FANUC

GE Fanuc Automation

Programmable Control Products

Generation D
Real-Time Operating System

Programming Manual

GFK-2205

August 2002

GFL-002

Warnings, Cautions, and Notes as Used in this Publication

Warning

Warning notices are used in this publication to emphasize that hazardous voltages, currents, temperatures, or other conditions that could cause personal injury exist in this equipment or may be associated with its use.

In situations where inattention could cause either personal injury or damage to equipment, a Warning notice is used.

Caution

Caution notices are used where equipment might be damaged if care is not taken.

Note

Notes merely call attention to information that is especially significant to understanding and operating the equipment.

Alarm Master	Genius	PROMACRO	Series Six
CIMPLICITY	Helpmate	PowerMotion	Series Three
CIMPLICITY 90–ADS	Logicmaster	PowerTRAC	VersaMax
CIMSTAR	Modelmaster	Series 90	VersaPro
Field Control	Motion Mate	Series Five	VuMaster
GEnet	ProLoop	Series One	Workmaster

Introduction

Purpose of This Programming Manual

The *Generation D Real-Time Operating System (RTOS) Programming Manual* is your guide to the basic application program design and maintenance of the DspMotion® products. The computer screen examples in the text were created in CCS 5.1.1 for Windows. All examples assume that the you are using CCS version 5.1.1 or later.

CIMPLICITY® Motion Developer Users

If you are using CIMPLICITY Motion Developer software to communicate with IMC or IMJ motion controllers, note that the screen captures and procedures described in this revision were developed using CCS for Windows software. Your CIMPLICITY Motion Developer menus and screens will be different. Please use CIMPLICITY's online help or refer to the *S2K Series Standalone Motion Controller User's Manua*l, GFK-1848 for software-specific examples and information. Use this Generation D RTOS manual for your system setup, application program development principles, and operating system resources.

Conventions

Symbol Codes

The commands and registers in Appendix A of this manual are used with IMCs, IMJs, or the Target® automation rack system. In some cases, a particular register, command, or feature may apply only to a specific product—those distinctions are noted with the following symbols:

jr	Applies to IMJ
I	Applies to IMC
⊙	Applies to Target® ARS

Registers and Commands

Appendix A contains specifics (e.g., syntax, parameters, range) about each register and command in the Generation D RTOS. Appendix I provides quick reference lists of those registers and commands by class (i.e., System Registers, Motion Commands, etc.).

Related Publications

The following publications are available at http://www.gefanuc.com/support/plc/m-MotionSolutions.htm.

Generation D RTOS Programming Manual, GFK-2205

IMC Hardware Manual, GFK-2201

Target® ARS Hardware Manual, GFK-2200

DeviceNet Reference Guide, GFK-2208

*S2K Series Standalone Motion Controller User's Manua*l, GFK-1848

IMCjr Hardware Manual, Pub 330

DeviceNet Reference Guide (for Early Firmware Revisions), Pub 305

For an in-depth DeviceNet resource, please consult the *DeviceNet Specification*, release 2.0, Errata 3, published by the Open DeviceNet Vendor Association (www.odva.org).

Contents

Chapter 1 DspMotion Overview

In This Chapter

- DspMotion system resources and capabilities
- Features of the Generation D Real-Time Operating System (RTOS)
- Modes of operation.

DspMotion System Resources and Capabilities

The DspMotion family provides all of the resources required for state-of-the-art automation machinery.

Computing Power

The DspMotion design uses a two-processor approach. This dual processing power allows the control system to provide response rates that can measurably improve machine accuracy and throughput. A 32-bit CISC microprocessor supervises the user application program, while a Digital Signal Processor (DSP) supervises the motion loop. The Target ARS has a DSP on each optional Analog and Digital Input/Output Module.

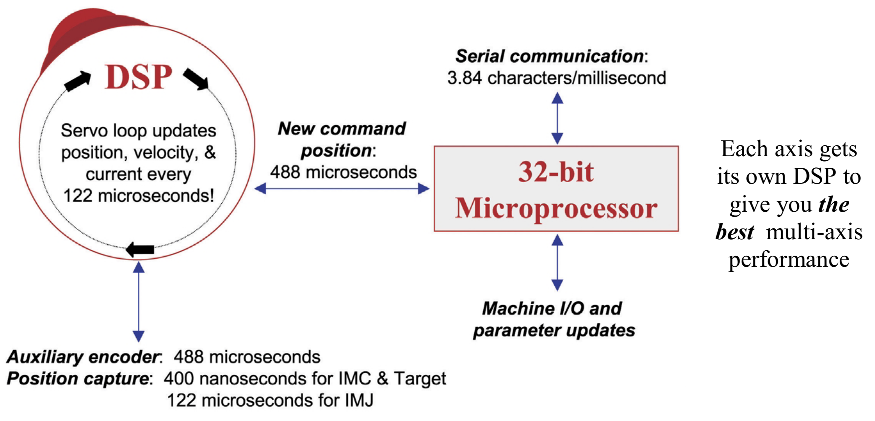

DSP
Servo loop updates position, velocity, & current every 122 microseconds!
New command position: 488 microseconds
Serial communication: 3.84 characters/millisecond
32-bit Microprocessor
Each axis gets its own DSP to give you the best multi-axis performance
Machine I/O and parameter updates
Auxiliary encoder: 488 microseconds
Position capture: 400 nanoseconds for IMC & Target
122 microseconds for IMJ

Figure 1.1: DspMotion Processing Diagram

Generation D Real-Time Operating System for Machine Control

The Generation D Real-Time Operating System (RTOS) provides the DspMotion product family with a platform that ensures that all tasks are serviced when required to meet machine timing demands. Generation D RTOS includes several features for maximum flexibility in control system architecture and machine design:

- Multitasking programs
- Motion blocks
- Labels for GOTO and GOSUB statements
- Conditional and Wait statements
- Custom, complex infix mathematical expressions
- Timers
- Built-in and custom fault handling
- Immediate mode command execution.

DspMotion Capabilities

Figure 1.2: DspMotion Capabilities

Motions	IMC & IMJ	Target
Standard trapezoidal and triangular position moves	√	√
Complex, multiple-speed position moves	√	√
Torque-limited moves	√	√
Electronic gearing and line shafting	√	√
Index synchronization	√	√
Phase synchronization	√	√
Electronic camming	√	√
Secondary position feedback	√	√
Multi-axis event-synchronized moves	√	√
Multi-axis time-synchronized moves		√
Two- to eight-axis linear interpolation		√
Two- and three-axis circular interpolation		√
Jerk-limited acceleration and deceleration (S-curve)	√	√

Programming Environment

The Generation D RTOS has been designed specifically for motion and machine control. The language uses common constructs for the IMC, the IMJ, and the Target Automated Rack System (ARS) so that applications developed for one product can be easily transferred to the other.

Registers, Commands, and Operators

Registers, commands, and operators/operands are the basic tools that you will need to create your motion control application programs. Detailed information on each is provided in Appendices A, B, and C.

Typical Syntax

A typical command line would adhere to the following syntax structure, for example:

command line: Wait IP When Not DI3 Joto 310

action: wait until axis is in position or if digital input 3 is not true, then go to label 310.

Registers can be loaded directly with a data value or indirectly with the contents of a variable, for example:

register: MPI=1000
MPI=VF100

action: the Incremental Move Position (MPI) register can be loaded with either the value 1,000 or the contents of floating point variable 100.

Math and Logical Operations

The Generation D RTOS supports full floating point math and operators for complex mathematical and logical operations (see figure 1.3).

Multifunction, single-line math operations use standard infix notation to simplify program readability and flow, for example:

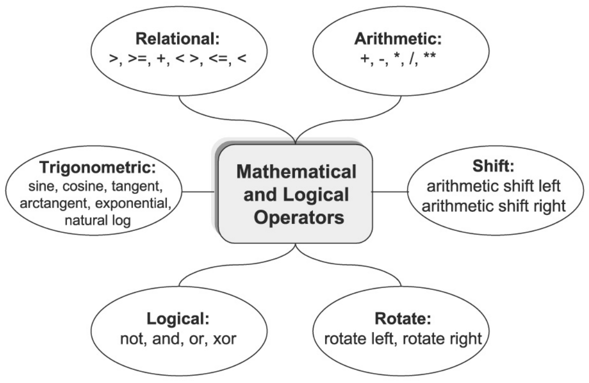


Figure 1.3: Operators in the Generation D RTOS

mathematical equation: VF1=SQR(VF2**2.+VF3**2.)

calculation: result stored in floating point variable 1 equals the square root of the sum of the squares of floating point variables 2 and 3.

Modes of Operation

The Generation D RTOS supports two modes of operation: ***preprogrammed task execution*** and ***immediate mode***.

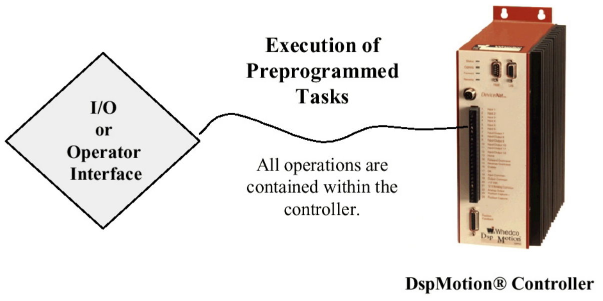


Figure 1.4: Preprogrammed Task Execution

In ***immediate mode***, the communications port functions as your control port, allowing you to send commands or load registers online and in real-time from an external source. Immediate mode is useful for applications in which motion register values and/or commands are not known in advance and may be a function of operations performed elsewhere on the machine.

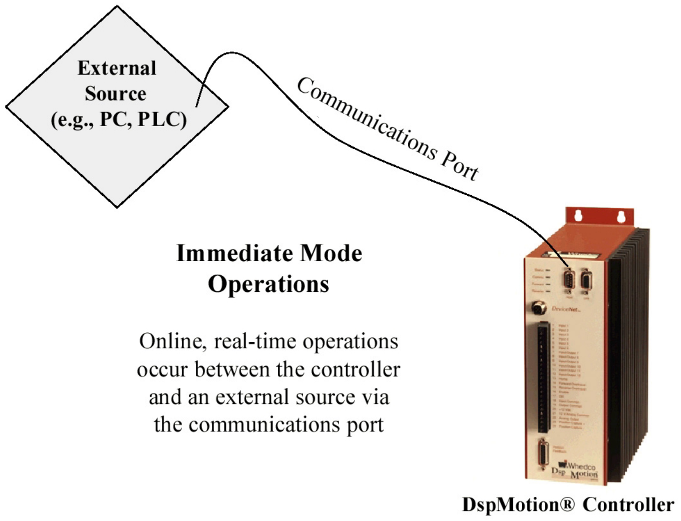


Figure 1.5: Immediate Mode Operation

Immediate mode allows the following real-time operations:

1. Send/receive variables
2. Send immediate mode commands (e.g., AUTOTUNE, CLM)
3. Load/send new register values
4. Query system status and register values
5. Send motion commands.

Related Publications

Publications dedicated to DspMotion controller set-up, configuration, and programming include:

IMC Hardware Manual, GFK-2201

Target ARS Field Service Manual, GFK-2200

IMCjr Hardware Manual, Pub 330

DeviceNet Reference Guide, GFK-2208

What's Next?

DspMotion lets you incorporate leading-edge motion control technology into a wide variety of automation machinery. Turn to the remaining chapters and appendices in this manual for the instructions, information, and examples that will maximize the potential of the Generation D RTOS in your system design process.

Chapter 2 Getting Started

In This Chapter

What you will need to:

- ❑ Complete a basic set-up
- ❑ Install CCS 5.1.1 or later for Windows operating system
- ❑ Communicate with your DspMotion controller
- ❑ Autotune your servo motor
- ❑ Make the motor move forward and reverse
- ❑ Stop the motor.

What You Will Need

Supplied Components

DspMotion controller
Motor (except IMC-2000 series)
CCS for Windows version 5.1.1 or later
Cables
DC power to digital I/O (provided by IMC output 20; IMJ output 19; Target Digital I/O Module output 19)

User-Supplied Components

AC power (to controller and PC)
AC power connection
16-gauge wire to jumper I/O connector(s)
Computer
Drive and motor (required only for IMC-2000 series)

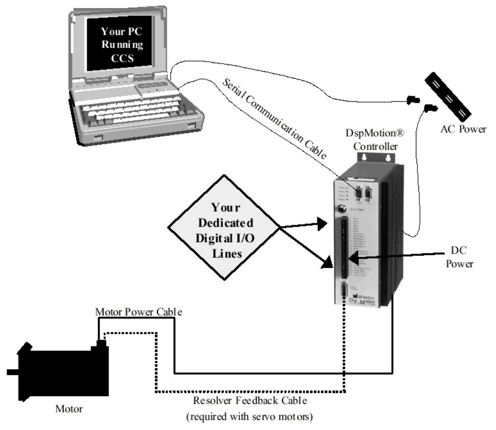

Your PC Running CCS
Serial Communication Cable
AC Power
DspMotion® Controller
Your Dedicated Digital I/O Lines
DC Power
Motor Power Cable
Motor
Resolver Feedback Cable (required with servo motors)

Figure 2-1: A DspMotion Control System

The Process for Basic Set-up

The flowchart in figure 2.2 documents the process for completing a basic setup for an IMC, IMJ, or a Target system. The remainder of this chapter expands upon each action in figure 2.2 with step-by-step instructions and illustrations for each part of the procedure. Once you have completed this basic set-up, you will be ready to start programming your DspMotion control system.

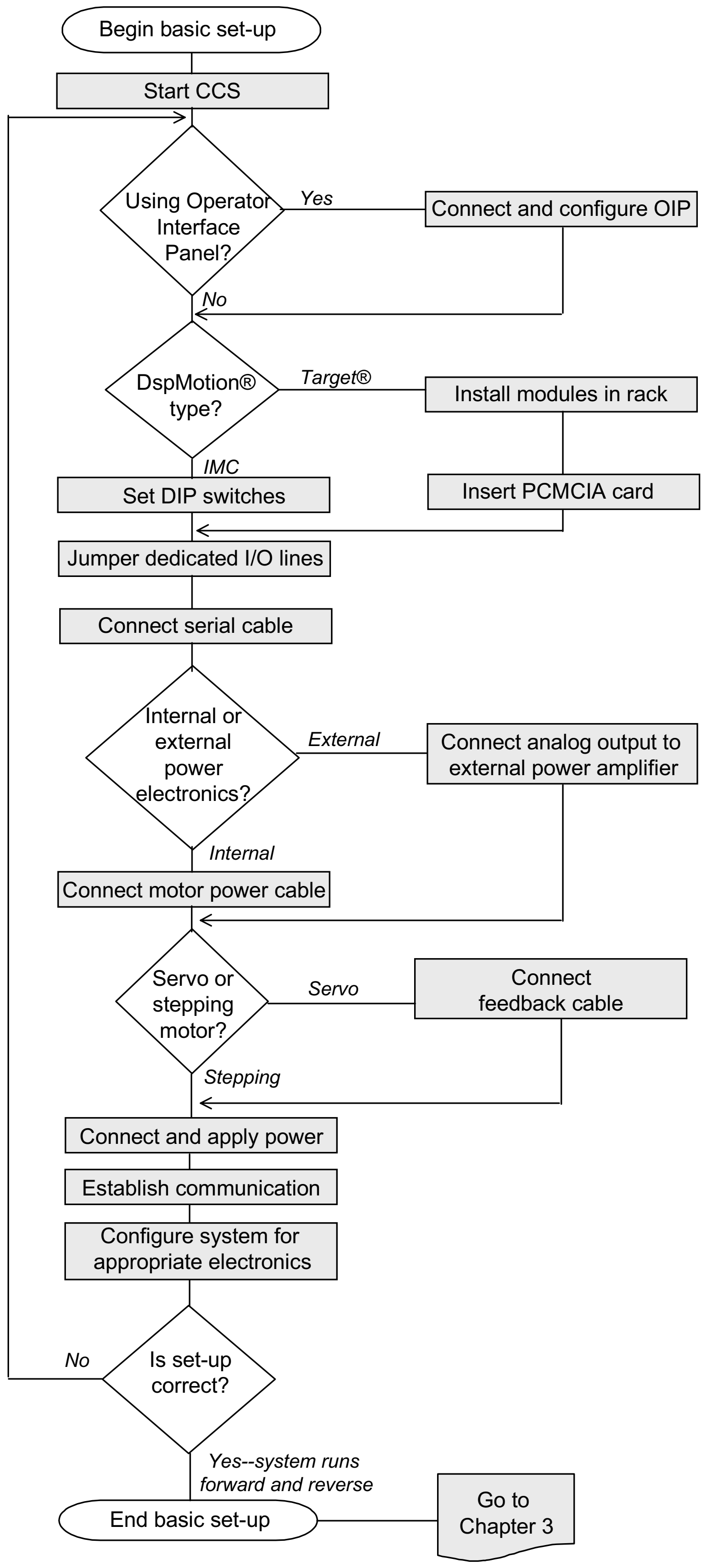


Figure 2-2: The Process for Basic Set-up

Start CCS Version 5.1.1 or Later

Figure 2.3

Minimum System Requirements for CCS for Windows	
Microprocessor	486 and faster recommended
Operating System	v. 5.1.1 -- Windows 3.1, 95, or NT v. 6.0 or later -- Windows 95, 98, or NT 4.0
Disk Space Required	v. 5.1.1 -- 4 MB; v. 6.0 or later -- 8 MB
Serial Port	RS-232 or RS-422 communicating at 1,200; 9,600; 19,200; and 38,400 baud

Install CCS on Your PC

1. Close all Windows applications.
2. Insert the CCS disk or CD into your PC drive.
3. Windows 3.1—Click **Program Manager**/**Run**. Then type `a:\Setup`
   Windows 95, 98, or NT—From Windows Explorer, view the contents of the CD and click **Setup.exe .** Follow the on-screen prompts to install CCS.

Run CCS

Double-click on the CCS icon; or, from the **Start** menu, select **Programs**/**CCS for Windows**. CCS will open the Terminal window.

Leaving CCS open, continue through the basic set-up process to wire and apply power to the controller. If you are using an Operator Interface Panel (OIP), complete the OIP connection described on the following page. If you are not using an OIP, please skip to page 2-7 for IMJ set-up; page 2-13 for IMC set-up; or page 2-29 for Target set-up.

Terminal Window

The Terminal window in CCS allows you to communicate directly with your DspMotion controller over its serial port. Here are a few tips for talking to your controller:

1. DspMotion controllers accept new commands and registers on a line-by-line basis. After you load a register or enter a command, press the <Enter> key on your computer keyboard.

2. The DspMotion controller will tell you if it accepts the command or register with one of the following responses on the next line in the Terminal window:

accepted	"*" followed by no response or by a requested answer means that your last entry was okay and the controller is waiting for the next entry.
not accepted	"?" followed by a message, e.g., `INVALID COMMAND`, indicates that the last entry was not accepted. Additional messages are contained in Appendix D.

3. Registers are loaded using the assignment command =. For example, to load a velocity value of 100 axis units per second into an IMC, you would enter `MVL=100`.
4. You can interrogate the DspMotion controller to find the contents of registers using either the Q or ? command. For example, to learn the value of the velocity register, type `MVLQ` <Enter> or `MVL?` <Enter>. These are equivalent statements. The controller will return the contents of the velocity register on the next line, for example: `*100`.
5. You can ask the controller its status by interrogating the status and fault registers. **To learn more about these registers, turn to Chapter 5 of this manual.** For now, you can try this by typing `SRSQ` <Enter> to query the system status register.

Connect and Configure Operator Interface (OIP)—Optional

Set DIP Switches

Set the DIP switches on the bottom of the OIP to match the baud rate of the controller.

Figure 2.4: Baud Rate Settings

Baud Rate	1	2
1.200	U	U
9.600	D	U
19,200	U	D
38,400	D	D
U = up; D = down		

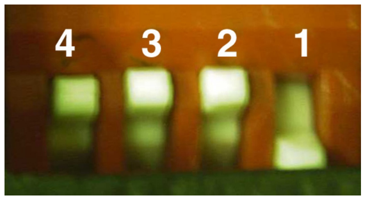


Figure 2.5: DIP Switches on OIP

IMC and IMJ Users

1. Hardwire the COM and VDC pins on the bottom of the OIP.
2. Connect the opposite end of the COM wire to the following pins on the front of the controller:

   IMC: 12 V/Analog Common (pin 21)
   IMJ: 12VCom (pin 20).

3. Connect the opposite end of the VDC wire to the following pins on the front of the controller:

   IMC: +12 VDC (pin 20)
   IMJ: +12 VDC (pin 19).

4. Connect OIP Cable (*IMC*: CBL-HSLK-6;
   IMJ: CBL-OIJR-6)

   a. *IMC*: Connect one end to the *Host* port on the front of the IMC.
      IMJ: Connect one end to the *Serial* port (figure 2.6b) on the front of the IMJ. Tighten the screws to fasten the connector.

   b. Connect the other end to its port on the OIP. Tighten the screws to fasten the connector.

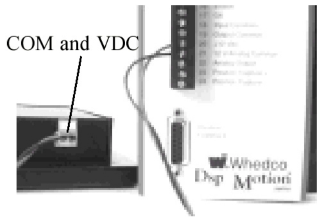


Figure 2.6: Detail of COM and VDC Wiring between OIP and Controller

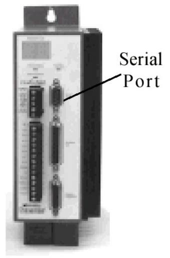


Figure 2.7: Location of Serial Port on IMJ

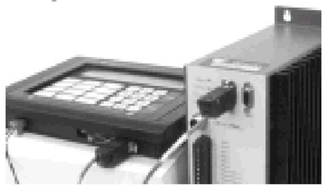


Figure 2.8: OIP Cable Connected to OIP and Controller

Target Users

1. Hardwire the COM and VDC pins on the bottom of the OIP.
2. Open the door to the System Module and remove terminal connector J6.
3. Connect the opposite end of the COM wire to J6 pin 12.
4. Connect the opposite end of the VDC wire to J6 pin 11.
5. Connect OIP Cable (CBL-HSLK-6)

   a. Connect one end to *System Program Port* J3 (the top receptacle) in the System Module. Tighten the screws to fasten the connector.

   b. Connect the other end to its port on the OIP. Tighten the screws to fasten the connector.

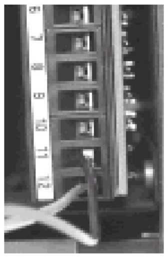


Figure 2.9: Detail of COM Wiring to Terminal Connector J6

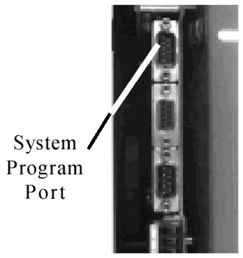


Figure 2.10: System Program Port J3

DspMotion Controller Type?

Information on basic IMJ set-up begins with *Step 1: Jumper Dedicated I/O Lines*. For the IMC set-up procedure, refer to page 2-13. Target users please refer to page 2-29.

Note to IMJ Users

The IMJ configurations in this section illustrate how to set up IMJ controllers with internal power electronics and with servo and stepping motors.

The concepts described by the steps outlined in this chapter can be applied to larger and more complex systems, but the steps themselves are not sufficient to configure a complete system. To configure a complete system, consult the *IMCjr Hardware Manual*, pub 330; then follow the initialization procedure outlined in Chapter 5 of this manual. The remainder of this chapter will guide you through a basic set-up and allow you to run your motor.

Configure IMJ

Step 1: Jumper Dedicated I/O Lines

The IMJ controller has several inputs that must be connected before the controller will run the motor. Use the guidelines provided below to wire your controller in either a sinking or sourcing configuration. These I/O configurations will allow you to use your controller in a most basic manner. See the *IMCjr Hardware Manual*, pub 330 for information on setting up the user I/O for your specific application.

For Sinking (i.e., Low-True) Connections

IMJ-_ _ _E and IMJ-_ _ _D: Jumper connections 15 & 18; 18 & 20; and 17 & 19 (on TB2 for IMJ-_ _ _E and IMJ-_ _ _D; on TB3 for IMJ-31_D).

For Sourcing (i.e., High-True) Connections

IMJ-_ _ _E, IMJ-_ _ _D, and IMJ-31_D: Jumper connections 15 & 18; 18 & 19; and 17 & 20 (on TB2 for IMJ-_ _ _E and IMJ-_ _ _D; on TB3 for IMJ-31_D).

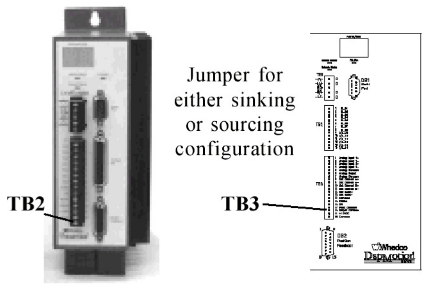


Figure 2.11: Location of TB2 and TB3 connectors on the IMC*jr*

Note: If outputs are low true, or sinking, then inputs must also be low true. If outputs are high true, or sourcing, then inputs must also be high true.

Step 2: Connect Motor Power Cable (CBL-13-MP-10, CBL-14-MP-10)

1. ***For IMJ-313_-X-D and IMJ-31_D Servo Motor Controllers:*** Connect the flying leads labeled R, S, T, and ground to the appropriately labeled slots on the bottom of the IMJ.
2. ***For IMJ-105_-1-D Stepping Motor Controllers:*** Connect the flying leads labeled Ground, B+, A/B-, and A+ to the appropriately labeled slots on the bottom of the IMJ.
3. Connect the MS connector to its mate on the motor. Push the connector into place. Twist the locking mechanism into place.

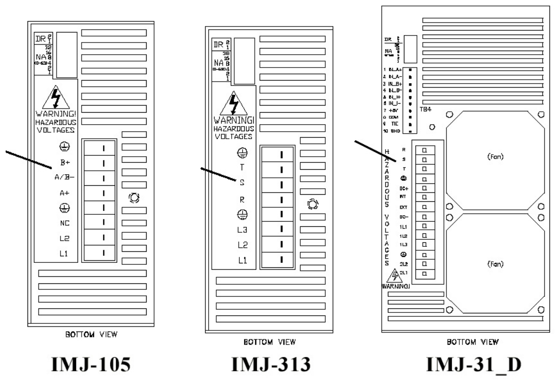


Figure 2.12: Location of Motor Power and AC Power Connections on the IMJ

Step 3: Connect Position Feedback Cable (Servo only)

1. Connect the D-shell connector to its mate, labeled position feedback on the front of the IMJ. Tighten the screws to fasten the connector.
2. Connect the MS connector to its mate on the motor. Push the connector into place. Twist the locking mechanism to secure.

Step 4: Connect and Apply AC Power

Single-Phase AC Input

IMJ-105_-1-D and IMJ-313: Connect power wires to the L1, L2, and ground connections on the bottom of the controller (see figure 2.12).

IMJ-317: Connect power wires to the 1L1, 1L2, and ground connections on the bottom of the controller (see figure 2.12). To supply power to the logic circuit, jumper the 1L2 to the 2L2 connection; then jumper the 1L1 to the 2L1 connection.

CAUTION! DO NOT *jumper the 1L3 connection.*

Three-Phase AC Input

IMJ-313: Connect power wires to the L1, L2, L3, and ground connections on the bottom of the controller (see figure 2.12).

IMJ-317, IMJ-31GD, and IMJ-31TD: Connect power wires to the 1L1, 1L2, 1L3, and ground connections on the bottom of the controller (see figure 2.12). To supply power to the logic circuit, jumper the 1L2 to the 2L2 connection; then jumper the 1L1 to the 2L1 connection.

Apply Power to the IMCjr

Apply the proper AC voltage to the controller (see the table in figure 2.13 below):

Figure 2.13: AC Input Power Requirements

Model	Rating	VAC	Input Frequency
IMJ-105_-1-D	5 Amps Continuous	90-130 VAC, single-phase @ 10.0 Amps	50 – 440 Hz
IMJ-313_-X-D	3 Amps Continuous	90-250 VAC single-phase @ 7 Amps or three-phase @ 4 Amps	50 – 440 Hz
IMJ-317_-X-D	7.2 Amps Continuous	90-250 VAC single-phase @ 17 Amps or three-phase @ 9 Amps	50 – 440 Hz
IMJ-31GD-2-D	16 Amps Continuous	180-250 VAC three-phase @ 19 Amps	50 – 440 Hz
IMJ-31TD-2-D	28 Amps Continuous	180-250 VAC three-phase @ 34 Amps	50 – 440 Hz

If You Have a Motor with a Brake ...

Apply the proper DC voltage to the brake to release it. See figure 2.14:

Figure 2.14

DC Voltage to Release Brake		
Motor Type	**Brake Type**	
	B	**9**
3N20, 3N30, 3S20, 3S30, 3S40	24 VDC	100 VDC
3S60, 3S80	24 VDC	90 VDC

Step 5: Establish Communication using CCS for Windows

Connect Serial Communication Cable (CBL-H1IC-10)

CCS allow users to communicate serially to their DspMotion controllers. Getting connected is easy!

1. Connect the end labeled “IMC or OIP” to the *Serial* port on the front of the IMJ. Tighten the screws to fasten the connector.
2. Connect the end labeled “RS232 Port” into the RS-232 serial communication port on your computer. Tighten the screws to fasten the connector.

Establish Communication

1. From the CCS Terminal window, click **Options/Communication Setup**.
2. Check ***Serial*** communication.
3. Select a COM Port for the IMJ.
4. Select **9600** as the Baud Rate and click ***OK.***
5. Click ***Options/Controller Settings.***
6. Select **Serial** as the ***Communication Type.***
7. Select **IMJ** as the ***Controller Type*** and click ***OK***.

Press the <Enter> key several times until the IMJ signs on.

Note*: Turn to page 2-4 for tips on working in the CCS Terminal window.*

Step 6: Configure the System

This setup procedure presumes that the IMJ retains its factory default configuration. If your IMJ has been previously configured, you must clear the memory by typing KLALL <Enter>, then CLM <Enter> in the CCS Terminal window.

Configure the Drive

1. Set the continuous current output in the CURC register. See motor and drive product labels for continuous current ratings. Use the following equation to calculate CURC:

   CURC = motor continuous current rating/drive continuous current rating, e.g.,

   100% x 2.8 Amps/3.0 Amps = 93%
   Type `CURC=93` <Enter> in the Terminal window.

Use the tables in figures 2.15 and 2.16 to determine the correct CURC value for your servo or stepper system.

2. ***Servo motor users:*** set the motor inductance in the KL register. Use the table in figure 2.15 to determine the correct KL value for your system.
3. ***Stepping motor users:*** set the power save current output in the CURS register, e.g.,

   Type `CURS=50` <Enter> in the Terminal window.

   This sets the power save current (current produced when the motor is at rest) to 50% of the continuous current rating of the drive.
4. ***Stepping motor users:*** set the motor number register, KM, to the KM number on your stepping motor label; or use the values provided in figure 2.16.

Configure Servo Motor

The servo motor must **not** be connected to a load. Type `MOTORSET` <Enter> in the Terminal window.

Tune Servo Motor

The servo motor **must be** connected to the load. Type `AUTOTUNE` <Enter> in the CCS Terminal window.

Figure 2.15: Servo Motor CURC and KL Values

Servo Motor	**CURC** *3 Amps*	**CURC** *7.2 Amps*	**CURC 16 Amps**	**CURC** *28 Amps*	**Servo KL**
3N21-H	100	42	n/a	n/a	4
3N22-H	100	42	n/a	n/a	6
3N24-G	87	36	n/a	n/a	9
3N31-H	100	46	21	n/a	10
3N32-G	100	42	n/a	n/a	18
3N32-H	100	86	39	22	4
3N33-G	93	39	n/a	n/a	25
3N33-H	100	79	36	20	6
3S22-G	50	21	n/a	n/a	21
3S32-G	100	42	n/a	n/a	23
3S33-G	100	44	20	n/a	22
3S33-H	100	88	40	23	6
3S34-G	100	42	n/a	n/a	30
3S35-G	100	42	n/a	n/a	42
3S43-G	97	40	n/a	n/a	53
3S43-H	100	79	36	20	13
3S45-G	100	76	34	20	20
3S45-H	100	100	69	39	5
3S46-G	100	76	34	20	25

Servo Motor	CURC *3 Amps*	CURC *7.2 Amps*	CURC 16 Amps	CURC *28 Amps*	Servo KL
3S46-H	100	100	69	40	6
3S63-G	100	100	69	40	9
3S63-H	100	100	100	79	2
3S65-G	100	100	68	39	14
3S65-H	100	100	100	77	3
3S67-G	100	100	71	40	18
3S67-H	100	100	100	81	5
3S84-G	100	100	100	100	3
3S86-G	100	100	100	100	4
3S88-G	100	100	100	100	4
3S8A-G	100	100	100	88	7

Figure 2.16: Stepping Motor CURC and KM Values

Stepping Motor	Stepper CURC	Stepper KM
1221-_-A-E-_	70	TBD
1231-_-A-E-_	62	TBD
1324-_-D-E-_	54	6
1337-_-D-E-_	82	3
1350-_-A-E-_	100	1
1350-_-D-E-_	80	4

See your stepper motor label for KM for motors not on this list.

Step 7: Verify that Set-up is Correct

Verify Feedback Connection (Servo Only)

1. Query the position register PSA to learn the motor position. Type PSAQ <Enter> or PSA? <Enter> at the Terminal window.
2. Manually turn the motor shaft to a new position. Query the position register once again. A new value should be displayed; if not, check your cable connections.

Enable the Drive

1. Type RSF <Enter> to clear the Fault condition. The digital LED on the front of the controller should now read OK to indicate that the drive is enabled and that the CPU and operating system are functional.

   Note: *To set the controller to the faulted state, type* STF *<Enter>—this will change the digital LED to SF (software fault) status.*

2. The motor will now have holding torque. Try to turn the motor shaft manually—it should resist your efforts to turn it. The *Fwd/Rev* LED on the front of the controller will turn green for a clockwise turn or yellow for a counterclockwise turn.

Know How to Stop or Halt the Motor

To Stop the Motor: Type `ST` <Enter> in the Terminal window—the motor will decelerate to a stop.

To Halt the Motor: Type `HT` <Enter> in the Terminal window—the motor will immediately hard-stop all motion.

Run the Motor

Type `MVL=50000` <Enter> to change the default velocity value.
Type `MAC=50000` <Enter> to change the default acceleration value.

Type `RVF` <Enter> to run the motor forward.
Type `RVR` <Enter> to run the motor in reverse.
Type `ST` <Enter> to stop either motion.

If your motor runs forward and reverse, congratulations! You have successfully completed a basic system set-up.

If your IMJ set-up is incorrect, return to step 1 to check your settings and connections.

Note to IMC Users

The IMC configurations in this section illustrate how to set up IMCs with internal and external power electronics and with servo and stepping motors.

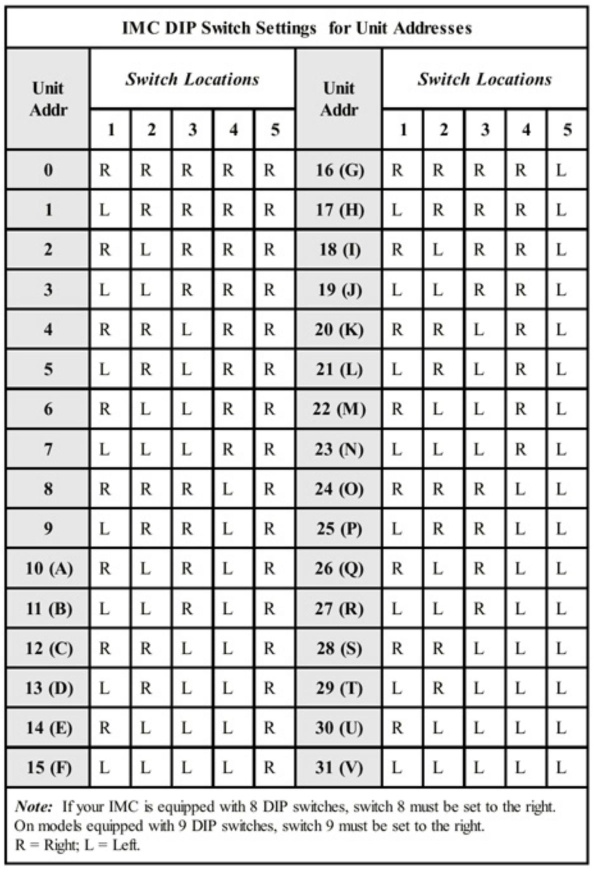
The concepts described by the steps outlined in this chapter can be applied to larger and more complex systems, but the steps themselves are not sufficient to configure a complete system. To configure a complete system, consult GFK-2201, the *IMC Hardware Manual*, for installation and wiring information; then follow the initialization procedure outlined in Chapter 5 of this manual.

Configure IMC(s)

Step 1: Set DIP Switches for Serial Port Configuration

Unit Address. Ensure that IMC power is **off**. Use the DIP switches, located on the bottom of the IMC, to set the IMC address. Switch positions 1 through 5 let you set addresses from 0 through 31. The table shown in figure 2.17 indicates the DIP switch setting you must use for each address. The letters A through V are used as the address characters for addresses 10 through 31.

Figure 2.17

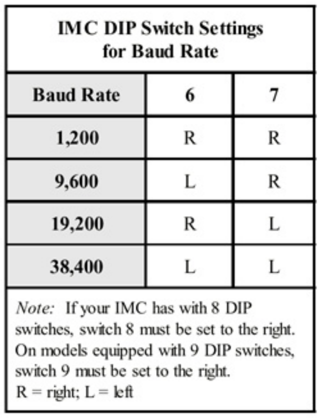


IMC DIP Switch Settings for Unit Addresses											
Unit Addr	*Switch Locations*					Unit Addr	*Switch Locations*				
	1	2	3	4	5		1	2	3	4	5
0	R	R	R	R	R	**16 (G)**	R	R	R	R	L
1	L	R	R	R	R	**17 (H)**	L	R	R	R	L
2	R	L	R	R	R	**18 (I)**	R	L	R	R	L
3	L	L	R	R	R	**19 (J)**	L	L	R	R	L
4	R	R	L	R	R	**20 (K)**	R	R	L	R	L
5	L	R	L	R	R	**21 (L)**	L	R	L	R	L
6	R	L	L	R	R	**22 (M)**	R	L	L	R	L
7	L	L	L	R	R	**23 (N)**	L	L	L	R	L
8	R	R	R	L	R	**24 (O)**	R	R	R	L	L
9	L	R	R	L	R	**25 (P)**	L	R	R	L	L
10 (A)	R	L	R	L	R	**26 (Q)**	R	L	R	L	L
11 (B)	L	L	R	L	R	**27 (R)**	L	L	R	L	L
12 (C)	R	R	L	L	R	**28 (S)**	R	R	L	L	L
13 (D)	L	R	L	L	R	**29 (T)**	L	R	L	L	L
14 (E)	R	L	L	L	R	**30 (U)**	R	L	L	L	L
15 (F)	L	L	L	L	R	**31 (V)**	L	L	L	L	L

Note: If your IMC is equipped with 8 DIP switches, switch 8 must be set to the right. On models equipped with 9 DIP switches, switch 9 must be set to the right.
R = Right; L = Left.

Baud Rate. Using switches 6 and 7, set the baud rate to 1,200; 9,600; 19,200; or 38,400. Switch position 8 must be set to the right for serial port setting. Figure 2.18 maps DIP switch settings to their appropriate baud rates.

Figure 2.18



IMC DIP Switch Settings for Baud Rate		
Baud Rate	**6**	**7**
1,200	R	R
9,600	L	R
19,200	R	L
38,400	L	L

Note: If your IMC has with 8 DIP switches, switch 8 must be set to the right. On models equipped with 9 DIP switches, switch 9 must be set to the right.
R = right; L = left

Step 2: Jumper Dedicated I/O Lines

The IMC controller has several inputs that must be connected before the controller will run the motor. The I/O configuration shown here will allow you to use your controller in a most basic manner—see the *IMC-D Hardware Manual* (GFK-2201) for information on setting up the user I/O for your specific application.

For Sinking (i.e., Low-True) Connections

Jumper connections 14, 15, 16 & 19
Jumper connections 19 & 21
Jumper connections 18 & 20

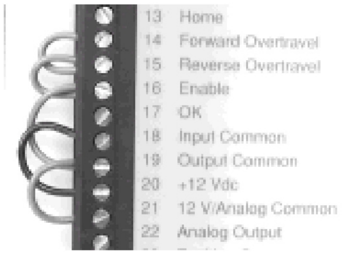


Figure 2.19: Sinking Connections

For Sourcing (i.e., High-True) Connections

Jumper connections 14, 15, 16 & 19
Jumper connections 19 & 20
Jumper connections 18 & 21

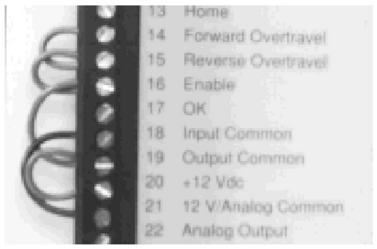


Figure 2.20: Sourcing Connections

***Note**: If outputs are low true, or sinking, then inputs must also be low true. If outputs are high true, or sourcing, then inputs must also be high true.*

Go to DspMotion Controller Model-Specific Instructions

The remaining *getting started* instructions are specific to the IMC model that you have purchased. Please turn to the page that is appropriate for your model, and then complete step 3 through the end:

IMC-1000 Series	Page 2-16
IMC-2000 Series	Page 2-20
IMC-3000 Series	Page 2-24

IMC-1000 Series

Step 3: Connect Motor Power Cable (CBL-13-MP-10, CBL-14-MP-10)

1. Connect the flying leads labeled A+, A-, B+, B-, SH, and ground (G) into the appropriately labeled slots on the bottom of the controller. Connect the SH (shield) wire to the second (lower) ground pin labeled G.
2. Connect the MS connector to its mate on the motor. Push the connector into place. Twist the “locking mechanism” into place.

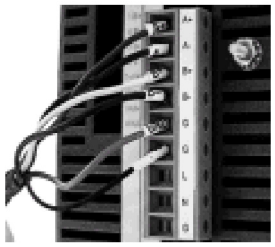


Figure 2.21: Detail of Motor Power Lead Wires to IMC

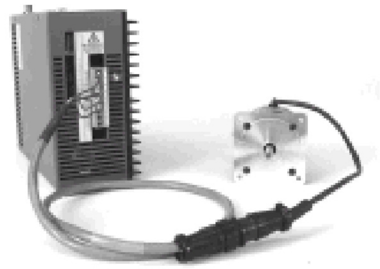


Figure 2.22: CBL-13-MP-10 Stepping Motor Power Cable for Standard Construction Motor

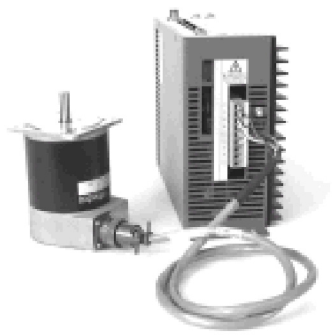


Figure 2.23: CBL-14-MP-10 Stepping Motor Power Cable for Splashproof Construction Motor

Step 5: Connect and Apply AC Power

Connect Single-Phase AC Input

Connect power wires to the L, N, and ground (G) connections on the bottom of the controller.

Apply Power to the IMC

Apply the proper AC voltage to the controller. The IMC-1000 Series is rated for 90-130 VAC single-phase input at 50-440 Hz.

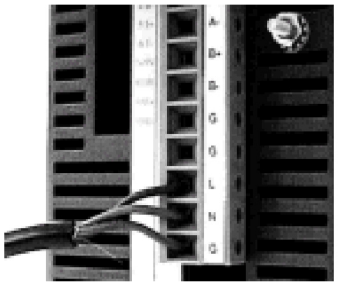


Figure 2.24: Detail of Input Power Lead Wires to IMC

Step 6: Establish Communication

Connect Serial Communication Cable (CBL-H1IC-10)

CCS allow users to communicate serially to their DspMotion controllers. Getting connected is easy!

1. Connect the end labeled "IMC or OIP" to the *Host* port on the front of the IMC. Tighten the screws to fasten the connector.

   Note: *If you are using a OIP, connect the end labeled "IMC or OIP" to its mate on the OIP.*

2. Connect the end labeled "RS232 Port" into the RS-232 serial communication port on your computer. Tighten the screws to fasten the connector.

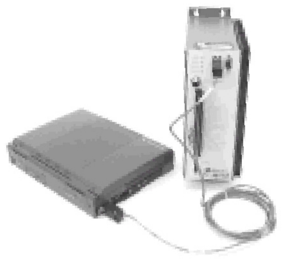


Figure 2.25: CBL-H1IC-10 Serial Communication Cable

Establish Communication

1. Click ***Options/Communication Setup*** from CCS.
3. Check ***Serial*** communication.
4. Select a COM Port for your Motion controller.
5. Select the appropriate ***Baud Rate*** (must match DIP switch settings on IMC) and click ***OK.***
6. Click ***Options/Controller Settings.***
7. Select **Serial** as the ***Communication Type.***
8. Select **IMC** as the ***Controller Type.***
9. Select the ***Controller Address*** (0 through V) that matches your IMC DIP switch settings and click *OK.*
10. Press the <Enter> key several times until the IMC signs on.

Step 7: Configure the System

This setup procedure presumes that the IMC retains its factory default configuration. If your IMC has been previously configured, you must clear the memory by typing `KLALL` <Enter>, then `CLM` <Enter> in the Terminal window.

Configure the Drive

1. Set the continuous current output in the CURC register. See motor and drive product labels for continuous current ratings. Use the following equation to calculate CURC:

   CURC = motor continuous current rating/drive continuous current rating, e.g.,

   100% x 2.7 Amps/5.0 Amps = 54%
   Type CURC=93 <Enter> in the Terminal window.

   Use the table in figure 2.26 to determine the correct CURC value for your system.

Figure 2.26: Motor CURC Values IMC Stepper Unit

Motor	*CURC*
1221-_-A-E-_	70
1231-_-A-E-_	62
1324-_-A-E-_	100
1324-_-D-E-_	54
1337-_-A-E-_	100
1337-_-D-E-_	82
1350-_-A-E-_	100
1350-_-D-E-_	80
1362-_-A-E-_	100
1454-_-A-E-_	100
1480-_-A-E-S	100

2. Set the power save current output in the CURS register, e.g.,

   Type `CURS=50` <Enter> in the Terminal window.

   This sets the power save current (current produced when the motor is at rest) to 50% of the continuous current rating of the drive.

Step 8: Verify that Set-up is Correct

Enable the Drive

1. Type RSF <Enter> to clear the Fault condition. The Status LED on the front of the controller will change from red to green.

   Note: *To set the controller to the faulted state, type* `STF` *<Enter>—this will change the Status LED to red.*

2. The motor will now have holding torque. Try to turn the motor shaft manually—it should resist your efforts to turn it.

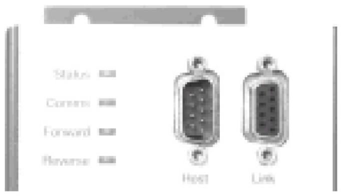


Figure 2.27: Front Panel LEDs

Know How to Stop or Halt the Motor

To Stop the Motor: Type `ST` <Enter> in the Terminal window—the motor will decelerate to a stop.

To Halt the Motor: Type `HT` <Enter> in the Terminal window—the motor will immediately hard-stop all motion.

Run the Motor

Type `MVL=50000` <Enter> to change the default velocity value.
Type `MAC=50000` <Enter> to change the default acceleration value.

Type `RVF` <Enter> to run the motor forward.
Type `RVR` <Enter> to run the motor in reverse.
Type `ST` <Enter> to stop either motion.

If your motor runs forward and reverse, congratulations! You have successfully completed a basic system set-up.

If your set-up is incorrect, return to step 1 to check your settings and connections.

IMC-2000 Series

Step 3: Connect Analog Output Cable (CBL-20-AT-10)

1. Connect the flying lead labeled ALG COM to pin 21 and the lead labeled ALG to pin 22 on the front of the IMC.
2. Connect the ends labeled +IN, –IN, and COM to their appropriate pins on your external power electronics.

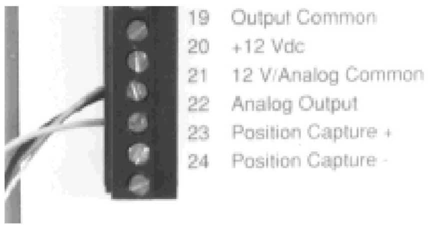


Figure 2.28: Detail of Analog Connection to IMC

Step 4: Connect Motor Power Cable

Use the connection procedure that is appropriate for your external power electronics.

Step 5: Connect Encoder Feedback Cable (CBL-20-ED-10)

1. Insert the D-Shell connector into the Position Feedback port on the front of the IMC. Tighten the screws to fasten the connector.
2. Connect the leads labeled A+, A-, B+, B-, I+, I-, and ground (G) to your external power electronics.

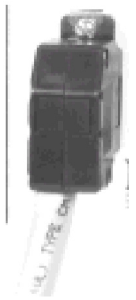


Figure 2.29: Encoder Feedback Connection to IMC

Step 6: Connect and Apply AC Power

Connect Single-Phase AC Input

Connect power wires to the L1, L2, and ground (G) connections on the bottom of the controller.

Apply Power to the IMC

Apply the proper AC voltage to the controller. The IMC-2000 Series is rated for 90-250 VAC single-phase input at 50-440 Hz.

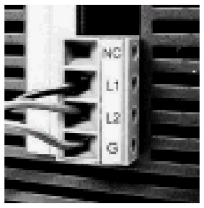


Figure 2.30: Detail of AC Power Connections to IMC

Step 7: Establish Communication

Connect Serial Communication Cable (CBL-H1IC-10)

1. Connect the end labeled "IMC or OIP" to the *Host* port on the front of the IMC. Tighten the screws to fasten the connector.

   Note: *If you are using an OIP, connect the end labeled "IMC or OIP" to its mate on the OIP.*

2. Connect the end labeled "RS232 Port" into the RS-232 serial communication port on your computer. Tighten the screws to fasten the connector.

Establish Communication

1. Click ***Options/Communication Setup*** from CCS.
3. Check ***Serial*** communication.
4. Select a COM Port for your Motion controller.
5. Select the appropriate ***Baud Rate*** (must match DIP switch settings on IMC) and click ***OK***.
6. Click ***Options/Controller Settings.***
7. Select ***Serial*** as the ***Communication Type.***
8. Select ***IMC*** as the ***Controller Type.***

9. Select the ***Controller Address*** (0 through V) that matches your IMC DIP switch settings and click *OK*.
10. Press the <Enter> key several times until the IMC signs on.

    Note: *Turn to page 1-4 for tips on working in the CCS Terminal window.*

Step 8: Configure the System

This setup procedure presumes that the IMC retains its factory default configuration. If your IMC has been previously configured, you must clear the memory by typing `KLALL` <Enter>, then `CLM` <Enter> in the Terminal window.

Tune the Motor Using Autotune

Type `FR=n` <Enter> *(n is the appropriate value for your system. See Appendix A for the FR register description.)*

Type `AUTOTUNE` <Enter>

AUTOTUNE will execute correctly only if the IMC-2000 is connected to a drive whose output current is proportional to the IMC output voltage (+/-10 V) and the drive produces full continuous current when the IMC output voltage is 5.0 volts. Drives that operate as described above are often referred to as *torque mode* drives.

Tune the Motor Manually

If your drive doesn't meet the above requirements for AUTOTUNING, consult factory for a manual tuning procedure.

Step 9: Verify that Set-up is Correct

Verify Feedback Connection

1. Query the position register PSA to learn the motor position. Type `PSAQ` <Enter> or `PSA?` <Enter> at the Terminal window.
2. Manually turn the motor shaft to a new position. Query the position register again. A new value should be displayed; if not, check your cable connections.

Enable the Drive

1. Type `RSF` <Enter> to clear the Fault condition. The Status LED on the front of the controller will change from red to green.

   Note: To set the controller to the faulted state, type `STF` <Enter>—this will change the *Status* LED to red.

2. The motor will now have holding torque. Try to turn the motor shaft manually—it should resist your efforts to turn it.

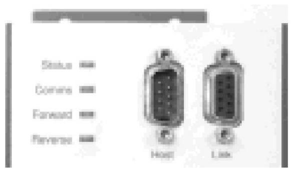


Figure 2.31: Front Panel LEDs

Know How to Stop or Halt the Motor

To Stop the Motor: Type `ST` <Enter> in the Terminal window—the motor will decelerate to a stop.

To Halt the Motor: Type `HT` <Enter> in the Terminal window—the motor will immediately hard-stop all motion.

Run the Motor

Type `MVL=10000` <Enter> to change the default velocity value.
Type `MAC=10000` <Enter> to change the default acceleration value.

Type `RVF` <Enter> to run the motor forward.
Type `RVR` <Enter> to run the motor in reverse.
Type `ST` <Enter> to stop either motion.

If your motor runs forward and reverse, congratulations! You have successfully completed a basic system set-up.

If your set-up is incorrect, return to step 1 to check your settings and connections.

IMC-3000 Series

Step 3: Connect Motor Power Cable (CBL-34-MP-10, CBL-3C-MP-10, CBL-3P-MP-10, CBL-38-MP-10)

1. Connect the flying leads labeled R, S, T, and ground into the screw terminal located at the bottom of the controller. Match the label from each lead to the appropriately labeled terminal slot.
2. Connect the MS connector to its mate on the motor. Push the connector into place. Twist the locking mechanism into place.

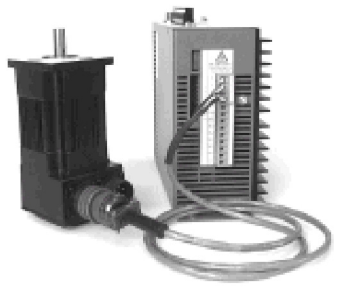


Figure 2.32: CBL-34-MP-10 Motor Power Cable

Step 4: Connect Feedback Cable (CBL-3C-RD-10, CBL-34-ED-10)

The IMC-3100 uses a resolver feedback cable ***(CBL-3C-RD-10)***. The IMC-3000 uses an encoder feedback cable ***(CBL-34-ED-10)***.

1. Connect the D-shell connector to its mate, labeled "position feedback," on the front, lower-left side of the controller. Tighten screws to fasten connector.
2. Connect the MS connector to its mate on the motor. Push the connector into place. Twist the locking mechanism into place.

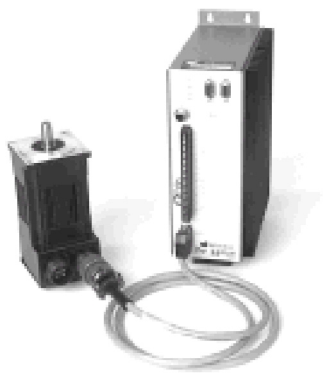


Figure 2.33: CBL-3C-RD-10 Resolver Feedback Cable

Step 5: Connect and Apply AC Power

Single-Phase AC Input

1. Connect power wires to the 1L1, 1L2, and ground connections on the bottom of the controller.
2. To supply power to the logic circuit, jumper the 1L2 connection to the 2L2, and jumper the 1L1 connection to the 2L1.

CAUTION! DO NOT *jumper the 1L3 connection.*

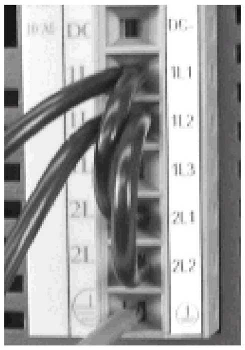


Figure 2.34: Detail of Single-Phase AC Input

Three-Phase AC Input

1. Connect power wires to the 1L1, 1L2, 1L3, and ground connections on the bottom of the controller.
2. To supply power to the logic circuit, jumper the 1L2 connection to the 2L2. Then jumper the 1L1 connection to the 2L1.

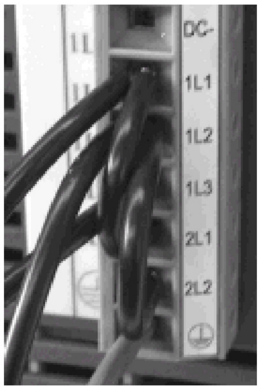


Figure 2.35: Detail of Three-Phase AC Input

Apply Power to the IMC

Apply the proper AC voltage to the controller. The IMC-3000 Series is rated as follows:

3 and 6 Amp Units: 90-250 VAC single- or three-phase input at 50-440 Hz.
12 and 24 Amp Units: 180-250 VAC three-phase input at 50-440 Hz.

If You Have a Motor with a Brake…

Apply the proper DC voltage to the brake to release it. See figure 2.36:

Figure 2.36

DC Voltage to Release Brake		
Motor Type	**Brake Type**	
	B	**9**
3N20, 3N30, 3S20, 3S30, 3S40	24 VDC	100 VDC
3S60, 3S80	24 VDC	90 VDC

Step 6: Establish Communication

Connect Serial Communication Cable (CBL-H1IC-10)

1. Connect the end labeled "IMC or OIP" to the *Host* port on the front of the IMC. Tighten the screws to fasten the connector.

   Note: *If you are using an OIP, connect the end labeled "IMC or OIP" to its mate on the OIP.*

2. Connect the end labeled "RS232 Port" into the RS-232 serial communication port on your computer. Tighten the screws to fasten the connector.

Establish Communication

1. Click ***Options/Communication Setup*** from CCS.
3. Check ***Serial*** communication.
4. Select a COM Port for your Motion controller.
5. Select the appropriate ***Baud Rate*** (must match DIP switch settings on IMC) and click ***OK***.
6. Click ***Options/Controller Settings.***
7. Select **Serial** as the ***Communication Type.***
8. Select **IMC** as the ***Controller Type.***

9. Select the ***Controller Address*** (0 through V) that matches your IMC DIP switch settings and click *OK.*

10. Press the <Enter> key several times until the IMC signs on.

Note: Go to page 1-4 for tips on working in the CCS Terminal window.

Step 7: Configure the System

This setup procedure presumes that the IMC retains its factory default configuration. If your IMC has been previously configured, you must clear the memory by typing `KLALL` <Enter>, then `CLM` <Enter> in the Terminal window.

Configure the Drive

1. Set the continuous current output in the CURC register. Use the following equation to calculate CURC:

   CURC = motor continuous current rating/drive continuous current rating, e.g.,

   100% x 5.6 Amps/6.0 Amps = 93%
   Type `CURC=93` <Enter> in the Terminal window.

   Use the table in figure 2.37 to determine the correct CURC value for your system.

Figure 2.37: Motor CURC Values

Motor	CURC-3 Amp Drive	CURC-6 Amp Drive	CURC-12 Amp Drive	CURC-24 Amp Drive
3S22-G	46	23	n/a	n/a
3S32-G	96	48	24	n/a
3S33-G	100	53	26	n/a
3S33-H	100	100	53	26
3S34-G	100	50	25	n/a
3S35-G	96	48	24	n/a
3S43-G	96	48	24	n/a
3S43-H	100	93	46	23
3S45-G	100	91	45	22
3S45-H	100	100	91	45
3S46-G	100	91	45	22
3S46-H	100	100	91	45
3S63-G	100	100	91	45
3S65-G	100	100	89	44
3S67-G	100	100	94	47
3S88-G	100	100	100	100
3S8A-G	100	100	100	100

2. Set the peak current output in the CURP register, e.g.,

   Type `CURP=100` <Enter> in the Terminal window.

   This sets the peak current output of the controller to 100% of maximum. The maximum peak current is two times the drive's continuous rating.

Configure the Motor

The motor must **not** be connected to a load. Type `MOTORSET` <Enter> in the Terminal window.

Tune the Motor

The motor **must be** connected to the load. Type `AUTOTUNE` <Enter> in the CCS Terminal window.

Step 8: Verify that Set-up is Correct

Verify Feedback Connection

1. Query the position register PSA to learn the motor position. Type `PSAQ` <Enter> or `PSA?` <Enter> at the Terminal window.
2. Manually turn the motor shaft to a new position. Query the position register once again. A new value should be displayed; if not, check your cable connections.

Enable the Drive

1. Type `RSF` <Enter> to clear the Fault condition. The Status LED on the front of the controller will change from red to green.

   Note: *To set the controller to the faulted state, type* `STF` *<Enter>—this will change the Status LED to red.*

2. The motor will now have holding torque. Try to turn the motor shaft manually—it should resist your efforts to turn it.

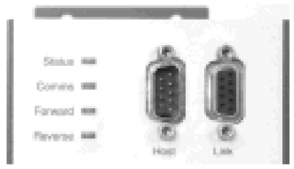


Figure 2.38: Front Panel LEDs

Know How to Stop or Halt the Motor

To Stop the Motor: Type `ST` <Enter> in the Terminal window—the motor will decelerate to a stop.

To Halt the Motor: Type `HT` <Enter> in the Terminal window—the motor will immediately hard-stop all motion.

Run the Motor

Figure 2.39: To Run Your Motor with an IMC-3000 or IMC-3100

IMC-3000 Series	IMC-3100 Series	Action
Type `MVL=50000` <Enter>	Type `MVL=10000` <Enter>	Changes the default velocity value
Type `MAC=50000` <Enter>	Type `MAC=10000` <Enter>	Changes the default acceleration/deceleration value

Then complete the following steps:

Type `RVF` <Enter> to run the motor forward.
Type `RVR` <Enter> to run the motor in reverse.
Type `ST` <Enter> to stop either motion.

If your motor runs forward and reverse, congratulations! You have successfully completed a basic system set-up.

If your set-up is incorrect, return to step 1 and check your settings and connections.

Note to Target Users

The Target Automation Rack System (ARS) configuration in this section describes a single axis in a single rack. Power modules, if used, are not paralleled.

A single Target system, when fully configured, can comprise up to eight axes and as many as three racks. The concepts described by the steps outlined in this chapter for the single-axis case can be applied to larger and more complex systems, but the steps themselves are not sufficient to configure a complete system. To configure a complete system, consult the *Target ARS Hardware Manual, GFK-2200* for installation and wiring information; then follow the initialization procedure outlined in Chapter 5 of this manual.

Target

Step 1: Install Modules in Rack

A Target rack can hold up to nine Target modules. The System Module should be placed in the right-most slot of the rack. For easy I/O jumpering, install the Axis Module next to the System Module. The Power Module must be placed in the left-most slot. Then install the Servo Module next to the Power Module. To install any Target module:

1. Align the bottom edges of the module with the slot.
2. Gently push the module into the rack (do not force the module in—it should slide easily if the edges are properly aligned).
3. Click the module into place.

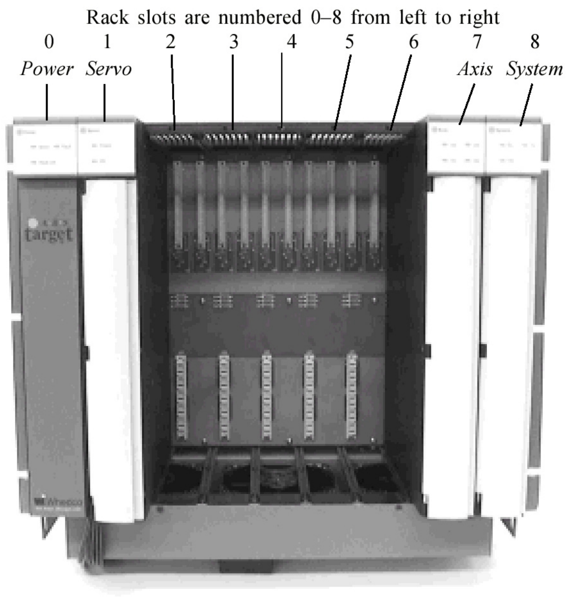


Figure 2.40: Target Rack with Required Modules Installed

Step 2: Insert PCMCIA Card

The PCMCIA card belongs in the upper slot on the front of the System Module. To insert the card:

1. Open the cover on the front of the System Module.
2. Align pins on the PCMCIA card with their receptacles in the slot.
3. Push the card completely into the slot.

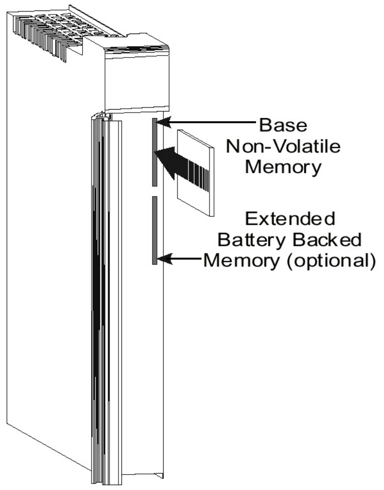


Figure 2.41: Installing the Flash Memory PCMCIA Card

Step 3: Jumper Dedicated I/O Lines

Axis Module

Jumper connections 6, 7, 8, 12, 13, 14, and 17
Connect Axis Module pin 17 to System Module pin 11
Connect Axis Module pin 4 to System Module pin 12

System Module

Jumper connections 1 and 12
Jumper connections 4, 6, and 11

Step 5: If Your Power Electronics are Internal…(if not, go to step 6)

Connect Motor Power Cable (CBL-34-MP-10, CBL-3C-MP-10, CBL-3P-MP-10, CBL-38-MP-10)

1. Connect the flying leads labeled R, S, T, and ground to the appropriately labeled Servo Module slots.
2. Connect the MS connector to its mate on the motor. Push the connector into place. Twist the locking mechanism into place.

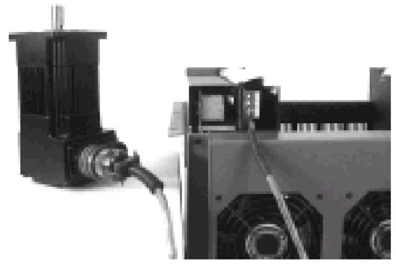


Figure 2.42: CBL-34-MP-10 Motor Power Cable to Target ARS

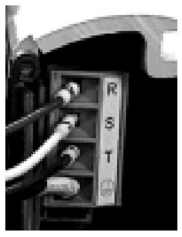


Figure 2.43: Detail of Motor Power Cable Connections

Connect Resolver Feedback Cable (CBL-3C-RD-10)

1. Connect the D-shell connector to its mate, the upper-most receptacle, on the Axis Module. The Axis 1 Feedback Location is labeled on the module door. Tighten screws to fasten the connector.
2. Connect the MS connector to its mate on the motor. Push the connector into place. Twist the locking mechanism into place.

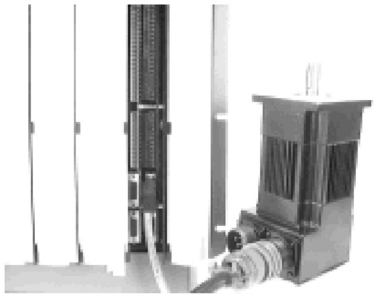


Figure 2.44: CBL-34-RD-10 Resolver Feedback Cable to Target ARS

Step 6: If Your Power Electronics are External…(if not, go to step 7)

Connect Analog Output Cable (CBL-20-AT-10)

1. Axis Module terminal strip: connect the flying lead labeled ALG COM to connector pin 3 and ALG to connector pin 1.
2. Connect the ends labeled +IN, -IN, and COM to their appropriate pins on your external power electronics.

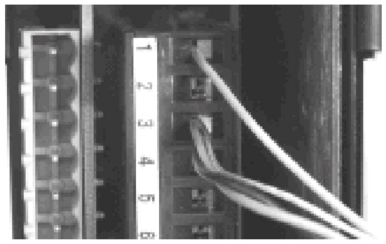


Figure 2.45: CBL-20-AT-10 Analog Output Cable to Target ARS

Connect Encoder Feedback (CBL-20-ED-10)

1. Axis Module terminal strip: connect flying leads labeled A+, A-, B+, B-, and ground to the auxiliary encoder connections as labeled on the door of the Axis Module. Tighten screws to fasten the connector.
2. Connect the opposite ends to their appropriate pins on your external power electronics.

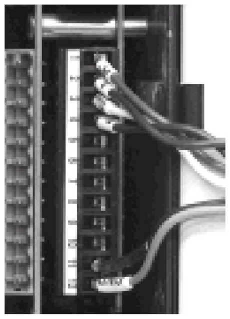


Figure 2.46: CBL-20-ED-10 Encoder Feedback Cable to Target ARS

Step 7: Connect and Apply AC Power

Connect Three-Phase AC Input

Internal power electronics:

1. Connect the power wires labeled L1-M, L2-M, L3-M, and ground to the corresponding labeled pins on the right side of the Power Module in the left-most slot of the Target rack.
2. Jumper L1-M to L1-R, and jumper L2-M to L2-R.

External power electronics:

Connect the power wires labeled L1-R, L2-R, and ground to the corresponding labeled pins on the right side of the Power Module in the left-most slot of the Target rack.

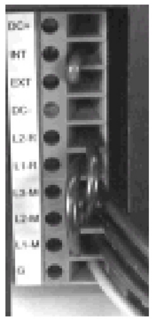


Figure 2.47: Detail AC Power Input to Target ARS

Apply Power to the Target

Apply the proper AC voltage to the controller. The Target ARS is rated as follows:

Rack input:	180-250 VAC single-phase logic power input at 50-440 Hz.
Drive input:	180-250 VAC single- or three-phase motor power input at 50-440 Hz.

If You Have a Motor with a Brake…

Apply the proper DC voltage to the brake to release it.

Figure 2.48

DC Voltage to Release Brake		
Motor Type	**Brake Type**	
	B	**9**
3N20, 3N30, 3S20, 3S30, 3S40	24 VDC	100 VDC
3S60, 3S80	24 VDC	90 VDC

Step 8: Establish Communication

Connect Serial Communication Cable (CBL-H1IC-10)

1. Connect the end labeled "IMC or OIP" to the Host port on the Target's System Module. The host port is the top-most communication port in the System Module—labeled *System Program Port*. Tighten the screws to fasten the connector.

   Note: *If you are using an OIP, connect the end labeled "IMC or OIP" to its mate on the OIP.*

2. Connect the end labeled "RS232 Port" into the RS-232 serial communication port on your computer. Tighten the screws to fasten the connector.

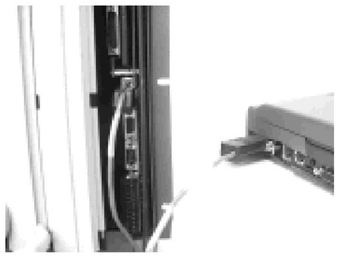


Figure 2.49: CBL-H1IC-10 Serial Communication Cable

Establish Communication

1. Click ***Options/Communication Setup*** from CCS.
3. Check ***Serial*** communication.
4. Select a COM Port for the Target and click ***OK***.
5. Click ***Options/Controller Settings.***
6. Select **Serial** as the ***Communication Type.***
7. Select **Target** as the ***Controller Type*** and click OK.

The Target is now ready to receive communication from your PC. Press the <Enter> key until the controller signs on.

Step 9: Configure System for Appropriate Electronics

This setup procedure presumes that the Target retains its factory default configuration. If your Target has been previously configured, you must clear the memory by typing `KLALL` <Enter>, then `CLM` <Enter> in the Terminal window.

Configure the Drive

1. Assign Axis and Servo Module parameters

   Internal power electronics:

   a. Type `AXIS1=SERVO` <Enter>.
   b. Type `SM1=11` <Enter> to assign Servo Module for axis 1 to rack one, slot 1.
   c. The Servo Module's green "OK" LED will turn on.

