stops 4000 counts after the registration input transitions from low to high.

```
CAPDIR  1              ;SET CAPDIR FOR
                       ;LOW TO HIGH
CAP  ON                ;ENABLE CAPTURE
J  2000                ;BEGIN MOVE
TIL  CAP  EQ  0        ;WAIT FOR
                       ;POSITION
                       ;CAPTURE
JT  PCAP+4000  0
```

Note that the motor comes to rest 4000 counts after the position that was captured, not 4000 counts after the JT command is executed. If 4000 counts was not enough distance, ERROR 42, "MOVE W/O TIME," would be generated. This means that the commanded speed change cannot be accomplished given DEC, the deceleration limit. Note also that you must leave an additional 10-15 milliseconds for the TIL (see Chapter 5) and JT commands to be executed.

The JT command example given here brings the system to rest. As an alternative, you can change the speed to any value the motor can run, as long as you do not attempt to change direction with one JT command. For example, the following command replaces the above JT command when you want to change speed to 100 RPM at 4000 counts past PCAP.

```
JT  PCAP+4000  100
                ;CHANGE SPEED TO
                ;100 RPM.  BEGIN
                ;DECEL SO THE
                ;SPEED IS JUST
                ;REACHING 100 RPM
                ;WHEN THE POSITION IS
                ;4000 COUNTS PAST
                ;REGISTRATION MARK
```

For more information about registration, see Industrial Drives application note "Cut to Length."

Multiple JF/JT Commands

Many applications require that multiple Jog From (JF) and Jog To (JT) commands be executed sequentially. In most cases, you will have to insert a delay in your program between JT and JF commands. For example, if you enter the following program, you might think the motor will first jog to 100 RPM, then to 400 RPM (at 20,000 counts), and finally come to rest at 30,000 counts. Actually, the motor will jog to about 40 RPM and continue at that speed until it comes to rest at 30,000 counts. This is because the JF/JT commands cause the motion profile to hold the velocity command constant, even if an acceleration is commanded from the previous motion command.

```
55$
EN                     ;ENABLE BJX
ACC  100000            ;SET ACCEL AND
                       ;DECEL RATES
DEC  100000
NORM  0                ;NORMALIZE TO
                       ;ZERO POSITION
J  100                 ;JOG TO 100 RPM
JT  20000  400         ;ERROR--SHOULD
                       ;DELAY TIL SPEED
                       ;REACHES 100 RPM
                       ;BEFORE
                       ;EXECUTING JT
                       ;COMMAND.
JT  30000  0           ;ERROR--SHOULD
                       ;DELAY TIL SPEED
                       ;REACHES 400 RPM
                       ;BEFORE
                       ;EXECUTING JF
                       ;COMMAND.
DIS
B
```

The solution is to insert delays to force the program to wait until the motor reaches the final speed from the previous motion command. For example, the above program can be modified as follows.

```
55$
EN                     ;ENABLE BJX
ACC  100000            ;SET ACCEL AND
                       ;DECEL RATES
DEC  100000
NORM  0                ;NORMALIZE TO
                       ;ZERO POSITION
J  100                 ;JOG TO 100 RPM
TIL  VCMD  EQ  100     ;WAIT TIL SPEED
                       ;REACHES 100 RPM
JT  20000  400         ;EXECUTE JT
                       ;COMMAND
TIL  VCMD  EQ  400     ;WAIT TIL SPEED
                       ;REACHES 400 RPM
JT  30000  0           ;EXECUTE JT
                       ;COMMAND
                       ;DIS
B
```

Although delays with the TIL command work, delays usually should be inserted with the WAIT (W) command. The WAIT (W) command takes less space and works better with multi-tasking, a subject discussed in Chapter 5. For our example, the first TIL command can be replaced with "W 2" and the second can be replaced with "W 3."

Changing Profiles During Motion

Position dependent jogs can also be used to change the speed or endpoints of an MA or MI command that is already in progress. For example, if you want to change the speed of a profile depending on an input, you could write the following program to reduce the speed when I1 is 1.

```
X1 = 10000              ;X1 STORES THE
                        ;ENDPOINT
MA  X1  5000            ;BEGIN AT 5000
                        ;RPM
TIL SEG EQ 0 GOSUB 25
                        ;25$ WATCHES I1
                        ;TO CHANGE
                        ;SPEED
B

25$
? I1  EQ  0  RET        ;CHANGE ONLY IF
                        ;I1 = 1
J  1000                 ;REDUCE SPEED
                        ;TO 1000 RPM
TIL  SEG  EQ  2         ;WAIT UNTIL SPEED
                        ;IS 1000 RPM
JT  X1  0               ;USE JT TO GET TO
                        ;ORIGINAL
                        ;ENDPOINT AT NEW
                        ;SPEED
TIL  SEG  EQ  0         ;WAIT FOR MOTION
                        ;TO STOP
B                       ;DONE
```

You must be careful not to begin the motion too late in the profile. For example, suppose I1 became 1 after the profile was well into deceleration, and the speed was, say 200 RPM. In this case, the JT command would generate an error because by the time it was executed, the motor position would be past X1, the original endpoint. This is because the unit would accelerate up to 1000 RPM before the JT command was executed. In general, you must limit the time during which you are looking for the speed change. After this point, the profile must either continue along the original profile or the endpoint must be extended. For example, the program section beginning at label 25 could be rewritten so that it watched a position trip point, X1-2000.

```
25$
? PFB  GT  X1-2000  RET     ;DO NOT
                            ;REDUCE
                            ;SPEED IF
                            ;PFB >
                            ;SETPOINT
;
;REST OF 25$ PROGRAM THE SAME
;
```

What value to use for the setpoint varies from one application to another. These values must be set based on experience. In many applications, the input will not request a speed reduction near an endpoint, so this may not be a problem.

Motion Segments

All moves and jogs occur in segments. Normal jogs have two segments: accel/decel and traverse. Moves (MI and MA) have three segments: accel, traverse, and decel. Position dependent jogs have three segments: traverse to position, accel/decel, and traverse. Table 3.6 shows the different segments for BJx moves.

Table 3.6 Segments for Different Moves

Segment	MI,MA	J	JT/JF
1	Accel	Accel/Decel	Traverse
2	Traverse	Traverse	Accel/Decel
3	Decel	N.A.	Traverse

You can use the SEG to determine when motion is complete, since SEG is zero when the BJx is not commanding a profile.

CONTROL LOOPS

Four sections of control loops are of interest: input, output, feedback, and tuning variables. The input is compared to the feedback to generate an error. The error signal is modified using the tuning variables to generate the output. The tuning variables can be modified to produce higher levels of performance; unfortunately, higher performance brings with it greater noise susceptibility and reduced stability. The system designer must optimize noise and performance for the application.

BJx control loops have one or two tuning variables. All BJx loops follow the convention that larger

constants provide higher gain. Each BJx loop is described below and shown in Figure 3.8.

Position Loop

The Position Loop input is the variable PCMD, the Position command. The feedback is PFB, the position feedback. The output is VCMD, Velocity command, and its two tuning variables are KP, the position loop gain, and KF, the position loop feed-forward gain.

The position loop calculates the position error (PE) as the difference of PCMD and PFB. As a secondary command source, PCMD is differentiated (d/dt)PCMD. The position loop then performs the following calculations:

VCMD=KP * PE + KF * (d/dt)PCMD.

The position loop is optional. If the switch PL is on, then the position loop is enabled; if it is off, then the position loop is bypassed. PL is turned on at power-up.

The feed-forward gain reduces position error at high speed. Without feed-forward, the velocity command is generated only from position error; a large position error is required to command a high speed. If KF is large enough, then a high velocity command can be generated with little or no position error. The BJx scales KF so that unity feed-forward occurs when KF equals 16,384. In other words, if KF is 16,384, no position error is required to generate the velocity command in steady-state running conditions. Larger KF makes the system more responsive to commands; however, KF should never be larger than 16,384.

Unfortunately, large values of KF cause overshoot. KP must be reduced to reduce overshoot. If you need to minimize position error when the motor is turning, you will need to optimize KF and KP. Typically, KF ranges from 2000 to 10,000.

TQ (Torque mode) should be off when PL is turned on. The system becomes unstable when PL and TQ are both on. If you do not turn TQ off before turning PL on, the BJx will force TQ off.

NOTE

When PL is turned on, TQ is turned off automatically.

Velocity Loop

The velocity loop takes its input from the position loop if PL is on. If PL is off, motion commands directly control the velocity command (VCMD). The feedback is VFB, velocity feedback, and the difference of these two signals is VE, velocity error. Velocity error can be used in two control loops: proportional and integrating.

Proportional Velocity Loop

If a proportional velocity loop is selected, then the velocity error is multiplied by KPROP, the proportional constant, to generate ICMD, the current command. Proportional velocity loop is selected when the PROP switch is on. PROP is turned off on power-up.

Proportional velocity loops are much easier to stabilize than integrating loops, so they are often used during machine setup. However, they also allow steady-state velocity error and, therefore, they are generally replaced with integrating loops when the machine is fully operational.

Integrating Velocity Loop

If an integrating velocity loop is selected, then the velocity error is integrated and multiplied by KVI, the velocity integration constant. Velocity feedback is subtracted from this signal, then the signal is multiplied by KV, the velocity loop gain, to form ICMD. This velocity loop is selected when PROP is off.

Torque Command

In a few applications, the BJx is given a "torque" command. Actually, this is a current command, but at lower speeds, motor torque is approximately proportional to current. In this case, VCMD is multiplied by KPROP to form ICMD. Note that this differs from the proportional velocity loop only in that VFB is not subtracted from VCMD. The switch TQ must be on to select the torque mode and off for all other modes. The position loop should be off (PL off) when the BJx is running in Torque command mode. The BJx will turn PL off when TQ is turned on.

NOTE

When TQ is turned on, PL is forced off.

Power-Up Control Loops

The BJx has, at power-up, the following settings:

- Position loop enabled (PL on).
- No feed-forward (KF=0)
- Integrating Velocity Loop (PROP off, TQ off)

These settings meet the requirements of a large number of applications. Figure 3.8 shows each of the five BJx controller modes.

UNITS

The BJx provides user units for the convenience of the operator and the programmer. You can define the units of acceleration, current, velocity, and position for your machine. Also, if your BJx has an external input (See Chapter 4), you can define independent units of external position and external velocity.

The BJx uses internal units which are designed for efficient computer processing. User-unit constants scale the BJx internal units to more convenient units. For example, if you type:

```
VOSPD = 1000
```

the 1000 is multiplied by VNUM/VDEN before it is stored in the BJx memory.

With a simple, step-by-step procedure, you can define your units as RPM, inches/minutes, degrees/second, or any other units that are convenient.

Current Units

The BJx commands current with a digital-to-analog converter (DAC). The BJx internal current unit is 1/4095th of full-scale current. (Full-scale current refers to the peak rating of your BJx, not the continuous rating. For example, the peak rating of a BJR-4004 is 8 Amps.)

The conversion constants that determine user current units are INUM, current units numerator, and IDEN, current units denominator, as shown in Figure 3.9. Notice that IDEN and INUM switch places depending on whether you are setting or reading the variable.

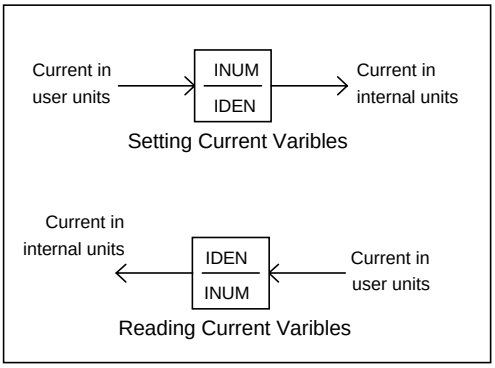


Figure 3.9 BJx Current Units

INUM and IDEN have a range of 0 to 2^{31}. For standard current units (percent), INUM is 4095 and IDEN is 100. For example, when setting ILIM to 100, type

```
ILIM=100                ;SET ILIM TO 100%
```

The BJx converts the 100% to 4095 BJx internal units as shown in Figure 3.10.

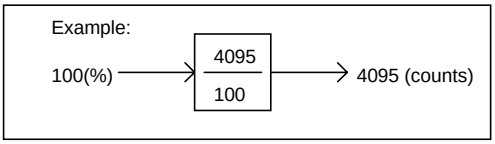


Figure 3.10 BJx Current Units Example

This sets ILIM to 4095 or 100% of full current. When you type:

```
P  ILIM
```

the BJx converts the 4095 BJx-internal units to 100% by multiplying by IDEN and dividing by INUM.

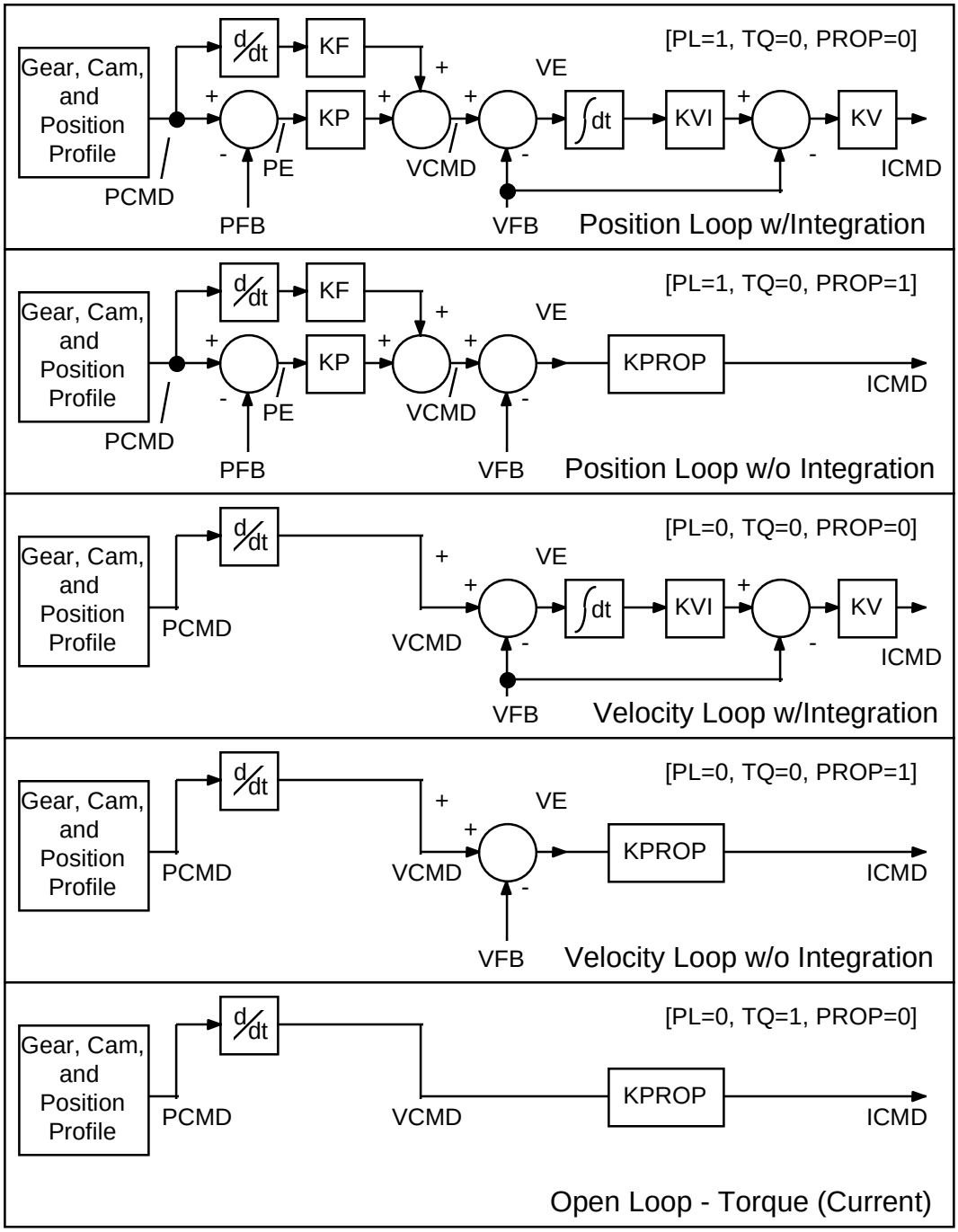


Figure 3.8
BJx Control Modes

Setting Variables

All variables that have units associated with them should be set after you have specified the user units. This is because the values actually stored in the variables are in BJx-internal units, not user units. Changing the user units will not affect the fundamental value stored in the variables.

NOTE

Always set unit-based variables after the units are defined in the BJx.

For example, if you want VOSPD to be 100 inches/minute and you type:

```
VOSPD = 100
```

when velocity units are in RPM, VOSPD would be 100 RPM. If you change the velocity units to inches/minute, VOSPD would remain 100 RPM--it would simply be converted to the equivalent of 100 RPM in inches/minute. Refer to Appendix E, which lists all variables and the units associated with them.

Position Units

Standard units for position are counts. To select counts as units, set PNUM and PDEN to 1. To select other units, scale counts using PNUM and PDEN as a fractional multiplier. For example, suppose you want units to be revolutions. For this example, assume a 1000-line encoder or, with quadrature, 4000 counts per revolution. Now use PNUM/PDEN to form a scale factor of 4000:

```
PNUM = 4000
PDEN  =  1
```

For most applications, revolutions are far too coarse. Suppose we select degrees; here, we want to scale by 4000/360. Since the BJx language is integer based, we can't set PNUM to 4000/360 = 11.1111^{-}, at least not very accurately. Using PNUM and PDEN we can enter the scale factor as an exact ratio:

```
PNUM = 4000
PDEN  = 360
```

The exact representation is important. As we will see in the section concerning PROTARY, PNUM, and PDEN, in rotary-table application, even the smallest inaccuracy will accumulate substantial error after many revolutions.

In general, you are free to select your units. Be those units millimeters, thousands of inches, or arc-minutes, you need only:

1. Determine the scale factor from counts.
2. Separate the scale factor into an integer numerator and denominator.
3. Set PNUM and PDEN accordingly.

One subtlety that can cause confusion is determining how to divide the scale factor into PNUM and PDEN. Normally, you will select units less resolved than counts. In that case, PNUM will be larger than PDEN. For example, changing units from counts to revolutions (decreasing resolution) will increase PNUM relative to PDEN. Also, as you may have noticed, although the ratio PNUM/PDEN is determined by your application, the individual values are somewhat arbitrary. For example, the following sets of PNUM/PDEN produce identical results:

Set 1	PNUM = 400	PDEN = 36
Set 2	PNUM = 4000	PDEN = 360
Set 3	PNUM = 40,000	PDEN = 3600

The only limitation is that both numbers must be less than 2^{31}. However, better accuracy is maintained if PNUM and PDEN are less than 2^{15}

Velocity Units

The standard velocity unit is RPM. The following table shows how you set VNUM and VDEN for a given encoder resolution.

Table 3.7 Standard Velocity Units (RPM)

Units	Encoder Resolution	VNUM	VDEN
	500	21,845	10
RPM	1000	43,691	10
	2000	87,382	10
	(Other)η	η∗43.691	10

Note that when VDEN is fixed, VNUM is proportional to encoder resolution. To calculate units other than RPM, first determine the scale between RPM and the desired units, then adjust VNUM and VDEN according to the scale factor.

As with PNUM/PDEN, decreasing the resolution requires increases in the ratio VNUM/VDEN. However, it is a bit more complex with velocity units because VNUM and VDEN are not 1 as were PNUM and PDEN.

Accelerator Units

The standard unit for acceleration is RPM/second. The following table shows how to set ANUM and ADEN for a given encoder resolution.

Table 3.8 Standard English Acceleration Units (RPM)

Unit	Encoder Resolution	ANUM	ADEN
RPM/sec	500	21,845	10,000
	1000	43,691	10,000
	2000	87,382	10,000
	Other (η)	η*43.691	10,000

External Units

External units are for the external pulse inputs (Connector J13), VEXT and PEXT. The user units are set by VXNUM and VXDEN for external velocity (VEXT) and by PXNUM and PXDEN for external position (PEXT). Define external units the same way you defined position and velocity units.

Metric Units

To convert Tables 3.7 and 3.8 to metric units (rad/s and rad/s^2), multiply VNUM and ANUM by 9.55.

Indirectly Coupled Feedback

If the feedback encoder is belted or geared to the motor, the belt or gear ratio must be accounted for. Do this by including that ratio in the encoder resolution.

For example, if a 1000-line encoder is connected to the motor through a 3:1 (reducing) gear ratio, the theoretical encoder resolution is 3000 lines per revolution.

Example of Machine Specific Units

An example of a lead-screw will illustrate machine specific units:

Encoder resolution 1000 lines (4000 counts)
Lead Screw Pitch 10 revs/in
Desired Units:
Position mils
Velocity mils/min
1 mil = 40 counts
100 mil/minute = 1 RPM

Table 3.9 Scaling Units Example

	Standard	Scale	Resolution	Units
Position	PNUM=1 PDEN=1	40 cts=1 mil	decrease	PNUM=40 PDEN=1
Velocity	VNUM = 43,691 VDEN = 10	1RPM=100 mil/min	increase	VNUM = 437 VDEN = 10

Position Rotary Mode, ROTARY, & PROTARY

The BJx stores position in a 32-bit number. This number is large enough to count many revolutions. For example, the 32-bit number will store the counts from a 1000-line encoder for about 10 million revolutions before the 32-bit limit is exceeded. Although this is large, it may not be large enough. Some applications require the motor to rotate in one direction indefinitely. Eventually, the 32-bit limit will be exceeded, resulting in an error. The *Rotary* mode allows the BJx to support these unidirectional applications.

The Rotary mode forces all position-related variables to "roll-over" after position feedback (PFB) exceeds a specified limit. The variables that are rolled over are PFB, PCMD, and PFNL. The rotary distance (the specified limit before roll-over) is stored in PROTARY. PROTARY is in position units.

When ROTARY is on, the Rotary mode is enabled. If PFB is greater than PROTARY, then PFB, PCMD, and PFNL are decremented by PROTARY. If PFB is less than zero, then PFB, PCMD, and PFNL are incremented by PROTARY. Note that DIR=0 does not work well with the Rotary mode as PCMD, PFB, and PFNL are always less than zero. You cannot change PNUM, PDEN, or PROTARY when

ROTARY is ON. In addition, you must normalize PFB so that 0 ≤ PFB ≤ PROTARY before turning ROTARY ON. Enable the Rotary mode by typing:

```
ROTARY  ON
```

Choosing PROTARY, PNUM, and PDEN

If you have a rotary application such as a printing drum, set PROTARY in position user units to be the exact equivalent of one revolution of the drum. PROTARY must be exact or position error will accumulate over many revolutions. For example, suppose the motor of an application is connected through a 5:3 gearbox. For convenience, assume the user units are in degrees of the table. PROTARY would be one revolution of the table or 360 degrees. How do you select PNUM, PDEN, and PROTARY?

The key is selecting PNUM and PDEN so that PROTARY can be represented exactly as an integer. This does not mean that PROTARY must be an integer number of counts. In fact, it normally will not be. Returning to the example, a motor movement of 5 revolutions would cause 3 revolutions of machine (table) rotation, or 1080 user units (degrees).

PNUM = 4096 * 5 and PDEN = 360 * 3

Thus, PROTARY would be 360. Notice that PROTARY is not exact in counts; it is 5/3 of a revolution or 6826 and 2/3 counts. However, it is exact in user units. Therefore, error will not accumulate as the table rotates. The incorrect way to choose PNUM, PDEN, and PROTARY would be to select PNUM and PDEN so that PROTARY could not be represented as an integer. For example, we could have stated that 5/3 revolution of the motor would cause one revolution of the machine. Then:

PNUM = INT(4096 * 5/3) = 6827
PDEN = 360

In this case, PROTARY would be 359.98 (not 360) degrees, so that error would accumulate as the table turned.

NOTE

Position Units must be exact in the rotary mode to prevent error accumulation.

Rotary Mode and Absolute Moves

When the BJx is in the Rotary mode, you must limit the final position of all absolute moves to between 0 and PROTARY. If you want to move more than PROTARY, you can use incremental moves. For example, the following is a legal command.

```
MI  50*PROTARY
```

SERIAL COMMUNICATIONS

This section discusses details of BJx serial communications. This includes autobauding, multidrop connections, and transferring your program to and from the BJx. If you are using Motion Link, the Kollmorgen software package for the BJx, you do not need to read the sections on transmitting and receiving your program, or on system dump. Motion Link provides facilities for these functions.

Autobauding

It is not necessary to set the baud rate on the BJx directly. The BJx can determine the current baud rate and set its own baud rate accordingly. This is called autobauding. After the BJx determines the correct baud rate, it will store this rate away in the variable BAUD. The BJx will flash the FAULT LED to indicate that it is autobauding. At this point, the BJx is waiting for a few carriage returns. Motion Link automatcally sends carriage returns if you select "AUTOBAUD" at intialization. If you are using a terminal, press the enter key several times.

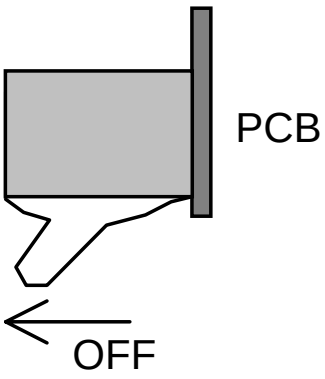
Setting the BJx to Autobaud

There are two ways for the BJx to autobaud at power-up:

1. Powering-up with the Autobaud switch on.
2. Setting the value of the variable BAUD to an invalid value (say, 1000) before the next power-up.

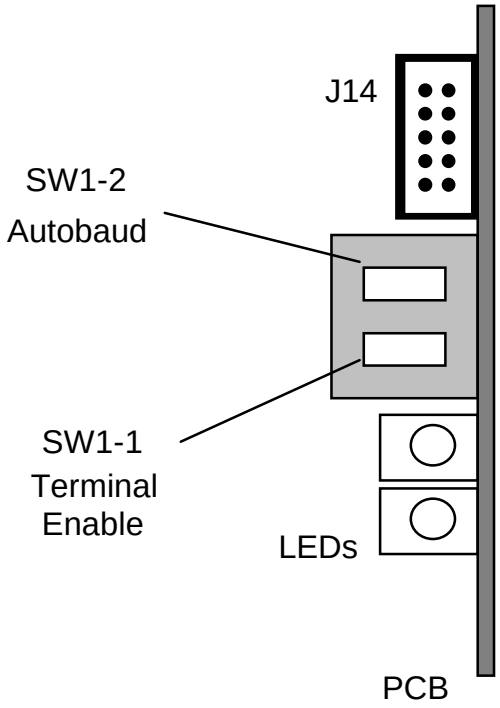
ABAUD: The Autobaud Switch

The Autobaud switch on the front of the BJx (SW1-2) sets the unit to autobaud. Note that this also sets ADDR to zero. Normally, you will need to enable the terminal as well. To do both, set switches SW1-1 and SW1-2 ON (move the handles to near the PCB.).



SW1
Top View

SW1 has two switches, SW1-1 (Terminal Enable) and SW-2 (Autobaud). SW1-1 is nearest the diodes; SW1-2 is next to J14. One way to remember is that SW1-1 is the closer of the two to J15 and enables and disables J15.



SW1
Front View

If you do not want your BJx to autobaud when the unit is powered-up, then turn ABAUD off. This is important if you want the BJx to run the Power-Up Label (POWER-UP$), because if ABAUD is on, the BJx will not execute the program until communications have been established.

Baud Rate, BAUD

If ABAUD is off, then the system will check the variable BAUD for the desired baud rate. If it is not a valid baud rate, the BJx will autobaud. After a successful autobaud, an error is generated indicating that the baud rate was out-of-range on power-up.

Prompts

The BJx issues a prompt when it is ready to receive a new command. The BJx allows you to suppress the prompt characters by typing:

```
PROMPT OFF
```

PROMPT is turned on at power-up. Prompts are particularly important when communicating with computers, since the computer that is transmitting to the BJx must wait for a prompt before beginning a new line. After the prompt is received, the computer can transmit at the full baud rate, without inserting delays.