   External power electronics with encoder feedback:

   a. Type `AXIS1=EXTERNAL` <Enter>.
   b. Type `QTX1=Q4` <Enter>.
   c. Type `FR1=n` <Enter> where n is the appropriate value for your system. See Appendix A for a description of the FR register.
   d. Type `PFE1=1` <Enter>.

   External power electronics with resolver feedback:

   Type `AXIS1=EXTERNAL` <Enter>

2. Set the continuous current output in the CURC register. Use the following equation to calculate CURC:

   CURC = motor continuous current rating/drive continuous current rating, i.e.,

   100% x 5.6 Amps/6.0 Amps = 93%
   Type `CURC1=93` <Enter> in the Terminal window.

   Use the table in figure 2.50 on the following page to determine the correct CURC value for your system.

Figure 2.50: Motor CURC Values

Motor	1 Servo Module	2 Servo Modules	3 Servo Modules	4 Servo Modules
3S22-G	23	n/a	n/a	n/a
3S32-G	48	24	n/a	n/a
3S33-G	53	26	n/a	n/a
3S33-H	100	53	35	26
3S34-G	50	25	n/a	n/a
3S35-G	48	24	n/a	n/a
3S43-G	48	24	n/a	n/a
3S43-H	93	46	31	23
3S45-G	91	45	30	22
3S45-H	100	91	61	45
3S46-G	91	45	30	22
3S46-H	100	91	61	45
3S63-G	100	91	61	45
3S65-G	100	89	59	44
3S67-G	100	94	63	47
3S88-G	100	100	100	100
3S8A-G	100	100	100	100

3. Set the peak current output in the CURP register, i.e.,

   Type `CURP1=50` <Enter> in the Terminal window.

   This sets the peak current output of the controller to 50% of maximum. The maximum peak current is two times the drive's continuous rating.

Configure the Motor—Internal Drive Electronics Only

The motor must **not** be connected to a load. Type `MOTORSET1` <Enter> in the Terminal window.

Tune the Motor

The motor must be connected to the load. Type `AUTOTUNE1` <Enter> in the CCS Terminal window.

Step 10: Verify that Set-up is Correct

Verify Feedback Connection

1. Query the position register PSA to learn the motor position. Type `PSA1Q` <Enter> or `PSA1?` <Enter> at the Terminal window (where 1=axis number).
2. Manually turn the motor shaft to a new position—it should turn freely. Query the position register once again. A new value should be displayed; if not, check your cable connections.

Enable the Drive

1. Type `RSFALL` <Enter> to clear the Fault condition. The "OK" LED on the front of the System & Axis Modules will turn green.

   Note: To set the controller to the faulted state, type `STFALL` <Enter>—this will turn off the *"OK"* LEDs.

2. The motor will now have holding torque. Try to turn the motor shaft manually—it should resist your efforts to turn it.

Know How to Stop or Halt the Motor

To Stop the Motor: Type `ST1` <Enter> in the Terminal window—the motor will decelerate to a stop.

To Halt the Motor: Type `HT1` <Enter> in the Terminal window—the motor will immediately hard-stop all motion.

Run the Motor

Type `MVL1=10000` <Enter> to change the default velocity value.
Type `MAC1=10000` <Enter> to change the default acceleration/deceleration value.

Type `RVF1` <Enter> to run the motor forward.
Type `RVR1` <Enter> to run the motor in reverse.
Type `ST1` <Enter> to stop either motion.

If your system runs forward and reverse, congratulations! You have successfully completed a basic system set-up.

If your set-up is incorrect, return to step 1 and check your settings and connections.

Chapter 3 Creating Application Programs in CCS

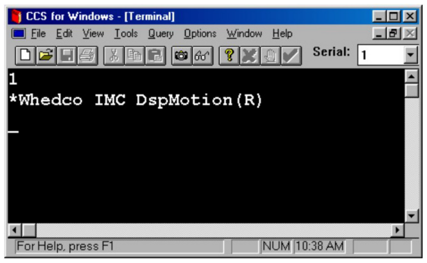
In This Chapter

- ❑ Program development tools overview
- ❑ Use the CCS ASCII file editor to create an application program
- ❑ Rules of the basic application program structure
- ❑ Send an application program to your DspMotion controller
- ❑ Run an application program
- ❑ Fix an error that occurs during a file send
- ❑ Create end user application with the free pack and go utility DspComm.

Program Development Tools Overview

CCS is a Windows-based development tool exclusively for use with DspMotion controllers. CCS for Windows provides several utilities to support your program development. Those utilities are introduced below and are described in greater detail where indicated:

Terminal Window: Gives direct communication to DspMotion controller via its serial port. The Terminal window in figure 3.1 shows communication established with a DspMotion controller. See "Terminal Windows" in chapter 2 to learn how to communicate with your controller through the Terminal window.


CCS for Windows - [Terminal]
File Edit View Tools Query Options Window Help
Serial: 1
For Help, press F1
NUM 10:38 AM

```
1
*Whedco IMC DspMotion(R)
_
```

Figure 3.1: Terminal Window

ASCII File Editor: Lets user create ASCII .txt files containing user application programs. Instructions for using the ASCII file editor begin on page 3-3.

Send Files (figure 3.2)***:*** Lets the user send ASCII files containing application programs to the controller's memory. Instructions for sending files begin on page 3-6.

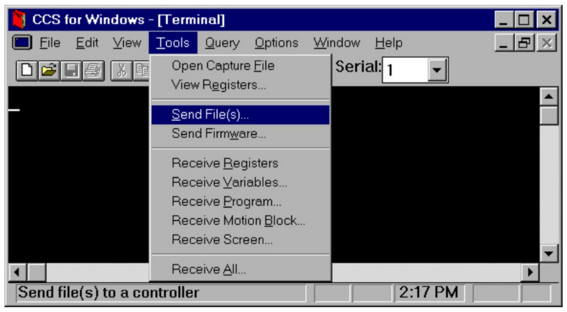


Figure 3.2: The CCS Tools Menu

ScreenView™: Gives the user an easy, graphical way to configure the display of the Operator Interface (OIP-DSP1-C). See "Creating Custom Screens" in chapter 4 to learn how to create standard and custom display screens.

Real-time Diagnostics: Query window lets the user monitor system parameters in real time. Turn to Chapter 6 to learn how to query real-time values in CCS.

Open Capture File: Captures and saves any data in a Terminal window session. Use capture to create a record of your online work. See Chapter 6 for instructions on creating a capture file in CCS.

Receive Data: Allows user to receive all or portions of the controller's memory contents. The user can then modify and/or save the memory contents in a new ASCII file. See Chapter 7 for instructions.

Online Help (figure 3.3)***:*** Gives quick access to the data provided in the appendices of this manual, along with how-to tips. Use the search engine to find registers, commands, and information by topic.

Figure 3.3: CCS Online Help

Motion Templates: Open CCS Help and cut and paste from the templates given for each motion type to build your own application program. Motion templates are also included in printed form in Appendix G.

Utility Templates: Provides a guide for creating your own FIFO buffers, PID algorithm solutions, OIP reporting, and jog and teach routines.

Excel™ Template: Lets the user create a customized function key legend insert for the OIP (figure 3.4). This template installs with CCS for Windows. Click **Start**/**CCS for Windows**/**OIPLegend** to open Excel™ and create your legend.

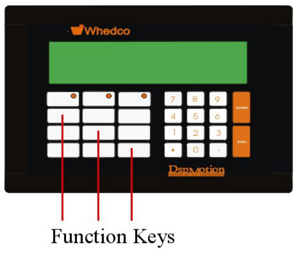


Figure 3.4: Operator Interface Panel (OIP)

Create End User Application: Bundles your application program file with the free executable for end users called DspComm. DspComm allows end users to send and receive application programs to and from controllers on systems where CCS is not installed.

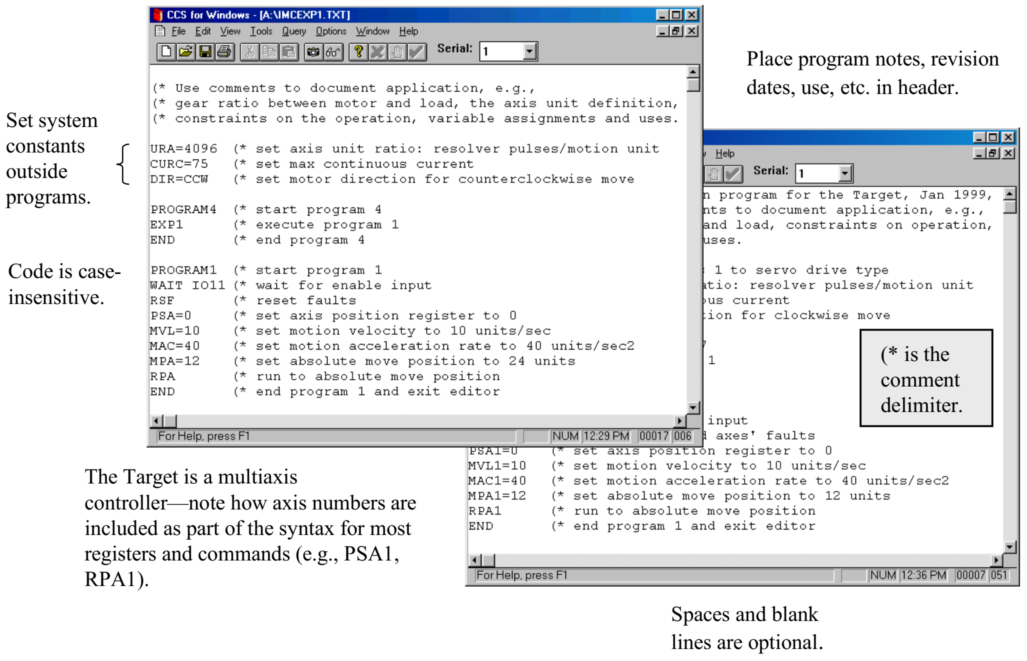
Using The ASCII File Editor

DspMotion products allow you to create an application program as an ASCII file and send it to the controller using CCS. CCS includes a resident ASCII file editor that you can use to create files labeled with a .txt file extension. Whenever you open an existing text file or create a new text file, CCS automatically enables the ASCII file editor.

Rules for Creating .txt Files in CCS

Place your system constants in the same file in which you maintain your application program. When you send the .txt file to the controller, you will simultaneously initialize the controller with the proper parameters.

Use the ASCII file comment delimiter, (*, to document your program in your .txt file. When you send your application program to the controller's memory, the controller will ignore and not store any characters that follow the (* delimiter. Comments are optional, but highly recommended for program documentation. Use the REM command to embed and store critical program flow comments directly in programs or motion blocks.



Place program notes, revision dates, use, etc. in header.

Set system constants outside programs.

Code is case-insensitive.

```
(* Use comments to document application, e.g.,
(* gear ratio between motor and load, the axis unit definition,
(* constraints on the operation, variable assignments and uses.

URA=4096  (* set axis unit ratio: resolver pulses/motion unit
CURC=75   (* set max continuous current
DIR=CCW   (* set motor direction for counterclockwise move

PROGRAM4  (* start program 4
EXP1      (* execute program 1
END       (* end program 4

PROGRAM1  (* start program 1
WAIT IO11 (* wait for enable input
RSF       (* reset faults
PSA=0     (* set axis position register to 0
MVL=10    (* set motion velocity to 10 units/sec
MAC=40    (* set motion acceleration rate to 40 units/sec2
MPA=12    (* set absolute move position to 24 units
RPA       (* run to absolute move position
END       (* end program 1 and exit editor
```

```
n program for the Target, Jan 1999,
nts to document application, e.g.,
and load, constraints on operation,
uses.

 1 to servo drive type
atio: resolver pulses/motion unit
ous current
ion for clockwise move

7
 1


 input
d axes' faults
PSA1=0     (* set axis position register to 0
MVL1=10    (* set motion velocity to 10 units/sec
MAC1=40    (* set motion acceleration rate to 40 units/sec2
MPA1=12    (* set absolute move position to 12 units
RPA1       (* run to absolute move position
END        (* end program 1 and exit editor
```

(* is the comment delimiter.

The Target is a multiaxis controller—note how axis numbers are included as part of the syntax for most registers and commands (e.g., PSA1, RPA1).

Spaces and blank lines are optional.

Figure 3.5: Example Application Programs for the IMC/IMJ (*left*) and Target (*right*)

Create an Application Program

With CCS for Windows open, complete the following steps:

1. Click **File**/**New** to open the following screen:

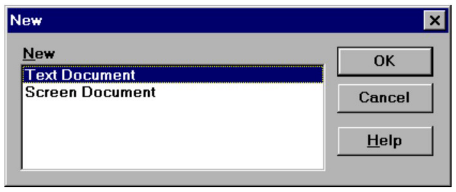
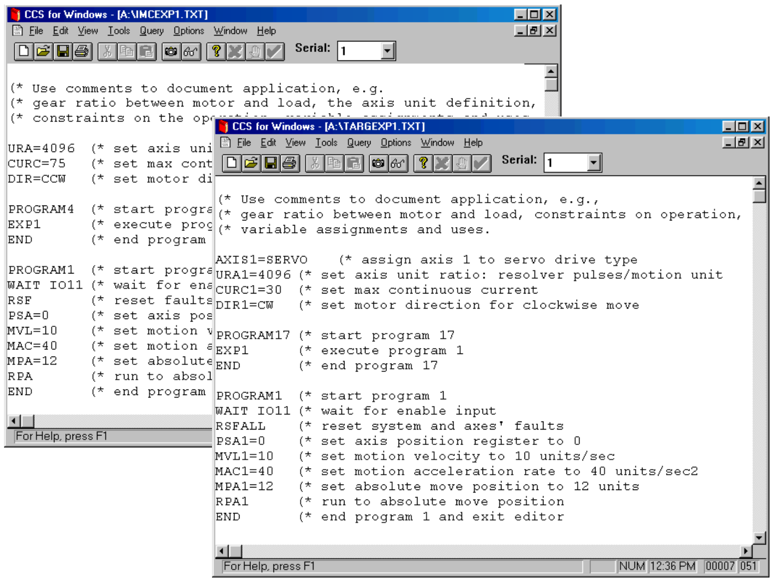


Figure 3.6: Choosing a New File Type

2. Click **Text Document**

3. Click **OK** to open the CCS ASCII file editor window.

Figure 3.7 shows simple example application programs for the IMC/IMJ and the Target. Both programs will execute a simple motion, i.e., set the axis position register to 0 and run a single axis 12 units in the forward direction.


CCS for Windows - [A:\IMCEXP1.TXT]
File Edit View Tools Query Options Window Help
Serial: 1
For Help, press F1
CCS for Windows - [A:\TARGEXP1.TXT]
File Edit View Tools Query Options Window Help
Serial: 1
For Help, press F1
NUM 12:36 PM 00007 051

```
(* Use comments to document application, e.g.
(* gear ratio between motor and load, the axis unit definition,
(* constraints on the op

URA=4096  (* set axis uni
CURC=75   (* set max cont
DIR=CCW   (* set motor di

PROGRAM4  (* start progra
EXP1      (* execute prog
END       (* end program

PROGRAM1  (* start progra
WAIT IO11 (* wait for ena
RSF       (* reset faults
PSA=0     (* set axis pos
MVL=10    (* set motion v
MAC=40    (* set motion a
MPA=12    (* set absolute
RPA       (* run to absol
END       (* end program
```

```
(* Use comments to document application, e.g.,
(* gear ratio between motor and load, constraints on operation,
(* variable assignments and uses.

AXIS1=SERVO    (* assign axis 1 to servo drive type
URA1=4096 (* set axis unit ratio: resolver pulses/motion unit
CURC1=30  (* set max continuous current
DIR1=CW   (* set motor direction for clockwise move

PROGRAM17 (* start program 17
EXP1      (* execute program 1
END       (* end program 17

PROGRAM1  (* start program 1
WAIT IO11 (* wait for enable input
RSFALL    (* reset system and axes' faults
PSA1=0    (* set axis position register to 0
MVL1=10   (* set motion velocity to 10 units/sec
MAC1=40   (* set motion acceleration rate to 40 units/sec2
MPA1=12   (* set absolute move position to 12 units
RPA1      (* run to absolute move position
END       (* end program 1 and exit editor
```

Figure 3.7: ASCII File Editor Windows Displaying Example Application Programs for the IMC/IMJ (*left*) and Target (*right*)

You may want to copy either of the application program example from figure 3.7—or create your own simple example that you can send to the controller and run later in this chapter as those sections are introduced. Click to place your cursor in the ASCII file editor window and use the following procedure to create an application program .txt file.

1. Type any (* delimited header text that you want to save in your .txt file, e.g.,

```
(* Example programs 1 and 4 for the IMC
```
<Enter> <Enter>

2. Type your system constants, e.g.,

   `URA=4096` <Tab> `(* set axis unit ratio` <Enter>

   Note: *For brevity's sake, the application program examples in this manual include only the minimum system constants required to make each example work. The application program that you design will require several other system constants. Turn to chapter* ***5*** *for the complete procedure for setting system constants.*

3. If you are using an IMC or IMJ, type `PROGRAM4` as your first program line. If you are using a Target, type `PROGRAM 17` as your first program line. <Tab>

4. Type a (* delimited comment for your first program line, e.g.,

   `PROGRAM4` <Tab> `(* start program 4` <Enter>

   Note: *Comments are optional but highly recommended for program documentation.*

5. Type your remaining program 4 or program 17 text, one command or register per line.

6. Type `END` <Enter> <Enter> to mark the end of your program 4 or 17.

7. Enter program 1 text, one command or register per line, e.g.,

   `PROGRAM1` <Tab> `(* start program 1` <Enter>

8. Type `END` <Enter> to mark the end of your program 1.

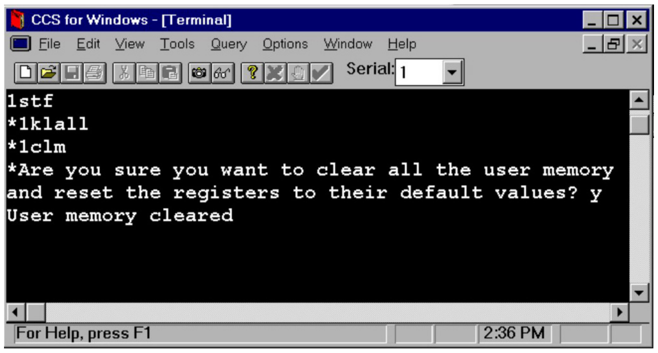
9. Save your application program as a .txt file.

When you have completed your application program, continue to the next section to learn how to send a .txt file to your DspMotion controller.

Note*: Turn to Chapter 5 to learn more about*
setting system constants
how programs, or tasks, interact
how to develop a complete application program.

Send An Application Program To Your DspMotion Controller

Before you send a program to the IMC, IMJ, or the Target, you must ensure that the controller is faulted and that it is not executing any programs. Use the following procedure to send any application program from your computer to the controller. Note that
(* delimited comments **are not sent** to the controller.


CCS for Windows - [Terminal]
File Edit View Tools Query Options Window Help
Serial: 1
For Help, press F1
2:36 PM

```
1stf
*1klall
*1clm
*Are you sure you want to clear all the user memory
and reset the registers to their default values? y
User memory cleared
```

Click Options/Controller Settings to set controller address. IMC, address must match DIP switch setting. Target address must be set to Target.

Figure 3.8: CCS Terminal Window

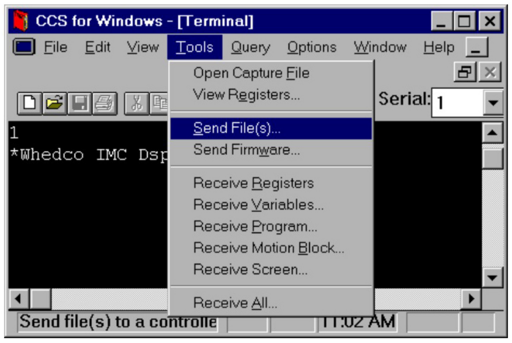
1. From the Terminal window type `STF` (i.e., for the IMC/IMJ) or `STFALL` (i.e., for the Target) <Enter>
2. Type `KLALL` <Enter>

   ***Note**: The order of execution of these commands is critical. If STF is executed after KLALL, then the Fault program will re-execute and you will not be able to send your program.*

3. Type `UPS=0` (UPS must be set to its default value of zero before the CLM command will work).
4. Type `CLM` <Enter>

   ***Note**: CLM clears your axis initialization settings! You'll have to reset them if you have not included them with the .txt file that you're about to send to the controller.*

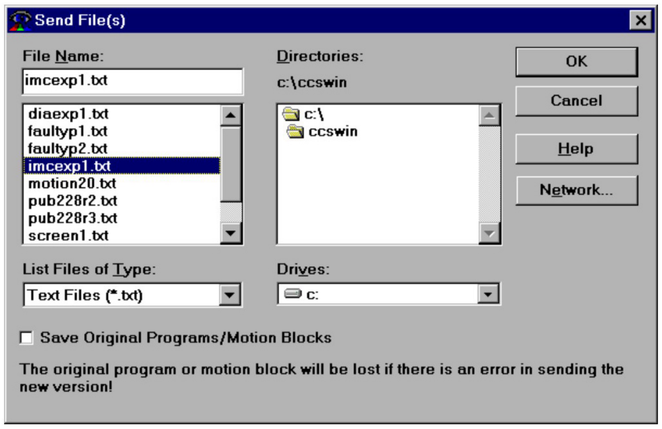
5. Click **Tools/Send Files**



```
1
*Whedco IMC Dsp
```

Figure 3.9: Selecting Tools/Send Files

6. Select the file you wish to send
7. Click **OK**
8. Wait for the file to transmit.



The original program or motion block will be lost if there is an error in sending the new version!

Figure 3.10: Selecting a File to Send

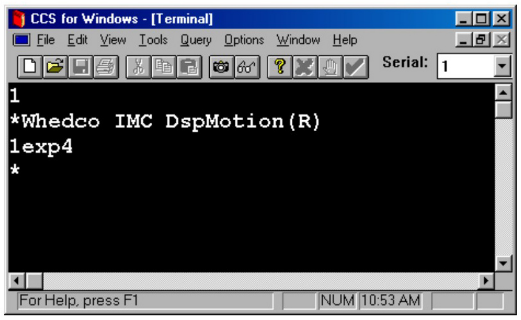
If no error occurs, continue to the next section and run your program. If an error occurs during the file send, turn to 3-9 for instructions on how to fix the problem.

Run An Application Program

You've enjoyed an error-free file send. Now it's time to test your application program on the DspMotion controller and make sure that it performs as expected.

Using the example application program from figure 3.7, we'll run the application program from the CCS Terminal window:

1. If you are connected to an IMC or IMJ, type `EXP4` <Enter> to execute your application program. If you are connected to a Target, Type `EXP17` <Enter>.


CCS for Windows - [Terminal]
File Edit View Tools Query Options Window Help
Serial: 1
For Help, press F1
NUM 10:53 AM

```
1
*Whedco IMC DspMotion(R)
1exp4
*
```

Figure 3.11: IMC Example Application Program Executed from CCS Terminal Window

2. Evaluate the system results.

Let's say that you want to change the absolute move position from 12 to 24. You can use the CCS ASCII file editor to change your application program.

Change An Application Program Using The ASCII File Editor

The ASCII file editor makes it easy to view your entire application program with comments and make changes to the text. In the following examples, we will change our absolute move position in the IMC program 1.

1. Click **File/Open**
2. Select the name of the application program's .txt file (this example uses the *imcexp1.txt* example from figure 3.7)

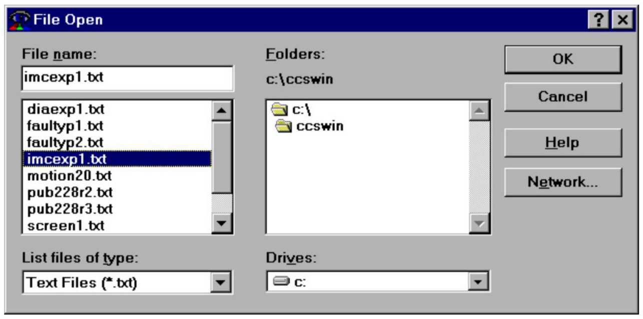
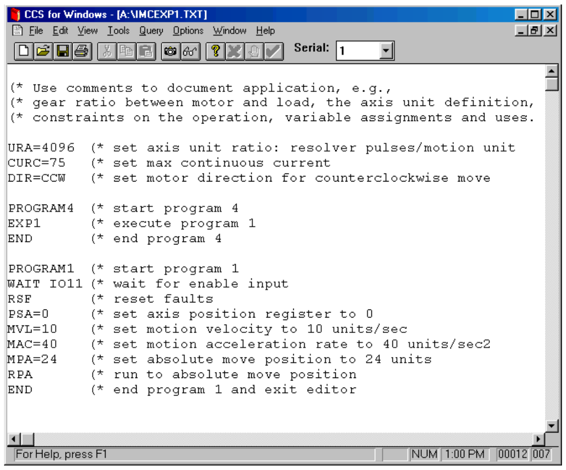


Figure 3.12: Opening a .txt File

3. Click **OK** to open the ASCII file editor and display the application program.
4. Edit your text file—just click and type!
5. Save your changes.
6. Click **File**/**Close** to exit the ASCII file editor.

When you have made your change, you must send your updated .txt file to the controller before you can run it to your new absolute move position. Repeat the file send procedure found on page 3-6.

MPA changed from 12 to 24 units.


CCS for Windows - [A:\IMCEXP1.TXT]
File Edit View Tools Query Options Window Help
Serial: 1
For Help, press F1
NUM 1:00 PM 00012 007

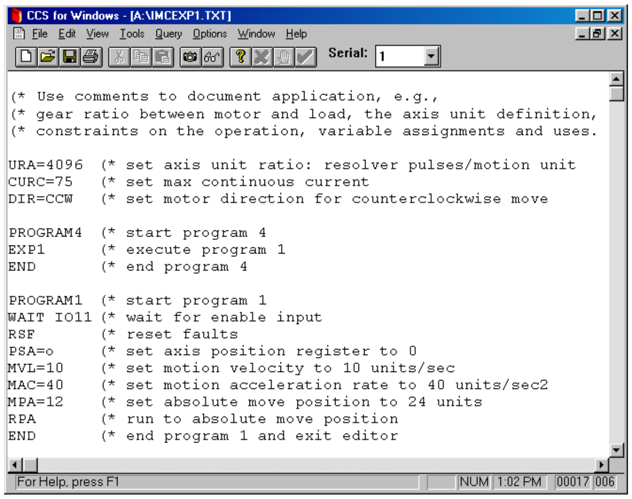
```
(* Use comments to document application, e.g.,
(* gear ratio between motor and load, the axis unit definition,
(* constraints on the operation, variable assignments and uses.

URA=4096  (* set axis unit ratio: resolver pulses/motion unit
CURC=75   (* set max continuous current
DIR=CCW   (* set motor direction for counterclockwise move

PROGRAM4  (* start program 4
EXP1      (* execute program 1
END       (* end program 4

PROGRAM1  (* start program 1
WAIT IO11 (* wait for enable input
RSF       (* reset faults
PSA=0     (* set axis position register to 0
MVL=10    (* set motion velocity to 10 units/sec
MAC=40    (* set motion acceleration rate to 40 units/sec2
MPA=24    (* set absolute move position to 24 units
RPA       (* run to absolute move position
END       (* end program 1 and exit editor
```

Figure 3.13: Editing a .txt File in the CCS ASCII File Editor

Fix An Error That Occurs During a File Send

You have already learned how to send a .txt file to your controller and seen how a successful *send* works. In the following example, the program author mistakenly set the axis position register to letter O instead of numeric 0. (For further information on errors, see Appendix D, *Command Fault and Status Messages*).

PSA set to alphabetic o instead of zero.


CCS for Windows - [A:\IMCEXP1.TXT]
File Edit View Tools Query Options Window Help
Serial: 1
For Help, press F1
NUM 1:02 PM 00017 006

```
(* Use comments to document application, e.g.,
(* gear ratio between motor and load, the axis unit definition,
(* constraints on the operation, variable assignments and uses.

URA=4096  (* set axis unit ratio: resolver pulses/motion unit
CURC=75   (* set max continuous current
DIR=CCW   (* set motor direction for counterclockwise move

PROGRAM4  (* start program 4
EXP1      (* execute program 1
END       (* end program 4

PROGRAM1  (* start program 1
WAIT IO11 (* wait for enable input
RSF       (* reset faults
PSA=o     (* set axis position register to 0
MVL=10    (* set motion velocity to 10 units/sec
MAC=40    (* set motion acceleration rate to 40 units/sec2
MPA=12    (* set absolute move position to 24 units
RPA       (* run to absolute move position
END       (* end program 1 and exit editor
```

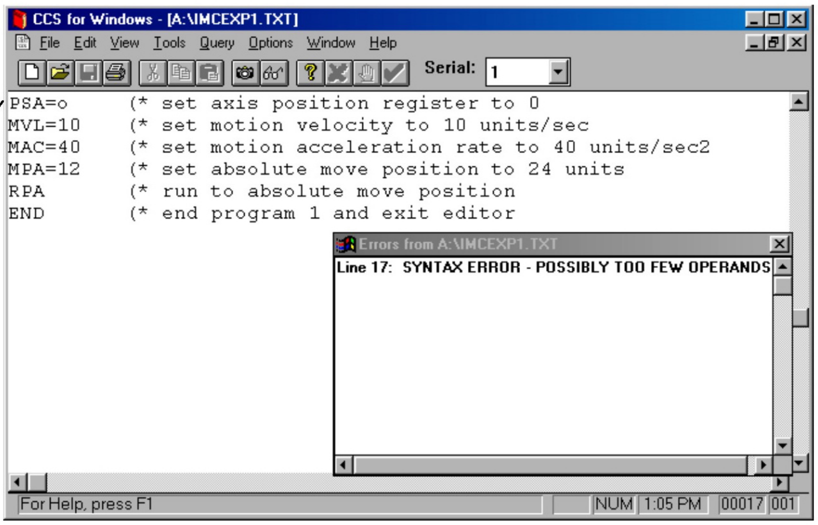
Figure 3.14: Example IMC Applications Program with Syntax Error that Will Cause File Send to Fail

Let's try to send the program to the controller:

1. Type `STF` (STFALL for the Target)
2. Type `KLALL`
3. Type `UPS=0` (UPS must be set to its default value of zero before the CLM command will work)
4. Type `CLM` (CLM will clear your axis initialization settings! You'll have to reset them if you have not included them in your .txt file.)
5. Click **Tools**/**Send Files**
6. Click on file name (in this example, IMCEXP1.TXT)
7. Click **OK**

The controller detects the error and does not accept the file. CCS takes you directly into the ***ASCII file editor*** (see figure 3.15) and opens the original .txt application program file on the line containing the first error (keep in mind that a program could have more than one bug).

ASCII file editor opens with cursor at line 17, which contains the error.


CCS for Windows - [A:\IMCEXP1.TXT]
File Edit View Tools Query Options Window Help
Serial: 1
Errors from A:\IMCEXP1.TXT
For Help, press F1
NUM 1:05 PM 00017 001

```
PSA=o      (* set axis position register to 0
MVL=10     (* set motion velocity to 10 units/sec
MAC=40     (* set motion acceleration rate to 40 units/sec2
MPA=12     (* set absolute move position to 24 units
RPA        (* run to absolute move position
END        (* end program 1 and exit editor
```

```
Line 17:  SYNTAX ERROR - POSSIBLY TOO FEW OPERANDS
```

Figure 3.15: When an Error Occurs During the Tools/Send Files Process

1. Correct the bug (just click and type)
2. Click **File**/**Save**
3. Click **File**/**Close** to exit the ASCII file editor and return to the Terminal window

Send the updated file to the controller:

1. Type `STF` (i.e., for IMC/IMJ) or `STFALL` (i.e., for Target)
2. Type `KLALL`
3. Type `UPS=0` (UPS must be set to its default value of zero before the CLM command will work)
4. Type `CLM` (CLM will clear your axis initialization settings! You'll have to reset them if you have not included them in your .txt file.)
5. Click **Tools**/**Send Files**
6. Click on your application program's .txt file name (IMCEXP1.TXT in this example)
7. Click **OK**

The file send completes successfully!

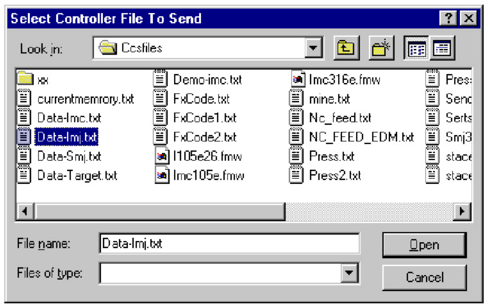
Create End User Application

CCS version 6.0 and later features a pack-and-go utility called DspComm, which allows you to create application program executables that function on end user systems where CCS is not installed. DspComm is an interface that allows end users to communicate with their DspMotion controller (e.g., send application program files, receive the contents of the controller memory). DspComm is freely distributable to users of Motion controllers and may be used on any PC running Windows 95, 98, or NT.

To package and distribute your end user's application program with the free DspComm utility, The ***Create End User Application*** selection under the Tools menu in CCS allows you to .

Use the following procedure to package and distribute your end user's application program with DspComm:

1. From CCS, select **Tools** > **Create End User Application Program**
2. Locate and select the text file containing your application program on the *Select Controller File to Send* screen



Select Controller File To Send

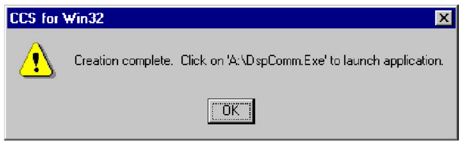
Look in: Ccsfiles

File name: Data-lmj.txt

Files of type:

Figure 3.16: *CCS Select Controller File to Send* Screen

3. Click **Open**
4. Select Destination Path (you must open a directory that is different from the one that contains the "Send" file you selected in step 2 above.)
5. Click **Open**
6. Click **OK**. The application program file and DspComm are now located in your selected destination and are ready to be sent to your end user. The end user needs only to load the files you send, double click on DspComm.exe to launch the application, and send / receive files to and from the Motion controller.



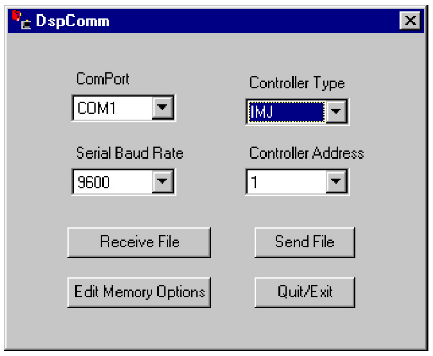
CCS for Win32

Creation complete. Click on 'A:\DspComm.Exe' to launch application.

Figure 3.17: CCS Confirms End User Application Program has been Successfully Packed with DspComm

End User’s Memory Options in DspComm

When you distribute DspComm to your end users, you give them the ability to send and receive application program files to and from their DspMotion controllers. DspComm requires only that the user select a COM port, serial baud rate, controller type, and controller address from the pick lists provided on the main DspComm screen. Once the correct settings have been selected, files can be sent and received over the serial port.

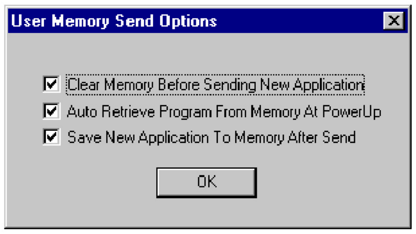


ComPort: COM1
Controller Type: IMJ
Serial Baud Rate: 9600
Controller Address: 1

Figure 3.18: End User Screen in DspComm

> *Note: DspComm requires a serial connection to a Motion controller. Those who have both CCS and DspComm running on the same system are reminded to close CCS before attempting to communicate with the controller via DspComm.*

User memory safeguards are enabled by default in DspComm to protect application programs. For maximum system flexibility, however, the *Edit Memory Options* button in DspComm will permit the end user to disable those memory safeguards prior to a file send. The user memory options are:



☑ Clear Memory Before Sending New Application
☑ Auto Retrieve Program From Memory At PowerUp
☑ Save New Application To Memory After Send

Figure 3.19: End User Memory Options in DspComm

Clear Memory before Sending New Application: equivalent to issuing the CLM command, which clears user memory.

Auto Retrieve Program from Memory at PowerUp: equivalent to issuing the AUTORET command, which enables the controller to automatically retrieve user memory on controller power-up

Save New Application to Memory after Send: equivalent to issuing the SAVE command, which saves user memory.

When the user clicks DspComm’s *Send File* button, he or she is warned that any currently executing programs will halt and controller memory will be cleared. The user also gets a reminder to upload their controller memory to a text file prior to sending a new file to the controller. This file upload stores the old program.

Chapter 4 Application Programming Resources

In This Chapter

DspMotion system resources let you manage application programs from the simple to the complex.

Figure 4.1: DspMotion System Resources

	IMC	IMJ	Target
Motion blocks	100	100	400
Flow control			
Labels per program	999	999	999
Nested GOSUBS per program	32	32	32
Variables			
Boolean variables	256	256	256
Floating point variables(a, b)	14,336	2,048	2,048 standard 131,072 optional(c)
Integer variables(a)	28,672	4,096	4,096 standard 262,144 optional(c)
String variables	144	144	144 standard 272 optional(c)
Timing devices			
Countdown timers	8	8	16
Counters/pulse timers	n/a	n/a	4 per digital I/O module; 32 maximum
Real-time clock	1	n/a	1

(a) Integer and floating point variable memory space is shared; numbers are maximum for each but not for both concurrently. Floating point variables require twice as much memory as integer variables. Thus, for example, in the IMC case, if 2,048 floating point variables are used, 24,556 integer variables are possible.

(b) Floating point variables use a 32-bit mantissa and are precise to 9 decimal digits.

(c) Optional variables require 1 Megabyte optional variable memory.

This chapter will educate you about these resources in the Generation D RTOS, along with math functions, set point outputs, and the OIP; so that when you start to develop your own application, you can take full advantage of the computing power in the DspMotion products.

Program Maps

DspMotion controllers handle multiple multitasking programs:

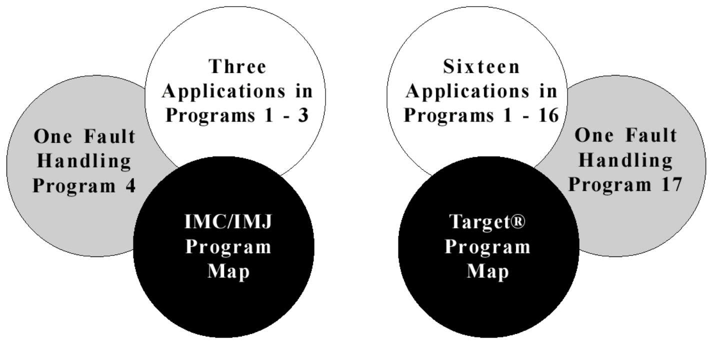

Three Applications in Programs 1 - 3
One Fault Handling Program 4
IMC/IMJ Program Map
Sixteen Applications in Programs 1 - 16
One Fault Handling Program 17
Target® Program Map

Figure 4.2: Program Maps for the IMC/IMJ and the Target

Motion Blocks

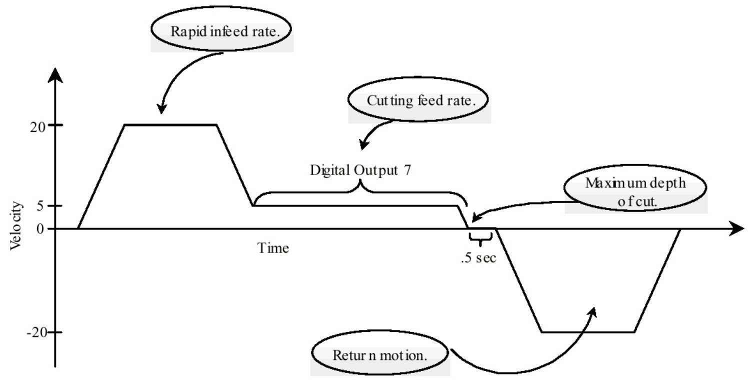
Motion blocks allow you to define motions that can be called and used by any program or executed in *immediate mode* from an external control device. You can create, send, receive, and edit motion blocks in the same way that you do programs, except motion blocks begin with the **MOTION** command instead of the PROGRAM command. The IMC and IMJ support up to 100 motion blocks. The Target ARS supports up to 400 motion blocks.

Figure 4.3

Rules of Motion Block Execution			
1. Motion blocks complete executing one line of code before proceeding to the next line of code.	2. You can concurrently execute only one motion block per axis with the executing program(s).	3. Once a motion block is executed, it overrides the currently executing motion block or motion.	4. No labels allowed!

Example of a Motion Block for the IMC

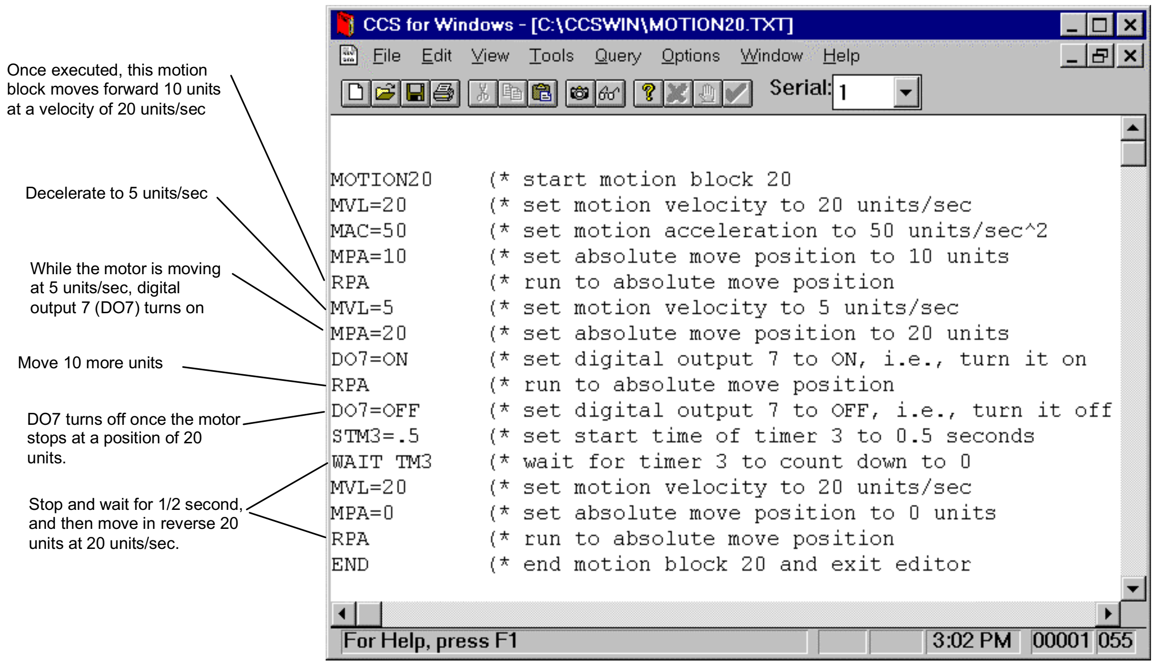
Motion blocks allow the user to create complex motions such as blended moves without a series of conditional and wait statements. For example, for a spindle infeed on a machine tool, you may want to define a move like the one shown in the following diagram:



In figure 4.4 the motor is assumed to be at 0 units (PSA=0) before the motion block is executed.

Figure 4.4: A Complex, Blended Move Defined by a Motion Block

Use the ASCII file editor to create a motion block that will execute this motion:



CCS for Windows - [C:\CCSWIN\MOTION20.TXT]
File Edit View Tools Query Options Window Help
Serial: 1
For Help, press F1
3:02 PM
00001 055

Once executed, this motion block moves forward 10 units at a velocity of 20 units/sec

Decelerate to 5 units/sec

While the motor is moving at 5 units/sec, digital output 7 (DO7) turns on

Move 10 more units

DO7 turns off once the motor stops at a position of 20 units.

Stop and wait for 1/2 second, and then move in reverse 20 units at 20 units/sec.

```
MOTION20    (* start motion block 20
MVL=20      (* set motion velocity to 20 units/sec
MAC=50      (* set motion acceleration to 50 units/sec^2
MPA=10      (* set absolute move position to 10 units
RPA         (* run to absolute move position
MVL=5       (* set motion velocity to 5 units/sec
MPA=20      (* set absolute move position to 20 units
DO7=ON      (* set digital output 7 to ON, i.e., turn it on
RPA         (* run to absolute move position
DO7=OFF     (* set digital output 7 to OFF, i.e., turn it off
STM3=.5     (* set start time of timer 3 to 0.5 seconds
WAIT TM3    (* wait for timer 3 to count down to 0
MVL=20      (* set motion velocity to 20 units/sec
MPA=0       (* set absolute move position to 0 units
RPA         (* run to absolute move position
END         (* end motion block 20 and exit editor
```

Assigning Target Axes to Motion Blocks (MBA)

The Target ARS gives you eight axes to use in an application program. When you create motion blocks for your Target application program, you must designate which axis each motion block will use, e.g.,

```
Motion20    (* start motion block 20
MBA14       (* assign axes 1 and 4 to motion block 20
```

You could assign all motion blocks to a single axis—keep in mind that only one motion block ***per axis*** can execute at one time.

Flow Control

Labels and Subroutines (LABEL, GOTO, GOSUB, IF…GOTO, IF…GOSUB)

A ***label*** is an integer number from 1 to 999 that immediately precedes a program statement and serves as a reference point. Assign labels to delineate program sections or to identify starting points for GOSUB and GOTO routines.

A ***subroutine*** is a section of a program containing an encapsulated routine that the GOSUB command can access multiple times from any point within the program. A program may contain up to 32 nested GOSUBS, (a nested GOSUB is simply a subroutine within a subroutine).

Use the commands GOTO, GOSUB, IF…GOTO, IF…GOSUB, RETURN, RSTSTK, and POP to get to and from the subroutines in your programs.

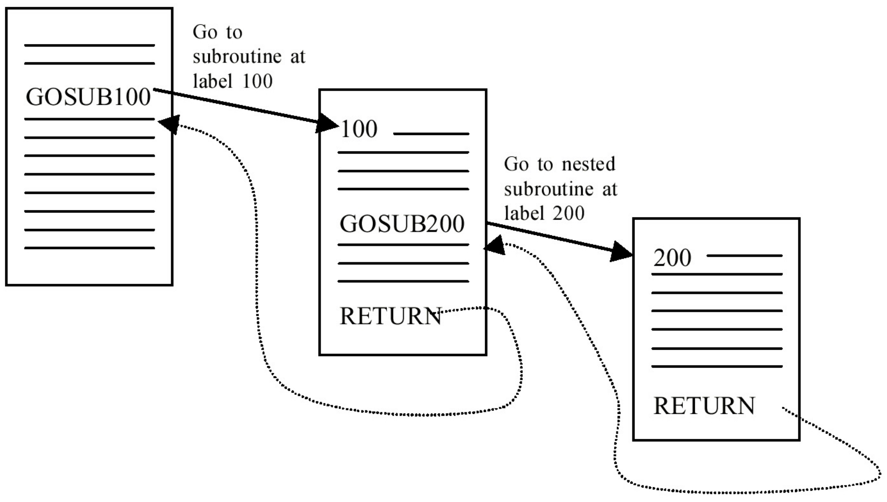

Go to subroutine at label 100
GOSUB100
100
GOSUB200
RETURN
Go to nested subroutine at label 200
200
RETURN

Figure 4.6: Program 1 with GOSUBS

Flow Control Commands

Figure 4.7: Flow Control Commands

EXP	Runs program *n*. If program *n* is currently running, then EXP*n* has no effect on program flow.
GOSUB	Goes to a label that functions as a subroutine.
GOTO	*Goes to* any labeled program statement in the currently executing program: `GOTO10           (*jump to the program line with label 10`
IF…GOSUB	If *condition* is true then GOSUB label.
IF…GOTO	If *condition* is true then GOTO label.
IF…THEN	If *condition* is true then execute the next line of the program; otherwise, skip the next line.

KLP	Kills program *n*. If program *n* is not running, then KLP*n* has no effect on program flow. For example:
	`IF DI4 THEN              (* if input 4 then`
	`EXP2                     (* start program 2`
	`IF NOT DI5 THEN          (* if not input 5 then`
	`KLP3                     (* kill program 3`

POP	The POP command retrieves and discards the top of the gosub stack. POP lets you leave a subroutine without executing a RETURN. For example:
	`...        (* subroutine entered with a GOSUB`
	`WAIT IP1 WHEN DI1.12 GOTO 700`
	`           (* wait for axis 1 in position. When not`
	`           (* emergency stop input, leave`
	`           (* subroutine, and execute E-stop code`
	`...`
	`RETURN     (* normal subroutine return`
	`700 POP    (* retrieve and discard subroutine return address`
	`...        (* E-stop code`

REPEAT	Causes a motion block to repeat from the beginning.

RETURN	Returns to the statement in program immediately following the GOSUB, e.g.,
	`GOSUB 100  (* save the program counter on the gosub stack,`
	`           (* then load the program`
	`           (* counter with the line at label 100`
	`...        (* another line of program`
	`100        (* subroutine code`
	`...`
	`RETURN     (* return to another line`

RSTSTK	Empties the GOSUB stack.
STVB*n* GOTO	Sets Boolean variable *n* and, if VB*n* was not already set, goes to the label specified.
WAIT	Causes the program or motion block to wait until the specified condition is true before advancing to the next line of code.

WAIT…WHEN… GOTO	WAIT ... WHEN ... GOTO *label* waits for the first expression (…) to become true, or when the second expression becomes true, it goes to the label:
	`WAIT IP1 WHEN DI1.1 GOTO 10`
	`           (* wait for axis 1 to be in`
	`           (* position, or when input 1, goto 10.`

Math Functions

The Generation D RTOS supports full floating point math and operators for complex mathematical and logical operations:

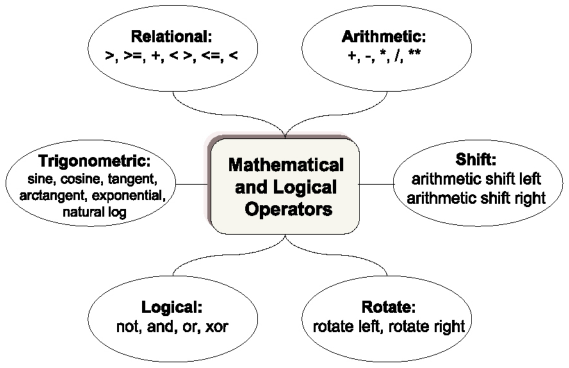


Figure 4.8: Operators in the Generation D RTOS

Multifunction, single-line math operations use standard infix notation to simplify program readability and flow, e.g.,

Mathematical equation:	VF1=SQR(VF2**2.+VF3**2.)
Calculation:	result stored in floating point variable 1 equals the square root of the sum of the squares of the floating point variables 2 and 3.

Data Typing

The Generation D RTOS enforces data typing in register and variable assignments and in all math operations, including comparisons. Data typing rules are listed in the following table:

Figure 4.9

Data Type	**Load with**
Floating point register	real or integer number
Floating point variable	
Integer register	integer number
Integer variable	integer number, Boolean variable, or register

Data typing is enforced in all register-to-register, register-to-variable, variable-to-register, and variable-to-variable assignments.

Note: *Boolean variables are treated like integer variables in math operations.*

Variables

Types of Variables (VBn, VIn, VFn, VSn)

In some of the commands that you use, the parameter (e.g., *p1*, *p2*, etc.) that is part of the command's syntax can be a variable expression. You can also set most of the registers to a variable expression.

Variables can also be used in mathematical operations. DspMotion controllers support the variable types shown in figure 4.10.

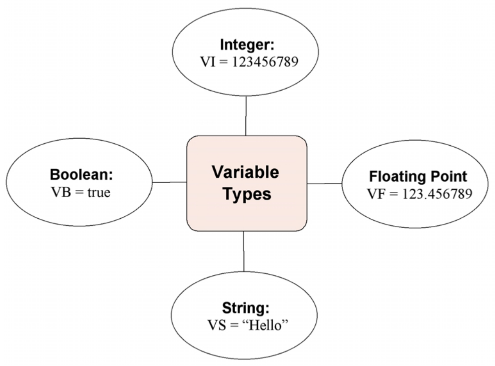


Figure 4.10: Variable Types in the Generation D RTOS

Boolean Variables (VB)

Boolean variables (VB*n*) can have a value of 0 or 1 and are used mainly in conditional statements such as IF...GOTO and WAIT. They can also be used to change the value of Boolean registers (e.g., GRE, CIE, POE). Boolean variables are treated like integer variables in math expressions.

Floating Point Variables (VF, VFA, VFEA)

Floating point variables, **VF***n*, can store any floating point value between 1.5 x 10^{-39} (absolute value) to 1.7 x 10^{38} (absolute value) with up to nine digits precision. Use floating point variables in expressions and to store parameters. Load floating point variables with either real or integer numbers.

Integer Variables (VI)

Integer variables, **VI***n*, can store any integer value between -2,147,483,648 and 2,147,483,647. They are used mainly in expressions and to store parameters. Integer variables are as precise as floating point variables and can represent fractional values with appropriate scaling factors. Load integer variables with integers, Boolean variables, or registers.

String Variables (VS)

String variables, **VS***n*, can be loaded with a message up to 127 characters long. String variables are used in I/O commands (e.g., GET, IN, and OUT) and in I/O registers that store information for display screens (e.g., SCRL and SCRD).

For example, you could use the OUT command to send a message stored in string variable 1 to the serial port or user serial port:

```
KLALL             (* kill any executing programs
VS1="This is a test.$N"
OUT VS1           (* output This is a test to serial port
```

You could also store commands within string variables and then use the EXVS command to execute them:

```
VS1= "MPA=10"     (* set string variable 1
EXVS1             (* execute command stored in string var. 1
```

Variable Pointers

Integer variables can point to other variables, allowing you to construct many different kinds of data structures including the following:

- Linear array
- Push down stack
- Circular buffer.

A pointer contains the number of the variable to which you want to point. If you want to have a pointer access floating point variable 53, you can set any integer variable, such as integer variable 10, to 53. For example:

```
VI10 = 53         (* load pointer
VF100 = VFVI10    (* load VF100 with value of floating point
                  (* var. pointed to by VI10 [i.e., VF53]
```

is equivalent to: `VF100 = VF53`.

You can also use pointers to shorten programs. For example, you can send to the display a long list of characters whose ASCII values are stored in integer variables. Suppose you have ASCII codes stored in integer variables 100 through 200. You *could* send them to a OIP or display device using the PUT command one hundred times:

```
PUT CHR(VI100)
PUT CHR(VI101)
...
PUT CHR(VI200)
```

Or you could make the process quicker and far less tedious with variable pointers:

```
VI1=100                 (* load the pointer 100
1  PUT CHR(VIVI1)       (* send ASCII characters stored in
                        (* VIVI1 to display
VI1=VI1+1               (* increment VI1 by 1
IF VI1<=200 GOTO 1      (* continue to increment by 1 if
                        (* VI1 <= 200
```

When VI1 is less than or equal to 200, the program loops, sending all ASCII codes stored in variables 100 through 200 in the process. When VI1 is greater than 200, it fails the check and goes to the next program line.

Timing Devices

Countdown Timers (STM, TM)

The IMC and IMJ have 8 countdown timers; the Target ARS has 16 countdown timers. Use the **STM*n*** = xx.xxx (i.e., xx.xxx is a time in seconds) command to set these timers. Once set, a timer counts from the starting value down to zero. The timer automatically resets to the initial value and continues counting each time it reaches zero.

The timer flag, **TM*n***, is set each time the timer reaches zero and reset each time the flag is read. You can use TM*n* in conjunction with the WAIT command for conditional program flow. For example:

```
STM10 = 0.333     (* start timer 10 with a period of 333 ms
WAIT TM10         (* wait until timer 10 reaches zero
```

Counters/Pulse Timers (CTR, TMI, TMP)

In Target systems, digital inputs 1 through 4 on a Digital I/O Module can be used as counters and/or pulse timers. Counters tally inputs to the system from the first four inputs of a Digital I/O Module. Use the CTR register to reset a counter to zero or to query a counter's value:

```
CTR1.3=0    (* set counter 3 of digital module 1 to zero
```

TMI and TMP are read-only registers. Query TMI for the time between two successive activations of a digital input. Query TMP for the time during which a digital input remains active:

```
TMI5.3?     (* report interval timer 3 of digital module
            (* five
TMP5.4?     (* report pulse timer 4 of digital module five
```

Note: *Counters and pulse timers are not available on the IMC.*

Real-time Clock (TIME, DAY, DATE, MONTH)

The IMC and the Target ARS each have a clock that you can set and query with the TIME register:

```
TIME="20:40:15"   (* set time to 8:40 p.m., 15 seconds
TIME?             (* report time
```

***Note**: The IMJ does not feature a real-time clock.*

Set Point Outputs

Set point outputs are position-based outputs that turn on automatically for a specified position range. Set point outputs are defined with a beginning and an end and thus are direction-sensitive. Set point outputs provide high-speed response, turning on within 50 microseconds of reaching the beginning position. The Target contains one dedicated set point output per axis. The IMC has six set point outputs, A through F, that are assigned to the following digital outputs:

A = DO11	C = DO7	E = DO9
B = DO12	D = DO8	F = DO10

***Note**: Set point outputs are not available in the IMJ*

OIP (Optional)

The Generation D RTOS includes built-in utilities to support an optional Operator Interface Panel (OIP). Unlike third-party, human-machine interfaces, you do not have to program the OIP separately. Instead, you can control the OIP from within your DspMotion application program. The OIP is an ASCII I/O device and is ideal for replacing discrete operators, adding machine diagnostics, and setting up your system.

The Liquid Crystal Display (LCD)

The OIP includes a 4-line by 40-character, back-lit LCD. Several options exist for creating and using display screens in a DspMotion control system.

Creating Standard Screens

You can create up to 50 standard screens for use in programs in DspMotion controllers. Each line in standard screens may have one data field anywhere in the line. Choose one of the following two methods to create standard screens.

Method 1: Use ScreenView™ in CCS for Windows. CCS for Windows contains a utility called ScreenView that allows you to configure the display in an easy-to-use, graphical format. To create a new screen using ScreenView, follow these simple steps:

1. Click **File/New/Screen**
2. Click **OK** to create a new screen in ScreenView
3. Enter text directly on the screen graphic (see figure 4.12)

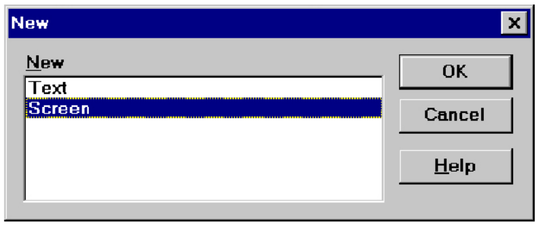


Figure 4.11: Creating a New Screen

Click to position the cursor and type what you want to appear on the screen, e.g., *Axis Position:*

Data fields 1 through 4 are mapped to their corresponding lines 1 through 4 on the display

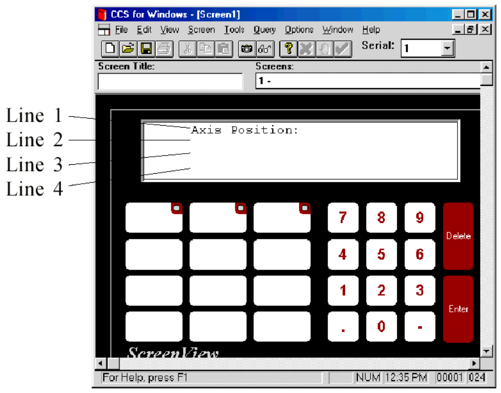


Figure 4.12: Using ScreenView

4. Click **Edit**/**Data Field 1** to enter a data field on line one of the screen. Then complete steps *a* through *e* in figure 4.13.

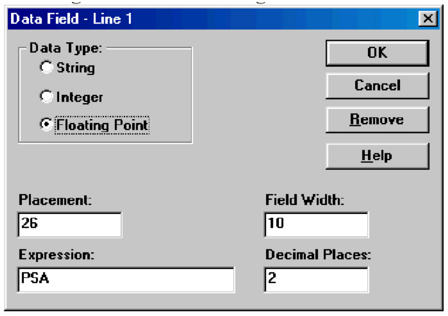
   a. Click to select a **Data Type**

   b. Enter a **Placement** value to determine a horizontal location for the data field

   c. Enter an **Expression**, or register to be displayed in the data field

   d. Enter a **Field width** (for integer and floating point variables only) to specify the field width in characters

   e. Enter the number of **Decimal Places** (for floating point variables only).


Data Field - Line 1

Data Type: String [✓ Floating Point] Integer

Placement: 26

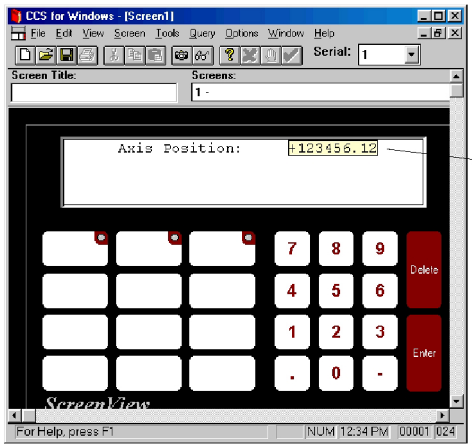
Field Width: 10

Expression: PSA

Decimal Places: 2

Figure 4.13: Entering Data Field 1

5. Click **OK** to see the new data field on the screen graphic (figure 4.14)



This is a floating point register resolved to 2 decimal places with a maximum field width of 10 characters

Figure 4.14: Using ScreenView

CCS lets you save any screen in an ASCII file with an .oip extension (e.g., *Screen1.oip*). You can then send the .oip file to the DspMotion controller using the same **Tools/File Send** procedure that you would use to send .txt files (see Chapter 3).

Method 2: Use the Generation D RTOS Screen Registers to Create Screens. You may elect to create screens using the screen registers, ***SCRD, SCRL*** and ***SCRP,*** with the ASCII file editor provided in CCS for Windows. These registers let you directly specify string data, strings, and cursor position for a specific screen line within a screen file. These registers can be used in your application program.

Using Standard Screens in Programs

In a program, you can use two screen control functions to output screens and update data fields as shown in figure 4.15:

Figure 4.15: Screen Control Functions

OUTS	Output all string data for a given screen to the OIP.
UPS	Automatically update the data fields every 250 milliseconds for the screen assigned to this register (e.g., `UPS=1` automatically updates the data field in screen 1 every 250 milliseconds)

Creating Custom Screens

To create more elaborate display screens, such as those with multiple data fields per line, use the character control commands provided in the Generation D RTOS. Using these commands, you can control the output to the display on a character-by-character basis. These commands will be an integral part of the programs contained in your DspMotion controller and stored as a part of the .txt files that you create with the ASCII file editor. The character control commands are shown in figure 4.16:

Figure 4.16: Character Control Commands[1]

BS	Backspaces the cursor on the display.
CLL	Clears current line and places cursor at the beginning of the line on the display.
CLS	Clears display and places the cursor at the home position.
CR	Places cursor at the beginning of the next line down on the display.
CRH	Places the cursor at the home position.
CRM	Remembers current cursor position.
CRP	Places cursor on line *p1*, column *p2* of the display.
CRR	Places cursor at the CRM position.
OUT **OUTW**	Sends data to the display. Also used to send ASCII[1] codes to support the following functions: *line feed*, *carriage return*, *cursor up*, *cursor down*, *cursor left*, *cursor right*, *all function keys enabled*, *all function keys disabled*. This command can also be used to output string data from individual screen lines from screen files in the form `OUT SCRL1.1` (output string data from line 1 of screen 1).
PUT **PUTW**	Puts one character to the display.
Note: [1]The Whedco OIP can also use ASCII codes for each of the character control commands. These ASCII codes can be concatenated following the OUT and OUTW commands. Some programmers find this ability useful because it allows the user to construct a complete display function from a single line of code. ASCII codes for the character control commands are provided where applicable in Appendix A.	

Using Custom Screens in Programs

Program loops must be written to update display strings and data fields created using the character control commands. The CCS utility template, *Solve PID Algorithm*, contains an example of a display loop update routine (see Appendix H).

The Keypad

The OIP includes a membrane keypad with 12 function keys as well as standard numeric keys. Each key, when ***pressed***, outputs a distinct string of ASCII characters[2]. Each key, when ***released***, outputs a different string of ASCII characters[2]. This allows these function keys to replace momentary push buttons for applications like jogging the motor. The ASCII Character Map is shown in figure 4.17.

Figure 4.17: OIP ASCII Character Map of Key Outputs[2]

Key Action	Function Keys			Numeric Keys			Data Entry Keys
	A	**B**	**C**	**7**	**8**	**9**	**Delete**
press:	*65*	*66*	*67*	*55*	*56*	*57*	
release:	*97*	*98*	*99*	*none*	*none*	*none*	
	D	**E**	**F**	**4**	**5**	**6**	
press:	*68*	*69*	*70*	*52*	*53*	*54*	*127*
release:	*100*	*101*	*102*	*none*	*none*	*none*	*none*
	G	**H**	**I**	**1**	**2**	**3**	**Enter**
press:	*71*	*72*	*73*	*49*	*50*	*51*	
release:	*103*	*104*	*105*	*none*	*none*	*none*	
	J	**K**	**L**	**.**	**0**	**-**	
press:	*74*	*75*	*76*	*46*	*48*	*45*	*13*
release:	*106*	*107*	*108*	*none*	*none*	*none*	*none*

Notes: [2] Be aware that the keys on the Whedco OIP actually preface the ASCII characters listed in the Character Map above with WKY command. This command automatically puts the ASCII character above into the DspMotion® controller's key buffer and thus does not require the user to parse the WKY command.

The DspMotion controller receives these ASCII characters as inputs over its serial port. Use these inputs to cause actions and direct program flow.

Using Key Input in Programs

Using the key control commands in figure 4.18, write program loops to monitor the state of the keys. Use the flow control commands (*see figure 4.7*) to direct program flow as a function of the key input state:

Figure 4.18: Key Control Commands

EKB	Empties the key buffer.
FUNCTION	Allows the user to create a map between function keys and labels within a program. When your system encounters this command, it will divert program flow to the label mapped to the function key if the function key has been pressed.
GET **GETW**	Gets one character from the key buffer and loads it into the specified *p1* variable register.
IN **INW**	Inputs register value from the key buffer.
KY **WKY**	Puts one character into the key buffer.
KYA	Sets output for function keys 1 12 (i.e., keys labeled A L) so that single (key-pressed) or double (key-pressed/key-released) codes are put into the key buffer.

Labeling the Function Keys

CCS for Windows installs with a Microsoft Excel™ template that makes it easy to create a customized function key legend insert to place behind the keypad membrane on your OIP. To open the template, click **Start**/**CCS for Windows**/**OIP Legend**.

Follow the instructions included in the Excel file to create and insert your function key label.

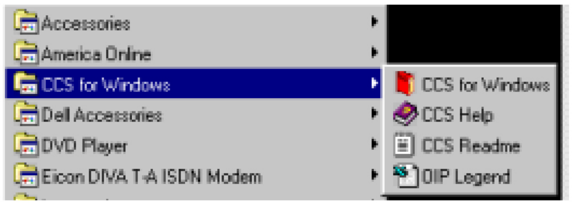


Figure 4.19: Opening the OIP Excel Template File

LEDs

The OIP has three green LEDs (i.e., LED 1, LED 2, and LED 3) located respectively on function keys A, B, and C. These LEDs can be turned on and off under program control and are useful to indicate functions such as *Active Mode* status.

Using LEDs in Programs

Use the **LED** register to set and reset LEDs at any time in a program—just set the desired LED to the appropriate state, for example:

```
LED1=1 (LED1=on)
LED1=0 (LED1=off).
```

You can also use Boolean variables to change the state of the LEDs by setting the LED equal to one of the Boolean variables, for example:

```
VB1=1       (* sets Boolean variable equal to 1/on
LED1=VB1    (* sets state of LED1 equal to state of
            (* Boolean variable 1
```

Setting an LED equal to a Boolean variable allows the state of the LED to be monitored in a program.

Chapter 5 Developing an Application Program

In This Chapter

- ❑ The structure of the Generation D Real-time Operating System (RTOS)
- ❑ What is multitasking?
- ❑ How to develop a complete application program, including the following steps:

  1. Set system constants
  2. Assess task interaction
  3. Structure a fault handling program
  4. Structure program 1 and additional tasks
  5. Manage your completed application program.

Structure of the Generation D RTOS

The Generation D RTOS allows you to create a control system for complex motion applications with real-time machine control and human-machine interface functions. The Generation D RTOS is multitasking and has global resources (shown in figure 5.1) that are shared by all tasks.

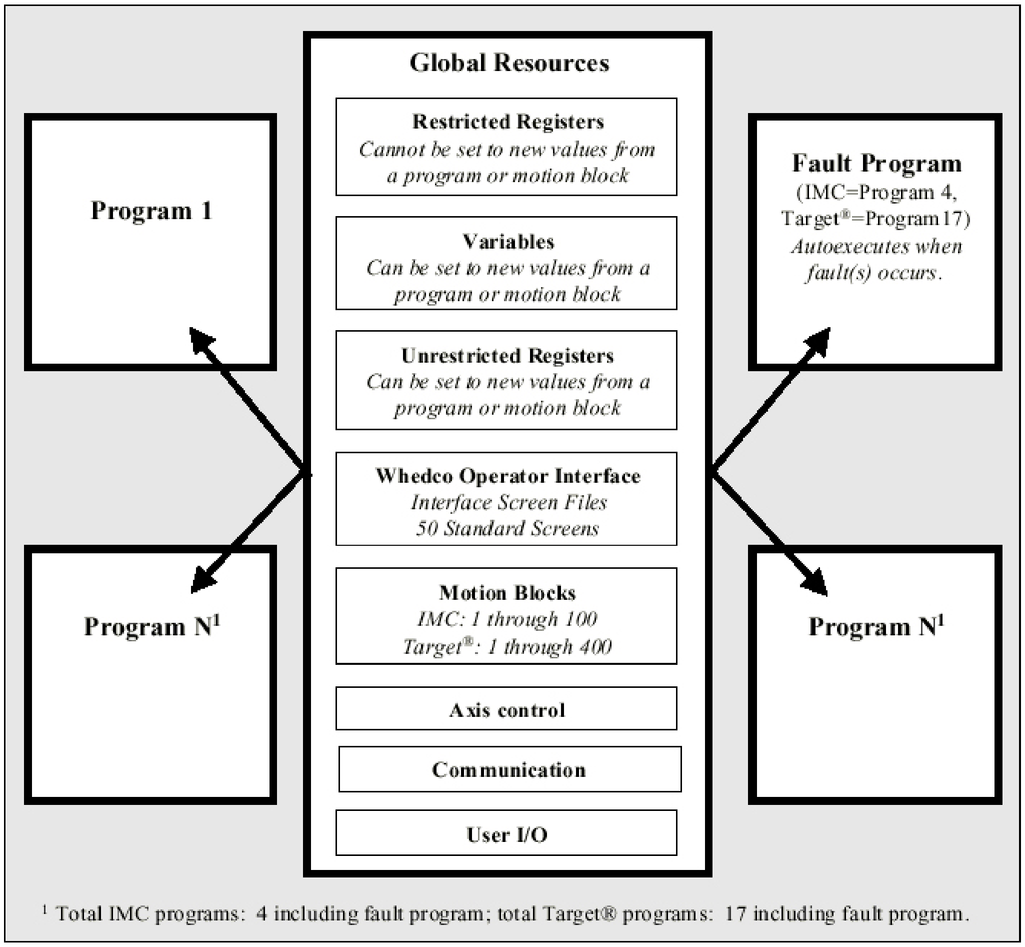

Global Resources
Restricted Registers
Cannot be set to new values from a program or motion block
Variables
Can be set to new values from a program or motion block
Unrestricted Registers
Can be set to new values from a program or motion block
Whedco Operator Interface
Interface Screen Files
50 Standard Screens
Motion Blocks
IMC: 1 through 100
Target®: 1 through 400
Axis control
Communication
User I/O
Program 1
Program N[1]
Fault Program
(IMC=Program 4, Target®=Program 17)
Autoexecutes when fault(s) occurs.
Program N[1]
[1] Total IMC programs: 4 including fault program; total Target® programs: 17 including fault program.

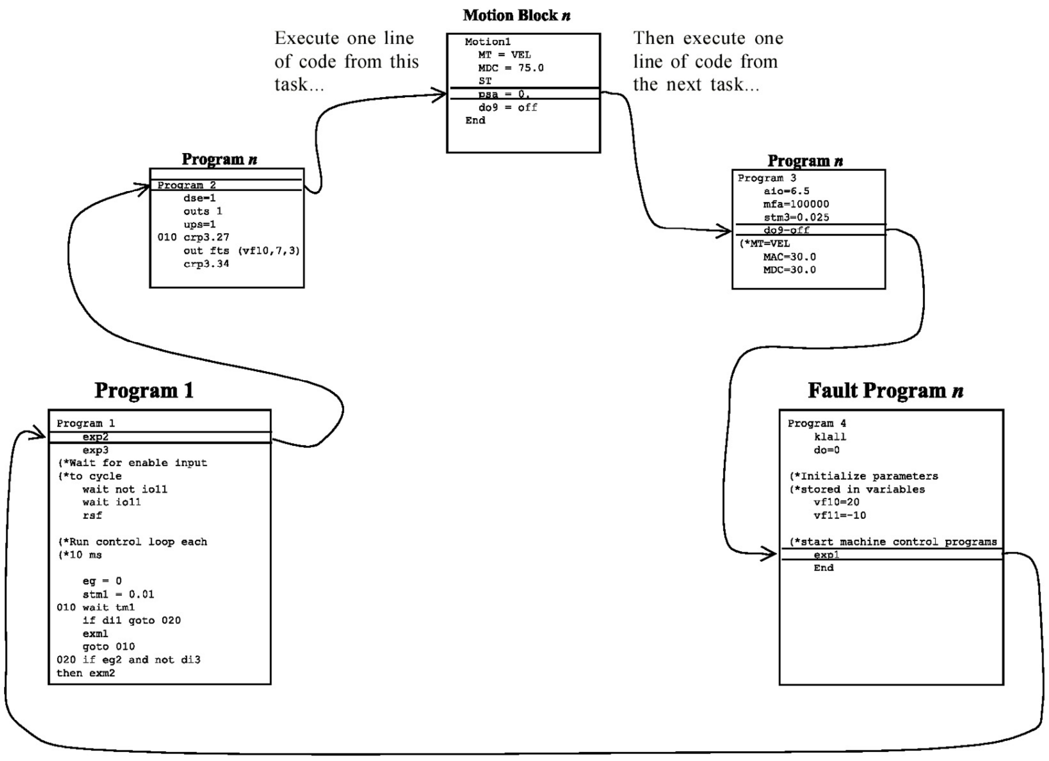
Figure 5.1: Structure of the Generation D RTOS

Multitasking

Multitasking provides a convenient and reliable technique for adding versatility and performance to real-time control systems. The IMC supports up to 6 concurrent tasks, including up to 4 programs, 1 motion block, and 1 communication port. The Target ARS supports up to 26 concurrent tasks, including up to 17 programs, 8 motion blocks, and 1 communication port. Your communication port allows you to receive registers or commands while you are executing other tasks.

How Multitasking Works (EXP, KLP, EXM, KLALL)

The Generation D RTOS resources are shared among all executing tasks. The arrows in figure 5.2 illustrate how those tasks are executed on a round-robin basis—one line of code is executed from a given task before the processor continues to the next task. Tasks run independently of each other except as designed by the programmer. Each program has equal priority and the same access to system resources.


Motion Block *n*
Execute one line of code from this task...
Then execute one line of code from the next task...
Program *n*
Program *n*
Program 1
Fault Program *n*

```
Motion1
  MT = VEL
  MDC = 75.0
  ST
  psa = 0.
  do9 = off
End
```

```
Program 2
     dse=1
     outs 1
     ups=1
010 crp3.27
     out fts (vf10,7,3)
     crp3.34
```

```
Program 3
     aio=6.5
     mfa=100000
     stm3=0.025
     do9-off
(*MT=VEL
     MAC=30.0
     MDC=30.0
```

```
Program 1
     exp2
     exp3
(*Wait for enable input
(*to cycle
     wait not io11
     wait io11
     rsf

(*Run control loop each
(*10 ms

     eg = 0
     stm1 = 0.01
010 wait tm1
     if di1 goto 020
     exm1
     goto 010
020 if eg2 and not di3
then exm2
```

```
Program 4
     klall
     do=0

(*Initialize parameters
(*stored in variables
     vf10=20
     vf11=-10

(*start machine control programs
     exp1
     End
```

Figure 5.2: Multitasking in the Generation D RTOS

Multitasking Commands

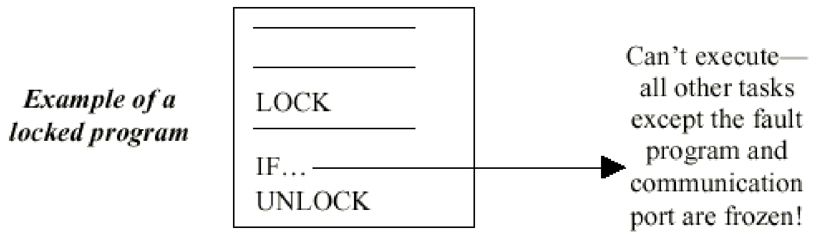
Use **EXP***n* and **KLP***n* to start and stop individual programs; use **EXM***n* to start a motion block. The **KLALL** command will stop all programs. Figure 5.3 tells you more about the multitasking commands:

Figure 5.3: Multitasking Commands

EXM*n*	Kills any currently running motion block that includes the same MBA assignment and runs motion block *n*. If the motion block is currently running, then EXM*n* will restart the motion block at its beginning. For example:
	`IF DI4 THEN          (* if input 4 then`
	`EXM5                 (* start motion block 5`

EXP*n*	Runs program *n*. If program *n* is currently running, then EXP*n* has no effect on program flow.
KLALL	Stop all programs but *not* motion blocks.

KLP*n*	Kills program *n*. If program *n* is not running, then KLP*n* has no effect on program flow. For example:
	`IF DI4 THEN          (* if input 4 then`
	`EXP2                 (* start program 2`
	`IF NOT DI5 THEN      (* if not input 5 then`
	`KLP3                 (* kill program 3`

LOCK/ UNLOCK	LOCK increases the execution rate for time-critical functions by allocating all of the controller's CPU resources to one task. It prevents any other programs and motion blocks from executing concurrently. If you use LOCK, be sure to UNLOCK before your program tries to execute a line of code that requires interaction with another program or motion block:  *Example of a locked program* LOCK IF… UNLOCK Can't execute—all other tasks except the fault program and communication port are frozen!

Now that you understand how multitasking works in the Generation D RTOS, you are ready to create your own application program. This process begins with *Step 1: Set System Constants* on the following page.

Step 1: Set System Constants

You will need to configure several registers when first using your DspMotion controller. The flowcharts that follow (figures 5.5 and 5.6) will take you through the necessary steps to set IMC and Target system constants. Place all system constants in your .txt application program file. When you send the file to the controller, you will simultaneously initialize the IMC or Target ARS with the proper parameters. Registers shown with an underscore are restricted and cannot be set to new values from within a program or motion block—they can, however, be included in your .txt application program file.

If you are using an IMC, your procedure begins on page 5-6. If you are using a Target ARS, please turn to page 5-11.

Once you have set system constants, proceed to *step 2* of your application program development: *Assess Task Interaction* on page 5-17.

Setting IMC System Constants

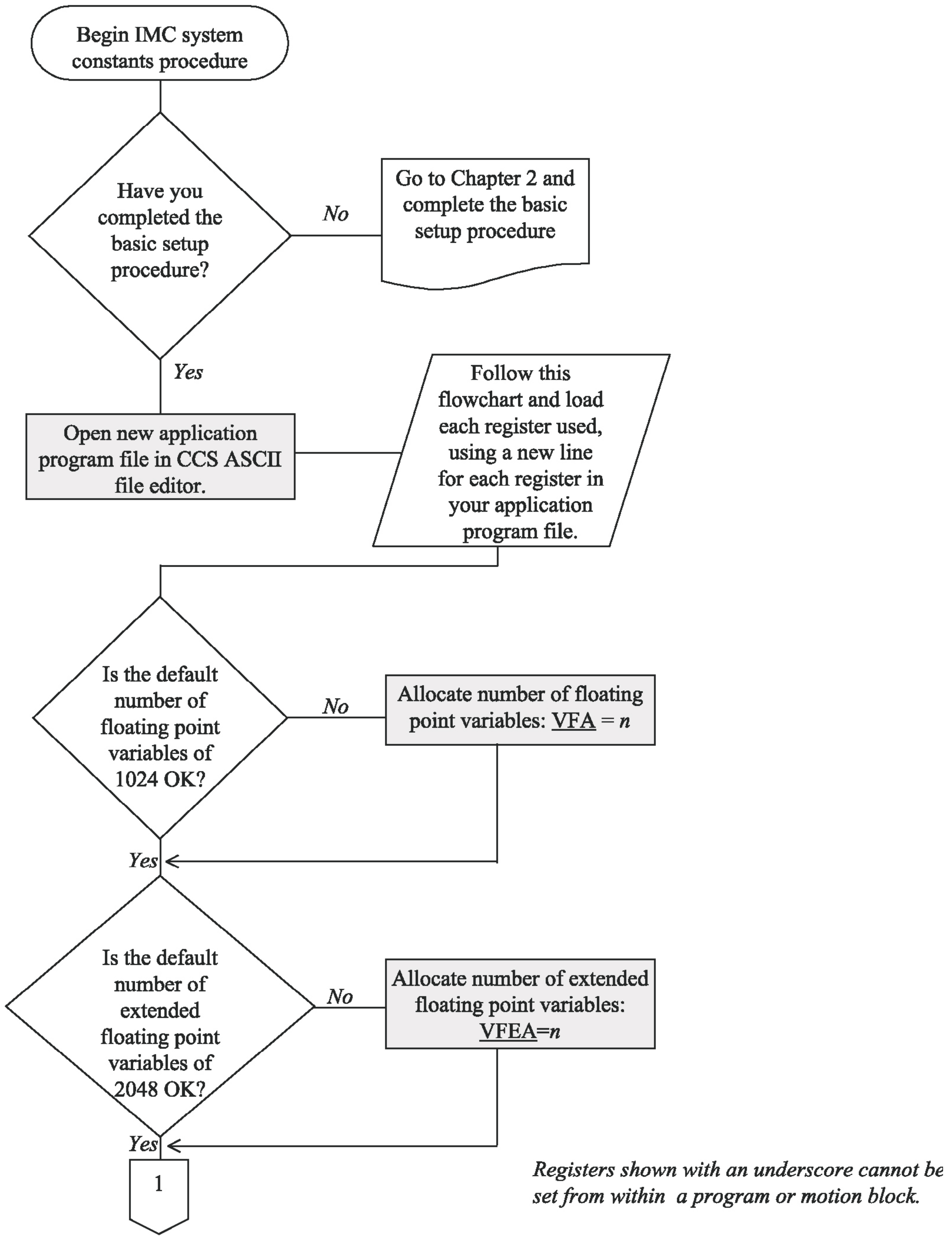

Begin IMC system constants procedure
Have you completed the basic setup procedure?
No
Go to Chapter 2 and complete the basic setup procedure
Yes
Open new application program file in CCS ASCII file editor.
Follow this flowchart and load each register used, using a new line for each register in your application program file.
Is the default number of floating point variables of 1024 OK?
No
Allocate number of floating point variables: VFA = n
Yes
Is the default number of extended floating point variables of 2048 OK?
No
Allocate number of extended floating point variables: VFEA=n
Yes
1
Registers shown with an underscore cannot be set from within a program or motion block.

Figure 5.4: Procedure for Setting IMC System Constants

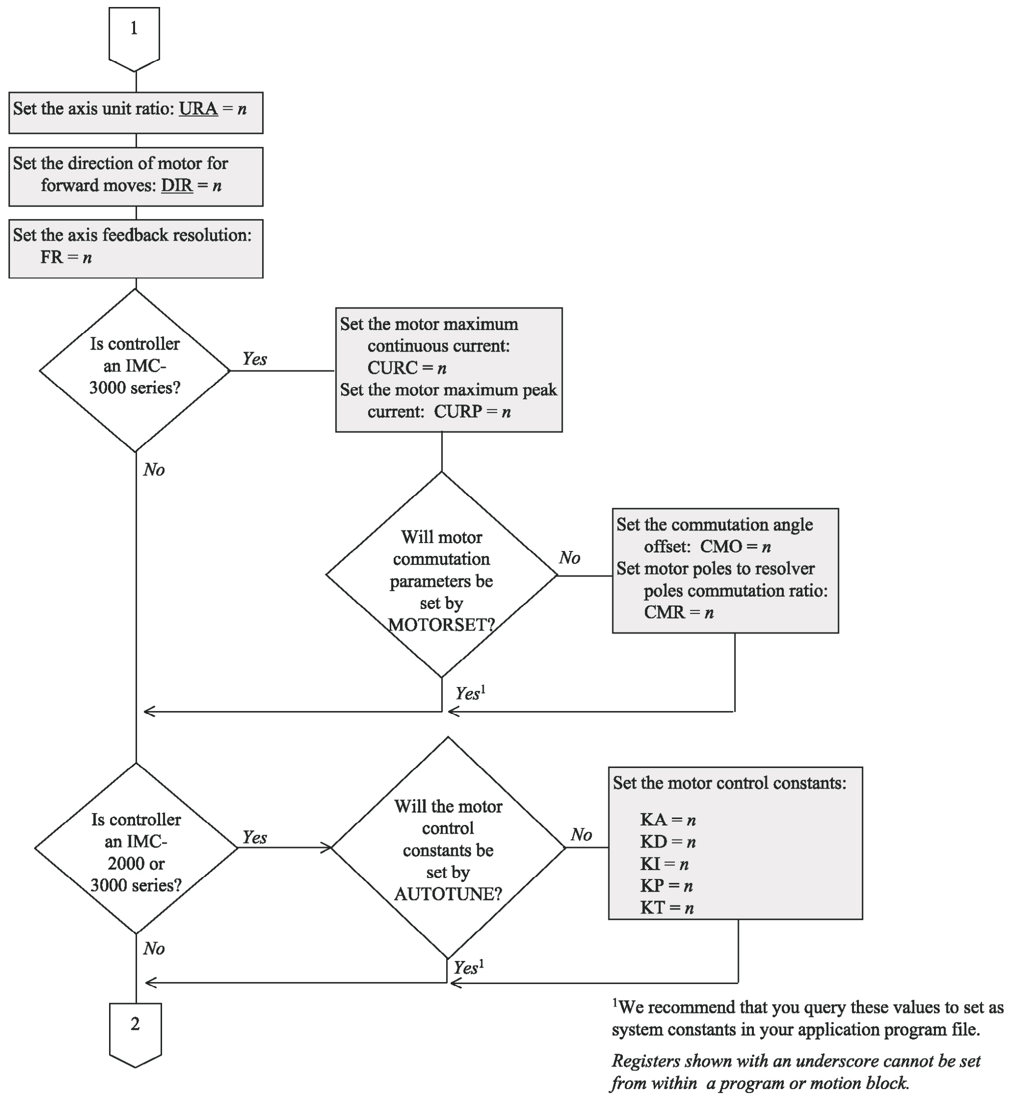

1
Set the axis unit ratio: URA = *n*
Set the direction of motor for forward moves: DIR = *n*
Set the axis feedback resolution: FR = *n*
Is controller an IMC-3000 series?
Yes
No
Set the motor maximum continuous current: CURC = *n*
Set the motor maximum peak current: CURP = *n*
Will motor commutation parameters be set by MOTORSET?
No
Yes[1]
Set the commutation angle offset: CMO = *n*
Set motor poles to resolver poles commutation ratio: CMR = *n*
Is controller an IMC-2000 or 3000 series?
Yes
No
Will the motor control constants be set by AUTOTUNE?
No
Yes[1]
Set the motor control constants:
KA = *n*
KD = *n*
KI = *n*
KP = *n*
KT = *n*
2
[1]We recommend that you query these values to set as system constants in your application program file.
Registers shown with an underscore cannot be set from within a program or motion block.

Figure 5.4: Procedure for Setting IMC System Constants (continued)

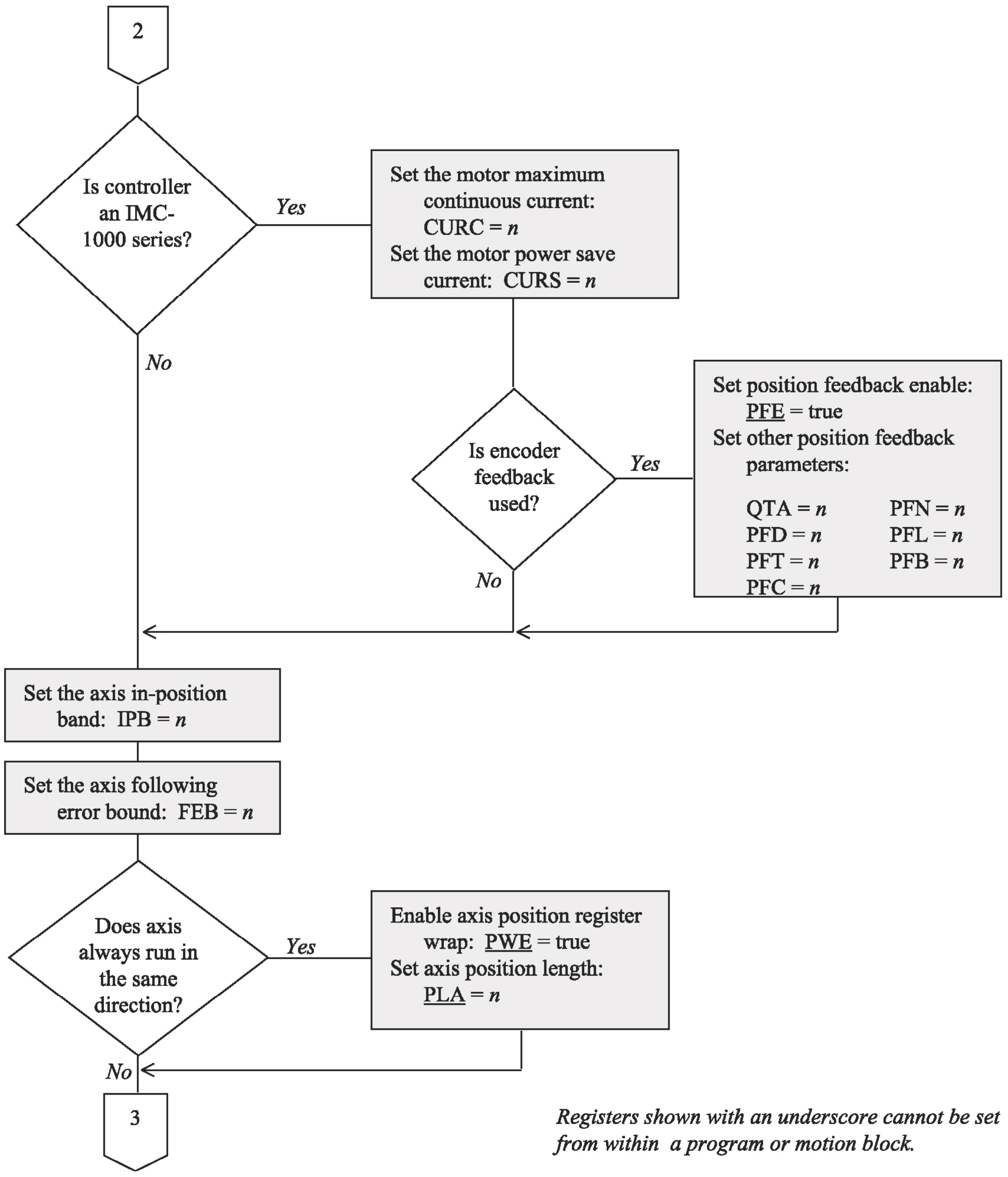

2
Is controller an IMC-1000 series?
Yes
Set the motor maximum continuous current: CURC = *n*
Set the motor power save current: CURS = *n*
No
Is encoder feedback used?
Yes
Set position feedback enable: PFE = true
Set other position feedback parameters:
QTA = *n*
PFN = *n*
PFD = *n*
PFL = *n*
PFT = *n*
PFB = *n*
PFC = *n*
No
Set the axis in-position band: IPB = *n*
Set the axis following error bound: FEB = *n*
Does axis always run in the same direction?
Yes
Enable axis position register wrap: PWE = true
Set axis position length: PLA = *n*
No
3
Registers shown with an underscore cannot be set from within a program or motion block.

Figure 5.4: Procedure for Setting IMC System Constants (continued)

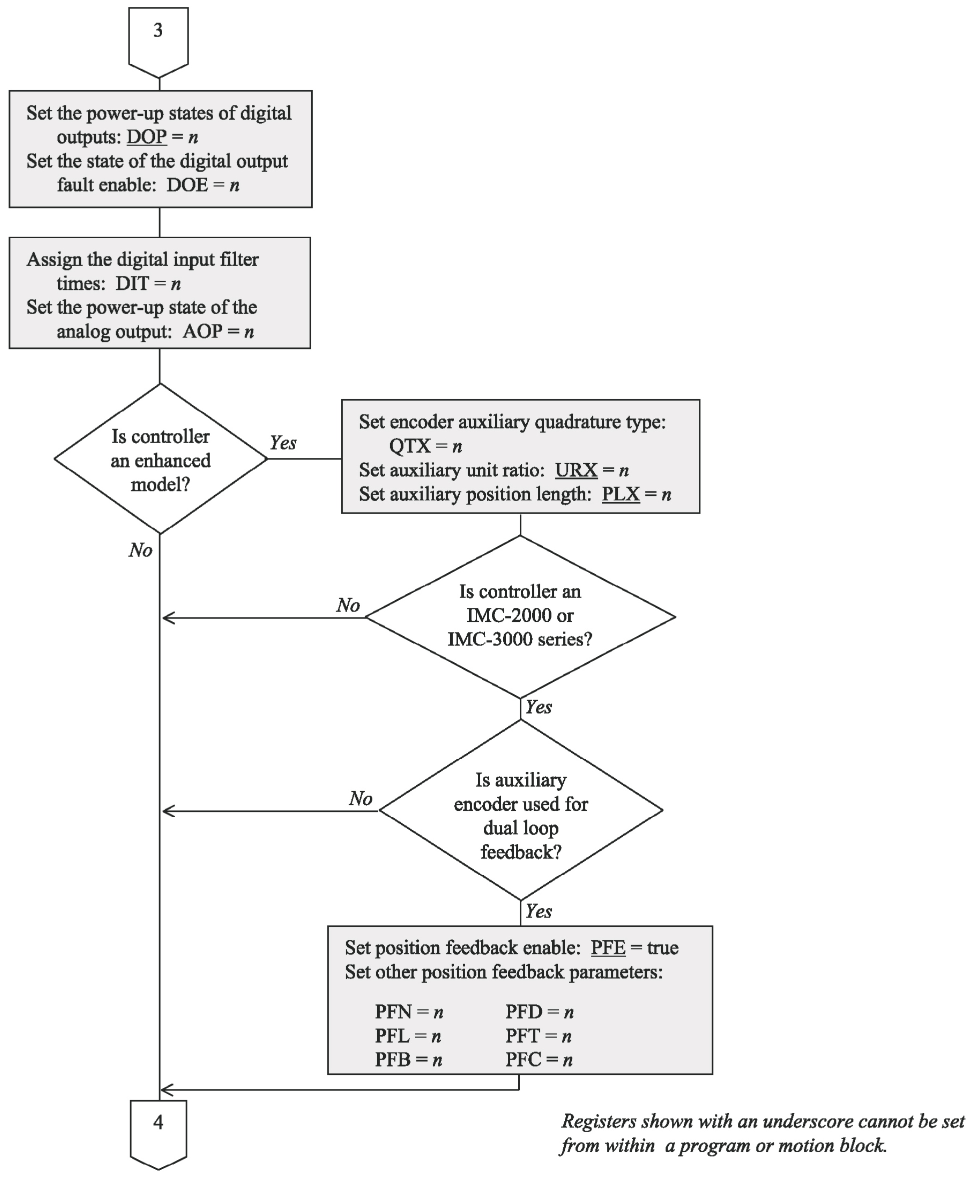
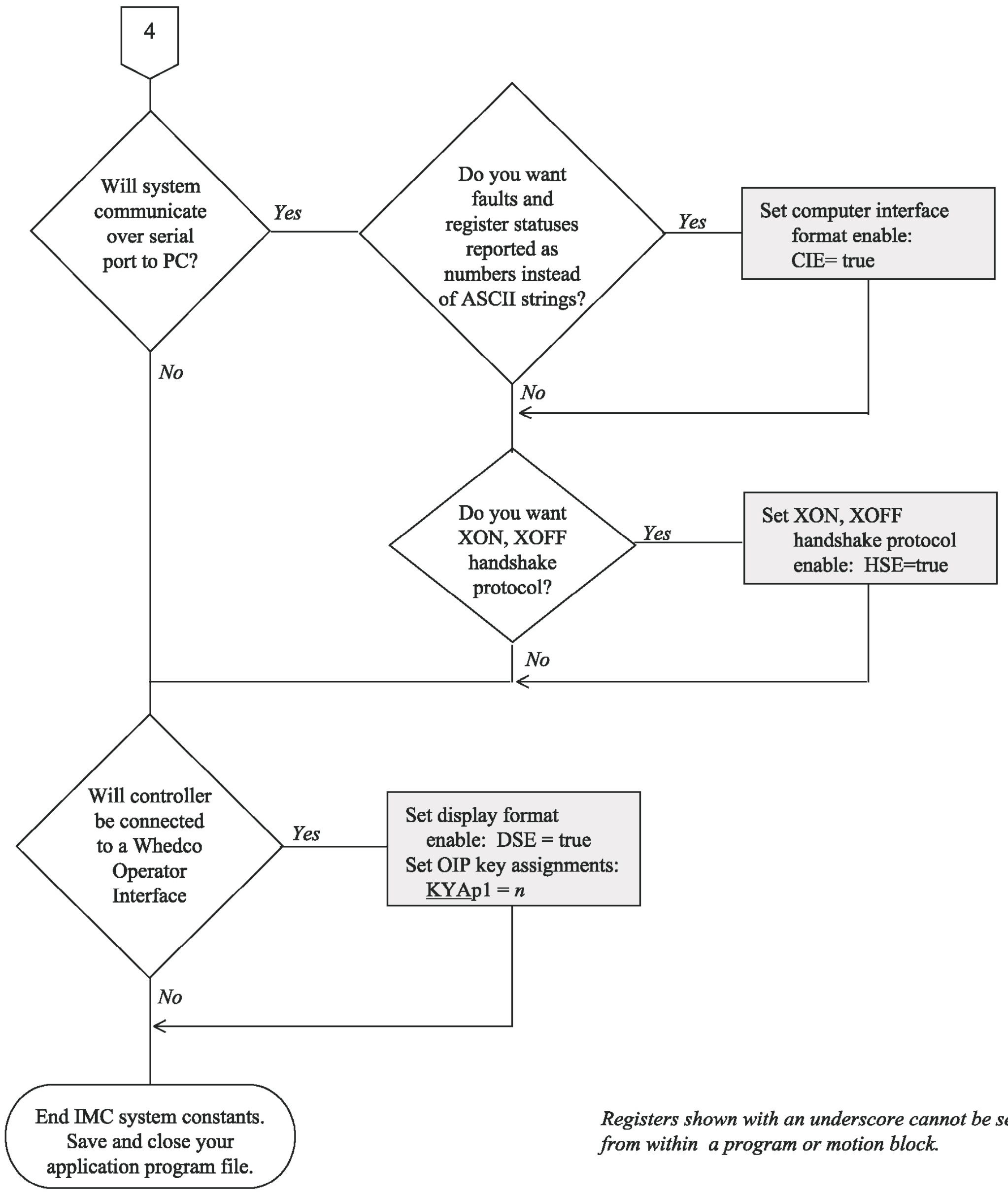

3
Set the power-up states of digital outputs: DOP = n
Set the state of the digital output fault enable: DOE = n
Assign the digital input filter times: DIT = n
Set the power-up state of the analog output: AOP = n
Is controller an enhanced model?
Yes
No
Set encoder auxiliary quadrature type: QTX = n
Set auxiliary unit ratio: URX = n
Set auxiliary position length: PLX = n
Is controller an IMC-2000 or IMC-3000 series?
No
Yes
Is auxiliary encoder used for dual loop feedback?
No
Yes
Set position feedback enable: PFE = true
Set other position feedback parameters:
PFN = n
PFD = n
PFL = n
PFT = n
PFB = n
PFC = n
4
Registers shown with an underscore cannot be set from within a program or motion block.

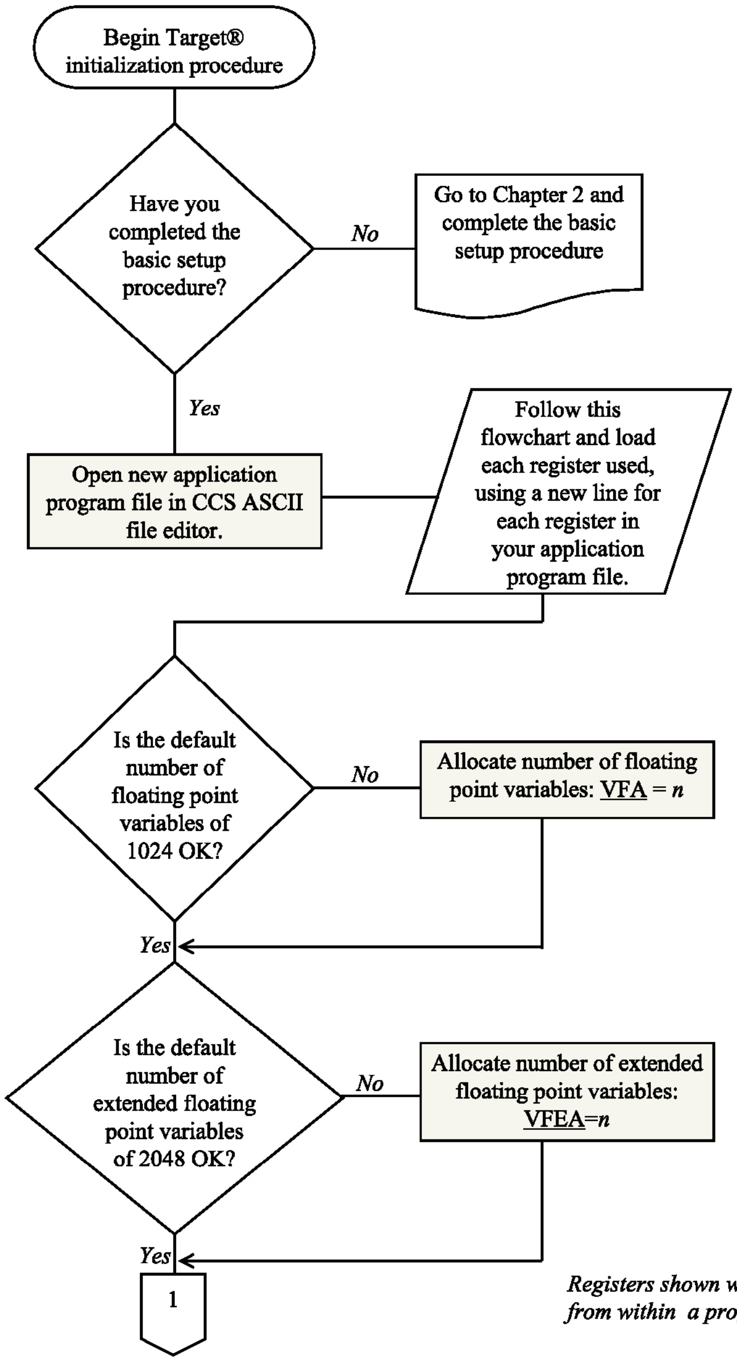
Figure 5.4: Procedure for Setting IMC System Constants (continued)


4
Will system communicate over serial port to PC?
Yes
No
Do you want faults and register statuses reported as numbers instead of ASCII strings?
Yes
No
Set computer interface format enable: CIE= true
Do you want XON, XOFF handshake protocol?
Yes
No
Set XON, XOFF handshake protocol enable: HSE=true
Will controller be connected to a Whedco Operator Interface
Yes
No
Set display format enable: DSE = true
Set OIP key assignments: KYAp1 = n
End IMC system constants. Save and close your application program file.

Registers shown with an underscore cannot be set from within a program or motion block.

Figure 5.4: Procedure for Setting IMC System Constants (continued)

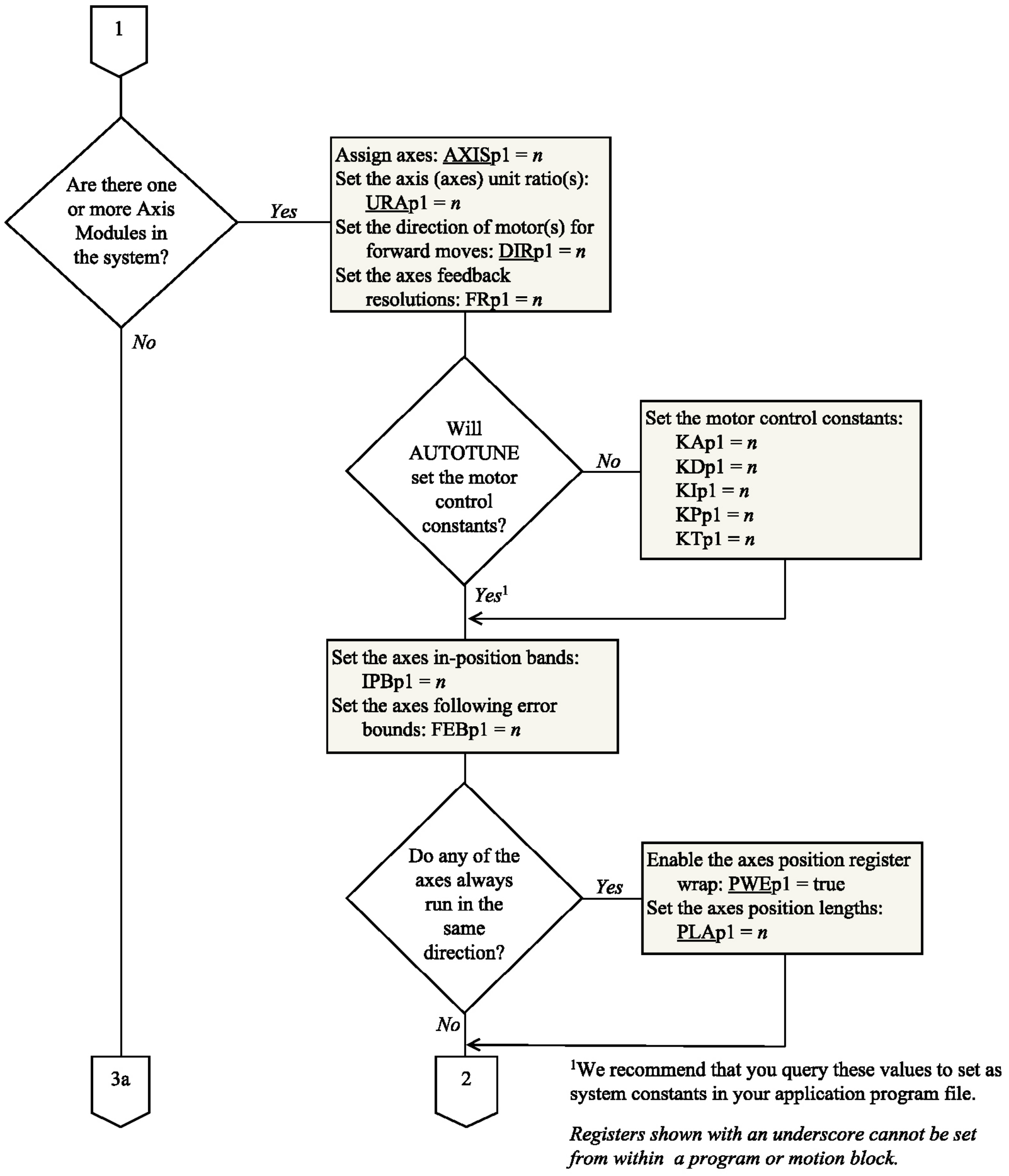
Setting Target System Constants



Begin Target® initialization procedure
Have you completed the basic setup procedure?
No
Go to Chapter 2 and complete the basic setup procedure
Yes
Open new application program file in CCS ASCII file editor.
Follow this flowchart and load each register used, using a new line for each register in your application program file.
Is the default number of floating point variables of 1024 OK?
No
Allocate number of floating point variables: VFA = n
Yes
Is the default number of extended floating point variables of 2048 OK?
No
Allocate number of extended floating point variables: VFEA=n
Yes
1

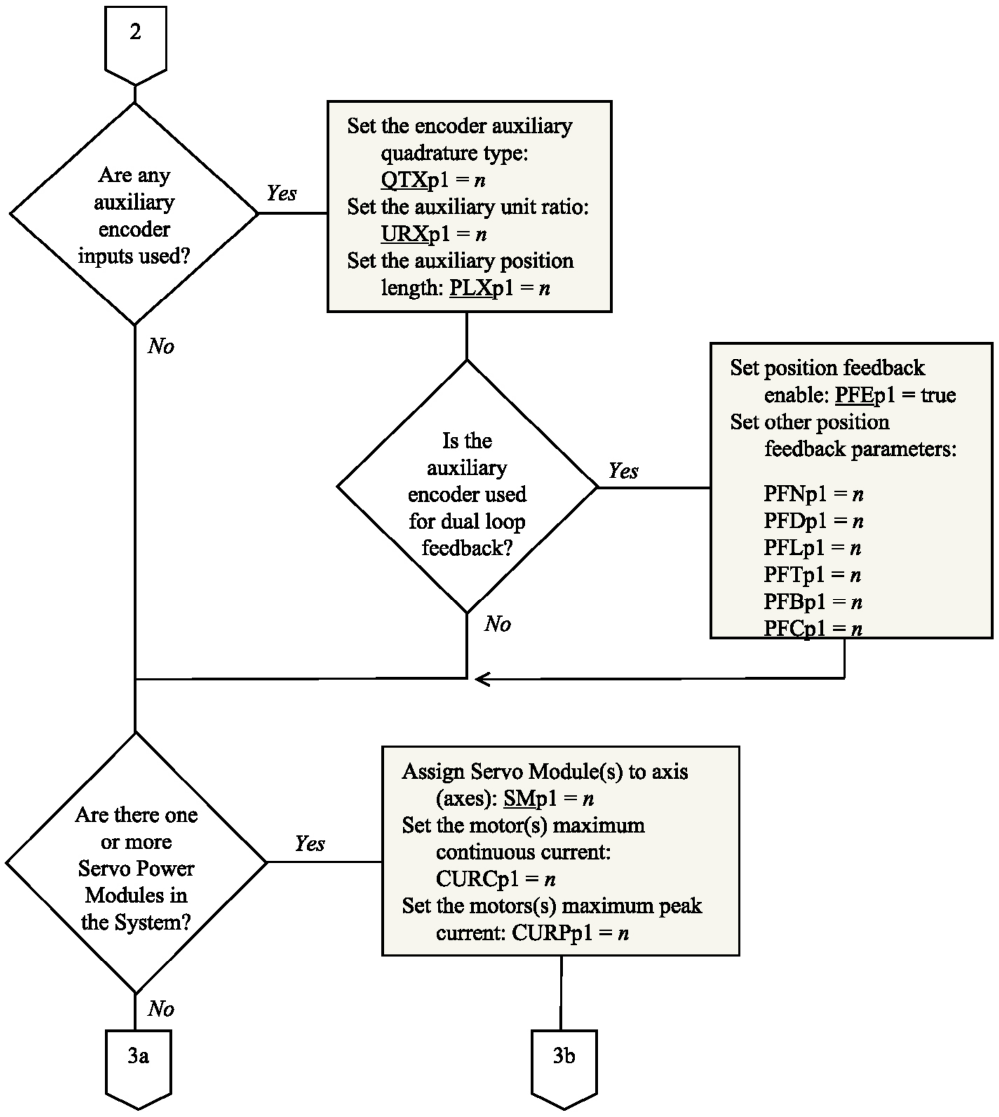
Registers shown with an underscore cannot be set from within a program or motion block.

Figure 5.5: Procedure for Setting Target System Constants



1
Are there one or more Axis Modules in the system?
Yes
Assign axes: AXISp1 = *n*
Set the axis (axes) unit ratio(s): URAp1 = *n*
Set the direction of motor(s) for forward moves: DIRp1 = *n*
Set the axes feedback resolutions: FRp1 = *n*
No
Will AUTOTUNE set the motor control constants?
No
Set the motor control constants:
KAp1 = *n*
KDp1 = *n*
KIp1 = *n*
KPp1 = *n*
KTp1 = *n*
Yes[1]
Set the axes in-position bands: IPBp1 = *n*
Set the axes following error bounds: FEBp1 = *n*
Do any of the axes always run in the same direction?
Yes
Enable the axes position register wrap: PWEp1 = true
Set the axes position lengths: PLAp1 = *n*
No
3a
2
[1]We recommend that you query these values to set as system constants in your application program file.
Registers shown with an underscore cannot be set from within a program or motion block.

Figure 5.5: Procedure for Setting Target System Constants (continued)


2
Are any auxiliary encoder inputs used?
Yes
No
Set the encoder auxiliary quadrature type: QTXp1 = *n*
Set the auxiliary unit ratio: URXp1 = *n*
Set the auxiliary position length: PLXp1 = *n*
Is the auxiliary encoder used for dual loop feedback?
Yes
No
Set position feedback enable: PFEp1 = true
Set other position feedback parameters:
PFNp1 = *n*
PFDp1 = *n*
PFLp1 = *n*
PFTp1 = *n*
PFBp1 = *n*
PFCp1 = *n*
Are there one or more Servo Power Modules in the System?
Yes
No
Assign Servo Module(s) to axis (axes): SMp1 = *n*
Set the motor(s) maximum continuous current: CURCp1 = *n*
Set the motors(s) maximum peak current: CURPp1 = *n*
3a
3b

Registers shown with an underscore cannot be set from within a program or motion block.

Figure 5.5: Procedure for Setting Target System Constants (continued)

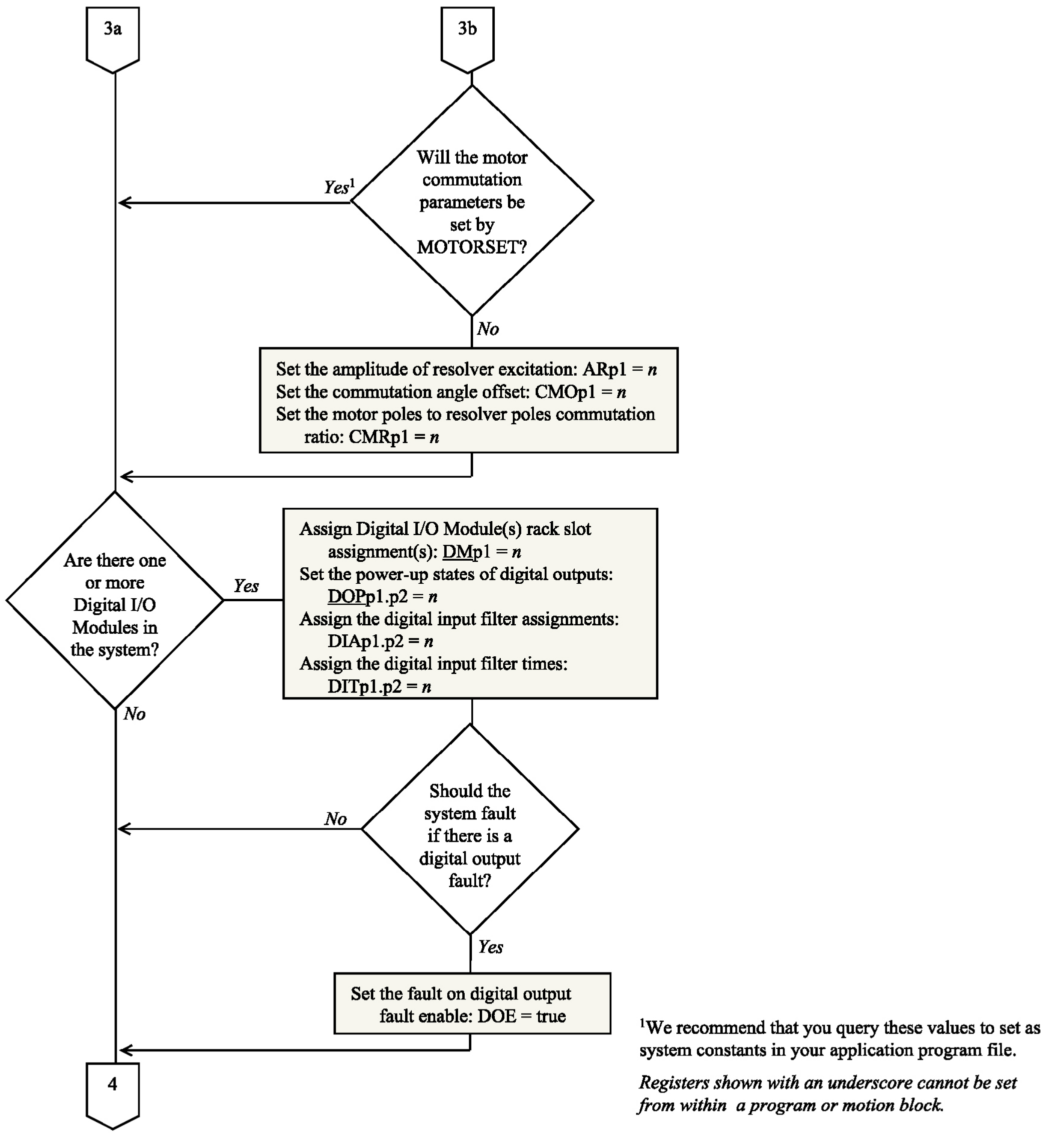
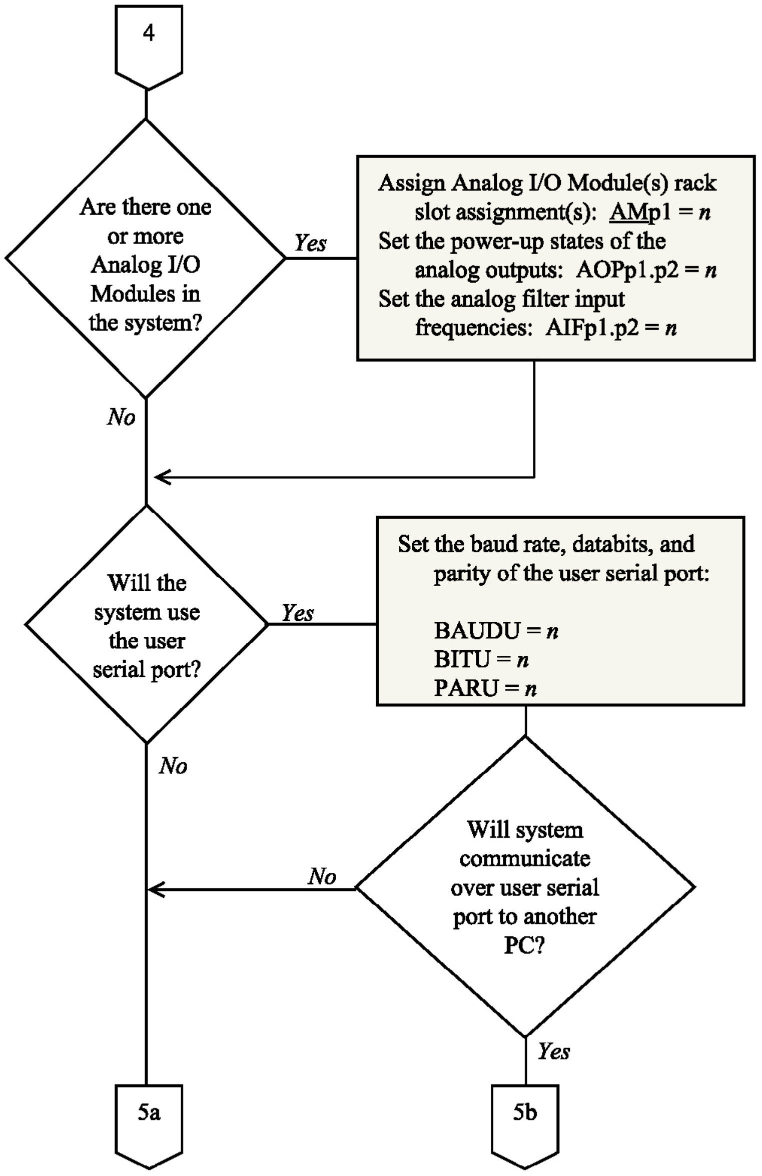

3a
3b
Will the motor commutation parameters be set by MOTORSET?
Yes¹
No
Set the amplitude of resolver excitation: ARp1 = *n*
Set the commutation angle offset: CMOp1 = *n*
Set the motor poles to resolver poles commutation ratio: CMRp1 = *n*
Are there one or more Digital I/O Modules in the system?
Yes
No
Assign Digital I/O Module(s) rack slot assignment(s): DMp1 = *n*
Set the power-up states of digital outputs: DOPp1.p2 = *n*
Assign the digital input filter assignments: DIAp1.p2 = *n*
Assign the digital input filter times: DITp1.p2 = *n*
Should the system fault if there is a digital output fault?
No
Yes
Set the fault on digital output fault enable: DOE = true
4
¹We recommend that you query these values to set as system constants in your application program file.
Registers shown with an underscore cannot be set from within a program or motion block.

Figure 5.5: Procedure for Setting Target System Constants (continued)


4
Are there one or more Analog I/O Modules in the system?
Yes
Assign Analog I/O Module(s) rack slot assignment(s): AMp1 = *n*
Set the power-up states of the analog outputs: AOPp1.p2 = *n*
Set the analog filter input frequencies: AIFp1.p2 = *n*
No
Will the system use the user serial port?
Yes
Set the baud rate, databits, and parity of the user serial port:
BAUDU = *n*
BITU = *n*
PARU = *n*
No
Will system communicate over user serial port to another PC?
No
Yes
5a
5b

Registers shown with an underscore cannot be set from within a program or motion block.

Figure 5.5: Procedure for Setting Target System Constants (continued)

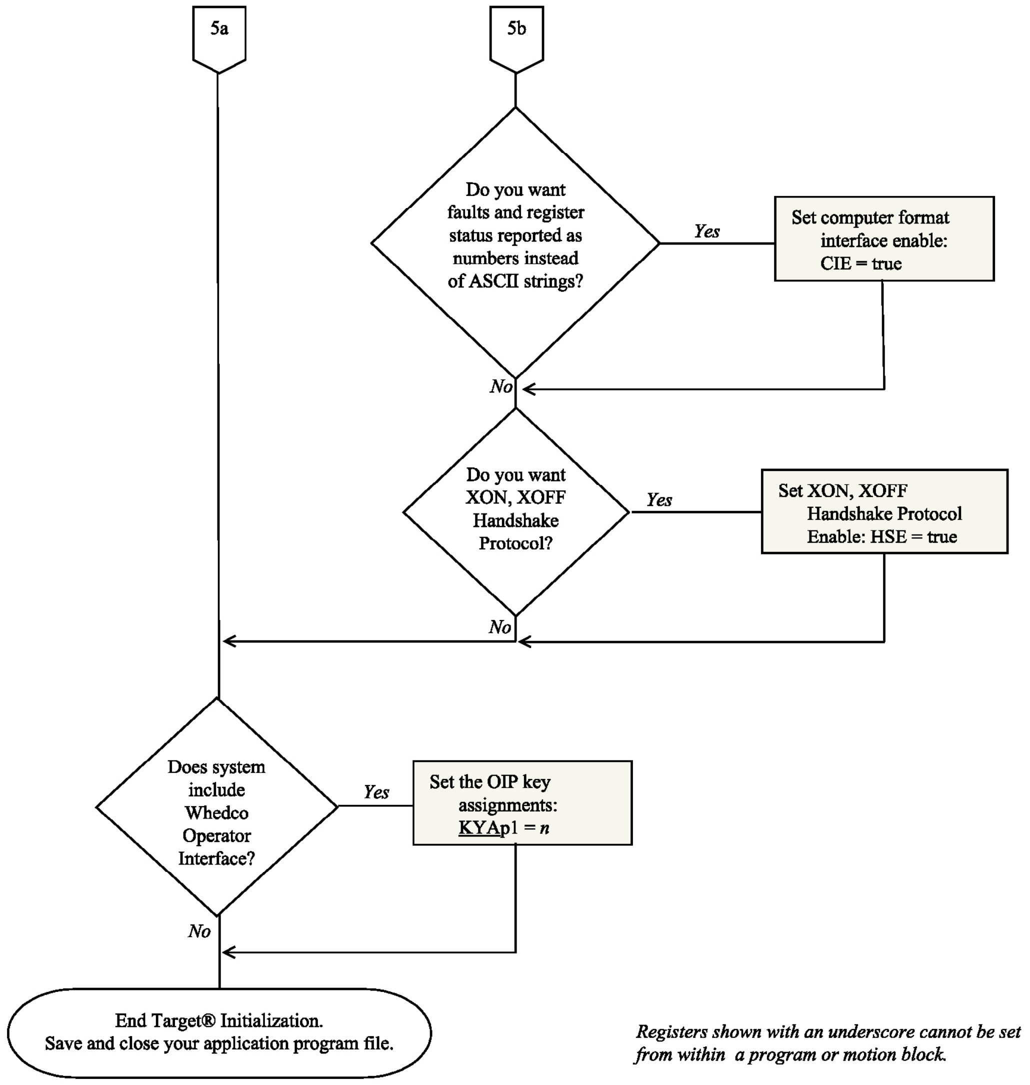

5a
5b
Do you want faults and register status reported as numbers instead of ASCII strings?
Yes
Set computer format interface enable: CIE = true
No
Do you want XON, XOFF Handshake Protocol?
Yes
Set XON, XOFF Handshake Protocol Enable: HSE = true
No
Does system include Whedco Operator Interface?
Yes
Set the OIP key assignments: KYAp1 = n
No
End Target® Initialization. Save and close your application program file.
Registers shown with an underscore cannot be set from within a program or motion block.

Figure 5.5: Procedure for Setting Target System Constants (continued)

Step 2: Assess Task Interaction

After determining your system constants, assess how the tasks within your application program should interact with each other and with your DspMotion firmware. All Generation D RTOS application programs are inherently multitasking and must comprise a minimum of three parts:

- Program 1 for main machine control functions
- Program 4 or 17 for fault handling
- The system constants that you set in *step 1*.

The use of additional programs will be application-dependent.

Figure 5.6 illustrates how the tasks, or individual programs and motion blocks, interact within your total application program once you have set your system constants:

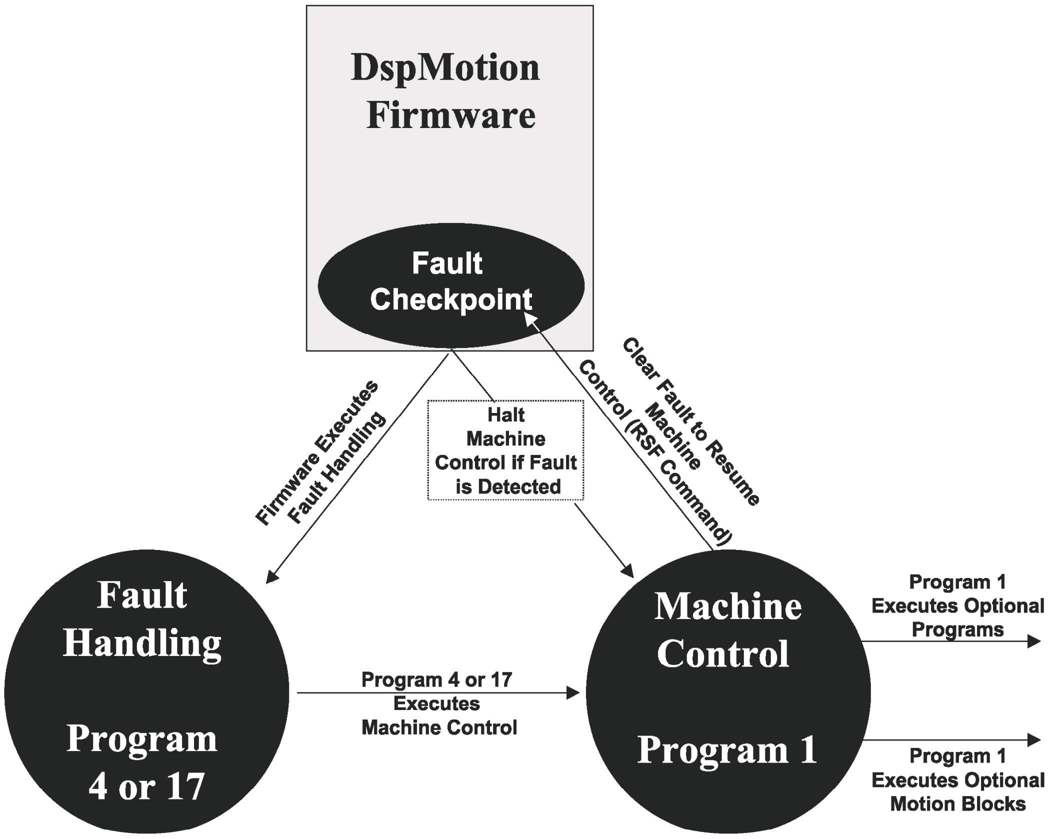

DspMotion Firmware
Fault Checkpoint
Firmware Executes Fault Handling
Halt Machine Control if Fault is Detected
Clear Fault to Resume Machine Control (RSF Command)
Fault Handling Program 4 or 17
Program 4 or 17 Executes Machine Control
Machine Control Program 1
Program 1 Executes Optional Programs
Program 1 Executes Optional Motion Blocks

Figure 5.6: Task Interaction

The basics of task interaction are simple yet critical elements of your application program design. As figure 5.6 shows, the fault handling program should execute program 1; program 1 should RSF (clear any detected faults) and execute any optional, secondary tasks. With these task interaction basics in mind, proceed to *Step 3: Structure a Fault Handling Program*.

Step 3: Structure a Fault Handling Program

What Happens When a Fault Occurs?

When the DspMotion control system detects one or more fault conditions, the DspMotion Controller automatically indicates that one or more fault conditions exist. The flowcharts in figures 5.7 and 5.8 document fault behaviors for the IMC and the Target ARS.

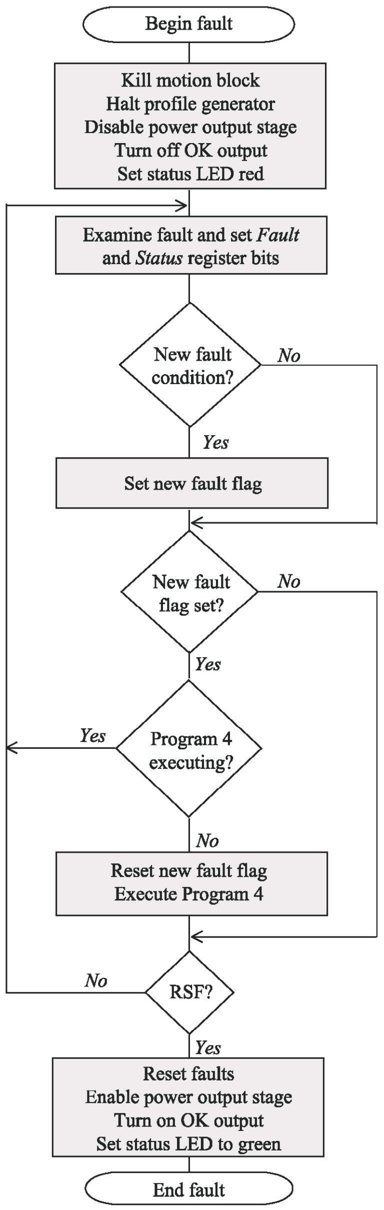


Figure 5.7: IMC Fault Behavior

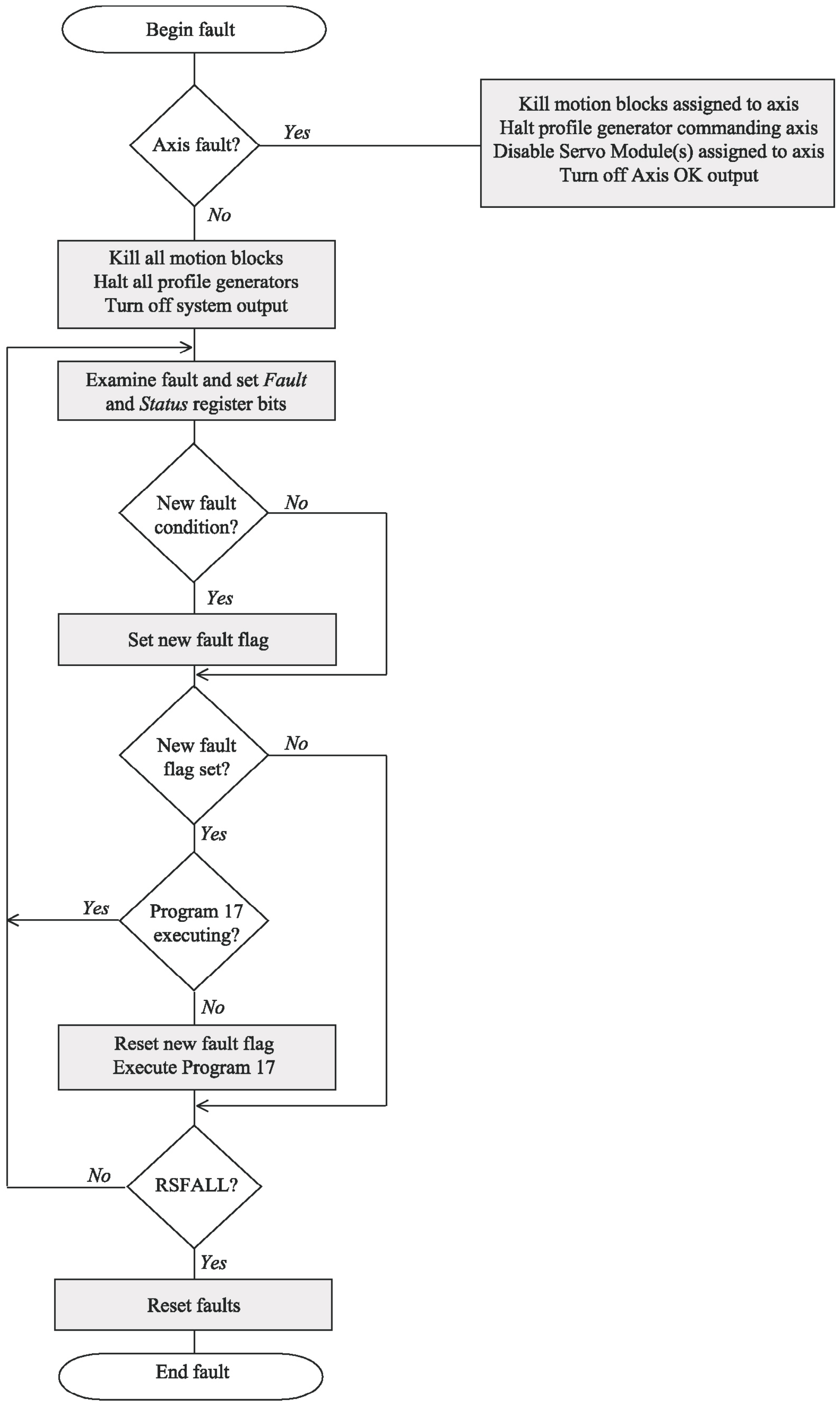


Figure 5.8: Target Fault Behavior

What Causes a Fault?

Many events can cause a fault—the two most common causes are *power loss* and *enable loss*. One or both of these conditions occur in the course of normal machine operation, for example, on power cycles or in an e-stop condition. Your fault handling program must diagnose the cause of the fault and determine the appropriate system behavior.

Any event causing a controller fault will start the fault program (i.e., program 4 for the IMC, or program 17 for the Target ARS). The following fault code registers contain the condition(s) that caused the fault:

Figure 5.9

Fault Code Registers (See Appendices E and F)	
FC	Fault Code Register
FCA	Axis Fault Code Register
FCS	System Fault Code Register

The IMC and the Target ARS also include status registers that provide additional information about the state of the controller and the dedicated I/O:

Figure 5.10

Status Registers (See Appendices E and F)	
SRA	Axis Status Register
SRAM	Analog Module Status Register
SRC	Communication Status Register
SRDM	Digital Module Status Register
SRP	Program Status Register
SRS	System Status Register
SRSM	Servo Module Status Register
SRT	Tertiary Port Status Register

Clearing Faults

Include either the RSF (for the IMC) or RSFALL (for the Target ARS) command in program 1 to clear fault conditions—these commands will work only when all of the conditions that caused the fault(s) have been corrected. If either RSF or RSFALL does not clear the fault(s), further diagnostics are required.

Recommended Fault Handling

Write your fault program so that the DspMotion Controller will efficiently analyze the fault conditions and direct program flow appropriately. The flowchart shown in figure 5.11 provides a recommended operation sequence for fault handling.

Incorporate the items included in figure 5.11 into your fault handling program. Be sure to document your program for future reference using the comment delimiter (*.

Use the REM command to embed critical program flow comments directly in programs or motion blocks.

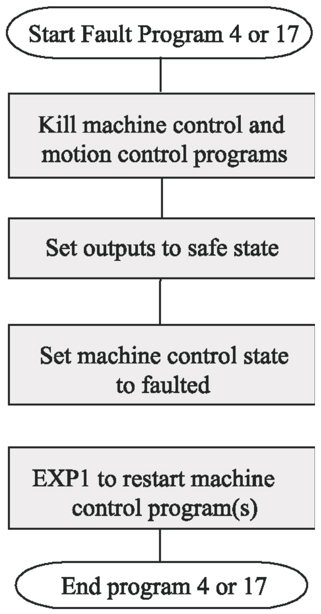

Start Fault Program 4 or 17
Kill machine control and motion control programs
Set outputs to safe state
Set machine control state to faulted
EXP1 to restart machine control program(s)
End program 4 or 17

Figure 5.11: Structure of the Fault Handling Program

Step 4: Structure Program 1 and Additional Tasks

Begin with a thorough assessment of your system needs, keeping in mind that the Generation D RTOS is a flexible operating system. Some good questions to ask include the following:

1. What tasks do I want to perform through programs?
2. What motions do I need to cause through motion blocks?
3. How can I divide my motion control tasks to get the maximum multitasking efficiency?

Document your answers to these questions and then use the following guidelines to determine how the tasks within your complete application program will interact:

- Use only as many tasks as are required to perform your application. Tasks include program 1 and any additional programs and motion blocks. Total execution efficiency is proportional to the number of total tasks executing.
- When using additional programs (program 2 and 3 in the IMC and programs 2-16 in the Target ARS), allocate specific functions to separate programs. Figure 5.12 shows an example in which one program runs the motor, a second program handles operator interface functions, and a third program outputs motor torque and sets position feedrate.
- Discipline yourself to use global resources (see figure 5.1 on page 5-2) in blocks that are unique to individual programs or motion blocks. This practice avoids interactions between programs or motion blocks that could load a variable or register with a value that is nonconforming in another program. An example of this practice would be to use integer variables 1-49 in program 1, 50-99 in program 2, and so forth.

- Document your ASCII file for future reference using the comment delimiter **(***.
- Embed critical program flow comments directly in programs or motion blocks with the REM command.

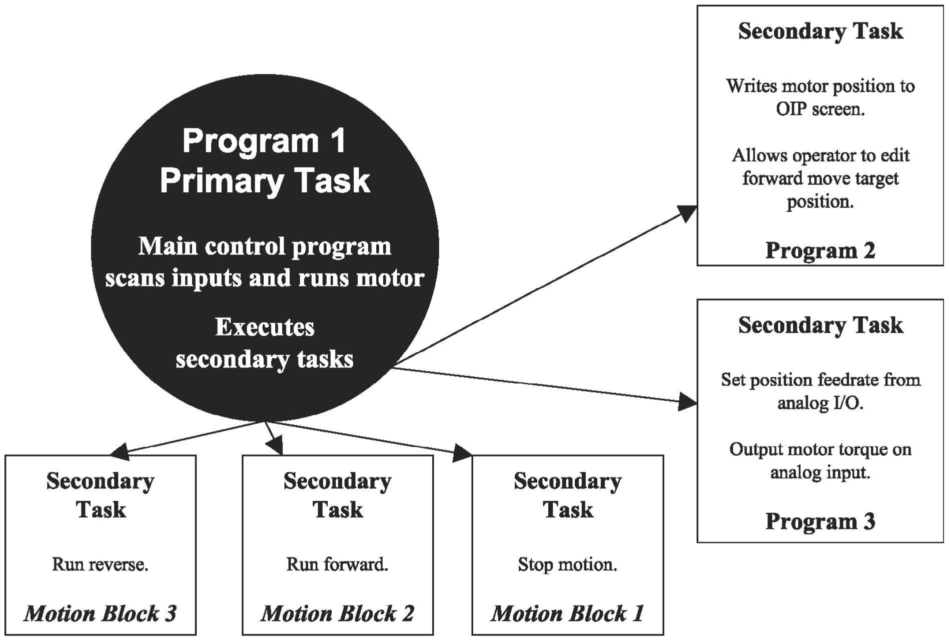

Program 1
Primary Task
Main control program scans inputs and runs motor
Executes secondary tasks
Secondary Task
Writes motor position to OIP screen.
Allows operator to edit forward move target position.
Program 2
Secondary Task
Set position feedrate from analog I/O.
Output motor torque on analog input.
Program 3
Secondary Task
Run reverse.
Motion Block 3
Secondary Task
Run forward.
Motion Block 2
Secondary Task
Stop motion.
Motion Block 1

Figure 5.12: Example of Application Program Structure and Task Division

Moving Forward…

When you have planned the tasks and motions that you want to perform and have decided how best to design your program 1 and additional tasks, use the Generation D RTOS registers, commands, operators, and operands (see appendices) to write your programs.