Serial Watchdog

The BJx provides a serial watchdog timer for applications where a command should be received from a computer on a regular basis. If a complete command is not received from the serial port in the specified time, an error will be generated that will disable the BJx and break the user program.

The serial watchdog enhances safety in some applications since it will normally disable the BJx if the communications line breaks. The serial watchdog waits for a carriage return to signify a completed command. It does not test the validity of the command. For example, if your computer fails and begins sending random carriage returns, the serial watchdog will not generate an error.

WARNING

The serial watchdog is not designed for use as a safety device.

Do not use the serial watchdog to prevent personal injury.

The BJx serial watchdog is intended to detect a broken serial communications line. It does not test the validity of data received from your computer.

Set WTIME in milliseconds to the time that you want the serial watchdog to timeout. To enable the serial watchdog, type:

WATCH ON

Serial Checksum

Serial checksum provides error checking of commands transmitted to the BJx through the serial port. This communication mode requires all commands be appended with a two character a suffix which represents an 8-bit checksum. Commands with correct checksums are acknowledged with the "ACK" character (06H) and executed. Commands with incorrect or missing checksums are acknowledged with the "NACK" character (15H) and are not executed.

Calculating the Serial Checksum

To calculate the serial checksum, follow this procedure:

1. Sum the values of the string. Note that Appendix B contains ASCII values for all BJx characters.

   Example "SCKSUM=0"

Char.	ASCII Value
S	83
C	67
K	75
S	83
U	85
M	77
=	61
0	48
TOTAL	579

2. Convert the total to hex. Continuing our example,

   579 Decimal = 2 x 256 +
   4 x 16 +
   3 x 1
   = 243H

3. Take the two least-significant (LS) bytes: Two LS-Bytes (243H) = 43H.

4. Append the two ASCII characters to the command string. For our example, append 43:

SCKSUM=043

Note that serial checksum is case sensitive. That is, using lower-case characters in commands changes the value of the checksum. Note also that the carriage return which terminates the serial command is not included in checksum calculations. Finally, note that backspace is processed before the serial checksum is calculated. In other words, in the command

SCKSUMQ<BACKSPACE>=043

the value of 43 is still correct.

Enabling/Disabling Serial Checksum

To enable serial checksum, type:

SCKSUM=1

To disable serial checksum, type

SCKSUM=043

Note that the 43 is the checksum for "SCKSUM=0" and must be appended for the command to be accepted.

Echo and Prompt

The BJx response to commands with serial checksum enabled is

<Command (Echoed)>
<ACK or NACK><CR><LF><Prompt>

To reduce communication time, you can disable echoing by typing

ECHO=0

(Note: the line as shown is entered before enabling serial checksum. Otherwise, append the checksum.)

Now the BJx response is:

<ACK or NACK><Prompt>

You can also disable the prompt by typing:

```
PROMPT=0
```

Now the only response to serial commands will be the <ACK> or <NACK> characters (06H or 15H).

Transmitting the Program

Two other serial commands are the >BDS and <BDS for transmitting the user program directly to and from the BJx (i.e., without using Motion Link). These commands are documented in Chapter 5.

RECORD AND PLAY

The RECORD command allows you to record most BJx variables in real time for later playback. You can simultaneously record up to four variables. You can record any variable except PE, TMR1, TMR2, TMR3, TMR4, VAVG, VXAVG, or any user switches. You can specify the time between points from one millisecond to one minute. You can record up to 1000 instances of 1 variable, 500 instances of 2 variables, 333 instances of 3, and 250 instances of 4 variables.

The format of the RECORD command is:

RECORD <Number> <Time> <1 to 4 Variables>

where Number is the number of intervals over which the variables will be recorded, and Time is the time in milliseconds of each interval.

Note: <Number> <= 1000 for 1 Variable
<Number> <= 500 for 2 Variables
<Number> <= 333 for 3 Variables
<Number> <= 250 for 4 Variables

For example,

```
405$                    ;BEGINNING LABEL
EN                      ;ENABLE BJX
RECORD 500 1 VFB        ;RECORD VFB FOR
J  1000                 ;1/2 SECOND JOG
B                       ;1000 RPM
```

Records the velocity response of the BJx to a jog command.

After data is recorded, you can use the PLAY command to print each point on the screen. However, Motion Link provides all the routines to retrieve, plot, print, and store recorded data on your computer and line printer.

The RECORD command is useful when tuning a system because you can display the BJx response to commands without an oscilloscope. However, it is not limited to tuning. For example, you can record VCMD to plot a motion profile, or you can plot VEXT to watch the master encoder. You can also plot user variables to watch the performance of your program.

System Dump

The BJx can transmit all variables in addition to the user program. This is called a system dump, and you request it with the DUMP command. For example, type:

```
DUMP
```

and the BJx will provide pages of information including the program, all BJx variables, user variables, and user switches. This also includes all protected variables.

The system dump is provided so that the information from the dump can be directly re-transmitted to any BJx. This changes all *NON-PROTECTED* variables. The DUMP command precedes protected variables with a semicolon (;). This makes the line a comment so that when the line is re-transmitted, it has no effect. If the ";" were not there, re-transmitting the dump information would generate an error when a protected variable was changed. Every line of the user program is preceded with a semicolon for the same reason.

Version Dump

Your BJx will print out its firmware version at any time with the DUMP VERSION command:

```
DUMP  VERSION
```

Multidrop Communications

Multidrop communication allows you to have many (up to 32) axes on one serial line. When the BJx is in Multidrop mode, each axis must have a unique address. This address is a prefix on all communications to and from the BJx. The address is

stored in variable ADDR. ADDR is set to 0 for standard (single-drop) communications. Valid addresses are 48 (ASCII '0') through 57 (ASCII '9') and 65 (ASCII 'A') through 90 (ASCII 'Z') (see Appendix B). Note that the address must be set before multiple units are connected to the same serial line.

Assuming that autobauding is turned off, when the BJx powers-up in Multidrop mode it is "asleep." When asleep, the BJx continues to execute programs and control the motor, but it does not communicate over the serial line. The BJx executes commands that normally print to the serial port (P, PS, R, RS, INPUT, and errors) except that the output is not sent to the serial transmitter. The delays incurred by printing are still present.

When you transmit its address, the BJx wakes up and communicates. The address is a backslash (\) followed by the ASCII character represented by ADDR. For example, type:

```
ADDR=65              ;SET ADDRESS TO
                     ;65=ASCII A
\A                   ;WAKE UP "A"
P  "THIS IS AXIS"  ADDR
                     ;PRINT ADDR
ADDR=0
                     ;RESET DRIVE TO
                     ;SINGLE-DROP
```

NOTE

This example sets the address of this unit to upper case A.

Setting ADDR to 65 makes this axis address "A" and automatically puts the BJx in Multidrop mode. This axis then waits for the "\A." After this, BJx is awakened and remains awake until it receives a "\." A backslash puts ALL drives on the serial line to sleep. If you select an axis in multidrop, only that axis transmits and receives.

During multidrop, the prompts are changed. If you typed in the example from above, you would have noticed the prompt going from "-->" to "A->" after you typed in the second line. All prompts in a multidrop system have the axis address as the first character of the prompt. This allows you to know which axis you are communicating with at all times. In this way, each prompt from each axis is unique.

Table 3.10 BJx Prompts

Non-multidrop (ADDR=0)	Multidrop (ADDR = 65)
-->	A->
==>	A=>
s->	As>
t..	At.
e->	Ae>
i->	Ai>
f->	Af>
c->	Ac>

Broadcast

You may want to send all BJx units on the serial line a command simultaneously. This is called a broadcast. You can broadcast by sending "*." In this case, all BJx units execute the command. During a broadcast, none of the BJx units can transmit, but all will receive and execute the command.

CHAPTER 4
MASTER SLAVING

INTRODUCTION

This chapter discusses the various master/slave modes of the BJx. Normally, the master input is from an encoder. However, the BJx can be field-configured to accept external pulse inputs, such as a pulse train from a stepper motor controller, a frequency generator, or a customer synthesized encoder signal. The external input can control motion in all three BJx Master/Slave modes: electronic gearbox, camming, and profile regulation. The BJx, acting as the slave, accepts commands from these external sources. See Chapter 2 (Connector J13) of the *Installation* manual for information on wiring the master input.

The master/slave option must be specified when ordering your unit.

NOTE

Master-Slave must be specified when the BJx is ordered.

VEXT and VXAVG

Your program has direct access to the external input through the variables VEXT and PEXT. The frequency of the external input is provided in VEXT. VEXT is in external velocity units (VXNUM and VXDEN). PEXT is the accumulation of counts from the external input. PEXT can be set to any value from the terminal or from your program at any time; this is equivalent to normalizing the external position. PEXT is in external position units (PXNUM and PXDEN). If the external input comes from a motor, VEXT and PEXT represent the "master" motor's velocity and position. In this way, PEXT, the master position, is similar to PFB, the slave position. Likewise, VEXT is similar to VFB. If the "master" motor has the same resolution as the slave, then set PXNUM, PXDEN, VXNUM, and VXDEN equal to PNUM, PDEN, VNUM, and VDEN, respectively. Otherwise, see Chapter 3 for more information on calculating the units.

VXAVG is the average of VEXT over the previous 16 milliseconds. Occasionally, the normal sample-to-sample variation of VEXT is undesirable. In these cases, use VXAVG in place of VEXT.

MASTER MODE

The master input is accepted in four formats specified by the variable MSTRMODE:

MSTRMODE	MODE
1	A/B Quadrature (Encoder)
2	UP/DOWN
3	COUNT/DIRECTION
4	COUNT ONLY

1. **A/B Quadrature**
   A/B Quadrature is the standard format of two-channel encoders. Each transition of either channel increments (or decrements) PEXT.

2. **UP/DOWN**
   The UP/DOWN format is common stepper motor format. Each positive transition of the UP channel increments PEXT. Each positive transition of DOWN decrements PEXT.

3. **COUNT/DIRECTION**
   The COUNT/DIRECTION is also a common stepper motor controller format. If COUNT channel is set to UP, each positive transition of the COUNT channel increments PEXT; if COUNT is set to DOWN, each transition of COUNT decrements PEXT.

4. **COUNT ONLY**
   COUNT ONLY is similar to COUNT/DIRECTION except the direction of counting is fixed. Channel B input is ignored. With COUNT ONLY, each position transition of Channel A (COUNT) increments PEXT.

MENCDIR

The switch MENCDIR (Master Encoder Direction) inverts the direction of the master input. Normally, MENCDIR will be set to 1. The following figure shows increasing PEXT when MENCDIR is 1. Notice that A leads B in this mode. Setting MENCDIR to 0 reverses the effect on PEXT so that A-leads-B will generate decreasing PEXT.

In mode 3 (i.e. MENCDIR = 1), PEXT normally counts up when Channel B = 1. If MENCDIR = 0, (assuming Channel B remains 1) PEXT will count down. In all cases, VEXT is inverted. The following table shows detailed interaction between modes 2-4 and MENCDIR.

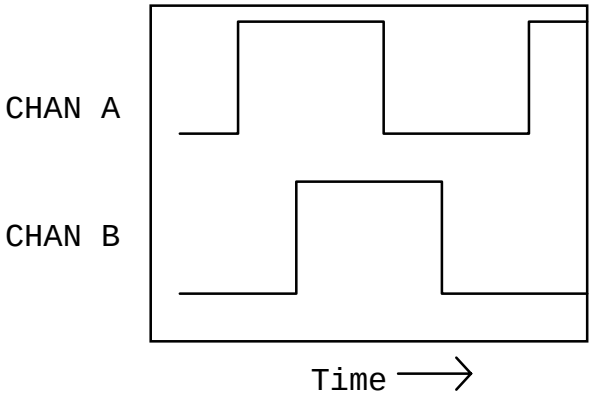


Figure 4.1 Increasing PEXT when MENCDIR is 1

Table 4.1 Effects of MENCDIR

MODE	MENCODER	MENC-DIR	CHAN A	CHAN B
UP/ DOWN	2	1	UP	DOWN
		0	DOWN	UP
COUNT/ DIR	3	1	COUNT	UP/DOWN
		0	COUNT	DOWN/UP
COUNT ONLY	4	1	UP	N/A
		0	DOWN	N/A

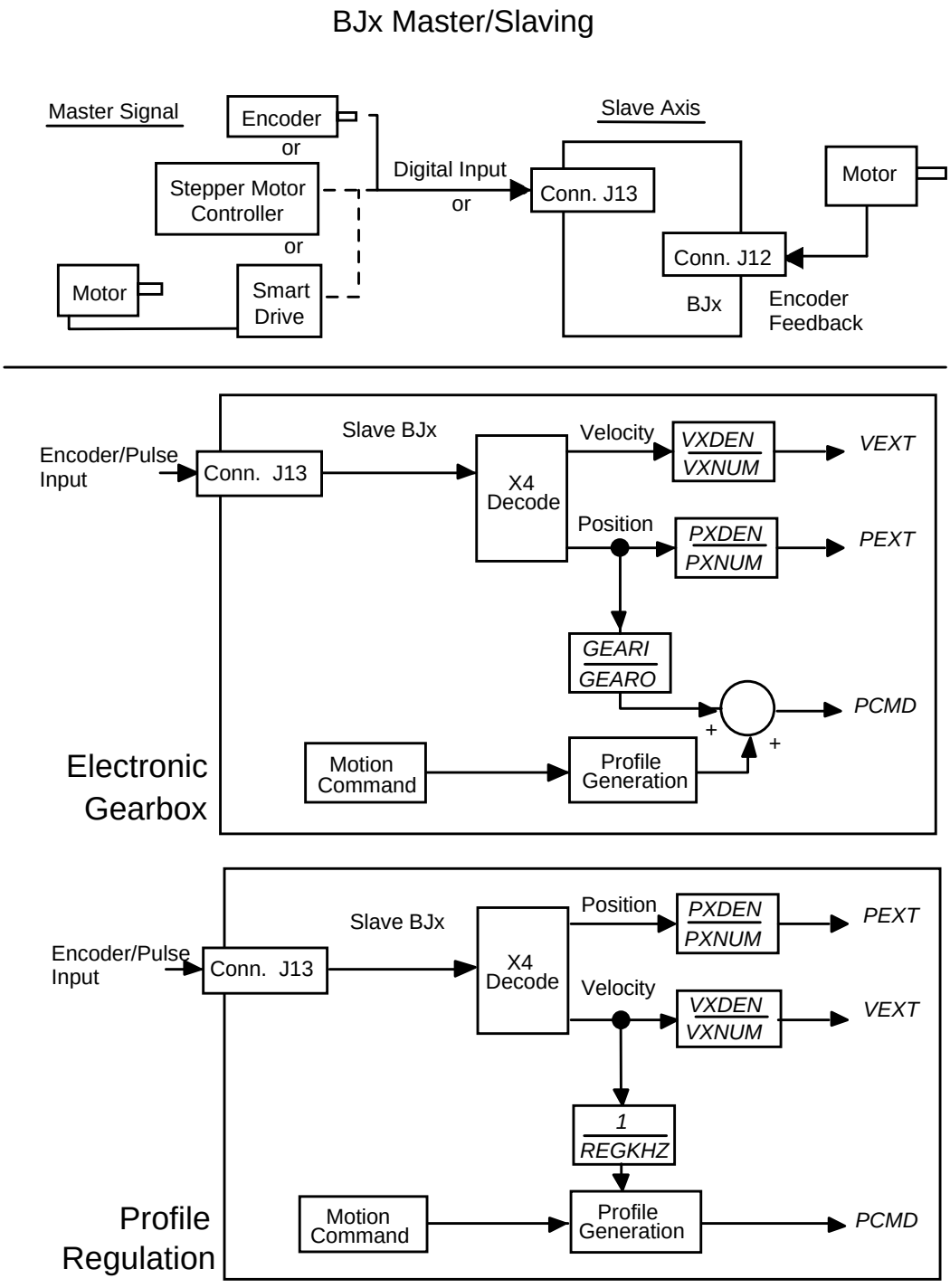
ELECTRONIC GEARBOX

Electronic gearbox is one of three BJx Master/Slave modes. Refer to Figure 4.2. Electronic gearbox is used to link two motors together so that the velocity of one is proportional to the velocity of the other. The constant of proportionality can be negative, allowing the velocities to be in opposite directions.

Gear Ratio, GEARI & GEARO

In electronic gearbox, the command signal comes from the external input. The pulses are multiplied by a gear ratio to form the position or velocity

Figure 4.2 BJx Master Slaving



command. The ratio is defined by two variables: input gear teeth (GEARI) and output gear teeth (GEARO). GEARI must be between ±32,767; GEARO must be between 1 and 32,767. If the sign of GEARI is changed, then the direction of rotation will be reversed. The direction can also be reversed by inverting the switch MENCDIR.

If the master is a motor or encoder, calculate GEARI and GEARO with:

$$\frac{GEARI}{GEARO} = \frac{REV_{SLAVE}}{REV_{MASTER}} \times \frac{RESOLUTION_{SLAVE}}{RESOLUTION_{MASTER}}$$

where:

REV_{MASTER} is an arbitrary number of revolutions of the master motor,

REV_{SLAVE} is the corresponding number of revolutions of the slave motor,

$RESOLUTION_{SLAVE}$ is the resolution of the slave motor in counts/revolution, and

$RESOLUTION_{MASTER}$ is the resolution of the master motor in counts/revolution.

If the master is a pulse train that does correspond to a motor or encoder, calculate GEARI and GEARO with:

$$\frac{GEARI}{GEARO} = \frac{REV_{SLAVE} \times RESOLUTION_{SLAVE}}{COUNTS_{MASTER}}$$

where:

$COUNTS_{MASTER}$ is an arbitrary number of counts of the master signal, and

REV_{SLAVE} and $RESOLUTION_{SLAVE}$ are as before.

To enable the Gearbox mode, type:

```
GEAR ON
```

If the ratio is not an integer, the BJx does not "drop pulses." The BJx keeps track of partial pulses to eliminate dropping pulses over time. If the number of pulses coming into the BJx is at a rate that is too large, then ERROR 97, "GEAR OVERFLOW," will be generated. This error can also be caused by the ratio of GEARO to GEARI being too large. Note that large feed-forward (KF > 4000) is normally undesirable in electronic gearbox systems because it causes overshoot.

Gearbox Example 1

Two BJx units are connected in a master/slave system. Both have 1000-line encoders so that one revolution is equivalent to 4000 counts. Suppose we want the slave motor to rotate at one-third the speed of the master motor. What are the values of GEARI and GEARO?

$$\frac{GEARI}{GEARO} = \frac{REV_{SLAVE}}{REV_{MASTER}} \times \frac{RESOLUTION_{SLAVE}}{RESOLUTION_{MASTER}}$$

$$\frac{GEARI}{GEARO} = \frac{1}{3} \times \frac{4000}{4000} = \frac{1}{3}$$

You can select any integer values for GEARI and GEARO that have the ratio 1:3.

Gearbox Example 2

Suppose the master signal in Example 1 came from a 500-line encoder. With quadrature encoding, a 500-line encoder will generate 2000 counts per revolution. If you still wanted 1:3 gearing, then:

$$\frac{GEARI}{GEARO} = \frac{REV_{SLAVE}}{REV_{MASTER}} \times \frac{RESOLUTION_{SLAVE}}{RESOLUTION_{MASTER}}$$

$$\frac{GEARI}{GEARO} = \frac{1}{3} \times \frac{4000}{2000} = \frac{4000}{6000} = \frac{2}{3}$$

So, GEARI would be 2 and GEARO would be 3.

Profiles and Gearbox

The command from electronic gearing can be summed with incremental moves and jogs. MI commands are summed with the gearbox command to form the profile. This can be used for "phase adjustment," a common function used with electronic gearbox. Phase adjustment means that the slave will be locked to the master through the electronic gearbox, but you can add a profile on top of the gearbox command. For example, you may want to increase the slave position (phase) by 90° while remaining in gear. In this case, enter the following commands:

```
GEAR ON      ;ENABLE ELECTRONIC
             ;GEARBOX
;
;  ...NORMALLY, SOME TIME WOULD
;PASS BETWEEN THESE
COMMANDS...
;
MI  1000  10   ;PHASE ADJUST 90
               ;DEGREES AT 10 RPM.
               ;SYSTEM REMAINS IN
         ;GEARBOX THROUGH
         ;THE PHASE
               ;ADJUSTMENT.
```

You cannot use MA commands when GEAR is on. Also, you cannot use position-dependent jogs (JT or JF) when GEAR is on.

Gearing and KF

Note that you should avoid large KF (>4000) when gearing. Deviations in the master input frequently cause unacceptable levels of busyness.

NOTE

Avoid KF > 4000 when using Gearing.

Velocity Offset, VOFF

VOFF, velocity offset, is added to the velocity command when the gearbox is enabled. VOFF is in velocity units. VOFF can be changed at any time. Note that VOFF is set to zero when GEAR is enabled, because if VOFF is large (say, 2000 RPM), enabling the gearbox would immediately command motion.

NOTE

VOFF is set to zero when GEAR is turned on.

Gearbox, ACC/DEC, and Jogs

When the BJx is run as a velocity loop (PL off), acceleration and deceleration rates can be limited by the variables ACC and DEC. This allows you to limit the acceleration from external velocity commands that are otherwise unlimited. If you want the acceleration and deceleration to be limited by ACC and DEC, type:

```
RAMP  ON     ;LIMIT ACC AND
             ;DEC WHEN PL IS OFF
```

PROFILE REGULATION

Profile regulation allows you to synchronize the rate of profile execution according to the external input. This modifies the velocity and acceleration of move commands without affecting the final position of the move. The rate of the move is dependent on the frequency of an external clock, which is connected to the external input, in addition to the normal limits of the move. The external input may be a master motor to which all moves must be synchronized (such as a conveyor belt motor), or it may be a signal that you generate electronically.

Profile regulation is based on an accumulation of counts from the external input during the move. If the external frequency changes during a move, the velocity of that move will be proportional to the clock frequency. In fact, if the external input frequency goes to zero, then motion will stop. Note that if the external input changes rapidly, the profile is not limited to ACC or DEC. For example, if the external frequency stopped suddenly, the BJx would command motion to stop just as suddenly. Note also that large feed-forward (KF > 4000) is normally undesirable during regulation because it causes overshoot.

WARNING

In Profile Regulation, rapid changes in the frequency can produce unexpected results.

The profile acceleration is not limited by ACC or DEC.

NOTE

Avoid KF > 4000 when using Profile Regulation.

REG & REGKHZ

REG enables the Profile Regulate mode. If REG is on, then profile regulation is enabled. REG and GEAR cannot be on at the same time.

To use profile regulation, you must determine:

1. The maximum frequency of the external input. Set REGKHZ to this value.

2. The desired speed of the move when the external input frequency is REGKHZ. Use this value as the commanded velocity of the profile.

The maximum frequency of the external input is stored in the variable REGKHZ in kHz. The profile will execute normally (that is, at the specified velocity and acceleration) when the external input frequency is equal to REGKHZ. If the input frequency is less than REGKHZ, then the profile will move the specified distance, but the acceleration and velocity will be reduced in proportion to the input frequency. The move will never go faster than specified in the original move command, even if the input frequency goes above REGKHZ. However, the input frequency should always be less than REGKHZ. REGKHZ is only resolved to 1 kHz (for example, 499.5 kHz is converted to 500 kHz).

REGKHZ is somewhat arbitrary; it must be greater than the maximum frequency of the external input and less than 2 MHz. Beyond those limits you can set it to any frequency that is convenient and adjust the commanded motion by changing the speed of the profile.

Profile Regulation works with standard moves (MA and MI) and all jogs (J, JT, and JF).

NOTE

The frequency of the external input should always be less than REGKHZ.

Profile Regulation and Counting Backwards

In general, if you use profile regulation, the external input should count forward (that is, VEXT should be positive when VXNUM and VXDEN are positive). The profile regulation firmware allows the input to count backwards for up to 30,000 counts. This is useful for applications such as conveyor belts that generally go forward but can go backward for short distances. If the external input counts backwards, the Profile Regulation mode works as follows:

- The profile stops (that is, no motion is commanded) during backward counting.

- The backward counting must be limited to 30,000 counts. Otherwise, ERROR 64 is generated.

- The profile does not continue as soon as forward counting begins. The forward counts must completely offset the backward counts before the profile will continue.

- At the point where forward counts offset backward counts, the profile continues as if the input had never gone backwards.

Regulation Example

A machine has an axis that operates on parts passing by on a conveyor belt. The profiles executed by the motor must be at a rate proportional to the conveyor belt speed. The belt normally moves at about 200 inches/minute. An encoder has been placed on the conveyor, and the maximum belt speed of 275 inches/minute is equivalent to 780 kHz on the encoder. If the belt is at maximum speed, the profile of the motor is to rotate one revolution at a peak speed of 400 RPM.
Solution: Connect the conveyor belt motor encoder to the input channel of the BJx, as shown in the *Installation* Manual. The following program should be executed:

```
REG  ON              ;ENABLE
                     ;PROFILE
                     ;REGULATION
REGKHZ=780;SET THE MAX
                ;EXTERNAL
                ;FREQUENCY TO
                ;780 KHZ
MI  4000  400        ;MOVE ONE
                     ;REVOLUTION
                ;AT 400 RPM
```

In the case above, the MI move will generate a one-revolution move at a speed proportional to the external input frequency, with 400 RPM the maximum rate when the external input frequency is 780 kHz.
Note that the belt speed never reaches 275 inches/minute. However, REGKHZ must be higher than the worst case maximum belt speed. For example, the above program can be modified to allow an even larger belt speed.

```
REG  ON              ;ENABLE
                     ;PROFILE
                     ;REGULATION
REGKHZ=1560          ;SET THE MAX
                     ;EXTERNAL
                     ;FREQUENCY
                     ;TO 1.56 MHZ
MI  4096  800        ;MOVE ONE
                     ;REVOLUTION
                ;AT 800 RPM
```

Notice that REGKHZ was doubled. However, since the speed of the move was also doubled to 800 RPM, the commanded move is identical.

CAMMING

The BJx provides electronic camming. However, the unit must be configured with the extended memory and master slave options. These options must be specified at the time your unit is ordered.

NOTE

Camming requires the BJx with Extended Memory and Master-Slave Options.

Conventional Cams

Conventional cams convert rotational motion to linear motion. As Figure 4.3 shows, the "command" signal comes from a master-drive shaft fitted with a cam lobe. The cam generates linear motion on a tool which is driven by a "follower." Cams are used when a specific profile must be generated each time a drive shaft turns one revolution. The cam lobe can have a wide variety of shapes.

Electronic cams offer two important advantages over mechanical cams: the profile of an electronic cam is much easier to change, and the profiles are not subject to mechanical wear. One of the most important features of an electronic cam is that the master drive can rotate in one direction indefinitely. With conventional positioners, this will eventually cause an error because the internal position counter will overflow. Also, the electronic cam controller must support gear ratios between the drive shaft and the follower. Again, the drive can rotate indefinitely, and the controller must not lose counts. The BJx has been designed with both of these criteria.

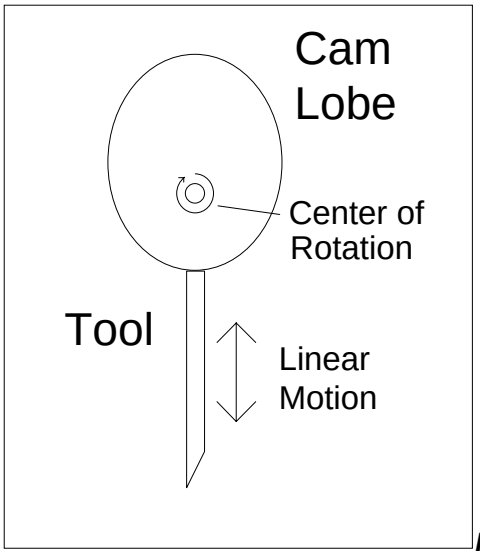


Figure 4.3 Cam Lobe

CAM Setup

To use BJx camming, you need to follow these steps:

1) Generate a cam table and enter it into the BJx.
2) Scale the BJx electronic gearbox.
3) Align the machine and enable camming.