When you are ready to download your application program, run it, and diagnose any problems, turn to Chapters 3 and 6 for instructions.

Step 5: Manage Your Application Program

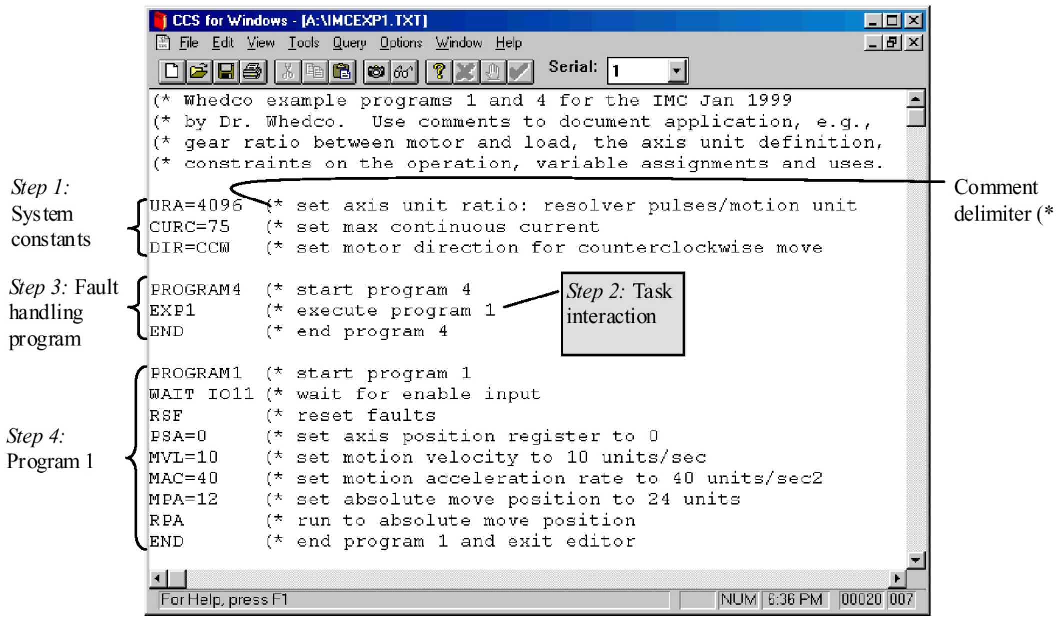
Archiving Your Program

When you complete your application program, we recommend that you adhere to the following discipline:

1. Incorporate all of the files (if there are more than one) into a single ASCII file. This file should include all programs, motion blocks, operator interface screens, and system constants.
2. Use the comment delimiter **(*** to fully document all programs within your ASCII file.
3. Practice good file management: store the ASCII file with any other project files.

We do not recommend that you use the controller itself as the archival device. Although CCS includes a **Tools**/**Receive**… utility for uploading the controller's application program, this utility is a diagnostic tool and not a means of program maintenance. Process information can change while a program is running, so uploaded programs may not be exactly the same as properly archived, original, and fully documented ASCII files.

The screen in figure 5.13 shows an archived application program file comprising system constants, Program 1, and a fault handling program (i.e., Program 4):



```
(* Whedco example programs 1 and 4 for the IMC Jan 1999
(* by Dr. Whedco.  Use comments to document application, e.g.,
(* gear ratio between motor and load, the axis unit definition,
(* constraints on the operation, variable assignments and uses.

URA=4096   (* set axis unit ratio: resolver pulses/motion unit
CURC=75    (* set max continuous current
DIR=CCW    (* set motor direction for counterclockwise move

PROGRAM4   (* start program 4
EXP1       (* execute program 1
END        (* end program 4

PROGRAM1   (* start program 1
WAIT IO11  (* wait for enable input
RSF        (* reset faults
PSA=0      (* set axis position register to 0
MVL=10     (* set motion velocity to 10 units/sec
MAC=40     (* set motion acceleration rate to 40 units/sec2
MPA=12     (* set absolute move position to 24 units
RPA        (* run to absolute move position
END        (* end program 1 and exit editor
```

Figure 5.13 Archived IMC Application Program

Using SECURE to Block User Access to Programs (Optional)

Use the SECURE command to protect your intellectual property—it will prevent programs and motions blocks from being received from the controller. This command also blocks use of the FAULT command. To enable the secure feature, first send the application program file to the controller, and then, from the Terminal window in CCS, type `SECURE`. To disable the SECURE feature, type `CLM` to clear the memory and start over.

Using PASSWORD Protection (Optional)

The PASSWORD command is intended to prohibit program modification in the field. To password-protect your DspMotion controller program:

- Type **`PASSWORD`** from the terminal window in CCS
- At the *Enter Password* prompt, type the four- to ten-character password of your choice.

__Caution!__ Do __NOT__ forget your password. After you set the password, you will have to enter the password before accessing the program. If you do not enter the correct password, you will be able to use only diagnostic commands—you will not be able to clear the memory (i.e., use the CLM command) to start over. To start over, you must return the controller to the factory. There is no back door!

To change the password, type `CHANGEPW` in the CCS Terminal window and follow the prompts.

Storing Your Program in Flash EPROM (Optional)

The DspMotion controller contains two types of user memory: BBRAM and EPROM. BBRAM stores and preserves data through power cycles. Flash EPROM is nonvolatile memory for permanent data storage that is not battery-dependent.

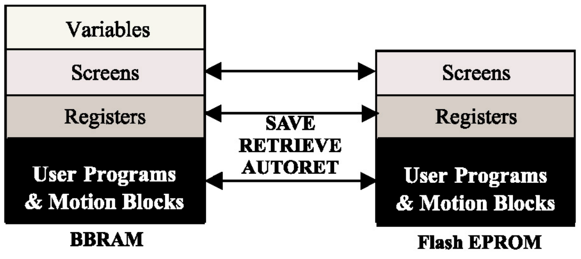


Figure 5.14: Memory in the DspMotion Controller

SAVE Command

Use the SAVE command to write your programs, motion blocks, registers, and screens from BBRAM to Flash EPROM. To execute the SAVE function:

1. Set faults
   a. For the IMC, type `STF`
   b. For the Target ARS, type `STFALL`
2. Type `KLALL`
3. Type `SAVE`

RETRIEVE Command

Use the RETRIEVE command to write your data from Flash EPROM to BBRAM. To execute this function:

1. Set faults
    a. For the IMC, type `STF`
    b. For the Target ARS, type `STFALL`
2. Type `KLALL`
3. Type `SAVE`

AUTORET Command

Use the AUTORET command to retrieve your data from Flash EPROM automatically upon each power-up of the DspMotion controller. This function is useful when your program is complete and the controller is installed in the application.

Note: *Be sure to SAVE any desired program changes before power-down.*

Complete the following steps to autoretrieve:

1. Set faults
    a. For the IMC, type `STF`
    b. For the Target ARS, type `STFALL`
2. Type `KLALL`
3. Type `AUTORET`
4. Type `SAVE`

Disable AUTORET Command

To disable the AUTORET function, you must clear the Flash EPROM memory.

Note*: Be sure you have a copy of the program file before clearing the memory.*

To disable autoretrieve:

1. Set faults
    a. For the IMC, type `STF`
    b. For the Target, type `STFALL`
2. Type `KLALL`
3. Type `UPS=0` (UPS must be set to its default value of zero before the CLM command will work.)
4. Type `CLM`
5. Type `SAVE`

Chapter 6 Application Program Diagnostics and Debugging Tools

In This Chapter

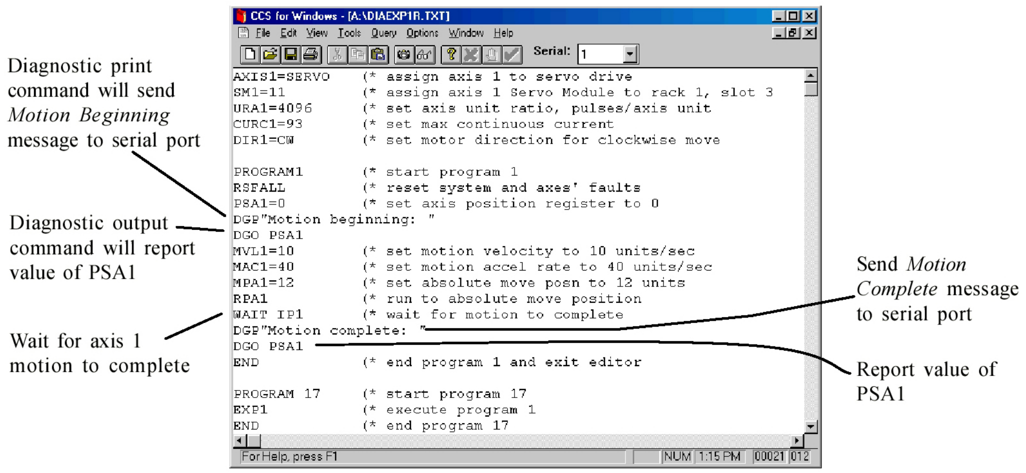
- ❑ Embed and enable diagnostics in an application program
- ❑ Runtime debugging tools
- ❑ About the *line editor*
- ❑ Find a bug with the FAULT command
- ❑ Fix a bug
- ❑ Monitor real-time machine parameters with **Query**/**Start** (Q, ?)
- ❑ Query registers for moment-in-time data (Q, ?)
- ❑ Run an application program in single-step mode
- ❑ Run an application program in trace mode
- ❑ Capture an online Terminal session.

Embed and Enable Diagnostics in an Application Program

DGP and DGO Commands

The Generation D RTOS includes several diagnostic commands that you can use with CCS for Windows to debug your application programs. You can integrate the diagnostic commands DGP and DGO into an application program to check register values or report other conditions during program execution without affecting program performance. The DGE command enables diagnostics—the controller ignores any diagnostic commands in an application program until you set DGE=1.

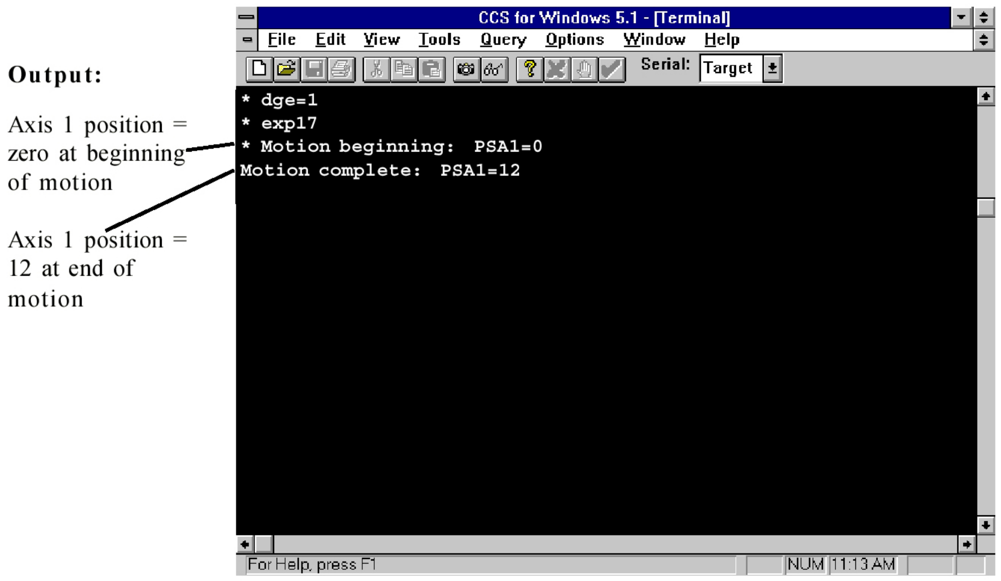
In the following example, we have clicked **File**/**New** and entered a Target application programs, including some diagnostics, in the CCS ASCII file editor:


Diagnostic print command will send *Motion Beginning* message to serial port
Diagnostic output command will report value of PSA1
Wait for axis 1 motion to complete
Send *Motion Complete* message to serial port
Report value of PSA1
CCS for Windows - [A:\DIAEXP1R.TXT]
File Edit View Tools Query Options Window Help
Serial: 1
For Help, press F1
NUM 1:15 PM 00021 012

```
AXIS1=SERVO     (* assign axis 1 to servo drive
SM1=11          (* assign axis 1 Servo Module to rack 1, slot 3
URA1=4096       (* set axis unit ratio, pulses/axis unit
CURC1=93        (* set max continuous current
DIR1=CW         (* set motor direction for clockwise move

PROGRAM1        (* start program 1
RSFALL          (* reset system and axes' faults
PSA1=0          (* set axis position register to 0
DGP"Motion beginning: "
DGO PSA1
MVL1=10         (* set motion velocity to 10 units/sec
MAC1=40         (* set motion accel rate to 40 units/sec
MPA1=12         (* set absolute move posn to 12 units
RPA1            (* run to absolute move position
WAIT IP1        (* wait for motion to complete
DGP"Motion complete: "
DGO PSA1
END             (* end program 1 and exit editor

PROGRAM 17      (* start program 17
EXP1            (* execute program 1
END             (* end program 17
```

Figure 6.1: Example of Diagnostics in a Target Application Program (DIAEXP1R.txt)

When we set DGE=1 and then execute the application program with the EXP17 command, we receive the following diagnostic information in the Terminal window:


Output:
Axis 1 position = zero at beginning of motion
Axis 1 position = 12 at end of motion
CCS for Windows 5.1 - [Terminal]
File Edit View Tools Query Options Window Help
Serial: Target
For Help, press F1
NUM 11:13 AM

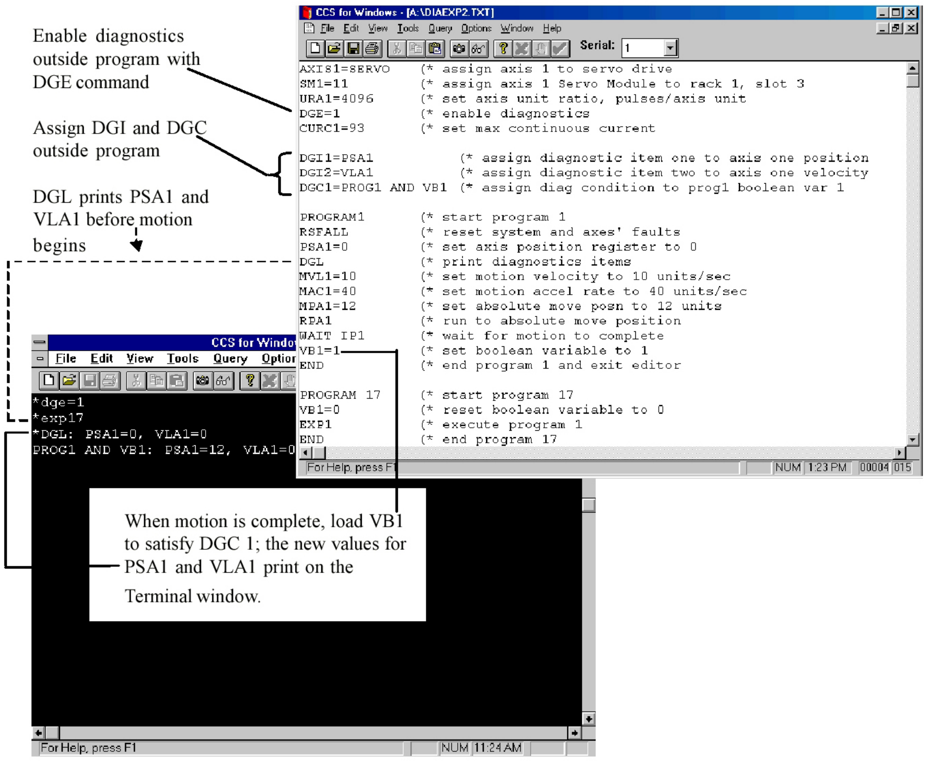
```
* dge=1
* exp17
* Motion beginning:  PSA1=0
Motion complete:  PSA1=12
```

Figure 6.2: Diagnostic Output Produced from Target Example Application Program DIAEXP1R.txt

DGC, DGI, and DGL Commands

The previous example showed you how to write diagnostics into your application program. Other diagnostics, such as the commands DGC and DGI, are not allowed *within* programs but are useful to assign diagnostic conditions or items to your system.

In the following Target example, we have assigned diagnostic items 1 and 2, established a diagnostic condition, and then created an application program that uses the DGL command.


Enable diagnostics outside program with DGE command
Assign DGI and DGC outside program
DGL prints PSA1 and VLA1 before motion begins
When motion is complete, load VB1 to satisfy DGC 1; the new values for PSA1 and VLA1 print on the Terminal window.
CCS for Windows - [A:\DIAEXP2.TXT]
File Edit View Tools Query Options Window Help
Serial: 1
For Help, press F1
NUM 1:23 PM 00004 015
CCS for Windo
File Edit View Tools Query Optior
For Help, press F1
NUM 11:24 AM

```
AXIS1=SERVO     (* assign axis 1 to servo drive
SM1=11          (* assign axis 1 Servo Module to rack 1, slot 3
URA1=4096       (* set axis unit ratio, pulses/axis unit
DGE=1           (* enable diagnostics
CURC1=93        (* set max continuous current

DGI1=PSA1            (* assign diagnostic item one to axis one position
DGI2=VLA1            (* assign diagnostic item two to axis one velocity
DGC1=PROG1 AND VB1   (* assign diag condition to prog1 boolean var 1

PROGRAM1        (* start program 1
RSFALL          (* reset system and axes' faults
PSA1=0          (* set axis position register to 0
DGL             (* print diagnostics items
MVL1=10         (* set motion velocity to 10 units/sec
MAC1=40         (* set motion accel rate to 40 units/sec
MPA1=12         (* set absolute move posn to 12 units
RPA1            (* run to absolute move position
WAIT IP1        (* wait for motion to complete
VB1=1           (* set boolean variable to 1
END             (* end program 1 and exit editor

PROGRAM 17      (* start program 17
VB1=0           (* reset boolean variable to 0
EXP1            (* execute program 1
END             (* end program 17
```

```
*dge=1
*exp17
*DGL: PSA1=0, VLA1=0
PROG1 AND VB1: PSA1=12, VLA1=0
```

Figure 6.3: Example of Target Application Program *DIAEXP2.txt* and Diagnostic Output after Program Execution in Terminal Window

You can use the **DGC** command to assign up to 4 IMC diagnostic conditions and up to eight Target diagnostic conditions that tell the system to print a diagnostic line of items to the Terminal window any time the condition is satisfied. Diagnostic conditions can be any Boolean expression, for example, program *n* executing (PROG*n*), timer *n* timed out (TM*n*) or motion generator enabled (SRA0).

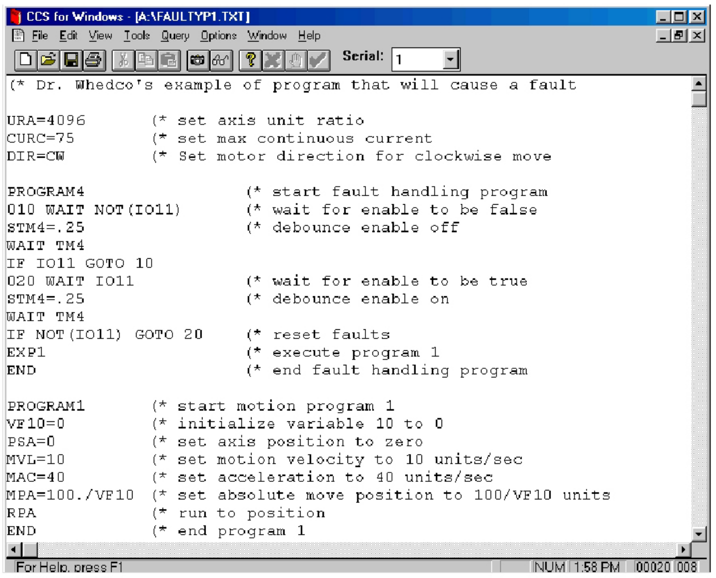
You can define up to eight diagnostic items using the **DGI** command. A diagnostic item is any system register that can be queried using ? or Q, such as axis position (PSA), axis velocity (VLA), or variable values (VB*n*, VI*n*, VF*n*, VS*n*).

To *unassign* a diagnostic item or condition, set it to OFF (e.g., **DGC1 = OFF**).

> ***Note: Remember to set DGE=1*** *to enable your diagnostics—otherwise, your controller will ignore them!*

Runtime Debugging Tools

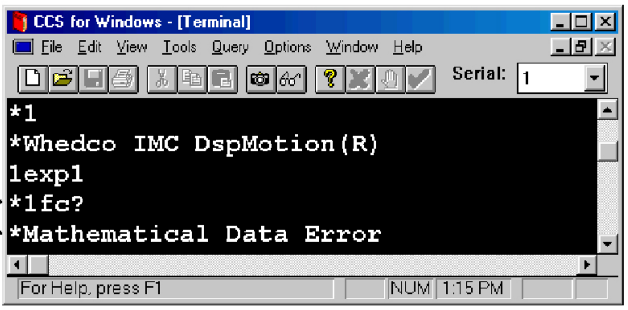
It’s probably no surprise—sometimes you’ll send a program to the controller without a hitch, and then it won’t run. For demonstration purposes, there is a bug into the following IMC program that is sent to the controller:


Can’t
divide
by zero!
CCS for Windows - [A:\FAULTYP1.TXT]
File Edit View Tools Query Options Window Help
Serial: 1
For Help, press F1
NUM 1:58 PM 00020 008

```
(* Dr. Whedco's example of program that will cause a fault

URA=4096        (* set axis unit ratio
CURC=75         (* set max continuous current
DIR=CW          (* Set motor direction for clockwise move

PROGRAM4                 (* start fault handling program
010 WAIT NOT(IO11)       (* wait for enable to be false
STM4=.25                 (* debounce enable off
WAIT TM4
IF IO11 GOTO 10
020 WAIT IO11            (* wait for enable to be true
STM4=.25                 (* debounce enable on
WAIT TM4
IF NOT(IO11) GOTO 20     (* reset faults
EXP1                     (* execute program 1
END                      (* end fault handling program

PROGRAM1        (* start motion program 1
VF10=0          (* initialize variable 10 to 0
PSA=0           (* set axis position to zero
MVL=10          (* set motion velocity to 10 units/sec
MAC=40          (* set acceleration to 40 units/sec
MPA=100./VF10   (* set absolute move position to 100/VF10 units
RPA             (* run to position
END             (* end program 1
```

Figure 6.4: Faulty Example IMC Application Program That Will Load into Controller but Will Not Run

When we type `EXP1` to run the program in which the error occurs (see figure 6.5):


The motor does nothing; and the Terminal window displays an *
Query fault code
Result of query
CCS for Windows - [Terminal]
File Edit View Tools Query Options Window Help
Serial: 1
For Help, press F1
NUM 1:15 PM

```
*1
*Whedco IMC DspMotion(R)
1exp1
*1fc?
*Mathematical Data Error
```

Figure 6.5: Terminal Window Displaying Results of a Program that Caused a Fault

To find the source of the error, type **FC?** to query the fault code register. As expected, the controller reports Mathematical Data Error because of our divide-by-zero operation. The following page tells you how to use the FAULT command to help pinpoint the exact location of the problem within the program.

About the Line Editor

Each DspMotion Controller has a resident line editor that gives you the means to scroll through the program that resides in your controller's memory. The line editor and the ASCII file editor are two different tools: the ***ASCII file editor*** is your tool for writing programs and saving them as .txt files; the ***line editor*** is your tool for finding bugs on-the-fly, while you are connected in real-time with your controller. The ***line editor*** scrolls through only one line of code at a time at your command—the ***ASCII file editor*** displays the entire .txt file on your screen. Any changes that you make in the line editor will not affect your master application program .txt file; but they will change the controller's program and affect the behavior of the controller.

To use the ***line editor*** to identify specific lines of defective code:

- Type `FAULT` when your program does not execute properly due to a bug.
- Type `PROGRAM`*n*, where n is the number of the program through which you wish to scroll.
- Type `MOTION`*n*, where n is the number of the motion block through which you wish to scroll.

Then use the commands in figure 6.6:

Figure 6.6: Line Editor Commands

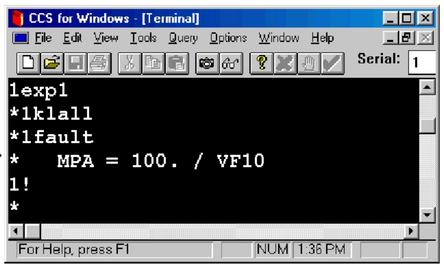
X	**Step forward**
L	**Step backward**
DEL	**Delete entire line of code**
!	**Exit line editor**
Note: The line editor will try to insert any keyboard input into the program or motion block	

Find a Bug with the FAULT Command

The FAULT command is a diagnostic tool used to find faults in the program structure. If a program operation causes a fault, FAULT gives you an online connection to the ***line editor***. Use the following procedure, shown in figure 6.7, to diagnose a fault in an application program:

1. Type `KLALL`
2. Type `FAULT`—this opens the line editor at the faulty program line.
3. Type `!` to exit the line editor

Here's the bug: Can't divide 100 by VF10 because VF10 has been set to zero!


CCS for Windows - [Terminal]
File Edit View Tools Query Options Window Help
Serial: 1
For Help, press F1
NUM 1:36 PM

```
1exp1
*1klall
*1fault
*   MPA = 100. / VF10
1!
*
```

Figure 6.7: Terminal Window Showing Use of FAULT Command to Open Line Editor at Faulty Line

Most application programs will not be as short as our example. If your program has a hundred lines, it's possible that you will issue the FAULT command, get the faulty program line displayed on your screen, and not know exactly where to find that program line to fix it.

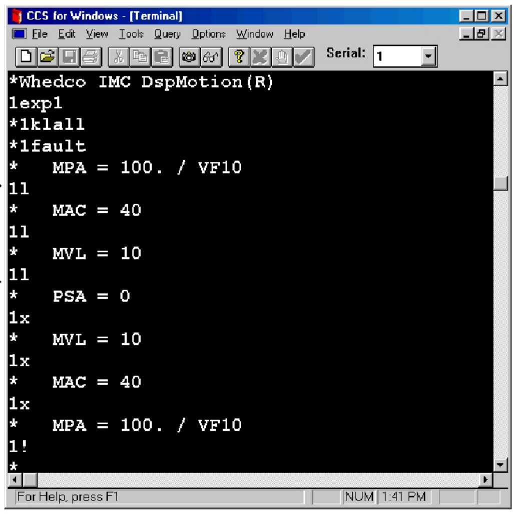
The ***line editor*** lets you step backward and forward through your program, displaying one line at a time, until you pinpoint the location of the fault.

Type **DEL** to delete any erroneous line(s).

Type **L** to step the editor backward

Type **X** to step the editor forward.

Type **!** to exit the editor.


CCS for Windows - [Terminal]
File Edit View Tools Query Options Window Help
Serial: 1
For Help, press F1
NUM 1:41 PM

```
*Whedco IMC DspMotion(R)
1exp1
*1klall
*1fault
*   MPA = 100. / VF10
1l
*   MAC = 40
1l
*   MVL = 10
1l
*   PSA = 0
1x
*   MVL = 10
1x
*   MAC = 40
1x
*   MPA = 100. / VF10
1!
*
```

Caution: The line editor will try to insert any keyboard input into the program!

Figure 6.8: Scrolling through a Program in the Line Editor

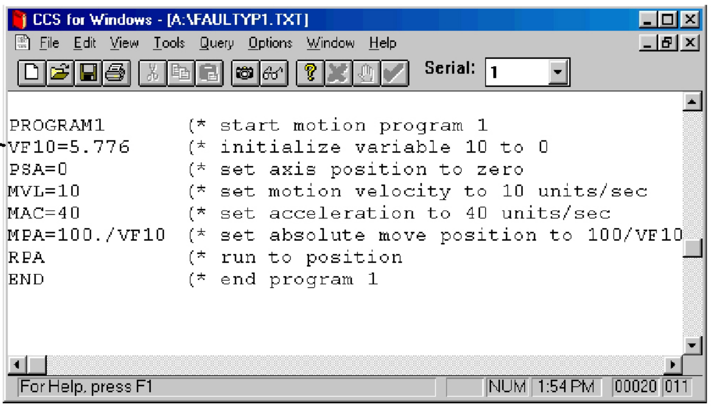
Scroll until you have found a familiar reference point—when you know exactly where that fault exists in your application program, it's time to exit the DspMotion Controller's ***line editor*** and fix the bug using the CCS *ASCII **file editor***.

Fix a Bug

To correct the fault in your program, you must open the original, master .txt file in the CCS ***ASCII file editor***. Complete the following steps:

1. Click **File**/**Open**
2. Click on your application program's .txt file name
3. Click **OK**
4. Click to place your cursor at the faulty program line
5. Correct your text

VF10 changed from 0 to 5.776


CCS for Windows - [A:\FAULTYP1.TXT]
File Edit View Tools Query Options Window Help
Serial: 1
For Help, press F1
NUM 1:54 PM 00020 011

```
PROGRAM1      (* start motion program 1
VF10=5.776    (* initialize variable 10 to 0
PSA=0         (* set axis position to zero
MVL=10        (* set motion velocity to 10 units/sec
MAC=40        (* set acceleration to 40 units/sec
MPA=100./VF10 (* set absolute move position to 100/VF10
RPA           (* run to position
END           (* end program 1
```

Figure 6.9: Correcting a Program Bug in the ASCII File Editor

6. Click **File**/**Save**
7. Click **File**/**Close** to exit the ASCII file editor and return to the Terminal window

From the Terminal window, send the corrected .txt file to the controller:

1. Type `STF` (i.e., for the IMC) or `STFALL` (i.e., for the Target ARS)
2. Type `KLALL`
3. Type `UPS=0` (UPS must be set to its default value of zero before the CLM command will work.)
4. Type `CLM` (remember that CLM will clear your system constants! You'll have to reset them if you have not included them in your .txt file.)
5. Click **Tools**/**Send Files**
6. Click on your application program's .txt file name
7. Click **OK**

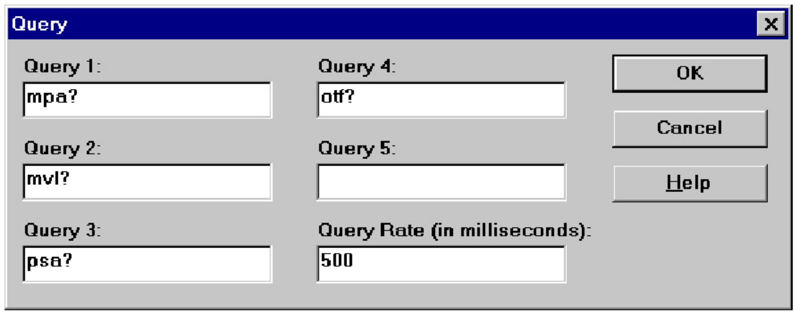
Run the program with your correction:

1. Type `EXP`*`n`* (n is the number of your program)

Monitor Real-time Machine Parameters with Query/Start (Q, ?)

Use query to monitor almost any machine parameter while your system is executing an application program. **Query**/**Start** provides you with a constant update of changing speeds, positions, and almost any other condition that you might want to track and evaluate.

1. Click **Query**/**Start**.
2. Enter the registers you wish to query, followed by a ? or Q—one register per field. You can query up to five registers simultaneously. You may also change the speed of the query update (e.g., 50 ms).



Query 1: mpa?
Query 2: mvl?
Query 3: psa?
Query 4: otf?
Query 5:
Query Rate (in milliseconds): 500

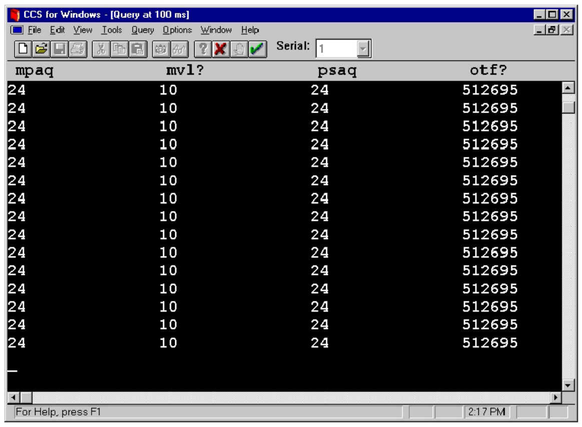
Figure 6.10: Query Screen with Four IMC Register Queries Entered

3. Click **OK**—real-time values for the registers you have chosen will report on-screen (see figure 6.11).

Screen Registers Shown:

MPA:Absolute Move Position

MVL:Motion Velocity


CCS for Windows - [Query at 100 ms]
File Edit View Tools Query Options Window Help
Serial: 1
For Help, press F1
2:17 PM

```
mpaq               mvl?               psaq               otf?
24                 10                 24                 512695
24                 10                 24                 512695
24                 10                 24                 512695
24                 10                 24                 512695
24                 10                 24                 512695
24                 10                 24                 512695
24                 10                 24                 512695
24                 10                 24                 512695
24                 10                 24                 512695
24                 10                 24                 512695
24                 10                 24                 512695
24                 10                 24                 512695
24                 10                 24                 512695
24                 10                 24                 512695
24                 10                 24                 512695
```

Screen Registers Shown:

PSA:Axis Position

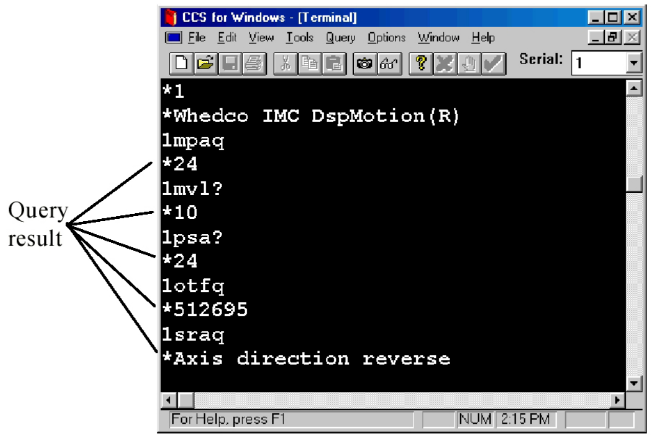
OTF:Forward Software Overtravel

Figure 6.11: Results of IMC Register Queries

To stop the query, click **Query**/**End**; to pause, click **Query**/**Pause** and **Query**/**Resume**.

Query Registers for Moment-in-Time Data (Q, ?)

You can also use the ? and Q commands in Terminal window to query and report a register value at a moment in time. The ? and Q commands report a one-time register value, rather than give you a continuous update like the Query/Start function does. In the following example, the user has queried several registers:


CCS for Windows - [Terminal]
File Edit View Tools Query Options Window Help
Serial: 1
For Help, press F1
NUM 2:15 PM
Query result

```
*1
*Whedco IMC DspMotion(R)
1mpaq
*24
1mvl?
*10
1psa?
*24
1otfq
*512695
1sraq
*Axis direction reverse
```

Figure 6.12: Results of Register Queries in the Terminal Window

Run an Application Program in Single-Step Mode

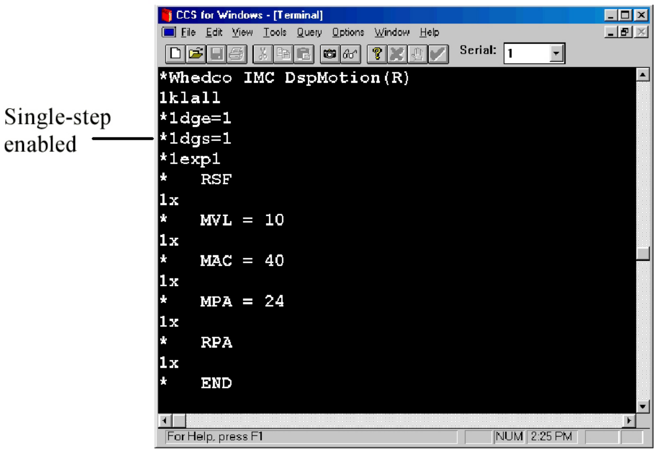
Single-step mode is another tool for diagnosing program conditions. With single-step mode enabled, you execute one line of a program at a time using the **X** command. You may use a number *n* with the X command (i.e., **X*n***), to step through *n* lines of the program.

***Note**: You can place only one program at a time in single-step mode.*

To enable single-step mode from the CCS Terminal window:

1. Type `KLALL`
2. Type `DGE=1`
3. Type `DGS= p1`, where *p1* is the program number
4. Type `EXPn` to execute program (in this example, we'll use program 1)

Single-step through the program until you execute line END.


Single-step enabled
CCS for Windows - [Terminal]
File Edit View Tools Query Options Window Help
Serial: 1
For Help, press F1
NUM 2:25 PM

```
*Whedco IMC DspMotion(R)
1klall
*1dge=1
*1dgs=1
*1exp1
*    RSF
1x
*    MVL = 10
1x
*    MAC = 40
1x
*    MPA = 24
1x
*    RPA
1x
*    END
```

Figure 6.13: Executing a Program in Single-Step Mode from the Terminal Window

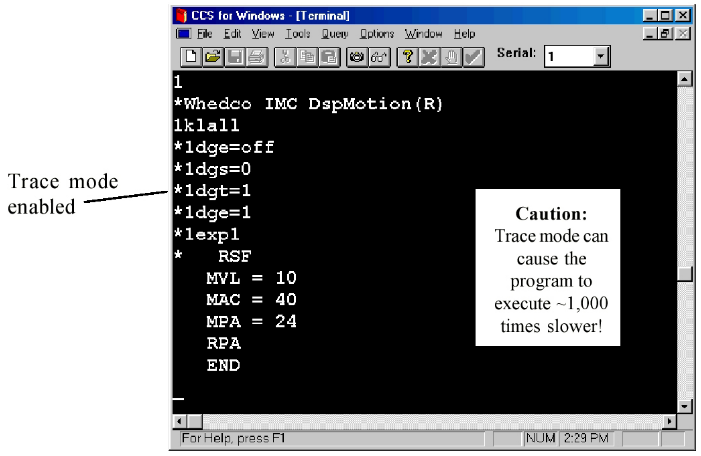
Run an Application Program in Trace Mode

Trace mode outputs one program line at a time to the Terminal window as the program is executing. No X command input is required.

Note: *Only one program at a time can be in trace mode.*

To enable trace mode:

1. Type `KLALL`
2. Type `DGE=OFF`
3. Type `DGS=0`
4. Type `DGT=p1`, where *p1* is the program number
5. Type `DGE=1`
6. Type `EXPp1`, where *p1* is the program number


Trace mode enabled
CCS for Windows - [Terminal]
File Edit View Tools Query Options Window Help
Serial: 1
For Help, press F1
NUM 2:29 PM

```
1
*Whedco IMC DspMotion(R)
1klall
*1dge=off
*1dgs=0
*1dgt=1
*1dge=1
*1exp1
*   RSF
   MVL = 10
   MAC = 40
   MPA = 24
   RPA
   END
_
```

Caution: Trace mode can cause the program to execute ~1,000 times slower!

Figure 6.14: Executing a Program in Trace Mode from the Terminal Window

Capture an Online Terminal Session

You can capture any data from your CCS Terminal window and save it as a .txt file. Use the capture feature to record any register queries, line editor activity, or diagnostic output that you want to review.

The following example opens a capture file to document program-debugging activity. Use this procedure to capture any Terminal window activity:

1. Click **Tools/Open Capture File**
2. Name the file (e.g., *capt1.txt*)
3. Click **OK**.

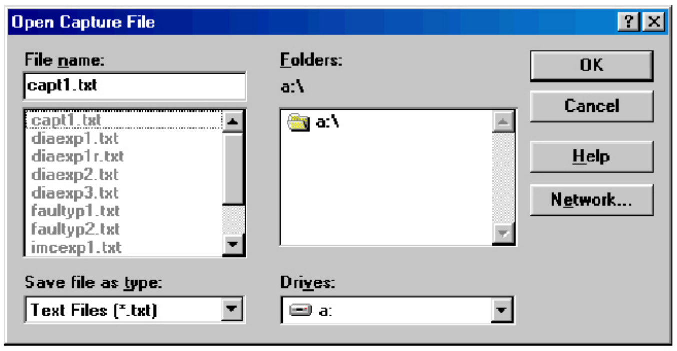
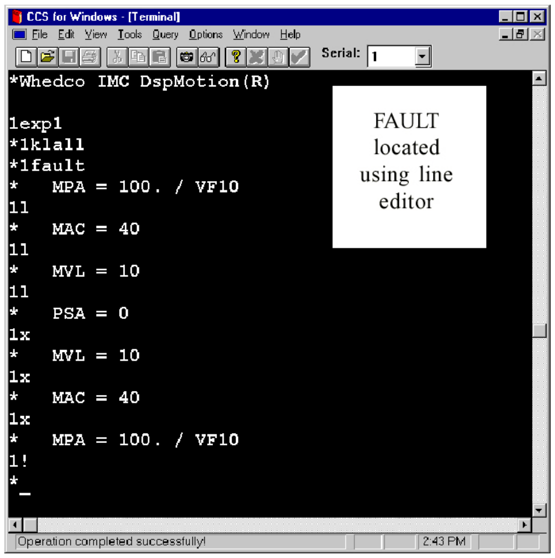


Figure 6.15: Opening a Capture File

Activity in the Terminal window will now be recorded in the capture file until you close the capture session. In figure 6.16, we have determined the location of a bug in relation to the rest of the program.

To close the capture session and review the captured data:

1. Click **Tools/Close Capture File**
2. Click **File/Open**, select the file name, and click **OK**

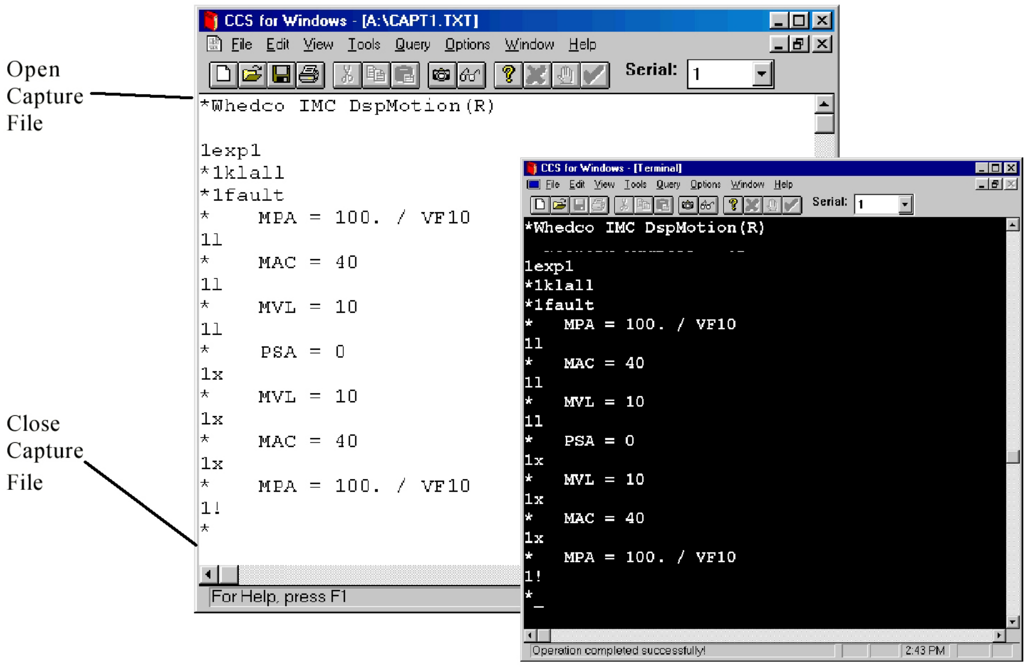

CCS for Windows - [Terminal]
File Edit View Tools Query Options Window Help
Serial: 1
Operation completed successfully!
2:43 PM

```
*Whedco IMC DspMotion(R)

lexp1
*1klall
*1fault
*    MPA = 100. / VF10
1l
*    MAC = 40
1l
*    MVL = 10
1l
*    PSA = 0
1x
*    MVL = 10
1x
*    MAC = 40
1x
*    MPA = 100. / VF10
1!
*_
```

FAULT located using line editor

Figure 6.16: Scrolling Through Program Lines in the Line Editor

The ASCII file editor opens and displays the selected file. Compare the original Terminal window in figure 6.17 (below-right) with the .txt file (below-left) created from the capture session—the .txt file shows all Terminal window activity between the time we opened and closed the capture file.


Open Capture File
Close Capture File
CCS for Windows - [A:\CAPT1.TXT]
File Edit View Tools Query Options Window Help
Serial: 1
For Help, press F1
CCS for Windows - [Terminal]
Operation completed successfully!
2:43 PM

```
*Whedco IMC DspMotion(R)

lexp1
*1klall
*1fault
*    MPA = 100. / VF10
1l
*    MAC = 40
1l
*    MVL = 10
1l
*    PSA = 0
1x
*    MVL = 10
1x
*    MAC = 40
1x
*    MPA = 100. / VF10
1!
*
```

Figure 6.17: Reviewing Data in a Capture File

Chapter 7 Receiving Data from a DspMotion Controller to Your PC

In This Chapter

- ❑ Overview
- ❑ Receive variables
- ❑ Receive all.

Overview

CCS lets you receive data from your DspMotion Controller to your PC using the **Tools** menu. This function provides a quick and easy way to sift out a particular set of data from your application program and then review and make changes to it in the ASCII file editor. You can then use the **Tools**/**Send Files** option to send your revised data to your controller.

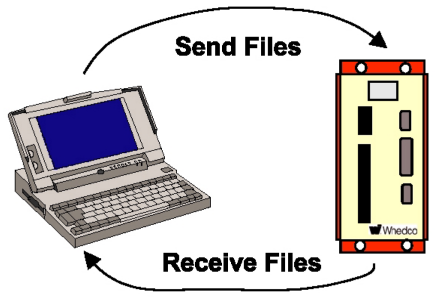
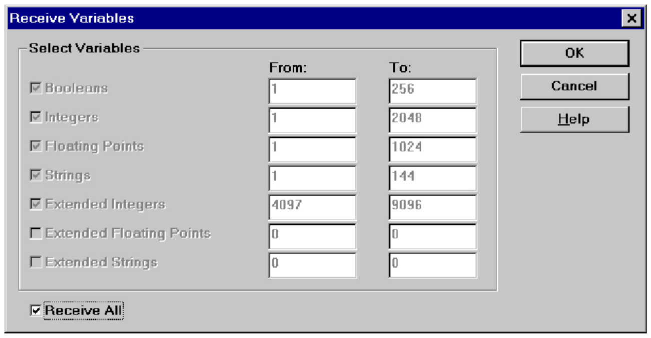


Figure 7.1: Sending and Receiving Data between Your DspMotion Controller and Your PC

Receive Variables

In the following example, we will receive a range of variables from our controller. Receiving variables may take several minutes—to make the process as efficient as possible, select only the range of variables that you really need.

1. Click **Tools/Receive Variables** to open the screen in figure 7.2.



	From:	To:
☑ Booleans	1	256
☑ Integers	1	2048
☑ Floating Points	1	1024
☑ Strings	1	144
☑ Extended Integers	4097	9096
☐ Extended Floating Points	0	0
☐ Extended Strings	0	0

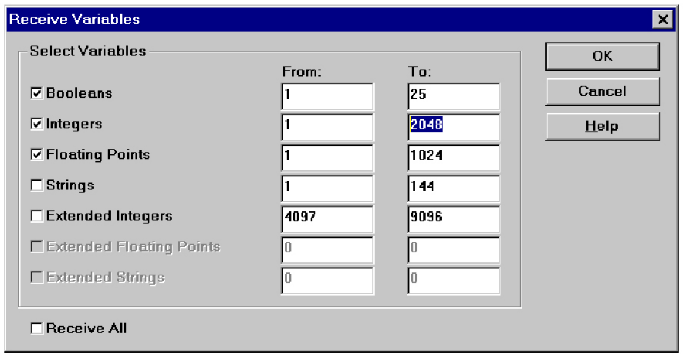
☑ Receive All

Figure 7.2: CCS *Receive All* Variables Screen

2. Click the checked **Receive All** box to deselect it.

3. Select the variables that you want to receive (in this example, we have selected Booleans, Integers, and Floating Points).

4. Edit the range of variables that you want to receive in the ***From:*** and ***To:*** fields (just click in those fields and type). Use these fields to eliminate any values that you don't need to minimize file receive time.



	From:	To:
☑ Booleans	1	25
☑ Integers	1	2048
☑ Floating Points	1	1024
☐ Strings	1	144
☐ Extended Integers	4097	9096
☐ Extended Floating Points	0	0
☐ Extended Strings	0	0

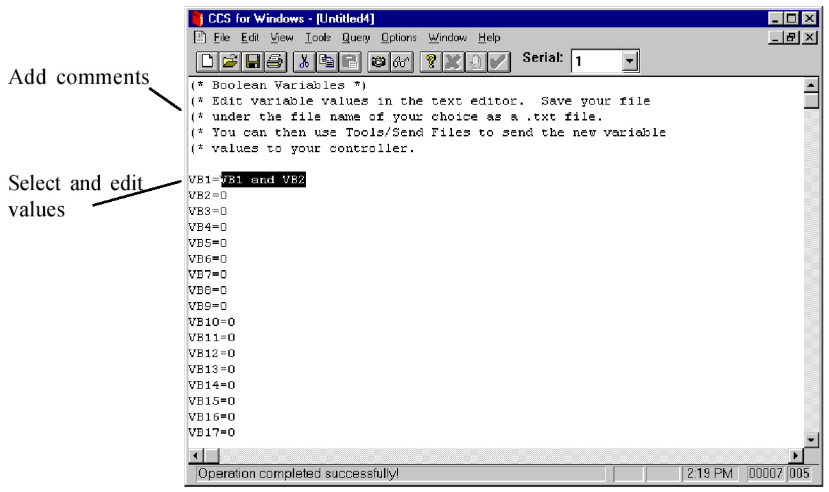
☐ Receive All

Figure 7.3: CCS Receive *Select Variables* Screen

5. Click **OK**

6. Wait for your PC to receive the variables.

When the receive process is complete, CCS opens your ASCII file editor and displays the values in an untitled file. You can add tab-delimited comments, edit the data, and save your changes under a new file name in .txt format.


Add comments
Select and edit values
CCS for Windows - [Untitled4]
File Edit View Tools Query Options Window Help
Serial: 1

```
(* Boolean Variables *)
(* Edit variable values in the text editor.  Save your file
(* under the file name of your choice as a .txt file.
(* You can then use Tools/Send Files to send the new variable
(* values to your controller.

VB1=VB1 and VB2
VB2=0
VB3=0
VB4=0
VB5=0
VB6=0
VB7=0
VB8=0
VB9=0
VB10=0
VB11=0
VB12=0
VB13=0
VB14=0
VB15=0
VB16=0
VB17=0
```

```
Operation completed successfully!    2:19 PM   00007  005
```

Figure 7.4: Received Data Displayed in the ASCII File Editor

The process for receiving variables also works for receiving registers, programs, motion blocks, or screens. Just click the appropriate option under the **Tools** menu.

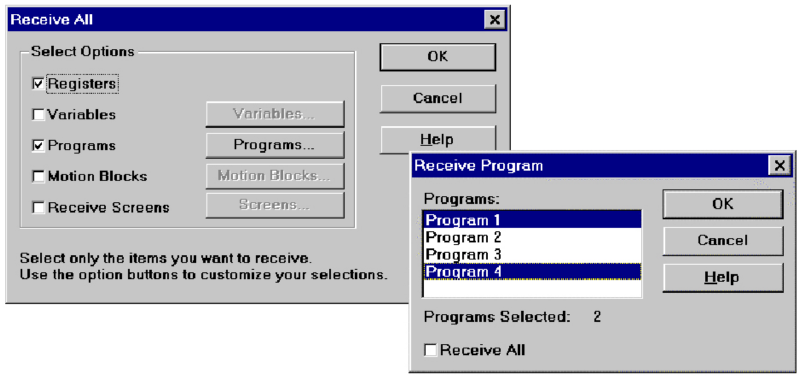
Receive All

From the **Tools** menu, you can click **Receive All** and then *deselect* any file types that you don't want. Receiving ***All*** may take several minutes—to make the process as efficient as possible, select only those options that you really need. In the following example, we have chosen to receive all registers and programs. Note that you cannot select particular registers—you get all or none. You can, however, select particular variables, programs, motion blocks, and screens when you Receive All. Use the following procedure to customize the **Receive All** options:

1. Click **Tools/Receive All**

2. Click to select the options you want (in this example, we'll receive *Registers* and *Programs*)

3. Click **Programs** to select particular programs (we have selected *Program 1* and *Program 4*)



Select Options
- [✓] Registers
- [ ] Variables
- [✓] Programs
- [ ] Motion Blocks
- [ ] Receive Screens

Select only the items you want to receive.
Use the option buttons to customize your selections.

Programs:
- [✓ Program 1]
- Program 2
- Program 3
- [✓ Program 4]

Programs Selected: 2

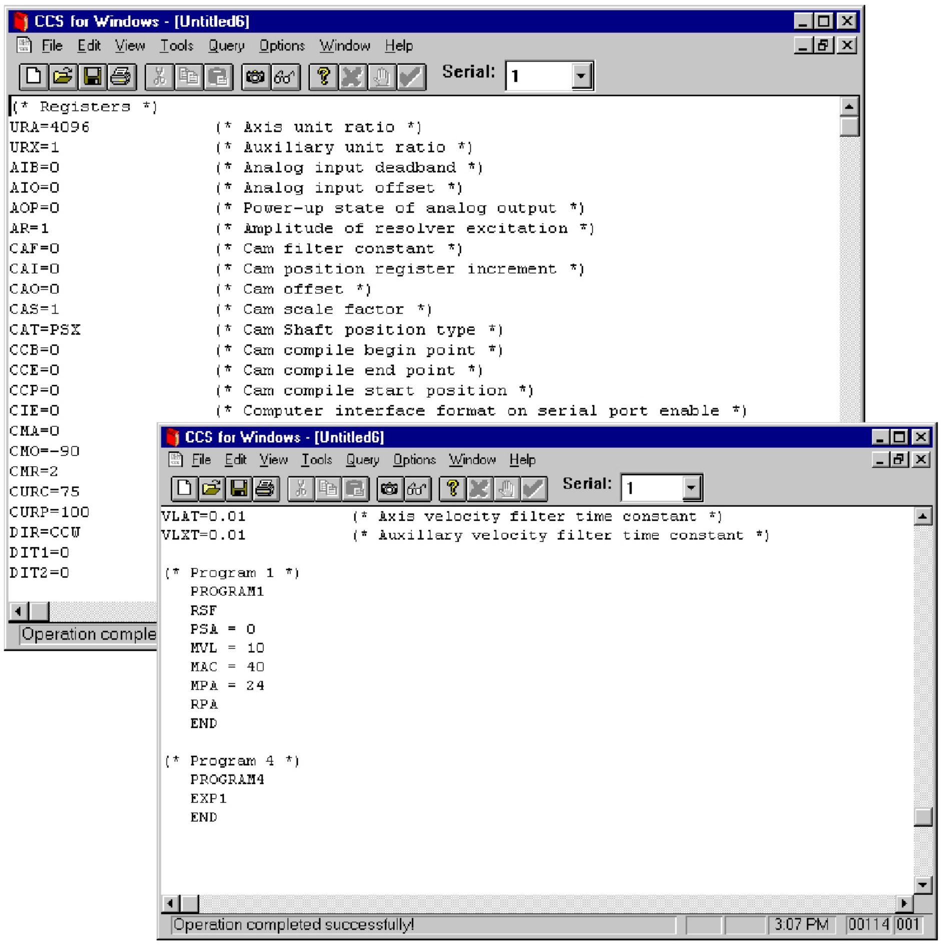
- [ ] Receive All

Figure 7.5: CCS Tools/Receive All Screens

4. Click **OK.**

> ***Note:*** *If you have enabled the SECURE feature, you must type CLM before your controller will receive any programs or motion blocks.*

When the receive process is complete, CCS opens your ASCII file editor and displays the values in an untitled file (see figure 7.6). You can add tab-delimited comments, edit the data, and save your changes under a new file name in the .txt format.


CCS for Windows - [Untitled6]
File Edit View Tools Query Options Window Help
Serial: 1

```
(* Registers *)
URA=4096          (* Axis unit ratio *)
URX=1             (* Auxiliary unit ratio *)
AIB=0             (* Analog input deadband *)
AIO=0             (* Analog input offset *)
AOP=0             (* Power-up state of analog output *)
AR=1              (* Amplitude of resolver excitation *)
CAF=0             (* Cam filter constant *)
CAI=0             (* Cam position register increment *)
CAO=0             (* Cam offset *)
CAS=1             (* Cam scale factor *)
CAT=PSX           (* Cam Shaft position type *)
CCB=0             (* Cam compile begin point *)
CCE=0             (* Cam compile end point *)
CCP=0             (* Cam compile start position *)
CIE=0             (* Computer interface format on serial port enable *)
CMA=0
CMO=-90
CMR=2
CURC=75
CURP=100
DIR=CCW
DIT1=0
DIT2=0
```

Operation comple

```
VLAT=0.01         (* Axis velocity filter time constant *)
VLXT=0.01         (* Auxiliary velocity filter time constant *)

(* Program 1 *)
   PROGRAM1
   RSF
   PSA = 0
   MVL = 10
   MAC = 40
   MPA = 24
   RPA
   END

(* Program 4 *)
   PROGRAM4
   EXP1
   END
```

Operation completed successfully! 3:07 PM 00114 001

Figure 7.6: ASCII File Editor Windows Displaying Registers and Programs Received

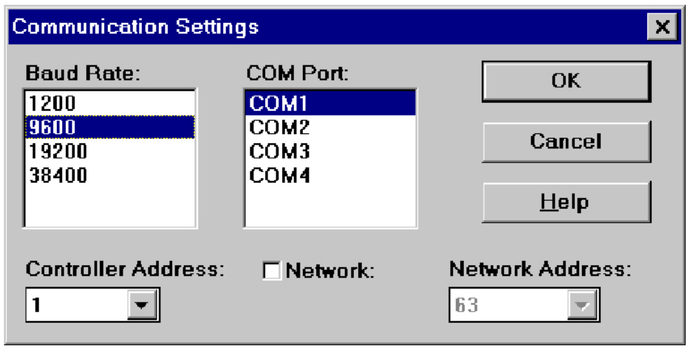
Chapter 8 Troubleshooting

My Controller Doesn't Communicate

Probable Fix for the IMC: Check Communication Configuration

Check the software and controller communication configuration. CCS has three communication settings that must match the controller hardware settings: controller address, communication baud rate, and computer communication port address. To change these settings in CCS:

1. Click **Options/Communication Settings**.



Communication Settings

Baud Rate: 1200, 9600 (selected), 19200, 38400

COM Port: COM1 (selected), COM2, COM3, COM4

Controller Address: 1

Network: ☐

Network Address: 63

Figure 8.1: Communication Settings in CCS for Windows

2. Check the controller address and communication baud rate set on the controller:
   a. Locate the DIP switches on the bottom of the controller.
   b. The factory default settings are 9600 baud communication rate and controller address 1. (See the ADDS and BAUD registers in Appendix A for applicable DIP switch settings.)
   c. To change the DIP switches, turn off the power to the controller. Change the switches and restore controller power.
3. Check the comms port setting.
   a. Check your computer to determine which port you are using.
   b. Make sure the software setting is the same.

4. Examine the cable orientation—one end of the cable is labeled IMC/OIP, while the other end is labeled PC. Verify that the cable ends are connected to the proper equipment.
5. When settings match, press <Enter> twice to verify communication.

Probable Fix for the Target

Check cable connections. The Target ARS autoconfigures your controller address, baud rate, and communications port—those settings should be correct unless you have altered them.

Operator Interface Panel Displays Meaningless Information

If you send characters faster than the buffer can receive them, the receive buffer will overflow—hence, junk on the display. Turn on the UPS input/output register to refresh the screen data automatically every ¼ second.

Before you enter any new text, turn off UPS by setting **`UPS=0`**.

When I Enable the Servo Drive, My Motor Jumps and Then Faults

Your motor could jump and then fault if your motor constants aren't set correctly. Make sure the motor is not connected to a load, and then use MOTORSET to set up motor constants automatically for CMO, CMR, and/or AR.

You can also manually set CMO and CMR to the following values:

For N-Series motors:	CMO = 90
	CMR = 3

For S-Series motors:	CMO = -90
	CMR = 2

Another possible fix is to use the AUTOTUNE system command to reset the control constants KA, KD, KI, KP, and KT automatically. Before you issue the AUTOTUNE command, verify that the following conditions exist:

- The system and axis are faulted.
- No programs are executing.
- Motor constants are set.
- The motor is connected to the load.
- The axis is free to move ½ revolution in the forward direction.

Where are My (* Delimited Comments?

DspMotion controllers ignore and do not store any (* delimited comments contained within an application program, so when you send a program to or receive a program from your controller, the comments do not go along for the ride. If you want to embed comments within a program and have them stored in controller memory, use the REM command (see Appendix A).

I Forgot/Lost the Password!

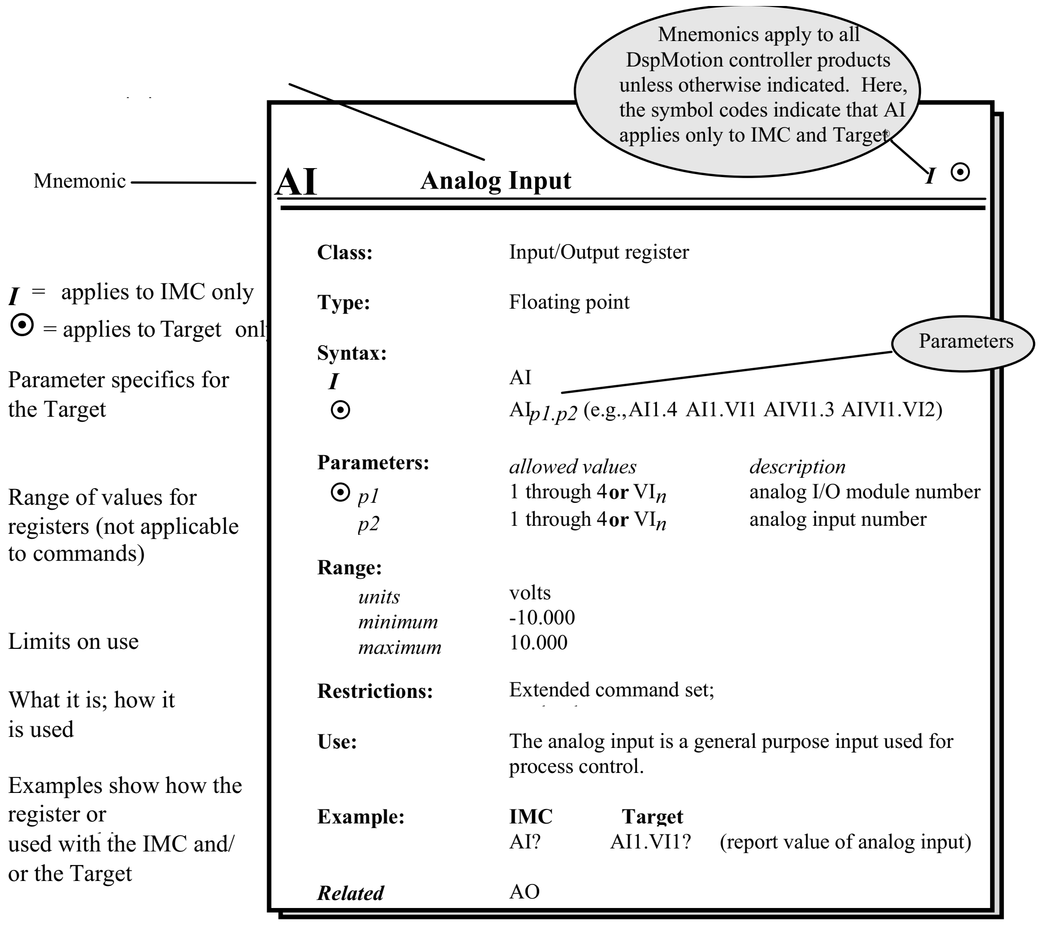
If you lose or forget your password, contact GE Fanuc Customer Care at 1-800-433-2682 to get a return merchandise authorization (RMA) number.

My Controller Is Not Faulted, But the Motor Will Not Move!

Your system is looking for normally closed contacts on +/- overtravels. Put in normally closed contacts or hardwire overtravels *on*.

Appendix A Registers and Commands

The registers and commands in Appendix A are alphabetized and formatted according to the template shown below. Note that not all of the fields (i.e., *Type*, *Restrictions*, etc.) apply to every register and command.



Mnemonics apply to all DspMotion controller products unless otherwise indicated. Here, the symbol codes indicate that AI applies only to IMC and Target
Mnemonic
I = applies to IMC only
⊙ = applies to Target onl
Parameter specifics for the Target
Parameters
Range of values for registers (not applicable to commands)
Limits on use
What it is; how it is used
Examples show how the register or used with the IMC and/ or the Target

AI **Analog Input** *I* ⊙

Class:	Input/Output register		
Type:	Floating point		
Syntax:			
I	AI		
⊙	AI$_{p1.p2}$ (e.g.,AI1.4 AI1.VI1 AIVI1.3 AIVI1.VI2)		
Parameters:	*allowed values*	*description*	
⊙ *p1*	1 through 4 **or** VI$_n$	analog I/O module number	
p2	1 through 4 **or** VI$_n$	analog input number	
Range:			
units	volts		
minimum	-10.000		
maximum	10.000		
Restrictions:	Extended command set;		
Use:	The analog input is a general purpose input used for process control.		
Example:	**IMC**	**Target**	
	AI?	AI1.VI1?	(report value of analog input)
Related	AO		

! Exits Line Editor

Class: Program Command

Syntax: !

Restrictions: Allowed only in programs or motion blocks.

Use: This command exits the line editor.

Examples:

IMC/IMJ	Target ARS	
PROGRAM1	PROGRAM1	(* edit program 1)
* PSA=0	* PSA1=0	
X	X	(* step through program)
* MAC=10	* MAC1=10	
!	!	(* exit line editor)
*	*	

Related Commands: PROGRAM, END, MOTION

? Reports Value of Register

Class:	Diagnostic Command	
Syntax:	*p1*? (e.g., CURC? SRS? PSAVI1? MPA2?)	
Parameters:	*allowed values*	*description*
p1	any register	register
Restrictions:	Not allowed in programs or motion blocks.	
Use:	This command is used to report the value of any register. It is identical to the Q command.	
Related Commands:	DGO, Q	

ADDR Address of RTU Port *I jr*

Class:	System Register
Type:	Integer
Syntax:	ADDR
Range:	
default	1
minimum	1
maximum	247
Restrictions:	Cannot be assigned in motion blocks. Available in IMJ firmware 2.2 and higher; IMC firmware 3.2 and higher.
Use:	The address of the RTU port is a number used to identify the RTU port.
Related Registers:	RTU

ADDS Address of Serial Port *I*

Class: System Register

Type: Integer

Syntax: ADDS

Range:

default	set by DIP switch
minimum	0
maximum	31

Restrictions: Cannot be assigned in programs or motion blocks.

Use: The address of the serial port is a number used to identify the serial port.

Remarks: If DIP switch 8 is set to the left, the ADDS register value defaults to 1 on power-up. If, however, DIP switch 8 is set to the right, DIP switches 1–5 determine the serial port address from 0 through 31. The table below shows which DIP switch setting is to be used for a specific address.

DIP Switch Settings for Unit Addresses

	Switch Locations						**Switch Locations**				
Address	**1**	**2**	**3**	**4**	**5**	**Address**	**1**	**2**	**3**	**4**	**5**
0	R	R	R	R	R	16 (G)	R	R	R	R	L
1	L	R	R	R	R	17 (H)	L	R	R	R	L
2	R	L	R	R	R	18 (I)	R	L	R	R	L
3	L	L	R	R	R	19 (J)	L	L	R	R	L
4	R	R	L	R	R	20 (K)	R	R	L	R	L
5	L	R	L	R	R	21 (L)	L	R	L	R	L
6	R	L	L	R	R	22 (M)	R	L	L	R	L
7	L	L	L	R	R	23 (N)	L	L	L	R	L
8	R	R	R	L	R	24 (O)	R	R	R	L	L
9	L	R	R	L	R	25 (P)	L	R	R	L	L
10 (A)	R	L	R	L	R	26 (Q)	R	L	R	L	L
11 (B)	L	L	R	L	R	27 (R)	L	L	R	L	L
12 (C)	R	R	L	L	R	28 (S)	R	R	L	L	L
13 (D)	L	R	L	L	R	29 (T)	L	R	L	L	L
14 (E)	R	L	L	L	R	30 (U)	R	L	L	L	L
15 (F)	L	L	L	L	R	31 (V)	L	L	L	L	L

AI Analog Input *I* 🗎

Class:	Input/Output Register	
Type:	Floating point	
Syntax:		
I	AI	
⊙	AI*p1.p2* (e.g., AI1.4 AI1.VI1 AIVI1.3 AIVI1.VI2)	
Parameters:	allowed values	description
⊙ *p1*	1 through 4 **or** VI*n*	analog module number
p2	1 through 4 **or** VI*n*	analog input number
Range:		
units	volts	
minimum	-10.000	
maximum	10.000	
Restrictions:	For IMCs, this function available only with the extended command set; read only.	
Use:	The analog input is a general purpose input used for process control. AI defines the value in volts of the hardware analog inputs.	

Examples:

IMC	Target ARS	
AI?	AI1.VI1?	(* report value of analog input)

Related Registers: AO

AI*p1* Analog Input *jr*

Type:	Floating Point
Syntax:	AI*p1*
Parameters	*allowed values*
p1	1 or 2 (analog input number)
Range:	
units	volts
minimum	-10.000
maximum	10.000
Restrictions:	Read only.
Use:	The analog input is a general purpose input used for process control. AI defines the value in volts of one of the two hardware analog inputs.
Examples:	
AI1?	(* report value of analog input one)
AI2?	(* report value of analog input two)
Related Registers:	AO

AIB Analog Input Deadband *I* 🗎

Class:	Input/Output Register	
Type:	Floating point	
Syntax:		
I	AIB	
⊙	AIB*p1.p2* (e.g., AIB1.4 AIB1.VI1 AIBVI1.3 AIBVI1.VI2)	
Parameters:	*allowed values*	*description*
⊙ *p1*	1 through 4 **or** VI*n*	analog module number
p2	1 through 4 **or** VI*n*	analog input number
Range:		
units	volts	
default	0	
minimum	0	
maximum	10.000	

Restrictions: For IMCs, this function available only with the extended command set; cannot be assigned in motion blocks.

Use: The analog input deadband defines a range over which the analog input remains constant at 0 volts. When the analog input, AI, is less than or equal to AIB, the analog input is set to 0.

Examples:

IMC	**Target ARS**	
AIB=1.5	AIB1.2=1.5	(* set analog input deadband equal to 1.5 V)
AIB?	AIB1.VI1?	(* report value of analog input deadband)

Related Registers: AI

AIB*p1* Analog Input Deadband *jr*

Type:	Floating Point
Syntax:	AIB*p1* (e.g., AIB1 AIB2)
Parameters	*allowed values*
p1	1 or 2 (analog input number)
Range:	
units	volts
default	0
minimum	0
maximum	10.000
Restrictions:	Cannot be assigned in motion blocks.
Use:	The analog input deadband defines a range over which the analog input remains constant at 0 volts. When the analog input AI1 is less than or equal to AIB1, the analog input is set to 0. When the analog input AI2 is less than or equal to AIB2, the analog input is set to 0.
Examples:	
AIB2=1.5	(* set analog input deadband equal to 1.5 V)
AIB2?	(* report value of analog input deadband)
Related Registers:	AI*p1*

AIF Analog Input Filter Frequency

Class:	Input/Output Register	
Type:	Floating point	
Syntax:	AIF*p1.p2* (e.g., AIF1.4 AIF1.VI1 AIFVI1.3 AIFVI1.VI2)	
Parameters:	*allowed values*	*description*
p1	1 through 4 **or** VI*n*	analog module number
p2	1 through 4 **or** VI*n*	analog input number
Range:		
units	Hertz	
default	1,000	
allowed values	10; 20; 50; 100; 200; 500; 1,000	
Restrictions:	Cannot be assigned in motion blocks.	
Use:	The analog input filter frequency is the cutoff frequency of the lowpass filter of the analog input. Basically, any frequencies above the cutoff frequency defined by AIF are filtered out.	
Example:		
AIF1.2=200	(* set analog input filter frequency for input two of analog module one equal to 200 Hertz)	
AIF1.VI1?	(* report value of analog input filter frequency of analog input VI1 of analog module one)	
Related Registers:	AI	

AIO Analog Input Offset *I* 🗎

Class:	Input/Output Register	
Type:	Floating point	
Syntax:		
I	AIO	
⊙	AIO *p1.p2* (e.g., AIO1.4 AIO1.VI1 AIOVI1.3 AIOVI1.VI2)	
Parameters:	*allowed values*	*description*
⊙ *p1*	1 through 4 **or** VI*n*	analog module number
p2	1 through 4 **or** VI*n*	analog input number
Range:		
units	volts	
default	0	
minimum	-10.000	
maximum	10.000	

Restrictions: For IMCs, this function available only with the extended command set; cannot be assigned in motion blocks.

Use: The analog input offset is used to add a voltage offset to the analog input.

Examples:

IMC	Target ARS	
AIO=2.5	AIO1.2=2.5	(* set analog input offset equal to 2.5 V)
AIO?	AIO1.VI1?	(* report value of analog input offset)

Related Registers: AI

AIO*p1* Analog Input Offset *jr*

Class:	Input/Output Register	
Type:	Floating point	
Syntax:	AIO*p1* (e.g., AIO1 AIO2)	
Parameters	*allowed values*	*description*
p1	1 or 2	analog input number

Range:

units	volts
default	0
minimum	-10.000
maximum	10.000

Restrictions: Cannot be assigned in motion blocks.

Use: The analog input offset one, AIO1, is used to add a voltage offset to the analog input one, AI1. Analog input offset two, AIO2, is used to add a voltage offset to the analog input two, AI2.

Examples:

```
AIO1=2.5        (* set analog input offset equal to 2.5 V)
AIO1?           (* report value of analog input offset)
```

Related Registers: AI*p1*

AM Analog Module Rack Slot Assignment

Class:	System Register	
Syntax:	AM*p1* (e.g., AM2 AM3)	
Parameters:	*allowed values*	*description*
p1	1 through 4	analog module number
Range:		
default	0	
allowed values	0; 11 through 18; 21 through 28; 31 through 38	
Restrictions:	Not allowed in programs, motion blocks, or expressions.	

Use: The analog module rack slot assignment is used to define in which slot an analog module resides. The analog module rack slot assignment consists of two digits. The first digit is the rack number, and the second digit is the slot number. If AM*p1* is equal to 0, it means that analog module *p1* is not used in the system.

Remarks: To assign the analog expansion card installed in an analog module, set AM*p1* for the expansion card to the next higher slot number from the slot in which the module is installed.

Example:

AM1=17	(* set analog module rack slot assignment of analog module one to rack one, slot seven)
AM3?	(* report analog module rack slot assignment of analog module 3)

Related Registers: DM, SM, AME

AME Analog Module Assignment Error

Class:	System Register	
Type:	Integer, Boolean	
Syntax:	AME*p1* (e.g., AME AME8 AMEVI2)	
Parameters:	*allowed values*	*description*
p1	none **or** 0 through 23 **or** VI*n*	analog module assignment error register bit number
Range:		
allowed values	0 through FFFFFF$_{16}$ **or** 0 and 1	
Restrictions:	Read only.	

Use: The analog module assignment error register is used to determine if any of the analog modules are not properly assigned by the system.

Remarks: 1. When the AME? command is executed, the module assignment error register value will be given as an English statement. If all analog module assignments are correct, the message given is *All module assignments are correct.*
2. If the computer interface format is enabled, and the AME? command is executed, the module assignment error register value will be given as an integer number. If all analog module assignments are correct, the module assignment error register is set to 0. The possibilities are listed below:

bit	*message*
0	Module in rack one, slot one did not respond to assignment
1	Module in rack one, slot two did not respond to assignment
2	Module in rack one, slot three did not respond to assignment
3	Module in rack one, slot four did not respond to assignment
4	Module in rack one, slot five did not respond to assignment
5	Module in rack one, slot six did not respond to assignment
6	Module in rack one, slot seven did not respond to assignment
7	Module in rack one, slot eight did not respond to assignment
8	Module in rack two, slot one did not respond to assignment
...	...
...	...
22	Module in rack three, slot seven did not respond to assignment
23	Module in rack three, slot eight did not respond to assignment

AO Analog Output

Class:	Input/Output Register	
Type:	Floating point	
Syntax:		
I, *jr*	AO	
⊙	AO*p1.p2* (* e.g., AO1.4 AO1.VI1 AOVI1.3 AOVI1.VI2)	
Parameters:	*allowed values*	*description*
⊙ *p1*	1 through 4 **or** VI*n*	analog module number
p2	1 through 4 **or** VI*n*	analog output number
Range:		
I, *jr* *units*	volts	
default	0	
allowed values	-10.000 through 10.000 VLA (velocity of axis) CMD (control output) FE (following error)	
⊙ *units*	volts	
default	0	
minimum	-10.000	
maximum	10.000	
Restrictions:	Brushless servo and stepper only.	
Use:	The analog output is a general purpose output used for process control.	
Remarks:	Setting the analog output to VLA, CMD, or FE enables the analog output to assume a value based on the following: VLA (10 Volts = 20 Krpm); CMD (10 Volts = maximum peak rating of drive); *I*, ⊙ FE (10 Volts = 2,048 pulses of following error); *jr* FE (10 Volts = 128 pulses of following error).	

Examples:

IMC/IMJ	**Target ARS**	
AO=1.5	AO1.2=1.5	(* set analog output equal to 1.5 V)
AO=CMD		(* set analog output equal to control output)
AO?	AO1.VI1?	(* report value of analog output)

Related Registers: AI, AI*p1*, AOP

AOP Power-Up State of Analog Output

Class:	Input/Output Register	
Type:	Floating point	
Syntax:		
I*, *jr	AOP	
⊙	AOP*p1.p2* (e.g., AOP1.4 AOP1.VI1 AOPVI1.3 AOPVI1.VI2)	
Parameters:	*allowed values*	*description*
⊙ *p1*	1 through 4 **or** VI*n*	analog module number
p2	1 through 4 **or** VI*n*	analog output number

Range:

I*, *jr *units*	volts
default	0
allowed values	-10.000 through 10.000 VLA (velocity of axis, 10 V = 20 Krpm) CMD (control output, 10 V = maximum peak rating of drive) ***I***, ⊙ FE (following error, 10 V = 2,048 pulses of following error) ***jr*** FE (10 Volts = 128 pulses of following error).
⊙ *units*	volts
default	0
minimum	-10.000
maximum	10.000

Restrictions: Brushless servo and stepper only; not allowed in motion blocks.

Use: The power-up state of the analog output is the voltage that the analog output takes on upon system power-up.

Examples:

IMC/IMJ	Target ARS	
AOP=5	AOP1.2=5	(* set power-up state of analog output to 5 V)
AOP=FE		(* set AOP to equal following error)
AOP?	AOP1.VI1?	(* report value of power-up state of analog output)

Related Registers: AO

AR Amplitude of Resolver Excitation *I* 🗎

Class:	Axis Register	
Type:	Integer	
Syntax:		
I	AR	
⊙	AR*p1* (e.g., AR1 ARVI3)	
Parameters:	*allowed values*	*description*
⊙ *p1*	1 through 8 **or** VI*n*	axis number
Range:		
I *default*	1	
minimum	1	
maximum	4	
⊙ *default*	120	
minimum	1	
maximum	250	
Restrictions:	Resolver feedback brushless servo only.	
Use:	The amplitude of the signal needed for resolver excitation is one of the motor constants needed to operate a resolver feedback servo motor. The value of AR is determined by the transformation ratio of the resolver. In a Target system, this value can be set automatically by the MOTORSET command. In an IMC, AR=1 corresponds to a resolver transformation of ½ and AR=2 corresponds to 1. AR values 3 and 4 are reserved.	
Related Registers:	CMO, CMR	
Related Commands:	MOTORSET	

AUTORET Enables Auto Retrieving of User Memory

Class:	System Command
Syntax:	Autoret
Restrictions:	Not allowed in programs or motion blocks.
Use:	This command is used to enable auto retrieving of user memory from nonvolatile memory on power-up. This command must be included in the configuration data for the controller or the contents of the nonvolatile memory will not be restored when controller power is cycled.
Related Commands:	RETRIEVE, SAVE

AUTOTUNE Automatically Sets Up Control Constants

Class:	System Command
Syntax:	
I, jr	AUTOTUNE
⊙	AUTOTUNE*p1* (e.g., AUTOTUNE3)
Parameters:	*allowed values* / *description*
⊙ *p1*	1 through 8 / axis number
Restrictions:	Servo only; not allowed in programs or motion blocks.
Use:	This command automatically sets up the control constants, which are KA, KD, KI, KP, and KT.

Remarks:

This command will execute only when the controller or system and axis are faulted, the axis *Enable* input is true, and no programs or motion blocks are executing. The motor should be connected to the load when you use this command. When executed, it causes the axis to move half a revolution in the forward direction. Be sure that the axis is free to move this far before executing this command. This command takes about two seconds to execute; and, when finished, the controller will return either an asterisk (*) indicating successful completion or a question mark (?) followed by the appropriate error message. The possible error messages are as follows:

1. TORQUE TO INERTIA RATIO TOO LOW — the torque to inertia ratio of the axis is less than 125 radians/sec^2.
2. TORQUE TO INERTIA RATIO TOO HIGH — the torque to inertia ratio of the axis is greater than 125,000 radians/sec^2.
3. TORQUE RESPONSE NON-LINEAR — autotuning won't work.

This command will execute correctly in an ampless servo controller only if the servo controller is connected to a drive whose output current is proportional to the servo controller output voltage (+/- 10 V) and the drive produces full continuous current when the servo controller output voltage is 5.0 volts.

Related Commands: MOTORSET

Registers Used: KA, KD, KI, KP, KT, FR, CURC

AXE Axis Assignment Error

Class:	System Register	
Type:	Integer, Boolean	
Syntax:	AXE*p1* (e.g., AXE AXE8 AXEVI2)	
Parameters:	*allowed values*	*description*
p1	none **or** 0 through 7 **or** VI*n*	axis assignment error register bit number
Range:		
allowed values	0 through FF_{16} **or** 0 and 1	
Restrictions:	Read only.	

Use: The axis assignment error register is used to determine if any of the axes are not properly assigned by the system.

Remarks: 1. When the AXE? command is executed, the axis assignment error register value will be given as an English statement. If all axes are assigned correctly, the controller returns an *All axis assignments are correct* message.
2. If the computer interface format is enabled, and the AXE? command is executed, the axis assignment error register value will be given as an integer number. This number is the result of adding together all the powers of two associated with each axis status register bit equal to 1. If all axes are assigned correctly, the axis assignment error register will be set to 0. The possibilities are listed below:

bit	*message*
0	Axis one did not respond to assignment
1	Axis two did not respond to assignment
2	Axis three did not respond to assignment
3	Axis four did not respond to assignment
4	Axis five did not respond to assignment
5	Axis six did not respond to assignment
6	Axis seven did not respond to assignment
7	Axis eight did not respond to assignment

AXIS Axis Assignment

Class:	System Register	
Syntax:	AXIS*p1* (e.g., AXIS1)	
Parameters:	*allowed values*	*description*
p1	1 through 8	axis number
Range:		
default	NA	not assigned
allowed values	NA	not assigned
	SERVO	servo
	EXTERNAL	external drive
	IO	using I/O of axis only
Restrictions:	Not allowed in programs, motion blocks, or expressions.	
Use:	The axis assignment register is used to define the type of drive to which an axis is assigned.	
Related Registers:	AXE	

BAUD Baud Rate of Serial Port *I jr*

Class: System Register

Type: Integer

Syntax: BAUD

Range:

I *default*	set by DIP switch
jr *default*	9,600
I*, *jr *allowed values*	1,200; 9,600; 19,200; 38,400

Restrictions: Cannot be assigned in motion blocks.

Use: The baud rate of the serial port is the rate at which bit transfer takes place to and from the serial port.

Remarks:

I If DIP switch 8 is set to the left, the BAUD register value will default to 9,600 on power-up. If, however, DIP switch 8 is set to the right, DIP switches 6 and 7 determine the serial port baud rate as indicated in the table below.

jr The IMC*jr* sets the baud rate to 9,600 on power up. To make the baud rate a different value, put the *BAUD = n* command in a program that is executed on power up.

Related Registers: BIT, PAR

DIP Switch Setting for Baud Rate of Controller

	Switch Locations		
Baud Rate	**6**	**7**	**8**
1,200	R	R	R
9,600	L	R	R
19,200	R	L	R
38,400	L	L	R

BAUDP Baud Rate of Program Port

Class:	System Register
Type:	Integer
Syntax:	BAUDP
Range:	
default	automatically set
allowed values	1,200; 2,400; 4,800; 9,600; 19,200; 38,400
Restrictions:	Cannot be assigned in motion blocks.
Use:	The baud rate of the program port is the rate at which bit transfer takes place to and from the program port.
Related Registers:	BAUDU, BITU, PARU, PARP, BITP

BAUDU Baud Rate of User Serial Port

Class:	System Register
Type:	Integer
Syntax:	BAUDU
Range:	
default	9,600
allowed values	1,200; 2,400; 4,800; 9,600; 19,200
Restrictions:	Cannot be assigned in motion blocks.
Use:	The baud rate of the user serial port is the rate at which bit transfer takes place to and from the serial port.
Related Registers:	PARU, BITU, BAUDP, PARP, BITP

BIT Databits of Serial Port *I jr*

Class:	System Register
Type:	Integer
Syntax:	BIT
Range:	
default	7
allowed values	7, 8
Restrictions:	Cannot be assigned in motion blocks.
Use:	The databits of the serial port are the number of databits used to transfer characters to and from the serial port.
Remarks:	Setting PAR to NONE and BIT to 7 at the same time is not allowed. This register defaults to 7 on power-up.
Related Registers:	BAUD, PAR

BITP Databits of Program Port

Class:	System Register
Type:	Integer
Syntax:	BITP
Range:	
default	7
allowed values	7, 8
Restrictions:	Cannot be assigned in motion blocks.
Use:	The databits of the program port are the number of databits used to transfer characters to and from the program port.
Remarks:	Setting PARP to NONE and BITP to 7 at the same time is not allowed.
Related Registers:	PARP, BITU, PARU, BAUDU, BAUDP

BITU Databits of User Serial Port

Class:	System Register
Type:	Integer
Syntax:	BITU
Range:	
default	7
allowed values	7, 8
Restrictions:	Cannot be assigned in motion blocks.
Use:	The databits of the user serial port is the number of databits used to transfer characters to and from the user serial port.
Remarks:	Setting PARU to NONE and BITU to 7 at the same time is not allowed.
Related Registers:	PARP, BITP, PARU, BAUDU, BAUDP

BS Backspaces Cursor

Class:	Input/Output Command
Syntax:	BS
Use:	This command backspaces the cursor on the display.
Remarks:	This command is used in conjunction with the display when DSE is set to 1.
Related Commands:	CR, CRH, CRP
Related Registers:	DSE
ASCII Code:	$08

BSC Ballscrew Compensation

Class:	Axis Register	
Syntax:	BSC*p1.p2* (e.g., BSC3.5)	
Parameters:	*allowed values*	*description*
p1	1 through 8	axis number
p2	1 through 8	compensation pair number

Range:

units	seconds
default	0,0
minimum	-2,000,000,000 pulses, -10,000 pulses
maximum	2,000,000,000 pulses, 10,000 pulses

Use: Use this register to define a compensation position at up to eight different positions along the length of a ballscrew. The axis will linearly interpolate between the entered compensation points.

Remarks:

1. The compensation pairs must be loaded in numerical order with the smallest position loaded into pair one and the largest position loaded into pair eight.
2. All of the pairs must be loaded with appropriate data for the compensation to work properly.
3. If the axis is moved past the smallest or largest compensation pair position, then the compensation value is maintained at the last compensation value entered.
4. The numerical values for the default, minimum, and maximum of this register are assuming that the axis unit ratio, URA, is set at its default value of 1. If URA is set to a value other than 1, the default, minimum, and maximum values will change appropriately (see URA).

Example:

BSC1.1=5.5,.15	(* set compensation of 0.15 axis units at position 5.5 units)

Related Registers: BSE

BSE Ballscrew Compensation Enable

Class:	Axis Register	
Syntax:	BSE*p1* (e.g., BSE2)	
Parameters:	*allowed values*	*description*
p1	1 through 8	axis number
Range:		
default	0	
allowed values	0, 1	
Restrictions:	Not allowed in programs, motion blocks, or expressions.	
Use:	Use to enable/disable ballscrew compensation. If BSE*p1* is set to 1, ballscrew compensation is enabled; and if BSE*p1* is set to 0, ballscrew compensation is disabled.	
Registers Used:	BSC	

CAE Cam Enable

Class:	Motion Register	
Type:	Boolean	
Syntax:		
I, ***jr***	CAE	
⊙	CAE*p1* (e.g., CAE1 CAE245 CAEVI3)	
Parameters:	*allowed values*	*description*
⊙ *p1*	1 through 8 **or** list of numbers 1 through 8 **or** VI*n*	axis number
Range:		
default	0	
allowed values	0, 1	
Restrictions:	For IMCs, this function available only with the extended command set.	
Use:	The cam enable is used to enable cam motion. If CAE is set to 1, then cam motion is enabled; and if CAE is set to 0, it is disabled.	
Remarks:	When the cam is initially enabled (CAE=1) the controller reads the current cam master position in register CAP and generates an absolute move on the axis to its position that corresponds to that master position in the cam table. Current accel (MAC/MAP), decel (MDC/MDP) and velocity (MVL) constraints are used for this move. CAE is reset to zero when a fault occurs or the cam table is zeroed using the CAZ command.	
Registers Used:	CAM, CAO, CAP, CAS, CAF, CAI, CAR, CAT, CAZ	
Motion Templates:		
I, ***jr***, ⊙	Single-axis electronic camming	
⊙	Multi-axis synchronized electronic camming	

CAF Cam Filter Constant

Class:	Motion Register
Type:	Integer
Syntax:	CAF
Range:	
default	0
minimum	0
maximum	3
Restrictions:	For IMCs, this function available only with the extended command set.
Use:	The cam filter constant is used to smooth the motion of the axis when using cam following. A moving average filter of 1, 4, 8, or 16 past values of the cam master input is selected by the corresponding values of 0, 1, 2, or 3 for the cam filter constant.
Remarks:	As the length of the moving average filter increases, the axis will increasingly lag the correct cam position. Use as little filtering as the application will allow.

CAI Cam Position Register Increment

Class:	Motion Register
Type:	Integer
Syntax:	CAI
Range:	
units	degrees/sec
default	0
minimum	-10,000
maximum	10,000
Restrictions:	For IMCs, this function available only with the extended command set.
Use:	This register is used to define the rate at which to increment the cam position register, CAR.
Related Registers:	CAR, CAT

CAM Cam Point

Class: Motion Register

Type: Floating point

Syntax:

I, ***jr***	CAM*p1* (e.g., CAM1 CAM32.4 CAMVI4)
⊙	CAM*p1.p2* (e.g., CAM2.36.5 CAMVI4.25 CAM1.VI4)

Parameters:

	allowed values	*description*
I, ***jr*** *p1*	0.0 through 359.9	cam position in degrees
	or VI*n*	cam position in degrees times ten
⊙ *p1*	1 through 8 **or** VI*n*	axis number
p2	0.0 through 359.9	cam position in degrees
	or VI*n*	cam position in degrees times ten

Range:

units	axis units
default	0 pulses
I, ***jr*** *minimum*	-2,000,000,000 pulses
I, ***jr*** *maximum*	2,000,000,000 pulses
⊙ *minimum*	-8,000,000 pulses
⊙ *maximum*	8,000,000 pulses

Restrictions: For IMCs, this function available only with the extended command set.

Use: This register is used to define the axis **absolute** position at the specified cam master position for each point in a cam table.

Remarks:

1. The cam table comprises 3,600 cam points that are always equally spaced at 0.1 degree increments. The user may not need to enter every point in the table since the controller fills in any missing cam points by linearly interpolating between the points entered by the user.
2. The *zero cam table*, CAZ, command should be executed before a new set of cam points is entered. This command clears the cam table of all previous data points and disables the cam function (CAE=0).
3. The controller can only store one cam table at a time. If multiple tables are required, subsequent tables must be loaded after the current table execution is completed. The controller variables can be used to store additional tables.
4. The numerical values for the default, minimum, and maximum of this register are assuming that the axis unit ratio, URA, is set at its default value of 1. If URA is set to a value other than 1, the default values will be changed according to:

Minimum = -2,000,000,000 pulses/URA
Maximum = 2,000,000,000 pulses/URA

5. The axis will make an absolute move to the axis position that corresponds to the current *cam master position* (CAP) at the instant the cam is enabled (CAE=1 is executed).
6. The *cam scale factor* (CAS) command is used to scale the magnitude of every axis position value in the cam table. The programmer must ensure that all cam points multiplied by the *cam scale factor* (CAS) are within the settings for the software overtravel limits (OTR and OTF) as follows:

OTR <= CAM*CAS <= OTF

7. The cam table positions wrap at either end of the table. The cam profile executes continuously until camming is disabled.

Examples:

IMC/IMJ	**Target ARS**	
CAZ	CAZ3	(* zero cam table)
CAM0=0	CAM3.0=0	(* set axis position at 0 degrees to 0 units)
CAM180=10	CAM3.180=10	(* set axis 1 position at 180 degrees to 10 units)
CAM0=0	CAM3.0=0	(* fill rest of table from 180 degrees to 0)
CAE=1	CAE3=1	(* enable cam following)

What Will Happen: The cam table is cleared and the three CAM data points construct an absolute move on the axis from zero to absolute position 10 and then back to zero. When the cam is enabled the controller reads the current master position (CAP which was initialized to zero) and moves the axis to its corresponding position from the cam table (in this case 0). The axis executes the 0-10-0 profile continuously until camming is disabled.

Related Commands: CAE, CAO, CAS, CAT, CAZ

CAO Cam Offset

Class:	Motion Register	
Type:	Floating point	
Syntax:		
I, jr	CAO	
⊙	CAOp1 (e.g., CAO1 CAOVI4)	
Parameters:	*allowed values*	*description*
⊙ *p1*	1 through 8 **or** VI*n*	axis number
Range:		
units	degrees	
default	0	
minimum	-180.0	
maximum	180.0	

Restrictions: For IMCs, this function available only with the extended command set.

Use: The cam offset register is used to define an offset on the cam master position. This has the effect of shifting all points on the cam table by the offset value and is often used to set phasing or timing of the cam relative to other motion on the machine.

Remarks: The value of the CAO register does not change the value stored in the PSX, CAR or CAP position registers. The value of CAO is summed with the value in the CAP register to offset the position of the cam master.

CAP Cam Shaft Position

Class:	Motion Register
Type:	Floating point
Syntax:	CAP
Range:	
units	degrees
minimum	0.000
maximum	359.999
Restrictions:	Read only; For IMCs, this function available only with the extended command set.
Use:	This register is used to determine the cam shaft (master) position within the defined 0–359,999 degree master cycle.

Remarks:

The defining input for this register is selected by the *cam shaft position type*, CAT, register. If CAT is set to CAR , then the CAP command will report the cam master position based on the value of the internal time-based *cam position register* (CAR). If CAT is set to PSX, then the CAP command will report the cam master position based on the value of the *auxiliary (encoder) position register* (PSX). This register cannot be set directly.

When CAT=PSX, the *auxiliary position length* (PLX) register is used to set the range of auxiliary encoder travel required to generate one complete cam cycle. For example, if the auxiliary encoder is a 1,000 line device (4,000 pulses) and the desired scaling is one auxiliary encoder revolution for one cam cycle (0–360 degrees span on CAP register), the PLX register must be set to 2,000 pulses (since PLX sets the aux. encoder position rollover to +/- PLX, one half the number of pulses for an encoder revolution are used). This configuration will cause CAP to count from 0–180 degrees as PSX counts from 0–1,999 pulses. PSX then rolls over to -2,000 pulses and counts back to zero as CAP completes the cycle from 181-359.999 degrees.

Related Registers: CAT, CAR, PSR, PSX

CAR Cam Position Register

Class:	Motion Register
Type:	Floating point
Syntax:	CAR
Range:	
units	degrees
default	0
minimum	0.000
maximum	359.999
Restrictions:	For IMCs, this function available only with the extended command set.
Use:	This register is used to define an internal time-base as a virtual master for cam following.
Remarks:	The cam shaft position, CAP, is set to the value of this register when the cam shaft position type, CAT, is set to CAR. In this case the index rate for this register is defined by the *cam position register increment* (CAI) command in degrees/second. The CAR register increments at the rate defined by CAI while the cam function is enabled (CAE=1) and stops incrementing when camming is disabled (CAE=0). Camming can be enabled/disabled by a program and is automatically disabled when a controller fault occurs or the cam table is cleared (CAZ command is executed).
Related Registers:	CAT

CAS Cam Scale Factor

Class:	Motion Register	
Type:	Floating point	
Syntax:		
I, jr	CAS	
⊙	CASp1 (e.g., CAS2 CASVI5)	
Parameters:	*allowed values*	*description*
⊙ *p1*	1 through 8 **or** VI*n*	axis number
Range:		
default	1	
minimum	.010000	
maximum	100.000000	
Restrictions:	For IMCs, this function available only with the extended command set.	
Use:	This register is used to define a scale factor to be applied to the magnitude of every axis position entered in the cam point table.	
Remarks:	CAS allows the user to create normalized cam tables that can then be rescaled for different parts.	
Related Registers:	CAM	

CAT Cam Shaft Position Type

Class:	Motion Register
Syntax:	CAT
Range:	
I, jr *allowed values*	PSX (auxiliary position) CAR (cam position)
⊙ *allowed values*	CAR (cam position) PSR*a* (resolver position of selected axis)
Restrictions:	For IMCs, this function available only with the extended command set; cannot be used in expressions.
Use:	This register selects the position register to use for cam following. For normal cam following, CAT should be set to PSX or PSRa. This makes the axis track the auxiliary encoder or axis resolver on the cam shaft. To make the axis move without the physical cam shaft turning, set CAT to CAR and set CAI to increment CAR at the desired rate.

Example:

```
CAT=CAR              (* set cam type to cam position register)
CAI=100              (* set increment to 100 degrees/sec)
```

Related Registers: PSX, CAR, CAP, CAI, PSR, PLX, PLA

CAZ Zeros Cam Table

Class:	Motion Command	
Syntax:		
I, jr	CAZ	
⊙	CAZp1 (e.g., CAZ5 CAZ234 CAZVI2)	
Parameters:	*allowed values*	*description*
⊙ *p1*	1 through 8 **or** list of numbers 1 through 8 **or** VI*n*	axis number
Restrictions:	For IMCs, this function available only with the extended command set.	
Use:	This command zeros the cam table. This must be done before a new set of cam points is entered.	
Registers Used:	CAM	
Motion Templates:		
I, jr, ⊙	Single-axis electronic camming	
⊙	Multi-axis synchronized electronic camming	

CCB Cam Compile Begin Point

Class:	Motion Register
Type:	Floating point
Syntax:	CCB
Range:	
units	degrees
default	0
minimum	0.0
maximum	359.9
Restrictions:	For IMCs, this function available only with the extended command set.
Use:	This register is used to define the beginning point for compiling the cam motion.
Related Registers:	CCE
***Related Commands*:**	CCM

CCE Cam Compile End Point

Class:	Motion Register
Type:	Floating point
Syntax:	CCE
Range:	
units	degrees
default	0
minimum	0.0
maximum	359.9
Restrictions:	For IMCs, this function available only with the extended command set.
Use:	This register is used to define the ending point for compiling the cam motion.
Related Registers:	CCB
***Related Commands*:**	CCM

CCM Compile Cam Motion

Class:	Motion Command	
Syntax:		
I, jr	CCM	
⊙	CCMp1 (e.g., CCM3 CCM125 CCMVI6)	
Parameters:	*allowed values*	*description*
⊙ *p1*	1 through 8 **or** list of numbers 1 through 8 **or** VI*n*	axis number

Restrictions: For IMCs, this function available only with the extended command set.

Use: This command compiles motion into the cam table. Axis motion starts at *cam compile start position*, CCP, and ends at the value specified for the axis absolute move, MPA. The axis position data is put in the cam table starting at the cam master position specified by the *cam compile begin point*, CCB, and ending at the *cam compile end point,* CCE. The axis motion is also defined by the usual parameters MAP, MDP, and MJK.

Remarks: The cam table can be populated with known master/slave position point pairs using simply the *cam point* (CAM) command; however, the *cam compile* (CCM) command allows the user to break the cam cycle into segments (specific range of cam master motion) and define an axis absolute motion profile for each segment. The compile command computes the cam points in the required 0.1 degree increments and populates the cam table accordingly. It is necessary to define segments that encompass the entire 360 degree cam cycle.

Examples:

IMC/IMJ	**Target ARS**	
CCB=60	CCB=60	(* set cam compile beginning point to 60 degrees)
CCE=250	CCE=250	(* set cam compile ending point to 250 degrees)
CCP=0	CCP1=0	(* set starting axis position to 0)
MPA=10	MPA1=10	(* set ending axis position to 10)
MAP=30	MAP1=30	(* set acceleration/deceleration percent to 30)
MJK=100	MJK1=100	(* set jerk percent to 100)
CCM	CCM1	(* compile axis motion into the cam table)

Registers Used: CCB, CCE, CCP, MPA, MAP, MDP, MJK

Motion Templates:

I, jr, ⊙	Single-axis electronic camming
⊙	Multi-axis electronic camming

CCP Cam Compile Start Position

Class:	Motion Register	
Type:	Floating point	
Syntax:		
I, jr	CCP	
⊙	CCPp1 (e.g., CCP2 CCPVI5)	
Parameters:	*allowed values*	*description*
⊙ *p1*	1 through 8 **or** VI*n*	axis number
Range:		
I, jr *units*	axis units	
default	0 pulses	
minimum	-2,000,000,000 pulses	
maximum	2,000,000,000 pulses	
⊙ *units*	axis units	
default	0 pulses	
minimum	-8,000,000 pulses	
maximum	8,000,000 pulses	

Restrictions: For IMCs, this function available only with the extended command set.

Use: This register is used to define the starting position of the axis for compiling the cam motion.

Remarks: The numerical values for the default, minimum, and maximum of this register are assuming that the *axis unit ratio*, URA, is set at its default value of 1. If the axis unit ratio is set to a value other than 1, the default, minimum, and maximum values will change appropriately (see URA).

Related Registers: MPA

Related Commands: CCM

CE Conversion Error

Class:	System Register	
Type:	Boolean	
Syntax:	CE*p1* (e.g., CE1 CEVI4)	
Parameters:	*allowed values*	*description*
I, jr *p1*	1 through 4 **or** VI*n*	program number
⊙ *p1*	1 through 17 **or** VI*n*	program number
Range:		
allowed values	0 **or** 1	
Restrictions:	Read only.	
Use:	The conversion error operand is used to determine whether a conversion operation in one of the programs worked correctly. A conversion error occurs when one data type (e.g. string) is converted to another type (e.g. floating point) and results in invalid data. If a conversion in program *p1* resulted in a conversion error, CE*p1* is set to 1; and if no error has occurred, CE*p1* is set to 0. Note that CE*p1* is updated after every conversion in program *p1*.	
Example:		
CEVI1?	(* report conversion error for program VI1)	
Related Registers:	SRP, ASC, CHR, ITF, STF, FTI, STI, FTS, ITB, ITH, ITS, ITD, ITT	

CHANGEPW Prompts for Password Change

Class:	System Command
Syntax:	CHANGEPW
Restrictions:	Not allowed in programs or motion blocks.
Use:	This command is used to prompt the user for an initial password or a password change.
Remarks:	If there is no existing password the CHANGEPW command will prompt for a new password. After the new password has been entered, the controller will prompt for the new password again for verification. If a password already exists the controller will prompt for the old password. After the old password has been entered, the controller will then prompt for the new password. The password can be from four to ten characters long. After the new password has been entered, the controller will prompt for the new password again for verification. Once this has been entered, the password will be changed to the new value. By entering no characters when prompted for the new password, the password function will be disabled. **WARNING: Once set there is no way to recover normal use of the controller without a valid password. Be sure to record the password and store in a safe location.**
Related Commands:	PASSWORD

CIE Computer Interface Format Enable

Class:	System Register
Type:	Boolean
Syntax:	CIE
Range:	
default	0
allowed values	0, 1
Restrictions:	Cannot be assigned in motion blocks.
Use:	The computer interface format enable register is used to define whether the computer interface format on the serial/program port is enabled. If CIE is set to 1, computer interface format is enabled, and if set to 0, computer interface format is disabled. See Appendix D for fault and status register details.
Remarks:	When the computer interface format is enabled, queries to fault and status registers return numerical values instead of message strings.
Related Registers:	HSE, FC, FI, IO, SRA, SRP, SRS

CLL Clears Line and Positions Cursor at Beginning of Line

Class:	Input/Output Command
Syntax:	CLL
Use:	This command clears the current line and positions the cursor at the beginning of the line on the display.
Remarks:	This command is used in conjunction with the display when DSE is set to 1.
Related Commands:	CLS
Related Registers:	DSE
ASCII Codes:	$1B$49

CLM Clears User Memory; Resets Registers to Defaults

Class:	System Command
Syntax:	CLM
Restrictions:	Not allowed in programs or motion blocks.
Use:	This command removes all programs and motion blocks and resets all registers to default values.
Remarks:	1. This command is irreversible; you cannot retrieve any programs, motion blocks, or registers that you have previously set after you execute this command. 2. This command will execute only when the controller or system and all axes are faulted, the UPS register is set to zero, and no programs or motion blocks are executing.

CLS Clears Display and Positions Cursor at Home

Class:	Input/Output Command
Syntax:	CLS
Use:	This command clears the display and positions the cursor at home (i.e., the first column of the first line of the display).
Remarks:	This command is used in conjunction with the display when DSE is set to 1.
Related Commands:	CLL
Related Registers:	DSE
ASCII Codes:	$1B$4A

CLX Clears Extended Memory Card

Class:	System Command
Syntax:	CLX
Restrictions:	Not allowed in programs or motion blocks.
Use:	This command sets all extended variables to default values.
Remarks:	1. This command is irreversible; you cannot retrieve any extended variables that you have previously set after you execute this command. 2. This command will execute only when the system and all axes are faulted and no programs or motion blocks are executing.

CMA Commutation Angle Advance *I* 🗎

Class:	Axis Register	
Type:	Floating point	
Syntax:		
I	CMA	
⊙	CMA*p1* (e.g., CMA2 CMAVI5)	
Parameters:	*allowed values*	*description*
⊙ *p1*	1 through 8 **or** VI*n*	axis number
Range:		
units	degrees per 75,000 pulses/sec	
default	0	
minimum	-90.0	
maximum	90.0	
Restrictions:	Brushless servo only.	
Use:	The commutation angle advance is used to compensate for the lag in the commutation angle at high speed introduced by the inductance of the motor.	

Examples:

IMC	**Target ARS**	
CMA=5	CMA1=5	(* set commutation angle advance)
CMA?	CMAVI2?	(* report commutation angle advance)

Related Registers: CMO

CMD Position Controller Command Output

Class:	Axis Register	
Type:	Floating point	
Syntax:		
I, jr	CMD	
⊙	CMDp1 (e.g., CMD2 CMDVI5)	
Parameters:	*allowed values*	*description*
⊙ *p1*	1 through 8 **or** VI*n*	axis number
Range:		
units	%	
minimum	-20,000.0	
maximum	20,000.0	
Restrictions:	Read only.	

Use: The position controller command output is used to control the position of the axis. It is a percentage of the controller *continuous current setting*, CURC.

Example:

IMC/IMJ	**Target ARS**	
CMD?	CMDVI2?	(* report position controller command output)

Related Registers: CURC

CMO Commutation Angle Offset

Class:	Axis Register	
Type:	Floating point	
Syntax:		
I, jr	CMO	
⊙	CMOp1 (e.g., CMO2 CMOVI5)	
Parameters:	*allowed values*	*description*
⊙ *p1*	1 through 8 **or** VI*n*	axis number
Range:		
units	degrees	
I ⊙ *default*	-90.0	
jr *default*	90.0	
minimum	-180.0	
maximum	180.0	
Restrictions:	Brushless servo only.	
Use:	The commutation angle offset of the motor is set by the motor manufacturer. If necessary, this value, along with the value of CMR, can be set automatically by the MOTORSET command.	
Related Registers:	CMA, CMR, AR	
Related Commands:	MOTORSET	

CMR Motor Poles to Resolver Poles Commutation Ratio

Class:	Axis Register	
Type:	Integer	
Syntax:		
I, jr	CMR	
⊙	CMRp1 (e.g., CMR1 CMFVI8)	
Parameters:	*allowed values*	*description*
⊙ *p1*	1 through 8 **or** VI*n*	axis number
Range:		
I, ⊙ *default*	2	
jr *default*	3	
minimum	1	
maximum	16	
Restrictions:	Brushless servo only	

Use: The motor poles to resolver poles commutation ratio is one of the motor constants needed to operate a resolver feedback servo motor. This value, along with the value of CMO, can be set automatically by the MOTORSET command.

Related Registers: CMO, AR

Related Commands: MOTORSET

COPYFLASH Copies Extended Memory Card

Class:	System Command
Syntax:	COPYFLASH
Restrictions:	Not allowed in programs or motion blocks.
Use:	This command copies the contents of the extended memory card into the flash memory card in the firmware slot.
Remark:	This command will execute only when the system and all axes are faulted and no programs or motion blocks are executing.

COPYRAM Copies Extended Memory Card

Class:	System Command
Syntax:	COPYRAM
Restrictions:	Not allowed in programs or motion blocks.
Use:	This command copies the contents of the extended memory card into the RAM memory card in the firmware slot.
Remark:	This command will execute only when the system and all axes are faulted and no programs or motion blocks are executing.

CR Positions Cursor at Beginning of Next Line Down

Class:	Input/Output Command
Syntax:	CR
Use:	This command positions the cursor at the beginning of the next line down on the display. It sends the ASCII codes for a carriage return ($0D) followed by a line feed ($0A) to the serial port. This command is typically used to positions the cursor at the beginning of the next line on an ASCII compliant operator display connected to the controller serial port.
Remarks:	This command is used in conjunction with the display when DSE is set to 1.
Related Commands:	BS, CRH, CRP
Related Registers:	DSE
ASCII Codes:	$0D$0A

CRH Positions Cursor at Home

Class:	Input/Output Command
Syntax:	CRH
Use:	This command positions the cursor at home (i.e., the first column of the first line of the display).
Remarks:	This command is used in conjunction with the display when DSE is set to 1.
Related Commands:	BS, CR, CRP
Related Registers:	DSE
ASCII Codes:	$1B$48

CRM Remembers Cursor Position

Class:	Input/Output Command
Syntax:	CRM
Use:	This command is used to remember the current position of the cursor.
Remarks:	CRM is used in conjunction with the display when DSE is set to 1.
Related Commands:	CRR
ASCII Codes:	$1B$3F

CRP Positions Cursor

Class:	Input/Output Command	
Syntax:	CRP*p1.p2* (e.g, CRP1.3 CRPVI2.3 CRP2.VI1 CRPVI1.VI2)	
Parameters:	*allowed values*	*description*
p1	1 to 4 **or** VI*n*	line position
p2	1 to 40 **or** VI*n*	column position

Use: This command positions the cursor on line *p1*, column *p2* of the display.

Remarks: This command is used in conjunction with the display when DSE is set to 1.

Example:

CRP1.2	(* position cursor at line 1, column 2 of the display)
CRP1.VI1	(* position cursor at line 1, column VI1 of the display)

Related Commands: BS, CR, CRH

Related Registers: DSE

ASCII Codes: $1B$46 $(p2+20h) $(p1+20h)

CRR Positions Cursor at Remembered Position

Class:	Input/Output Command
Syntax:	CRR
Use:	This command is used to place the cursor at the position remembered by the CRM command.
Remarks:	This command is used in conjunction with the display when DSE is set to 1.
Related Commands:	CRM
Related Registers:	DSE
ASCII Codes:	$1B$40

CTR Counter

Class:	Input/Output Register	
Type:	Integer	
Syntax:	CTR*p1.p2* (e.g., CTR1.3 CTR5.VI1 CTRVI2.1 CTRVI1.VI2)	
Parameters:	*allowed values*	*description*
p1	1 through 8 **or** VI*n*	digital module number
p2	1 through 4 **or** VI*n*	counter number
Range:		
default	0	
minimum	0	
maximum	2,000,000,000	

Use: Counters are used to count inputs to the system. These inputs are taken from the first four inputs of a digital I/O module. For example, counter one takes its count from digital input one, counter two from digital input two, etc. Counters can be reset (i.e., set to zero), but they cannot be set to any other value.

Example:

```
CTR1.3=0          (* set counter three of digital module one to zero)
CTR5.VI1?         (* report counter VI1 of digital module five)
```

Related Registers: TMI, TMP

CURC Continuous Current

Class: Axis Register

Type: Floating point

Syntax:

I, jr	CURC
⊙	CURC*p1* (e.g., CURC1 CURCVI3)

Parameters:

	allowed values	*description*
⊙ *p1*	1 through 8 **or** VI*n*	axis number

Range:

units	%
default	60.0 (stepper) and 100.0 (brushless servo)
minimum	1.0
maximum	100.0

Restrictions: Stepper and brushless servo only.

Use: The continuous current setting limits the current that the drive will continuously supply to the motor. It is a percentage of the maximum continuous current rating of the drive.

Remarks: The equation for CURC = motor continuous current rating / drive continuous current rating x 100%. Continuous current ratings are listed on the drive and motor product labels. Use the following values for CURC:

IMC/IMJ Stepper Unit

Motor	***5 Amps***
1221-_-A-E-_	70
1231-_-A-E-_	62
1324-_-A-E-_	100
1324-_-D-E-_	54
1337-_-A-E-_	100
1337-_-D-E-_	82
1350-_-A-E-_	100
1350-_-D-E-_	80
1362-_-A-E-_	100
1454-_-A-E-_	100
1480-_-A-E-S	100

IMC Servo Unit CURC Values

Motor	***3 Amps***	***6 Amps***	***12 Amps***	***24 Amps***
3S22-G	46	23	n/a	n/a
3S32-G	96	48	24	n/a
3S33-G	100	53	26	n/a
3S33-H	100	100	53	26
3S34-G	100	50	25	n/a
3S35-G	96	48	24	n/a
3S43-G	96	48	24	n/a
3S43-H	100	93	46	23
3S45-G	100	91	45	22
3S45-H	100	100	91	45
3S46-G	100	91	45	22
3S46-H	100	100	91	45
3S63-G	100	100	91	45
3S65-G	100	100	89	44
3S67-G	100	100	94	47
3S88-G	100	100	100	100
3S8A-G	100	100	100	100

Related Registers: CURP, CURS, TLC

IMJ Servo Unit CURC Values

Motor	***3 Amps***	***7.2 Amps***	***16 Amps***	***28 Amps***
3N21-H	100	42	n/a	n/a
3N22-H	100	42	n/a	n/a
3N24-G	87	36	n/a	n/a
3N31-H	100	46	21	n/a
3N32-G	100	42	n/a	n/a
3N32-H	100	86	39	22
3N33-G	93	39	n/a	n/a
3N33-H	100	79	36	20
3S22-G	50	21	n/a	n/a
3S32-G	100	42	n/a	n/a
3S33-G	100	44	20	n/a
3S33-H	100	88	40	23
3S34-G	100	42	n/a	n/a
3S35-G	100	42	n/a	n/a
3S43-G	97	40	n/a	n/a
3S43-H	100	79	36	20
3S45-G	100	76	34	20
3S45-H	100	100	69	39
3S46-G	100	76	34	20
3S46-H	100	100	69	40
3S63-G	100	100	69	40
3S63-H	100	100	100	79
3S65-G	100	100	68	39
3S65-H	100	100	100	77
3S67-G	100	100	71	40
3S67-H	100	100	100	81
3S84-G	100	100	100	100
3S86-G	100	100	100	100
3S88-G	100	100	100	100
3S8A-G	100	100	100	88

Target ARS Servo Motor CURC Values

Motor	*1 Servo Modules*	*2 Servo Modules*	*3 Servo Modules*	*4 Servo Modules*
3S22-G	23	n/a	n/a	n/a
3S32-G	48	24	n/a	n/a
3S33-G	53	26	n/a	n/a
3S33-H	100	53	35	26
3S34-G	50	25	n/a	n/a
3S35-G	48	24	n/a	n/a
3S43-G	48	24	n/a	n/a
3S43-H	93	46	31	23
3S45-G	91	45	30	22
3S45-H	100	91	61	45
3S46-G	91	45	30	22
3S46-H	100	91	61	45
3S63-G	100	91	61	45
3S65-G	100	89	59	44
3S67-G	100	94	63	47
3S88-G	100	100	100	100
3S8A-G	100	100	100	100

IMJ Servo Unit CURC Values

Motor	***3 Amps***	***7.2 Amps***	***16 Amps***	***20 Amps***	***28 Amps***
3T11-G	32	n/a	n/a	n/a	n/a
3T12-G	63	n/a	n/a	n/a	n/a
3T13-G	91	n/a	n/a	n/a	n/a
3T21-G	57	n/a	n/a	n/a	n/a
3T22-G	88	n/a	n/a	n/a	n/a
3T23-G	90	38	n/a	n/a	n/a
3T23-H	100	47	n/a	n/a	n/a
3T23-I	100	69	n/a	n/a	n/a
3T24-H	100	46	n/a	n/a	n/a
3T24-I	100	74	n/a	n/a	n/a
3T42-G	93	39	n/a	n/a	n/a
3T42-H	100	65	29	24	n/a
3T43-G	100	51	23	n/a	n/a
3T43-H	100	64	29	23	n/a
3T43-I	100	100	63	51	n/a
3T43-J	100	100	45	36	n/a
3T44-G	100	50	23	n/a	n/a
3T44-H	100	75	34	27	n/a
3T44-I	100	100	63	51	n/a
3T44-J	100	100	45	36	n/a
3T45-G	100	50	23	n/a	n/a
3T45-H	100	99	44	36	n/a
3T45-I	100	100	63	50	n/a
3T53-G	n/a	94	43	34	24
3T53-H	n/a	100	61	49	35
3T54-G	n/a	99	44	36	25
3T54-H	n/a	100	66	53	38
3T55-G	n/a	99	44	36	25
3T55-H	n/a	100	66	53	38
3T55-I	n/a	100	100	100	76
3T57-G	n/a	100	61	49	35
3T57-H	n/a	100	100	98	70
3T65-G	n/a	n/a	71	57	40
3T65-H	n/a	n/a	100	100	75
3T66-G	n/a	n/a	71	57	40
3T66-H	n/a	n/a	100	100	74
3T67-G	n/a	n/a	100	100	74
3T69-G	n/a	n/a	100	100	74

CURP Peak Current

Class:	Axis Register	
Type:	Floating point	
Syntax:		
I, jr	CURP	
⊙	CURP*p1* (e.g., CURP1 CURPVI3)	
Parameters:	*allowed values*	*description*
⊙ *p1*	1 through 8 **or** VI*n*	axis number
Range:		
units	%	
default	100.0	
minimum	1.0	
maximum	100.0	

Restrictions: Brushless servo only.

Use: The peak current setting limits the peak value of the current that the drive will supply to the motor. It is a percentage of the maximum peak current rating of the drive. The maximum peak current is two times the drive's continuous rating.

Remarks: Use the following equation to calculate CURP:

100% x (motor peak current rating / drive peak current rating)

For example, when using a 5 Amp motor with a 4.3 Amp drive (8.6 Amp peak), CURP = *100% x (5 Amps / 8.6 Amps) = 58*%.

Related Registers: CURC

CURS Power Save Current *I jr*

Class:	Axis Register
Type:	Floating point
Syntax:	CURS
Range:	
units	%
default	60.0
minimum	0.0
maximum	100.0
Restrictions:	Stepper only.
Use:	The power save current is used to reduce motor heating when the axis is stopped. While the axis is in position, the continuous current value, CURC, is reduced to the percentage loaded into CURS. For example, if CURC=50 and CURS=20, the value of CURC will be reduced to 10 percent while the axis is in position.
Related Registers:	CURC

DATE Date *I* 🗎

Class:	System Register
Type:	String
Syntax:	DATE
Range:	
allowed values	1994-1-1 (i.e., January 1, 1994) through 2060-12-31 (i.e., December 31, 2060)
Restrictions:	Cannot be assigned in motion blocks.
Use:	The DATE register is used to keep track of the current date. The format is *year-month-day*. For example, to set the date to August 9, 1998, the command would be DATE="1998-8-9".

Example:

```
DATE="1998-7-21"        (* set date to 1998-7-21 [i.e., July 21, 1998])
DATE?                   (* report date)
*"1998-07-21"
```

Related Registers: TIME, DAY, MONTH

DAY Day *I*

Class:	System Register
Type:	String
Syntax:	DAY
Range:	
allowed values	Sunday, Monday, Tuesday,...Saturday
Restrictions:	Read only.
Use:	The DAY register is used to keep track of the current day of the week.
Example:	
DAY? *Tuesday	(* report day)
Related Registers:	TIME, DATE, MONTH

DEL Deletes Current Statement in Line Editor

Class: Program Command

Syntax: DEL

Restrictions: Allowed only in programs or motion blocks.

Use: This command is used to edit programs or motion blocks in the terminal window line editor. It deletes the current statement in the line editor and makes the next statement the current statement.

Remarks: To use the terminal window for program editing, use the PROGRAM and MOTION commands. Edits you make in the line editor are not saved to your original program or motion block's .txt file.

While in the line editor each line is prefixed by an asterisk (*). The exclamation point (!) command is used to exit the terminal window line editor.

Examples:

```
IMC/IMJ      Target ARS

PROGRAM1     PROGRAM1     (* define program 1)
*  PSA=0     *  PSA1=0
X            X            (* step through program)
*  MVL=10    *  MVL1=10
X            X            (* step through program)
*  MAC=40    *  MAC1=40
DEL          DEL          (* delete current statement)
*  MPA=12    *  MPA1=12
MAC=10       MAC1=10      (* set motion acceleration)
*  MPA=12    *  MPA1=12
!            !            (* exit line editor)
```

What will happen: This program example changes "MAC=40" to "MAC=10" in program 1.

Related Commands: PROGRAM, L, LABEL, X, !

DGC Loads Diagnostic Condition for Printing

Class: Diagnostic Command

Syntax: DGC*p1*=*p2* (e.g., DGC1=MB1, DGC2=TL1 or IP1)

Parameters:

	default	*allowed values*	*description*
I, jr *p1*		1 through 4	diagnostic condition number
p2	OFF	any Boolean expression **or** OFF	diagnostic condition
⊙ *p1*		1 through 8	diagnostic condition number
p2	OFF	any Boolean expression **or** OFF	diagnostic condition

Restrictions: Not allowed in programs or motion blocks.

Use: This command assigns diagnostic condition *p1*. When one of the user defined diagnostic conditions is satisfied, and if diagnostics are enabled, a diagnostic line of items is sent to the terminal (see DGL, DGI).

Remarks: Upon clearing the memory with the CLM command, all diagnostic conditions and items are set to the value “OFF,” which means that there are no diagnostic conditions/items assigned. If you wish to eliminate the assignment of diagnostic condition *p1*, use the DGC command and set parameter *p2* to “OFF.” For example, DGC1=OFF will eliminate the assignment of diagnostic condition one.

Example:

```
STM2=0.5                (* set start time of timer two to 0.5 seconds)
DGC1=TM2 AND
PROG1                   (* assign diagnostic condition 1)
```

What will happen: Setting the start time of timer 2 and assigning diagnostic condition 1 will send a diagnostic line of items to the terminal every 0.5 seconds while program 1 is executing. Each diagnostic line will begin with the diagnostic condition satisfied, which in this case would be “TM2 AND PROG1,” and then be followed by a colon and the diagnostic items loaded.

Related Commands: DGE, DGI, DGL, DGP

DGE Enables Diagnostics

Class:	Diagnostic Command		
Syntax:	DGE=*p1* (e.g., DGE=0)		
Parameters:	*default*	*allowed values*	*description*
p1	0	0 and 1	diagnostic enable bit
Restrictions:	Not allowed in programs or motion blocks. Diagnostics work only via serial communication.		
Use:	This command is used to enable the diagnostic mode of the system. When DGE is set to 1, diagnostics are enabled, and when set to 0, diagnostics are disabled.		
Remarks:	DGE is set to 0 upon power-up.		
Related Commands:	DGC, DGI, DGL, DGP, DGO, DGS, DGT		

DGI Assigns Diagnostic Item to Print

Class:	Diagnostic Command		
Syntax:	DGI*p1*=*p2* (e.g., DGI1=VLA DGI3=PHR1)		
Parameters:	*default*	*allowed values*	*description*
p1		1 through 8	diagnostic item number
p2	OFF	any register **or** OFF	diagnostic item

Restrictions: Not allowed in programs or motion blocks.

Use: This command assigns a diagnostic item to be printed whenever a DGL is executed or whenever one of the user-defined diagnostic conditions is met.

Remarks: Upon clearing the memory with the CLM command, all diagnostic conditions and items are set to the value “OFF,” which means that there are no diagnostic conditions/items assigned. If you wish to eliminate the assignment of diagnostic item *p1*, use the DGI command and set parameter *p2* to “OFF.” For example, DGI1=OFF will eliminate the assignment of diagnostic item one.

Examples:

IMC/IMJ	**Target ARS**	
DGI1=PSA	DGI1=PSA1	(* assign diagnostic item one)
DGI2=VLA	DGI2=VLA1	(* assign diagnostic item two)
DGI3=FE	DGI3=FE1	(* assign diagnostic item three)
DGI4=PSR	DGI4=RSR1	(* assign diagnostic item four)

What will happen: Assigning these diagnostic items when diagnostics are enabled will send the diagnostic items to the terminal when the DGL command is executed.

Related Commands: DGE, DGC, DGL, DGP

DGL Prints Diagnostic Line of Items

Class: Diagnostic Command

Syntax: DGL

Use: This command prints to the terminal a diagnostic line of items that have been assigned with the DGI command. This works only while diagnostics are enabled.

Remarks: Since this command is ignored when diagnostics are not enabled, it can be left in programs even when you are not using diagnostics.

Examples:

```
IMC/IMJ        Target ARS
DGI1=PSA       DGI=1PSA1        (* assign diagnostic item one)
DGI2=VLA       DGI2=VLA1        (* assign diagnostic item two)
DGI3=FE        DGI3=FE1         (* assign diagnostic item three)
DGI4=PSR       DGI4=PSR1        (* assign diagnostic item four)
DGL            DGL              (* print diagnostic line of items)
*DGL: PSA=0,   *DGL: PSA1=0,
    VLA=0          VLA1=0
DGL: FE=0,     DGL: FE1=0,
    PSR=3061       PSR1=3061
```

Related Commands: DGE, DGC, DGI, DGP

DGO Outputs Diagnostic Register Value to Serial Port

Class: Diagnostic Command

Syntax:

I, jr	DGO *p1* (e.g., DGO VLA, DGO IO)
⊙	DGO *p1* (e.g., DGO VLA1, DGO IOS)

Parameters:

	allowed values	*description*
p1	any register	register

Use: This command outputs a diagnostic register value to the serial or program port when diagnostics are enabled (DGE=1). It works the same as the "?" command, but it can also be used in programs and motion blocks.

Remarks: Since this command is ignored when diagnostics are not enabled, it can be left in programs even when you are not using diagnostics.

Examples:

DGO PSA	(* outputs axis position to the serial port)
DGO VLA	(* outputs axis velocity to the serial port)

Related Commands: ?, DGE, DGL, DGP

DGP Prints Diagnostic Message to Serial Port

Class:	Diagnostic Command	
Syntax:	DGP“*p1*” (e.g., DGP“Drill operating”)	
Parameters:	*allowed values*	*description*
p1	any string, 0 through 127 characters long	diagnostic message

Use: This command prints the diagnostic message *p1* to the serial or program port. It works only when diagnostics are enabled.

Remarks: Since this command is ignored when diagnostics are not enabled, it can be left in programs even when you are not using diagnostics.

Example:

```
DGE=1                          (* enable diagnostics)
DGP“Diagnostics enabled”       (* send diagnostic message to serial or program port)
*Diagnostics enabled
```

Related Commands: DGE, DGC, DGI, DGL

DGS Sets Program to Single Step Mode

Class: Diagnostic Command

Syntax: DGS=*p1* (e.g., DGS=2)

Parameters:	*default*	*allowed values*	*description*
I, jr *p1*	0	0 through 4	program number (0 = no program in single step mode)
⊙ *p1*	0	0 through 17	program number (0 = no program in single step mode)

Restrictions: Not allowed in motion blocks.

Use: This command sets program *p1* to single step mode. If DGS is set to 0, single step mode is disabled. Single step mode can occur only when diagnostics are enabled.

Remarks: To execute a program while in single step mode, use the X command to step through the program (i.e., execute the program one statement at a time). As each line of the program is executed, it is sent to the terminal.

Examples:

```
IMC/IMJ    Target ARS
DGE=1      DGE=1      (* enable diagnostics)
DGS=3      DGS=3      (* set program three to single step mode)
EXP3       EXP2       (* execute program three)
* PSA=0    * PSA1=0
X          X          (* step through program)
* MVL=25   * MVL1=25
X          X          (* step through program)
* MAC=10   * MAC1=10
X          X
* MPI=40   * MPI1=40
X          X
* RPI      * RPI1
X          X
* END      * END
X          X
*          *
```

What will happen: Enabling diagnostics, setting program three to single step mode, and executing program three will cause only the first line of the program to execute. The X command causes the program to execute the next line, send that line to the terminal, and so on until it reaches the end of the program.

Related Commands: DGE, DGT

DGT Sets Program to Trace Mode

Class:	Diagnostic Command		
Syntax:	DGT=*p1* (e.g., DGT=2)		
Parameters:	*default*	*allowed values*	*description*
I, jr *p1*	0	0 through 4	program number (0 = no program in trace mode)
⊙ *p1*	0	0 through 17	program number (0 = no program in trace mode)

Restrictions: Not allowed in motion blocks.

Use: This command sets program *p1* to trace mode. If DGT is set to 0, trace mode is disabled. Trace mode can occur only when diagnostics are enabled.

Remarks: 1. When trace mode is enabled, each line of program *p1* is sent to the terminal as it is executing.
2. **CAUTION:** Trace mode can cause the program to run approximately 1,000 times slower than normal!

Examples:

```
IMC/IMJ     Target ARS
DGE=1       DGE=1       (* enable diagnostics)
DGT=3       DGT=3       (* set program three to trace mode)
EXP3        EXP3        (* execute program three)
* PSA=0     * PSA1=0
  MVL=25      MVL1=25
  MAC=10      MAC1=10
  MPI=40      MPI1=40
  RPI         RPI1
  END         END
*           *
```

What will happen: Enabling diagnostics, setting program three to trace mode, and executing program three will cause each line of the program to be sent to the terminal while it is executing.

Related Commands: DGE, DGS

DI Digital Input

Class: Input/Output Register

Type: Integer, Boolean

Syntax:

I, jr	DI*p1* (e.g., DI DI4 DIVI1)
⊙	DI*p1.p2* (e.g., DI1 DIVI2 DI1.4 DI1.VI1 DIVI1.3)

Parameters:

	allowed values	*description*
IMJ-_ _ _D *p1*	none **or** 1 through 14 **or** VI*n*	digital input number
IMJ-_ _ _E *p1*	none **or** 1 through 21 **or** VI*n*	digital input number
I *p1*	none **or** 1 through 12 **or** VI*n*	digital input number
⊙ *p1*	1 through 8 **or** VI*n*	digital module number
⊙ *p2*	none **or** 1 through 32 **or** VI*n*	digital input number

Range:

	allowed values
IMJ-_ _ _-D	0 through $3FFF_{16}$ **or** 0 and 1
IMJ-_ _ _-E	0 through $1FFFFF_{16}$ **or** 0 and 1
I	0 through FFF_{16} **or** 0 and 1
⊙	0 through $FFFFFFFF_{16}$ **or** 0 and 1

Restrictions: Read only.

Use: The digital input register contains the values of digital inputs, which are general purpose inputs used for process control.

Remarks:

I*, *jr	1. When the DI*p1*? command is executed, the value of the digital input *p1* will be given as a Boolean number.
I*, *jr	2. When DI? is executed, the digital inputs will be reported as binary numbers. The left-most bit represents digital input 12, 14, or 21, depending on your controller model number (see ***Parameters*** above); and the right-most bit represents digital input 1.
jr	3. DI1 = home; DI2 = forward overtravel (+OT); DI3 = reverse overtravel (-OT). Set the OTE register = 1 to enable the overtravels.
⊙	1. When the DI*p1.p2*? command is executed, the value of the digital input *p2* of digital module *p1* will be given as a Boolean number.
⊙	2. When DI*p1*? is executed, digital inputs 1 through 32 of digital module *p1* will be reported as a binary number with the bits in groups of four. The left-most bit represents digital input 32, and the right-most bit represents digital input 1.

Examples:

IMC/IMJ	**Target**	
DI?	DIVI1?	(* report value of digital input register)
DI4?	DI1.4?	(* report value of digital input four)

Related Registers: EG, DO, DID, IO, IOA, IOS

DIA Digital Input Filter Assignment

Class:	Input/Output Register	
Type:	Integer	
Syntax:	DIA*p1.p2* (e.g., DIA1.2 DIA1.VI1 DIAVI1.1 DIAVI1.VI2)	
Parameters:	*allowed values*	*description*
p1	1 through 8 **or** VI*n*	digital module number
p2	1 and 2 **or** VI*n*	1 — digital inputs 1 through 16 2 — digital inputs 17 through 32

Range:

allowed values	0 through $FFFF_{16}$

Restrictions: Cannot be assigned in motion blocks.

Use: The digital input filter assignment is used to define which of the digital inputs are to be filtered. This number is a 16-bit binary number, where the left-most bit represents input 16 for *p2*=1 (32 for *p2*=2) and the right-most bit represents input 1 for *p2*=1 (17 for *p2*=2). If an input's corresponding bit is set to 1, then the input is to be filtered; and if set to 0, it is not to be filtered.

Example:

DIA1.1=2#0001_0001_1010_0000	(* set digital input filter assignment of digital module one to inputs 6, 8, 9, and 13)
DIA1.2=2#0001_0001_1010_0000	(* set digital input filter assignment of digital module one to inputs 22, 24, 25, and 29)

Related Registers: DI, DIT

DID Digital Input Digit

Class:	Input/Output Register	
Type:	Integer	
Syntax:	DID*p1.p2* (e.g., DID1.4 DID1.VI1 DIDVI1.3 DIDVI1.VI2)	
Parameters:	*allowed values*	*description*
p1	1 through 8 **or** VI*n*	digital module number
p2	1 through 8 **or** VI*n*	digital input digit number
Range:		
minimum	0	
maximum	15	
Restrictions:	Read only.	
Use:	The digital input digits are hexadecimal digits, each of which are taken from four digital inputs. For example, digital input digit 1 comes from digital inputs 1 through 4, digit 2 comes from inputs 5 through 8, and so on up to digit 8, which comes from digital inputs 29 through 32.	
Example:		
DIDVI1.3?	(* report value of digital input digit three of digital module VI1)	
Related Registers:	DI	

DIR Direction of Motor for Forward Moves

Class:	Axis Register	
Type:	String	
Syntax:		
I*, *jr	DIR	
⊙	DIR*p1* (e.g., DIR2 DIRVI1)	
Parameters:	*allowed values*	*description*
⊙ *p1*	1 through 8 **or** VI*n*	axis number
Range:		
default	CW	
allowed values	CW, CCW	
Restrictions:	Not allowed in motion blocks.	

Use: This register is used to define the direction of the motor assigned to the axis for forward moves. If DIR is set to CW, a forward move by the motor is clockwise, facing the motor shaft. If DIR is set to CCW, a forward move by the motor is counterclockwise, facing the motor shaft. The Fwd/Rev LED on the front of the controller illuminates green when the axis is moving in the forward direction and yellow when moving in the negative direction. In a program, this register can be set only when the controller is faulted.

DIT Digital Input Filter Time

Class:	Input/Output Register	
Type:	Floating point	
Syntax:		
I, ***jr***	DIT*p1* (e.g., DIT2 DITVI3)	
⊙	DIT*p1.p2* (e.g., DIT1.2 DIT1.VI1 DITVI1.1 DITVI1.VI2)	
Parameters:	*allowed values*	*desc*ription
I, ***jr*** *p1*	1 through 12 **or** VI*n*	digital input number
⊙ *p1*	1 through 8 **or** VI*n*	digital module number
p2	1 and 2 **or** VI*n*	1— digital inputs 1 through 16 2 — digital inputs 17 through 32
Range:		
I, ***jr*** *units*	seconds	
default	0	
minimum	0	
maximum	4.000	
⊙ *units*	seconds	
default	0.001	
minimum	0.001	
maximum	4.000	

Restrictions: Cannot be assigned in motion blocks.

Use: The digital input filter time is used to represent the minimum duration of a pulse that the filter will allow to pass. This filter time is applied to the digital input specified. DIT allows up to three decimal places.

Remarks: The primary use for this command is to debounce a contact connected to a digital input. Generally, contact bounce lasts for less than 30 milliseconds; so setting DIT=.03 should debounce the contact. Because filtering slows input response, use the smallest value for filter time that works for the application.

Examples:

IMC/IMJ	**Target ARS**
DIT3=.03	DIT1.1=.03 (* set digital input filter time to 30 ms)

Related Registers: DI, DIA

DM Digital Module Rack Slot Assignment

Class:	System Register	
Syntax:	DM*p1* (e.g., DM4 DMVI2)	
Parameters:	*allowed values*	*description*
p1	1 through 8	digital module number
Range:		
default	0	
allowed values	0; 11 through 18; 21 through 28; 31 through 38	
Restrictions:	Not allowed in programs, motion blocks, or expressions.	
Use:	The digital module rack slot assignment is used to define in which slot a digital module resides. The digital module rack slot assignment consists of two digits. The first digit is the rack number, and the second digit is the slot number. If DM*p1* is equal to 0, it means that digital module *p1* is not used in the system.	
Example:		
DM1=18	(* set digital module rack slot assignment of digital module one to rack one, slot eight)	
DM5?	(* report digital module rack slot assignment of digital module 5)	
Related Registers:	AM, SM, DME	

DME Digital Module Assignment Error

Class:	System Register	
Type:	Integer, Boolean	
Syntax:	DME*p1* (e.g., DME DME8 DMEVI2)	
Parameters:	*allowed values*	*description*
p1	none **or** 0 through 23 **or** VI*n*	digital module assignment error register bit number
Range:		
allowed values	0 through $FFFFFF_{16}$ **or** 0 and 1	
Restrictions:	Read only.	
Use:	The digital module assignment error register is used to determine if any of the digital modules are not properly assigned by the system.	

Remarks:

1. When the DME? command is executed, the module assignment error register value will be given as an English statement. If all digital module assignments are correct, the message given is *All module assignments are correct*.
2. If the computer interface format is enabled, and the DME? command is executed, the module assignment error register value will be given as an integer number. If all digital module assignments are correct, the module assignment error register is set to 0. The possibilities are listed below:

bit	*message*
0	Module in rack one, slot one did not respond to assignment
1	Module in rack one, slot two did not respond to assignment
2	Module in rack one, slot three did not respond to assignment
3	Module in rack one, slot four did not respond to assignment
4	Module in rack one, slot five did not respond to assignment
5	Module in rack one, slot six did not respond to assignment
6	Module in rack one, slot seven did not respond to assignment
7	Module in rack one, slot eight did not respond to assignment
8	Module in rack two, slot one did not respond to assignment
...	...
...	...
...	...
22	Module in rack three, slot seven did not respond to assignment
23	Module in rack three, slot eight did not respond to assignment

DO Digital Output

Class:	Input/Output Register	
Type:	Integer, Boolean	
Syntax:		
I, ***jr***	DO*p1* (e.g., DO DO9 DOVI1)	
⊙	DO*p1.p2* (e.g., DO1 DOVI3 DO1.4 DO1.VI1 DOVI1.VI2)	
Parameters:	*allowed values*	*description*
IMJ-_ _ _D *p1*	none **or** 9 through 14 **or** VI*n*	digital output number
IMJ-_ _ _E *p1*	none **or** 12 through 21 **or** VI*n*	digital output number
I *p1*	none **or** 7 through 12 **or** VI*n*	digital output number
⊙ *p1*	1 through 8 **or** VI*n*	digital module number
p2	none **or** 1 through 32 **or** VI*n*	digital output number
Range:	*allowed values*	
IMJ-_ _ _-D	0 through $3F00_{16}$ **or** 0 and 1	
IMJ-_ _ _-E	0 through $1FF800_{16}$ **or** 0 and 1	
I	0 through $FC0_{16}$ **or** 0 and 1	
⊙	0 through $FFFFFFFF_{16}$ **or** 0 and 1	

Use:

The digital output register contains the values of digital outputs. The digital outputs are general purpose outputs used for process control.

Remarks:

I, jr

1. When the DO*p1*? command is executed, the value of the digital output *p1* will be given as a Boolean number.
2. When DO? is executed, the digital outputs will be reported as binary numbers. The left-most bit represents digital output 12, 14, or 21, depending on your controller model number (see ***Parameters*** above); and the right-most bit represents digital output 1.

I

1. The IMC has six set point outputs, A–F, that are assigned to the following digital outputs: A=11, B=12, C=7, D=8, E=9, F=10.

⊙

1. When the DO*p1.p2*? command is executed, the value of the digital output *p2* of digital module *p1* will be given as a Boolean number.
2. When DO*p1*? is executed, digital outputs 1 through 32 of digital module *p1* will be reported as a binary number with the bits in groups of four. The left-most bit represents digital output 32, and the right-most bit represents digital output 1.