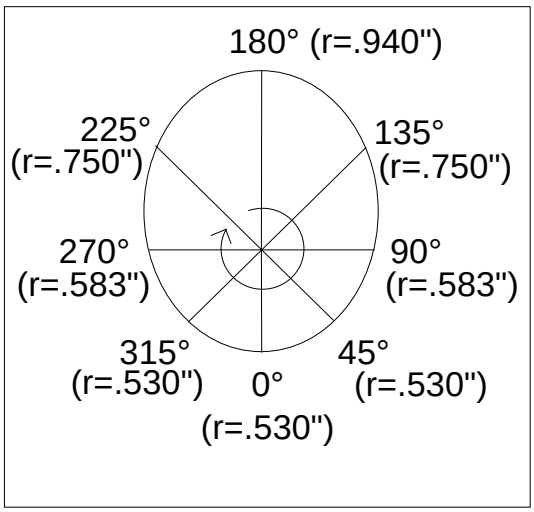


Figure 4.4 Dividing a CAM Lobe

Generating a Table

To generate a table, start with a graph showing the master drive position versus the follower position. Divide this graph into 128 evenly spaced sections. Calculate or measure the radius from the lobe center-of-rotation at each of the 128 points. Load these radii

into the BJx user variables X100-X227 as shown below:

Segment Number	Master Position (Degrees)	Load Position in this User Variable
1	0	X100
2	2.81	X101
3	5.62	X102
4	8.43	X103
.	.	.
.	.	.
.	.	.
126	351.56	X225
127	354.38	X226
128	357.19	X227

Figure 4.5 Camming Table

At the end of the table, the position command wraps around X227 to X100.

For the example in Figures 4.3 and 4.4, 8 positions in the table would be

Lobe Position	Register	Tool Position
0° (=360°)	x100	0.530"
45°	x116	0.530"
90°	x132	0.583"
135°	x148	0.750"
180°	x164	0.940"
225°	x180	0.750"
270°	x196	0.583"
315°	x212	0.530"

Note that all user variables are zeroed at power-up. You must reload these variables from the user program.

NOTE

The Cam Table (X100-X227) must be reloaded at every power-up.

Scale the Gearbox

The BJx processes the master drive signal through the gearbox so you can select the gear ratio you need. You must select the gear ratio so that when the master rotates 360 degrees, 32,768 counts are generated in the BJx. Think of the 360-degree horizontal axis as being 32,768 counts long. For example, suppose the sensor on the drive shaft is a 1000-line encoder. Because of quadrature, the encoder would generate 4000 counts for every rotation. So the 4000 counts should be scaled through the gearbox to generate 32,768 counts. The gear ratio would be:

$$\frac{GEARI}{GEARO} = \frac{32,768}{4000} = \frac{1024}{125}$$

or, GEARI=1024 and GEARO=125.

If you want to test your scaling, enable your BJx (without camming), turn GEAR ON, and rotate the master input 360 degrees. PEXT should increase or decrease by 32,768 counts.

Align the Machine

On power-up, the cam axis must be aligned to the drive shaft. The simplest method is to assume the cam cycle always begins at the start of the cam table as recorded in X100.

1) Home using GOHOME.
2) Move to X100 using MA.
3) Turn CAM on and normalize with NORM 0 CAM.

When the "NORM <Master Drive Position> CAM" command is executed, PCMD is read from the cam table. PFB is set to the same value as PCAM and, therefore, there is no PE (position error).

If you are using position units, the most convenient place to normalize for camming is when the master drive is at zero. This is because the master drive position (PCMD) uses position units (PNUM and PDEN), which are normally scaled for the follower. When you normalize to zero, the units do not have any effect. However, if you want to normalize the master to a non-zero position, you must

A) Determine the position of the drive master to which you will normalize;

B) Convert the position to counts so 360 degrees equals 32,768 counts;

C) Temporarily set position units to 1:1 (PNUM = PDEN = 1);

D) Normalize to the position in counts:
EN
NORM <Master Drive Position in Counts> CAM

E) Restore the position units to their original values.

For example, if you know the follower is 3 inches and the drive is 90 degrees:

A) Drive position = 90 degrees.
B) Drive position = 8192 counts.
C) PNUM=1
PDEN=1
D) EN
NORM 8192 CAM
E) Restore PNUM and PDEN

Finally, you must turn GEAR on. This connects the drive to the follower. If you want to test your system before you connect the master-drive, you can use VOFF. VOFF is the offset speed for the electronic gearbox. For example, if the master drive is not moving and you turn GEAR ON and set VOFF to 100 RPM, the position command will increase at a rate of 100 RPM. This has the same effect as the encoder option input running at 100 RPM.

PCAM and PCMD

The BJx uses a special variable for camming, PCAM. PCAM is the position command from the cam table. This is usually the role for PCMD (position command). However, when camming is enabled, PCAM represents the position from the electronic gearbox; that is, the position that goes into the table. PCAM is the output from the table.

PCMD is automatically in a "ROTARY" mode where the distance of one rotation is fixed at 32,768 counts. PCAM can be printed or recorded with PC-Scope. In fact, if you want to see your cam profile, you can record PCAM and PCMD simultaneously. For example, the following line records both positions for 0.5 seconds.

```
RECORD 500 1 PCMD PCAM
```

You can then use PC-Scope to verify your profile. Also, the PS and RS commands display "CAMMING" if GEAR and CAM are both ON.

Camming and KF

Note that you should avoid large KF (>4000) when camming. Deviations in the master input frequently cause unacceptable levels of busyness.

NOTE

Avoid KF > 4000 when using camming.

Limitations

There are several limitations for camming applications, reflecting the fact that many BJx functions are not useful when camming. For example, profile commands (MI, MA, J, JT, JF) are not allowed. ROTARY must be OFF. Any error that disables the drive also disables camming; you must re-normalize after such errors. Error 23, SOFTWARE OVERTRAVEL, normally disables the BJx and breaks the user program; when camming is off, this error only breaks the user program.

Implementation

Camming is implemented as a modification of the BJx gearbox. As Figure 4.6 shows, the standard gearbox produces PCMD by multiplying PEXT by the ratio GEARI/GEARO. PE is formed by subtracting PFB from PCMD. Usually, PEXT is generated from another motor's feedback sensor. In this way, a master motor position (PEXT) controls the slave motor position (PFB).

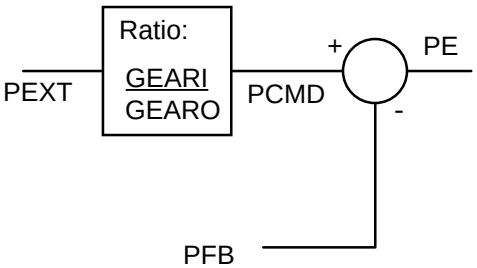


Figure 4.6 BJx Gearbox Position Error

To implement camming, Kollmorgen modified the gearbox by adding a look-up table. As Figure 4.7 shows, PEXT is processed by the gearbox to form PCMD. PCMD is then used as an index into the CAM look-up table to produce a new variable,

PCAM. When camming is enabled, PCAM is used to form PE.

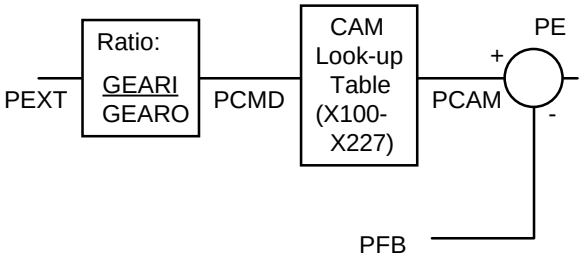


Figure 4.7 BJx Cam Position Error

Interpolation

The BJx Cam Table has 128 points but, using interpolation, the BJx is able to generate a 32,768-point cam. The interpolation algorithm splits each cam table point into 256 linearly interpolated mid-point positions based on the master input. This is why the gear ratio must be chosen so that each revolution of the Master Cam input generates 32,768 counts of PCMD to the cam table. This scaling is easily accomplished by programming the GEARI variable to 32,768 and then programming the number of counts generated by one revolution of the master input CAM into GEARO. An example follows.

Selecting GEARI and GEARO

The master input device is a 2000-line encoder. One cam profile should be generated each time the master encoder turns one revolution.

Each revolution of this encoder would produce 8000 (2000x4) counts of position command. For one cam profile, the master input should receive 8000 counts. This means that the ratio of GEARO to GEARI should be 8000/32,768. Note that GEARI cannot be set to 32,768 as the limit is 32,767 and that small difference will cause error to accumulate. Instead, we must reduce GEARI while maintaining the original ratio to GEARO. In this example, set GEARO to 125 and GEARI to 1024.

NOTE

The variable GEARO must be a number between 1 and 32,767. The variable GEARI must be a number between -32,768 and 32,767.

Test Program

The following test program will perform these tasks:

1. Load X100-X227 with a triangle wave where x100 = 0, x101 = 100, x102 = 200, etc. The midpoint of the triangle wave is X164 so that X164 = 6400, X163 = 6300, X164 = 6400, X165=6300, X166 = 6200, and so on.
2. Enable the BJx and camming.
3. Use VOFF to move PCMD through the cam cycle. VOFF is usually used with the gearbox to add an offset speed. Here, we use it to simplify the test.
4. Calculate which segment this iteration is in (it repeats from segment 0 to segment 127 every 128 iterations) and store it in X2.
5. Determine PCMD at the end of the segment.
6. Wait until PCMD reaches the boundary. Store the commanded position (PCAM).
7. Calculate error between what the command is (X4) and what it should be (X(X2)).
8. Subroutines to print whether iteration tested good or bad.

Program Listing

```
;CAM TEST
;
;USE VOFF FOR THIS TEST TO KEEP
;VCMD MOVING AT A CONSTANT
;SPEED.  SELECT THAT SPEED TO ;BE 20
RPM, WHICH IS ABOUT 1300
;COUNTS/SECOND FOR A 1000-LINE
;ENCODER.  THE ENTIRE CAM ;CYCLE IS
32768 COUNTS.
;EACH OF THE 128  SEGMENTS IS ;256
COUNTS.  SO 20 RPM CYCLES
;THROUGH THE CAM CYCLE AT
;ONE CYCLE PER 32768/1300  OR 25
;SECONDS.
;
;1$
;THIS SECTION LOADS CAM VARS
;WITH A TRIANGLE PROFILE WHERE
;EACH SEGMENT IS DIFFERENT
;FROM THE LAST BY 100 COUNTS

;LOAD VARIABLES X100-X164
;XI=100                  ;STARTING
                         ;VARIABLE
;2$                      ;LOOP START
X(X1)=(X1-100)*100       ;LOAD THE "UP
                         ;SIDE" OF THE
                         ;TRIANGLE
X1=X1+1                  ;INCREMENT
                         ;LOOP  CTR
```

```
?X1 LE 164 GOTO 2  ;TEST --KEEP
                   ;GOING TO X164
;
;NOW, LOAD VARIABLES X165-X227
;WITH THE "DOWN SIDE" OF THE
;COUNTER
3$                 ;LOOP START
X(X1)=X164-(X1-164)*100
X1=X1+1            ;INCREMENT
                   ;LOOP CTR
? X1 LE 227 GOTO 3 ;TEST --KEEP
                   ;GOING TO X227
;
;NOW, IT'S TIME TO START THE CAM
4$
PLIM OFF      ;STANDARD LINE
              ;TO ENABLE

GEAR OFF           ;DISABLE GEAR
                   ;SO WE CAN
                   ;ENABLE CAM
EN
NORM 0 CAM;NORMALIZE &
                   ;ENABLE CAM
GEAR ON            ;NOW, WE CAN
                   ;ENABLE GEAR
VOFF 20            ;USE OFFSET
                   ;SPEED OF 20
                   ;RPM TO GO
                   ;THROUGH
                   ;CAM CYCLE
B                  ;BREAK TO
                   ;IMMEDIATE
```

CONTINUE

The CONTINUE command is provided as a controlled way to turn off master/slave position control. The CONTINUE command tells the BJx to keep the motor going at its present speed while simultaneously turning off REG and GEAR. One use of this command is to cause a controlled deceleration to 200 RPM, when the electronic gearbox is enabled, for example. If you just type:

```
J  200
```

it would have the effect of adding 200 RPM to the command from the gearbox. However, if you type:

```
CONTINUE
J  200
```

the CONTINUE would disable the electronic gearbox while commanding the motor to continue at whatever speed it was going when the command was executed. Then the *J 200* command would bring about a controlled deceleration to 200 RPM.
CONTINUE normally looks at the velocity command for 1 millisecond. If the velocity command is generated from the electronic gearbox or a regulated profile, it can vary considerably. The CONTINUE command allows you to specify a time period, up to 1 second, over which velocity command is averaged. For example, if you entered:

```
CONTINUE  50
```

the CONTINUE command would change the velocity command to the average velocity command over the previous 50 milliseconds. CONTINUE always sets SEG to 1.

CHAPTER 5
USER PROGRAMS

INTRODUCTION

This chapter explains the capabilities of the BJx programming language. Examples of programming techniques will aid you in developing applications.

User programs are combinations of BJx commands. These programs are stored in non-volatile RAM and retained during power-down. User programs are composed of commands such as those in Chapters 3 and 4. In addition, there are commands that control the way the program executes; these commands are covered in this chapter.

PROGRAMMING TECHNIQUES

This section discusses programming practices. The BJx has a flexible language. Follow accepted programming practices to ensure that this flexibility does not lead to overly-complex programs. If you follow good programming practices you will:

- be able to modify programs when the application changes;
- have fewer programming errors;
- have an easier time fixing the programming errors that do occur; and
- be able to get help with errors you cannot fix.

1. DO NOT PROGRAM SAFETY FUNCTIONS.

Always hardwire personal safety functions. Never program these functions.

Always hardwire safety functions. This includes EMERGENCY STOP or ESTOP. You should not depend on your program for safety functions because of three potential problems: 1) you can easily make programming errors; 2) a function on the BJx may not work in exactly the way you expect it to in every condition; and 3) a critical component in your system may fail and prevent the function from working. Remember, safety functions are rarely exercised so that if one of these problems does occur, it can go undetected indefinitely. If personal safety is involved, always hardwire the function.

2. USE CAUTION WHEN PROGRAMMING EQUIPMENT PROTECTION FUNCTIONS.

Programming errors can damage equipment.

Use caution when programming equipment-protection functions.

Sometimes you can hardwire equipment protection functions, but other times this is impractical and you must program the functions. If this is the case, be careful. Remember, if your program has an error, it can result in damage to your equipment.

3. WRITE A SIMPLE SPECIFICATION FOR YOUR APPLICATION.

Write an outline of all the functions your application will require before you start programming. This will serve as a specification. Everyone who is involved with your system (customers, supervisors, co-workers, operators) should agree on the specification. While last-minute requests for program changes will still occur, this is a reasonable step towards reducing the incidence of such requests.

4. WRITE A FLOWCHART OF YOUR PROGRAM.

Kollmorgen strongly recommends the use of flowcharts. They are required if you need programming help from Kollmorgen customer service.

5. COMMENT YOUR PROGRAM.

Always comment your programs. Comments help explain your program to other people. Keep in mind that others may need to modify your program in the future. Comments also help you remember why you chose certain ways to do things.

6. AVOID SPAGHETTI CODE.

A program with too much branching is often called spaghetti code because of the look of the flowcharts. Avoid branching, especially branching up (toward the top of your program); logic in programs that branches down is more intuitive and, thus, less prone to errors.

Customer Service

If you need help with software or understanding BJx functions, you can contact the Regional Kollmorgen Sales Office (See Appendix F). Ask for the Applications Engineer. Please observe the following procedure:

1. Be prepared to provide the following items:
   a. A written specification of the system;
   b. A flowchart; and
   c. A hard copy of the program.

2. Be prepared to take the following actions, should they be necessary:
   a. Strip out sections of your program to help locate a problem.
   b. Rewrite sections of your program that do not conform to the programming practices described in this chapter.
   c. Video tape your machine to help demonstrate the problem.

If you need help with your program, Kollmorgen is committed to helping you. BJx software support is provided by:

1. Helping you organize your program.
2. Explaining proper programming practices.
3. Discussing BJx functions.

Test Program

Chapter 2 provided an in-depth procedure for installing and using Motion Link. This section provides you with enough information to get started. Enter a simple program with the following procedure:

1. Establish communication with the BJx as discussed in Chapter 3 of the *Installation* manual.

2. Press the right arrow key to display the menu bar. Select PROGRAM.

3. Select NEW.

4. Enter this program:

```
10$
P  "HELLO WORLD"
B
```

5. Press the escape key to exit the Motion Link Editor.

6. Motion Link will ask you if you want to save your program. Enter "Y" and name your program "TEST."

7. Motion Link will now ask you if you want to transmit the program to the BJx. Enter "Y."

8. After the transmission is complete, you should receive the interactive prompt (-->). Type:

```
RUN  10
```

Your program should print:

```
HELLO WORLD
-->
```

See Chapter 2 for a complete description of Motion Link.

BUILDING A PROGRAM

Programs are sequences of commands, many of which can also be executed directly from the keyboard. Examples of these commands are MI, MA, and P (Print). However, in order for a program to run properly, other commands, called *program control commands*, are required. Examples of these commands are GOTO and GOSUB.

Basic Commands

Labels

Labels are used to mark places in the program where execution begins or continues. There are two kinds of labels: general purpose and dedicated.

General purpose labels are numbers from 0 to 500 followed by a dollar sign ($). You can execute a program that begins at a general purpose label with the RUN command. You can jump to a label from within your program with the GOTO and GOSUB commands. RUN, GOTO, and GOSUB are described later in this chapter.

Dedicated labels each have specific functions; these include alarms, auto programs, and the user error handler. These labels are *letters* or *words* followed by a dollar sign. For example, A$ is the A-Alarm label. Dedicated labels cannot be used by the RUN, GOTO, or GOSUB commands. These labels are discussed with multi-tasking later in this chapter.

RUN

The RUN command is used to start the program from the Interactive mode. For example, type:

```
RUN  3
```

If there are no errors, program execution begins at label 3. If the label is not in the program, an error is generated and no part of the program is executed. You cannot use the RUN command for dedicated labels.

Before the program is run, the BJx searches the entire program for some types of errors. If, after you enter a RUN command, an error is detected, the BJx will display the appropriate error message together with the offending line. Also, RUN checks for program corruption and generates a "PROGRAM CORRUPT" error if necessary. The program corrupt error can be cleared by retransmitting the program from Motion Link. If a "Program Corrupt" error occurs, it may indicate a serious condition. Contact the factory.

Break (B)

The Break (B) command stops program execution and returns to the interactive state. The Break command does not stop motion. Profile commands are allowed to continue until they are complete. If you want to break the program and stop motion, precede the Break command with the Stop (S) command.

GOTO

The GOTO command is used within the program to jump to a label.

GOSUB and RET

The GOSUB command goes to a subroutine at the specified label. For example:

```
GOSUB  66
```

begins a subroutine at label 66. The RET command returns from the subroutine and begins executing the program below the original GOSUB command. GOSUBs can be nested up to four levels.

For example, type in the following program:

```
4$
GOSUB  5
P  "RETURNED FROM SUBROUTINE 5"
B
;
5$
P  "EXECUTING SUBROUTINE 5"
RET
```

Exit the Editor and type:

```
RUN  4
```

The result should be:

```
EXECUTING SUBROUTINE 5
RETURNED FROM SUBROUTINE 5
```

CONDITIONAL COMMANDS

The BJx provides several conditional commands that allow your program to make decisions. Conditional commands include ? (Quick-IF), TIL, IF, ELIF, and ELSE. These commands all depend on *conditions*. A condition is an arithmetic comparison of two numbers, variables, or expressions. The BJx supports all 6 common types of arithmetic conditions. Use the following two-character codes:

Table 5.1 BJx Conditions

GT	Greater Than
GE	Greater Than Or Equal To
LT	Less Than
LE	Less Than Or Equal To
EQ	Equal To
NE	Not Equal To

QUICK IF (?) COMMAND

The ?, or *Quick IF*, is a single-line command that allows you to specify a condition, a command to be executed if the condition is true, and another to be executed if the condition is false. The format of the ? command is:

? condition TRUE-command : FALSE-command

TRUE-command is executed if the condition is true; otherwise, the FALSE-command is executed. Both TRUE-command and FALSE-command are optional, although at least one must be present.

Some examples of the ? command are:

```
? VFB  GT  3000  P  "FAST"  :  P  "SLOW"
? X1  GT  5  P  "X1 > 5"  :  P  "X1 <= 5"
? 2*X2-5  LE  X1/100  GOSUB  40
? X1/2*2  EQ  X1  GOTO  5
                        ;GOTO 5 IF X1 IS
                ;EVEN.  DO
                        ;NOTHING IF X1 IS
                ;ODD.
? I4  EQ  1  J  2000    ;I4 IS A JOG
                ;BUTTON
```

The ? command can be used to make a loop counter, as demonstrated by the following statements:

```
X30 = 1                 ;X30 IS THE
                ;LOOP COUNTER
12$                     ;THE LOOP BEGINS
                ;AT 12$
GOSUB  10               ;GO TO
                ;SUBROUTINE 10
X30 = X30+1             ;INCREMENT THE
                ;LOOP COUNTER
? X30  LE  25  GOTO  12
                ;LOOP 25 TIMES
```

Nesting ? Commands

You can nest one ? command inside another. For example, suppose you want to break program execution if X1 is less than 100 and greater than -100:

```
? X1 LT 100  ? X1 GT -100  B
```

Nesting two ? commands is the same as ANDing the two conditions. Nesting of ? commands is limited by the number of entries and the maximum length of a line. BJx commands are limited to 15 entries (the example above has 9 entries: ? , X1, LT, 100, ?, X1, GT, -100, and B). Since each level of ? command nesting requires 4 entries, you cannot have more than 3 levels of nesting. Also, a ? command must be less than 80 characters long since it must fit on a single line.

TIL COMMAND

TIL is a single-line command that allows you to specify a condition and a command to be executed repeatedly until that condition is true. The TIL command has the following format:

TIL condition FALSE-command

FALSE-command is repeatedly executed as long as the condition is false. If the condition is true at the beginning of the TIL command, then *FALSE-command* is never executed. In this case, program execution continues to the next step. An example of the TIL command would be to print a line to the operator continuously until the variable PFB is greater than 10,000. This statement delays program execution until the condition is true and also refreshes the display while the program waits:

```
TIL  PFB  GT  10000  P  "WAITING"
```

The TIL command can simply delay a program until a condition occurs; in this case, do not enter a *FALSE-command.* For example:

```
TIL  PFB  GT  10000
```

NOTE

TIL can be used to delay program execution.

More examples of the TIL command are:

```
TIL  I1  EQ  ON          ;DELAY EXECUTION
TIL  I1  EQ  ON  P  "PRESS INPUT #1"

TIL  SEG  EQ  0          ;DELAY UNTIL
                         ;MOTION STOPS
TIL  SEG  EQ  0  P  PFB
                         ;PRINT UNTIL
                    ;MOTION STOPS
```

IF, ELIF, ELSE, and ENDIF Commands

The IF command, together with ELIF, ELSE, and ENDIF, will allow you to conditionally execute large blocks of commands. These commands are provided because the ? command is limited to a single line. Use the IF command to write more readable programs.

The format of the IF, ELIF, ELSE, and ENDIF commands follows. Note that the conditions have the same format as the conditions for the TIL and ? commands. Note also that *block* can indicate any number of commands:

```
IF  IF-condition
  Block-IF
ELIF  ELIF-condition #1
  ELIF-block #1
ELIF  ELIF-condition #2
  ELIF-block #2
ELSE
  ELSE-block
ENDIF
```

The above example shows two ELIF commands. You can have any number of ELIF commands. The operation of this example IF command is as follows:

If... IF-condition is TRUE,

All commands in the Block-IF are executed.

No other blocks are executed.

Program execution continues after the ENDIF command.

Otherwise if... ELIF-condition #1 is TRUE,

All commands in ELIF-block #1 are executed.

No other blocks are executed.

Program execution continues after the ENDIF command.

Otherwise if... ELIF-condition #2 is TRUE,

All commands in ELIF-block #2 are executed.

No other blocks are executed.

Program execution continues after the ENDIF command.

Otherwise...

All commands in ELSE-block are executed

Program execution continues after the ENDIF command.

Note that only the first block with a true condition is executed. The IF, ELIF, ELSE, and ENDIF commands have several restrictions and options:

Table 5.2 Block-IF Restrictions and Options

Each IF/ELIF/ELSE/ENDIF set...
...must have one and only one IF.
...may have any number of ELIFs.
...need not have any ELIFs.
...may have one ELSE.
...need not have an ELSE.
...must have one and only one ENDIF.

IF vs. ?

You can use ? in place of IF commands. For example, clamping applications make decisions based on the final position of the motor after a move. For our example, assume that the PFB should be between 50 and -50. If PFB is within range, the program should turn output O1 on and print an appropriate message. If it is out of range, O1 should be turned off and a message should be printed. The table below shows the desired operation:

Table 5.3 Desired Operation of Program Example

PFB RANGE	O1	MESSAGE TO PRINT
PFB > 50	OFF	PFB TOO LARGE
PFB < -50	OFF	PFB TOO SMALL
-50 < PFB < 50	ON	PFB WITHIN RANGE

The IF, ELIF, ELSE, and ENDIF commands implement the desired functions:

```
IF  PFB  GT  50          ;BEGIN BLOCK-IF
 O1  OFF                 ;O1 MEANS "WITHIN
                 ;RANGE"
 P  "PFB EXCEEDED MAXIMUM"
                 ;PRINT ERROR
        ;MESSAGE
ELIF  PCMD  LT  -50  ;CHECK THE
                 ;NEGATIVE LIMIT
 P  "PFB EXCEEDED MINIMUM"
                 ;PRINT ERROR
        ;MESSAGE
 O1  OFF                 ;O1 MEANS "WITHIN
                 ;RANGE"
ELSE                     ;IF HERE, THEN
                 ;WITHIN RANGE
 O1  ON                  ;TURN ON O1
 P  "PFB WITHIN RANGE"
                 ;PRINT MESSAGE
ENDIF                    ;END OF BLOCK-IF
```

This example could have been written with ? commands, but it would have been less intuitive.

Nesting IF commands

You can nest IF commands. The following program shows two levels of nesting:

```
55$
IF  X1  GT  0
  IF  X2  GT  0
    P  "BOTH X1 AND X2 > 0"
  ELSE
    P  "ONLY X1 GT 0"
  ENDIF
ELSE
  IF  X2  GT  0
    P  "ONLY X2 GT 0"
  ELSE
    P  "NEITHER X1 NOR X2 > 0"
  ENDIF
ENDIF
B
```

You can nest IF commands indefinitely. You should be careful to include all of the ENDIFs to close each level of nested IF. The indentation shown above is not required but is present to make the program more readable. The BJx ignores the indentation.

IFs with GOTO and GOSUB

You can use the GOSUB command from within a Block-IF, even if you have another Block-IF in that subroutine. In this case, the IF in the subroutine is like a nested IF. However, be careful to return from the subroutine after you have executed the ENDIF. You should never return from a subroutine from between IF and ENDIF. Finally, you may use a GOTO to jump completely out of an IF-ELSE control structure. When a GOTO is executed after an IF has been executed but before an ENDIF has been executed, all ENDIFs are automatically executed. This means you cannot jump to a label within any IF-ELSE structure. Note that jumping out of a control structure in such a manner is often a poor programming practice and should be avoided.

NOTE

You cannot GOTO the middle of an IF/ENDIF set. You should never execute a RET from between an IF and ENDIF.

SYNCHRONIZING YOUR PROGRAM

This section describes the functions and variables that allow you to synchronize the program to events, both external and internal.

Idling Commands

Idling commands are advanced alternatives to the TIL command. Idling commands are important when you are using multi-tasking, which is discussed later in this chapter. For now, be aware that while TIL delays execution of all parts of the program, idling commands stop only the current task; other tasks can execute during the delay.

There are four idling commands: Hold (H), Dwell (D), Wait (W), and INPUT. This section discusses the first three; INPUT is discussed later in this chapter. Hold waits for switches to change state, Dwell waits for a timer, and Wait waits for a motion segment.