Examples:

IMC/IMJ	**Target ARS**	
DO=16#3400	DO1=16#11A00000	(* set digital output register)
DOVI1=1	DO1.VI1=1	(* set digital output VI1)
DO?	DO1?	(* report digital output register)
DO12?	DO1.12?	(* report value of digital output 12)

Related Registers: DI, DO, DOD, DOP

DOD Digital Output Digit

Class:	Input/Output Register	
Type:	Integer	
Syntax:	DOD*p1.p2* (e.g., DOD1.4 DOD1.VI1 DODVI1.3 DODVI1.VI2)	
Parameters:	*allowed values*	*description*
p1	1 through 8 **or** VI*n*	digital module number
p2	1 through 8 **or** VI*n*	digital output digit number
Range:		
default	0	
minimum	0	
maximum	15	

Use: The digital output digits are hexadecimal digits, each of which are taken from four digital outputs. For example, digital output digit 1 comes from digital outputs 1 through 4; digit 2 comes from outputs 5 through 8; and so on up to digit 8, which comes from digital outputs 29 through 32.

Example:

```
DOD1.2=12        (* set digital output digit two of digital module one to 12)
DODVI1.3?        (* report value of digital output digit three of digital
                 module VI1)
```

Related Registers: DO

DOE Fault on Digital Output Fault Enable

Class:	Input/Output Register
Type:	Boolean
Syntax:	DOE
Range:	
default	0
allowed values	0, 1
Restrictions:	Cannot be assigned in motion blocks.
Use:	This register is used to enable the system to fault on a digital output fault. A digital output fault occurs when the state of the digital output is true but the state of the associated digital input is not (after a time of 4 ms). If DOE is set to 1, the fault on digital output fault is enabled; and if DOE is set to 0, it is disabled.

DOP Power-up State of Digital Outputs *I* 🗎

Class: Input/Output Register

Syntax:

I	DOP
⊙	DOP*p1.p2* (e.g., DOP1.2 DOP4.VI1 DOPVI1.1 DOPVI1.VI2)

Parameters:	*allowed values*	*description*
⊙ *p1*	1 through 8 **or** VI*n*	digital module number
⊙ *p2*	1 and 2 **or** VI*n*	1 — digital outputs 1 through 16 2 — digital outputs 17 through 32

Range:

default	OFF
allowed values	OFF (all off) LAST (last state)

Restrictions: Not allowed in programs, motion blocks, or expressions.

Use: The power-up state of digital outputs is the state that the digital output assumes upon system power-up. “LAST” means that the power-up state of the digital outputs is the same as the state they were in before the system was powered off.

Examples:

IMC	**Target ARS**	
DOP=OFF	DOP1.1=OFF	(* set power-up state of digital outputs to OFF)
DOP?	DOPVI1.2?	(* report value of power-up state of digital outputs)

Related Registers: DO

DSE Display Format Enable *I jr*

Class:	System Register
Type:	Boolean
Syntax:	DSE
Range:	
default	1
allowed values	0, 1
Restrictions:	Cannot be assigned in motion blocks.
Use:	This command is used to enable the display format on the serial port. If DSE is set to 1, the display format is enabled; and if set to 0, the display format is disabled.
Remarks:	When the display format is enabled, output strings from the PUT and OUT commands are prefixed by control code 11_{16} and suffixed by control code 12_{16}. The OIP display intercepts all strings delimited by the control codes and does not send those strings to its host port.
Related Commands:	PUT, OUT

This page left blank intentionally.

EG Positive-edge-sensitive Digital Input

Class:	Input/Output Register
Type:	Integer, Boolean

Syntax:

I, ***jr***	EG*p1* (e.g., EG EG4 EGVI3)
⊙	EG*p1.p2* (e.g., EG1 EGVI2 EG1.4 EG1.VI1 EGVI1.3)

Parameters:	*allowed values*	*description*
IMJ-_ _ _D *p1*	none **or** 1 through 14 **or** VI*n*	positive-edge-sensitive digital input number
IMJ-_ _ _E *p1*	none **or** 1 through 21 **or** VI*n*	positive-edge-sensitive digital input number
I *p1*	none **or** 1 through 12 **or** VI*n*	positive-edge-sensitive digital input number
⊙ *p1*	1 through 8 **or** VI*n*	digital module number
⊙ *p2*	none **or** 1 through 32 **or** VI*n*	positive-edge-sensitive digital input number

Range:

IMJ-_ _ _-D	0 through $3FFF_{16}$ **or** 0 and 1
IMJ-_ _ _-E	0 through $1FFFFF_{16}$ **or** 0 and 1
I *allowed values*	0 through FFF_{16} **or** 0 and 1
⊙ *allowed values*	0 through $FFFFFFFF_{16}$ **or** 0 and 1

Use: EG contains the values of all digital inputs that have made a low to high transition since they were last cleared. These general purpose inputs are used for process control.

Remarks:

I, ***jr*** 1a. When the EG*p1*? command is executed, its value will be given as a Boolean number. A value of 1 means DIGITAL INPUT *p1* made a low to high state change since its EG value was last read (i.e., cleared).

I, ***jr*** 2a. When EG? is executed, the positive-edge-sensitive digital inputs will be reported as binary numbers. The left-most bit represents digital input 12, 14, or 21, depending on your controller model (see ***Parameters*** above); the right-most bit represents digital input 1.

⊙ 1b. When the EG*p1.p2*? command is executed, the value of *p2* of digital module *p1* will be given as a Boolean number.

⊙	2b. When EG*p1*? is executed, positive-edge-sensitive digital inputs 1 through 32 of digital module *p1* will be reported as a binary number with the bits in groups of four. The left-most bit represents digital input 32; the right-most bit represents digital input 1.
I, ***jr,*** ⊙	3. After the state of an input is read using the EG command, the EG value of that input is set to zero.
I, ***jr,*** ⊙	4. When setting the positive-edge-sensitive digital inputs, note that a zero will reset the input, and a 1 will not change the state of the input.

Examples:

IMC/IMJ	**Target ARS**	
EG=16#1A0	EG1=16#1A0	(* set EG to $1A0_{16}$, [i.e., don't (*change inputs 6, 8, and 9, but (* reset all others])
EGVI1=0	EG1.VI1=0	(* set EG VI1 to 0 [i.e., reset the (* input])
EG?	EG1?	(* report positive-edge-sensitive (* register)

Related Registers: DI

EKB Empties Key Buffer

Class:	Input/Output Command
Syntax:	EKB
Restrictions:	Not allowed in motion blocks.
Use:	This command empties the key buffer.
Related Commands:	KY, GET, GETW, IN, INW, WKY
Related Registers:	KEY, KEYW

END Ends Program or Motion Block and Exits Editor

Class:	Program Command
Syntax:	END
Restrictions:	Allowed only in programs or motion blocks.
Use:	This command marks the end of a program or motion block and exits the terminal window line editor.
Remarks:	**Caution:** When used in the terminal window line editor this command will delete all program/motion block statements that follow it. If you want only to exit the editor, use the ! command.

Examples:

```
IMC/IMJ    Target ARS
PROGRAM1   PROGRAM1  (* define program 1)
PSA=0      PSA1=0    (* set axis position register)
MVL=10     MVL1=10   (* set motion velocity)
MAC=40     MAC1=40   (* set motion acceleration)
MPA=12     MPA1=12   (* set absolute move position)
RPA        RPA1      (* run to absolute position)
END        END       (* end program 1 and exit editor)
```

Related Commands: !

EOT Encoder Output Type *jr*

Class:	Axis Register
Type:	Integer
Syntax:	EOT
Range:	
units	lines per revolution
default	0
allowed values	Resolver Feedback Controllers: 0; 250; 256; 500; 512; 1,000; 1,024 Encoder Feedback Controllers: 0; 500; 625; 1,000; 1,250; 2,000; 2,500
Restrictions:	Brushless servo only; not allowed in motion blocks.
Use:	This register sets the output type for the encoder output. When this register is set to zero, the encoder output buffers the encoder input. When the register is non-zero, the encoder output tracks the motor feedback. The lines per revolution of the motor is set by the number entered in the register.

Examples:

EOT=0	(* encoder output buffers encoder input)
EOT=1000	(* encoder output provides 1,000 lines per revolution of the motor)

ETB Empties Tertiary Port Buffer

Class:	Input/Output Command
Syntax:	ETB
Restrictions:	Not allowed in motion blocks.
Use:	The command empties the tertiary port buffer.
Related Commands:	INT, GETT
Related Registers:	KEYT

EUB Empties User Port Buffer

Class:	Input/Output Command
Syntax:	EUB
Restrictions:	Not allowed in motion blocks.
Use:	The command empties the user port buffer.
Related Commands:	GET, IN
Related Registers:	KEY

EXM Executes Motion Block

Class:	Program Command	
Syntax:	EXM*p1* (e.g., EXM50 EXMVI10)	
Parameters:	*allowed values*	*description*
I, ***jr*** *p1*	1 through 100 **or** VI*n*	motion block number
⊙ *p1*	1 through 400 **or** VI*n*	motion block number

Restrictions: Not allowed in motion blocks.

Use: This command executes motion block *p1*. Motion blocks behave like run macros. They are not programs and are not killed by KLALL. Use HT or ST to end a motion block.

Remarks:

I, ***jr*** If a motion block is executing, the EXM command will quit executing that motion block and then execute motion block *p1*. If motion block *p1* is already executing, EXM*p1* will restart it. One motion block cannot start another motion block.

⊙ If a motion block that has the same axes assigned as motion block *p1* is executing, the EXM command will quit executing that motion block and then execute motion block *p1*. If motion block *p1* is already executing, EXM*p1* will restart it. One motion block cannot start another motion block.

Examples:

```
IMC/IMJ  Target ARS

MOTION1  MOTION1  (* edit motion block 1)
         MBA1     (* assign axis one to motion block)
MVL=10   MVL1=10  (* set motion velocity)
MAC=40   MAC1=40  (* set motion acceleration)
MPI=15   MPI1=15  (* set incremental move position)
RPI      RPI1     (* run to incremental move position)
END      END      (* end motion block 1 and exit editor)

EXM1     EXM1     (* execute motion block 1)
```

What will happen: Issuing the EXM1 command will cause the axis to move 15 units in the forward direction.

Related Commands: EXP

EXP Executes Program

Class:	Program Command	
Syntax:	EXP*p1* (e.g., EXP4 EXPVI9)	
Parameters:	*allowed values*	*description*
I, ***jr*** *p1*	1 through 4 **or** VI*n*	program number
⊙ *p1*	1 through 17 **or** VI*n*	program number
Restrictions:	Not allowed in motion blocks.	
Use:	This command executes program *p1*.	
Remarks:	If program *p1* is already executing, then this command does nothing.	

Examples:

```
IMC/IMJ     Target ARS
PROGRAM1    PROGRAM1  (* edit program 1)
PSA=0       PSA1=0    (* set axis position register)
MVL=10      MVL1=10   (* set motion velocity)
MAC=40      MAC1=40   (* set motion acceleration)
MPA=12      MPA1=12   (* set absolute move position)
RPA         RPA1      (* run to absolute position)
END         END       (* end program 1 and exit editor)

EXP1        EXP1      (* execute program 1)
```

What will happen: Issuing the EXP1 command will cause the axis to move 12 units in the forward direction.

Related Commands: EXM

EXVS Execute Command Stored in String Variable

Class:	Program Command	
Syntax:	EXVS*p1* (e.g., EXVS12 EXVSVI6)	
Parameters:	*allowed values*	*description*
p1	1 through 144 **or** VI*n*	string variable number

Restrictions: Not allowed in motion blocks.

Use: This command executes the command stored in string variable *p1*.

Remarks: Commands that are not allowed in programs cannot be executed using EXVS.

Examples:

IMC/IMJ	**Target ARS**	
VS1="MPA=10"	VS1="MPA1=10"	(* set string variable 1)
EXVS1	EXVS1	(* execute command stored in string variable 1)

What will happen: Loading string variable 1 and executing the command stored in string variable 1 will set the absolute move position, MPA, to 10 units.

FAULT Enters Editor at Faulting Statement

Class:	Program Command
Syntax:	FAULT
Restrictions:	Not allowed in programs or motion blocks.
Use:	This command enters the editor and makes the statement that faulted the system the current statement.
Remarks:	This command will execute only when all axes have stopped and no programs or motion blocks are executing.

Examples:

```
IMC/IMJ    Target ARS
PROGRAM1   PROGRAM1   (* edit program 1)
PSA=0      PSA1=0     (* set axis position register)
STF        STFS       (* set fault)
END        END        (* end program 1 and exit editor)

EXP1       EXP1       (* execute program 1)
FAULT      FAULT      (* enter editor and make statement
                      that faulted system the current
                      statement)
*STF       *STFS
```

FC Fault Code *I jr*

Class:	System Register	
Type:	Integer, Boolean	
Syntax:	FC*p1* (e.g., FC FC5 FCVI3)	
Parameters:	*allowed values*	*description*
p1	none **or** 0 through 31 **or** VI*n*	fault code register bit number
Range:		
allowed values	0 through $FFFFFFFF_{16}$ **or** 0 and 1	
Restrictions:	Read only.	
Use:	The fault code register is used to identify what type of fault has taken place.	

Remarks:

1. When the FC? command is executed from the terminal window, the fault code register value will be given as an English statement. If no fault has occurred, the message given is *Controller functional*.
2. The Fault Code register is latched. Once a bit is set true it will not be cleared until faults are reset (RSF command executed).
3. When FCx is executed the Boolean status of bit ‘x’ will be given.
4. If the computer interface format is enabled (CIE=1), and the FC? command is executed, the fault code register value will be given as an integer number equal to the decimal equivalent of the register’s binary value. If no fault has occurred, the fault code register is set to 0. The possibilities are listed below:

bit	*message*	*bit*	*message*
0	Power Failure	20	Duplicate Network Address
1	Reserved	21	Excessive Following Error
2	Software Fault	22	Excessive Command Increment
3	Lost Enable	23	Position Register Overflow
4	Digital Output Fault	24	Position Feedback Lost
5	Invalid Command in String	25	Motor Power Over-Voltage
6	Transmit Buffer Overflow	26	*(3 & 4.3 Amp IMJ)* Motor Power Clamp Excessive Duty Cycle
7	Resource Not Available		*(3 & 6 Amp IMC; 7 Amp IMJ)* Motor Power Clamp Excessive
8	Invalid Variable Pointer		Duty Cycle—Under-Voltage
9	Mathematical Overflow		*(12–28 Amp)* Motor Power Under-Voltage
10	Mathematical Data Error	27	*(3 & 4.3 Amp IMJ)* Reserved
11	Value Out of Range		*(3 & 6 Amp IMC; 7 Amp IMJ)* Motor Power Clamp
12	String Too Long		Over-Current Fault
13	Nonexistent Label		*(12–28 Amp)* Motor Power Clamp Excessive Duty Cycle
14	Gosub Stack Underflow	28	Motor Over-Current Fault
15	Gosub Stack Overflow	29	Motor Over-Temperature
16	Invalid Motion	30	Controller Over-Temperature
17	Reserved	31	Network Communication Error
18	Reserved		
19	Network Power Failure		

FCA Axis Fault Code

Class:	System Register	
Type:	Integer, Boolean	
Syntax:	FCA*p1.p2* (e.g., FCA1 FCAVI1.3 FCA2.VI3 FCAVI1.VI2)	
Parameters:	*allowed values*	*description*
p1	1 through 8 **or** VI*n*	axis number
p2	none **or** 0 through 31 **or** VI*n*	axis fault code register bit number
Range:		
allowed values	0 through $FFFFFFFF_{16}$ **or** 0 and 1	

Restrictions: Read only.

Use: The axis fault code register is used to identify what type of axis fault has taken place.

Remarks: 1. When the FCA*p1*? command is executed, the axis fault code will be given as an English statement that says which axis faults have occurred. If no axis fault has occurred, the message given is *Axis functional*.
2. If the computer interface format is enabled, and the FCA*p1*? command is executed, the axis fault code will be given as an integer number. If no axis fault has occurred, the axis fault code register is set to 0. The possibilities are listed below:

bit	*message*	*bit*	*message*
0	Power Failure	11	Motor Power Clamp Current Fault
1	Encoder Supply Fault	12	Servo Module Current Fault
2	Software Fault	13	Servo Module Over-Temperature
3	Lost Enable	14	Power Module Over-Temperature
4	Excessive Following Error	15	Motor Over-Temperature
5	Excessive Command Increment	16–19	Reserved
6	Position Register Overflow	20	Set Point Output Fault
7	Position Feedback Lost	21–23	Reserved
8	Motor Power Under-Voltage	24	System Communication Error
9	Motor Power Over-Voltage	25	Servo Module Communication Error
10	Motor Power Clamp Excessive Duty Cycle	26–30	Reserved
		31	Servo Module Assignment Error

FCS System Fault Code

Class:	System Register	
Type:	Integer, Boolean	
Syntax:	FCS*p1* (e.g., FCS FCS2 FCSVI1)	
Parameters:	*allowed values*	*description*
p1	none **or** 0 through 31 **or** VI*n*	system fault code register bit number
Range:		
allowed values	0 through FFFFFFFF$_{16}$ **or** 0 and 1	
Restrictions:	Read only.	
Use:	The system fault code register is used to identify what type of system fault has taken place.	

Remarks:

1. When the FCS? command is executed, the system fault code register value will be given as an English statement. If no system fault has occurred, the message given is *System functional*.
2. If the computer interface format is enabled, and the FCS? command is executed, the system fault code register value will be given as an integer number. If no system fault has occurred, the system fault code register is set to 0. The possibilities are listed below:

bit	*message*	*bit*	*message*
0	Power Failure	16	Invalid Motion
1	24 Volt Supply Fault	17	Inconsistent Axis Groupings
2	Software Fault	18	Duplicate Network Address
3	Lost Enable	19	Network Power Failure
4	Digital Output Fault	20	Set Point Output Fault
5	Invalid Command in String	21	Tertiary Transmit Buffer Overflow
6	User Transmit Buffer Overflow	22	Program Transmit Buffer Overflow
7	Resource Not Available	23	Firmware Load Error
8	Invalid Variable Pointer	24	Axis Communication Error
9	Mathematical Overflow	25	I/O Communication Error
10	Mathematical Data Error	26	User Port Communication Error
11	Value Out of Range	27	Network Communication Error
12	String Too Long	28	Axis Assignment Error
13	Nonexistent Label	29	Analog Module Assignment Error
14	Gosub Stack Underflow	30	Digital Module Assignment Error
15	Gosub Stack Overflow	31	Servo Module Assignment Error

FE Axis Following Error

Class:	Axis Register	
Type:	Floating point	
Syntax:		
I, jr	FE	
⊙	FE*p1* (e.g., FE1 FEVI3)	
Parameters:	*allowed values*	*description*
⊙ *p1*	1 through 8 **or** VI*n*	axis number
Range:		
units	axis units	
minimum	0 pulses	
maximum	16,000 pulses	
Restrictions:	Read only.	
Use:	The axis following error is the difference between the axis position, PSA, and the command position, PSC.	
Remarks:	The numerical values for the minimum and maximum of this register are assuming that the axis unit ratio, URA, is set at its default value of 1. If the axis unit ratio is set to a value other than 1, the minimum and maximum values will change appropriately (see URA).	
Related Registers:	PSA, PSC, FEB, URA	

FEB Following Error Bound

Class:	Axis Register	
Type:	Floating point	
Syntax:		
I, jr	FEB	
⊙	FEB*p1* (e.g., FEB1 FEBVI3)	
Parameters:	*allowed values*	*description*
⊙ *p1*	1 through 8 **or** VI*n*	axis number
Range:		
units	axis units	
defaults	400 pulses (resolver feedback brushless servo) 5,000 pulses (2,500 line count encoder servo) 5,000 pulses (stepper) 100 pulses (ampless servo)	
minimum	0 pulses	
maximum	16,000 pulses	

Use: The following error bound is a limit set on the following error. If this limit is exceeded, the system will fault and the motor will free-wheel to a stop.

Remarks: **This value must always be set to a non-zero value. If FEB is set to zero the controller will fault when initiating any motion command or block.** The numerical values for the default, minimum, and maximum of this register are assuming that the axis unit ratio, URA, is set at its default value of 1. If the axis unit ratio is set to a value other than 1, the default, minimum, and maximum values will change appropriately (see URA).

Examples:

IMC/IMJ	**Target ARS**
FEB=0.5	FEB1=5 (* set following error bound)
FEB?	FEBVI3? (* report value of following error bound)

Related Registers: FE, URA

FI Fault Input *I jr*

Class:	System Register	
Type:	Integer, Boolean	
Syntax:	FI*p1* (e.g., FI FI8 FIVI7)	
Parameters:	*allowed values*	*description*
p1	none **or** 0 through 15 **or** VI*n*	fault input register bit number
Range:		
allowed values	0 through $FFFF_{16}$ **or** 0 and 1	
Restrictions:	Read only.	
Use:	The fault input register is used to identify what type of faults are currently active.	

Remarks:

1. When the FI? command is executed from the terminal window, the fault input register value will be given as an English statement as shown below. If no faults are active, the message given is *No fault input is active*.
2. If the computer interface format is enabled, and the FI? command is executed, the fault input register value will be given as an integer number equal to the decimal equivalent of the register's binary value. If no faults are active, the fault input register is set to 0. The possibilities are listed below.
3. When FIx is executed the Boolean status of bit 'x' will be given.

bit	*message*
0	Position feedback lost input active
1	Motor power over-voltage input active
2	*(3 Amp IMJ)* Motor power clamp input active *(3 & 6 Amp IMC; 7 Amp IMJ)* Motor power clamp or under-voltage input active *(12–28 Amp)* Motor power under-voltage input active
3	*(3 Amp IMJ)* Reserved *(3 & 6 Amp IMC; 7 Amp IMJ)* Motor power clamp over-current input active *(12–28 Amp)* Motor power clamp input active
4	Motor over-current input active
5	Motor over-temperature input active
6	Controller over-temperature input active
7	Network power failure input active
8–15	Reserved

FIRMWARE Downloads and Saves Firmware

Class:	System Command
Syntax:	FIRMWARE
Restrictions:	Not allowed in programs or motion blocks.
Use:	This command, when executed from the terminal window, sets the controller in a mode to receive an updated firmware file, downloads the controller or system firmware, and saves it in nonvolatile memory.
Remarks:	This command will execute only when the controller or system and all axes are faulted and no programs or motion blocks are executing.

FR Axis Feedback Resolution

Class:	Axis Register	
Type:	Integer	
Syntax:		
I, jr	FR	
⊙	FR*p1* (e.g., FR1 FRVI3)	
Parameters:	*allowed values*	*description*
⊙ *p1*	1 through 8 **or** VI*n*	axis number
Range:		
units	pulses/revolution	
defaults	4,096 (resolver feedback brushless servo) 10,000 (2,500 line count encoder servo) 1,000 (ampless servo)	
minimum	500	
maximum	1,000,000	
Restrictions:	Servo only.	
Use:	The axis feedback resolution is defined as the number of feedback pulses per revolution of the axis.	
Related Commands:	AUTOTUNE	

FUNCTION Goes to Label Associated with Key Pressed

Class: Input/Output Command

Syntax: FUNCTION *p1, p2, p3, p4, p5, p6, p7, p8, p9, p10, p11, p12*

Parameters:

	allowed values	*description*
p1	0, 1 through 999	label associated with function key A
...	...	...
p12	0, 1 through 999	label associated with function key L

Restrictions: Allowed only in programs.

Use: This command, when executed in a program, first fetches the key code from the key buffer. If there is no key in the key buffer, it will wait for a key to be pressed. If a function key has been pressed, program execution is then transferred to the statement at the label associated with the function key pressed. If any other key has been pressed, the key code goes back into the key buffer and execution continues at the next program statement.

Remarks: If one or more of the function keys have been disabled by setting KYA*p1* to OFF, where *p1* is the number of the function key, it is appropriate to set the associated label(s) in the FUNCTION statement equal to 0.

Examples:

```
IMC/IMJ               Target
PROGRAM1              PROGRAM1              (* edit program 1)
PSA=0                 PSA1=0                (* set axis position)
MVL=5                 MVL1=5                (* set motion velocity)
MAC=40                MAC1=40               (* set motion acceleration)
5 FUNCTION 10,20,     5 FUNCTION 10,20,
30,5,5,5,5,5,5,5,5,5  30,5,5,5,5,5,5,5,5,5  (* go to label associated with key pressed)
GET VS1               GETW VS1              (* get character from key buffer)
GOTO 5                GOTO 5                (* go back and wait for another key press)
10 RVF                10 RVF1               (* run forward)
GOTO 5                GOTO 5                (* go back and wait for another key press)
20 RVR                20 RVR1               (* run reverse)
GOTO 5                GOTO 5                (* go back and wait for another key press)
30 ST                 30 ST1                (* stop axis)
WAIT IP               WAIT IP1              (* wait for axis to be in position)
GOTO 5                GOTO 5                (* go back and wait for another key press)
END                   END                   (* end program 1 and exit editor)
```

What will happen: This program, once executed, will set the axis position, motion velocity, and acceleration. It will then wait for a key to be pressed, and then the program execution will go to label 10, 20, 30, or 5, depending on which function key was pressed. If some other key was pressed, then the key code is taken out of the key buffer (GET VS1) and execution goes back to label 5.

Related Commands: GOTO

GET Gets One Character from Key Buffer *I jr*

Class:	Input/Output Command	
Syntax:	GET *p1* (e.g., GET VI5 GET VS10)	
Parameters:	*allowed values*	*description*
p1	any variable register	variable register
Restrictions:	Allowed only in programs.	

Use: This command gets one character from the key buffer (256 bytes maximum) and loads it into the variable *p1*. If no character is available in the key buffer, then this command waits until a character is put into the key buffer.

Remarks:
1. If *p1* is a Boolean variable, VB*n* or VBVI*n*, the resulting value will be 0 if the character is ASCII 0; otherwise the resulting value is 1.
2. If *p1* is a floating point or integer variable, VF*n*, VFVI*n*, VI*n*, VIVI*n*, the resulting value will be the ASCII value of the character.
3. If *p1* is a string variable, VS*n* or VSVI*n*, the resulting value is the actual character.

Example:

```
PROGRAM1        (* edit program 1)
GET VI1         (* get one character from the key buffer)
GET VS1         (* get one character from the key buffer)
END             (* end program 1 and exit editor)

EXP1            (* execute program 1)
KYE             (* put one character into key buffer)
KYE             (* put one character into key buffer)
VI1?            (* report value of integer variable register)
* 69
VS1?            (* report value of string variable register)
* E
```

Related Commands: PUT, IN, OUT, EKB

GET — Gets One Character from User Serial Port

Class:	Input/Output Command	
Syntax:	GET *p1* (e.g., GET VI5 GET VS10)	
Parameters:	*allowed values*	*description*
p1	any variable register	variable register
Restrictions:	Allowed only in programs	

Use: This command gets one character from the user serial port and loads it into the variable *p1*.

Remarks:
1. If *p1* is a Boolean variable, VB*n* or VBVI*n*, the resulting value will be 0 if the character is ASCII 0; otherwise the resulting value is 1.
2. If *p1* is a floating point or integer variable, VF*n*, VFVI*n*, VI*n*, VIVI*n*, the resulting value will be the ASCII value of the character.
3. If *p1* is a string variable, VS*n* or VSVI*n*, the resulting value is the actual character.

Example:

```
PROGRAM1          (* edit program 1)
GET VI1           (* get one character from the user serial port buffer)
GET VS1           (* get one character from the user serial port buffer)
END               (* end program 1 and exit editor)
```

Related Commands: PUT, IN, OUT, EUB

GETT Gets One Character from Tertiary Port

Class:	Input/Output Command	
Syntax:	GETT *p1* (e.g., GETT VI5 GETT VS10)	
Parameters:	*allowed values*	*description*
p1	any variable register	variable register
Restrictions:	Allowed only in programs.	
Use:	This command gets one character from the tertiary port and loads it into the variable *p1*. If no character is available, then this command waits until a character is available.	
Remarks:	1. If *p1* is a Boolean variable, VB*n* or VBVI*n*, the resulting value will be 0 if the character is ASCII 0; otherwise the resulting value is 1. 2. If *p1* is a floating point or integer variable, VF*n*, VFVI*n*, VI*n*, VIVI*n*, the resulting value will be the ASCII value of the character. 3. If *p1* is a string variable, VS*n* or VSVI*n*, the resulting value is the actual character.	

Example:

```
PROGRAM1        (* edit program 1)
GETT VI1        (* get one character from the tertiary port buffer)
GETT VS1        (* get one character from the tertiary port buffer)
END             (* end program 1 and exit editor)
```

Related Commands: PUTT, INT, OUTT, ETB

GETW Gets One Character from Key Buffer

Class:	Input/Output Command	
Syntax:	GETW *p1* (e.g., GETW VI5 GETW VS10)	
Parameters:	*allowed values*	*description*
p1	any variable register	variable register
Restrictions:	Allowed only in programs	
Use:	This command gets one character from the key buffer and loads it into the variable *p1*. If no character is available, then this command waits until a character is available.	
Remarks:	1. If *p1* is a Boolean variable, VB*n* or VBVI*n*, the resulting value will be 0 if the character is ASCII 0; otherwise the resulting value is 1. |2. If *p1* is a floating point or integer variable, VF*n*, VFVI*n*, VI*n*, VIVI*n*, the resulting value will be the ASCII value of the character. 3. If *p1* is a string variable, VS*n* or VSVI*n*, the resulting value is the actual character.	

Example:

```
PROGRAM1        (* edit program 1)
GETW VI1        (* get one character from the key buffer)
GETW VS1        (* get one character from the key buffer)
END             (* end program 1 and exit editor)

EXP1            (* execute program 1)
WKYE            (* put one character into key buffer)
WKYE            (* put one character into key buffer)
VI1?            (* report value of integer variable register)
* 69
VS1?            (* report value of string variable register)
* E
```

Related Commands: PUTW, INW, OUTW, EKB

GOSUB Unconditionally “Gosubs” Label

Class: Program Command

Syntax: GOSUB*p1* (e.g., GOSUB349 GOSUBVI10)

Parameters:	*allowed values*	*description*
p1	1 through 999 **or** VI*n*	label number

Restrictions: Allowed only in programs.

Use: This command causes the program execution to go unconditionally to the subroutine at label *p1*. The program will return to the line immediately following the GOSUB command when it encounters the RETURN command.

Remarks: There can be up to 32 nested gosub statements in a program.

Examples:

```
IMC/IMJ                               Target
PROGRAM1                              PROGRAM1                    (* edit program 1)
PSA=0                                 PSA1=0                      (* set axis position register)
MVL=1                                 MVL1=1                      (* set motion velocity)
MAC=10                                MAC1=10                     (* set motion acceleration)
RVF                                   RVF1                        (* run to velocity forward)
GOSUB5                                GOSUB5                      (* unconditionally gosub 5)
VI1=6                                 VI1=6                       (* load integer variable)
GOSUBVI1                              GOSUBVI1                    (* unconditionally gosub 6)
GOTO10                                GOTO10                      (* unconditionally goto 10)
5  OUT “Press any key to stop axis $N”  5 OUTW “Press any key to stop axis $N”
                                                                  (* output string expression to display)
GET VI2                               GETW VI2                    (* get one character from key buffer)
ST                                    ST 1                        (* stop axis)
RETURN                                RETURN                      (* return from gosub)
6 OUT “Axis  position is “+ FTS(PSA,  6 OUTW “Axis position is “+ FTS(PSA1,
   5,2) + “ units.$N”                 5,2) + “ units.$N”          (* output string expression to display)
RETURN                                RETURN                      (* return from gosub)
10 END                                10 END                      (* end program 1 and exit editor)
```

What will happen: This program, once executed, runs the axis in the forward direction. Then the execution goes to the subroutine at label 5, which waits for a character from the key buffer and returns upon receiving the character. Next, the execution goes to the subroutine at label 6, which prints the axis position on the display and returns. It then goes to the statement at label 10, which ends the program.

Related Commands: GOTO, RETURN, POP, RSTSTK

GOTO Unconditionally “Gotos” Label

Class:	Program Command	
Syntax:	GOTO*p1* (e.g., GOTO50 GOTOVI43)	
Parameters:	*allowed values*	*description*
p1	1 through 999 **or** VI*n*	label number
Restrictions:	Allowed only in programs.	
Use:	This command causes the program execution to go unconditionally to the statement at label *p1*.	

Examples:

```
IMC/IMJ                             Target
PROGRAM1                            PROGRAM1             (* edit program 1)
PSA=0                               PSA1=0               (* set axis position register)
MVL=1                               MVL1=1               (* set motion velocity)
MAC=10                              MAC1=10              (* set motion acceleration)
RVF                                 RVF1                 (* run to velocity forward)
GOSUB5                              GOSUB5               (* unconditionally gosub 5)
VI1=6                               VI1=6                (* load integer variable)
GOSUBVI1                            GOSUBVI1             (* unconditionally gosub 6)
GOTO10                              GOTO10               (* unconditionally goto 10)
5 OUT “Press any key to stop axis $N”  5 OUTW “Press any key to stop axis $N”
                                                         (* output string expression to display)
GET VI2                             GETW VI2             (* get one character from key buffer)
ST                                  ST1                  (* stop axis)
RETURN                              RETURN               (* return from gosub)
6 OUT “Axis position is “ + FTS(PSA,   6 OUTW “Axis position  is “+ FTS(PSA1,
   5,2) + “ units.$N”                  5,2) + “ units.$N”   (* output string expression to display)
RETURN                              RETURN               (* return from gosub)
10 END                              10 END               (* end program 1 and exit editor)
```

What will happen: This program, once executed, runs the axis in the forward direction. Then the execution goes to the subroutine at label 5, which waits for a character from the key buffer and returns upon receiving the character. Next, the execution goes to the subroutine at label 6, which prints the axis position on the display and returns. It then goes to the statement at label 10, which ends the program.

Related Commands: GOSUB

GRB Gearing Bound

Class:	Motion Register	
Type:	Floating point	
Syntax:		
I, jr	GRB	
⊙	GRB*p1* (e.g., GRB2 GRBVI5)	
Parameters:	*allowed values*	*description*
⊙ *p1*	1 through 8 **or** VI*n*	axis number
Range:		
units	axis units/sec	
default	0 pulses/sec	
minimum	0 pulses/sec	
maximum	16,000,000 pulses/sec	

Restrictions: For IMCs, this function available only with the extended command set.

Use: This register sets a bound on the maximum axis pulses per second that the electronic gearing function can command. If the pulse input rate times the gearing ratio, GRN/GRD, results in a value outside of the bound, then the extra pulses are discarded (i.e., the rate is clamped at the bound limit). When the value of GRB is zero, there is no bound on electronic gearing.

Remarks: The numerical values for the default, minimum, and maximum of this register are assuming that the axis unit ratio, URA, is set at its default value of 1. If the axis unit ratio is set to a value other than 1, the default, minimum, and maximum values must be divided by the value of URA (see URA).

Related Registers: GRN, GRD

GRD Gearing Denominator

Class:	Motion Register	
Type:	Integer	
Syntax:		
I, jr	GRD	
⊙	GRD*p1* (e.g., GRD2 GRDVI3)	
Parameters:	*allowed values*	*description*
⊙ *p1*	1 through 8 **or** VI*n*	axis number
Range:		
default	1	
minimum	1	
maximum	10,000	

Restrictions: For IMCs, this function available only with the extended command set.

Use: The gearing denominator is a parameter used in electronic gearing. It is defined as the denominator of the gearing ratio between the axis and the gearing input. The gearing input source is typically the auxiliary encoder input unless the handwheel input is enabled (HWE=1). Change the sign on the GRN parameter to change motor direction while gearing is enabled (GRE=1).

Axis pulses = gearing input pulses * GRN/GRD.

If either GRN or GRD is outside the allowed range, try dividing both register values by a prime number (2, 3, 5, 7, 11, etc.) until both values are integers within the allowable range.

Related Registers: GRN, GRE, GRI, HWE, QTX

GRE Gearing Enable

Class:	Motion Register	
Type:	Boolean	
Syntax:		
I, jr	GRE	
⊙	GRE*p1* (e.g., GRE2 GRE245 GREVI3)	
Parameters:	*allowed values*	*description*
⊙ *p1*	1 through 8 **or** list of numbers 1 through 8 **or** VI*n*	axis number
Range:		
default	0	
allowed values	0, 1	
Restrictions:	For IMCs, this function available only with the extended command set.	
Use:	The gearing enable is used to enable electronic gearing. If GRE is set to 1, then electronic gearing is enabled and the axis will follow the gearing input based on the gearing ratio (GRN/GRD). If GRE is set to 0, it is disabled.	
Remarks:	Electronic gearing does not use acceleration/deceleration limits and will accelerate/decelerate as quickly as system constraints will allow when the GRE bit is set true/false. Use pulse-based motion when acceleration limits are required. When the gearing enable bit is set true the controller will begin to accumulate master encoder pulses. If gearing is enabled while the master is moving the axis will overspeed within system constraints in an attempt to decrement any master pulses that accumulate while the axis is accelerating. Gearing is automatically disabled when a controller fault occurs.	
Registers Used:	GRD, GRI, GRN, GRB, GRF	
Motion Templates:		
I, jr	Single-axis electronic gearing	
⊙	Multi-axis electronic gearing	
Utility Template:		
⊙	Jog using analog input	

GRF Gearing Filter Constant

Class:	Motion Register	
Type:	Integer	
Syntax:		
I, jr	GRF	
⊙	GRF*p1* (e.g., GRF2 GRFVI3)	
Parameters:	*allowed values*	*description*
⊙ *p1*	1 through 8 **or** VI*n*	axis number
Range:		
default	0	
minimum	0	
maximum	8	
Restrictions:	For IMCs, this function available only with the extended command set.	
Use:	The gearing filter constant is used to filter the output of electronic gearing. The amount of filtering increases by the value as a power of two from 0 (no filter) to 8 (a filter of 256 samples). Note that higher values slow system response so use the smallest acceptable value.	
Related Registers:	GRB, GRN, GRD	

GRI Gearing Input

Class:	Motion Register	
Syntax:	GRI*p1* (e.g., GRI2 GRIVI3)	
Parameters:	*allowed values*	*description*
p1	1 through 8 **or** VI*n*	axis number

Range:

default	FREQ	
allowed values	FREQ	frequency source (2,048 pulses/sec)
	PSX*a*	auxiliary input of selected axis (*a*: 1 through 8)
	PSC*a*	command position of selected axis (*a*: 1 through 8)
	PSA*a*	axis position of selected axis (*a*: 1 through 8)

Use: The gearing input is used in electronic gearing as a source for position information for axis *p1*. This, along with the gearing ratio (defined by GRN*p1* and GRD*p1*), defines the motion of axis *p1*.

Example:

```
GRI1=PSA5        (* set gearing input for axis one to the axis position of axis five)
GRI2=FREQ        (* set gearing input for axis two to the frequency source)
GRIVI1?          (* report gearing input for axis VI1)
```

Related Registers: GRN, GRD, GRE

GRN Gearing Numerator

Class:	Motion Register	
Type:	Integer	
Syntax:		
I, jr	GRN	
⊙	GRN*p1* (e.g., GRN2 GRNVI3)	
Parameters:	*allowed values*	*description*
⊙ *p1*	1 through 8 **or** VI*n*	axis number
Range:		
default	1	
minimum	-10,000	
maximum	10,000	

Restrictions: For IMCs, this function available only with the extended command set.

Use: The gearing numerator is a parameter used in electronic gearing. It is defined as the numerator of the gearing ratio between the axis and the gearing input. The gearing input source is typically the auxiliary encoder input unless the handwheel input is enabled (HWE=1). Changing the sign of the GRN value will change the direction of the motor while gearing is enabled (GRE=1).

Axis pulses = gearing input pulses * GRN/GRD.

If either GRN or GRD is outside the allowed range, try dividing both register values by a prime number (2, 3, 5, 7, 11, etc.) until both values are integers within the allowable range.

Related Registers: GRD, GRE, GRI, HWE, QTX

HSE XON, XOFF Handshake Protocol Enable

Class:	System Register
Type:	Boolean
Syntax:	HSE
Range:	
default	0
allowed values	0, 1
Restrictions:	Cannot be assigned in motion blocks.
Use:	This register is used to enable the XON, XOFF handshake protocol on the serial/program port. If HSE is set to 1, then handshake protocol is enabled; and if HSE is set to 0, then it is disabled.
Related Registers:	CIE

HT Halts Motion

Class:	Motion Command	
Syntax:		
I, jr	HT	
⊙	HT*p1* (e.g., HT HT5 HT146 HTVI3)	
Parameters	*allowed values*	*description*
⊙ *p1*	none **or** 1 through 8 **or** list of numbers 1 through 8 **or** VI*n*	axis number
Restrictions:		
⊙	Not allowed in motion blocks without specified axis.	

Use: This command immediately halts all axis motion.

Remarks: This command should be used only at low velocities or in extreme situations as the sudden stop may damage mechanical components in the system.

Examples:

IMC/IMJ	**Target ARS**	
MVL=10	MVL1=10	(* set motion velocity)
MAC=10	MAC1=10	(* set motion acceleration)
RVF	RVF1	(* run to velocity forward)
HT	HT1	(* halt motion)

What will happen: Setting the velocity and acceleration and issuing the RVF command will cause the axis will to run in the forward direction. Issuing the HT command will cause the axis to halt immediately.

Related Commands: ST, HTT

HTT Halts Trajectory Motion

Class:	Motion Command
Syntax:	HTT
Use:	This command immediately halts trajectory motion.
Remarks:	This command should be used only at low velocities or in extreme situations as the sudden stop may damage mechanical components in the system.

Example:

```
TVL=5          (* set trajectory velocity)
TFP=100        (* set trajectory feedrate to 100 percent)
TFA=500        (* set trajectory feedrate acceleration)
MPI1=10        (* set incremental position)
MPI2=20        (* set incremental position)
RLI12          (* run incremental linear)
HTT            (* halt trajectory motion)
```

What will happen: Setting the trajectory velocity, trajectory feedrate acceleration, and incremental positions and issuing the RLI command will cause axes one and two to move in a line. Issuing the HTT command will cause the axes to halt immediately.

Related Commands: STT, HT

HWE Handwheel Input Enable *I jr*

Class:	Motion Register
Type:	Boolean
Syntax:	HWE
Range:	
default	0
allowed values	0, 1
Restrictions:	For IMCs, this function available only with the extended command set.
Use:	The handwheel input enable is used to enable handwheel quadrature input on digital inputs 5 (channel A) and 6 (channel B) to be used in place of the auxiliary encoder input for electronic gearing. If HWE is set to 1, then handwheel input is enabled; and if HWE is set to 0, it is disabled; and the auxiliary encoder is used as the electronic gearing input source. The axis will follow the auxiliary input based on the values of GRN and GRD as shown below:
	Axis pulses = Handwheel Input Pulses * GRN/GRD
Remarks:	The electronic handwheel is used in place of the auxiliary input to position the axis for electronic gearing. The maximum pulse rate is 500 pulses/second.
Utility Template:	Jog using electronic handwheel

IF...GOSUB Conditionally "Gosubs" Label

Class: Program Command

Syntax: IF *p1* GOSUB*p2* (e.g., IF VB5 GOSUB35)

Parameters:

	allowed values	*description*
p1	any Boolean expression	Boolean expression
p2	1 through 999 **or** VI*n*	label number

Restrictions: Allowed only in programs.

Use: This command causes the program execution to go conditionally to the subroutine at label *p2* if *p1* is true (i.e., evaluates to 1). The program will return when it encounters the RETURN command.

Remarks: There can be up to 32 nested gosub statements in a program.

Examples:

```
IMC/IMJ                Target
PROGRAM1               PROGRAM1          (* edit program 1)
PSA=0                  PSA1=0            (* set axis position register)
MVL=1                  MVL1=1            (* set motion velocity)
MAC=10                 MAC1=10           (* set motion acceleration)
RVF                    RVF1              (* run to velocity forward)
OUT "Press any key     OUTW "Press any key
   to stop axis  $N"   to stop axis $N"  (* output string expression to display)
1 IF KEY GOSUB5        1 IF KEYW GOSUB5  (* conditionally gosub 5)
IF IP GOTO10           IF IP1 GOTO10     (* conditionally goto 10)
GOTO1                  GOTO1             (* unconditionally goto 1)
5 OUT "Axis position   5 OUTW "Axis position
   is " + FTS(PSA,       is " + FTS(PSA1,
   5,2) + " units.$N"    5,2) + " units.$N"  (* output string expression to display)
EKB                    EKB               (* empty key buffer)
ST                     ST   1            (* stop axis)
RETURN                 RETURN            (* return from gosub)
10 END                 10 END            (* end program 1 and exit editor)
```

What will happen: This program runs the axis in the forward direction. It then waits for a character from the key buffer and goes to the subroutine at label 5 upon receiving the character. This subroutine prints the axis position on the display, empties the key buffer, stops the axis, and returns. Once the axis is in position (IP or IP1), the execution goes to the statement at label 10, which ends the program.

Related Commands: GOSUB, IF...GOTO, RETURN, POP, RSTSTK

IF...GOTO Conditionally "Gotos" Label

Class: Program Command

Syntax: IF *p1* GOTO*p2* (e.g., IF VB3 GOTO11)

Parameters:

	allowed values	*description*
p1	any Boolean expression	Boolean expression
p2	1 through 999 **or** VI*n*	label number

Restrictions: Allowed only in programs.

Use: This command causes the program execution to go conditionally to label *p2* if *p1* is true (i.e., evaluates to 1).

Examples:

```
IMC/IMJ                 Target
PROGRAM1                PROGRAM1              (* edit program 1)
PSA=0                   PSA1=0                (* set axis position register)
MVL=1                   MVL1=1                (* set motion velocity)
MAC=10                  MAC1=10               (* set motion acceleration)
RVF                     RVF1                  (* run to velocity forward)
OUT "Press any key      OUTW "Press any key
   to stop axis$N"      to stop axis$N"       (* output string expression to display)
1 IF KEY GOSUB5         1 IF KEYW GOSUB5      (* conditionally gosub 5)
IF IP GOTO10                IF IP1 GOTO10     (* conditionally goto 10)
GOTO1                   GOTO1                 (* unconditionally goto 1)
5 OUT "Axis position    5 OUTW "Axis position
   is " + FTS(PSA,          is " + FTS(PSA1,
   5,2) + " units.$N"       5,2) + " units.$N"  (* output string expression to display)
EKB                     EKB                   (* empty key buffer)
ST                      ST   1                (* stop axis)
RETURN                  RETURN                (* return from gosub)
10 END                  10 END                (* end program 1 and exit editor)
```

What will happen: This program, once executed, runs the axis in the forward direction. It then waits for a character from the key buffer and goes to the subroutine at label 5 upon receiving the character. This subroutine prints the axis position on the display, empties the key buffer, stops the axis, and returns. Once the axis is in position (IP or IP1), the execution goes to the statement at label 10, which ends the program.

Related Commands: GOTO, IF...GOSUB, IF...THEN

IF...THEN Conditionally Executes Next Command

Class:	Program Command	
Syntax:	IF *p1* THEN (e.g., IF VF3>1.1 THEN)	
Parameters:	*allowed values*	*description*
p1	any Boolean expression	Boolean expression
Restrictions:	Allowed only in programs and motion blocks.	
Use:	This command conditionally executes the next command in the program. If condition *p1* is true the next program line is executed. Otherwise, the next line is skipped.	

Example:

```
PROGRAM1          (* edit program 1)
VB1=0             (* set Boolean variable)
IF VB1 THEN       (* conditionally execute next command)
VF5=30            (* set floating point variable)
END               (* end program 1 and exit editor)
```

What will happen: This program, once executed, sets Boolean variable one to zero and does not set floating point variable to 30 because the condition of the IF...THEN command was false.

Related Commands: IF...GOTO

IN Inputs Register Value from Key Buffer *I jr*

Class:	Input/Output Command	
Syntax:	IN *p1* (e.g., IN VI5 IN VS10)	
Parameters:	*allowed values*	*description*
p1	any variable register	variable register
Restrictions:	Allowed only in programs.	

Use: This command inputs a register value from the key buffer. The characters entered are echoed back to the display device until an invalid character or a carriage return is entered. If the display format enable is set and an invalid character is entered, then the command aborts and the offending character is left in the key buffer.

Remarks: 1. If *p1* is a Boolean, floating point, or integer variable, VB*n*, VBVI*n*, VF*n*, VFVI*n*, VI*n*, VIVI*n*: a.) if the number is greater than 40 characters long, or if it is out of the numerical range of the variable, then bit 5 in the program status register will be set to 1, which means "String value out of range." A zero will be loaded into the variable. b.) if one or more of the characters is not valid, then bit 4 in the program status register will be set to 1, which means "Invalid digit in string." A zero will be loaded into the variable.

2. If *p1* is a string variable, VS*n,* or VSVI*n*, and the string entered is greater than 127 characters, only the first 127 characters will be loaded. The rest will stay in the key buffer.

Example:

```
     PROGRAM1                    (* edit program 1)
     OUT "Enter an integer:$N"   (* output string expression to display)
1    IN VI1                      (* input register value from key buffer)
     IF NOT CE1 GOTO2            (* conditionally goto 2)
     OUT "Invalid number -
        Enter again$N"           (* output string expression to display)
     EKB                         (* empty key buffer)
     GOTO1                       (* unconditionally goto 1)
2    OUT "Enter a string:$N"     (* output string expression to display)
     IN VS1                      (* input register value from key buffer)
     END                         (* end program 1 and exit editor)
```

What will happen: This program, once executed, will prompt the user to enter an integer. After the user enters the number, the program checks to see if both program status register bits 4 and 5 (CE1) are not set. If either one is set, the program prints an error message and asks the user to enter it again. If neither one is set, the program goes to 2, where the user will be prompted to enter a string. Once it is entered, the program ends.

Related Commands: GET, OUT

Registers Used: CE

IN — Inputs Register Value from User Serial Port

Class:	Input/Output Command	
Syntax:	IN *p1* (e.g., IN VI5 IN VS10)	
Parameters:	*allowed values*	*description*
p1	any variable register	variable register
Restrictions:	Allowed only in programs.	
Use:	This command inputs a register value from the user serial port.	

Remarks:

1. If *p1* is a floating point or integer variable, VF*n*, VFVI*n*, VI*n*, VIVI*n*: a.) if the number is greater than 40 characters long, or if it is out of the numerical range of the variable, then bit 5 in the program status register will be set to 1, meaning "String value out of range." A zero will be loaded into the variable; b.) if one or more of the characters are not valid, then bit 4 in the program status register will be set to 1, which means "Invalid digit in string." A zero will be loaded into the variable.
2.) If *p1* is a string variable, VS*n* or VSVI*n*, and the string entered is greater than 127 characters, only the first 127 characters will be loaded. The rest will stay in the user serial port buffer.

Example:

```
    PROGRAM1                  (* edit program 1)
    OUT "Enter an integer:$N" (* output string expression to user serial port)
1   IN VI1                    (* input register value from user serial port)
    IF NOT CE1 GOTO2          (* conditionally goto 2)
    OUT "Invalid number -
        Enter again$N"        (* output string expression to user serial port)
    GOTO1                     (* unconditionally goto 1)
2   OUT "Enter a string:$N"   (* output string expression to user serial port)
    IN VS1                    (* input register value from user serial port)
    END                       (* end program 1 and exit editor)
```

What will happen: This program, once executed, will prompt the user to enter an integer. After the user enters the number, the program checks to see if both program status register bits 4 and 5 (CE1) are not set. If either one is set, the program prints an error message and asks the user to enter it again. If neither one is set, the program goes to 2, where the user will be prompted to enter a string. Once it is entered, the program ends.

Related Commands: GET, OUT

Registers Used: CE

INT Inputs Register Value from Tertiary Port

Class:	Input/Output Command	
Syntax:	INT *p1* (e.g., INT VI5 INT VS10)	
Parameters:	*allowed values*	*description*
p1	any variable register	variable register

Restrictions: Allowed only in programs.

Use: This command inputs a register value from the tertiary port.

Remarks: 1. If *p1* is a floating point or integer variable, VF*n*, VFVI*n*, VI*n*, VIVI*n*: a.) if the number is greater than 40 characters long, or if it is out of the numerical range of the variable, then bit 5 in the program status register will be set to 1, meaning "String value out of range." A zero will be loaded into the variable; b.) if one or more of the characters are not valid, then bit 4 in the program status register will be set to 1, which means "Invalid digit in string." A zero will be loaded into the variable.
2.) If *p1* is a string variable, VS*n* or VSVI*n*, and the string entered is greater than 127 characters, only the first 127 characters will be loaded. The rest will stay in the tertiary port buffer.

Example:

```
     PROGRAM1                      (* edit program 1)
     OUTT "Enter an integer:$N"    (* output string expression to tertiary port)
1    INT VI1                       (* input register value from tertiary port)
     IF NOT CE1 GOTO2              (* conditionally goto 2)
     OUTT "Invalid number -
          Enter again$N"           (* output string expression to tertiary port)
     GOTO1                         (* unconditionally goto 1)
2    OUTT "Enter a string:$N"      (* output string expression to tertiary port)
     INT VS1                       (* input register value from tertiary port)
     END                           (* end program 1 and exit editor)
```

What will happen: This program, once executed, will prompt the user to enter an integer. After the user enters the number, the program checks to see if both program status register bits 4 and 5 (CE1) are not set. If either one is set, the program prints an error message and asks the user to enter it again. If neither one is set, the program goes to 2, where the user will be prompted to enter a string. Once it is entered, the program ends.

Related Commands: GETT, OUTT

Registers Used: CE

INW Inputs Register Value from Key Buffer

Class:	Input/Output Command	
Syntax:	INW *p1* (e.g., INW VI5 INW VS10)	
Parameters:	*allowed values*	*description*
p1	any variable register	variable register
Restrictions:	Allowed only in programs.	

Use: This command inputs a register value from the key buffer. The characters entered are echoed back to the display device until an invalid character or a carriage return is entered. If an invalid character is entered, then the command aborts and the offending character is left in the key buffer.

Remarks: 1. If *p1* is a Boolean, floating point, or integer variable, VB*n*, VBVI*n*, VF*n*, VFVI*n*, VI*n*, VIVI*n*: a.) if the number is greater than 40 characters long, or if it is out of the numerical range of the variable, then bit 5 in the program status register will be set to 1, which means "String value out of range." A zero will be loaded into the variable. b.) if one or more of the characters is not valid, then bit 4 in the program status register will be set to 1, which means "Invalid digit in string." A zero will be loaded into the variable.

2. If *p1* is a string variable, VS*n,* or VSVI*n*, and the string entered is greater than 127 characters, only the first 127 characters will be loaded. The rest will stay in the key buffer.

Example:

```
    PROGRAM1                     (* edit program 1)
    OUTW "Enter an integer:$N"   (* output string expression to display)
1   INW VI1                      (* input register value from key buffer)
    IF NOT CE1 GOTO2             (* conditionally goto 2)
    OUTW "Invalid number -
       Enter again$N"            (* output string expression to display)
    EKB                          (* empty key buffer)
    GOTO1                        (* unconditionally goto 1)
2   OUTW "Enter a string:$N"     (* output string expression to display)
    INW VS1                      (* input register value from key buffer)
    END                          (* end program 1 and exit editor)
```

What will happen: This program, once executed, will prompt the user to enter an integer. After the user enters the number, the program checks to see if both program status register bits 4 and 5 (CE1) are not set. If either one is set, the program prints an error message and asks the user to enter it again. If neither one is set, the program goes to 2, where the user will be prompted to enter a string. Once it is entered, the program ends.

Related Commands: GETW, OUTW

Registers Used: CE

IO General I/O *I jr*

Class:	Input/Output Register	
Type:	Integer, Boolean	
Syntax:	IO*p1* (e.g., IO IO4 IOVI8)	
Parameters:	*allowed values*	*description*
p1	none **or** 0 through 15 **or** VI*n*	I/O register bit number
Range:		
allowed values	0 through $FFFF_{16}$ **or** 0 and 1	
Restrictions:	Read only.	
Use:	The general I/O register is used to identify what inputs and outputs are active.	

Remarks:

1. When the IO? command is executed, the general I/O register will be given as an English statement that says what inputs or outputs, if any, are active. If none of the inputs or outputs are active, the message given is *No I/O is active*.
2. If the computer interface format is enabled, and the IO? command is executed, the general I/O register will be given as an integer number equal to the decimal equivalent of the register's binary value. If none of the inputs or outputs are active, the I/O register is set to 0. The possibilities are listed below.
3. When IOx is executed, the Boolean status of bit 'x' is given.

bit	*message*	*bit*	*message*
0	(IMC) Capture input 2 active (IMJ) Reserved	7	Marker input active
		8	Home input active
1	(IMC) Capture input 2 edge (IMJ) Reserved	9	Forward overtravel input active
		10	Reverse overtravel input active
2	Axis channel A input active	11	Enable input active
3	Axis channel B input active	12	Capture input 1 active
4	Auxiliary channel A input active	13	Capture input 1 edge
5	Auxiliary channel B input active	14	Reserved
6	Auxiliary index input active	15	OK output active

Related Registers: DI, DO

IOA Axis I/O

Class:	Input/Output Register	
Type:	Integer, Boolean	
Syntax:	IOA*p1.p2* (e.g., IOA1 IOAVI1.3 IOA2.VI3 IOAVI1.VI2)	
Parameters:	*allowed values*	*description*
p1	1 through 8 **or** VI*n*	axis number
p2	none **or** 0 through 15 **or** VI*n*	axis I/O register bit number
Range:		
allowed values	0 through $FFFF_{16}$ **or** 0 and 1	
Restrictions:	Read only.	
Use:	The axis I/O register is used to identify what inputs and outputs of an axis are active.	

Remarks:

1. When the IOA*p1*? command is executed, the axis I/O register will be given as an English statement that says what inputs or outputs, if any, are active. If none of the axis inputs or outputs are active, the message given is *No axis I/O is active*.

2. If the computer interface format is enabled, and the IOA*p1*? command is executed, the axis I/O register will be given as an integer number. If none of the axis inputs or outputs are active, the axis I/O register is set to 0. The table below lists the possibilities:

bit	*message*	*bit*	*message*
0	Set point output active	8	Home input active
1	Set point input active	9	Forward overtravel input active
2	Axis channel A input active	10	Reverse overtravel input active
3	Axis channel B input active	11	Enable input active
4	Auxiliary channel A input active	12	Capture input active
5	Auxiliary channel B input active	13	Capture input edge
6	Position feedback lost input active	14	Motor over-temperature input active
7	Marker input active	15	OK output active

Related Registers: DI, DO

IOS System I/O

Class:	Input/Output Register	
Type:	Integer, Boolean	
Syntax:	IOS*p1* (e.g., IOS IOS3 IOSVI4)	
Parameters:	*allowed values*	*description*
p1	none **or** 0 through 15 **or** VI*n*	system I/O register bit number
Range:		
allowed values	0 through $FFFF_{16}$ **or** 0 and 1	
Restrictions:	Read only.	
Use:	The system I/O register is used to identify what inputs and outputs of the system are active.	

Remarks:

1. When the IOS? command is executed, the system I/O register will be given as an English statement.
2. If the computer interface format is enabled, and the IOS? command is executed, the system I/O register will be given as an integer number. The table below lists the possibilities:

bit	*message*
0	Set point output active
1	Set point input active
2	Flash memory card inserted
3	Flash memory card write protected
4	Extended memory card inserted
5	Extended memory card write protected
6	Extended memory card battery low
7	Extended memory card battery dead
8	Teach pendant available
9	Suspend input active
10	Resume input active
11	Enable input active
12	Network power failure input active
13	Reserved
14	Ready output active
15	OK output active

Related Registers: DI, DO

IP Axis in Position

Class:	System Register	
Type:	Boolean	
Syntax:		
I, jr	IP	
⊙	IP*p1* (e.g., IP3 IPVI4)	
Parameters:	*allowed values*	*description*
⊙ *p1*	1 through 8 **or** VIn	axis number
Range:		
allowed values	0, 1	
Restrictions:	Read only.	
Use:	The axis in position register is used to determine whether the axis is in position. If the axis is in position, then IP will be 1; and if the axis is not in position, then IP will be 0. The axis is in position when the position error (PSC-PSA) is less than the value set by the In Position Band (IPB) register. **For continuous moves initiated by the RVF or RVR commands, IP is set true at the end of the acceleration segment.**	
Related Registers:	IPALL, IPB, SRA	

IPALL All Axes in Position

Class:	System Register
Type:	Boolean
Syntax:	IPALL
Range:	
allowed values	0, 1
Restrictions:	Read only.
Use:	The all axes in position register is used to determine whether all of the axes are in position. If all axes are in position, the IPALL will be 1; and if not, IPALL will be 0.
Related Registers:	IP, IPB, SRS

IPB In-Position Band

Class:	Axis Register	
Type:	Floating point	
Syntax:		
I, jr	IPB	
⊙	IPB*p1* (e.g., IPB1 IPBVI3)	
Parameters:	*allowed values*	*description*
⊙ *p1*	1 through 8 **or** VI*n*	axis number
Range:		
units	axis units	
default	0 pulses	
minimum	0 pulses	
maximum	16,000 pulses	

Use: The in-position band register defines the maximum amount of position error (PSC-PSA) that the axis can have and still be in position.

Remarks: The numerical values for the default, minimum, and maximum of this register are assuming that the axis unit ratio, URA, is set at its default value of 1. If the axis unit ratio is set to a value other than 1, the default, minimum, and maximum values must be divided by the value of URA (see URA).

Related Registers: URA, IP

KA Acceleration Feedforward

Class:	Axis Register	
Type:	Integer	
Syntax:		
I, jr	KA	
⊙	KA*p1* (e.g., KA1 KAVI2)	
Parameters:	*allowed values*	*description*
⊙ *p1*	1 through 8 **or** VI*n*	axis number
Range:		
default	0	
minimum	0	
maximum	64,000	
Restrictions:	Servo only.	

Use: The acceleration feedforward constant is used to reduce following error during acceleration or deceleration. The equation for setting KA based on the torque to inertia ratio and the axis feedback resolution, FR, is:

$$KA = \frac{2^{32}\pi}{FR} \times \frac{1}{\left(\frac{torque}{inertia}\right)}$$

This value along with the values of all the other control constants can be set automatically by the AUTOTUNE command.

Related Registers: FR

Related Commands: AUTOTUNE

KD Derivative Control Gain

Class:	Axis Register	
Type:	Integer	
Syntax:		
I, jr	KD	
⊙	KD*p1* (e.g., KD1 KDVI2)	
Parameters:	*allowed values*	*description*
⊙ *p1*	1 through 8 **or** VI*n*	axis number
Range:		
defaults	500 (ampless servo) 200 (2,500 line count encoder servo) 500 (resolver feedback brushless servo)	
minimum	0	
maximum	8,000	
Restrictions:	Servo only.	

Use:

The derivative control gain is used to multiply the time derivative of the following error to control the position of the axis. The equations for setting KD based on the torque to inertia ratio and the axis feedback resolution, FR, are listed below:

For resolver and 2,500 line count encoder models (i.e., IMJ, Target, and standard model IMCs):

$$KD = \frac{316{,}022{,}860}{FR} \times \frac{1}{\sqrt{\frac{torque}{inertia}}}$$

For sinusoidal encoder models (i.e., custom-order IMCs only):

$$KD = \frac{1{,}035{,}461{,}530}{FR} \times \frac{1}{\sqrt{\frac{torque}{inertia}}}$$

This value along with the values of all the other control constants can be set automatically by the AUTOTUNE command.

Related Registers: FR

Related Commands: AUTOTUNE

KEY Character in Key Buffer *I jr*

Class:	System Register
Type:	Boolean
Syntax:	KEY
Range:	
allowed values	0, 1
Restrictions:	Read only.
Use:	This register is used to determine whether a character is in the key buffer. KEY is equal to 1 when there is a character in the key buffer, and it is equal to 0 when there is none. The key buffer can hold up to 256 bytes.
Related Registers:	KYA, SRS
Related Commands:	KY, EKB, GET, IN

KEY Character in User Receive Buffer

Class:	System Register
Type:	Boolean
Syntax:	KEY
Range:	
allowed values	0, 1
Restrictions:	Read only.
Use:	This register is used to determine whether a character is in the user receive buffer. KEY is equal to 1 when there is a character in the user receive buffer, and it is equal to 0 when there is none.
Related Registers:	SRS
Related Commands:	EUB, GET, IN

KEYT Character In Tertiary Receive Buffer

Class:	System Register
Type:	Boolean
Syntax:	KEYT
Range:	
allowed values	0, 1
Restrictions:	Read only.
Use:	This register is used to determine whether a character is in the tertiary receive buffer. KEYT is equal to 1 when there is a character in the buffer, and it is equal to 0 when there is none.
Related Registers:	SRT
Related Commands:	ETB, GETT, INT

KEYW Character in Key Buffer

Class:	System Register
Type:	Boolean
Syntax:	KEYW
Range:	
allowed values	0, 1
Restrictions:	Read only.
Use:	This register is used to determine whether a character is in the key buffer. KEY is equal to 1 when there is a character in the key buffer, and it is equal to 0 when there is none.
Related Registers:	KYA, SRT
Related Commands:	WKY, EKB, GETW, INW

KI Integral Control Gain

Class:	Axis Register	
Type:	Integer	
Syntax:		
I, jr	KI	
⊙	KI*p1* (e.g., KI1 KIVI2)	
Parameters:	*allowed values*	*description*
⊙ *p1*	1 through 8 **or** VI*n*	axis number
Range:		
default	0	
minimum	0	
maximum	64,000	
Restrictions:	Servo only.	

Use: The integral control gain is used to multiply the time integral of the following error to control the position of the axis. The equations for setting KI based on the torque to inertia ratio and the axis feedback resolution, FR, are shown below:

For resolver and 2,500 line count encoder models (i.e., IMJ, Target, and standard model IMCs):

$$KI = \frac{686{,}310}{FR} \times \sqrt{\frac{torque}{inertia}}$$

Torque is the continuous torque of the motor in inch-pounds, and inertia is the system inertia in inch-pounds/sec^2. This value along with the values of all the other control constants can be set automatically by the AUTOTUNE command.

Related Registers: FR

Related Commands: AUTOTUNE

KL Motor Inductance *jr*

Class:	Axis Register
Type:	Integer
Syntax:	KL
Range:	
units	mH
default	10 mH
minimum	1 mH
maximum	100 mH
Restrictions:	Servo only.

Use: The motor inductance is used to tune the digital current controller to the attached motor. This register should be set to the motor's **line-line inductance** in mH—use the following table for your KL values:

Motor Inductance Values — N and S Series Motors

Motor	KL	Motor	KL
3N21-H	4	3S46-G	25
3N22-H	6	3S46-H	6
3N24-G	9	3S63-G	9
3N31-H	10	3S63-H	2
3N32-G	18	3S65-G	14
3N32-H	5	3S65-H	3
3N33-G	25	3S67-G	18
3S22-G	21	3S67-H	5
3S32-G	23	3S84-G	3
3S33-G	22	3S86-G	4
3S33-H	6	3S88-G	4
3S34-G	30	3S8A-G	7
3S35-G	42		
3S43-G	53		
3S43-H	13		
3S45-G	20		
3S45-H	5		

Motor Inductance Values — T Series Motors

Motor	KL	Motor	KL
3T11-G	7	3T53-G	15
3T12-G	4	3T53-H	7
3T13-G	3	3T54-G	16
3T21-G	11	3T54-H	7
3T22-G	7	3T55-G	20
3T23-G	11	3T55-H	9
3T23-H	7	3T55-I	2
3T23-I	3	3T57-G	13
3T24-H	9	3T57-H	3
3T24-I	4	3T65-G	20
3T42-G	26	3T65-H	5
3T42-H	9	3T66-G	24
3T43-G	20	3T66-H	7
3T43-H	13	3T67-G	8
3T43-I	3	3T69-G	10
3T43-J	5		
3T44-G	27		
3T44-H	12		
3T44-I	4		
3T44-J	7		
3T45-G	33		
3T45-H	9		
3T45-I	4		

KLALL Kills All Programs

Class:	Program Command
Syntax:	KLALL
Restrictions:	Not allowed in motion blocks.
Use:	This command kills all programs (i.e., it stops their execution).
Remarks:	1. This command will not stop any motion caused by any previously executed programs. 2. If this command is executed in a program, then the program that executes the command will not be killed
Related Commands:	KLP

KLP Kills Program

Class:	Program Command	
Syntax:	KLP*p1* (e.g., KLP3 KLPVI30)	
Parameters:	*allowed values*	*description*
I, jr *p1*	1 through 4 **or** VI*n*	program number
⊙ *p1*	1 through 17 **or** VI*n*	program number
Restrictions:	Not allowed in motion blocks.	
Use:	This command kills program *p1* (i.e., it stops its execution).	
Remarks:	This command will not stop any motion caused by program *p1*.	
Related Commands:	KLALL	

KM Motor Number *jr*

Class: Axis Register

Type: Integer

Syntax: KM

Range:

Units	none
Default	1
Minimum	1
Maximum	20

Restrictions: Stepper only.

Use: The motor number parameter is used to tune the stepper controller current loop to provide optimum performance for the attached stepper motor. This register must be set to the KM number found on the stepper motor label or selected from the following table. The KM value is used as a pointer by the controller to look-up a number of tuning constants for a given motor. If the value for KM is not recognized by the controller, a set of default tuning constants are used and may not be optimum for the connected motor.

Motor	KM	Wiring*	Max Current
1350x-A	1	Parallel	7.9 Amps
Reserved	2	-	-
1337x-D	3	Series	4.1 Amps
1350x-D	4	Series	4.0 Amps
Reserved	5	-	-
1324x-D	6	Series	2.7 Amps
1221x-D	7	Series	2.0 Amps
1N42xx-A	8	Parallel	6.4 Amps
1N31xx-A	9	Parallel	6.6 Amps
1231x-D	10	Series	2.3 Amps
Reserved	11	-	-
1N32-xxD	12	Series	4.1 Amps
Reserved	13 – 20	-	-

KP Proportional Control Gain

Class:	Axis Register	
Type:	Integer	
Syntax:		
I, jr	KP	
⊙	KP*p1* (e.g., KP1 KPVI2)	
Parameters:	*allowed values*	*description*
⊙ *p1*	1 through 8 **or** VI*n*	axis number
Range:		
default	10	
minimum	0	
maximum	8,000	
Restrictions:	Servo only.	

Use: The proportional control gain is used to multiply the following error to control the position of the axis. The equation for setting KP based on the axis feedback resolution, FR, is:

For resolver and 2,500 line count encoder models (i.e., IMJ, Target, and standard model IMCs):

$$KP = \frac{327{,}680}{FR}$$

This value along with the values of all the other control constants can be set automatically by the AUTOTUNE command.

Related Registers: FR

Related Commands: AUTOTUNE

KT Filter Time Constant

Class:	Axis Register	
Type:	Integer	
Syntax:		
I, jr	KT	
⊙	KT*p1* (e.g., KT1 KTVI2)	
Parameters:	*allowed values*	*description*
⊙ *p1*	1 through 8 **or** VI*n*	axis number
Range:		
jr *default*	3	
I *default*	1	
minimum	0	
maximum	5	
Restrictions:	Servo only.	

Use:

The filter time constant is used to eliminate dither. Generally, the lower the bandwidth of a servo system, the higher the filter time constant should be. The equation for setting KT based on the torque to inertia ratio is:

For IMCs:
$$KT = \left\lfloor \frac{120}{\sqrt{\frac{torque}{inertia}}} + 0.3 \right\rfloor$$

For IMJs:
$$KT = \left\lfloor \frac{280}{\sqrt{\frac{torque}{inertia}}} + 0.5 \right\rfloor$$

where the brackets mean to take the integer part of the number only. Torque is the continuous torque of the motor in in-lbs and inertia is the system inertia in in-lb-sec^2 . This value, along with the values of all the other control constants, can be set automatically by the AUTOTUNE command.