HOLD (H)

The HOLD command waits for a switch to be either on or off. You specify the HOLD command with the switch and the desired state. For example,

```
H  I1  ON              ;HOLD UNTIL INPUT
                ;I1 IS ON
H  O2  OFF             ;HOLD UNTIL
                ;OUTPUT O2 IS
H  TRIP1  ON           ;OFF HOLD UNTIL
                ;PFB > PTRIP1
```

Enter the following program:

```
29$
P  "TURN I1 ON"
H I1 ON
P  "I1 IS NOW ON"
B
```

Now transmit the program, turn input I1 off, and observe the action of the HOLD command by typing:

```
RUN 29
```

You can Hold for any switch except some user switches. Only user switches XS1-XS10 are allowed with the HOLD command.

DWELL (D)

Sometimes it is desirable to delay execution for a specified amount of time. The Dwell (D) command is the easiest way to do this. The delay is specified in milliseconds. For example:

```
D  1000        ;DWELL 1000
       ;MILLISECONDS
```

delays execution for 1000 milliseconds or 1 second. Dwell can be demonstrated by typing in the following simple program:

```
6$
P  "BEGIN 5 SECOND DWELL"
D  5000
P  "END   5 SECOND DWELL"
B
```

Now exit and type:

```
RUN  6
```

The result should be:

```
BEGIN   5 SECOND DWELL
END     5 SECOND DWELL
```

with 5 seconds between lines being printed. Dwells can be up to 2,147,483,647 milliseconds or about 25 days.

WAIT (W)

When using move and jog commands, it is often necessary to synchronize the execution of your program to motion. The Wait (W) command can be used to wait for the specified motion segment to start. (Refer to Chapter 3 for a discussion of Motion Segments and the variable SEG.) Examples of the Wait command are:

```
W 1             ;WAIT FOR MOTION
      ;COMMAND TO BEGIN
W 2             ;WAIT FOR SEGMENT 2 TO
                ;START
W 0             ;WAIT FOR MOTION TO
      ;STOP
```

W 1 delays program execution until the last motion command entered has started segment 1. W 2 delays until segment 2. *W 0* delays program execution until the last motion command has stopped.

In the example below, WAIT is used to delay the calculations of the third move until the second move has begun:

```
MI  10000  100        ;BEGIN 1ST MOVE
MI  10000  200        ;CALCULATE THE
                ;2ND MOVE DURING
      ;THE 1ST.
W  1                  ;DELAY PROGRAM
                ;EXECUTION UNTIL
                      ;2ND MOVE STARTS
MI  10000  300        ;CALCULATE THE
                ;3RD MOVE
```

If a second move is entered while the first is running, Wait always waits for the second move:

```
MI  -50000  1000      ;BEGIN 1ST MOVE
MI  -50000  1000      ;CALCULATE 2ND
                ;MOVE
W  2                  ;WAIT FOR SEG 2
                ;OF 2ND MOVE
```

Another example of the WAIT (W) command is seen when using multiple JOG TO/JOG FROM commands. Normally, you should place a WAIT (W) command between these commands, because the initial traverse of a JOG FROM/JOG TO command begins as soon as the command is entered. Usually, you will want the traverse to begin at the end of the last specified acceleration segment. For example, consider:

```
J  1000               ;START MOTION
W  2                  ;WAIT TIL JOG
                ;ACCEL IS DONE
JT  10000  200        ;ENTER JT FOR
                ;FIRST DECEL
W  3                  ;WAIT TIL JT DECEL
                ;IS DONE
JT  11000  0          ;ENTER FINAL
                ;SEGMENT OF
                      ;MOVE
```

Using Timers, TMR1-4

The general purpose timers, TMR1-4, are provided for timing that is too complex for the Dwell command. The timers are set in milliseconds and are limited to 2,147,483,647 milliseconds or about 25 days. You set the timer; the BJx counts it down until it reaches zero.

Type in this example, which continuously reprints a message for 1 second:

```
8$
TMR1=1000
TIL  TMR1  LE  0  P  "WAIT FOR 1 SEC"
B
```

and type:

```
RUN 8
```

Type in this example showing how multiple delays can be based on one timer setting:

```
9$
TMR1  3000   ;SET TMR1 TO 3 SECONDS
P  "3 SECONDS"
TIL  TMR1  LE  2000
P  "2 SECONDS"
TIL  TMR1  LE  1000
P  "1 SECOND"
TIL  TMR1  EQ  0
B
```

and type:

```
RUN 9
```

Regulation Timer, RD

Fixed length delays can be added into a program with the Dwell (D) command. In some applications, especially those that use profile regulation, it is necessary to add a delay with a length that varies with the regulating frequency. The Regulated-Dwell (RD) command is provided for these occasions. When the external input frequency is equal to REGKHZ, the delay of the RD command is in milliseconds, just like D command. However, when the external input frequency decreases, the regulated dwell time lengthens so that the Dwell is proportional to the inverse of the external frequency. For example:

```
45$
REGKHZ  100;SET REGKHZ TO
        ;100 KHZ
RD  2000                ;REG DOES NOT
                ;NEED TO BE ON
P  "DELAY COMPLETE"
B
```

In this case, the RD command causes a 2-second dwell when the external input frequency is 100 kHz and a 4-second dwell when the frequency is 50 kHz. Note that RD delays are always regulated by the external frequency, even when REG is off.

USING GENERAL PURPOSE INPUTS

General purpose inputs can be used to control the program. From Chapter 3 you may recall that these inputs can be referred to one at a time using variables I1-I8, or collectively IN. If the program must wait for a particular input to be on or off before continuing execution, the TIL command can be used:

```
TIL  I5  EQ  0
```

If this statement is executed from the program, the program will delay execution until I5 is 0.

If the program must wait for many inputs to be on or off, then the TIL command can be expanded. For example, if inputs 1, 4, 5, and 6 must all be on, either of the following TIL instructions can be used:

```
TIL  I1+I4+I5+I6  EQ  4
                ;THIS USES
        ;ALGEBRAIC MATH
TIL  I1&I4&I5&I6  EQ  1
                ;THIS USES
        ;LOGICAL MATH
                        ;BOTH WORK
```

It is slightly more complicated if the program must wait for some inputs to be on and others off. For example, if inputs 1, 4, and 5 must be on, and input 6 must be off, the following TIL instructions can be used:

```
TIL  I1+I4+I5+(1-I6)  EQ  4
                ;ALGEBRAIC MATH
TIL  I1&I4&I5&(1-I6)  EQ  1
                ;LOGICAL MATH
```

Notice the use of (1-I6). This is a logical NOT.

If more than a few inputs must be tested, then referencing them one at a time can be cumbersome. As an alternative, IN can be used. This can be demonstrated with the example above. If the program must wait for inputs 1, 4, and 5 to be on and input 6 to be off, logical math can be used to mask the inputs that are not supposed to be tested: inputs 2, 3, and 7, 8. A mask is a binary word with a 0 for each input that is not tested and a 1 for each that is. In this example, the mask would be:

Input Number	8	7	6	5	4	3	2	1
Test Input?	N	N	Y	Y	Y	N	N	Y
Binary Mask	0	0	1	1	1	0	0	1

Since the mask can be in hex or decimal, it can be expressed as:

00111001 (BINARY) equals 39 (HEX) or 57 (DECIMAL), which equals 1+8+16+32 (DECIMAL).

Now that the mask is known, the condition must be determined. The condition is formed much like the mask. In this case, there is a binary 1 for each input that must be on and a binary 0 for each input that is either off or masked:

I/O Number	8	7	6	5	4	3	2	1
I/O On?	N	N	N	Y	Y	N	N	Y
Binary Mask	0	0	0	1	1	0	0	1

Since the condition must be in hex or decimal, it can be expressed as:

00011001 (BINARY) equals 19 (HEX) or 25 (DECIMAL),

which equals 1+8+16 (DECIMAL).

Now the mask and the condition can be used in a TIL instruction in the format:

TIL IN&*mask* EQ *condition*

For our example,

```
TIL  IN&39H  EQ  19H  ;THIS USES HEX
                      ;CONSTANTS
```

This accomplishes the same function as the TIL instruction, which refers to inputs one at a time. However, using the IN word allows the function to be done in a less cumbersome manner.

OPERATOR INTERFACE

This section covers interfacing via the serial port. Often, it is necessary to have the BJx send information to the operator or ask the operator for information. For example, it may be useful to output speed and position, or ask the operator for a new speed command. This is easily accomplished using BJx serial I/O instructions.

PRINT (P)

The PRINT (P) command prints text and variables to the terminal. Text and variables may be freely intermixed, limited only by the 80-character maximum instruction length. The following command prints the speed on the terminal:

```
P  "SPEED = "  VFB  " RPM"
```

Assuming VFB is 1962, the BJx will respond with:
SPEED = 1962 RPM

Note that text must be enclosed by double quotes, and that text and/or variables must be separated by at least one blank space.

Printing Decimal Numbers

Variables are normally printed as decimal integers in a field that is 12 characters wide. Formatting can be used to adjust the field width or to print decimal points.

To change the width of the field, follow the variable name with the width enclosed in square brackets ([ ]). Referring to the above example,

```
P  "SPEED = "  VFB[5]  " RPM"
```

will cause the BJx to print:

SPEED = 1962 RPM

If you try to print a number and do not have enough space in the format for the number, then the BJx will fill the format width with X's. For example,

```
P  "SPEED = "  VFB[3]  " RPM"
```

will result in:

SPEED = XXX RPM

(again, assuming the speed is 1962 RPM).

Printing Decimal Points

You can also use the BJx to print a decimal point. The BJx performs calculations with integers. However, it is often desirable to convert integers to floating point numbers, especially when printing out information for the operator. This allows you to make the integer math of the BJx transparent to the operator. For our example, suppose you would prefer to print out the speed in KRPM (thousands of RPM). You can use print formatting to convert the program units (RPM) to KRPM with the following print command:

```
P  "SPEED = "  VFB[5.3] "  KRPM"
```

Assuming VFB is 1962, this command would produce:

SPEED = 1.962 KRPM.

The ".3" following the "5" in the format causes the BJx to insert a decimal point three places from the right of the number. To the operator, this is more convenient, though the programmer still must work in integer units.

You also have the option of printing fewer digits than follow the decimal point. This also can be specified in the format. For example, suppose you only wanted to print one digit after the decimal point. The print command from above would be changed to limit the number of digits to be printed:

```
P  "SPEED = "  VFB[5.3.1]  " KRPM"
```

This command would produce:

SPEED = 1.9 KRPM

So the general format for decimal format is:

[OVERALL WIDTH.DECIMAL POSITION.PRINTABLE DIGITS]

For the example above ([5.3.1]), the overall width was 5, the decimal position was 3, and the number of printable digits after the decimal was 1. You can leave off any of these three specifications. The overall width defaults to 12, the decimal position to zero, and the printable digits to the value of the decimal position.

Printing Hex Numbers

To print a variable in hexadecimal, follow the variable name with an H enclosed in square brackets ([H]). The variable will be printed in a field 9 characters wide, including an appended "H" to indicate hex. The default field width 9 can be changed by following the "H" with the desired field width. For example:

```
X1 = 255
P  "X1 = "  X1[H]
P  "X1 = "  X1[H3]
```

will cause the BJx to print:

```
X1 =      FFH
X1 = FFH
```

Two's-complement notation is used when printing in hex. This means that printing negative hex values requires the full field width of 9 characters. When printing in hex format, the field must be wide enough to include the appended "H."

Printing Binary Numbers

To print a variable in binary format, follow the variable name with a B enclosed in square brackets ([B]). The variable will be printed in a field 33 characters wide, including an appended "B" to indicate binary. All of the leading zeros will be printed. The default field width of 33 can be reduced by following the "B" with the desired field width. For example:

```
X2 = 127
P  "X2 = "  X2[B]
P  "X2 = "  X2[B10]
```

will cause the BJx to print:

```
X2 = 00000000000000000000000001111111B
X2 = 001111111B
```

Printing Switches

Formatted printing can also be used to display switches (any variable with a value of 0 or 1) either as Y or N or as on or off. This allows you to communicate with the operator better than just printing 0 or 1. The switch format (on or off) is printed with a bracketed S ([S]) following the variable:

```
XS1 = 1
P  "USER SWITCH #1 IS "  XS1[S]
```

These commands would result in:

```
USER SWITCH #1 IS ON
```

Similarly,

```
XS1 = 0
P  "USER SWITCH #1 IS "  XS1[S]
```

results in:

```
USER SWITCH #1 IS OFF
```

In addition, you can print a switch as Y or N if you follow the switch with a bracketed Y ([Y]). For example,

```
P  XS1[Y]
```

will print either Y or N depending on whether XS1 is 1 or 0, respectively. This format is useful with the input command which we will discuss later. The input command allows the operator to respond with Y or N and stores 1 or 0 in a BJx variable. This print format allows you to print the previous answer on the screen the way it was entered.

Printing Expressions

The P instruction is not restricted to printing only variables. In general, any numeric expression can be formatted and printed. All the following examples are valid:

```
P  "MINUS 1 IN HEX IS "  -1[H]
P  X1+X3  "= X1+X3"
P  "SENSE OF DIRECTION IS "  DIR*2-1[2]
P  "DISTANCE TO GO IS "  PFNL-PFB[.3]
```

Printing ASCII Characters

The BJx will also convert numbers to ASCII format before printing. You can do this by following the variable or expression with a bracketed C ([C]). This will cause the BJx to print out the character for which the number is an ASCII code. For example,

```
X6 = 65
P  "THE NUMBER " X6[2] " IS THE ASCII
CODE FOR "  X6[C]
```

will result in:

```
THE NUMBER 65 IS THE ASCII CODE FOR A
```

If the number is greater than 127 (that is, the eighth bit is set), the BJx removes the eighth bit before transmitting the character. For example:

```
P  65[C]  " IS THE SAME AS "  128+65[C]
```

since the BJx removes the eighth bit of the expression on the right, which has the end effect of reducing the number by 128. If the number is larger than 255, the BJx divides the variable or expression into 4 bytes and prints them out separately. For example:

```
X2 = 256*256*256*65+256*256*65+256*65+65
P  X2[C]
```

prints:

```
AAAA
```

since the number stored in X2 is equivalent to 4 bytes of 65.

The default field width of the character format is 4, and you can change the field width by following the C with the desired format.

Printing Control Characters

The BJx uses the standard ASCII character set as shown in Appendix B. There are unprintable characters, such as the bell (ASCII 7) and carriage return (ASCII 0DH). These characters have an effect on the terminal but do not print anything on the screen. Unprintable characters range from ASCII 1 to 1FH. The BJx cannot print ASCII 0.

As Appendix B shows, each unprintable character can be produced with a control sequence. For example, most terminals will sound a bell when you press <Control>G (hold down the control key while pressing the G key). As Appendix B shows, <Control>G produces 07 or the ASCII bell. You can use the BJx to produce unprintable characters by preceding the appropriate character with the carat (^) to signify an unprintable character. For example, the following BJx command will sound the bell on your terminal:

```
P  "^G"
```

You can also use the character format to print control characters. For example:

```
P  07[C]
```

also sounds the bell. The character format allows you to print variables as ASCII codes. However, the easiest way to print control characters is normally with the carat (^). One reason for this is that control characters can be within text strings. For example:

```
P  "BELL = <CONTROL>G. ^G SOUNDS
A BELL"
```

If you use the carat to specify an invalid control character, such as ^1, the BJx will print the carat and the 1 ("^1"). Only ^A to ^Z, ^[, ^/, ^], ^^ and ^_ are allowed.

Cursor Addressing

Many displays allow you to address the cursor. For example, the DEP-01 from Kollmorgen is a 40-character display that allows you to address any location from 0 (leftmost top line) to 39 (rightmost bottom line). First, send ASCII 27 ("^[") followed by the address of ASCII 0 ("^@") through ASCII 79 ("O"). For example, you can address the rightmost space of line one (space #39) with the control character sequence ^['. The ^[specifies cursor addressing and "'" (ASCII 39) specifies space #40.

One problem with cursor addressing is that the BJx cannot transmit ASCII 0 (^@). This is a common limitation for terminals. If you want to address space #0, you must first address space #1, then transmit a backspace (ASCII 8 or "^H"). For example, if the following line is executed from the user program while the BJx serial port is connected to the DEP-01, "X" will be printed on space #0.

```
P "^[^A^HX MARKS THE FIRST SPACE"
```

Printing BJx Status (PS)

The PRINT STATUS (PS) command is like the P command except that it appends the BJx status to the end of the printed line. There are five different status words that can be printed with the PS command. Each is listed below with its meaning.

Table 5.4 Printing BJx Status

Status	Explanation
OFF	BJx is OFF
READY	BJx is ready, but REMOTE is OFF.
ACTIVE	BJx is active, but no motion.
FAULT	BJx has a fault condition.
JOG	BJx is jogging.
PROFILE	BJx is executing profile.
GEAR	BJx is in gear mode.

You can use all formats and combinations with PS that you did with P. These results are identical except that the BJx status is appended onto the line.

REFRESH (R & RS) Commands

The REFRESH commands, R and RS, are identical to P and PS, except that R and RS send only a carriage return. The P and PS commands print lines that end with linefeed and carriage return pairs. R and RS commands display lines that can be overwritten.

The following example demonstrates how the REFRESH commands work. Type in this example from the Editor:

```
7$
RS  "VELOCITY FEEDBACK="  VFB
GOTO  7
```

Now exit the Editor and type:

```
RUN  7
```

Rotate the motor shaft by hand so that the velocity feedback changes. Press the escape key and enter the Break command to break program execution. Notice that the velocity is continuously updated, but the line appears to be stationary. A similar program with the P or PS commands would cause the lines to scroll to the top of the screen.

INPUT

The INPUT command causes the BJx to print a message to the terminal and wait for a response from the operator. The input information can be stored in any programmable variable. This allows the operator to change or enter information without making any changes to the program itself.

Type in the following example INPUT instruction:

```
INPUT  "ENTER NEW SPEED : "  X2
```

This causes the BJx to print:

```
ENTER NEW SPEED :
```

Type the new speed into the terminal. After you are prompted, enter a number and press the enter key. The number you enter is stored in the variable X2. If you press the enter key without entering a number, the variable X2 is left unchanged. Use the Print command to display the new value of X2:

```
P  X2
```

INPUT Limits

You can also specify an upper and lower limit for the operator entry. If the above INPUT instruction were written as:

```
INPUT  "ENTER SPEED : "  X2  10  100
```

the BJx would force the operator to input a value between the specified low limit (10) and high limit (100). If the input is invalid or outside the range, an error message is sent and the operator is prompted again.

The limits can be constants, as shown above, as well as any valid numerical expression. If the limits are outside the variable's normal range, they are ignored. If they are not specified at all, the variable's normal range is used as the limit. For example, the limits on ACC are 0 and AMAX. Type in this command:

```
X1=ACC                  ;STORE ACC
INPUT  "ENTER ACC : "  ACC  -1000  1000
```

The BJx knows that the lower limit on ACC is 0 so that no negative numbers will be accepted. If AMAX is less than 1000, AMAX will be the upper limit. Otherwise, 1000 will be the upper limit. If you specify limits outside the variable's program limits, the BJx uses the program limits. Appendix E lists all variables and their program limits.

INPUT and Decimal Point

You can use the INPUT to prompt the operator for values that include a decimal point. You must specify the number of characters after the decimal point. This is the only way you can enter numbers having a fractional part into the BJx. For example, suppose your user position units are mils (0.001 inch). You can prompt the operator for any position in inches with the INPUT. The following example stores the results of the INPUT command in X1. Enter this short program in your BJx, then type RUN 44:

```
44$
INPUT  "ENTER NEW POSITION: "  X1[3]
P  "NEW POSITION = "  X1[.3]
P  "ACTUALLY, X1= "  X1
B
```

Notice the bracketed 3 following X1 in the INPUT command. This causes the operator input to be multiplied by 1000 (10^3) before it is stored in X1. The print statements that follow display X1 in inches (as the operator would prefer to see it), then in mils (as the BJx motion commands process it).

SERIAL Switch

You can use the SERIAL switch to make sure that the serial port is not busy before you execute a command. If SERIAL is on, the serial port is ready. Otherwise, the serial port is not ready. For example, suppose you do not want to execute an INPUT command if the serial port is busy. It might be busy from a print command, or from a previously executed input command. In that case, use these commands:

```
? SERIAL EQ ON INPUT "ENTER SPEED" X1
```

MULTI-TASKING

Multi-tasking, an important feature of the BJx, allows you to write separate *tasks* that run *concurrently*, which means more than one task executes at the same time. For example, you can write a program with two separate tasks: one to ask the operator questions and another to command motion. These two tasks can run independently so that while the operator is answering questions, the motion continues.

Each task has a priority level. The BJx has 6 different task levels as shown in Table 5.5. High priority means that if two tasks need to run at the same time, then the commands from the task with highest priority will execute first. For example, Alarm A has the highest priority. If Alarm A and Alarm B are "fired" at the same time, Alarm A will run until it is complete, then Alarm B will run.

Multi-Tasking and Autobauding

If you set the BJx to autobaud, multi-tasking will not be enabled until communication has been established. This means the BJx will not operate if a terminal or computer is not present. Therefore, you normally will want to disable autobauding by turning ABAUD switch (on the front of the BJx) off.

NOTE

Turn ABAUD Front-Panel Switch off when using multi-tasking.

MULTI

If you want to disable Alarm C, the variable input routine, and background, type:

```
MULTI OFF
```

For example, if you have a time-critical section of code, you may turn MULTI off at the beginning of the section and back on at the end of the section.

END Command

Tasks are normally terminated with the END command. END signifies the end of the task, whereas Break (B) implies that all tasks stop executing. For example, if you end an alarm with the Break command, the entire program stops running and the BJx returns to the Interactive mode. However, if you end an alarm with the END command, the alarm stops, but the other tasks continue running.

Enabling and Disabling Multi-tasking

Multi-tasking is always enabled when a program is running. For example, if you have a program that starts at 55$ and has 2 alarms, then the alarms will be active if you type

```
RUN 55
```

If your program ends with a Break command, then the program will stop executing and multi-tasking will be disabled; that is, the BJx will return to the Interactive mode. If your program ends with an END command, then only the task level that executed the END will stop executing; other tasks will continue executing. If there are no other tasks executing, then the BJx does not return to the Interactive mode but instead becomes dormant. In this case, multi-tasking remains enabled. For example, alarms will continue to be serviced.

If you want to enable multi-tasking without running a particular program, type

```
RUN
```

without entering a label.

Tables 5.6 and 5.7 show how to turn multi-tasking on and off:

Table 5.6 How to Enable Multi-Tasking

1.	Run any label (Type "RUN <label>").
2.	Run multi-tasking (Type "RUN").
3.	Include a POWER-UP$ label and power-up.

Table 5.7 How to Disable Multi-Tasking

1.	Execute a Break from your program.
2.	Enter a Break from the Monitor mode.
3.	Cause an error that breaks execution.

Idling

Idling is a necessary part of multi-tasking. So far in our discussion, higher priority tasks run until they are complete. Actually, commands from the highest priority task that is not idle execute. For example, if an alarm cannot run because it is waiting for some condition (such as waiting for motion to stop), it is *idle*. If a task is running and becomes idle, then a lower priority task can run until the higher priority task is no longer idle. A task can be idled with pre-execution idling and post-execution idling.

Pre-Execution Idle

A task can be idled by waiting for a condition before executing a command. This is called a "pre-execution idle" because the task is idled before executing the command that causes the idle. There are two conditions that can cause a pre-execution idle. A task about to execute a motion command (MI, MA) will be idled if the motion buffer is full. Also, a task about to execute a printing command (P, PS, R, RS, or INPUT) will be idled until the previous printing command is finished.

For example, the BJx can store up to two MI or MA commands. This was called *buffering* in Chapter 3. This means that if you wrote a task with three MI commands in a row, then the third MI command could not be executed until the first move was complete. Thus, the task would be idled until the first move finished. If there was another, lower-priority task, it would not execute until the first move finished. When the first move finished, the first task would no longer be idled and thus would proceed.

Task Level	Task Name	Task Labels	How to Start Task	Typical Uses of Task
1 (Highest Priority)	ALARM A	A$	Hardware or Software Switch	Monitor Inputs
2	ALARM B	B$	Hardware or Software Switch	
3	ALARM C	C$	Hardware or Software Switch	
4	VARIABLE INPUT	VARIABLE$	”ATTN” from DEP-01 or ^V from a PC or a Terminal	Prompt Operator for Input
5	POWER-UP PROGRAM	POWER-UP$	Power-up BJx and Establish Communication	Initialize BJx for Application
	AUTO PROGRAM	AUTO$	Manual Switch Off and Positive Transition Of Cycle Input	Run One Cycle of Auto Program
	MANUAL PROGRAM	MANUAL$	Manual Switch On	Run Manual Program Continuously
	GENERAL PURPOSE PROGRAM	0$ - 500$	Run <LABEL>	General Purpose Programs
	USER ERROR HANDLER	ERROR$	Any Error That Breaks Execution	Gracefully Exit on Error Condition
6 (Lowest Priority)	BACKGROUND PRINT AND MONITOR	BACKGROUND$	All Other Tasks Idle	Print Messages to the Screen

Table 5.5 Multi-Tasking Overview

Consider the following program. It has two tasks: a routine starting at 1$ (Task Level 5) and a background task starting at *BACKGROUND$* (Task Level 6). The background task is the lowest priority task and will only execute when the general purpose task is idle. In the following example, the task is idle between the second and third motion command. Use the BJx Editor to enter this program.

```
;TASK LEVEL 5

1$                      ;MAIN PROGRAM
EN
MI  10000  10           ;FIRST MOVE
P  "FIRST MOVE PROCESSED"
MI  10000  10           ;SECOND MOVE
P  "SECOND MOVE PROCESSED"
MI  10000  10           ;THIRD MOVE
P  "THIRD MOVE PROCESSED"
B
.....

;TASK LEVEL 6

BACKGROUND$
P  "UPPER TASK IDLED"
D  250                  ;DWELL 0.25 SEC.
END
```

Now type:

```
RUN 1
```

The result should be:

FIRST MOVE PROCESSED
SECOND MOVE PROCESSED
UPPER TASK IDLED
UPPER TASK IDLED
. . .
UPPER TASK IDLED
UPPER TASK IDLED
THIRD MOVE PROCESSED

The first and second moves are processed immediately, then Task Level 5 is idled while the first move finishes. While Task Level 5 is idle, the background task executes over and over, printing the simple message on the screen.

Post-Execution Idle

A task also can be idled by waiting for a condition after executing a command. This is called a "post-execution idle" because the task is idled after executing the command that causes the idle. Commands that cause post-execution idling are called *idling commands*. As discussed earlier in this chapter, there are four idling commands:

Wait (W)
Dwell (D)
Hold (H)
Input (INPUT)

For example, you can modify the above program to make one move, then run the background routine until motion has stopped. Enter this program:

```
;TASK LEVEL 5
1$                      ;MAIN PROGRAM
EN
MI  10000  10           ;START MOVE
P  "MOVE PROCESSED"
W  0                    ;WAIT FOR MOVE
P  "ALL MOTION STOPPED"
B
.....
;TASK LEVEL 6
BACKGROUND$
P  "UPPER TASK IDLED"
D  250                  ;DWELL 0.25 SEC.
END
```

Type:

```
RUN 1
```

The result should be:

MOVE PROCESSED
UPPER TASK IDLED
UPPER TASK IDLED
. . .
UPPER TASK IDLED
UPPER TASK IDLED
ALL MOTION STOPPED

Note that Task Level 5 immediately processes the move and then is idled by the Wait command until motion stops. While task 5 is idled, the lower level, background task executes continuously.

Avoiding Idling

You can avoid idling the BJx by using the TIL command in place of Dwell, Wait, or Hold. For example,

```
TIL  SEG  EQ  0
```

is the same as:

```
W  0
```

except the TIL command locks out lower priority tasks since it is not an idling command. The Wait command allows lower level tasks to execute since it is an idling command.

Alarms (Task Levels 1-3)

Alarms are the highest priority tasks. There are three alarms: A, B, and C. A is the highest priority. Normally, alarms are used to monitor hardware inputs, but they can monitor any user switches (XS1 - XS50) and MANUAL. Using an alarm relieves you of having to write your program so that it checks switches. After you define an alarm, the BJx will watch the switch and automatically execute the code that you specify, should the alarm "fire."