Related Commands: AUTOTUNE

KY Puts One Character into Key Buffer *I jr*

Class:	Input/Output Command	
Syntax:	KY*p1* (e.g., KY1 KYB)	
Parameters:	*allowed values*	*description*
p1	any ASCII character	ASCII character
Restrictions:	Not allowed in motion blocks.	
Use:	This command puts one character into the key buffer.	

Example:

```
KYE        (* put "E" into key buffer)
KY1        (* put "1" into key buffer)
```

Related Commands: GET, IN

KYA Key Assignment

Class:	Input/Output Register	
Syntax:	KYA*p1* (e.g., KYA2)	
Parameters:	*allowed values*	*description*
p1	1 through 12	function key A-L (see table below)

Range:

default	SINGLE
allowed values	OFF (no key codes are put in the key buffer) SINGLE (only key-pressed code is put into key buffer) DOUBLE (key-pressed/key-released codes are put into key buffer)

Restrictions: Not allowed in programs, motion blocks, or expressions.

Use: This register is used to determine what function key codes are put into the key buffer after pressing and releasing function key *p1*.

Related Registers: KEY, KEYW

Function Key	Value
A	1
B	2
C	3
D	4
E	5
F	6
G	7
H	8
I	9
J	10
K	11
L	12

L Makes Last Statement the Current Statement in Line Editor

Class:	Program Command
Syntax:	L
Restrictions:	Allowed only in programs or motion blocks.
Use:	This command makes the last statement the current statement in the line editor.

Examples:

```
IMC/IMJ        Target ARS
PROGRAM1       PROGRAM1       (* edit program 1)
*  PSA=0       *  PSA1=0
X              X              (* step through program)
*  MVL=10      *  MVL1=10
X              X              (* step through program)
*  MAC=40      *  MAC1=40
L              L              (* make last statement the
                              current statement)
*  MVL=10      *  MVL1=10
!              !              (* exit line editor)
*              *
```

Related Commands: PROGRAM, MOTION, X

LABEL Makes Statement at Label the Current Statement

Class:	Program Command	
Syntax:	LABEL*p1* (e.g., LABEL53)	
Parameters:	*allowed values*	*description*
p1	1 through 999	label number
Restrictions:	Allowed only in programs being edited in the terminal window line editor.	
Use:	This command makes the statement at label *p1* the current statement in the terminal window line editor.	

Examples:

```
IMC/IMJ              Target
PROGRAM1             PROGRAM1       (* edit program 1)
* PSA=0              *  PSA1=0
LABEL5               LABEL5    (* make statement at label 5 current statement)
*005OUT “Press       *005OUTW “Press
  any key to           any key to
  stop axis$N”         stop axis$N”
!                    !              (* exit line editor)
*                    *
```

Related Commands: PROGRAM, L, X, !

LED State of Display Led

Class:	Input/Output Register	
Type:	Boolean	
Syntax:	LED*p1* (e.g., LED2 LEDVI5)	
Parameters:	*allowed values*	*description*
p1	1 through 3 **or** VI*n*	LED number
Range:		
default	0 on power-up	
allowed values	0, 1	
Restrictions:	Write only.	
Use:	This register contains the state of one of the display LEDs.	
Remarks:	Please note that this register is write only. It cannot be read.	
Example:		
LED1=1	(* set state of display LED one)	
ASCII Codes:	See the following table	

Code (Hex)	**Description**	**Command**
31	Turn LED1 on	LED1=1, OUT “$1B$31”
32	Turn LED2 on	LED2=1, OUT “$1B$32”
33	Turn LED3 on	LED3=1, OUT “$1B$33”
34	Turn LED 1 off	LED1=0, OUT “$1B$34”
35	Turn LED 2 off	LED2=0, OUT “$1B$35”
36	Turn LED 3 off	LED3=0, OUT “$1B$36”

LOCK Locks Interpreter to Program

Class:	Program Command
Syntax:	LOCK
Restrictions:	Allowed only in programs.
Use:	This command locks the interpreter to the program, which causes other currently executing programs to be to suspended.
Remarks:	Once a program containing the LOCK command is done executing, the interpreter will automatically be unlocked from that program. LOCK will not prevent program 4 from executing when a fault occurs.

Example:

```
PROGRAM1          (* edit program 1)
STM1=0.01         (* load start time of timer 1 and start timer 1)
1 WAIT TM1        (* wait for expression to be true)
LOCK              (* lock interpreter to program)
IF KEY GOTO2      (* conditionally goto 2)
UNLOCK            (* unlock interpreter from program)
GOTO1             (* unconditionally goto 1)
2 END             (* end program and exit editor)
```

What will happen: This program, once executed, will first wait for 10 ms. Then, it locks the interpreter and checks for KEY to be true (i.e., for a character to be entered into the key buffer). If KEY is true, then the program goes to the statement at label 2, which ends the program. If it is not, then it unlocks the interpreter and goes to the statement at label 1, which waits for 10 ms, etc.

Related Commands: UNLOCK

MAC Motion Acceleration/Deceleration

Class:	Motion Register	
Type:	Floating point	
Syntax:		
I, jr	MAC	
🗎	MAC*p1* (e.g., MAC2 MACVI3)	
Parameters:	*allowed values*	*description*
🗎 *p1*	1 through 8 **or** VI*n*	axis number

Range:

units	axis units/sec^2
default	100 pulses/sec^2
minimum	100 pulses/sec^2
maximum	1,000,000,000 pulses/sec^2

Use: This register is used to define both an acceleration and a deceleration rate for the axis. Define the deceleration rate separately with MDC. In cases where the acceleration rate differs from the deceleration rate, you must set MAC first and MDC second. MAC is used only when the motion type, MT, is set to velocity (MT=VEL).

Remarks: The numerical values for the default, minimum, and maximum of this register are assuming that the axis unit ratio, URA, is set at its default value of 1. If the URA is set to a value other than 1, the default, minimum, and maximum values must be divided by the value of URA (see URA).

Examples:

IMC/IMJ	**Target ARS**	
PSA=0	PSA1=0	(* set axis position)
MVL=10	MVL1=10	(* set motion velocity)
MAC=40	MAC1=40	(* set motion acceleration)
MPA=12	MPA1=12	(* set absolute move position)
RPA	RPA1	(* run to absolute position)

What will happen: Setting the axis position, velocity, acceleration, and absolute move position and issuing the RPA command will cause the axis to move 12 units in the forward direction. It will accelerate at 40 units/sec^2 to a velocity of 10 units/sec, and then decelerate at 40 units/sec^2 to zero velocity.

Related Registers: MDC, MAP, MT, URA

MAP Motion Acceleration/Deceleration Percentage

Class:	Motion Register	
Type:	Integer	
Syntax:		
I, jr	MAP	
🖹	MAP*p1* (e.g., MAP2 MAPVI3)	
Parameters:	*allowed values*	*description*
🖹 *p1*	1 through 8 **or** VI*n*	axis number
Range:		
units	%	
default	50	
minimum	1	
maximum	99	

Use:

Time based moves (MT=TIME): This register defines both acceleration and a deceleration percentage for the axis. The deceleration percentage can be defined separately with the MDP command. In cases where the acceleration percentage differs from the deceleration percentage, you must set MAP first and MDP second. The acceleration percentage is the percentage of axis move time that the axis will accelerate. The deceleration percentage is similarly defined (see MDP).

For Compiled Cam Profile Segments (MT=VEL): For compiled cam motion the MAP register defines the percentage of the total segment length over which acceleration/deceleration will take place. MAP also sets the Motion Deceleration Percentage register (MDP) to the same value. When using MDP to specify a deceleration value that is different from the acceleration value you must first set MAP and then set MDP.

Pulse-based moves (MT=PULSE or PULVEL): This register defines the percentage of total auxiliary units (defined by the MPL) over which axis acceleration or deceleration will occur during an incremental or absolute pulse-based move. For example if MAP=20 the acceleration will take 20% of the total move pulses, deceleration will take 20% and the constant velocity segment will take the remaining 60%. MAP is **not** required for continuous pulse-based moves initiated by the RVF and RVR commands. For applications requiring different acceleration and deceleration values the MDP register must be set **after** the MAP register.

Remarks:

1. If MAP is set to a value greater than 50, then MDP is automatically set to the value of MAP subtracted from 100. Otherwise, MDP=MAP.
2. If MAP and MDP are assigned separately, their values cannot be set so that MAP+MDP>100.

Examples:

IMC/IMJ	**Target ARS**	
MPI=5	MPI1=5	(* set incremental move position)
MT=TIME	MT1=TIME	(* set motion type to time)
MTM=10	MTM1=10	(* set move time)
MAP=40	MAP1=40	(* set acceleration percentage)
RPI	RPI1	(* run to incremental move position)

What will happen: The example used above will cause the axis to move 5 units in the forward direction in 10 seconds. It will accelerate 40% of the move time (i.e., 4 seconds), then stay at a constant speed for 20% of move time, then decelerate for the last 40% of move time (i.e., 4 seconds).

Related Registers: MDP, MAC, MT, MTM

MB Motion Block Executing

Class:	System Register	
Type:	Boolean	
Syntax:		
I, jr	MB	
🗎	MB*p1* (e.g., MB3 MBVI4)	
Parameters:	*allowed values*	*description*
🗎 *p1*	1 through 8 **or** VI*n*	axis number
Range:		
allowed values	0, 1	
Restrictions:	Read only.	
Use:	This register is used to determine whether a motion block is executing. If the motion block is executing, then MB is equal to 1; and when it is not executing, then MB is equal to 0.	
Related Registers:	MBANY, SRA	

MBA Assigns Axes to Motion Block

Class:	Program Command	
Syntax:	MBA*p1* (e.g., MBA3 MBA1234 MBAVI2)	
Parameters:	*allowed values*	*description*
p1	1 through 8 **or** list of numbers 1 through 8 **or** VI*n*	axis number
Restrictions:	Allowed only in motion blocks	
Use:	This command assigns the axes that the motion block will use.	
Remarks:	This command must be the first command in a motion block.	

Example:

```
MOTION1        (* edit motion block 1)
MBA13          (* assign axes 1 and 3 to motion block)
MVL1=10        (* set axis one velocity)
MAC1=10        (* set axis one acceleration)
MVL3=5         (* set axis three velocity)
MAC3=10        (* set axis three acceleration)
RVF13          (* run axes 1 and 3 forward)
END            (* end motion block)
```

MBANY Any Motion Block Executing

Class:	System Register
Type:	Boolean
Syntax:	MBANY
Range:	
allowed values	0, 1
Restrictions:	Read only.
Use:	This register is used to determine whether any of the motion blocks are executing. If any of the motion blocks are executing, then MBANY is equal to 1; and if none of the motion blocks are executing, then MBANY is equal to 0.
Related Registers:	MB, SRS

MDA Absolute Move Distance

Class:	Motion Register	
Type:	Floating point	
Syntax:	MDA*p1* (e.g., MDA2 MDAVI3)	
Parameters:	*allowed values*	*description*
p1	1 through 8 **or** VI*n*	axis number

Range:

units	axis units
default	0 pulses
minimum	-2,000,000,000 pulses
maximum	2,000,000,000 pulses

Use: This register is used to define the absolute move distance of the axis for arc segment moves.

Remarks: The numerical values for the default, minimum, and maximum of this register are assuming that the axis unit ratio, URA, is set at its default value of 1. If the axis unit ratio is set to a value other than 1, the default, minimum, and maximum values will change appropriately (see URA).

Example:

```
PSA1=0      (* set axis one position)
PSA2=0      (* set axis two position)
MPA1=0      (* set axis one absolute position)
MPA2=0      (* set axis two absolute position)
MDA1=3      (* set axis one absolute distance)
MDA2=3      (* set axis two absolute distance)
TVL=5       (* set trajectory velocity)
TFP=100     (* set trajectory feedrate to 100 percent)
TFA=500     (* set trajectory feedrate percentage)
RCA12       (* run arc segment with center)
```

What will happen: Setting the axis position, absolute move, absolute distance, trajectory velocity, and trajectory feedrate acceleration and issuing the RCA command will cause axes one and two to move in a circle centered at (3, 3).

Related Registers: MDI, MDO, URA

Related Commands: RCA, RTA

MDC Motion Deceleration

Class:	Motion Register	
Type:	Floating point	
Syntax:		
I, jr	MDC	
🗎	MDC*p1* (e.g., MDC2 MDCVI3)	
Parameters:	*allowed values*	*description*
🗎 *p1*	1 through 8 **or** VI*n*	axis number
Range:		
units	axis units/sec^2	
default	100 pulses/sec^2	
minimum	100 pulses/sec^2	
maximum	1,000,000,000 pulses/sec^2	

Use: This register is used to define a deceleration rate for the axis when the deceleration rate must be different from the acceleration rate. In these cases, you must set MAC first and MDC second. MDC is used **only** when the motion type is set to velocity (MT=VEL). Deceleration for pulse- and time-based moves is set using MDP.

Remarks: The numerical values for the default, minimum, and maximum of this register are assuming that the axis unit ratio, URA, is set at its default value of 1. If the axis unit ratio is set to a value other than 1, the default, minimum, and maximum values must be divided by the value of URA (see URA).

Examples:

IMC/IMJ	**Target ARS**	
PSA=0	PSA1=0	(* set axis position)
MVL=10	MVL1=10	(* set motion velocity)
MAC=40	MAC1=40	(* set motion acceleration)
MDC=10	MDC1=10	(* set motion deceleration)
MPA=12	MPA1=12	(* set absolute move position)
RPA	RPA1	(* run to absolute position)

What will happen: Setting the axis position, velocity, acceleration, and absolute move position and issuing the RPA command will cause the axis to move 12 units in the forward direction. It will accelerate at 40 units/sec^2 to a velocity of 10 units/sec, and then decelerate at 10 units/sec^2 to zero velocity.

Related Registers: MAC, MDP, MT, URA

MDI Incremental Move Distance

Class:	Motion Register	
Type:	Floating point	
Syntax:	MDI*p1* (e.g., MDI2 MDIVI3)	
Parameters:	*allowed values*	*description*
p1	1 through 8 **or** VI*n*	axis number

Range:

units	axis units
default	0 pulses
minimum	-2,000,000,000 pulses
maximum	2,000,000,000 pulses

Use: This register is used to define the incremental move distance of the axis for arc segment moves.

Remarks: The numerical values for the default, minimum, and maximum of this register are assuming that the axis unit ratio, URA, is set at its default value of 1. If the axis unit ratio is set to a value other than 1, the default, minimum, and maximum values will change appropriately (see URA).

Example:

```
MPI1=0      (* set axis one incremental position)
MPI2=0      (* set axis two incremental position)
MDI1=3      (* set axis one incremental distance)
MDI2=3      (* set axis two incremental distance)
TVL=5       (* set trajectory velocity)
TFP=100     (* set trajectory feedrate to 100 percent)
TFA=500     (* set trajectory feedrate percentage)
RCI12       (* run arc segment with center)
```

What will happen: Setting the axis position, absolute move, absolute distance, trajectory velocity, and trajectory feedrate acceleration and issuing the RCI command will cause axes one and two to move in a circle centered at (3, 3) incrementally from their current position.

Related Registers: MDA, MDO, URA

Related Commands: RCI, RTI

MDO Offset Move Distance

Class:	Motion Register	
Type:	Floating point	
Syntax:	MDO*p1* (e.g., MDO1 MDOVI3)	
Parameters:	*allowed values*	*description*
p1	1 through 8 **or** VI*n*	axis number
Range:		
units	axis units	
default	0 pulses	
minimum	-2,000,000,000	
maximum	2,000,000,000	

Use: This register is used to define the offset move distance of the axis for arc segment moves.

Remarks: The numerical values for the default, minimum, and maximum of this register are assuming that the axis unit ratio, URA, is set at its default value of 1. If the axis unit ratio is set to a value other than 1, the default, minimum, and maximum values will change appropriately (see URA).

Example:

```
PSO1=0        (* set axis one position offset)
PSO2=0        (* set axis two position offset)
MPO1=0        (* set axis one offset position)
MPO2=0        (* set axis two offset position)
MDO1=3        (* set axis one offset move distance)
MDO2=3        (* set axis two offset move distance)
TVL=5         (* set trajectory velocity)
TFP=100       (* set trajectory feedrate to 100 percent)
TFA=500       (* set trajectory feedrate percentage)
RCO12         (* run arc segment with center)
```

What will happen: Setting the axis position, absolute move, absolute distance, trajectory velocity, and trajectory feedrate acceleration and issuing the RCO command will cause axes one and two to move in a circle centered at offset position (3, 3).

Related Registers: MDI, MDA, URA

Related Commands: RCO, RTO

MDP Motion Deceleration Percentage

Class: Motion Register

Type: Integer

Syntax:

I, jr	MDP
🖹	MDP*p1* (e.g., MDP2 MDPVI3)

Parameters:

	allowed values	*description*
🖹 *p1*	1 through 8 **or** VI*n*	axis number

Range:

units	%
default	50
minimum	1
maximum	99

Use:

Time based moves (MT=TIME): This register defines a deceleration percentage for the axis. The deceleration percentage is the percentage of axis move time that the axis will decelerate. In cases where the deceleration percentage differs from the acceleration percentage, you must set MAP first and MDP second.

For Compiled Cam Profile Segments (MT=VEL): For compiled cam motion the MDP register defines the percentage of the total segment length over which deceleration will take place. When using MDP to specify a deceleration value that is different from the acceleration value you must first set MAP and then set MDP.

Pulse-based moves (MT=PULSE or PULVEL): This register defines the percentage of total auxiliary units (defined by the MPL register) over which axis deceleration will occur during an incremental or absolute pulse-based move. For example if MDP=20 the deceleration will take 20% of the total MPL units. For applications requiring different acceleration and deceleration values the MDP register must be set **after** the MAP register. MDP is **not** required for continuous pulse-based moves initiated by the RVF and RVR commands.

Remarks:

1. If the deceleration percentage is the same as the acceleration percentage the MDP command is not necessary (MDP=MAP). In this case if MAP is set to a value greater than 50, then MDP is automatically set to the value of MAP subtracted from 100.
2. If MAP and MDP are assigned separately, their values cannot be set so that MAP+MDP>100.

Examples:

IMC/IMJ	**Target**	
MPI=5	MPI1=5	(* set incremental move position)
MT=TIME	MT1=TIME	(* set motion type to time)
MTM=10	MTM1=10	(* set move time)
MAP=25	MAP1=25	(* set acceleration percentage)
MDP=40	MDP1=40	(* set deceleration percentage)
RPI	RPI1	(* run to incremental move position)

What will happen:

Setting the incremental move position, move time, acceleration percentage, and deceleration percentage and issuing the RPI command will cause the axis to move 5 units in the forward direction in 10 seconds. It will accelerate 25% of the move time (i.e., 2.5 seconds), then stay at a constant speed for 35% of move time (i.e., 3.5 seconds), then decelerate for the last 40% of move time (i.e., 4 seconds).

Related Registers: MAP, MDC, MT, MVT

MEMORY Reports Memory Remaining

Class:	System Command
Syntax:	MEMORY
Restrictions:	Not allowed in programs or motion blocks.
Use:	This command reports the remaining memory in bytes.

MFA Motion Feedrate Acceleration/Deceleration

Class: Motion Register

Type: Integer

Syntax:

I, jr	MFA
🖹	MFA*p1* (e.g., MFA2 MFAVI3)

Parameters:

	allowed values	*description*
🖹 *p1*	1 through 8 **or** VI*n*	axis number

Range:

units	percent/second
default	1,000
minimum	1
maximum	200,000

Use: This register is used to define both an acceleration and a deceleration rate for the motion feedrate percentage. Define the deceleration rate separately with MFD. In cases where the acceleration rate differs from the deceleration rate, you must set MFA first and MFD second.

Examples:

```
IMC/IMJ   Target
MFP=40    MFP5=40    (* set motion feedrate percentage)
MFA=500   MFA5=500   (* set motion feedrate acceleration)
MFP=80    MFP5=80    (* set motion feedrate percentage)
```

What will happen: Setting motion feedrate acceleration to 500 and motion feedrate percentage to 80 will cause the controller to accelerate the motion feedrate from 40 percent to 80 percent at 500 percent/second.

Related Registers: MFD, MFP

MFD Motion Feedrate Deceleration

Class:	Motion Register	
Type:	Integer	
Syntax:		
I, jr	MFD	
🖹	MFD*p1* (e.g., MFD2 MFDVI3)	
Parameters:	*allowed values*	*description*
🖹 *p1*	1 through 8 **or** VI*n*	axis number
Range:		
units	percent/second	
default	1,000	
minimum	1	
maximum	200,000	

Use: This register is used to define a deceleration rate for the motion feedrate percentage. In cases where the acceleration rate differs from the deceleration rate, you must set MFA first and MFD second.

Examples:

IMC/IMJ	**Target**	
MFP=80	MFP5=80	(* set motion feedrate percentage)
MFD=500	MFD5=500	(* set motion feedrate deceleration)
MFP=40	MFP5=40	(* set motion feedrate percentage)

What will happen: Setting motion feedrate deceleration to 500 and the motion feedrate percentage to 40 will cause the controller to decelerate the motion feedrate from 80 percent to 40 percent at 500 percent/second.

Related Registers: MFA, MFP

MFP Motion Feedrate Percentage

Class:	Motion Register	
Type:	Floating point	
Syntax:		
I, jr	MFP	
🖹	MFP*p1* (e.g., MFP2 MFPVI3)	
Parameters:	*allowed values*	*description*
🖹 *p1*	1 through 8 **or** VI*n*	axis number

Range:

units	percent
default	100.00
minimum	0.00
maximum	100.00

Use: This register is used to define a feedrate percentage for the axis motion. The feedrate percentage causes the motion to run at a velocity that is a percentage of the motion velocity specified when the motion command was executed.

Remarks: This register is set to its default value on power-up.

Examples:

IMC/IMJ	**Target**	
MVL=20	MVL4=20	(* set motion velocity)
MAC=50	MAC4=50	(* set motion acceleration)
RVF	RVF4	(* run forward at velocity)
MFD=500	MFD4=500	(* set feedrate deceleration)
MFP=63	MFP4=63	(* set feedrate percentage)

What will happen: Setting motion velocity, acceleration, feedrate deceleration, and feedrate percentage and issuing the run forward to velocity command will cause the axis to run forward at 63% of 20 units/second, or 12.6 units/second.

Related Registers: MFA, MFD

Motion Templates: Velocity-based absolute move with feedrate override; time-based, single-axis absolute move with feedrate override

MI Motion Pulse Input

Class:	Motion Register	
Syntax:	MI*p1* (e.g., MI2 MIVI3)	
Parameters:	*allowed values*	*description*
p1	1 through 8 **or** VI*n*	axis number

Range:

default	PSX*a*	
allowed values	PSX*a*	auxiliary input of selected axis (*a*: 1 through 8)
	PSC*a*	command position of selected axis (*a*: 1 through 8)
	PSA*a*	axis position of selected axis (*a*: 1 through 8)

Restrictions: Not allowed in expressions.

Use: This register selects the pulse input source for pulse-based motion. MI is used when motion type, MT, is set to pulse.

Related Registers: MT, MPL, MPS

MJK Motion Jerk Percentage

Class:	Motion Register	
Type:	Integer	
Syntax:		
I, jr	MJK	
🗎	MJK*p1* (e.g., MJK2 MJKVI3)	
Parameters:	*allowed values*	*description*
🗎 *p1*	1 through 8 **or** VI*n*	axis number

Range:

units	%
default	0
minimum	0
maximum	100

Restrictions: MJK has no effect when MT is set to PULSE or PULVEL.

Use: This register is used to define a jerk percentage for the axis. The jerk percentage is the percentage of acceleration/deceleration time that the axis will jerk.

Remarks: If MJK is set to 0, there is no jerk limit (i.e., the jerk is infinite).

Examples:

IMC.IMJ	**Target**	
PSA=0	PSA1=0	(* set axis position)
MVL=5	MVL1=5	(* set motion velocity)
MAC=10	MAC1=10	(* set motion acceleration)
MPI=40	MPI1=40	(* set incremental move position)
MJK=100	MJK1=100	(* set motion jerk percentage)
RPI	RPI1	(* run to incremental move position)
MJK=0	MJK1=0	(* set motion jerk percentage)
RPI	RPI1	(* run to incremental move position)

What will happen: This program will cause the axis to move 40 units in the forward direction. The axis will smoothly ramp the acceleration and deceleration up to 10 units/sec^2 and back down to zero for the whole time it is accelerating and decelerating. Then, setting the jerk percentage to 0 and issuing the RPI command will enable the axis to achieve instantaneously the acceleration rate and deceleration rate during the move.

MONTH Month *I* 🗎

Class:	System Register
Type:	String
Syntax:	MONTH
Range:	
allowed values	January...December
Restrictions:	Read only.
Use:	The month register is used to keep track of the month.

Example:

```
MONTH?          (* report month)
*April
```

Related Registers: TIME, DATE, DAY

MOTION Edits Motion Block

Class:	Program Command	
Syntax:	MOTION*p1* (e.g., MOTION60)	
Parameters:	*allowed values*	*description*
I, *jr* *p1*	1 through 100	motion block number
🖹 *p1*	1 through 400	motion block number

Restrictions: Not allowed in programs or motion blocks.

Use: This command is used to enter the terminal window line editor at the first statement of motion block *p1*. It can be used either to view or edit motion blocks.

Remarks: This command will execute only when all axes have stopped and no programs or motion blocks are running.

Examples:

IMC/IMJ	**Target**	
MOTION1	MOTION1	(* edit motion block 1)
	MBA1	(* assign axis one to motion block)
MVL=10	MVL1=10	(* set motion velocity)
MAC=40	MAC1=40	(* set motion acceleration)
MPI=15	MPI1=15	(* set incremental move position)
RPI	RPI1	(* run to incremental move position)
END	END	(* end motion block 1 and exit editor)

Related Commands: PROGRAM, END, X, !, DEL, L, FAULT

MOTORSET Automatically Sets Up Motor Constants

Class: System Command

Syntax:

I, jr	MOTORSET
🖹	MOTORSET*p1* (e.g., MOTORSET5)

Parameters:

	allowed values	*description*
🖹 *p1*	1 through 8	axis number

Restrictions: Brushless servo only; not allowed in programs or motion blocks.

Use: This command automatically sets up the motor constants, which are CMO and CMR for the IMC; and CMO,CMR, and AR for the Target.

Remarks: This command will execute only when the controller or system and axis are faulted, the axis *Enable* input is true, and no programs or motion blocks are executing. **The motor must not be connected to a load when you use this command.** Executing MOTORSET with a load attached will yield improper values. When executed, it causes the motor rotor to line up with two locations of the stator vector. This command must be executed from the terminal window and takes from two to 30 seconds to execute; when finished, the controller or system will return either an asterisk (*) indicating successful completion or a question mark (?) followed by the appropriate error message. The possible error messages are as follows:

1. SWITCH MOTOR LEADS — two motor leads should be switched.
2. BAD POLES RATIO — the motor poles to resolver poles ratio was less than 1 or greater than 16.
3. BAD RESOLVER AMPLITUDE — the amplitude of the resolver signals could not be properly set.

Related Commands: AUTOTUNE

Registers Used: CMO, CMR, AR, CURC

MPA Absolute Move Position

Class:	Motion Register	
Type:	Floating point	
Syntax:		
I, jr	MPA	
🖹	MPA*p1* (e.g., MPA2 MPAVI3)	
Parameters:	*allowed values*	*description*
🖹 *p1*	1 through 8 **or** VI*n*	axis number
Range:		
units	axis units	
default	0 pulses	
minimum	-2,000,000,000 pulses	
maximum	2,000,000,000 pulses	

Use: For velocity-based, time-based and pulse-based moves this register is used to define the absolute position to which the axis will move. For compiled cam profile segments the MPA register defines the axis absolute position at the end of the profile segment.

Remarks: The numerical values for the default, minimum, and maximum of this register are assuming that the axis unit ratio, URA, is set at its default value of 1. If the axis unit ratio is set to a value other than 1, the default, minimum, and maximum values must be divided by the value of URA (see URA).

Examples:

IMC/IMJ	**Target**	
PSA=0	PSA1=0	(* set axis position)
MVL=10	MVL1=10	(* set motion velocity)
MAC=40	MAC1=40	(* set motion acceleration)
MPA=8	MPA1=8	(* set absolute move position)
RPA	RPA1	(* run to absolute position)

What will happen: Setting the axis position, velocity, acceleration, and absolute move position and issuing the RPA command will cause the axis to move 8 units in the forward direction.

Related Registers: MPI, MPO, URA

Related Commands: RPA, RLA, RCA, RTA

MPI Incremental Move Position

Class: Motion Register

Type: Floating point

Syntax:

I, jr	MPI
🖹	MPI*p1* (e.g., MPI2 MPIVI3)

Parameters:

	allowed values	*description*
🖹 *p1*	1 through 8 **or** VI*n*	axis number

Range:

units	axis units
default	0 pulses
minimum	-2,000,000,000 pulses
maximum	2,000,000,000 pulses

Use: This register is used to define the incremental move position of the axis.

Remarks: The numerical values for the default, minimum, and maximum of this register are assuming that the axis unit ratio, URA, is set at its default value of 1. If the axis unit ratio is set to a value other than 1, the default, minimum, and maximum values must be divided by the value of URA (see URA).

Examples:

IMC/IMJ	**Target**	
MVL=10	MVL1=10	(* set motion velocity)
MAC=40	MAC1=40	(* set motion acceleration)
MPI=12	MPI1=12	(* set incremental move position)
RPI	RPI1	(* run to incremental move position)

What will happen: Setting the velocity, acceleration, and incremental move position and issuing the RPI command will cause the axis to move 12 units in the forward direction.

Related Registers: MPA, MPO, URA

Related Commands: RPI, RLI, RCI, RTI

MPL Move Pulses

Class:	Motion Register
Type:	Floating point
Syntax:	
I, jr	MPL
🗎	MPL*p1* (e.g., MPL2 MPLVI3)
Parameters:	*allowed values* *description*
🗎 *p1*	1 through 8 **or** VI*n* axis number
Range:	
units	the units are the same as the pulse input selection
default	20,000,000 pulses
minimum	1 pulse
maximum	20,000,000 pulses
Restrictions:	For IMCs, this function available only with the extended command set.

Use:

Used only for pulse-based motion (MT=PULSE or PULVEL). This register is **not** used for time-based or velocity-based motion.

For Incremental or Absolute Moves: ***When MT=PULSE:*** this register defines the number of input pulses (or auxiliary position units if URX is not equal to 1) over which the axis makes its motion. When ***MT=PULVEL:*** this register defines the total auxiliary units over which the acceleration and deceleration for the axis motion will occur. The percentage of MPL used for acceleration is defined by MAP (i.e. axis acceleration will occur over MPL*MAP/100 aux. units). The remainder of MPL is then used for deceleration. MVP in this case defines the axis velocity as a ratio of axis units/aux. unit.

For Continuous Moves: The MPL register defines the number of auxiliary position units over which the acceleration or deceleration will occur.

Remarks:

The numerical values for the default, minimum, and maximum of this register assume that the pulse unit ratio is set at 1. If the unit ratio is set to a value other than 1, the default, minimum, and maximum must be divided by the value of URX (see URX).

Examples:

IMC/IMJ	**Target**	
MT=PULSE	MT1=PULSE	(* set motion type to pulse)
	MI1=PSX1	(* set motion pulse input)
PSA=0	PSA1=0	(* set axis position to zero)
PSX=0	PSX1=0	(* set auxiliary position to zero)
MPS=2	MPS1=2	(* set motion start position to 2 aux. units)
MPL=5	MPL1=5	(* set move pulses to 5 aux. units)
MAP=20	MAP1=20	(* set motion acceleration/deceleration percent to 20)
MPA=10	MPA1=10	(* set absolute move position to 10 axis units)
RPA	RPA1	(* run to absolute position)

What will happen:

After you issue the RPA command, the axis will wait until the auxiliary position reaches 2 units; then, while the auxiliary position moves to 7 units, the axis will move to 10 units, using 1 auxiliary unit of motion to accelerate, 3 units to run at a constant velocity, and 1 unit to decelerate to a stop.

Related Registers: MT, MPS, MVP, URX, MI

MPO Offset Move Position

Class:	Motion Register	
Type:	Floating point	
Syntax:		
I, jr	MPO	
🖹	MPO*p1* (e.g., MPO2 MPOVI3)	
Parameters:	*allowed values*	*description*
🖹 *p1*	1 through 8 **or** VI*n*	axis number
Range:		
units	axis units	
default	0 pulses	
minimum	-2,000,000,000 pulses	
maximum	2,000,000,000 pulses	

Use: This register is used to define the destination position for an offset move initiated by the Run to Offset Position (RPO) command. MPO is similar to MPA except that positions are with respect to the PSO register instead of the PSA register.

Remarks: The numerical values for the default, minimum, and maximum of this register are assuming that the axis unit ratio, URA, is set at its default value of 1. If the axis unit ratio is set to a value other than 1, the default, maximum, and minimum values must be divided by the value of URA (see URA).

Examples:

IMC/IMJ	**Target**	
PSO=0	PSO1=0	(* set offset position register)
MVL=10	MVL1=10	(* set motion velocity)
MAC=40	MAC1=40	(* set motion acceleration)
MPO=8	MPO1=8	(* set offset move position)
RPO	RPO1	(* run to offset move position)

What will happen: Setting the offset position register, velocity, acceleration, and offset move position and issuing the RPO command will cause the axis to move 8 units in the forward direction.

Related Registers: MPA, MPI, URA

Related Commands: RPO, RLO, RCO, RTO

MPS Motion Pulse Start Position

Class:	Motion Register	
Type:	Floating point	
Syntax:		
I, *jr*	MPS	
🖹	MPS*p1* (e.g., MPS2 MPSVI3)	
Parameters:	*allowed values*	*description*
🖹 *p1*	1 through 8 **or** VI*n*	axis number
Range:		
units	the units are the same as the pulse input selection	
default	0 pulses	
minimum	-2,000,000,000 pulses	
maximum	2,000,000,000 pulses	

Restrictions: For IMCs, this function available only with the extended command set.

Use: This register is used to define the auxiliary position (PSX) at which the pulse-based axis motion should start. To use the MPS register MT must be set to PULSE or PULVEL. It is not used for velocity-based or time-based motion.

Remarks: The meaning of the MPS register differs slightly for pulse-based incremental or absolute moves and pulse-based continuous moves.

For incremental (RPI) and absolute (RPA) moves: MPS defines the auxiliary position (PSX) where the axis motion will start.

For continuous moves (RVF or RVR): The MPS register is used to define the auxiliary position where either axis acceleration or deceleration will start. Therefore, program segments for continuous moves must use MPS twice. Once to specify where to start the acceleration segment and again to specify where to start the deceleration segment.

The numerical values shown for the default, minimum, and maximum of this register assume that the Auxiliary Unit Ratio (URX) is set to its default value of 1. If URX is set to a value other than 1, the default, maximum, and minimum values must be divided by the value of URX.

Examples:

IMC/IMJ	**Target**	
MT=PULSE	MT1=PULSE	(* set motion type to pulse)
	MI1=PSX1	(* set motion pulse input)
PSA=0	PSA1=0	(* set axis position to zero)
PSX=0	PSX1=0	(* set auxiliary position to zero)
MPS=2	MPS1=2	(* set motion start position to 2 aux. units)
MPL=5	MPL1=5	(* set move pulses to 5 auxiliary units)
MAP=20	MAP1=20	(* set motion acceleration/deceleration percent to 20)
MPA=10	MPA1=10	(* set absolute move position to 10 axis units)
RPA	RPA1	(* run to absolute position)

What will happen: After you issue the RPA command, the axis will wait until the auxiliary position reaches 2 units; then, while the auxiliary position moves to 7 units, the axis will move to 10 units, using 1 auxiliary unit of motion to accelerate, 3 units to run at a constant velocity, and 1 unit to decelerate to a stop.

Related Registers: MT, MPL, MVP, URX, MI

MT Motion Type

Class:	Motion Register	
Syntax:		
I, jr	MT	
🖹	MT*p1* (e.g., MT2 MTVI4)	
Parameters:	*allowed values*	*description*
🖹 *p1*	1 through 8 **or** VI*n*	axis number
Range:		
default	VEL	
allowed values	VEL (velocity) PULSE (pulse input) TIME (time) PULVEL (pulse/velocity)	

Restrictions: Not allowed in expressions; cannot be changed when motion generator is active. For IMCs, the PULSE and PULVEL settings are available only with the extended command set.

Use: The motion type register is used to define the type of commands that will be used to define a motion profile. The motion registers that are used for each of the allowed motion types are:

MT Setting	**Registers that Define Motion Profile**
MT=VEL	MAC, MDC, MJK, and MVL
MT=PULSE	MAP, MDP, MPL, MPS, and MVP
MT=PULVEL	MAP, MPL, MPS, and MVP
MT=TIME	MAP, MDP, MJK , and MTM

Remarks: MT can be changed between PULSE and PULVEL while the axis is in motion. The change will take effect when the next motion command is executed. The PULVEL mode function is the same as the PULSE mode except for incremental or absolute moves the axis velocity is specified by the MVP register as the ratio of axis units/aux. units.

Examples:

IMC/IMJ	**Target**	
MT=VEL	MT1=VEL	(* set motion type to velocity)
MT?	MTVI3?	(* report motion type of axis)

MTE Motor Temperature Input Enable *jr*

Class:	System Register
Type:	Boolean
Syntax:	MTE
Range:	
default	0
allowed values	0, 1
Restrictions:	Encoder feedback servo only.
Use:	The motor temperature input enable parameter defines whether the motor temperature input on the position feedback connector is enabled. If MTE is set to 1, the motor temperature input is enabled; and if MTE is set to 0, then the motor temperature input is disabled.
Related Registers:	FC

MTM Move Time

Class:	Motion Register	
Type:	Floating point	
Syntax:		
I, jr	MTM	
🖹	MTM*p1* (e.g. MTM2 MTMVI3)	
Parameters:	*allowed values*	*description*
🖹 *p1*	1 through 8 **or** VI*n*	axis number

Range:

units	seconds
default	10,000.000
minimum	.005
maximum	10,000.000

Use: The move time register defines the time in which the axis will move. MTM is used when the motion type, MT, is assigned to time.

Examples:

IMC/IMJ	**Target**	
MPI=5	MPI1=5	(* set incremental move position)
MT=TIME	MT1=TIME	(* set motion type to time)
MTM=10	MTM1=10	(* set move time)
MAP=40	MAP1=40	(* set motion acceleration percentage)
RPI	RPI1	(* run to incremental move position)

What will happen: Setting the incremental move position, move time, and acceleration percentage and issuing the RPI command will cause the axis to move 5 units in the forward direction in 10 seconds.

Related Registers: MT

MVL Motion Velocity

Class:	Motion Register	
Type:	Floating point	
Syntax:		
I, jr	MVL	
🖹	MVL*p1* (e.g., MVL2 MVLVI3)	
Parameters:	*allowed values*	*description*
🖹 *p1*	1 through 8 **or** VI*n*	axis number
Range:		
units	axis units/sec	
default	1 pulse/sec	
minimum	1 pulse/sec	
maximum	16,000,000 pulses/sec	

Use: This register is used to define the motion velocity of the axis. MVL is used when the motion type, MT, is assigned to velocity.

Remarks: The numerical values for the default, minimum, and maximum of this register assume that the axis unit ratio, URA, is set at its default value of 1. If URA is set to a value other than 1, the default, maximum, and minimum values will change appropriately (see URA).

Examples:

IMC/IMJ	**Target**	
PSA=0	PSA1=0	(* set axis position)
MVL=10	MVL1=10	(* set motion velocity)
MAC=40	MAC1=40	(* set motion acceleration)
MPA=12	MPA1=12	(* set absolute move position)
RPA	RPA1	(* run to absolute position)

What will happen: Setting the axis position, velocity, acceleration, and absolute move position and issuing the RPA command will cause the axis to move 12 units in the forward direction. It will accelerate at 40 units/sec^2 to a velocity of 10 units/sec, and then decelerate at 40 units/sec^2 to zero velocity.

Related Registers: MT, MAC, URA

MVM Motion Velocity for Run to Marker

Class:	Motion Register	
Type:	Floating point	
Syntax:		
I, jr	MVM	
🗎	MVM*p1* (e.g., MVM2 MVMVI3)	
Parameters:	*allowed values*	*description*
🗎 *p1*	1 through 8 **or** VI*n*	axis number
Range:		
units	axis units/sec	
default	4,096 pulses/sec	
minimum	1 pulse/sec	
maximum	4,096 pulses/sec	

Use: This register is used to define the motion velocity of the axis when one of the run to marker commands, RMF or RMR, is used.

Remarks: The numerical values for the default, minimum, and maximum of this register are assuming that the axis unit ratio, URA, is set at its default value of 1. If the axis unit ratio is set to a value other than 1, the default, maximum, and minimum values must be divided by the value of URA (see URA).

Examples:

IMC/IMJ	**Target**	
PROGRAM1	PROGRAM1	(* edit program one)
MVM=0.5	MVM1=0.5	(* set motion velocity for run to marker)
MAC=40	MAC1=40	(* set motion acceleration)
RMF	RMF1	(* run forward to marker)
WAIT IP	WAIT IP1	(* wait for axis one to be in position)
PSA=0	PSA1=0	(* set axis position)
END	END	(* end program one and exit program editor)

What will happen: This program, once executed, will set the velocity for run to marker and acceleration and then run the axis forward until the marker is encountered. It will then wait for the axis to be in position and set the axis position to 0.

Related Registers: MT, MVL, URA

Related Commands: RMF, RMR

MVP Motion Velocity of Pulse Move

Class:	Motion Register	
Type:	Floating point	
Syntax:		
I, jr	MVP	
🗎	MVP*p1* (e.g., MVP2 MVPVI3)	
Parameters:	*allowed values*	*description*
🗎 *p1*	1 through 8 **or** VI*n*	axis number
Range:		
units	axis units/pulse units	
default	.000001	
minimum	.000001	
maximum	1,000	

Restrictions: For IMCs, this function available only with the extended command set.

Use: This register defines the motion velocity only for pulse-based moves**.**

When MT=PULSE: The MVP register is used only for continuous moves (initiated using the RVF or RVR commands) and is expressed as a ratio of axis units to auxiliary units. For example, if both the axis and the auxiliary encoder are scaled for revolutions then MVP defines the number of revolutions the axis motor will move for each revolution of the auxiliary encoder.

When MT=PULVEL: In this mode the MVP register is used to define the axis velocity for incremental, absolute and continuous moves and is expressed as a ratio of axis units to auxiliary units. The MVP register is not used for velocity-based moves or time-based moves.

MVP cannot be changed for any move already armed (by executing the respective RPI , RPA, RVF or RVR command) or in process.

Examples:

IMC/IMJ	**Target**	
MT=PULSE	MT1=PULSE	(* set motion type to pulse)
	MI1=PSX1	(* set motion pulse input)
PSX=0	PSX1=0	(* set auxiliary position to zero)
MPS=1	MPS1=1	(* set motion start position to 1 aux. unit)
MPL=3	MPL1=3	(* set move pulses to 3 auxiliary units)
MVP=2.5	MVP1=2.5	(* set motion velocity to 2.5 axis units/ auxiliary units)
RVF	RVF1	(* run forward)
WAIT PSX>5.	WAIT PSX1>5.	(* wait for auxiliary position to be > 5 units)
MPS=10	MPS1=10	(* set motion start position to 10 aux. units)
MPL=2	MPL1=2	(* set move pulses to 2 auxiliary units)
ST	ST1	(* stop motion)

What will happen: After you issue the RVF command, the axis will wait until the auxiliary position reaches 1 unit; then, while the auxiliary position moves to 4 units, the axis will accelerate to 2.5 axis units/auxiliary units. After waiting for the auxiliary position to be greater than 5 units, the axis will wait until the auxiliary position reaches 10 units; then, while the auxiliary position moves to 12 units, the axis will decelerate to a stop.

Related Registers: MT, MPS, MPL, MI

OFA Axis Position Offset

Class:	Axis Register	
Type:	Floating point	
Syntax:		
I, jr	OFA	
🗎	OFA*p1* (e.g., OFA1 OFAVI4)	
Parameters:	*allowed values*	*description*
🗎 *p1*	1 through 8 **or** VI*n*	axis number
Range:		
units	axis units	
minimum	-2,000,000,000 pulses	
maximum	2,000,000,000 pulses	

Restrictions: Write only.

Use: This register defines an offset to be applied to the axis position register, PSA. The offset is not stored; rather, the value of the PSA register is changed by the offset.

Remarks: The numerical values for the default, minimum, and maximum of this register are assuming that the axis unit ratio, URA, is set at its default value of 1. If the axis unit ratio is set to a value other than 1, the default, maximum, and minimum values will change appropriately (see URA).

Example:

IMC/IMJ	**Target**	
PSA?	PSA3?	(* query value of axis position register)
*5.326	*-2.36	(* current position)
OFA=4.674	OFA=-1.64	(* offset position register)
PSA?	PSA3?	(* query value of axis position register)
*10	*-4	(* current position)

Related Registers: PSA, URA

OFX Auxiliary Position Offset

Class:	Axis Register	
Type:	Floating point	
Syntax:		
I, jr	OFX	
🗎	OFX*p1* (e.g., OFX1 OFXVI2)	
Parameters:	*allowed values*	*description*
🗎 *p1*	1 through 8 **or** VI*n*	axis number
Range:		
units	auxiliary units	
minimum	-2,000,000,000 pulses	
maximum	2,000,000,000 pulses	
Restrictions:	Write only.	

Use: This register defines an offset to be applied to the auxiliary position register, PSX. The offset is not stored. Rather, the value of the PSX register is changed by the offset. Wrapping of pulse motion is allowed.

Remarks: The numerical values for the default, minimum, and maximum of this register are assuming that the auxiliary unit ratio, URX, is set at its default value of 1. If the auxiliary unit ratio is set to a value other than 1, the default, maximum, and minimum values must be divided by the value of URX (see URX).

Example:

IMC/IMJ	**Target**	
PSX?	PSX3?	(* query value of auxiliary position register)
*5.326	*-2.36	(* current position)
OFX=4.674	OFX3=-1.64	(* offset position register)
PSX?	PSX3?	(* query value of auxiliary position register)
*10	*-4	(* current position)

Related Registers: PSX, URX

OTE Hardware Overtravel Enable *I jr*

Class:		Axis Register
Type:		Boolean
Syntax:		OTE
Range:		
jr	*default*	0
I	*default*	1
	allowed values	0, 1
Restrictions:		Cannot be assigned in motion blocks.

Use:

jr The OTE register is used to enable IMC*jr* hardware overtravel inputs using digital inputs 2 and 3 (IN_01 and IN_02). Input 2 is the forward overtravel input, and input 3 is the reverse overtravel input. Directional conventions are set by the DIR command.

I The OTE register is used to enable IMC hardware overtravel inputs.

Remarks: If the hardware overtravel inputs are disabled (OTE=0), they can be used as general purpose inputs. Use bits 9 and 10 of the IO register to read the state of the hardware overtravel inputs when enabled. Bit 10 of the Axis Status Register (SRA) also reports if either overtravel limit is active but cannot specify which specific limit is active. Controllers also support software travel limits set using the OTF and OTR commands. Generally when travel limits are used in an application the Position Wrap Enable function should be disabled (PWE=0).

Related Registers: IO

OTF Forward Software Overtravel

Class:	Axis Register	
Type:	Floating point	
Syntax:		
I, jr	OTF	
🖹	OTF*p1* (e.g., OTF2 OTFVI3)	
Parameters:	*allowed values*	*description*
🖹 *p1*	1 through 8 **or** VI*n*	axis number
Range:		
units	axis units	
default	2,100,000,000 pulses	
minimum	-2,100,000,000 pulses	
maximum	2,100,000,000 pulses	

Use: This register is used to define the forward software overtravel limit for the axis.

Remarks: The software overtravel limits are ignored during any of the homing functions (RHF, RHR, RMF, RMR, ROF, ROR). The numerical values for the default, minimum, and maximum of this register are assuming that the axis unit ratio, URA, is set at its default value of 1. If the axis unit ratio is set to a value other than 1, the default, maximum, and minimum values be divided by the value of URA (see URA).

Example:

IMC/IMJ	**Target**	
PSA=0	PSA1=0	(* set axis position)
MVL=10	MVL1=10	(* set motion velocity)
MAC=40	MAC1=40	(* set motion acceleration)
MPA=12	MPA1=12	(* set absolute move position)
OTF=10	OTF1=10	(* set forward software overtravel limit)
RPA	RPA1	(* run to absolute move position)

What will happen: By setting the axis position, velocity, acceleration, absolute move position, and forward software overtravel and issuing the RPA command, the axis will move 10 units in the forward direction and immediately halt all motion.

Related Registers: OTR, URA

OTR Reverse Software Overtravel

Class:	Axis Register
Type:	Floating point
Syntax:	
I, jr	OTR
🗎	OTR*p1* (e.g., OTR2 OTRVI3)

Parameters:	*allowed values*	*description*
🗎 *p1*	1 through 8 **or** VI*n*	axis number

Range:

units	axis units
default	-2,100,000,000 pulses
minimum	-2,100,000,000 pulses
maximum	2,100,000,000 pulses

Use: This register is used to define the reverse software overtravel limit for the axis.

Remarks: The software overtravel limits are ignored during any of the homing functions (RHF, RHR, RMF, RMR, ROF, ROR). The numerical values for the default, minimum, and maximum of this register are assuming that the axis unit ratio, URA, is set at its default value of 1. If the axis unit ratio is set to a value other than 1, the default, maximum, and minimum values will change appropriately (see URA).

Example:

IMC/IMJ	**Target**	
PSA=0	PSA1=0	(* set axis position)
MVL=10	MVL1=10	(* set motion velocity)
MAC=40	MAC1=40	(* set motion acceleration)
MPA=-15	MPA1=-15	(* set absolute move position)
OTR=-12	OTR=-12	(* set reverse software overtravel limit)
RPA	RPA1	(* run to absolute move position)

What will happen: Setting the axis position, velocity, acceleration, absolute move position, and reverse software overtravel and issuing the RPA command causes the axis to move 12 units in the reverse direction and immediately halts all motion.

Related Registers: OTF, URA

OUT Outputs String Expression to Serial Port *I jr*

Class: Input/Output Command

Syntax: OUT*p1* (e.g., OUT VS1, OUT "Hello")

Parameters: *allowed values* *description*

p1 any string expression string expression

Use: This command outputs a string expression to the serial port. The string operand "$" can be used to convert register and variable values to strings for use by the OUT command.

Remarks: The operand *p1* can be from 1 to 127 characters long. If the display format is disabled (i.e., DSE is set to 0), the string expression will be sent to the terminal.

Example:

```
VS1="TEST"      (* load string variable)
OUT VS1         (* output string expression to the serial port)
*TEST
```

Related Commands: PUT

Registers Used: DSE

ASCII Codes: See the following table

Code (Hex)	Description	Use	Command
08	backspace	Moves the cursor back one space and prints a space.	BS, OUT "$08"
0A	line feed	Moves the cursor down one line.	OUT "$0A"
0D	carriage return	Moves the cursor to the leftmost space.	OUT "$0D"
31	LED1 on	Turns LED1 on.	LED1=1, OUT "$1B$31"
32	LED2 on	Turns LED2 on.	LED2=1, OUT "$1B$32"
33	LED3 on	Turns LED3 on.	LED3=1, OUT "$1B$33"
34	LED1 off	Turns LED1 off.	LED1=0, OUT "$1B$34"
35	LED2 off	Turns LED2 off.	LED2=0, OUT "$1B$35"
36	LED3 off	Turns LED3 off.	LED3=0, OUT "$1B$36"
3C	alpha off	Disables the function key keypad.	OUT "$1B$3C"
3E	alpha on	Enables the function key keypad.	OUT "$1B$3E"
3F	cursor remember	Remembers the current cursor position.	CRM, OUT "$1B$3F"
40	cursor return	Returns the cursor to the remembered position.	CRR, OUT "$1B$40"
41	cursor up	Moves the cursor up one line.	OUT "$1B$41"
42	cursor down	Moves the cursor down one line.	OUT "$1B$42"
43	cursor right	Moves the cursor right one space.	OUT "$1B$43"
44	cursor left	Moves the cursor left one space.	OUT "$1B$44"
46	cursor position	Places the cursor in a specific position defined by the next two ASCII codes sent. The first is the horizontal position with an offset of 32 (33-62) and the second is the vertical position with an offset of 32 (33-36).	CRP*p1*,*p2*, OUT "$1B$46$*p3*$*p4*" *p3* - 33 through 62 (21 through 3E hex) *p4* - 33 through 36 (21 through 24 hex)
48	cursor home	Homes the cursor, i.e., moves it to the upper left-hand corner of the screen.	CRH, OUT "$1B$48"
49	clear line	Clears the current line and places the cursor at the beginning of the line.	CLL, OUT "$1B$49"
4A	clear display	Clears the display and homes the cursor.	CLS, OUT "$1B$4A"

OUT Outputs String Expression to Serial Port

Class:	Input/Output Command	
Syntax:	OUT*p1* (e.g., OUT VS1, OUT "Hello")	
Parameters:	*allowed values*	*description*
p1	any string expression	string expression

Use: This command outputs a string expression to the user serial port.

Example:

```
VS1="TEST"        (* load string variable)
OUT VS1           (* output string expression to the serial port)
```

Related Commands: PUT

OUTS Outputs Screen to OIP

Class:	Input/Output Command	
Syntax:	OUTS*p1* (e.g., OUTS2 OUTSVI1)	
Parameters:	*allowed values*	*description*
p1	1 through 50 **or** VI*n*	screen number
Use:	This command is used to output screen *p1* to the Operator Interface (OIP).	
Remarks:	This command is used in conjunction with the display when DSE is set to 1.	
Registers Used:	SCRL, DSE	

OUTT Outputs String Expression to Tertiary Port

Class:	Input/Output Command	
Syntax:	OUTT *p1* (e.g., OUTT VS1 OUTT “Hello”)	
Parameters:	*allowed values*	*description*
p1	any variable register	variable register
Use:	This command outputs a string expression to the tertiary port.	

Example:

```
VS1=“TEST”        (* load string variable)
OUTT VS1          (* output string expression to the tertiary port)
```

Related Commands: PUTT

OUTW Outputs String Expression to OIP

Class: Input/Output Command

Syntax: OUTW *p1* (e.g., OUTW VS1 OUTW "Hello")

Parameters:	*allowed values*	*description*
p1	any variable register	variable register

Use: This command outputs a string expression to the display.

Example:

```
VS1="TEST"          (* load string variable)
OUTW VS1            (* output string expression to the display)
```

Related Commands: PUTW

ASCII Codes: See the following table

Code (Hex)	Description	Use	Command
08	backspace	Moves the cursor back one space and prints a space.	BS, OUT "$08"
0A	line feed	Moves the cursor down one line.	OUT "$0A"
0D	carriage return	Moves the cursor to the leftmost space.	OUT "$0D"
31	LED1 on	Turns LED1 on.	LED1=1, OUT "$1B$31"
32	LED2 on	Turns LED2 on.	LED2=1, OUT "$1B$32"
33	LED3 on	Turns LED3 on.	LED3=1, OUT "$1B$33"
34	LED1 off	Turns LED1 off.	LED1=0, OUT "$1B$34"
35	LED2 off	Turns LED2 off.	LED2=0, OUT "$1B$35"
36	LED3 off	Turns LED3 off.	LED3=0, OUT "$1B$36"
3C	alpha off	Disables the function key keypad.	OUT "$1B$3C"
3E	alpha on	Enables the function key keypad.	OUT "$1B$3E"
3F	cursor remember	Remembers the current cursor position.	CRM, OUT "$1B$3F"
40	cursor return	Returns the cursor to the remembered position.	CRR, OUT "$1B$40"
41	cursor up	Moves the cursor up one line.	OUT "$1B$41"
42	cursor down	Moves the cursor down one line.	OUT "$1B$42"
43	cursor right	Moves the cursor right one space.	OUT "$1B$43"
44	cursor left	Moves the cursor left one space.	OUT "$1B$44"
46	cursor position	Places the cursor in a specific position defined by the next two ASCII codes sent. The first is the horizontal position with an offset of 32 (33-62) and the second is the vertical position with an offset of 32 (33-36).	CRP*p1*.*p2*, OUT "$1B$46$*p3*$*p4*" *p3* - 33 through 62 (21 through 3E hex) *p4* - 33 through 36 (21 through 24 hex)
48	cursor home	Homes the cursor, i.e., moves it to the upper left-hand corner of the screen.	CRH, OUT "$1B$48"
49	clear line	Clears the current line and places the cursor at the beginning of the line.	CLL, OUT "$1B$49"
4A	clear display	Clears the display and homes the cursor.	CLS, OUT "$1B$4A"

PAR Parity of Serial Port *I jr*

Class:	System Register
Syntax:	PAR
Range:	
default	ODD
allowed values	NONE, EVEN, ODD
Restrictions:	Not allowed in motion blocks or expressions.
Use:	This register is used to define the parity of the serial port.
Remarks:	Setting PAR to NONE and BIT to 7 at the same time is not allowed. This register defaults to ODD on power-up.
Related Registers:	BAUD, BIT, HSE

PARP Parity of Program Port

Class:	System Register
Syntax:	PARP
Range:	
default	automatically set to even or odd
allowed values	NONE, EVEN, ODD
Restrictions:	Not allowed in motion blocks or expressions.
Use:	This register is used to define the parity of the program port.
Remarks:	Setting PARP to NONE and BITP to 7 at the same time is not allowed.
Related Registers:	BITU, PARU, BAUDU, BAUDP, BITP

PARU Parity of User Serial Port

Class:	System Register
Syntax:	PARU
Range:	
default	ODD
allowed values	NONE, EVEN, ODD
Restrictions:	Not allowed in motion blocks or expressions.
Use:	This register is used to define the parity of the user serial port.
Remarks:	Setting PARU to NONE and BITU to 7 at the same time is not allowed.
Related Registers:	PARP, BITU, BAUDU, BAUDP, BITP

PASSWORD Prompts for Password

Class:	System Command
Syntax:	PASSWORD
Restrictions:	Not allowed in programs or motion blocks.
Use:	This command prompts the user to enter a password that was previously defined using the CHANGEPW command.
Remarks:	Enter the 4 to 10 character password at the *Enter password:* prompt to gain full access to the controller programming and configuration. If the correct password is not entered at the prompt, only diagnostic commands can be entered. To assign an initial password or to change an existing password use the CHANGEPW command.
	Warning! Do NOT forget your password. Clearing memory will not reset the password. You must return the unit to the factory for repair. THERE IS NO BACKDOOR! Consider using the SECURE command instead.
Related Commands:	CHANGEPW, SECURE

PCA Axis Position Capture

Class:	Axis Register	
Type:	Floating point	
Syntax:		
I, jr	PCA	
🖹	PCA*p1* (e.g., PCA3 PCAVI4)	
Parameters:	*allowed values*	*description*
🖹 *p1*	1 through 8 **or** VI*n*	axis number
Range:		
units	axis units	
minimum	-2,000,000,000 pulses	
maximum	2,000,000,000 pulses	

Restrictions: Read only.

Use: This register is used to store the value of the position captured by the position capture input when this input is used to capture the axis position.

Remarks: 1. If a position has not been captured, then the axis position capture register will be 0. Bit 13 of the I/O register (IO/IOA) will be set to 1 when a position has been captured. After a position has been captured, the position can be reported using the PCA? command. The register will then be set to 0, and bit 13 will be cleared until a position is captured again.
2. The numerical values for the minimum and maximum of this register are assuming that the axis unit ratio, URA, is set at its default value of 1. If the axis unit ratio is set to a value other than 1, the maximum and minimum values must be divided by the value of URA (see URA).

Related Registers: URA, PCX, IO, IOA

PCX Auxiliary Position Capture

Class:	Axis Register	
Type:	Floating point	
Syntax:		
I, jr	PCX	
🗎	PCX*p1* (e.g., PCX3 PCXVI4)	
Parameters:	*allowed values*	*description*
🗎 *p1*	1 through 8 **or** VI*n*	axis number
Range:		
units	auxiliary units	
minimum	-2,000,000,000 pulses	
maximum	2,000,000,000 pulses	

Restrictions: For IMCs, this function available only with the extended command set; read only.

Use: This register is used to store the value of the position captured when the position capture is used to capture the auxiliary encoder input of the axis.

Remarks:

1. If a position has not been captured, then the auxiliary position capture register will be 0. Bit 13 of the I/O register (IO/IOA) will be set to 1 when a position has been captured. After a position has been captured, the position can be reported using the PCX? command. The register will then be set to 0, and bit 13 will be cleared until a position is captured again.
2. To ensure proper operation of the edge trigger, always read PCA as well as PCX when using PCX.
3. The numerical values for the minimum and maximum of this register are assuming that the auxiliary unit ratio, URX, is set at its default value of 1. If the auxiliary unit ratio is set to a value other than 1, the maximum and minimum values must be divided by the value of URX (see URX).

Related Registers: URX, PCA, IO, IOA

PCX2 Auxiliary Position Capture Two *I*

Class:	Axis Register
Type:	Floating point
Syntax:	PCX2
Range:	
units	auxiliary units
minimum	-2,000,000,000 pulses
maximum	2,000,000,000 pulses
Restrictions:	For IMCs, this function available only with the extended command set; read only.
Use:	This register is used to store the value of the position captured when position capture input two captures the auxiliary encoder input of the axis.
Remarks:	1. If a position has not been captured, then the auxiliary position capture two register will be 0. Bit 1 of the I/O register (IO) will be set to 1 when a position has been captured. After a position has been captured, the position can be reported using the PCX2? command. The register will then be set to 0, and bit 1 will be cleared until a position is captured again. 2. The numerical values for the minimum and maximum of this register are assuming that the auxiliary unit ratio, URX, is set at its default value of 1. If the auxiliary unit ratio is set to a value other than 1, the maximum and minimum values will change appropriately (see URX).
Related Registers:	URX, PCX, IO

PDV Pulse Divisor *I*

Class:	Input/Output Register
Type:	Floating point
Syntax:	PDV
Range:	
units	auxiliary units
default	0
minimum	0 pulses
maximum	30,000 pulses
Restrictions:	For IMCs, this function available only with the extended command set.
Use:	This register is used to provide a pulse output on I/O 10. When this register is set to zero, I/O 10 has normal function. When PDV is non-zero, I/O 10 will change state for every PDV pulse of the auxiliary input.
Related Registers:	URX

PFB Position Feedback Deadband

Class:	Axis Register	
Type:	Floating point	
Syntax:		
I, jr	PFB	
🖹	PFB*p1* (e.g., PFB2 PFBVI3)	
Parameters:	*allowed values*	*description*
🖹 *p1*	1 through 8 **or** VI*n*	axis number
Range:		
units	axis units	
default	0 pulses (dual loop feedback servo) or 10 pulses (stepper)	
minimum	0 pulses	
maximum	16,000 pulses	
Restrictions:	Stepper or dual loop feedback servo only.	
Use:	The Position Feedback Deadband is the amount of static position error allowed before the controller attempts to correct the position error when the controller is configured for dual-loop mode (see PFE for more on axis position control modes.)	
Remarks:	The numerical values for the default, minimum, and maximum of this register are correct for an axis unit ratio, URA, of 1. If the axis unit ratio is set to a value other than 1, the default, maximum, and minimum values must be divided by the value of URA (see URA).	
Related Registers:	URA, PFL, PFT, PFE, PFN, PFD, PFC	

PFC Position Feedback Correction Numerator

Class:	Axis Register	
Type:	Integer	
Syntax:		
I, jr	PFC	
🗎	PFC*p1* (e.g., PFC2 PFCVI3)	
Parameters:	*allowed values*	*description*
🗎 *p1*	1 through 8 **or** VI*n*	axis number
Range:		
default	PFN	
minimum	0	
maximum	10,000	

Restrictions: Stepper or dual loop feedback servo only.

Use: The position feedback correction numerator is a parameter used when auxiliary encoder position feedback is used to control the position of a stepper servo (i.e. closed loop stepper) or when a servo controller is configured for dual-loop mode. PFC replaces the numerator of the feedback ratio PFN/PFD and is used to fine-tune this feedback ratio to eliminate hunting as the controller attempts to correct of the final position error (see PFE for more on axis position control modes.)

Remarks: Normally this parameter is left at the default of PFN, which means it has the same value as PFN. If there are problems with hunting for the final position, use this parameter to reduce the correction by setting it to a value less than PFN.

Related Registers: PFD, PFE, PFN

PFD Position Feedback Denominator

Class:	Axis Register	
Type:	Integer	
Syntax:		
I, jr	PFD	
🗎	PFD*p1* (e.g., PFD2 PFDVI3)	
Parameters:	*allowed values*	*description*
🗎 *p1*	1 through 8 **or** VI*n*	axis number
Range:		
default	1 (dual loop feedback servo) or 4 (stepper)	
minimum	1	
maximum	10,000	
Restrictions:	Closed-loop stepper or dual loop feedback servo only.	

Use:

The position feedback denominator is a parameter used when auxiliary encoder position feedback is used to control the position of a stepper servo (i.e. closed loop stepper) or when a servo controller is configured for dual-loop mode. PFD is defined as the denominator of the position feedback ratio (PFN/PFD) between the motor position feedback and the auxiliary encoder inputs. This ratio must equate the number of motor position feedback pulses to auxiliary encoder pulses per unit of load movement. This determination must include all gearing and mechanical translation in **both** the auxiliary encoder **and** motor connection to the load. For example, consider a servo application where a 1000 line auxiliary encoder is belted to the load end of a ball screw using a 2:1 ratio with the motor mounted to the opposite end of the screw through a 2:1 gearbox. For each screw revolution the auxiliary encoder makes 2 revolutions and generates 8,000 quadrature pulses to the controller (2 rev * 4000 pulses/rev). For the same 1 revolution of the screw the motor makes 2 revolutions and generates 8,192 quadrature pulses (2 rev * 10,000 pulses/rev). Therefore, the PFN/PFD ratio must be equivalent to 8,192/8000 and be within the allowable range.

Stepper Controller (PFN= non-zero & PFE=1):
For a stepper controller using encoder feedback the Position Feedback Ratio (PFN/PFD) is used to map the auxiliary encoder feedback to the 50,000 steps/revolution of the motor. This is done by setting the ratio equal to the number of motor pulses/rev (50,000) divided by the number of auxiliary encoder pulses generated during 1 motor revolution. In the simplest case where the encoder is mounted to the stepper motor the denominator would be the quadrature resolution of the auxiliary encoder. For example using a 1000 line encoder (4000 quad pulses) the ratio is 50000/4000. Since the PFN and PFD registers are limited to a range of 10,000 we can reduce this ratio to 50/4 or PFN=50 and PFD=4 which are the default register values. If the feedback encoder is mounted at the load this ratio must include all gearing and mechanical translation in **both** the auxiliary encoder **and** motor connection to the load (see example for dual-loop servo above except use 50,000 pulses/rev for the motor instead of 4,096/rev).

Related Registers: PFN, PFE

PFE Position Feedback Enable

Class:	Axis Register	
Type:	Boolean	
Syntax:		
I, jr	PFE	
🗎	PFE*p1* (e.g., PFE3 PFEVI4)	
Parameters:	*allowed values*	*description*
🗎 *p1*	1 through 8 **or** VI*n*	axis number
Range:		
default	0	
allowed values	0, 1	

Restrictions: Stepper or dual loop feedback servo only; not allowed in motion blocks. This register can be set only when the controller is faulted.

Use: The position feedback enable register is used to determine whether the axis receives position feedback from the motor position feedback or from the auxiliary encoder.

Servo Controller (PFE=0):
If PFE is set to 0, then the axis uses the motor position feedback. This is the controller's normal operating mode.

Servo Controller (PFE = 1 and PFN=0):
In this single-loop mode the auxiliary encoder is used for axis position feedback and directly updates the axis position register (PSA). The motor position feedback is still used for commutation.

Servo Controller (PFE = 1 and PFN=non-zero):
In this dual-loop mode the motor position feedback is the primary axis position feedback device and the auxiliary encoder is the secondary feedback device. The motor position feedback is used for axis position feedback while normal programmed motion is being executed while the secondary feedback is used to ensure accurate static position based on the auxiliary encoder feedback. This dual-loop mode offers the best servo stability when using a separate (load mounted) position feedback device in applications where there is lost motion in the motor drive train. The Position Feedback ratio (PFN/PFD) must be properly set when using this mode. Also the PFB, PFC, PFL and PFT registers are enabled in this mode.

Open Loop Stepper (PFE=0):
If PFE is set to 0, then the stepper controller runs open loop. This is the controller's default operating mode.

Closed Loop Stepper (PFE=1):
If PFE is set to 1, then the stepper controller uses the auxiliary encoder feedback to close the position loop. The Position Feedback Ratio (PFN/PFD) must be properly configured to map the auxiliary encoder feedback to the 50,000 steps/revolution of the stepper motor.

Registers Used: PFN, PFD, PFL, PFT, PFB, PFC

PFL Position Feedback Backlash

Class:	Axis Register	
Type:	Integer	
Syntax:		
I, jr	PFL	
🗎	PFL*p1* (e.g., PFL3 PFLVI4)	
Parameters:	*allowed values*	*description*
🗎 *p1*	1 through 8 **or** VI*n*	axis number
Range:		
units	pulses	
default	0	
minimum	0	
maximum	16,000	
Restrictions:	Stepper or dual loop feedback servo only.	
Use:	The position feedback backlash is used to compensate for mechanical backlash when using auxiliary encoder position feedback. Enable auxiliary encoder position feedback by setting PFE equal to 1. When configured for dual-loop servo (PFR=1 and PFN=non-zero), the PFL value is used to offset an equivalent number of pulses lost due to mechanical backlash or other sources of lost motion in the motor drive train when axis direction is reversed.	
Related Registers:	PFT, PFB, PFE	

PFN Position Feedback Numerator

Class:	Axis Register	
Type:	Integer	
Syntax:		
I, jr	PFN	
🖹	PFN*p1* (e.g., PFN2 PFNVI3)	
Parameters:	*allowed values*	*description*
🖹 *p1*	1 through 8 **or** VI*n*	axis number
Range:		
default	0 (dual loop feedback servo) or 50 (stepper)	
minimum	0	
maximum	10,000	
Restrictions:	Stepper or dual loop feedback servo only.	
Use:	The position feedback correction numerator is a parameter used in encoder position feedback. It is the numerator of the position feedback ratio between the axis and the encoder input.	
Related Registers:	PFD, PFE, PFC	

PFT Position Feedback Correction Time

Class:	Axis Register	
Type:	Floating point	
Syntax:		
I, jr	PFT	
🖹	PFT*p1* (e.g., PFT2 PFTVI3)	
Parameters:	*allowed values*	*description*
🖹 *p1*	1 through 8 **or** VI*n*	axis number
Range:		
units	seconds	
default	.010	
minimum	.001	
maximum	4.000	
Restrictions:	Stepper or dual loop feedback servo only.	
Use:	The position feedback correction time is the time that the system waits between position corrections when using position feedback. The position feedback is enabled by setting PFE equal to 1.	
Related Registers:	PFL, PFB, PFE	

PHB Phase Error Bound

Class:	Motion Register	
Type:	Integer	
Syntax:		
I, jr	PHB	
🗎	PHB*p1* (e.g., PHB1 PHBVI4)	
Parameters:	*allowed values*	*description*
🗎 *p1*	1 through 8 **or** VI*n*	axis number
Range:		
units	pulses	
default	32,000	
minimum	0	
maximum	32,000	

Restrictions: For IMCs, this function available only with the extended command set.