Alarms are specified on one line, along with the switch that triggers the alarm and the transition. For example, the A alarm can be defined to fire when input I1 transitions from off to on with this command:

```
A$  I1  ON
```

You can follow the alarm definition with the code that you want to execute when the alarm fires. For example, if I1 turned on, it might indicate an error condition. In this case you might disable the BJx, turn off all outputs, and break execution. The following program would accomplish this using the A alarm.

```
A$  I1  ON          ;DEFINE THE
                    ;ALARM
DIS                 ;DISABLE THE
                    ;BJX
OUT = 0             ;TURN OFF ALL
                    ;OUTPUTS
B                   ;BREAK EXECUTION
```

Restrictions of Alarms

Alarms have many restrictions. 1) You cannot execute GOTO, GOSUB, or RET commands from an alarm. 2) You cannot execute a label. 3) Alarms must be self-contained programs--they cannot "mix" with your program. 4) They must be terminated with an END, Kill (K), or Break (B) command. 5) Also, if all three alarms are present, the execution time of your program increases by about 3%. Most other commands are allowed for alarms, including motion commands and Block-IFs.

Printing with Alarms

You must be careful when executing print commands from alarms. If you need to print from an alarm task, always print after the critical commands have been executed. This is necessary because the input command from a lower task will stop any task, even a higher priority task, from printing. The input command stops all printing until the operator responds with a new value. For example, write your program like this:

```
B$ HOME ON          ;FIRE  ALARM
O1 = 0              ;TURN OFF OUTPUT
DIS                 ;DISABLE DRIVE
P "MESSAGE"         ;NOW,  PRINT A
                    ;MESSAGE
B
```

Do not print before you turn outputs off or disable the BJx. Otherwise, an INPUT command from another task may idle the alarm indefinitely.

Printing can incur large delays.

Do not print before critical commands can execute.

Variable Input (Task Level 4)

The variable input task is the next highest priority. Normally, this task is used to prompt the operator for input, while still allowing the main section of the program to continue. For example, the operator could be entering a new distance while the main program continues executing the program using the old distance. The variable input task is similar to an alarm, except that it is fired upon receiving a special character from the terminal or computer, which is ^V (control-V), or ASCII 16H. The "ATTN" button on the DEP-01 Data Entry Panel from Kollmorgen also transmits a ^V to fire the variable input task.

The variable input task begins with *VARIABLE$*. You can follow that label with various statements, usually printing and input commands. For example, the following program shows Task Level 5 continuing to increment X1 during an INPUT command:

```
;TASK LEVEL 4
VARIABLE$
P  "X1  IS"  X1
INPUT  "INPUT NEW VALUE OF X2" X2
P  "X1 IS NOW "  X1
B                    ;END EXECUTION
.....................
;TASK LEVEL 5
10$
X1 = 0
11$
X1 = X1+1
GOTO 11
```

Now you can enable multi-tasking by typing:

```
RUN  10
```

This program resets X1, then begins to count up. Now enter ^V from your PC or terminal, or ATTN from your DEP-01. The BJx should print the value of X1, which has been continuously incrementing since you typed RUN 10. Next, enter a new value for X2 and notice that the program prints out a new value for X1, which is larger than the value it printed at the beginning of the variable input task. This is because the variable input task was idle while you were entering the new value. Since the higher priority task is idle, the lower priority (11$) will run and continuously increment X1.

Using Variable Input with Profiles

You can use the variable input routine while the BJx is executing motion profiles. However, you must be careful if you are changing parameters of motion. Specifically, if you are changing two or more parameters that you want to take effect at the same time, you must write your program to store those values away. For example, suppose you are using the variable input routine to prompt for speed and distance. You might use a program like this:

```
;TASK LEVEL 4
VARIABLE$
INPUT  "INPUT NEW DISTANCE"  X1
INPUT  "INPUT NEW SPEED"  X2
END                  ;END VARIABLE$
.....................
;TASK LEVEL 5
20$
MI  X1  X2
GOTO 20
```

If you type:

```
RUN  20
```

this program will continuously move the motor X1 distance at X2 speed, even after you press ^V to start the variable input routine. However, after you have entered a new value for X1, the variable input routine will be idled, waiting for you to enter X2. In this case, the next MI command will be executed with the new X1 and the old X2. You can correct this problem by temporarily storing the input values in user variables and loading them all together. For example:

```
;TASK LEVEL 4
VARIABLE$
INPUT  "INPUT NEW DISTANCE"  X11
INPUT  "INPUT NEW SPEED"  X12
X1 = X11             ;LOAD X1 AND X2
                     ;WITH
X2 = X12             ; INPUT VALUES
END                  ;END VARIABLE$
.....................
;TASK LEVEL 5
20$
MI  X1  X2
GOTO 20
```

Temporarily storing the input values in X11 and X12 guarantees that the MI command will execute with either all new or all old values. Since there are no idling commands between the commands that load X1 and X2, there is no possibility for Task Level 5 to run until X1 and X2 are both loaded or neither is loaded.

In addition, if the variable input routine changes variables that are used in different lines of Task Level 5, you probably should turn MULTI off at the beginning of the block of lines and back on at the end. This prevents the variable input routine from

reloading the variables in the middle of a block of lines.

Restrictions of Variable Input

Like alarms, variable input has many restrictions. 1) You cannot execute GOTO, GOSUB, or RET commands from the variable input task. 2) You cannot execute a label. 3) The variable input must be self-contained--it cannot "mix" with other tasks. It must be terminated with an END, Kill (K), or Break (B) command. Again, most other commands are allowed for the variable input task, including motion commands and Block-IFs. If the variable input task is present, the execution time of your program increases by about 1%.

Main Program Level (Task Level 5)

Most of the time, your program will run at Task Level 5. All the program examples given earlier in this chapter executed at Task Level 5. Notice from "Multi-Tasking Overview" that all general purpose labels (0$ - 500$) and many dedicated labels (POWER-UP$, AUTO$, MANUAL$, and ERROR$) share Task Level 5. The routines that follow these labels share one task level and cannot run concurrently. For example, you cannot run AUTO$ and MANUAL$ concurrently. In other words, only one Task Level 5 routine can run at a time.

Alarms and the variable input task are higher priority than Task Level 5. For example, if an alarm fires while your program is running a task that begins at a general purpose label (Task Level 5), Task Level 5 will be suspended until the alarm is complete. The background program (BACKGROUND$) runs at the lowest level. Generally, alarms respond to conditions that are more urgent than most other sections of the program. Similarly, background is for tasks that are not critical, such as printing. Multi-tasking controls which task runs by executing commands from the highest priority task that is not idle.

The rest of this section will discuss the dedicated labels in Task Level 5: POWER-UP$, ERROR$, AUTO$, and MANUAL$.

Power-Up Routine (POWER-UP$)

On power-up, the BJx checks your program to see if you entered POWER-UP$. If you did, the power-up routine is executed. For example, enter the following program:

```
POWER-UP$
X1 = X1+1      ;SAMPLE COMMAND
B
```

Power-down your BJx for a few seconds and power-up again. After establishing communication, the BJx should display the sign-on message followed by:

```
EXECUTING POWER-UP LABEL
-->
```

indicating that the power-up routine was executed.

NOTE

The POWER-UP program will not run during autobauding.

If you want your program to start automatically on power-up, begin it with POWER-UP$. If POWER-UP$ is not found in the program, then the BJx powers-up in the Interactive mode. If the BJx is set to autobaud, then it will not execute the power-up label until communication has been established.

If you want to leave multi-tasking active after your power-up routine is done, end the power-up routine with the END command instead of the Break command. If your routine ends with the END command, then multi-tasking will be enabled, and the Alarms, Background, and other multi-tasking functions will be working. If you want to return to the Interactive mode after power-up, then end the power-up routine with the Break command.

Error Handler (ERROR$)

When a serious error occurs, the BJx breaks execution of your program and checks your program to see if you entered ERROR$. If you did, the error handler (that is, the routine that follows the ERROR$) is executed. All multi-tasking is suspended, including alarms, when the error handler is being executed.

Auto Routine (AUTO$)

If you want to start a program from an external switch, use the auto routine. You can use the auto routine to interface to simple operator panels or to programmable logic controllers (PLCs).

CYCLE (Connector J11) is a hardware input that, under the proper conditions, will cause the BJx to

begin executing one cycle of the auto program. The AUTO program begins at AUTO$.

The following conditions must be met for the BJx to execute the AUTO program.

Table 5.8 To Execute AUTO$...

1. Multi-tasking must be enabled.
2. AUTO$ must be present in the user program.
3. No routines can be executing at Task Level 5.
4. The MANUAL (I1) input must be off.
5. The CYCLE input must be low.

If these conditions are met, the CYCLE READY output will turn on. Then, when CYCLE turns on, the BJx will begin executing the user program at AUTO$.

Manual Program (MANUAL $)

MANUAL$ is based on the MANUAL input, which is shared with I1. The following conditions must be met for the BJx to execute the MANUAL program. When these conditions are met, the BJx will begin executing label MANUAL$.

Table 5.9 To Execute MANUAL$...

1. Multi-tasking must be enabled.
2. MANUAL$ must be present in the user program.
3. No routines can be executing at Task Level 5.
4. The MANUAL (I1) input must be on.

If these conditions are met, the BJx will execute the user program at MANUAL$. You may have noticed that AUTO and MANUAL are very similar. The important difference is that while the AUTO program begins when CYCLE START turns on, the MANUAL program runs continuously.

Typical AUTO/MANUAL Programs

Figure 5.1 shows typical AUTO and MANUAL programs. This flowchart shows the effects of the MANUAL and CYCLE switches. The sample AUTO program causes the motor to rotate one revolution each time the CYCLE switch transitions from off to on. The sample MANUAL program is written so that I2 and I3 are JOG+ and JOG- switches. So when the MANUAL switch (I1) is on, the BJx monitors the jog buttons; when MANUAL is off, the CYCLE button causes the motor to rotate one revolution. Note that both the AUTO and MANUAL programs end with the END command; this is the normal way to conclude these programs.

Background (Task Level 6)

The background task is the lowest priority. Normally, this task is used for non-critical tasks such as refreshing the display and checking low priority inputs. The background task runs continuously, as long as no other task is active.

The background task begins with *BACKGROUND$*. You can then follow that label with various statements, usually printing commands. For example, enter the following program:

```
BACKGROUND$
P  "EXECUTING BACKGROUND"
D  500
                    ;DWELL
END
```

Now you can enable multi-tasking by typing:

```
RUN
```

Notice that you did not need to specify a label. If you type RUN without a label, you will enable multi-tasking without executing a specific label. When you are done with this example, press ^X (control X) to break the program and return to the Interactive mode.

Restrictions of Background

Like alarms, background has many restrictions. 1) You cannot execute GOTO, GOSUB, or RET commands from background. 2) You cannot execute a label. 3) The background task must be self-contained--it cannot "mix" with other tasks. It must be terminated with an END, Kill (K), or Break (B) command. Again, most other commands are allowed for the background task, including Block-IFs. If the background task is present, the execution time of your program increases by about 1%.

Transmit/Receive Programs

The BJx provides two commands that allow programs to be transmitted directly to and from the BJx. These

commands are intended for applications requiring that a computer transmit and receive programs without Motion Link.

<BDS Command Receiving from the BJx

The <BDS command is used to send the BJx user program through the serial port to the terminal or computer. The transmission can be stopped by sending an escape character. You should not rely on the BJx to store all your programs. Keep back-up copies elsewhere. The <BDS command will cause the BJx to transmit the entire user program to your computer. It cannot be issued in the Program mode. For example, from the terminal type:

```
<BDS
```

and the BJx will respond by printing out the entire user program.

The >BDS Command Transmitting to the BJx

The >BDS command is used to send a new user program through the serial port to the BJx. The transmission is ended by sending an escape character. Note that this command writes over the contents of the user program stored in the BJx. This command allows the program to be directly entered, presumably by a computer, to the BJx. It cannot be issued in the program mode.

The >BDS command writes over the entire user program.

The BJx issues the "l->" prompt to indicate that it is ready to load a new program line. If you are loading from a computer, you must wait for the prompt before beginning to transmit a new line. The >BDS command is password protected. If a password is set (see Chapter 2), then it must be given in the >BDS command.

NOTE

Typing in these examples will erase the user program in the BJx. Do not type them in unless your program is backed up.

For example, if a password is not set, type:

```
>BDS
```

and begin transmitting the new program. If you press the escape key before typing anything else, the process will be aborted without changing the program in the BJx.

If a password is set, then that password must follow the command. For example, if the password was set as SECRET, type:

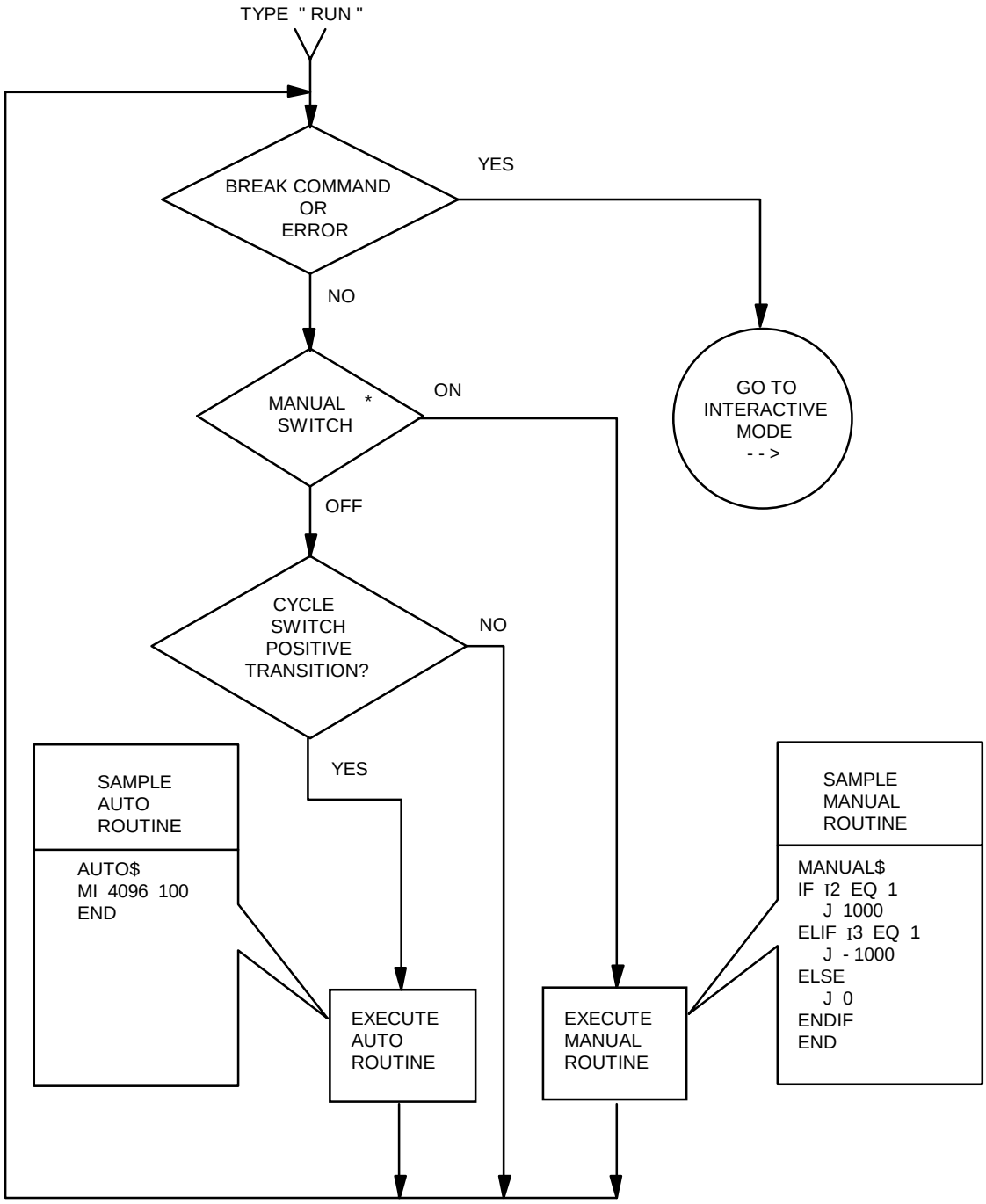
```
>BDS  SECRET
```

and the BJx will accept programs directly from the terminal.

The user program is stored in battery backed-up memory. If the program changes because of a hardware problem, the BJx issues a "USER PROGRAM CORRUPT" error. The >BDS command resets the user program memory, which eliminates this condition.

PROGRAM EXAMPLES

This section lists a typical application program as well as a sample velocity drive program. Use these programs as models for your own. This format uses extensive comments. The assumption is that you are using Motion Link so that these comments will not be transmitted to the BJx, as they would normally take an unacceptable amount of space. You are encouraged to use comments because they make the program easier to understand and correct.
For the velocity drive program, first you must select whether the input will be analog or digital (encoder equivalent). Be sure to set GEARI and GEARO for your application.



*MANUAL SWITCH IS SHARED WITH I1.

Figure 5.1 Auto/Manual Mode Flowchart

```
;
;NAME OF APPLICATION:  PRETZEL MACHINE
;
;DATE                  A.E. NEUMAN
;
;REVISION HISTORY:
;       8-9-95  ADDED JOG BUTTONS
;       7-17-95 CORRECTED TEACH BUG
;
;
;-------------------------------------------------
;ALARM DESCRIPTION
;
;       A$                      WATCH THERMOSTAT
;       B$,C$                   NOT USED
;       VARIABLE$               FILL X1 WITH SPEED
;       BACKGROUND$             BACKGROUND PRINTING
;
;-------------------------------------------------
;
;I/O DESCRIPTIONS
;
;GENERAL PURPOSE INPUTS
;       I2      JOG+ PUSH BUTTON
;       I3      JOG- PUSH BUTTON
;       I4      TEACH POSITION PUSH BUTTON
;       I5      CONTACTOR INTERLOCK SWITCH
;       I6      PLC INTERFACE
;       I7      HOME REQUEST PUSH BUTTON
;       I8      THERMOSTAT
;
;GENERAL PURPOSE OUTPUT
;       O1      COOLING FLUID PUMP
;       O2      SPINDLE MOTOR CONTACTOR
;       O3      PLC INTERFACE
;
;DEDICATED I/O
;       CYCLE CONNECTED TO PLC
;       GATE            NOT USED
;       HOME            CONNECTED TO HOME LIMIT SWITCH
;       LIMIT           CONNECTED TO OVERTRAVEL LIMIT SWITCH
;       MANUAL          (SHARED WITH I1)
;       MOTION          CONNECTED TO STOP PUSH BUTTON
;       STATUS          NOT USED
;
;-------------------------------------------------
;
;USER VARIABLES
;       X1      STORE NUMBER OF CYCLES RUN
;       X2      STORE LAST POSITION RUN TO
;       X3      INTERMEDIATE CALCULATION
;       X4      LOOP COUNTER
;       X5      LOOP COUNTER
;       X6-X250 NOT USED
;       ;
```

```
;USER SWITCHES
;       XS1-XS50        NOT USED
;
;-------------------------------------------------
;
;APPLICATION PROGRAM
;
POWER-UP$                       ;POWER-UP LABEL
PLIM  OFF                       ;SOFTWARE LIMITS NOT USED HERE
;CONTINUE YOUR POWER-UP PROGRAM HERE
END
;
;
A$  I8  OFF
P  "THERMOSTAT (INPUT I8) OPENED"
P  "PROCESS BEING CLOSED DOWN"
DIS                             ;DISABLE THE BJX
B                               ;BREAK PROGRAM EXECUTION
;
;
VARIABLE$
INPUT  "ENTER NEW SPEED"  X1
END
;
;
AUTO$                           ;AUTO LABEL
;WRITE YOUR AUTO PROGRAM HERE
END
;
;
MANUAL$                         ;MANUAL LABEL
;WRITE YOUR MANUAL PROGRAM HERE
END
;


;WRITE MORE OF YOUR PROGRAMS HERE
END
;
BACKGROUND$
;WRITE YOUR BACKGROUND PRINTING ROUTINE HERE
END
;
;
ERROR$                          ;ERROR HANDLER
;WRITE YOUR ERROR HANDLER HERE
B                               ;END OF SAMPLE PROGRAM
```

```
;VELOCITY DRIVE SAMPLE PROGRAM
;DATE                    NAME
;-----------------------------------------------------------------

POWER-UP$                ;EXECUTE ON POWER UP
PL OFF                   ;DISABLE THE POSITION LOOP

VNUM=447392              ;SETS VELOCITY UNITS TO RPM.
VDEN=100

ANUM=447392              ;SETS ACC UNITS TO RPM/SEC
ADEN=100000
;
AMAX=100000              ;SET THE MAX ACCEL RATE (RPM/SEC)
ACC=1000                 ;SET THE NORMAL ACCEL LIMIT
DEC=1000                 ;SET THE NORMAL DECEL LIMIT
;ACC AND DEC ARE RAMP LIMITS FOR GEAR MODE,
;ASSUMING THAT PL IS OFF.
;
;
GEARI=10                 ;THIS SETS THE GEAR MODE FOR 25%,
GEARO=40                 ;APPROX. 10 V = 3000 RPM FOR AN
                         ;ANALOG INPUT.  THE PROPER LEVEL OF
                         ;GEARI AND GEARO DEPENDS ON THE
                         ;SYSTEM AND THE INPUT FORMAT.  THE
                         ;ADJUSTMENT OF GEARI AND GEARO IS
                         ;EQUIVALENT TO A DC GAIN ADJUSTMENT OR
                         ;SCALE FACTOR POT FOUND ON MANY
                         ;ANALOG DRIVES.

;NOTE THAT ACC/DEC RATES ARE LIMITED BY ACC AND
;DEC ONLY WHEN PL IS OFF.
;
EN                       ;ENABLE DRIVE
GEAR  ON                 ;ENABLE ELECTRONIC GEARBOX
VOFF=0                   ;THIS SETS THE OFFSET VELOCITY.
                         ;VOFF IS SET TO ZERO WHEN GEAR IS
                         ;TURNED ON.
;IF THERE IS NEED TO ADJUST FOR VELOCITY
;DRIFT IN THE INPUT, THEN ADJUST VOFF
;TO THE PROPER LEVEL SO THAT DRIFT STOPS.
;
B                        ;DRIVE IS NOW IN ELECTRONIC GEARBOX
                         ;END OF SAMPLE PROGRAM
```

CHAPTER 6
DEBUGGING

INTRODUCTION

The information in this chapter will help you rectify problems you may have while programming the BJx. When you write programs, you may inadvertently include a few errors or *bugs*. The most effective method of dealing with errors is to prevent their occurrence by following the programming practices provided in this manual. Every effort has been made to make the BJx language as simple as possible with BASIC-like commands, algebraic math, and a variety of conditional commands. Still, some bugs are almost certain to surface in a new program. The BJx provides two execution modes to help you debug your program: Trace and Single-Step.

DEBUGGING MODES

Single-Step

If an error occurs in a section of your program that is not time-critical, you can use single-stepping to help track down the error. When you execute your program in the Single-Step mode (SS), each command is printed out. The BJx waits for you to press the ENTER key before executing the command. Use the nested-IF example given in Chapter 5. Enter the program, set X1 and X2 equal to 1, and turn SS on by typing *SS ON*. Begin execution at label 55 by typing *RUN 55*. The following line should be displayed:

```
55$
S-->
```

Press the ENTER key and the response should be:

```
IF X1 GT 0
S-->
```

You can probe the BJx variables from the Single-Step mode without stopping your program. For example, type:

```
P  X1
```

and the BJx should respond with:

```
      1
S-->
```

In this case, the BJx executed the print command and displayed the single-step prompt, indicating it is ready for another command. Now press the ENTER key repeatedly to step through the program.

This example shows several characteristics of the Single-Step mode:

- All commands are preceded by the single-step prompt:

```
S->
```

- Print statements are active in the Single-Step mode. Notice that the results of the P command are printed normally.

- Only the executed commands in the IF, ELIF, ELSE, and ENDIF sets are shown. Notice that none of the commands following the first print command are shown.

- You can execute commands from the Single-Step mode.

You can also enter the Single-Step mode from your program. To do this, you should include *SS ON* in your program. To exit the Single-Step mode, you can include *SS OFF* in your program or type it from the single-step prompt. You can also press the escape key twice.

Trace

If the error occurs in a section of your program that is not time-critical, you can use Trace to help track down the error. When you execute your program in the Trace mode (TRC), each command is printed out just before it is executed. Use the nested-IF example given in Chapter 5. Enter the program, set X1 and X2 equal to 1, and turn TRC on (*TRC ON)*. Begin execution at label 55 (*RUN 55)*, and the following lines should be displayed:

```
T...55$
T...IF X1 GT 0
T... IF X2 GT 0
T...  P "BOTH X1 AND X2 > 0"
BOTH X1 AND X2 > 0
T...  ELSE
T...  ENDIF
T...ELSE
T...ENDIF
T...B
-->
```

This example shows several characteristics of the Trace mode:

- All commands are preceded by the trace prefix:

```
T...
```

- Print statements are active in the Trace mode. Notice that the results of the P command are printed just below where the print command is displayed.

- Only the executed commands in IF, ELIF, ELSE, and ENDIF sets are shown. Notice that none of the commands following the first print command are shown. This helps you debug your program by only showing the commands that are executing.

- You cannot type in commands from your terminal while the BJx is executing in the Trace mode.

You can also enter the Trace mode from your program. To do this, include *TRC ON* in your program. To exit the Trace mode, include *TRC OFF* in your program or press the escape key twice.

Motion Link and Trace

Motion Link is the software communications package provided for the IBM-PC and compatibles. IBM-PC and compatibles can communicate at 9600 baud only in that they can receive and transmit a character at that frequency. However, they cannot receive an indefinite number of characters at that rate because the computers are not fast enough to process the characters. This leads to a problem in the Trace mode because the BJx can transmit characters much faster than most PCs can process them. This can lead to a delay of minutes from when the BJx transmits a character to when the computer displays it. The best way to cure this problem is to reduce the baud rate from Motion Link (use the ^U command), and power the BJx down and then up to cause a second autobaud (make sure ABAUD is on before powering down). Start with 1200 baud and see if the problem is cured.

DEBUGGING AND MULTI-TASKING

If your program uses multi-tasking, the Trace and Single-Step modes show you which level is currently being executed. For example, enter the program given on page 75. Turn on the Trace mode and type:

```
RUN 1
```

The result should be something like this:

```
T...1$                  ;MAIN
                        ;PROGRAM
T...EN
T...MI  10000  10       ;START MOVE
T...P  "MOVE PROCESSED"
MOVE PROCESSED
T...W  0                ;WAIT FOR
                        ;MOVE
T.*.BACKGROUND$
T.*.P  "UPPER TASK IDLED"
UPPER TASK IDLED
T.*.D  250              ;DWELL 0.25
                        ;SEC.
T.*.END

...

T.*.BACKGROUND$
T.*.P  "UPPER TASK IDLED"
UPPER TASK IDLED
T.*.D  250              ;DWELL 0.25
                        ;SEC.
T.*.END

(AT THIS POINT, ASSUME MOTION
STOPS AND TASK 5 IS NOT IDLED)

T...P  "ALL MOTION STOPPED"
ALL MOTION STOPPED
T...B
```

Notice that when the example is executing the background level task, an asterisk (*) is printed. Each task level prints out a different prompt in the Trace and Single-Step modes, as the following table shows:

Table 6.1 Multi-Tasking Debug Prompts

TASK LEVEL PROMPT	SINGLE-STEP PROMPT	TRACE PROMPT
Alarm A	s-A>	t.A.
Alarm B	s-B>	t.B.
Alarm C	s-C>	t.C.
Variable Input	s-V>	t.V.
Main Program	s-->	t...
Background	s-*>	t.*.

Removing Code

If you cannot find the bug in your program with Single-Step or trace, then remove sections of your code that you do not think are causing the problem a few lines at a time. (Save the original program on your computer for later use.) Removing lines you suspect are causing the problem can provide false leads. For example, the problem may be interaction between a section you removed (which was operating properly) and another, unsuspected section of your program (that was the actual source of the problem).

The best situation is when you can make a short (< 20 line) program demonstrate the problem, and then it is usually easy to determine the problem. You can call Kollmorgen for help. However, in order to make efficient use of your time and ours, you must trim down your program to a few lines that are not working. It is difficult for even a skilled person to help debug a large program over the telephone.

HINTS

The following section offers some hints concerning the most common problems. Most problems arise from a minor misuse or misunderstanding of a BJx function.

If you change your program in the Motion Link Editor and the program function does not change, you may have forgotten to transmit your updated program to the BJx.

If you command motion with MI, MA, J, JT, or JF and the motor does not move...

...make sure GATEMODE is not preventing motion (turn GATEMODE off if you are not certain).

...make sure CLAMP is not preventing motion (turn CLAMP off if you are not certain). If it is CLAMP, try raising the clamp limit, PECLAMP, somewhat. If that does not help, turn CLAMP off. If you now get PE OVERFLOW errors, it may be because the motor is undersized. See the hints for PE OVERFLOW errors below.

...make sure REG is not preventing motion (turn REG off if you are not certain). If REG is on, you may not be feeding in the master encoder signal properly. Remember, it must always count up. Check VEXT. It should be

greater than zero for profile regulation to work.

...make sure all tuning constants are well above zero. Check KP, KV, KVI, and KPROP. Each should be at least 100; generally, they are above 1000.

...make sure ILIM is not too small. If ILIM is below 10%, the motor may not be able to overcome frictional load.

...make sure you are commanding a visible speed. The BJx can command speeds as low as .0004 RPM or about one revolution every three days, depending on how you program velocity units. If you have changed VNUM or VDEN from the factory setting, temporarily restore them to see if the problem goes away.

If the motor moves and you get "PE OVERFLOW" error (ERROR 25)...

...if the error occurs occasionally, it may be because you have the limit (PEMAX) set too low. Raise it by 20% and see if the problem is corrected.

...use the BJx RECORD function to record ICMD when a PE overflow occurs. If ICMD is saturating (that is, equal to ILIM for more than a few milliseconds), you are commanding motion that your motor cannot perform. See hints on motor loading, ILIM, ACC, DEC, and PEMAX below. If the overflow occurs at high speeds and with low ICMD (below ILIM), see the hint about overspeed errors.

...make sure the load does not exceed the capability of the motor.

...make sure ILIM is set high enough.

...if the error occurs during acceleration or deceleration, make sure ACC and DEC are not set too high. If they are too high, the commanded profile will exceed the capability of the motor.

If overspeed errors occur (ERROR 13)...

...if the error occurs occasionally, it may be because you have the limit (VOSPD) set too low. Raise it by 20% (or as high as 120% of VMAX) and see if the problem is corrected.

...if it happens on acceleration, the motor may be tuned improperly. Is your motor overshooting or ringing? Retuning the motor should correct the problem.

If the system works differently on power-up than it does after your program starts running, remember that many switches are reset on power-up. Your program may set a switch that is cleared, or clear one that is set during the initial cycle, causing the program to operate differently. You may also be setting or clearing switches in your power-up routine that may have the same effect.

ERROR LOG

The BJx responds to a variety of conditions, both internal and external, hardware and software, which are grouped in a single broad category: errors. An error indicates a problem somewhere. More serious errors are grouped as faults.

Error Levels

BJx response to an error depends on that error's severity. There are four levels of severity, listed below in increasing order.

Table 6.2 Error Severity Levels and Actions

1.	Errors that cause warnings.
2.	Errors that cause a program break and stop motion, in addition to Level 1 Actions.
3.	Errors that disable the system, set the FAULT LED, and clear the OK LED/ Output, in addition to Level 2 Actions.
4.	Errors that disable almost all BJx functions (including communications) and flash the FAULT LED to indicate the error number. These are called firmware errors.

When any error except a firmware error occurs, a message is displayed on the screen. The following items are printed: the error number, the offending entry, and an abbreviated error message. For example, disable the drive and type in a jog:

```
DIS
J 100
```

The BJx will respond with:

```
ERR  50 'J 100'     BJX INHIBITED
```

The error number (50), the offending entry (the whole line), and the error message (you cannot command a jog when the drive is inhibited) are given on one 80-character line. The error message starts at character 40 so that if a 40-character display is used, the error message will not be printed. You can display the line directly, either with the Motion Link Editor (GOTO A LINE NUMBER selection or ^Q^I), or with the BJx Editor (P command).

Sometimes only an entry is bad and not the whole line. In this case only the bad entry is printed. For example,

```
PROP  2
```

generates:

```
ERR   83  '2'   ;BAD OR OUT OF
                ;RANGE
```

since PROP is a switch and cannot be set to 2. If the error comes from the program, the line number of the offending entry is also printed. Use the Editor to enter these lines at the top of the user program:

```
11$
PROP  2
B
```

Exit the Editor and type:

```
RUN  11
```

and the response should be:

```
ERR  83  LINE 2  '2'   ;BAD OR OUT
                       ;OF RANGE
```

This message shows that the error occurred on line 2. You can enter the Editor and type:

```
P  2
```

and the response will be:

```
PROP 2
```

DEP

If your BJx prints to a Data Entry Panel (DEP-01) or any other 40-character wide display, the standard error messages will not print properly, because error messages are based on an 80-character wide display . To correct this problem, the BJx provides the DEP switch which, when turned on, cuts all error messages down to 40 characters. If your BJx prints to a DEP-01, type:

```
DEP  ON
```

Error History

The BJx stores the twenty most recent errors in the Error History. To display the entire Error History, type:

```
ERR  HIST
```

This causes the Error History to be sent to the terminal, with the most recent error sent first. When the BJx is powered-up, a "DRIVE POWERED UP" message is inserted into Error History even though this is not an actual error.

To clear the Error History, type:

```
ERR  CLR
```

Error History remains intact even through power-down.

Displaying Error Messages

The ERR command can also be used to display an abbreviated description of the error. For example, type:

```
ERR  50
```

The BJx responds with:

```
ERR  50 BJX INHIBITED
```

You may display messages for errors from 1 through 999. If you type in an error number that the BJx does not recognize, it will respond with:

```
ERROR NOT FOUND
```

A description of all errors is given in Appendix D.

Firmware Errors

Firmware errors are an indication of a serious problem with the BJx. These errors stop communications, disable the drive, and flash the FAULT LED. The FAULT LED flashes several times, then turns off and pauses. The number of flashes represents the error number. These error numbers range from 2 to 9. See Appendix D for information on these errors. Contact the factory should one of these errors occur.

APPENDIX A
WARRANTY INFORMATION

Kollmorgen Corporation warrants that equipment delivered by it to the Purchaser will be of the kind and quality described in the sales agreement and/or catalog and that the equipment will be free of defects in design, workmanship, and material.

The terms and conditions of this Warranty are provided with the product at the time of shipping or in advance upon request.

The items described in this manual are offered for sale at prices to be established by Kollmorgen and its authorized dealers.

APPENDIX B
ASCII TABLE

The chart on the following pages is an ASCII Code and Hexadecimal conversion chart. The BJx does not support extended ASCII (128-255).

ASCII CODE AND HEX CONVERSION CHART

00 NUL 0	10 DLE ^ P 16	20 SP 32	30 0 48	40 @ 64	50 P 80	60 ` 96	70 p 112
01 SOH ^ A 1	11 DC1 ^ Q 17	21 ! 33	31 1 49	41 A 65	51 Q 81	61 a 97	71 q 113
02 STX ^ B 2	12 DC2 ^ R 18	22 “ 34	32 2 50	42 B 66	52 R 82	62 b 98	72 r 114
03 ETX ^ C 3	13 DC3 ^ S 19	23 # 35	33 3 51	43 C 67	53 S 83	63 c 99	73 s 115
04 EOT ^ D 4	14 DC4 ^ T 20	24 $ 36	34 4 52	44 D 68	54 T 84	64 d 100	74 t 116
05 ENQ ^ E 5	15 NAK ^ U 21	25 % 37	35 5 53	45 E 69	55 U 85	65 e 101	75 u 117
06 ACK ^ F 6	16 SYN ^ V 22	26 & 38	36 6 54	46 F 70	56 V 86	66 f 102	76 v 118
07 BEL ^ G 7	17 ETB ^ W 23	27 ‘ 39	37 7 55	47 G 71	57 W 87	67 g 103	77 w 119
08 BS ^ H 8	18 CAN ^ X 24	28 (40	38 8 56	48 H 72	58 X 88	68 h 104	78 x 120
09 HT ^ I 9	19 EM ^ Y 25	29) 41	39 9 57	49 I 73	59 Y 89	69 i 105	79 y 121
0A LF ^ J 10	1A SUB ^ Z 26	2A * 42	3A : 58	4A J 74	5A Z 90	6A j 106	7A z 122
0B VT ^ K 11	1B ESC ^ [27	2B + 43	3B ; 59	4B K 75	5B [91	6B k 107	7B { 123
0C FF ^ L 12	1C FS ^ \ 28	2C , 44	3C < 60	4C L 76	5C \ 92	6C l 108	7C | 124
0D CR ^ M 13	1D GS ^] 29	2D - 45	3D = 61	4D M 77	5D] 93	6D m 109	7D } 125
0E SO ^ N 14	1E RS ^ ^ 30	2E . 46	3E > 62	4E N 78	5E ^ 94	6E n 110	7E ~ 126
0F SI ^ O 15	1F US ^ _ 31	2F / 47	3F ? 63	4F O 79	5F _ 95	6F o 111	7F DEL 127

ASCII CODE AND HEX CONVERSION CHART (CONTD)

80 128	90 144	A0 160	B0 176	C0 192	D0 208	E0 224	F0 240
81 129	91 145	A1 161	B1 177	C1 193	D1 209	E1 225	F1 241
82 130	92 146	A2 162	B2 178	C2 194	D2 210	E2 226	F2 242
83 131	93 147	A3 163	B3 179	C3 195	D3 211	E3 227	F3 243
84 132	94 148	A4 164	B4 180	C4 196	D4 212	E4 228	F4 244
85 133	95 149	A5 165	B5 181	C5 197	D5 213	E5 229	F5 245
86 134	96 150	A6 166	B6 182	C6 198	D6 214	E6 230	F6 246
87 135	97 151	A7 167	B7 183	C7 199	D7 215	E7 231	F7 247
88 136	98 152	A8 168	B8 184	C8 200	D8 216	E8 232	F8 248
89 137	99 153	A9 169	B9 185	C9 201	D9 217	E9 233	F9 249
8A 138	9A 154	AA 170	BA 186	CA 202	DA 218	EA 234	FA 250
8B 139	9B 155	AB 171	BB 187	CB 203	DB 219	EB 235	FB 251
8C 140	9C 156	AC 172	BC 188	CC 204	DC 220	EC 236	FC 252
8D 141	9D 157	AD 173	BD 189	CD 205	DD 221	ED 237	FD 253
8E 142	9E 158	AE 174	BE 190	CE 206	DE 222	EE 238	FE 254
8F 143	9F 159	AF 175	BF 191	CF 207	DF 223	EF 239	FF 255

This side of the table is provided for Decimal to Hex Conversion.
The BJx does not support extended ASCII (128-255) Decimal to Hex Conversion.

Appendix C
Software Commands

EXPRESSIONS AND SYMBOLS

<Label>$ — One or two digits followed by a dollar sign. When using GOSUB or GOTO, a user variable can be used as <Label> if its value is between 0 and 99.

<Time> — Specifies time in milliseconds. Must be between 0 and 2,147,483,647 (about 25 days).

<Logical> — One of the following: GT, GE, LT, LE, EQ, or NE for greater-than, greater-than-or-equal-to, less-than, less-than-or-equal-to, equal-to, or not-equal-to, respectively.

<Expr> — Any valid math expression. Valid math expressions include user variables, indirect references to user variables, constants, algebraic and logical math operations, parenthesis.

Examples of valid expressions are:

X1*X2*X3
(X2-VFB)/VOFF
X1&07FH
PFB-PCMD
TMR1/100
(X1+X2)*(X1+(X2-X3)

<Position> — Any valid expression for position. The result is assumed to be in position units. The range is +/-2,147,483,647 counts. If your system has position units, then the limits are the position unit equivalent of +/-2,147,483,647.

<Velocity> — Any valid expression for velocity. The results is assumed to be in velocity units.

<Text> — <Text> is any text string of characters. The control character symbol (^) converts the succeeding character to a control character.

{} — Indicates an optional parameter.

Constants - ON, OFF, Y, and N — ON and Y are equivalent to 1. OFF and N are equivalent to 0. The constants can be used in any expression and in response to the Input command.

COMMANDS

The following commands are the instructions used to program the BJx

;

Comment. Comments can follow any instruction. Also, entire lines can be comments. The semicolon must be preceded by a space unless it is the first character in a line. Allowed on any line including the BJx Editor.

```
GOTO 5          ;THIS IS A COMMENT FOR A COMMAND
                ;THIS ENTIRE LINE IS A COMMENT
```

$

Labels. Labels can be 0-500 and cannot be repeated. They must be decimal constants. They are allowed only from the user program. The following labels are special purpose labels:

A$	A alarm label
B$	B alarm label
C$	C alarm label
VARIABLE$	variable input label
POWER-UP$	power-up label
AUTO$	AUTO label
MANUAL$	MANUAL label
ERROR$	error handler label
BACKGROUND$	background label

Alarm labels require that you specify the switch that starts the alarm and the state of the switch (ON or OFF) that should trigger the alarm. If the switch is in the specified state when execution is enabled, the alarm will be fired. Otherwise, the alarm is edge sensitive. Specifying ON is actually specifying the positive edge.

Format: <Label>$
<Alarm Label>$ <Switch> <On/Off>

Example:

```
55$
BACKGROUND$
A4 I1 ON
```

?

Quick If. Conditionally executes one instruction if the condition is true, and another instruction if the condition is false. Allowed from the interactive and monitor modes, and the user program.

Format: ?<Condition> {Instruction} {:} {Instruction}

Example:

```
? PFB GT 100 P PFB
? X1 EQ 1 P "X1 = 1" : P "X1 <> 1"
?X1*X2 NE X4/(X5+5) B
? LIMIT EQ ON : P "LIMIT IS OFF"
```

<Condition> is the same as <Expr> <Logical> <Expr>.

<Instruction> is any instruction except TIL.

B

Break program execution. Allowed from the user program or the monitor mode.

Format: B

CLEARX

Clear all User Registers and User Flags. X0-X250 and XS0-XS50 will be cleared. Allowed from the Interactive and Monitor modes and the User Program.

Format: CLEARX {1|2}
{1} will optionally clear only the User Registers
{2} will optionally clear only the User Flags

CONTINUE

Continue motion at the present speed. Turn REG and GEAR off. Optionally, you can specify the number of milliseconds, up to 1 second, that you want the present speed averaged over. If this time is not specified, the speed is averaged over 1 millisecond.

Format: CONTINUE
CONTINUE <time>

Example:

```
CONTINUE 100        ;AVERAGE SPEED FOR .1 SEC.
```

D

Delay program execution for a specified amount of time, up to 2,147,483,647 milliseconds or 25 days. D is an idling command (that is, if you are using multi-tasking, D suspends the task but lets other tasks proceed). Allowed only from the user program.

Format: D <Time>

Example:

```
D 1000        ;DWELL FOR 1 SECOND
```

DIS

Disable the BJx. This command turns off the variable READY. Refer to **Drawing C-84732 for more information. Allowed from the interactive mode, monitor mode, and user program.

Format: DIS

DUMP

Display all the variables and the user program on the terminal, or display the version. Allowed from interactive. Drive must be disabled.

Format:
```
DUMP              ;Dump variables and program
DUMP VERSION      ;Dump firmware version
```

ELIF

Part of block if. Conditionally begins block execution. Allowed from the user program. (See the IF command.)

Format: ELIF <Expr> <Logical> <Expr>

Example:

ELIF PFB GT 100

<Expr> <Logical> <Expr> is the condition.

ELSE

Part of block if. Begins last block execution. Allowed from the user program. (See the IF command.)

Format: ELSE

EN

Enable the BJx. This command turns on the variable READY. Refer to Drawing **C-84732 for more information. Allowed from the interactive mode, monitor mode, and user program.

Format: EN

END

End a task. If you are using multi-tasking, END ends that task. If there are no special labels present in the program (except POWER-UP$), then END is equivalent to Break (B). If there are special labels, the BJx becomes inactive waiting for a task to resume execution.

Format: END

ENDIF

Part of block if. Ends block if. Allowed from the user program. (See the IF command).

Format: ENDIF

ERR

Display an error message, display the error history, or clear the error history. Allowed from the interactive and monitor modes and user program.

Format:
```
ERR <Error Number>
ERR <Option>
```

Where <Error Number> is a valid error number and <Option> can be HIST or CLR.

Example:

```
ERR 25          ;DISPLAY MESSAGE FOR ERR 25
ERR HIST        ;DISPLAY ERROR HISTORY
ERR CLR         ;CLEAR ERROR HISTORY
```

GOHOME

Move to home position based on either the HOME input (J17) or the index pulse of the feedback encoder (J12).

Format: GOHOME<Vel1>{Vel2}{source}
Vel1 is the first (fast speed)
Vel2 is the second (slow speed)
Source is "HOME" or "INDEX"

GOSUB

Go to a subroutine. Allowed only from the user program.

Format: GOSUB <Label>

Example:

```
GOSUB 25
GOSUB X3
```

GOTO

Go to a program label. Allowed only from the user program.

Format: GOTO <Label>

Example:

```
GOTO 25
GOTO X5
```

H

Delay (Hold-up) execution of a task until a switch is in the specified state. You can use any switch except REMOTE and XS11-XS50 (XS1-XS10 are allowed). H is an idling command; if you are using multi-tasking, H suspends the task but lets other tasks proceed. Allowed only from the user program.

Format: H <Switch> <ON/OFF>

Example:

```
H XS1 ON
H I1 OFF
```

IF

Conditionally execute a block of instructions. Allowed from the user program.

Format: IF <Expr> <Logical> <Expr>

Example:

```
IF PFB GT 100
...                    ;FOLLOW WITH ELSE, ETC.

IF X1*X2 NE X4/(X5+5)
...                    ;FOLLOW WITH ELSE, ETC.
```

INPUT

Prompt the operator for an input variable. If limits are specified, then make sure operator stays within them. If they are not specified, then use the limits of the variable being prompted for. W is an idling command (that is, if you are using multi-tasking, INPUT suspends the task until the operator presses the enter key, but lets other tasks proceed). Allowed only from the user program.

Format: INPUT "<Text>" <Variable>{decimal}{Min}{Max}

Where <Variable> is any valid, programmable variable. You can optionally specify maximum and minimum limits (if you include one, you must include the other). {Min} is the minimum input allowed and {Max} is the maximum input allowed.

If you specify decimal, the input received from the operator will be multiplied by 10^{decimal}. The BJx does not use floating point math internally. The input command allows you to receive floating point input from the operator.

Example:

```
INPUT "ENTER NEW SPEED" X1[3] -5000 5000
INPUT "ENTER NEW CURRENT LIMIT" ILIM
```

In the first example, if the operator entered 1.234, the BJx would store 1234.0 in X1; that is, 1.234 is multiplied by 10^3 = 1000. Note that if you specify {decimal}, {Max} and {Min} limit the value after the multiplication. In the above example, {Max} = -5000 limits the operator to -5.000.

J

Jog at a continuous speed. Allowed from the interactive mode and the user program.

Format: J <Velocity>

Example:

```
J 1000
J X1
```

JF

Jog, but wait until the Position command (PCMD) crosses the specified position before beginning accel/decel. Speed must not be zero when executing this instruction. Allowed from the interactive mode and the user program.

Format: JF <Position> <Velocity>

Example:

```
JF 10000 10
JF 100*X1 4000
```

JT

Jog at a continuous speed, but delay beginning accel/decel so that the Position command will equal the specified position when the accel/decel is complete. Allowed from the interactive mode and the user program.

Format: JT <Position> <Velocity>

Example:

```
JT -610000 100
JT 100*X45 -800
```

K

Disable the drive and break the program. Allowed from interactive and monitor modes and the user program. See Drawing **C-84732 for more information.

Format: K

MA

Move to the specified position at the specified speed. If the speed is not specified, it is assumed to be VDEFAULT. Allowed from the interactive mode and the user program.

Format: MA <Position> {Velocity}

Example:

```
MA 10000 1000      ;MOVE AT 1000
MA 0               ;MOVE TO 0 AT VDEFAULT
```

MI

Incrementally move the specified distance at the specified speed. If the speed is not specified, it is assumed to be VDEFAULT. Allowed from the interactive mode and the user program.

Format: MI <Position> {Velocity}

Example:

```
MI 10000 1000      ;MOVE AT 1000
MI -1000           ;MOVE BACK 1000
```

NORM

Normalize the Position command and position feedback to the specified position. Allowed from the interactive mode and the user program when there is no commanded motion.

Format: NORM <Position>

Example:

```
NORM 1000
```

P

Print the variables specified with optional formats on a new line. Allowed from the interactive and monitor modes and the user program.

Format: P <Expr> {format} | "<Text>" ...

Where {format} is the print format specifying field width and Hex output. The ellipsis (...) indicates that the P can be followed by up to 15 different expressions and text strings.

Format can be:		
	B	Binary
	H	Hex
	S	ON or OFF
	C	ASCII Character
	Blank	Decimal Integer
	nn.m	Floating Point Output where nn is the total number of digits; m is the number of digits after the decimal point.
	nn.m.p.	Same as nn.m except only print p digits after the decimal point (p must be less than m).

Examples:

```
P PFB VFB IMON              ;PRINT 3 FEEDBACK VARS
P PFB[4]                    ;PRINT PFB IN 4 CHARS
P IN[H]                     ;PRINT INPUT IN HEX
P IN[5H]                    ;PRINT INPUT, 5 HEX CHARS
P 123456[.4]                ;PRINT 12.3456
P 123456[.4.2]              ;PRINT 12.34
P "BJX"                ;PRINT BJX ON THE SCREEN
P "XPOS=" PFB               ;PRINT PFB WITH TEXT
```

PA

Append information to the display. This command is the same as the "P" command except that no Carriage Return and no Line Feed is printed at the end of the line. This command can be used to append information to the last position printed to. Allowed from the Interactive and Monitor modes and the User Program. See the "P" command for formats and examples.

Format: PA<expr>{format} | "<Text>" ...

PAS

Append information to the display. This command is the same as the "PA" command, except that the status field is printed at the end of the line. The status field is an additional 8 characters wide. Allowed from the Interactive and Monitor modes and the User Program. See the "P" and "PS" commands for format and examples.

Format: PAS<expr>{format} | "<Text>" ...

PASSWORD

Enter or change password which protects user program from being overwritten.

Format: PASSWORD

and follow instructions.

PLAY

Playback recorded points. This command prints all the variables that were recorded by the last RECORD command. Normally, you should use Motion Links' PLAYBACK,FROM BJx command rather than the BJx PLAY command. Motion Link formats, plots, and prints data in a much more readable form than does the BJx.

PS

Print with status. This is identical to the P command, except status of the BJx is displayed on the end of the printed line. See P for format and examples. Allowed from the interactive and monitor modes and the user program.

R

Refresh screen. This command is the same as the P command except that no line feed is printed. This command can be used to overprint, the practice of refreshing the display by printing a line with new values over the same line with old values. It is generally used for status updating. See P for examples and formats. Allowed from the interactive and monitor modes and the user program.

RD

Delay program execution for a specified period of time, but use external clock to time the delay. REG need not be on for RD to function properly. Allowed only from the user program.

Format: RD <Time>

Example:

```
RD 1000
```

RECORD

Record 1-4 variables for a specified period of time. This command allows you to record most BJx variables in real time for later playback. You cannot record PE, REMOTE, TMR1, TMR2, TMR3, TMR4, VAVG, VXAVG, or any user switches. Allowed from the user program or from the interactive mode.

Format: RECORD <Number> <Time> <1 to 4 Variables>

Where <Number> is the number of intervals over which the variables will be recorded,

and <Time> is the time in milliseconds of each interval.

Note: <Number> <= 1000 for 1 Variable
<Number> <= 500 for 2 Variables
<Number> <= 333 for 3 Variables
<Number> <= 250 for 4 Variables

Examples:

```
RECORD 1000 1 VFB                ;RECORD VFB ONCE/MSEC FOR
                                 ;1 SECOND
RECORD 500 10 VCMD VFB           ;RECORD VCMD AND VFB ONCE/10
                                 ;MSEC FOR 5.0 SECOND
RECORD 100 1000 VCMD VFB PCMD    ;RECORD VCMD, VFB, AND PCMD
                                 ;ONCE/SECOND FOR 100 SECONDS
```

RET

Return from a subroutine. Allowed only from the user program.

Format: RET

RS

Refresh screen with status. This command is identical to the R command, except status of the drive is displayed at the end of the printed line. See P for format and examples. Allowed from the interactive and monitor modes and the user program.

RUN

Run a program starting at the specified label. Allowed from the interactive mode. If no label is specified, run multi-tasking.

Format: RUN <Label>
RUN ;RUN MULTI-TASKING

Example:

```
RUN 4
RUN X1
RUN
```

S

Stop motion using a deceleration of AMAX. Allowed from the interactive and monitor modes and the user program.

Format: S

TIL

Continuously execute an optional instruction until condition is true. If no instruction is specified, then delay program execution until the condition is true. <Instruction> cannot be another TIL. Allowed only from the user program.

Format: TIL <Expr> <Logical> <Expr> [Instruction]

Example:

```
TIL PFB GT 100 P PFB
TIL X1*X2 NE X4/(X5+5) GOSUB 100
TIL VFB LT 100                    ;DELAY EXECUTION
```

TPLAY

Plays back to the serial port the Program Trace information that the Real Time Record command (TRECORD) captured to memory. Allowed from the Interactive and Monitor modes.

Format: TPLAY NEWEST <lines>
TPLAY OLDEST <lines>
TPLAY <line>
<line1><line2>
Where NEWEST will print starting from the newest trace data,
OLDEST will print starting from the oldest trace data,
<line> is a specific line number to playback,
and <line1><line2> is a range of line numbers to playback.

Example:

```
TPLAY                ;PLAYBACK ENTIRE TRACE
TPLAY NEWEST 10      ;PLAYBACK THE 10 NEWEST
TPLAY OLDEST 10      ;PLAYBACK THE 10 OLDEST
TPLAY 20             ;PLAYBACK LINE 20
TPLAY 5 15           ;PLAYBACK LINES 5 TO 15
```

TRECORD

Records to memory real-time program trace information. This command will not operate if extended User Registers are enabled (EXTDX=1). This command will also overwrite data captured by the PC-Scope command (RECORD). Allowed from the Interactive and Monitor modes.

Format: TRECORD START
TRECORD STOP
TRECORD CONTINUE

Where START will start trace recording,
STOP will stop trace recording,
and CONTINUE will continue trace recording.

TUNE

Tune the motor to a new load. This command is used if the motor needs to be re-tuned. The tuning parameters (KP, KV, KVI, and KPROP) determine the motor stability and response time. Often when the motor load is changed, tuning parameters need to be reset. The Tune command specifies Bandwidth and Stability. Higher bandwidth will produce faster response time. Higher stability will produce less overshoot, but noisier performance. Allowed from the interactive mode and the user program.

Format: TUNE <Bandwidth> <Stability>

Where Bandwidth is 5, 10, 15,...50 Hz and stability is 1, 2, or 3.

Example:

```
TUNE 25 2
```

W

Wait for a specified motion profile segment to start before continuing program execution. W is an idling command (that is, if you are using multi-tasking, W suspends the etask but lets other tasks proceed). Allowed only from the user program.

Format: W

Examples:

```
W 3     ;WAIT FOR SEGMENT 3 TO START
W 0     ;WAIT FOR MOTION TO STOP
```

ZPE

Clear the position error. This command is useful when enabling the position loop when position error has been allowed to accumulate. Allowed from the interactive and monitor modes and the user program.

<BDS

Send (download) a program from the BJx Program Memory to the terminal. Allowed from the interactive mode and the user program.

Format: <BDS

>BDS

Receive (upload) a program from the terminal and store it in BJx program memory. A password may be specified. If the editor password has been set and the password is incorrect or not specified, then an error will result and the original program memory will remain. Allowed form the interactive mode and the user program.

Format: >BDS {PASS}

where PASS is the password as set in the editor.

Example:

```
>BDS SECRET      ;UPLOAD, PASSWORD=SECRET
>BDS             ;UPLOAD, NO PASSWORD
```

Appendix D
Error Codes

INTRODUCTION

BJx response to an error depends on the error's severity. There are four levels of severity, listed below in increasing order:

1. Errors that cause warnings.
2. Errors that cause a program break and stop motion, in addition to Level 1 actions.
3. Errors that cause the system to disable, set the FAULT LED, and clear the OK LED output, in addition to Level 2 actions.
4. Errors that disable almost all BJx functions (including communications) and flash the FAULT LED to indicate the error number. These are called firmware errors.

HARDWARE FAULTS

Firmware Faults

ERROR 2 ***"HARDWARE- U-P FAIL"*** ***SEVERITY 4***

The microprocessor cannot pass self-test. This fault causes the microprocessor to blink the FAULT LED **twice** and then pause. The BJx will not communicate or run the user program. Contact the factory.

ERROR 3 ***"HARDWARE-CHECKSUM"*** ***SEVERITY 4***

The microprocessor cannot pass the checksum self-test. This fault causes the microprocessor to blink the FAULT LED **three times** and then pause. The BJx will not communicate or run the user program. Contact the factory.

ERROR 4 ***"SOFTWARE WATCHDOG"*** ***SEVERITY 4***

The microprocessor has failed the software watchdog self-test. This fault causes the microprocessor to blink the FAULT LED **four times** and then pause. The BJx will not communicate or run the user program. Contact the factory.

BJx Faults

ERROR 10 ***"REMOTE OFF"*** ***SEVERITY 2***

You attempted to execute an instruction that requires the hardware input REMOTE on the signal connector to be active. This error breaks program execution. You can disable REMOTE by turning on the switch NOREMOTE.

ERROR 13 ***"OVER-SPEED"*** ***SEVERITY 3***

The BJx determined that the speed of the motor was greater than the variable VOSPD. If this occurs occasionally, it may be a nuisance fault that should be corrected by raising VOSPD by 5% or 10%. This error breaks program execution and disables the BJx .

ERROR 14 ***"POWER BUS"*** ***SEVERITY 3***

The power supply high voltage bus has either an overvoltage fault or an undervoltage fault. This error breaks program execution and disables the BJx .

ERROR 19 ***"MOTION (HDWR LINE)"*** ***SEVERITY 2***

The MOTION input was off at the beginning of a motion instruction, or it turned off during a motion instruction. This error breaks program execution. You can disable MOTION with the command NOMOTION ON.

ERROR 20 ***"TUNE FAILED"*** ***SEVERITY 3***

The TUNE command failed. Either the inertia on the motor is too large for the desired bandwidth, the motor is not functioning properly, the bus voltage is too low, or the BJx is not functioning properly. Try reducing the desired bandwidth to correct this problem. Make sure REMOTE is on. If this does not work, attempt to tune the system by hand.

ERROR 21 ***"DRIVE FAULT"*** ***SEVERITY 3***

The amplifier attached to the BJx reported a FAULT. For a BJR, the amplifier is internal. For a BJP, the amplifier is customer supplied.

Positioner Faults

ERROR 23 ***"SOFTWARE OVERTRAVEL"*** ***SEVERITY 2***

Software travel limits are enabled and either PMAX or PMIN (the software limits) have been exceeded. If your application does not need software travel limits, or if you want to disable software travel limits temporarily, type:

PLIM OFF

This error breaks program execution.

ERROR 24 ***"HARDWARE OVERTRAVEL"*** ***SEVERITY 3***

The BJx detected an overtravel condition while it was enabled. You can print the state of the overtravel limit switch by typing:

```
P LIMIT
```

If LIMIT is 0, then an overtravel condition exists. LIMIT should be connected to a limit switch that has contacts that are normally closed but which open where an overtravel condition occurs. Hardware overtravel limits cannot be disabled. This error breaks program execution and disables the BJx .

ERROR 25 ***"PE OVERFLOW"*** ***SEVERITY 3***

The variable PE, the position error, exceeded the variable PEMAX. This is also called a *following error overflow*. This error breaks program execution and disables the BJx .

ERROR 26 ***"PFB ROLLOVER"*** ***SEVERITY 3***

The variable PFB, the position feedback, exceeded +/-2,147,483,647 counts. If you are using position units, then PFB exceeded the position unit equivalent of +/-2,147,483,647 counts. This can occur if the motor rotates indefinitely in one direction. If your application requires this, consider using the rotary mode.

ERROR 2 ***"CURRENT SATURATION TIME-OUT"*** ***SEVERITY 3***

The ICMD was equal to ILIM for greater than SATTIME milliseconds. This can occur if the motor is stalled or if the load exceeds the rating of the motor and drive. Set SATTIME to 0 to disable this feature.

MOTION ERRORS

Position Calculation Errors

ERROR 30 ***"TOO MANY MOVES"*** ***SEVERITY 2***

You typed in too many move commands (MA, MI) from the interactive mode. You can have one move executing and the other pending. The error does not occur when move commands are executed from the user program, because the BJx sees that the motion buffer is full and delays execution to prevent the error. This error breaks program execution.

ERROR 32 ***"ACC/DEC TOO LOW"*** ***SEVERITY 2***

You entered a motion command that calculated a motion profile where either the acceleration or deceleration segment was more than 30 seconds long. You must increase ACC or DEC or reduce the speed change of the move. This error breaks program execution.

ERROR 33 ***"VEL OUT OF BOUNDS"*** ***SEVERITY 2***

You entered a motion command where the commanded velocity was out of the allowable range. The range for Jog (J) commands is ±VMAX. The range for other motion commands is 0 to +VMAX. This error breaks program execution.

Macro Move/JT/JF Errors

ERROR 40 ***"CHANGED DIRECTION"*** ***SEVERITY 2***

You attempted to change direction with an instruction that does not allow direction to change. These instructions include JT and JF. This error breaks program execution.

ERROR 41 ***"MOVE NEEDS MOTION"*** ***SEVERITY 2***

You attempted to execute an instruction that requires the motor to be in motion. These instructions include JT and JF. This error breaks program execution.

ERROR 42 ***"MOVE w/o TIME"*** ***SEVERITY 2***

You attempted to execute a move that required more time than was available. For example, you attempted a JT where the final position could not be reached because of acceleration limits. You may have attempted a JT or JF when you were already well beyond the specified position. This error breaks program execution.

SOFTWARE ERRORS

Programming Modes or Motion Modes

ERROR 50 ***"DRIVE INHIBITED"*** ***SEVERITY 2***

You attempted to execute an instruction that required the BJx to be enabled while it was inhibited. This error will break program execution if the instruction is issued from the user program.

ERROR 51 ***"DRIVE ENABLED"*** ***SEVERITY 2***

You attempted to execute an instruction that required the BJx to be inhibited while it was enabled. This error will break program execution if the instruction is issued from the user program.

ERROR 52 ***"NOT FROM TERMINAL"*** ***SEVERITY 1***

You attempted to execute an instruction from the terminal that is not allowed from the terminal. This error generates no action.

ERROR 53 ***"NOT FROM PROGRAM"*** ***SEVERITY 1***

You attempted to execute an instruction from the program that is not allowed from the program. This error breaks program execution.

ERROR 54 ***"NOT FROM MONITOR"*** ***SEVERITY 1***

You attempted to execute an instruction while in the Monitor mode that is not allowed from the Monitor mode. This error generates no action.

ERROR 55 ***"NOT FROM RECOVERY"*** ***SEVERITY 2***

You attempted to execute an instruction from the error recovery (the user's error handler or "ERROR$") that is not allowed. This includes attempting to enable the BJx, GOSUB, and GOTO. This error breaks execution.

ERROR 56 ***"NOT w/GEAR"*** ***SEVERITY 2***

You attempted to execute an instruction when the Gear mode was enabled that is not allowed with the Gear mode. For example, MA, JT, and JF are not allowed with the Gear mode on. This error breaks execution if the instruction was issued from the program.

ERROR 57 ***"NOT w/PROFILE"*** ***SEVERITY 2***

You attempted to execute an instruction that is not allowed while the BJx is profiling. Profiling occurs when move instructions (MA, MI) are executing. Other examples of this are the traverse segment before the accel/decel portion of position dependent jogs (JT, JF), and the accel/decel portions of all jogs (J, JT, JF). This error breaks execution.

ERROR 58 ***"NOT w/JOGGING"*** ***SEVERITY 2***

You attempted to execute an instruction that is not allowed when the BJx is jogging. This error breaks execution if the instruction was issued from the program.

ERROR 59 ***"NOT w/ROTARY"*** ***SEVERITY 2***

You attempted to execute an instruction that is not allowed when the BJx is in the Rotary mode. Type:

```
ROTARY OFF
```

to turn the Rotary mode off. This error breaks execution if the instruction was issued from the program.

ERROR 60 ***"OUTSIDE PROTARY"*** ***SEVERITY 2***

You attempted to make an MA beyond PROTARY. For example, if PROTARY is 1000 and you typed:

```
MA 2000
```

Use incremental moves (MI) if you want to move beyond the rotary limit. This error breaks execution if the instruction was issued from the program.

ERROR 61 ***"NORMALIZE FIRST"*** ***SEVERITY 2***

You attempted to turn on the Rotary mode when PFB was less than zero or greater than PROTARY. Use the NORM command to normalize the position to between 0 and PROTARY. This error breaks execution if the instruction was issued from the program.

ERROR 62 ***"RD ALREADY IN USE"*** ***SEVERITY 2***

You attempted to execute RD when RD was in use from some other task. This error occurs when two task levels attempt to simultaneously use the RD command. This error breaks program execution.

ERROR 63 ***"NOT AT THIS LEVEL"*** ***SEVERITY 2***

You attempted to execute a command that is not allowed at the present task level. For example, GOSUB and GOTO are not allowed from within an alarm. This error breaks program execution.

ERROR 64 ***"BACKWARD REGULATION"*** ***SEVERITY 3***

The external input counted backwards more than 30,000 counts when REG was on. This error breaks program execution and disables the BJx.

ERROR 65 ***"RECORD NOT READY"*** ***SEVERITY 3***

You entered a PLAY command when nothing had been recorded since the last time the BJx powered up.

Improper Use of Labels

ERROR 70 ***"LABEL NOT FOUND"*** ***SEVERITY 2***

You attempted to branch to a label (either from RUN, GOSUB, or GOTO) that does not exist. This error breaks program execution.

ERROR 71 ***"LABEL USED TWICE"*** ***SEVERITY 2***

The user program has a label that is used more than once. This error breaks program execution.

ERROR 74 ***"ERROR$ MUST BE LAST"*** ***SEVERITY 2***

The user's error (ERROR$) must be the last label in the program buffer. You cannot have labels after ERROR$, nor can you use the GOTO or GOSUB commands when the BJx is executing the error handler. The error handler is intended to provide a graceful exit during error conditions and cannot be used to restart the program. You can use the IF, TIL, and ? commands to execute conditional commands in the error handler. This error breaks program execution.

Invalid Instructions or Entries

ERROR 79 ***"BAD FORMAT"*** ***SEVERITY 2***

You entered a format that the BJx does not recognize. For example, you may have entered:

```
INPUT "INPUT X1" X1[.3]
```

In this case, the decimal point (following the "[") is incorrect. Pay careful attention to the rules for formats in Chapter 5. This error breaks program execution if the instruction is issued from the user program.

ERROR 80 ***"INVALID INSTRUCTION"*** ***SEVERITY 2***

You attempted to execute an instruction or change a variable that the BJx does not recognize. This error breaks program execution if the instruction is issued from the user program.

ERROR 81 ***"NOT PROGRAMMABLE"*** ***SEVERITY 2***

You attempted to change a variable that is not programmable. This error will break program execution if the instruction is issued from the user program.

ERROR 82 ***"BAD NUMBER ENTRIES"*** ***SEVERITY 2***

The instruction that is executing has too many or too few parameters. Look up the instruction in Appendix C to determine the correct number of entries. This error breaks program execution if the instruction is issued from the user program.

ERROR 83 ***"BAD OR OUT OF RANGE"*** ***SEVERITY 2***

You entered a parameter to an instruction that was too large or too small. Check Appendix C for limits on variables. This error can also occur when a parameter is in the wrong format, such as a character string where a number is expected. This error breaks program execution if the instruction is issued from the user program.

ERROR 84 ***"OUT OF BOUNDS"*** ***SEVERITY 2***

The variable listed is out of bounds. This error breaks execution.

ERROR 85 ***"BAD INDIRECTION"*** ***SEVERITY 2***

You attempted an indirect reference to a user variable that does not exist. For example:

```
X1 10000
P X(X1)
```

X(X1) refers to user variable X10000, which does not exist. The "P X(X1)" will generate this error. This error breaks program execution if the instruction is issued from the user program.

ERROR 86 ***"USER PROGRAM FULL"*** ***SEVERITY 2***

You attempted to load a program larger than the BJx can hold. This occurs with the >BDS instruction and from the Motion Link communications software "Program Transmit (^T)." This error breaks program execution.

ERROR 87 ***"EMBEDDED QUOTE"*** ***SEVERITY 2***

You entered a command with an embedded quote. A space must precede an opening quote and follow a closing quote. For example:

```
P"BAD COMMAND"
```

has an embedded quote after the "P." This error breaks program execution if the instruction is issued from the user program.

ERROR 88 ***"NO CLOSING QUOTE"*** ***SEVERITY 2***

You entered a command with an odd (as opposed to even) number of quotes. This error breaks program execution if the instruction is issued from the user program.

ERROR 89 ***"NOT FOR ALARM/HOLD/RECORD"*** ***SEVERITY 2***

You have specified a switch that is not an allowable switch for an alarm or a hold or record command. For example:

```
A$ XS1 ON  ;ERROR—XS1 NOT ALLOWED FOR ALARMS
```

This line causes Error 89 since XS1 is not allowed to fire an alarm.

ERROR 90 ***"TOO MANY POINTS"*** ***SEVERITY 2***

You specified too many points in a RECORD command. Only 1000 points total can be recorded. For example, if you are recording four variables, they can be recorded no more than 250 times, since 4*250 = 1000.

Math Errors

ERROR 92 ***"ZERO DIVIDE"*** ***SEVERITY 2***

You attempted to divide a number by 0. This error breaks program execution if the instruction is issued from the user program.

ERROR 93 ***"MATH OVERFLOW"*** ***SEVERITY 2***

The final result of a calculation or an intermediate result during the calculation of an expression was greater than 2^{31} or less than -2^{31}. This error breaks program execution.

ERROR 94 ***">2 PARENTHESES"*** ***SEVERITY 2***

The BJx evaluated an expression with more levels of parentheses than the BJx supports. Up to two levels of parentheses are allowed. This error breaks program execution.

ERROR 95 ***"UNEVEN PARENTHESES"*** ***SEVERITY 2***

The BJx encountered an expression in which the number of closing parentheses was not equal to the number of opening parentheses. This error breaks program execution.

ERROR 96 ***"SCALING OVERFLOW"*** ***SEVERITY 2***

During a conversion to or from user units, the result was greater than 2^{31} or less than -2^{31}. This error breaks program execution if the instruction is issued from the user program.

ERROR 97 ***"GEAR OVERFLOW"*** ***SEVERITY 3***

The BJx encountered an overflow when calculating the velocity from the external pulse input. This can be caused when the variable GEARI is too small or GEARO is too large. That is, the input times the ratio of GEARO/GEARI was greater than the highest allowable input frequency, 2 MHz. This error breaks program execution and disables the BJx.

Communication Errors

ERROR 103 ***"BAUD RATE"*** ***SEVERITY 1***

The variable BAUD contains a value that is not supported by the BJx. This error occurs during the autobaud sequence and so is never printed to the terminal. You will only see it in the error history buffer. This error has no action.

ERROR 104 ***"ABAUD & MULTIDROP"*** ***SEVERITY 1***

This error is caused by attempting to autobaud while in multidrop communications, which is not allowed. The front-panel switch ABAUD is on, indicating request for autobaud, and the variable ADDR is not zero, indicating multidrop communications. This error occurs during the autobaud sequence and so is never printed to the terminal. You will only see it in the error history buffer. This error has no action.

ERROR 105 ***"SERIAL WDOG"*** ***SEVERITY 3***

The serial port did not receive a valid command for WTIME milliseconds when the serial watchdog was enabled (that is, WATCH = 1). This error breaks program execution and disables the BJx.

Password Errors

ERROR 110 ***"EDIT PASSWORD"*** ***SEVERITY 1***

You attempted to execute an instruction that requires a password. This occurs with the >BDS command. In this case, you must follow the command with the password.

Errors From IF, TIL and GOSUB Commands

ERROR 115 ***"IF w/o ENDIF"*** ***SEVERITY 2***

The program executed an IF command to begin an IF BLOCK but could not find the corresponding ENDIF to end the IF block. This error breaks program execution.

ERROR 116 ***"IF NOT STARTED"*** ***SEVERITY 2***

An ELSE, ELIF, or ENDIF was encountered when there was no IF. This will occur, among other times, if you use a GOTO to branch to the middle of an IF/ELIF/ELSE/ENDIF block. This error breaks program execution.

ERROR 117 ***"TIL FOLLOWS ?/TIL"*** ***SEVERITY 2***

The ? or TIL instruction was used to execute a conditional TIL. This error breaks program execution.

ERROR 118 ***"TOO MANY GOSUBS"*** ***SEVERITY 2***

The last GOSUB was one GOSUB too many. The BJx has 4 levels of subroutines. This error breaks program execution.

ERROR 119 ***"RETURN w/o GOSUB"*** ***SEVERITY 2***

The BJx encountered a RET when it was not expecting one. This occurs when there are more returns than GOSUBs. This error breaks program execution.

Power-Up Marker (Not An Error)

ERROR 199 ***"DRIVE POWERED UP"*** ***N/A***

This is not a true error. ERROR 199 is used to mark the error history buffer when the BJx powers-up.

Internal Errors

ERROR 202 ***"USER PROGRAM CORRUPT"*** ***SEVERITY 3***

The user program is corrupt. Usually, this problem is caused by installing a new battery back-up RAM. This can also occur if power to the BJx is lost while editing the program. This error will break program execution. (See BJx Editor New Command {where?} to reset the user program; you will need to reload your program.

ERROR 204 ***"PROGRAM OVERRUN"*** ***SEVERITY 3***

This is an internal error. Contact the factory. This error breaks program execution and disables the BJx.

ERROR 205 ***"MBUF OVRRUN"*** ***SEVERITY 3***

This is an internal error. Contact the factory. This error breaks program execution and disables the BJx.

ERROR 206 ***"PROFILE OVERFLOW"*** ***SEVERITY 3***

This is an internal error. Contact the factory. This error breaks program execution and disables the BJx.

ERROR 208 ***"GENERAL INTERNAL"*** ***SEVERITY 3***

This is an internal error. Carefully write down the entire line that is printed with the error and contact the factory. This error breaks program execution and disables the BJx.

ERROR 209 ***"STACK OVERFLOW"*** ***SEVERITY 3***

This is an internal error. Carefully write down the entire line that is printed with the error and contact the factory. This error breaks program execution and disables the BJx.

ERROR 211-219 ***"INTERNAL 1-9"*** ***SEVERITY 3***

These are internal errors. Contact the factory. These errors break program execution and disable the BJx.

ERROR 255 ***"UNKNOWN"*** ***SEVERITY 3***

This is an internal error. If this error exists in the error history upon initial power-up, clear it with ERR CLR. Contact the factory if this error occurs during operation. This error breaks program execution and disables the BJx.

Appendix E
Variable Quick Reference Guide

INTRODUCTION

This appendix lists all the variables on the BJx. All variables are shown with the required programming conditions. For example, ACC has the programming condition "ALWAYS." This means ACC can be changed at any time. Other variables require the BJx to be enabled or disabled. Others, such as feedback variables, are never programmable.

Variables

VARIABLE	DESCRIPTION	PROGRAM CONDITION	UNITS	PROGRAM LIMITS[1]
A1N1-A1N3	Analog Input	Never	None	0-1023
ABAUD	Front Panel Switch	Never	None	0,1
ACC	Acceleration Rate	Always	ACC	0-AMAX
ACTIVE	Monitor Drive	Never	None	
ADDR	Multidrop Address	Always		0,48-57,65-90
ADEN	ACC Units Denominator	Always	None	Long
AMAX	ACC/DEC Maximum	Disabled	ACC	Long>0
ANUM	ACC Units Numerator	Always	None	long
AOUT	Analog Output	Always	None	-128 to 127
BAUD	Baud Rate	Always	None	300-19200
CAM	Monitor CAM Status	None	None	0,1

[1]See Table at end of Appendix E for description of long and short.

CAP	Enable Capture	Always	None	0,1
CAPDIR	Polarity of Capture	Always	None	0,1
CAPSRC	Capture trigger selection	Always	None	1,2,3
CLAMP	Enable Clamp Mode	Always	None	0,1
CYCLE	Start Cycle	Never	None	
DEC	Deceleration Rate	Always	ACC	0-AMAX
DEP	Shorten Error Msgs	Always	None	0,1
DIR	On if CW is Positive	Always	None	0,1
DRVSTAT	Current Status of Drive	Never	None	0,1
ECHO	Enable Serial Port Echo	Always	None	0,1
ENCDIR	Direction of Feedback Encoder	Disabled	None	0,1
EXT DX	Enable Extended Registers	Always	None	0,1
FAULT	On for BJx Fault	Always	None	0,1
GATE	Monitor GATE Input	Never	None	
GATEMODE	Enable Gate Mode	Always	None	0,1
GEAR	Enable Gear Mode	Always	None	0,1
GEARI	Input Gear Teeth	Always	None	Short
GEARO	Output Gear Teeth	Always	None	Short>0
HOME	Monitor HOME Input	Never	None	
ICMD	Commanded Current	Never	I	
IDEN	I Units Denominator	Always	None	Long
ILIM	Set Current Limit	Always	I	1-IMAX
I1-I8	Monitor Input Lines	Never	None	
IN	Input Word	Never	None	
INDEX	Index Pulse (Feedback Encoder)	Never	None	0, 1
INUM	I Units Numerator	Always	None	Long
KF	Feed-Forward Gain	Always	None	Short>0
KP	Pos Loop Gain	Always	None	Short>0
KPROP	Prop. Vel Loop Gain	Always	None	Short>0
KV	Integrating Vel Loop Gain	Always	None	Short>0
KVI	Integrating Vel Loop Gain	Always	None	Short>0
LIMIT	Monitor LIMIT Input	Never	None	

LPF	Enable Low Pass Filter	Always	None	0,1
LPFHZ	Low Pass Filter Freq	Always	Hz	0-500
LSTERR	Last error	Never	None	
LSTLBL	Last label executed	Always	None	
LSTTUNE	Tuning Assistance	Never	None	Long
MANUAL	Same at I1	Never	None	
MENCDIR	Master Encoder Direction	Always	None	0,1
MENCODER	Master Encoder Mode	Always	None	1-4
MINDEX	Index Pulse (Master Encoder)	Never	None	0, 1
MONITOR	Force BJx into Monitor Mode	Always	None	0,1
MOTION	Monitor MOTION Input	Never	None	
MSG	Enable Serial Port Messages	Always	None	0,1
MSTRMODE	Select Master Input Mode	Always	None	1-4
MULTI	Enable Multi-tasking	Always	None	0,1
N	Special Constant=0	Never	None	
NOLIMIT	Disable LIMIT	Always	None	0,1
NOMOTION	Disable MOTION	Always	None	0,1
NOREMOTE	Disable REMOTE	Always	None	0,1
O1-O5	Set/Monitor Output Lines	Always	None	0,1
OFF	Special Constant=0	Never	None	
OK	OK Latch	Always	None	0,1
OK2EN	OK to enable BJx	Never	None	
ON	Special Constant=1	Never	None	
OUT	Set/Monitor Output Word	Always	None	0-255
PCAM	Position Command from CAM Table	Never	POS	Long
PCAP	Capture Position	Never	POS	
PCMD	Position Command	Never	POS	
PDEN	POS Units Denominator	Always	None	Long
PE	Position Error	Never	POS	
PECLAMP	Clamp Position Error	Always	POS	Short>0
PEMAX	Maximum Position Error	Always	POS	Short>0

PEXT	External Position	Always	XPOS	Long
PEXTCAP	Capture External Position	Never	XPOS	Long
PFB	Position Feedback	No Motion	POS	Long
PFNL	Final Position	Never	POS	
PL	Enable Position Loop	Always	None	0,1
PLIM	Enable Soft Limits	Always	None	0,1
PMAX	Soft Upper Limit	Always	POS	Long
PMIN	Soft Lower Limit	Always	POS	Long
PROMPT	Enable Prompts	Always	None	0,1
PROTARY	Rotary Distance	Always	POS	Long
PNUM	POS Units Numerator	Always	None	Long
PROP	Enable Prop. Mode	Always	None	0,1
PTRIP1	Position Trip Point #1	Always	POS	Long
PTRIP2	Position Trip Point #2	Always	POS	Long
PXDEN	Extern. Pos Denominator	Always	None	Long
PXNUM	Extern. Pos Numerator	Always	None	Long
RAMP	Ramp control with gear	Always	None	0,1
READY	Enable Drive	Never	None	
REG	Enable Profile Regulation	Always	None	0,1
REGKHZ	Max Regulation Freq.	Always	kHz	1-2000
REMOTE	Monitor REMOTE Input	Never	None	
ROTARY	Enable Rotary Mode	Always	None	0,1
SAT	Monitor Saturation	Never	None	
SATTIME	Saturation Time	Always	Msec	Short>0
SCKSUM	Enable Serial Checksum	Always	None	0,1
SCRV	S-curve Type	Always	None	1-3
SEG	Motion Segment	Never	None	
SERIAL	Monitor Serial Port	Never	None	
SS	Enable Single Step	Always	None	0,1
STATSEN	Drive Status Sense (Normally 1)	Always	None	0,1
TMR1	Standard Timer	Always	Msec	Long>0
TMR2	Standard Timer	Always	Msec	Long>0

TMR3	Standard Timer	Always	Msec	Long>0
TMR4	Standard Timer	Always	Msec	Long>0
TRC	Enable Trace	Always	None	0,1
TRIP	Enable Trip Points	Always	None	0,1
TRIP1	Trip #1 Indicator	Never	None	0,1
TRIP2	Trip #2 Indicator	Never	None	0,1
TQ	Enable Torque Loop	Always	None	0,1
VAVG	Averaged VFB	Never	VEL	
VCMD	Velocity Command	Never	VEL	
VDEN	VEL Units Denominator	Always	None	Long
VE	Velocity Error	Never	VEL	
VER	Software Version Number	Never	None	Index entries
VEXT	External Velocity	Never	XVEL	
VFB	Velocity Feedback	Never	VEL	
VMAX	Maximum Velocity of BJx	Never	VEL	Long
VNUM	VEL Units Numerator	Always	None	Long
VOFF	Gearbox Velocity Offset	Always	VEL	Long
VOSPD	Overspeed Setpoint	Disabled	VEL	Long
VXAVG	Averaged VEXT	Never	XVEL	
VXDEN	External Vel Denominator	Always	None	Long
VXNUM	External Vel Numerator	Always	None	Long
WATCH	Enable Serial Watchdog	Always	None	0,1
WTIME	Serial Watchdog Timeout	Always	Msec	Short>0
X1-X250	User Variables	Always	None	Long
XS1-XS50	User Switches	Always	None	0,1
X(X1-X250)	User Indirect Vars	Always	None	Long
Y	Special Constant=1	Never	None	

Description of Program Limits

Long Limit	-2147483648	<	x	<	2147483647
Long>0 Limit	0	<	x	<	2147483647
Short Limit	-32768	<	x	<	32767
Short>0 Limit	0	<	x	<	32767

APPENDIX F

Customer Support

Kollmorgen is committed to quality customer service. Our goal is to provide the customer with information and resources as soon as they are needed. In order to serve in the most effective way, Kollmorgen offers a one-stop service center to answer all our customer's product needs. This one number provides order status and delivery information, product information and literature, and application and field technical assistance:

Kollmorgen Customer Support Network
203 Rock Road Suite A
Radford, VA 24141
Phone: (888) 774-KCSN (5276)
Fax: (540) 639-1640 Inside Sales
Fax: (540) 639-1574 Technical Support
Email: servo@Kollmorgen.com
Http://www.Kollmorgen.com

Note! If you are unaware of your local sales representative, please contact us at the number above. Visit our web site for MotionLink® software upgrades, technical articles, and the most resent version of our product manuals

Appendix G
Initial Settings

This appendix provides the recommended initial settings of BJx programmable variables:

```
;BJx { v02.0.5 09-14-95 14:34:08}
;
;
;
;UNITS:
;
PNUM    = 1
PDEN    = 1
VNUM    = 43690
VDEN    = 10
ANUM    = 43690
ADEN    = 10000
INUM    = 4095
IDEN    = 100
PXNUM   = 1
PXDEN   = 1
VXNUM   = 43690
VXDEN   = 10

;PROTECTED VARIABLES:
IMAX    = 100
VMAX    = 5000
;
;UNPROTECTED VARIABLES:
;
ACC     = 100000
ADDR    = 0
ADEN    = 10000
AMAX    = 1000000
ANUM    = 43690
AOUT    = 0
BAUD    = 9600
CAP     = 0
CAPDIR  = 1
CAPSRC  = 1
CLAMP   = 0
DEC     = 100000
DEP     = 0
DIR     = 1
ECHO    = 1
ENCDIR  = 1
EXTDX   = 0
FAULT   = 0
GATEMODE = 0
GEAR    = 0
GEARI   = 1
GEARO   = 1
HSTACH  = 0
IDEN    = 100
ILIM    = 100
INUM    = 4095
KF      = 2000
KP      = 1000
```

KPROP = 480
KV = 2500
KVI = 5536
LPF = 1
LPFHZ = 200
MENCDIR = 1
MONITOR = 0
MSG = 1
MSTRMODE = 1
MULTI = 1
NOLIMIT = 0
NOMOTION = 0
NOREMOTE = 0
O1 = 0
O2 = 0
O3 = 0
O4 = 0
O5 = 0
OK = 1
OUT = 0
PDEN = 1
PECLAMP = 100
PEMAX = 32767
PEXT = 0
PFB = 0
PL = 1
PLIM = 1
PMAX = 100000000
PMIN = 0
PNUM = 1
PROMPT = 1
PROP = 0
PROTARY = 4096
PTRIP1 = 0
PTRIP2 = 0
PXDEN = 1
PXNUM = 1
RAMP = 0
REG = 0
REGKHZ = 1000
ROTARY = 0
SATTIME = 2000
SCRV = 1
SS = 0
STATSEN = 1
TMR1 = 0
TMR2 = 0
TMR3 = 0
TMR4 = 0
TQ = 0
TRC = 0
TRIP = 1
VDEN = 10
VNUM = 43690
VOFF = 0
VOSPD = 6000
VXDEN = 10
VXNUM = 43690
WATCH = 0
WTIME = 1000

GLOSSARY

Absolute Position

Position referenced to a fixed zero position.

Absolute Positioning

Refers to a motion control system employing position feedback devices (absolute encoders) to maintain a given mechanical location.

Absolute Programming

A positioning coordinate reference wherein all positions are specified relative to some reference, or "home" position. This is different from incremental programming, where distances are specified relative to the current position.

AC Adjustable-Speed Drive

All equipment required to adjust the speed or torque of an AC electric motor by controlling both frequency and voltage applied to the motor.

AC Servo Drive

A servo drive used to control either or both synchronous or induction AC motors.

Acceleration

The change in velocity as a function of time, usually referring to an increase in velocity.

Accuracy

A measure of the difference between expected position and actual position of a motor or mechanical system. Motor accuracy is usually specified as an angle representing the maximum deviation from expected position.

Actuator

A device that creates mechanical motion by converting various forms of energy to mechanical energy.

Adaptive Control

A technique to allow the control to automatically compensate for changes in system parameters such as load variations.

Ambient Temperature

The temperature of the cooling medium, usually air, immediately surrounding the motor or another device.

Amplifier

Electronics that convert low level command signals to high power voltages and currents to operate a servomotor.

ASCII

American Standard Code for Information Interchange. This code assigns a number to each numeral, letter, or character on the keyboard. In this manner, information can be transmitted between machines as a series of binary numbers.

Back EMF

The voltage generated when a permanent magnet motor is rotated. This voltage is proportional to motor speed and is present regardless of whether the motor windings are energized or un-energized.

Bandwidth

The frequency range in which the magnitude of the system gain expressed in dB is greater than -3 dB.

Baud Rate

The number of binary bits transmitted per second on a serial communications link (such as RS-232).

Bit (Binary Digit)

A unit of information equal to 1 binary decision or having only a value 0 or 1.

Block Diagram

A simplified schematic representing components and signal flow through a system.

Bode Plot

A plot of the magnitude of system gain in dB and the phase of system gain in degrees versus the sinusoidal input signal frequency in logarithmic scale.

Brownout

Low-line voltage at which the device no longer functions properly.

Brush

Conducting material that passes current from the DC motor terminals to the rotating commutator.

Brushless Servo Drive

A servo drive used to control a permanent magnet synchronous AC motor. May also be referred to as an AC Servo Drive.

Bus

A group of parallel connections carrying pre-assigned digital signals. Buses usually consist of address and data information and miscellaneous control signals for the interconnection of microprocessors, memories, and other computing elements.

Byte

A group of 8 bits treated as a whole with 256 possible combinations of ones and zeros, each combination representing a unique piece of information.

CAM Profile

A technique used to perform nonlinear motion electronically, similar to that achieved with mechanical cams.

Characteristic Equation

1+GH = 0, where G is the transfer function of the forward signal path and H is the transfer function of the feedback signal path.

Circular Coordinated Move

A coordinated move where the path between endpoints is the arc of a circle.

Class B Insulation

A NEMA insulation specification. Class B insulation is rated to an operating temperature of 130 degrees centigrade.

Class H Insulation

A NEMA insulation specification. Class H insulation is rated to an operating temperature of 180 degrees centigrade.

Closed Loop

A broadly applied term relating to any system where the output is measured and compared to the input. The output is then adjusted to reach the desired condition. In motion control, the term is used to describe a system wherein a velocity or position (or both) transducer is used to generate correction signals by comparison to desired parameters.

Cogging

A term used to describe non-uniform, angular velocity. Cogging appears as a jerkiness, especially at low speeds.

Command Position

The desired angular or linear position of an actuator.

Commutation

Refers to the action of directing currents or voltage to the proper motor phases so as to produce optimum motor torque. In brush type motors, commutation is done electromechanically via the brushes and commutator. In brushless motors, commutation is done by the switching electronics using rotor position information typically obtained by hall sensors, a tachsyn, a resolver, or an encoder.

Commutator

A mechanical cylinder consisting of alternating segments of conductive and insulating material. This cylinder used in DC motors passes currents from the brushes into the rotor windings and performs motor commutation as the motor rotates.

Compensation

The corrective or control action in a feedback loop system which is used to improve system performance characteristics such as accuracy and response time.

Compensation, Feedforward

A control action that depends on the command only and not the error to improve system response time.

Compensation, Integral

A control action that is proportional to the integral or accumulative time error value product of the feedback loop error signal. It is usually used to reduce static error.

Compensation, Lag

A control action that causes the lag at low frequencies and tends to increase the delay between the input and output of a system while decreasing static error.

Compensation, Lead

A control action that causes the phase to lead at high frequencies and tends to decrease the delay between the input and output of a system.

Compensation, Lead Lag

A control action that combines the characteristics of lead and lag compensations.

Compensation, Proportional

A control action that is directly proportional to the error signal of a feedback loop. It is used to improve system accuracy and response time.

Compliance

The amount of displacement per unit of applied force.

Computer Numerical Control (CNC)

A computer-based motion control device programmable in a numerical word address format. A CNC product typically includes a CPU section, operator interface devices, input/output signal and data devices, software and related peripheral apparatus.

Control Systems or Automatic Control Systems

An engineering or scientific field that deals with controlling or determining the performance of dynamic systems such as servo systems.

Coordinated Motion

Multi-axis motion where the position of each axis is dependent on the other axis such that the path and velocity of a move can be accurately controlled. (Requires coordination between axes.)

Coupling Ratio

The ratio of motor velocity to load velocity for a load coupled to motor through a gear or similar mechanical device.

Critical Damping

A system is critically damped when the response to a step change in desired velocity or position is achieved in the minimum possible time with little or no overshoot.

Daisy Chain

A term used to describe the linking of several RS232C devices in sequence such that a single data stream flows through one device and on to the next. Daisy-chained devices usually are distinguished by device addresses which serve to indicate the desired destination for data in the stream.

Damping

An indication of the rate of decay of a signal to its steady state value. Related to setting time.

Damping Ratio

Ratio of actual damping to critical damping. Less than one is an underdamped system and greater than one is an overdamped system.

DC Adjustable-Speed Drive

All equipment required to adjust the speed or torque of a DC motor by controlling the voltages applied to the armature and/or field of the motor.

DC Drive

An electronic control unit for running a DC motor. The DC drive converts AC line current to a variable DC current to control a DC motor. The DC drive has a signal input that controls the torque and speed of the motor.

Dead Band

A range of input signals for which there is no system response.

Deceleration

A change in velocity as a function of time, referring to a decrease in velocity.

Decibel (dB)

A logarithmic measurement of gain. If G is a systems gain (ratio of output to input), then 20 log G = gain in decibels (dB).

Demag Current

The current level at which the motor magnets will be demagnetized. This is an irreversible effect which will alter the motor characteristics and degrade performance.

Detent Torque

The maximum torque that can be applied to an un-energized stepper motor without causing continuous rotating motion.

Dielectric Test

A high voltage breakdown test of insulation's ability to withstand an AC voltage. Test criterion limits the leakage current to a specified magnitude and frequency applied between specified test points.

Differential

An electrical input or output signal that uses two lines of opposite polarity referenced to the local signal ground.

Direct Numerical Control, DNC

Technique of transferring part program data to a numerical control system via direct electrical connection in place of paper tapes.

Distributed Processing

A technique to gain increased performance and modularity in control systems utilizing multiple computers or processors.

Drive

This is the electronics portion of the system that controls power to the motor.

Drive, Analog

Usually referring to any type of motor drive in which the input is an analog signal.

Drive, Digital

A motor drive in which the tuning or compensation is done digitally. Input may be an analog or digital signal.

Drive, Linear

A motor drive in which the output is directly proportional to either a voltage or current input. Normally, both inputs and outputs are analog signals. This is a relatively inefficient drive type.

Drive, PWM

A motor drive that uses Pulse-Width Modulation techniques to control power to the motor. Typically a high efficiency drive that can be used for high response application.

Drive, SCR

A DC motor drive that utilizes internal silicon controlled rectifiers as the power control elements. Usually used for low bandwidths, high power applications.

Drive, Servo

A motor drive that utilizes internal feedback loops for accurate control of motor current and/or velocity.

Drive, Stepper

Electronic controller that converts step and direction inputs to high power currents and voltages to drive a stepper motor. The stepprt motor driver is analogous to the servo motor amplifier.

Duty Cycle

For a repetitive cycle, the ratio of on time to total cycle time.

$$\text{Duty Cycle} = \frac{(\text{On Time})}{(\text{On Time} + \text{Off Time})} \times 100\%$$

Dynamic Braking

A passive technique for stopping a permanent magnet brush or brushless motor. The motor windings are shorted together through a resistor resulting in motor braking with an exponential decrease in speed.

Efficiency

The ratio of power output to power input.

Electrical Time Constant

The ratio of armature inductance to armature resistance.

Electronic Gearing

A technique used to electrically simulate mechanical gearing. Causes one closed loop axis to be slaved to another open or closed loop axis with a variable ratio.

EMI: Electro-Magnetic Interference

Noise produced by one device which can degrade operation of other electronic circuits.

Encoder

A type of feedback device that converts mechanical motion into electrical signals to indicate actuator position. Typical encoders are designed with a printed disc and a light source. As the disc turns with the actuator shaft, the light source shines through the printed pattern onto a sensor. The light transmission is interrupted by the patterns on the disc. These interruptions are sensed and converted to electrical pulses. By counting these pulses, actuator shaft position is determined.

Encoder, Absolute

A digital position transducer in which the output is representative of the absolute position of the input shaft within one (or more) revolutions. Output is usually a parallel digital word.

Encoder, Incremental

A position encoding device in which the output represents incremental changes in position.

Encoder, Linear

A digital position transducer that directly measures linear position.

Encoder Marker

An ounce-per-revolution signal provided by some incremental encoders to specify a reference point within that revolution. Also known as Zero Reference signal or index pulse.

Encoder Resolution

A measure of the smallest positional change that can be detected by the encoder.

Explosion-proof

A motor classification that indicates a motor is capable of withstanding internal explosions without bursting or allowing ignition to reach beyond the confines of the motor frame.

Fall Time

The time for the amplitude of system response to decay to 37% of its steady-state value after the removal of a steady-state step input signal.

Feed Forward

A technique used to pre-compensate control a loop for known errors due to motor, drive, or lead characteristics. Provides improved response.

Feedback

A signal that is transferred from the output back to the input for use in a closed loop system.

Field Weakening

A method of increasing the speed of a wound field DC motor; reducing stator magnetic field instantly by reducing magnet winding current.

Filter (Control Systems)

A transfer function used to modify the frequency or time response of a control system.

Flutter

Flutter is an error of the basic cycle of an encoder per one revolution.

Following Error

The positional error during motion resulting from use of a position control loop with proportional gain only.

Form Factor

The ratio of RMS current to average current. This number is a measure of the current ripple in a PWM or other switch mode type of controller. Since motor heating is a function of RMS current while motor torque is a function of average current, a form factor greater than 1.00 means some fraction of motor current is producing heat but not torque.

Four Quadrant

Refers to a motion system that can operate in all four quadrants i.e. velocity in either direction and torque in either direction. This means that the motor can accelerate, run, and decelerate in either direction.

Friction

A resistance to motion caused by surfaces rubbing together. Friction can be constant with varying speed (coulomb friction) or proportional to speed (viscous friction) or present at rest (static friction).

Full Load Current

The armature current of a motor operated at its full load torque and speed with rated voltage applied.

Full Load Speed

The speed of a motor operated with rated voltage and full load torque.

Gain

The ratio of system output signal to system input signal.

Hall Sensors

A feedback device used in a brushless servo system to provide information for the amplifier to electronically commutate the motor. The device uses a magnetized wheel and hall-effect sensors to generate the commutation signals.

Holding Torque

Sometimes called torque, it specifies the maximum external force or torque that can be applied to a stopped, energized motor without causing the rotor to rotate continuously.

Home Position

A reference position for all absolute positioning movements. Usually defined by a home limit switch and/or encoder marker. Normally set at power-up and retained for as long as the control system is operational.

Host Computer

An auxiliary computer system connected to a controller or controllers. The host computer in distributed control systems is frequently involved with controlling many remote and distributed motion control devices. It may also be used for off-line tasks such as program preparation, storage, and supervisory control and evaluation.

HP: Horsepower

One horsepower is equal to 746 watts. Since Power = Torque x Speed, horsepower is a measure of a motor's torque and speed capability (e.g. a 1 HP motor will produce 35 lb.-in. at 1800 rpm).

Hunting

The oscillation of the system response about a theoretical steady-state value.

Hybrid Stepper Motor

A motor designed to move in discrete increments or steps. The motor has a permanent magnet rotor and wound stator. These motors are brushless, and phase currents are commutated as a function of time to produce motion.

Hysteresis

The difference in response of a system to an increasing or decreasing input signal.

I/O: Input/Output

The reception and transmission of information between control devices. In modern control systems, I/O has two distinct forms: switches, relays, etc., in either an on or off state, or analog signals that are continuous in nature, such as speed, temperature, flow, etc.

Idle Current Reduction

A stepper motor driver feature that reduces the phase current to the motor when no motor motion (idle) is commanded for a specified period of time. This reduces motor heating and allows high machine throughput to be obtained from a given motor.

Incremental Motion

A motion control term that is used to describe a device that produces one step of motion for each step command (usually a pulse) received.

Indexer

Electronics that convert high-level motion commands from a host computer, programmable controller, or operator panel into step direction pulse streams for use by a stepper motor driver.

Inertia

The property of an object to resist changes in velocity unless acted upon by an outside force. Higher inertia objects require larger torques to accelerate and decelerate. Inertia is dependent upon the mass and shape of the object.

Inertial Match

An inertial match between motor and load is obtained by selecting the coupling ratio such that the load moment of inertia referred to the motor shaft is equal to the motor moment of inertia.

Inrush Current

The current surge generated when a piece of equipment, such as a servoamplifier, is connected to an AC line. This surge is typically due to the impulse charging of a large capacitor located in the equipment.

Instability

Undesirable motion of an actuator that is different from the command motion. Instability can take the form of irregular speed or hunting of the final rest position.

Lead Ball Screw

A lead screw that has its threads formed as a ball-bearing race; the carriage contains a circulating supply of balls for increased efficiency.

Lead Screw

A device for translating rotary motion into linear motion, consisting of an externally threaded screw and an internally threaded carriage (nut).

Least Significant Bit

The bit in a binary number that is the least important, or having the least weight.

Limits

Properly designed motion control systems have sensors called limits which alert the control electronics that the physical end of travel is being approached and that motion should stop.

Linear Coordinated Move

A coordinated move where the path between endpoints is a line.

Linearity

For a speed control system, linearity is the maximum deviation between actual and set speed expressed as a percentage of set speed.

Logic Ground

An electrical potential to which all control signals in a particular system are referenced.

Loop, Feedback Control

A control method that compares the input from a measurement device, such as an encoder or tachometer, to a desired parameter, such as a position or velocity and causes action to correct any detected error. Several types of loops can be used in combination (i.e. velocity and position together) for high performance requirements.

Loop Gain, Open

The product of the forward path and feedback path gains.

Loop, PID: Proportional, Integral, and Derivative Loop

Specialized very high performance control loop that gives superior response.

Loop, Position

A feedback control loop in which the controlled parameter is motor position.

Loop, Velocity

A feedback control loop in which the controlled parameter is mechanical velocity.

Master Slave Motion Control

A type of coordinated motion control where the master axis position is used to generate one or more slave axis position commands.

Mechanical Time Constant

The time for an unloaded motor to reach 63.2% of its final velocity after the application of a DC armature voltage.

Microstepping

An electronic control technique that proportions the current in a stepper motor's windings to provide additional intermediate positions between poles. Produces smooth rotation over a wide speed range and high positional resolution.

Mid-Range Instability

A phenomenon in which a stepping motor can fall out of synchronism due to loss of torque at mid-range speeds. The loss of torque is due to interaction between the motor's electrical characteristics and the driver electronics. Some drivers have circuitry to eliminate or reduce this phenomenon.

Most Significant Bit

The bit in a binary number that is the most important or that has the most weight.

Motor, AC

A device that converts electrical alternating current into mechanical energy. Requires no commutation devices such as brushes. Normally operated off commercial AC power. Can be single- or multiple-phase.

Motor, AC Asynchronous or Induction

An AC motor in which speed is proportional to the frequency of the applied AC. Requires no magnets or field coil. Usually used for non-precise constant speed applications.

Motor, AC Synchronous

Another term for brushless DC motor.

Motor Constant

The ratio of the motor torque to motor input power.

Motor, DC

A device that converts electrical direct current into mechanical energy. It requires a commutating device, either brushes or electronic. Usually requires a source of DC power.

Motor, DC Brushless

A type of direct current motor that utilizes electronic commutation rather than brushes to transfer current.

Motor, DC Permanent Magnet

A motor utilizing permanent magnets to produce a magnetic field. Has linear torque speed characteristics.

Motor, DC Wound Field

A direct current utilizing a coil to produce a magnetic field. Usually used in high power applications where constant horsepower operation is desired.

Motor, Stepping

A specialized AC motor that allows discrete positioning without feedback. Normally used for non-critical, low power applications, since positional information is easily lost if acceleration or velocity limits are exceeded. Load variations can also cause loss of position. If encoders are used, these limitations can be overcome.

NC, Numerical Control

Usually refers to any type of automated equipment or process used for contouring or positioning.

Negative Feedback

The type of feedbacks used in a closed loop system where the output value is inverted and combined with the input to be used to stabilize or improve system characteristics.

No Load Speed

Motor speed with no external load.

Open Collector

A term used to describe a signal output that is performed with a transistor. An open collector output acts like a switch closure with one end of the switch at ground potential and the other end of the switch accessible.

Open-Loop System

A system where the command signal results in actuator movement but, because the movement is not sensed, there is no way to correct for error. Open loop means no feedback.

Operator Interface

A device that allows the operator to communicate with a machine. This device typically has a keyboard or thumbwheel to enter instructions into the machine. It also has a display device that allows the machine to display messages.

Optically Isolated

A system or circuit that transmits signals with no direct electrical connection. Used to protectively isolate electrically noisy machine signals from low level control logic.

Oscillation

An effect that varies periodically between two values.

Overshoot

The amount of the parameter being controlled exceeds the desired value for a step input.

Phase-Locked Servo System

A hybrid control system in which the output of an optical tachometer is compared to a reference square wave signal to generate a system error signal proportional to both shaft velocity and position errors.

Phase Margin

The difference between 180 degrees and the phase angle of a system at the frequency where the open loop gain is unity.

PID

Proportional-Integral-Derivative. An acronym that describes the compensation structure that can be used in a closed-loop system.

PLC

Programmable Logic Controller. An industrial control device that turns on and off outputs based upon responses to inputs.

PMDC Motor

A motor consisting of a permanent magnet stator and a wound iron-core rotor. These are brush type motors and are operated by application of DC current.

Point to Point Move

A multi-axis move from one point to another where each axis is controlled independently. (No coordination between axes is required.)

Pole

A frequency at which the transfer function of a system goes to infinity.

Pole Pair, Electromechanical

The number of cycles of magnetic flux distribution in the air gap of a rotary electromechanical device.

Position Error

The difference between the present actuator (feedback) value and the desired position command for a position loop.

Position Feedback

Present actuator position as measured by a position transducer.

Power

The rate at which work is done. In motion control, Power = Torque x Speed.

Process Control

A term used to describe the control of machine or manufacturing processes, especially in continuous production environments.

Pull-In Torque

The maximum torque at which an energized stepping motor or synchronous motor will start and run in synchronism.

Pull-Out Torque

The maximum torque that can be applied to a stepping motor or synchronous motor running at constant speed without causing a loss of synchronism.

Pulse Rate

The frequency of the step pulses applied to a stepper motor driver. The pulse rate divided by the resolution of the motor/drive combination (in steps per revolution) yields the rotational speed in revolutions per second.

PWM

Pulse Width Modulation. An acronym that describes a switch-mode control technique used in amplifiers and drivers to control motor voltage and current. This control technique is used in contrast to linear control and offers the advantages of greatly improved efficiency.

Quadrature

Refers to signal characteristics of two-channel encoders. Using quadrature, 4 counts are given for each line of encoder resolution. For example, a 1000-line encoder provides 4000 counts per revolution.

Ramping

The acceleration and deceleration of a motor. May also refer to the change in frequency of the applied step pulse train.

Rated Torque

The torque producing capacity of a motor at a given speed. This is the maximum continuous torque the motor can deliver to a load and is usually specified with a torque/speed curve.

Regeneration

The action during motor braking, in which the motor acts as a generator and takes kinetic energy from the load, converts it to electrical energy, and returns it to the amplifier.

Repeatability

The degree to which the positioning accuracy for a given move performed repetitively can be duplicated.

Resolution

The smallest positioning increment that can be achieved. Frequently defined as the number of steps or feedback units required for a motor's shaft to rotate one complete revolution.

Resolver

A position transducer utilizing magnetic coupling to measure absolute shaft position over one revolution.

Resonance

The effect of a periodic driving force that causes large amplitude increases at a particular frequency. (Resonance frequency.)

RFI

Radio Frequency Interference.

Ringing

Oscillation of a system following sudden change in state.

Rise Time

The time required for a signal to rise from 10% of its final value to 90% of its final value.

RMS Current

Root mean square current. In an intermittent duty cycle application, the RMS current is equal to the value of steady state current which would produce the equivalent resistive heating over a long period of time.

RMS Torque

Root Mean Square Torque. For an intermittent duty cycle application, the RMS torque is equal to the steady state torque which would produce the same amount of motor heating over long periods of time.

Robot

A reprogrammable multifunctional manipulator designed to move material, parts, tools, or specialized devices through variable programmed motions for the performance of a variety of tasks.

Robot Control

A computer-based motion control device to control the servo-axis motion of a robot.

Rotor

The rotating part of a magnetic structure. In a motor, the rotor is connected to the motor shaft.

Serial Port

A digital data communications port configured with a minimum number of signal lines. This is achieved by passing binary information signals as a time series of "1"s and "0"s on a single line.

Servo Amplifier/Servo Drive

An electronic device that produces the winding current for a servo motor. The amplifier converts a low level control signal into high voltage and current levels top produce torque in the motor.

Servo System

An automatic feedback control system for mechanical motion in which the controlled or output quantity is position, velocity, or acceleration. Servo systems are closed loop systems.

Settling Time

The time required for a step response of a system parameter to stop oscillating or ringing and reach its final value.

Shunt Resistor

A device located in a servoamplifier for controlling regenerative energy generated when braking a motor. This device dissipates or "dumps" the kinetic energy as heat.

Single Point Ground

The common connection point for signal grounds in a control wiring environment.

Slew

In motion control the portion of a move made at a constant non-zero velocity.

Slew Speed

The maximum velocity at which an encoder will be required to perform.

Speed

In motion control, the concept used to describe the linear or rotational velocity of a motor or other object in motion.

Speed Regulation

For a speed control system, speed regulation is the variation in actual speed expressed as a percentage of set speed.

SPS

Steps-Per-Second. A measure of velocity used with stepping motors.

Stall Torque

The torque available from a motor at stall or zero rpm.

Static Torque

The angle the shaft rotates upon receipt of a single step command.

Stator

The non-rotating part of a magnetic structure. In a motor the stator usually contains the mounting surface, bearings, and non-rotating windings or permanent magnets.

Stiffness

The ability to resist movement induced by an applied torque. It is often specified as a displacement curve, indicating the amount a motor shaft will rotate upon application of a known external force when stopped.

Synchronism

A motor rotating at a speed correctly corresponding to the applied step pulse frequency is said to be in synchronism. Load torques in excess of the motor's capacity (rated torque) will cause a loss of synchronism.

Tachometer

An electromagnetic feedback transducer that produces an analog voltage signal proportional to rotational velocity. Tachometers can be either brush or brushless.

Tachsyn

A brushless, electromagnetic feedback transducer that produces an analog velocity feedback signal and commutation signals for a brushless servo motor. The tachsyn is functionally equivalent to hall sensors and a tachometer.

Torque

The rotary equivalent to force. Equal to the product of the force perpendicular to the radius of motion and distance from the center of rotation to the point where the force is applied.

Torque Constant

A number representing the relationship between motor input current and motor output torque. Typically expressed in units of torque/amp.

Torque Ripple

The cyclical variation of generated torque given by the product of motor angular velocity and number of commutator segments.

Torque-to-Inertia Ratio

Defined as a motor's torque divided by the inertia of its rotor, the higher the ratio the higher the acceleration will be.

Transducer

Any device that translates a physical parameter into an electrical parameter. Tachometers and encoders are examples of transducers.

Transfer Function

The ratio of the Laplace transforms of system output signal and system input signal.

Trapezoidal Profile

A motion profile in which the velocity vs. time profile resembles a trapezoid. Characterized by constant acceleration, constant velocity, and constant deceleration.

TTL

Transistor-Transistor Logic.

Variable Frequency Drive

An electronic device used to control the speed of a standard AC induction motor. The device controls the speed by varying the frequency of the winding current used to drive the motor.

Vector Control

A method of obtaining servo type performance from an AC motor by controlling two components of motor current.

Velocity

The change in position as a function of time. Velocity has both a magnitude and a direction.

Voltage Constant (or Back EMF Constant)

A number representing the relationship between Back EMF voltage and angular velocity. Typically expressed as V/Krpm.

Zero

A frequency at which the transfer function of a system goes to zero.

INDEX