Use: The phase error bound register is used to define a bound on the phase error of the phase-locked loop. If this limit is exceeded, the phase error is set to half of the phase error bound, and bit five of the axis status register, SRA, is set to 1. This corresponds to the axis status message *Phase error past bound*.

Related Registers: PHR, PHE

PHE Phase-Locked Loop Enable

Class:	Motion Register	
Type:	Boolean	
Syntax:		
I, jr	PHE	
🖹	PHE*p1* (e.g., PHE1 PHE245 PHEVI4)	
Parameters:	*allowed values*	*description*
🖹 *p1*	1 through 8 **or** list of numbers 1 through 8 **or** VI*n*	axis number
Range:		
default	0	
allowed values	0, 1	
Restrictions:	For IMCs, this function available only with the extended command set.	
Use:	This register is used to determine whether the phase-locked loop is enabled. If PHE is set to 1, then the phase-locked loop is enabled; and if PHE is set to 0, it is disabled.	
Registers Used:	PHB, PHG, PHL, PHM, PHO, PHP, PHR, PHT, PHZ	
Motion Templates:		
I, *jr*, 🖹	Single-axis, phase-locked loop	
🖹	Multi-axis phase-locked loop	

PHG Phase Gain

Class:	Motion Register	
Type:	Integer	
Syntax:		
I, jr	PHG	
🗎	PHG*p1* (e.g., PHG1 PHGVI4)	
Parameters:	*allowed values*	*description*
🗎 *p1*	1 through 8 **or** VI*n*	axis number
Range:		
default	0	
minimum	0	
maximum	255	
Restrictions:	For IMCs, this function available only with the extended command set.	
Use:	The phase gain is used to multiply the phase error, PHR, to adjust the value of the phase multiplier, PHM.	
Related Registers:	PHR, PHM, PHE	

PHL Phase Length

Class:	Motion Register	
Type:	Integer	
Syntax:		
I, jr	PHL	
🖹	PHL*p1* (e.g., PHL1 PHLVI4)	
Parameters:	*allowed values*	*description*
🖹 *p1*	1 through 8 **or** VI*n*	axis number
Range:		
units	pulses	
default	1,000	
minimum	500	
maximum	64,000	
Restrictions:	For IMCs, this function available only with the extended command set.	
Use:	The phase length register is used to define the number of pulses during one cycle of the reference input.	
Related Registers:	PHP	

PHM Phase Multiplier

Class:	Motion Register	
Type:	Floating point	
Syntax:		
I, jr	PHM	
🗎	PHM*p1* (e.g., PHL1 PHLVI4)	
Parameters:	*allowed values*	*description*
🗎 *p1*	1 through 8 **or** VI*n*	axis number
Range:		
minimum	0.0001	
maximum	10,000.0000	
Restrictions:	For IMCs, this function available only with the extended command set; read only.	
Use:	The phase multiplier is the ratio between the axis and the reference input when using the phase-locked loop.	
Related Registers:	PHE	

PHO Phase Offset

Class:	Motion Register	
Type:	Integer	
Syntax:		
I, jr	PHO	
🖹	PHO*p1* (e.g., PHO1 PHOVI4)	
Parameters:	*allowed values*	*description*
🖹 *p1*	1 through 8 **or** VI*n*	axis number
Range:		
units	pulses	
default	0	
minimum	-32,000	
maximum	32,000	
Restrictions:	For IMCs, this function available only with the extended command set.	
Use:	The phase offset register is used to define an offset on the reference position, PHP, of the phase-locked loop.	
Related Registers:	PHP	

PHP Phase Position

Class:	Motion Register	
Type:	Integer	
Syntax:		
I, jr	PHP	
🖹	PHP*p1* (e.g., PHP1 PHPVI4)	
Parameters:	*allowed values*	*description*
🖹 *p1*	1 through 8 **or** VI*n*	axis number
Range:		
units	pulses	
default	0	
minimum	-PHL/2	
maximum	PHL/2 - 1	
Restrictions:	For IMCs, this function available only with the extended command set.	
Use:	The phase position register is used to define the reference position of the phase-locked loop.	
Related Registers:	PHL, PHO, PHE	

PHR Phase Error

Class:	Motion Register	
Type:	Integer	
Syntax:		
I, jr	PHR	
🗎	PHR*p1* (e.g., PHR1 PHRVI4)	
Parameters:	*allowed values*	*description*
🗎 *p1*	1 through 8 **or** VI*n*	axis number
Range:		
units	pulses	
minimum	-32,000	
maximum	32,000	
Restrictions:	For IMCs, this function available only with the extended command set; read only.	
Use:	The phase error is the difference between the desired reference position and the reference position that was captured when the position capture input became active. It can be used, along with PHG and PHZ, to make corrections in the phase position.	
Related Registers:	PHG, PHZ, PHE	

PHT Phase Lockout Time

Class:	Motion Register	
Type:	Floating point	
Syntax:		
I, jr	PHT	
🗎	PHT*p1* (e.g., PHT1 PHTVI4)	
Parameters:	*allowed values*	*description*
🗎 *p1*	1 through 8 **or** VI*n*	axis number
Range:		
units	seconds	
default	0.05	
minimum	.001	
maximum	4.000	
Restrictions:	For IMCs, this function available only with the extended command set.	
Use:	The phase lockout time is the time interval, after the position capture, in which the position capture input is disabled. This time interval is used to account for any undesired position capture inputs.	
Related Registers:	PHE	

PHZ Phase Zero

Class:	Motion Register	
Type:	Integer	
Syntax:		
I, jr	PHZ	
🖹	PHZ*p1* (e.g., PHZ1 PHZVI4)	
Parameters:	*allowed values*	*description*
🖹 *p1*	1 through 8 **or** VI*n*	axis number
Range:		
default	245	
minimum	0	
maximum	255	
Restrictions:	For IMCs, this function available only with the extended command set.	
Use:	The phase zero register is used to define the zero of the compensator of the phase-locked loop. This, in conjunction with PHG, defines a method of correction of the phase in the phase-locked loop.	
Related Registers:	PHG, PHE	

PLA Axis Position Length

Class:	Axis Register	
Type:	Floating point	
Syntax:		
I, jr	PLA	
🗎	PLA*p1* (e.g., PLA1 PLAVI4)	
Parameters:	*allowed values*	*description*
🗎 *p1*	1 through 8 **or** VI*n*	axis number
Range:		
units	axis units	
default	2,000,000,000 pulses	
minimum	500 pulses	
maximum	2,000,000,000 pulses	

Restrictions: Not allowed in programs or motion blocks.

Use: This register is used to define the axis position length. This is actually half the axis position register length. The axis position register, PSA, will count from -PLA units to PLA-(1/URA) units if position register wrap, PWE, is enabled. PLA has no effect on the axis position register if PWE is disabled.

🗎 For the Target ARS, when CAT=PSR*p1*, PLA*p1* defines the cam shaft input length. The cam shaft input counts from -180 to 180 as PSA*p1* counts from -PLA*p1* to PLA*p1*-1.

Remarks: The numerical values for the default, minimum, and maximum of this register are assuming that the axis unit ratio, URA, is set at its default value of 1. If the axis unit ratio is set to a value other than 1, the default, maximum, and minimum values must be divided by the value of URA (see URA).

Related Registers: PWE, URA

PLX Auxiliary Position Length

Class:	Axis Register	
Type:	Floating point	
Syntax:		
I, jr	PLX	
🗎	PLX*p1* (e.g., PLX1 PLXVI4)	
Parameters:	*allowed values*	*description*
🗎 *p1*	1 through 8 **or** VI*n*	axis number
Range:		
units	auxiliary units	
default	2,000,000,000 pulses	
minimum	500 pulses	
maximum	2,000,000,000 pulses	

Restrictions: For IMCs, this function available only with the extended command set; not allowed in programs or motion blocks.

Use: This register is used to define the auxiliary position range. This is actually half the auxiliary position register length. The auxiliary position register, PSX, counts from -PLX units to PLX-(1/URX) units.

When Electronic Cam is Enabled:
When the electronic cam function is enabled (CAE=1) the auxiliary position register range defined above represents the cam master position range required to complete one cycle of the cam table. For example, assuming we have a 1000 line (4000 pulse/rev) auxiliary encoder, the axis and auxiliary units are both in revolutions (URA=10000; URX=4000) and PLX=0.5, then the PSX register will count from –0.5 to 0.49975 encoder revolutions to complete one cam cycle.

Remarks: The *position wrap enable* register (PWE) has no effect on the auxiliary position register rollover. The PSX register automatically rolls over at the limits defined above for PLX. Make sure the PSX register is initialized to a value that falls with this range. The numerical values for the default, minimum, and maximum of this register are assuming that the *auxiliary unit ratio*, URX, is set at its default value of 1. If the auxiliary unit ratio is set to a value other than 1, the default, maximum, and minimum values must be divided by the value for URX (see URX).

Related Registers: URX, PSX

PLY Playback Recorded Positions Enable

Class:	Motion Register	
Type:	Boolean	
Syntax:	PLY*p1* (e.g., PLY1 PLY245 PLYVI4)	
Parameters:	*allowed values*	*description*
p1	1 through 8 **or** list of numbers 1 through 8 **or** VI*n*	axis number
Range:		
default	0	
allowed values	0, 1	

Use: This register is used to determine whether position playback is enabled. If PLY is set to 1, then position playback is enabled; and if PLY is set to 0, it is disabled.

Example:

```
PPB2=5000                     (* set position pointer begin)
PPE2=10000                    (* set position pointer end)
PPI2=1                        (* set position pointer interval)
VI1=PPB2                      (* set integer variable)
MPA2=ITF(VIVI1)/ITF(URA2)     (* set motion absolute move)
RPA2                          (* run to absolute position)
PLY2=1                        (* enable playback of position)
```

What will happen: Setting the position pointer begin, end, and interval; loading the absolute move position with the first pointed position; issuing the run command; and enabling playback will cause the axis to move to the positions stored in integer variables 5,000 through 10,000 at 10 milliseconds per position.

Registers Used: PPB, PPE, PPI, PPR, PP

Utility Template: Multi-axis path recording

POE Power Output Stage Enable

Class:	Axis Register	
Type:	Boolean	
Syntax:		
I, jr	POE	
🖹	POE*p1* (e.g., POE2 POEVI1)	
Parameters:	*allowed values*	*description*
🖹 *p1*	1 through 8 **or** VI*n*	axis number
Range:		
default	1	
allowed values	0, 1	

Use: This register is used to determine whether the power output stage of the amplifier of the axis is enabled. If POE is set to 1, then the power output stage is enabled; and if POE is set to 0, it is disabled.

POP Pops "Gosub" Address from Top of "Gosub" Stack

Class: Program Command

Syntax: POP

Restrictions: Allowed only in programs

Use: This command pops the last gosub address from the top of the gosub stack. It causes the program to exit a subroutine without returning.

Examples:

IMC/IMJ	**Target**	
PROGRAM1	PROGRAM1	(* edit program 1)
MVL=5	MVL1=5	(* set motion velocity)
MAC=40	MAC1=40	(* set motion acceleration)
MPA=10	MPA1=10	(* set absolute move position)
GOSUB10	GOSUB10	(* unconditionally gosub 10)
GOTO20	GOTO20	(* unconditionally goto 20)
10 RPA	10 RPA1	(* run to absolute position)
11 IF IP GOTO12	11 IF IP1 GOTO12	(* conditionally goto 12)
IF FC <> 0 GOTO15	IF FCA1 <> 0 GOTO15	(* conditionally goto 15)
GOTO11	GOTO11	(* unconditionally goto 11)
12 RETURN	12 RETURN	(* return from gosub)
15 POP	15 POP	(* pop gosub address from top of gosub stack)
OUT "CONTROLLER FAULT$N"	OUTW "AXIS ONE FAULT$N"	(* output string expression to the display)
OUT "TYPE 'FC?' FOR MESSAGE$N"	OUTW "TYPE 'FCA1?' FOR MESSAGE$N"	(* output string expression to the display)
20 END	20 END	(* end program 1 and exit editor)

What will happen: This program, when executed, will set the velocity, acceleration rate, and absolute move position. Execution will then go to the subroutine at label 10, which will run the axis in the forward direction for 10 units. While the axis is running, the program checks two things: 1) to see if the axis is in position (IP or IP1); and 2) to see if a fault has occurred (FC<>0 or FCA1<>0). If a fault has occurred, the program execution will go to label 15. Then, the program will pop the address of label 10 off of the stack, print an error message, and end. If a fault does not occur, the program will return to the statement after "GOSUB10," which goes to the statement at label 20, which ends the program.

Related Commands: GOSUB, RETURN, RSTSTK

PP Position Pointer

Class:	Motion Register	
Type:	Integer	
Syntax:	PP*p1* (e.g., PP2 PPVI3)	
Parameters:	*allowed values*	*description*
p1	1 through 8 **or** VI*n*	axis number
Range:		
minimum	4,097	
maximum	262,144	
Restrictions:	Read only.	
Use:	This register contains the current value of the position pointer for record or playback.	
Related Registers:	PPB, PPE, PLY, REC	

PPB Position Pointer Begin

Class:	Motion Register	
Type:	Integer	
Syntax:	PPB*p1* (e.g., PPB2 PPBVI3)	
Parameters:	*allowed values*	*description*
p1	1 through 8 **or** VI*n*	axis number
Range:		
default	4,097	
minimum	4,097	
maximum	262,144	

Use: This register defines the beginning value of the position pointer for position playback or record. The pointer points to the extended integer variable space.

Remarks: The maximum value of this register assumes VFEA is set to the default of 2,048. If VFEA is set differently, the maximum will be reduced accordingly.

Example:

```
PPB2=5000                  (* set position pointer begin)
PPE2=10000                 (* set position pointer end)
PPI2=1                     (* set position pointer interval)
VI1=PPB2                   (* set integer variable)
MPA2=ITF(VIVI1)/ITF(URA2)  (* set motion absolute move)
RPA2                       (* run to absolute position)
PLY2=1                     (* enable playback of position)
```

What will happen: Setting the position pointer begin, end, and interval; loading the absolute move position with the first pointed position; issuing the run command; and enabling playback will cause the axis to move to the positions stored in integer variables 5,000 through 10,000 at 10 milliseconds per position.

Related Registers: PPE, PPI, PPR, PLY, REC

PPE Position Pointer End

Class:	Motion Register	
Type:	Integer	
Syntax:	PPE*p1* (e.g., PPE2 PPEVI3)	
Parameters:	*allowed values*	*description*
p1	1 through 8 **or** VI*n*	axis number
Range:		
default	4,097	
minimum	4,097	
maximum	262,144	

Use: This register determines the ending value of the position pointer for position playback or record. The pointer points to the extended integer variable space.

Remarks: The maximum value of this register assumes VFEA is set to the default of 2,048. If VFEA is set differently, the maximum will be reduced accordingly.

Example:

```
PPB2=5000                  (* set position pointer begin)
PPE2=10000                 (* set position pointer end)
PPI2=1                     (* set position pointer interval)
VI1=PPB2                   (* set integer variable)
MPA2=ITF(VIVI1)/ITF(URA2)  (* set motion absolute move)
RPA2                       (* run to absolute position)
PLY2=1                     (* enable playback of position)
```

What will happen: Setting the position pointer begin, end, and interval; loading the absolute move position with the first pointed position; issuing the run command; and enabling playback will cause the axis to move to the positions stored in integer variables 5,000 through 10,000 at 10 milliseconds per position.

Related Registers: PPB, PPI, PPR, PLY, REC

PPI Position Pointer Interval

Class:	Motion Register	
Type:	Floating point	
Syntax:	PPI*p1* (e.g., PPI2 PPI236 PPIVI3)	
Parameters:	*allowed values*	*description*
p1	1 through 8 **or** list of numbers 1 through 8 **or** VI*n*	axis number

Range:

default	1.0
minimum	0.1
maximum	10.0

Use: This register determines the time interval between positions during playback or record. The interval roughly corresponds to units of 10 milliseconds. For example, a value of 1.5 would be approximately 15 milliseconds.

Example:

```
PPB2=5000        (* set position pointer begin)
PPE2=10000       (* set position pointer end)
PPI2=1           (* set position pointer interval)
REC2=1           (* enable record positions)
```

What will happen: Setting the position pointer begin, end, and interval and enabling record positions will cause the Target to record the position of axis two in integer variables 5,000 to 10,000 every 10 milliseconds.

Related Registers: PLY, REC

PPR Position Pointer Repeat Enable

Class:	Motion Register	
Type:	Boolean	
Syntax:	PPR*p1* (e.g., PPR2 PPRVI3)	
Parameters:	*allowed values*	*description*
p1	1 through 8 **or** VI*n*	axis number
Range:		
default	0	
minimum	0, 1	
Use:	This register is used to determine whether position pointer repeat is enabled. If PPR is set to 1, then repeat is enabled; and when the position pointer reaches the end, it will be reloaded with the beginning value and continue. If PPR is set to 0, then repeat is not enabled.	
Related Registers:	PPB, PPE, PLY, REC	

PROG Program Executing

Class:	System Register	
Type:	Boolean	
Syntax:	PROG*p1* (e.g., PROG3 PROGVI4)	
Parameters:	*allowed values*	*description*
I, jr *p1*	1 through 4 **or** VI*n*	program number
🖹 *p1*	1 through 17 **or** VI*n*	program number
Range:		
allowed values	0, 1	
Restrictions:	Read only.	
Use:	The program executing register is used to determine whether a program is executing. If program *p1* is executing, then PROG*p1* will be 1; and if program *p1* is not executing, then PROG*p1* will be 0.	
Related Registers:	SRP	

PROGRAM Edits Program

Class:	Program Command	
Syntax:	PROGRAM*p1* (e.g., PROGRAM2)	
Parameters:	*allowed values*	*description*
I, jr *p1*	1 through 4	program number
🖹 *p1*	1 through 17	program number

Restrictions: Not allowed in programs or motion blocks.

Use: This command enters the line editor at the first statement of program *p1*. It is used either to view programs or to start editing them.

Remarks: This command will execute only when all axes have stopped and no programs or motion blocks are executing.

Examples:

IMC/IMJ	**Target**	
PROGRAM1	PROGRAM1	(* edit program 1)
PSA=0	PSA1=0	(* set axis position register)
MVL=10	MVL1=10	(* set motion velocity)
MAC=40	MAC1=40	(* set motion acceleration)
MPA=12	MPA1=12	(* load absolute move position)
RPA	RPA1	(* run to absolute move position)
END	END	(* end program 1 and exit editor)

Related Commands: MOTION, END, X, !, DEL, L, LABEL, FAULT

PSA Axis Position

Class:	Axis Register	
Type:	Floating point	
Syntax:		
I, jr	PSA	
🖹	PSA*p1* (e.g., PSA2 PSAVI3)	
Parameters:	*allowed values*	*description*
🖹 *p1*	1 through 8 **or** VI*n*	axis number
Range:		
units	axis units	
default	0 pulses	
minimum	-2,000,000,000 pulses	
maximum	2,000,000,000 pulses	

Use: This register is used to define the position of the axis.

Remarks: This register supports up to six decimal places. The numerical values for the default, minimum, and maximum of this register are assuming that the axis unit ratio, URA, is set at its default value of 1. If the axis unit ratio is set to a value other than 1, the default, maximum, and minimum values must be divided by the value of URA (see URA).

Example:

IMC/IMJ	**Target**	
PSA=0	PSA1=0	(* set axis position)
MVL=10	MVL1=10	(* set motion velocity)
MAC=40	MAC1=40	(* set motion acceleration)
MPA=8	MPA1=8	(* set absolute move position)
RPA	RPA1	(* run to absolute position)

What will happen: Setting the axis position, velocity, acceleration, and absolute move position and issuing the RPA command will cause the axis to move 8 units in the forward direction.

Related Registers: URA, PLA, PWE, OFA

PSC Command Position

Class:	Axis Register	
Type:	Floating point	
Syntax:		
I, jr	PSC	
🗎	PSC*p1* (e.g., PSC2 PSCVI3)	
Parameters:	*allowed values*	*description*
🗎 *p1*	1 through 8 **or** VI*n*	axis number
Range:		
units	axis units	
default	0 pulses	
minimum	-2,000,000,000 pulses	
maximum	2,000,000,000 pulses	
Restrictions:	Read only.	
Use:	This register is used to determine the command position of the axis. The command position is the controller's required position for the axis. The difference between this and the axis position, PSA, is called the following error, FE.	
Remarks:	The numerical values for the default, minimum, and maximum of this register are assuming that the axis unit ratio, URA, is set at its default value of 1. If the axis unit ratio is set to a value other than 1, the default, minimum, and maximum values must be divided by the value for URA (see URA).	
Related Registers:	PSA, FE, PSE	

PSE Command Position Only Enable

Class:	Axis Register	
Type:	Boolean	
Syntax:	PSE*p1* (e.g., PSE2 PSEVI1)	
Parameters:	*allowed values*	*description*
p1	1 through 8 **or** VI*n*	axis number
Range:		
default	0	
allowed values	0, 1	

Use: This register is used to determine whether the system will only calculate command positions and not move the motor to those positions. If PSE is set to 1, then no actual motion will occur; and only the command position will change when a move command is issued. If PSE is set to 0, then the axis will fault; and normal moves will be possible after the fault has been cleared.

PSO Offset Position

Class:	Axis Register	
Type:	Floating point	
Syntax:		
I, jr	PSO	
🖹	PSO*p1* (e.g., PSC2 PSCVI3)	
Parameters:	*allowed values*	*description*
🖹 *p1*	1 through 8 **or** VI*n*	axis number
Range:		
units	axis units	
default	0 pulses	
minimum	-2,000,000,000 pulses	
maximum	2,000,000,000 pulses	

Use: This register is used to define the offset position of the axis.

Remarks: This register supports up to six decimal places. The numerical values for the default, minimum, and maximum of this register are assuming that the axis unit ratio, URA, is set at its default value of 1. If the axis unit ratio is set to a value other than 1, the default, maximum, and minimum values must be divided by the value for URA (see URA).

Example:

IMC/IMJ	**Target**	
PSO=0	PSO1=0	(* set offset position)
MVL=10	MVL1=10	(* set motion velocity)
MAC=40	MAC1=40	(* set motion acceleration)
MPO=10	MPO1=10	(* set offset move position)
RPO	RPO1	(* run to offset move position)

What will happen: Setting the offset position, velocity, acceleration, and offset move position and issuing the RPO command will cause axis one to move 10 units in the forward direction.

Related Registers: URA

PSR Resolver Position

Class:	Axis Register	
Type:	Integer	
Syntax:		
I, jr	PSR	
🗎	PSR*p1* (e.g., PSR2 PSRVI3)	
Parameters:	*allowed values*	*description*
🗎 *p1*	1 through 8 **or** VI*n*	axis number
Range:		
minimum	0	
maximum	4,095 (resolver feedback brushless servo) or 65,535 (encoder feedback brushless servo)	
Restrictions:	Brushless servo only; read only.	
Use:	This register is used to determine the resolver position.	

PSX Auxiliary Position

Class:	Axis Register	
Type:	Floating point	
Syntax:		
I, jr	PSX	
🖹	PSX*p1* (e.g., PSX2 PSXVI3)	
Parameters:	*allowed values*	*description*
🖹 *p1*	1 through 8 **or** VI*n*	axis number
Range:		
units	auxiliary units	
default	0 pulses	
minimum	-2,000,000,000 pulses	
maximum	2,000,000,000 pulses	

Restrictions: For IMCs, this function available only with the extended command set.

Use: This register is used to define the auxiliary position of the axis. The auxiliary position is simply the position of the auxiliary encoder of the axis.

Remarks: This register supports up to six decimal places. The numerical values for the default, minimum, and maximum of this register are assuming that the auxiliary unit ratio, URX, is set at its default value of 1. If the auxiliary unit ratio is set to a value other than 1, the default, minimum, and maximum values must be divided by the value for URX (see URX).

Example:

IMC/IMJ	**Target**	
PSX=20	PSX1=20	(* set auxiliary position to 20 auxiliary units)
PSX?	PSXVI4?	(* report auxiliary position)

Related Registers: URX, PLX, OFX

PUT Puts One Character to Serial Port

Class:	Input/Output Command	
Syntax:	PUT *p1* (e.g., PUT VS1 PUT“A”)	
Parameters:	*allowed values*	*description*
p1	any string expression	string expression

Use: This command puts one character to the serial port. It takes the string expression and outputs only the first character to the serial port.

Example:

```
PUT VS1          (* put one character of string variable 1 to serial port)
PUT“Hello”       (* put one character of the string “Hello” to serial port [i.e., H])
```

Related Commands: GET, IN, OUT

PUTT Puts One Character to Tertiary Port

Class:	Input/Output Command	
Syntax:	PUTT *p1* (e.g., PUTT VS1 PUTT“A”)	
Parameters:	*allowed values*	*description*
p1	any string expression	string expression

Use: This command puts one character to the tertiary port. It takes the string expression and outputs only the first character to the tertiary port.

Example:

```
PUTT VS1        (* put one character of string variable 1 to tertiary port)
PUTT“Hello”     (* put one character of the string “Hello” to tertiary port
                [i.e., H])
```

Related Commands: GETT, INT, OUTT

PUTW Puts One Character to OIP

Class:	Input/Output Command	
Syntax:	PUTW *p1* (e.g., PUTW VS1 PUTW“A”)	
Parameters:	*allowed values*	*description*
p1	any string expression	string expression
Use:	This command puts one character to the display. It takes the string expression and outputs only the first character to the display.	

Example:

```
PUTW VS1          (* put one character of string variable 1 to display)
PUTW“Hello”       (* put one character of the string “Hello” to display [i.e., H])
```

Related Commands: GETW, INW, OUTW

PWE Position Register Wrap Enable

Class:	Axis Register	
Type:	Boolean	
Syntax:		
I, jr	PWE	
🖹	PWE*p1* (e.g., PWE2 PWEVI3)	
Parameters:	*allowed values*	*description*
🖹 *p1*	1 through 8 **or** VI*n*	axis number
Range:		
default	0	
allowed values	0, 1	
Restrictions:	Cannot be assigned in programs or motion blocks.	

Use: This register is used to determine whether position register wrap is enabled. If PWE is set to 1, position register wrap is enabled; and if PWE is set to 0, it is disabled.

Remarks: When position register wrap is enabled, the controller will use the axis position length, PLA, to define the upper and lower roll over limits for the *axis position* register (PSA) as -PLA axis units to PLA-(1/URA) axis units. Wrapping is required in unidirectional applications to prevent position register overflow or in applications where it makes sense to define a position modulus. PWE has no effect on the *auxiliary position* register (PSX), which always wraps. The setting of PWE has no effect on electronic cam mode.

Registers Used: PLA, PSA

PZA Axis Position Synchronized

Class:	Axis Register	
Type:	Floating point	
Syntax:		
I, jr	PZA	
🖹	PZA*p1* (e.g., PZA1 PZAVI3)	
Parameters:	*allowed values*	*description*
🖹 *p1*	1 through 8 **or** VI*n*	axis number
Range:		
units	axis units	
default	0 pulses	
minimum	-2,000,000,000 pulses	
maximum	2,000,000,000 pulses	

Restrictions: For IMCs, this function available only with the extended command set; read only.

Use: This register is used to synchronize the reading of the axis position and the auxiliary position. This register is read first, then the PZX register is read. By using these registers instead of the standard position registers (PSA and PSX), there will be no more than 10 microseconds between the two readings.

Remarks: Each time the PZA command is executed, the value in the axis position register (PSA) is latched into the PZA register and within 10 microseconds the value in the auxiliary position register (PSX) is latched into the PZX register. These values remain until the PZA command is executed again.

The numerical values for the default, minimum, and maximum of this register are assuming that the axis unit ratio, URA, is set at its default value of 1. If the axis unit ratio is set to a value other than 1, the default, minimum, and maximum values must be divided by the value for URA (see URA).

Examples:

IMC/IMJ	**Target**	
VF1=PZA-PZX	VF1=PZA1-PZX1	(* calculate difference between axis and auxiliary positions)

Related Registers: PZX, URA, PSA

PZX Auxiliary Position Synchronized

Class:	Axis Register	
Type:	Floating point	
Syntax:		
I, jr	PZX	
🖹	PZX*p1* (e.g., PZX1 PZXVI3)	
Parameters:	*allowed values*	*description*
🖹 *p1*	1 through 8 **or** VI*n*	axis number

Range:

units	auxiliary units
default	0 pulses
minimum	-2,000,000,000 pulses
maximum	2,000,000,000 pulses

Restrictions: For IMCs, this function available only with the extended command set; read only.

Use: This register is used to synchronize the readings of the auxiliary position and the axis position. The PZA register is read first, then this register is read. By using these registers instead of the standard position registers (PSA and PSX), there will be no more than 10 microseconds between the two readings.

Remarks: Each time the PZA command is executed the value in the axis position register (PSA) is latched into the PZA register and within 10 microseconds the value in the auxiliary position register (PSX) is latched into the PZX register. These values remain until the PZA command is executed again.

The numerical values for the default, minimum, and maximum of this register are assuming that the auxiliary unit ratio, URX, is set at its default value of 1. If the auxiliary unit ratio is set to a value other than 1, the default, minimum, and maximum values must be divided by the value for URX see URX).

Example:

IMC/IMJ	**Target**	
VF1=PZA-PZX	VF1=PZA1-PZX1	(* calculate difference between axis and auxiliary positions)

Related Registers: PZA, URX

Q Reports Value of Register

Class:	Diagnostic Command	
Syntax:	*p1*Q (e.g, AM1Q SRSQ PSAVI1Q MPAQ)	
Parameters:	*allowed values*	*description*
p1	any register	register
Restrictions:	Not allowed in programs or motion blocks.	
Use:	This command is used to report the value of any register. It is exactly the same as the ? command.	
Related Commands:	DGO, ?	

QTA Axis Feedback Quadrature Type *I*

Class:	Axis Register
Syntax:	QTA
Range:	
default	Q4
allowed values	PD pulse/direction Q1 quadrature x1 Q2 quadrature x2 Q4 quadrature x4
Restrictions:	Stepper and ampless servo only; not allowed in expressions.
Use:	This register is used to define the quadrature type for the axis feedback encoder input. The possibilities are listed below:
PD (pulse/direction)	Sets the input for a pulse input on channel A and a direction input on channel B.
Q1 (quadrature x1)	Sets the input for two pulse waveforms in quadrature with a pulse multiplier of 1.
Q2 (quadrature x2)	Sets the input for two pulse waveforms in quadrature with a pulse multiplier of 2.
Q4 (quadrature x4)	Sets the input for two pulse waveforms in quadrature with a pulse multiplier of 4.
Remarks:	The axis feedback is in the forward direction when 1) QTA=Q1, Q2, or Q4 and channel A leads channel B; 2) QTA=PD and channel B+ < channel B-.
Related Registers:	PSA, QTX

QTX Auxiliary Quadrature Type

Class: Axis Register

Syntax:

I, jr	QTX
🗎	QTX*p1* (e.g., QTX1 QTXVI2)

Parameters:

	allowed values	*description*
🗎 *p1*	1 through 8 **or** VI*n*	axis number

Range:

default	***All***	Q4
allowed values	***I*** 🗎	Q1 (quadrature x1)
	I 🗎	Q2 (quadrature x2)
	All	Q4 (quadrature x4)
	All	PD (pulse/direction)
	jr	CW (clockwise/counterclockwise)

Restrictions: For IMCs, this function available only with the extended command set; not allowed in expressions.

Use: This register is used to define the quadrature type for the auxiliary encoder input. The possibilities are listed below:

I 🗎 *Q1 (quadrature x1)* Sets the input for two pulse waveforms in quadrature with a pulse multiplier of 1.

I 🗎 *Q2 (quadrature x2)* Sets the input for two pulse waveforms in quadrature with a pulse multiplier of 2.

All *Q4 (quadrature x4)* Sets the input for two pulse waveforms in quadrature with a pulse multiplier of 4.

All *PD (pulse/direction)* Sets the input for a pulse input on channel A and a direction input on channel B.

jr *CW (clockwise/counterclockwise)* Sets the input for a pulse input on channel A for CW motion and a pulse input on channel B for CCW motion.

Remarks: The auxiliary encoder output will cause the auxiliary position register, PSX, to increase when:

I 🗎 1) QTX=Q1, Q2, or Q4 and channel A leads channel B;
I 🗎 2) QTX=PD and channel B+ < channel B-.

jr 1) QTX=Q4 and channel A leads channel B;
jr 2) QTX=PD and channel B+ > channel B-;
jr 3) QTX=CW and channel A has a pulse waveform and channel B does not.

Related Registers: PSX, QTA

RCA Runs Arc Segment Absolute Move with Center

Class: Motion Command

Syntax: RCA*p1,p2* (e.g., RCA34 RCA13,6 RCA345,678)

Parameters	*allowed values*	*description*
p1	2 or 3 axis numbers 1 through 8	trajectory axis numbers
p2	none **or** list of axis numbers 1 through 8	coordinated axis numbers

Use: This command runs the *p1* axes in an arc segment to their absolute move positions with a center at their absolute move distances. In the same amount of time, this command also runs the *p2* axes to their absolute move positions. When three trajectory axes are specified, the motion generated is a helix around a right-circular cylinder whose center line is specified by the three axes' MDA registers. The MDA register of the axis paralleled to the center line must be set to 2,000,000,000 pulses for a helical move.

Example:

```
PSA1=0        (* set absolute position)
PSA2=0        (* set absolute position)
PSA3=0        (* set absolute position)
MPA1=5        (* set absolute move position)
MPA2=5        (* set absolute move position)
MPA3=3.2      (* set absolute move position)
MDA1=2.5      (* set absolute move distance)
MDA2=2.5      (* set absolute move distance)
TAD=CW        (* set arc direction)
TVL=4         (* set trajectory velocity)
TFP=100       (* set trajectory feedrate to 100 percent)
TFA=500       (* set trajectory feedrate acceleration)
MAP3=20       (* set motion acceleration percentage to 20)
MJK3=100      (* set motion jerk percentage to 100)
RCA12,3       (* run to absolute move position)
```

What will happen: Setting the registers and issuing the RCA command will cause both axes 1 and 2 to move in a half circle clockwise to position 5 units at 4 units/second; axis 3 will move to position 3.2 units at the same time.

Related Commands: RCI, RCO, RTA

Registers Used: MPA, MDA, TAD, TFP, TVL, TFA, TFD, MAP, MDP, MJK

Motion Templates: 2-D arc segment using start, end, and center point: absolute move

RCI Runs Arc Segment Incremental Move with Center

Class:	Motion Command	
Syntax:	RCI*p1,p2* (e.g., RCI34 RCI13,6 RCI345,678)	
Parameters	*allowed values*	*description*
p1	2 or 3 axis numbers 1 through 8	trajectory axis numbers
p2	none **or** list of axis numbers 1 through 8	coordinated axis numbers

Use: This command runs the *p1* axes in an arc segment to their incremental move positions with a center at their incremental move distances. In the same amount of time, this command also runs the *p2* axes to their incremental move positions. When three trajectory axes are specified, the motion generated is a helix around a right-circular cylinder whose center line is specified by the three axes' MDA registers. The MDA register of the axis paralleled to the center line must be set to 2,000,000,000 pulses for a helical move.

Example:

```
MPI1=5          (* set incremental move position)
MPI2=5          (* set incremental move position)
MPI3=3.2        (* set incremental move position)
MDI1=2.5        (* set incremental move distance)
MDI2=2.5        (* set incremental move distance)
TAD=CW          (* set arc direction)
TVL=4           (* set trajectory velocity)
TFP=100         (* set trajectory feedrate to 100 percent)
TFA=500         (* set trajectory feedrate acceleration)
MAP3=20         (* set motion acceleration percentage to 20)
MJK3=100        (* set motion jerk percentage to 100)
RCI12,3         (* run to incremental move position)
```

What will happen: Setting the registers and issuing the RCI command will cause both axes 1 and 2 to move in a half circle clockwise incrementally 5 units at 4 units/second; axis 3 will move incrementally 3.2 units at the same time.

Related Commands: RCA, RCO, RTI

Registers Used: MPI, MDI, TAD, TFP, TVL, TFA, TFD, MAP, MDP, MJK

Motion Templates: 2-D arc segment using start, end, and center point: incremental move

RCO Runs Arc Segment Offset Move with Center

Class:	Motion Command	
Syntax:	RCO*p1,p2* (e.g., RCO34 RCO13,6 RCO345,678)	
Parameters	*allowed values*	*description*
p1	2 or 3 axis numbers 1 through 8	trajectory axis numbers
p2	none **or** list of axis numbers 1 through 8	coordinated axis numbers

Use: This command runs the *p1* axes in an arc segment to their offset move positions with a center at their offset move distances. In the same amount of time, this command also runs the *p2* axes to their offset move positions. When three trajectory axes are specified, the motion generated is a helix around a right-circular cylinder whose center line is specified by the three axes' MDA registers. The MDA register of the axis paralleled to the center line must be set to 2,000,000,000 pulses for a helical move.

Example:

```
PSO1=0          (* set offset position)
PSO2=0          (* set offset position)
PSO3=0          (* set offset position)
MPO1=5          (* set offset move position)
MPO2=5          (* set offset move position)
MPO3=3.2        (* set offset move position)
MDO1=2.5        (* set offset move distance)
MDO2=2.5        (* set offset move distance)
TAD=CW          (* set arc direction)
TVL=4           (* set trajectory velocity)
TFP=100         (* set trajectory feedrate to 100 percent)
TFA=500         (* set trajectory feedrate acceleration)
MAP3=20         (* set motion acceleration percentage to 20)
MJK3=100        (* set motion jerk percentage to 100)
RCO12,3         (* run to offset move position)
```

What will happen: Setting the registers and issuing the RCO command will cause both axes 1 and 2 to move in a half circle clockwise to position 5 units at 4 units/second; axis 3 will move to position 3.2 units at the same time.

Related Commands: RCI, RCA, RTO

Registers Used: MPO, MDO, TAD, TVL, TFP, TFA, TFD, MAP, MDP, MJK

Motion Templates: 2-D arc segment using start, end, and center point: offset move

REC Record Position Enable

Class:	Motion Register	
Type:	Boolean	
Syntax:	REC*p1* (e.g., REC1 REC245 RECVI4)	
Parameters:	*allowed values*	*description*
p1	1 through 8 **or** list of numbers 1 through 8 **or** VI*n*	axis number
Range:		
default	0	
allowed values	0, 1	

Use: This register is used to determine whether position record is enabled. If REC is set to 1, then position record is enabled; and if REC is set to 0, it is disabled.

Example:

```
PPB2=5000        (* set position pointer begin)
PPE2=10000       (* set position pointer end)
PPI2=1           (* set position pointer interval)
REC2=1           (* enable record positions)
```

What will happen: Setting the position pointer begin, end, and interval and enabling record positions will cause the Target to record the position of axis two in integer variables 5,000 to 10,000 every 10 milliseconds.

Registers Used: PPB, PPE, PPI, PPR, PP

Utility Template: Multi-axis path recording

REM Remark

Class:	Program Command	
Syntax:	REM*p1* (e.g., REM Program starts here)	
Parameters:	*allowed values*	*description*
p1	any string, 0 through 127 characters	text comment
Restrictions:	Allowed only in programs or motion blocks.	
Use:	This command is used to add textual comments to a program or motion block.	
Remarks:	Comments are stored as part of a program or motion block, but they are ignored while the program or motion block is executing.	

Example:

```
PROGRAM1            (* edit program 1)
REM Set update
   screen to 5      (* comment)
UPS=5               (* set update screen register)
```

REPEAT Repeats Motion from Start of Motion Block

Class:	Program Command
Syntax:	REPEAT
Restrictions:	Allowed only in motion blocks.
Use:	This command causes the motion block to repeat motion from the beginning of the motion block.

Examples:

IMC/IMJ	**Target**	
MOTION1	MOTION1	(* edit motion block 1)
	MBA1	(* assign axis one to motion block)
MVL=10	MVL1=10	(* set motion velocity)
MAC=40	MAC1=40	(* set motion acceleration)
MPI=15	MPI1=15	(* set incremental move position)
MPA=0	MPA1=0	(* set absolute move position)
RPI	RPI1	(* run to incremental position)
RPA	RPA1	(* run to absolute position)
REPEAT	REPEAT	(* repeat motion from beginning of motion block)
END	END	(* end motion block 1 and exit editor)

What will happen: This motion block, when executed, will load the velocity, acceleration rate, incremental move position, and absolute move position. Next, the axis will move 15 units in the forward direction. Once the motion is completed, the axis will then move 15 units in the reverse direction. It will repeat this motion until a motion command or another motion block is executed.

RETRIEVE Retrieves User Memory

Class:	System Command
Syntax:	RETRIEVE
Restrictions:	Not allowed in programs or motion blocks.
Use:	This command is used to retrieve user memory from nonvolatile memory.
Remarks:	This command will execute only when the controller or the system and all axes are faulted, the UPS register is set to zero, and no programs or motion blocks are executing.
Related Commands:	SAVE, AUTORET

RETURN Returns from "Gosub"

Class: Program Command

Syntax: RETURN

Restrictions: Allowed only in programs.

Use: This command causes the program to return from a subroutine to the statement after the gosub statement.

Examples:

IMC/IMC	**Target**	
PROGRAM1	PROGRAM1	(* edit program 1)
MVL=5	MVL1=5	(* set motion velocity)
MAC=40	MAC1=40	(* set motion acceleration)
MPA=10	MPA1=10	(* set absolute move position)
GOSUB10	GOSUB10	(* unconditionally gosub 10)
GOTO20	GOTO20	(* unconditionally goto 20)
10 RPA	10 RPA1	(* run to absolute position)
WAIT IP	WAIT IP1	(* wait for expression to be true)
OUT "Axis in position$N"	OUTW "Axis in position$N"	(* output string expression to display)
RETURN	RETURN	(* return from gosub)
20 END	20 END	(* end program 1 and exit editor)

What will happen: This program, when executed, will load the velocity, acceleration rate, and absolute move position. It will then go to the subroutine at label 10, which will run the axis in the forward direction for 10 units. Once the axis is in position, the program will print a string. The program will return to the statement after "GOSUB10," which goes to the statement at label 20, which ends the program.

Related Commands: GOSUB, POP, RSTSTK

REVISION Reports Firmware Revision

Class:	Diagnostic Command
Syntax:	REVISION
Restrictions:	Not allowed in programs or motion blocks.
Use:	This command reports the revision of the system firmware.

RHF Runs Forward to Home Input

Class:	Motion Command	
Syntax:		
I, jr	RHF	
🗎	RHF*p1* (e.g., RHF2 RHF357 RHFVI4)	
Parameters:	*allowed values*	*description*
🗎 *p1*	1 through 8 **or** list of numbers 1 through 8 **or** VI*n*	axis number

Use: This command runs forward to the home input.

Remarks: When this command is executed, the axis, or axes, will run forward until the home input is encountered. It will then stop and run back to the position where the home input was detected. The software overtravel limits, OTF and OTR, are ignored while the axis is homing.

Examples:

IMC/IMJ	**Target**	
PROGRAM1	PROGRAM1	(* edit program 1)
MVL=1	MVL1=1	(* set motion velocity)
MAC=50	MAC1=50	(* set motion acceleration)
RHF	RHF1	(* run forward to home input)
WAIT IP	WAIT IP1	(* wait for axis to be in position)
PSA=0	PSA1=0	(* set axis position register)
END	END	(* end program 1 and exit editor)

What will happen: This program, once executed, will first set the motion velocity and acceleration. It will then run the axis in the forward direction until the home input is encountered, wait for the axis to be in position, and then set the axis position register to 0.

Related Commands: RHR, RMF, ROF

Registers Used:

When MT=VEL	MVL, MAC, MDC, MJK, MFP, MFA, MFD
When MT=TIME	Command cannot be used
When MT=PULSE	Command cannot be used

Motion Templates: Run reverse until home input; run reverse until home and marker inputs

RHR Runs Reverse to Home Input

Class:	Motion Command
Syntax:	
I, jr	RHR
🖹	RHR*p1* (e.g., RHR3 RHR123 RHRVI9)

Parameters:	*allowed values*	*description*
🖹 *p1*	1 through 8 **or** list of numbers 1 through 8 **or** VI*n*	axis number

Use: This command runs reverse to the home input.

Remarks: When this command is executed, the axis, or axes, will run reverse until the home input is encountered. It will then stop and run back to the position where the home input was detected. The software overtravel limits, OTF and OTR, are ignored while the axis is homing.

Examples:

IMC/IMJ	**Target**	
PROGRAM1	PROGRAM1	(* edit program 1)
MVL=1	MVL1=1	(* set motion velocity)
MAC=50	MAC1=50	(* set motion acceleration)
RHR	RHR1	(* run reverse to home input)
WAIT IP	WAIT IP1	(* wait for axis to be in position)
PSA=0	PSA1=0	(* set axis position register)
END	END	(* end program 1 and exit editor)

What will happen: This program, once executed, will first set the motion velocity and acceleration. It will then run the axis in the reverse direction until the home input is encountered, wait for the axis to be in position, and then set the axis position register to 0.

Related Commands: RHF, RMR, ROR

Registers Used:

When MT=VEL	MVL, MAC, MDC, MJK, MFP, MFA, MFD
When MT=TIME	Command cannot be used
When MT=PULSE	Command cannot be used

Motion Templates: Run reverse until home input; run reverse until home and marker inputs

RLA Runs Linear Interpolation Absolute

Class:	Motion Command	
Syntax:	RLA*p1,p2* (e.g., RLA12 RLA45,8 RLA237,56)	
Parameters	*allowed values*	*description*
p1	1, 2, or 3 axis numbers 1 through 8	trajectory axis numbers
p2	none **or** list of axis numbers 1 through 8	coordinated axis numbers

Use: This command runs the *p1* axes in a line segment to their absolute move positions. In the same amount of time, this command also runs the *p2* axes to their absolute move positions.

Example:

```
PSA1=0          (* set absolute position)
PSA2=0          (* set absolute position)
PSA3=0          (* set absolute position)
MPA1=4          (* set absolute move position)
MPA2=6          (* set absolute move position)
MPA3=2.3        (* set absolute move position)
TVL=3           (* set trajectory velocity)
TFP=100         (* set trajectory feedrate to 100 percent)
TFA=500         (* set trajectory feedrate acceleration)
MAP3=20         (* set motion acceleration percentage to 20)
MJK3=100        (* set motion jerk percentage to 100)
RLA12,3         (* run to absolute move position)
```

What will happen: Setting the registers and issuing the RLA command will cause axes 1 and 2 to move to positions 4 units and 6 units at 3 units/second; at the same time, axis 3 will move to position 2.3 units.

Related Commands: RLI, RLO, RPA

Registers Used: MPA, MDA, TAD, TFP, TVL, TFA, TFD, MAP, MDP, MJK

Motion Templates: 2-D line segment: absolute move

RLI Runs Linear Interpolation Incremental

Class:	Motion Command	
Syntax:	RLI*p1,p2* (e.g., RLI12 RLI45,8 RLI237,56)	
Parameters	*allowed values*	*description*
p1	1, 2, or 3 axis numbers 1 through 8	trajectory axis numbers
p2	none **or**	
	list of axis numbers 1 through 8	coordinated axis numbers

Use: This command runs the *p1* axes in a line segment to their incremental move positions. In the same amount of time, this command also runs the *p2* axes to their incremental move positions.

Example:

```
MPI1=4          (* set incremental move position)
MPI2=6          (* set incremental move position)
MPI3=2.3        (* set incremental move position)
TVL=3           (* set trajectory velocity)
TFP=100         (* set trajectory feedrate to 100 percent)
TFA=500         (* set trajectory feedrate acceleration)
MAP3=20         (* set motion acceleration percentage to 20)
MJK3=100        (* set motion jerk percentage to 100)
RLI12,3         (* run to incremental move position)
```

What will happen: Setting the registers and issuing the RLI command will cause axes 1 and 2 to move incrementally 4 units and 6 units at 3 units/second; at the same time, axis 3 will move incrementally 2.3 units.

Related Commands: RLA, RLO, RPI

Registers Used: MPI, MDI, TAD, TFP, TVL, TFA, TFD, MAP, MDP, MJK

Motion Templates: 2-D line segment: incremental move

RLO Runs Linear Interpolation Offset

Class:	Motion Command	
Syntax:	RLO*p1,p2* (e.g., RLO12 RLO45,8 RLO237,56)	
Parameters	*allowed values*	*description*
p1	1, 2, or 3 axis numbers 1 through 8	trajectory axis numbers
p2	none **or** list of axis numbers 1 through 8	coordinated axis numbers

Use: This command runs the *p1* axes in a line segment to their offset move positions. In the same amount of time, this command also runs the *p2* axes to their offset move positions.

Example:

```
PSO1=0        (* set offset position)
PSO2=0        (* set offset position)
PSO3=0        (* set offset position)
MPO1=4        (* set offset move position)
MPO2=6        (* set offset move position)
MPO3=2.3      (* set offset move position)
TVL=3         (* set trajectory velocity)
TFP=100       (* set trajectory feedrate to 100 percent)
TFA=500       (* set trajectory feedrate acceleration)
MAP3=20       (* set motion acceleration percentage to 20)
MJK3=100      (* set motion jerk percentage to 100)
RLO12,3       (* run to offset move position)
```

What will happen: Setting the registers and issuing the RLO command will cause axes 1 and 2 to move to positions 4 units and 6 units at 3 units/second; at the same time, axis 3 will move to position 2.3 units.

Related Commands: RLI, RLA, RPO

Registers Used: MPO, MDO, TAD, TFP, TVL, TFA, TFD, MAP, MDP, MJK

Motion Templates: 2-D line segment: offset move

RMF Runs Forward to Marker

Class: Motion Command

Syntax:

I, jr	RMF
🗎	RMF*p1* (e.g., RMF3 RMF78 RMFVI5)

Parameters:

	allowed values	*description*
🗎 *p1*	1 through 8 or list of numbers 1 through 8 or VI*n*	axis number

Use: This command runs forward to the marker. The marker is defined as the zero position on the resolver when using resolver feedback or as the encoder channel index input when using encoder feedback units.

Remarks: When this command is executed, the axis, or axes, will run forward at the velocity specified in the MVM register until the marker is encountered. It will then stop and run back to the position where the marker was detected. The software overtravel limits, OTF and OTR, are ignored while homing the axis.

Examples:

IMC/IMJ	**Target**	
PROGRAM1	PROGRAM1	(* edit program 1)
MVM=1	MVM1=1	(* set motion velocity for run to marker)
MAC=50	MAC1=50	(* set motion acceleration)
RMF	RMF1	(* run forward to marker)
WAIT IP	WAIT IP1	(* wait for axis to be in position)
PSA=0	PSA1=0	(* set axis position register)
END	END	(* end program 1 and exit editor)

What will happen: This program, once executed, will first set the motion velocity for run to marker and acceleration. It will then run the axis in the forward direction until the marker is encountered, wait for the axis to be in position, and then set the axis position register to 0.

Related Commands: RMR, RHF, ROF

Registers Used:

When MT=VEL	MVM, MAC, MDC, MJK, MFP, MFA, MFD
When MT=TIME	Command cannot be used
When MT=PULSE	Command cannot be used

Motion Templates: Run reverse until marker input; run reverse until home and marker inputs; run reverse until overtravel and marker inputs

RMR Runs Reverse to Marker

Class: Motion Command

Syntax:

I, jr	RMR
🖹	RMR*p1* (e.g., RMR5 RMR257 RMRVI1)

Parameters:

	allowed values	*description*
🖹 *p1*	1 through 8 **or** list of numbers 1 through 8 **or** VI*n*	axis number

Use: This command runs reverse to the marker. The marker is defined as the zero position on the resolver of resolver feedback units, or the encoder channel index input on encoder feedback units.

Remarks: When this command is executed, the axis, or axes, will run reverse at the velocity specified in the MVM register until the marker is encountered. It will then stop and run back to the position where the marker was detected. The software overtravel limits, OTF and OTR, are ignored while the axis is homing.

Examples:

IMC/IMJ	**Target**	
PROGRAM1	PROGRAM1	(* edit program 1)
MVM=1	MVM1=1	(* set motion velocity for run to marker)
MAC=50	MAC1=50	(* set motion acceleration)
RMR	RMR1	(* run reverse to marker)
WAIT IP	WAIT IP1	(* wait for axis to be in position)
PSA=0	PSA1=0	(* set axis position register)
END	END	(* end program 1 and exit editor)

What will happen: This program, once executed, will first set the motion velocity for run to marker and acceleration. It will then run the axis in the reverse direction until the marker is encountered, wait for the axis to be in position, and then set the axis position register to 0.

Related Commands: RMF, RHR, ROR

Registers Used:

When MT=VEL	MVM, MAC, MDC, MJK, MFP, MFA, MFD
When MT=TIME	Command cannot be used
When MT=PULSE	Command cannot be used

Motion Templates: Run reverse until marker input; run reverse until home and marker inputs; run reverse until overtravel and marker inputs

ROF Runs Forward to Overtravel Input

Class: Motion Command

Syntax:

I, jr	ROF
🖹	ROF*p1* (e.g., ROF3 ROF267 ROFVI2)

Parameters:

	allowed values	*description*
🖹 *p1*	1 through 8 **or** list of numbers 1 through 8 **or** VI*n*	axis number

Use: This command runs forward to the forward overtravel input.

Remarks: When this command is executed, the axis, or axes, will run until the forward overtravel input is encountered. It will then stop and run back to the position where the forward overtravel input was detected. The software overtravel limits, OTF and OTR, are ignored while the axis is homing. The hardware overtravel inputs do not need to be enabled (OTE=1) to use this command.

Examples:

IMC/IMJ	**Target**	
PROGRAM1	PROGRAM1	(* edit program 1)
MVL=1	MVL1=1	(* set motion velocity)
MAC=50	MAC1=50	(* set motion acceleration)
ROF	ROF1	(* run forward to overtravel input)
WAIT IP	WAIT IP1	(* wait for axis to be in position)
PSA=0	PSA1=0	(* set axis position register)
END	END	(* end program 1 and exit editor)

What will happen: This program, once executed, will first set the motion velocity and acceleration. It will then run the axis in the forward direction until the forward overtravel input is encountered, wait for the axis to be in position, and then set the axis position register to 0.

Related Commands: ROR, RHF, RMF

Registers Used:

When MT=VEL	MVL, MAC, MDC, MJK, MFP, MFA, MFD
When MT=TIME	Command cannot be used
When MT=PULSE	Command cannot be used

Motion Templates: Run reverse until overtravel inputs; run reverse until overtravel and marker inputs

ROR Runs Reverse to Overtravel Input

Class: Motion Command

Syntax:

I, jr	ROR
🖹	ROR*p1* (e.g., ROR5 ROR136 RORVI4)

Parameters:

	allowed values	*description*
🖹 *p1*	1 through 8 **or** list of numbers 1 through 8 **or** VI*n*	axis number

Use: This command runs reverse to the reverse overtravel input.

Remarks: When this command is executed, the axis, or axes, will run until the reverse overtravel input is encountered. It will then stop and run back to the position where the reverse overtravel input was detected. The software overtravel limits, OTF and OTR, are ignored while the axis is homing. The hardware overtravel inputs do not need to be enabled (OTE=1) to use this command.

Examples:

IMC/IMJ	**Target**	
PROGRAM1	PROGRAM1	(* edit program 1)
MVL=1	MVL1=1	(* set motion velocity)
MAC=50	MAC1=50	(* set motion acceleration)
ROR	ROR1	(* run reverse to overtravel input)
WAIT IP	WAIT IP1	(* wait for axis to be in position)
PSA=0	PSA1=0	(* set axis position register)
END	END	(* end program 1 and exit editor)

What will happen: This program, once executed, will first set the motion velocity and acceleration. It will then run the axis in the reverse direction until the reverse overtravel input is encountered, wait for the axis to be in position, and then set the axis position register to 0.

Related Commands: ROF, RHR, RMR

Registers Used:

When MT=VEL	MVL, MAC, MDC, MJK, MFP, MFA, MFD
When MT=TIME	Command cannot be used
When MT=PULSE	Command cannot be used

Motion Templates: Run reverse until overtravel inputs; run reverse until overtravel and marker inputs

RPA Runs to Absolute Position

Class:	Motion Command	
Syntax:		
I, jr	RPA	
🖹	RPA*p1* (e.g., RPA6 RPA358 RPAVI4)	
Parameters:	*allowed values*	*description*
🖹 *p1*	1 through 8 **or** list of numbers 1 through 8 **or** VI*n*	axis number

Use: This command runs the axis, or axes, to the absolute move position.

Remarks: The run commands override each other unless they are used in a motion block.

Examples:

IMC/IMJ	**Target**	
PSA=0	PSA1=0	(* set axis position register)
MVL=10	MVL1=10	(* set motion velocity)
MAC=40	MAC1=40	(* set motion acceleration)
MPA=8	MPA1=8	(* set absolute move position)
RPA	RPA1	(* run to absolute move position)

What will happen: Setting the axis position register, velocity, acceleration, and absolute move position and issuing the RPA command will cause the axis to move 8 units in the forward direction.

Related Commands: RPI, RPO, RVF, RVR

Registers Used:

When MT=VEL	MPA, MVL, MAC, MDC, MJK, MFP, MFA, MFD
When MT=TIME	MPA, MTM, MAP, MDP, MJK, MFP, MFA, MFD
When MT=PULSE	MPA, MPS, MPL, MAP, MDP, MVP

Motion Templates:

I, jr, 🖹	Velocity-based absolute move; velocity-based blended moves; velocity-based absolute move with feedrate override; time-based, single-axis, absolute move; time-based, single-axis absolute move with feedrate override; pulse-based, single-axis, absolute move; pulse-based, single-axis, blended move
🖹	Multi-axis absolute move

RPI Runs to Incremental Position

Class: Motion Command

Syntax:

I, jr	RPI
🖹	RPI*p1* (e.g., RPI8 RPI1256 RPIVI3)

Parameters:

	allowed values	*description*
🖹 *p1*	1 through 8 **or** list of numbers 1 through 8 **or** VI*n*	axis number

Use: This command runs the axis, or axes, to the incremental move position, MPI (i.e., it runs from the current position of the axis to the current position incremented by the value of MPI).

Remarks: The run commands override each other unless they are used in a motion block.

Examples:

IMC/IMJ	**Target**	
MVL=10	MVL1=10	(* set motion velocity)
MAC=40	MAC1=40	(* set motion acceleration rate)
MPI=12	MPI1=12	(* set incremental move position)
RPI	RPI1	(* run to incremental move position)

What will happen: Setting the velocity, acceleration, and incremental move position and issuing the RPI command will cause the axis to move 12 units in the forward direction.

Related Commands: RPA, RPO, RVF, RVR

Registers Used:

When MT=VEL	MPI, MVL, MAC, MDC, MJK, MFP, MFA, MFD
When MT=TIME	MPI, MTM, MAP, MDP, MJK, MFP, MFA, MFD
When MT=PULSE	MPI, MPS, MPL, MAP, MDP, MVP

Motion Templates:

I, jr, 🖹	Velocity-based incremental move; time-based, single-axis, incremental move; pulse-based, single-axis, incremental move; single-axis index move after input; single-axis index move at predefined auxiliary position reference
🖹	Time-based, multi-axis, incremental move; pulse-based, multi-axis, incremental move

RPO Runs to Offset Position

Class:	Motion Command	
Syntax:		
I, jr	RPO	
🗎	RPO*p1* (e.g., RPO6 RPO3457 RPOVI2)	
Parameters:	*allowed values*	*description*
🗎 *p1*	1 through 8 **or** list of numbers 1 through 8 **or** VI*n*	axis numbers

Use: This command runs the axis, or axes, to the offset move position.

Remarks: The run commands override each other unless they are used in a motion block.

Examples:

IMC/IMJ	**Target**	
PSO=0	PSO1=0	(* set offset position register)
MVL=10	MVL1=10	(* set motion velocity)
MAC=40	MAC1=40	(* set motion acceleration rate)
MPO=8	MPO1=8	(* set offset move position)
RPO	RPO1	(* run to offset move position)

What will happen: Setting the offset position register, velocity, acceleration, and offset move position and issuing the RPO command will cause the axis to move 8 units in the forward direction.

Related Commands: RPA, RPI, RVF, RVR

Registers Used:

When MT=VEL	MPO, MVL, MAC, MDC, MJK, MFP, MFA, MFD
When MT=TIME	MPO, MTM, MAP, MDP, MJK, MFP, MFA, MFD
When MT=PULSE	MPO, MPS, MPL, MAP, MDP, MVP

Motion Templates:

I, jr	Velocity-based offset move; time-based, single-axis, offset move; pulse-based, single-axis, offset move
🗎	Velocity-based offset move; time-based, single-axis, offset move; time-based, multi-axis, offset move; pulse-based, multi-axis, offset move

RSF Resets Faults *I jr*

Class:	System Command
Syntax:	RSF
Restrictions:	Not allowed in motion blocks.
Use:	This command resets all controller faults.
Remarks:	The RSF command sets the axis commanded position equal to the actual position, thus making axis following error and motor torque output equal to zero. Faults should be automatically reset by a program only after allowing appropriate inspection into the source of the fault.
Related Commands:	STF
Related Registers:	FC

RSFA Resets Axis Faults

Class:	System Command	
Syntax:	RSFA*p1* (e.g., RSFA2 RSFA356 RSFAVI4)	
Parameters:	*allowed values*	*description*
p1	1 through 8 **or** list of numbers 1 through 8 **or** VI*n*	axis number
Use:	This command resets axis*p1* faults.	
Related Commands:	RSFALL, RSFS, STFA, STFSALL, STFS	
Related Registers:	FCS	

RSFALL Resets System and All Axes' Faults

Class:	System Command
Syntax:	RSFALL
Use:	This command resets system and all axes' faults.
Related Commands:	RSFA, RSFS, STFA, STFALL, STFS
Related Registers:	FCA, FCS

RSFS Resets System Faults

Class:	System Command
Syntax:	RSFS
Use:	This command resets system faults.
Related Commands:	RSFA, RSFALL, STFA, STFALL, STFS
Related Registers:	FCS

RSM Resumes Motion

Class:	Program Command	
Syntax:		
I, jr	RSM	
🖹	RSM*p1* (e.g., RSM2 RSM123 RSMVI5)	
Parameters:	*allowed values*	*description*
🖹 *p1*	1 through 8 **or** list of numbers 1 through 8 **or** VI*n*	axis number
Restrictions:	Not allowed in motion blocks.	
Use:	This command resumes suspended motion of axis.	
Related Commands:	RSMALL, SUP, SUPALL	

RSMALL Resumes All Motion

Class:	Program Command
Syntax:	RSMALL
Restrictions:	Not allowed in motion blocks.
Use:	This command resumes all suspended axis motion.
Related Commands:	RSM, SUP, SUPALL

RSTSTK Resets “Gosub” Stack to Empty

Class: Program Command

Syntax: RSTSTK

Restrictions: Allowed only in programs.

Use: This command resets the gosub stack to empty.

Remarks: This command will eliminate all gosubs that have been executed.

Example:

```
        PROGRAM1                             (* edit program 1)
        IN VI1                               (* input variable value from key buffer)
        IF VI1>=0 GOTO5                      (* conditionally goto 5)
        OUT“-”                               (* output string expression to serial port)
        VI1=-VI1                             (* set integer variable 1)
5       VI2=10                               (* set integer variable 2 with pointer)
        GOSUB10                              (* unconditionally gosub 10)
        GOTO30                               (* unconditionally goto 30)
10      VIVI2=VI1 - VI1/10*10 + 48           (* set integer variable VI2)
        VI1=VI1/10                           (* set integer variable 1)
        VI2=VI2+1                            (* set integer variable 2 with next pointer)
        IF VI2>16 GOTO20                     (* conditionally goto 20)
        IF VI1<>0 GOSUB10                    (* conditionally gosub 10)
        VI2=VI2-1                            (* set integer variable 2 with pointer)
        OUT CHR(VIVI2)                       (* output string expression to serial port)
        RETURN                               (* return from gosub)
20      RSTSTK                               (* reset gosub stack to empty)
        OUT“ERROR:$N”                        (* output string expression to serial port)
        OUT“Number more than 6 digits$N”     (* output string expression to serial port)
30      END                                  (* end program 1 and exit editor)
```

What will happen: This program inputs an integer variable value from the key buffer. If the value is negative, the program sends a negative sign to the display, sets the integer value positive, and continues to label 5, which sets the variable pointer to 10. The program then goes to the subroutine at label 10, which stores the ASCII code of the ones digit in VI10, the ASCII code of the tens digit in VI11, etc. If the number of digits is greater than 6, the program goes to label 20, which resets the gosub stack and prints an error message; otherwise, each character of integer number VI1 will be sent to the serial port and the program ends at label 30.

Related Commands: POP, GOSUB

RTA Runs Arc Segment Absolute Move with Third Point

Class: Motion Command

Syntax: RTA*p1,p2* (e.g., RTA56 RTA24,7 RTA68,12)

Parameters	*allowed values*	*description*
p1	2 axis numbers 1 through 8	trajectory axis numbers
p2	none **or** list of axis numbers 1 through 8	coordinated axis numbers

Use: This command runs the *p1* axes in an arc segment to their absolute move positions where the arc segment includes the point at their absolute move distances. In the same amount of time, this command also runs the *p2* axes to their absolute move positions.

Example:

```
PSA1=0          (* set absolute position)
PSA2=0          (* set absolute position)
PSA3=0          (* set absolute position)
MPA1=5          (* set absolute move position)
MPA2=5          (* set absolute move position)
MPA3=3.2        (* set absolute move position)
MDA1=6.04       (* set absolute move distance)
MDA2=2.5        (* set absolute move distance)
TVL=4           (* set trajectory velocity)
TFP=100         (* set trajectory feedrate to 100 percent)
TFA=500         (* t trajectory feedrate acceleration)
MAP3=20         (* set motion acceleration percentage to 20)
MJK3=100        (* set motion jerk percentage to 100)
RTA12,3         (* run to absolute move position)
```

What will happen: Setting the registers and issuing the RTA command will cause both axes 1 and 2 to move in a half circle counterclockwise to position 5 units at 4 units/second; at the same time, axis 3 will move to position 3.2.

Related Commands: RTI, RTO, RCA

Registers Used: MPA, MDA, TAD, TFP, TVL, TFA, TFD, MAP, MDP, MJK

RTF Retrieves Firmware from Nonvolatile Memory

Class:	System Command
Syntax:	RTF
Use:	This command retrieves firmware from the flash memory card and puts it in code memory. It also disables all other commands except SVF.
Related Commands:	SVF

RTI Runs Arc Segment Incremental Move with Third Point

Class:	Motion Command	
Syntax:	RTI*p1,p2* (e.g., RTI56 RTI24,7 RTI68,12)	
Parameters	*allowed values*	*description*
p1	2 axis numbers 1 through 8	trajectory axis numbers
p2	none **or** list of axis numbers 1 through 8	coordinated axis numbers

Use: This command runs the *p1* axes in an arc segment to their incremental move positions where the arc segment includes the point at their incremental move distances. In the same amount of time, this command also runs the *p2* axes to their incremental move positions.

Example:

MPI1=5	(* set incremental move position)
MPI2=5	(* set incremental move position)
MPI3=3.2	(* set incremental move position)
MDI1=6.04	(* set incremental move distance)
MDI2=2.5	(* set incremental move distance)
TVL=4	(* set trajectory velocity)
TFP=100	(* set trajectory feedrate to 100 percent)
TFA=500	(* set trajectory feedrate acceleration)
MAP3=20	(* set motion acceleration percentage to 20)
MJK3=100	(* set motion jerk percentage to 100)
RTI12,3	(* run to incremental move position)

What will happen: Setting the registers and issuing the RTI command will cause both axes 1 and 2 to move in a half circle counterclockwise incrementally 5 units at 4 units/second; at the same time, axis 3 will move incrementally 3.2 units.

Related Commands: RTA, RTO, RCI

Registers Used: MPI, MDI, TAD, TFP, TVL, TFA, TFD, MAP, MDP, MJK

RTO Runs Arc Segment Offset Move with Third Point

Class:	Motion Command	
Syntax:	RTO*p1,p2* (e.g., RTO56 RTO24,7 RTO68,12)	
Parameters	*allowed values*	*description*
p1	2 axis numbers 1 through 8	trajectory axis numbers
p2	none **or** list of axis numbers 1 through 8	coordinated axis numbers

Use: This command runs the *p1* axes in an arc segment to their offset move positions where the arc segment includes the point at their offset move distances. In the same amount of time, this command also runs the *p2* axes to their offset move positions.

Example:

```
PSO1=0          (* set offset position)
PSO2=0          (* set offset position)
PSO3=0          (* set offset position)
MPO1=5          (* set offset move position)
MPO2=5          (* set offset move position)
MPO3=3.2        (* set offset move position)
MDO1=6.04       (* set offset move distance)
MDO2=2.5        (* set offset move distance)
TVL=4           (* set trajectory velocity)
TFP=100         (* set trajectory feedrate to 100 percent)
TFA=500         (* set trajectory feedrate acceleration)
MAP3=20         (* set motion acceleration percentage to 20)
MJK3=100        (* set motion jerk percentage to 100)
RTO12,3         (* run to offset move position)
```

What will happen: Setting the registers and issuing the RTO command will cause both axes 1 and 2 to move in a half circle counterclockwise to position 5 units at 4 units/second; at the same time, axis 3 will move to position 3.2.

Related Commands: RTI, RTA, RCO

Registers Used: MPO, MDO, TAD, TFP, TVL, TFA, TFD, MAP, MDP, MJK

RTU Remote Terminal Unit Mode Enable *I jr*

Syntax:	RTU
Type:	Boolean
Range:	
default	0
allowed values	0, 1
Restrictions:	Cannot be assigned in motion blocks. Available in IMJ firmware 2.1 and higher; IMC firmware 3.1 and higher.
Use:	The RTU enables the controller to communicate with a remote terminal unit (RTU). If RTU is set to 1, RTU mode is enabled; if RTU is set to 0, RTU mode is disabled.
Remarks:	IMC and IMJ controllers allow users to toggle back and forth between RTU mode and serial communication mode. With the controller in RTU mode, press the <Enter> key 10 consecutive times to send the controller back to serial communication mode with the currently set baud rate, odd parity, and 7 data bits set. Once in this mode it is not possible to set RTU=1 and it is necessary to cycle power on the controller to re-enable RTU communications.
Related Registers:	ADDR, BAUD, BIT, RTUF

RTUF Remote Terminal Unit Communication Flag *I jr*

Syntax:	RTUF
Type:	Boolean
Range:	
allowed values	0, 1
Restrictions:	Read only. Available in IMJ firmware 2.1 and higher; IMC firmware 3.1 and higher.
Use:	This register is used to tell whether remote terminal unit (RTU) communication is occurring. This operand is set to one when a RTU communication occurs correctly and is cleared to zero when its value is tested. A program can monitor correct RTU communication by testing RTUF at a rate slower than the RTU communication rate. As long as RTUF continues to return a value of 1, RTU communication is correctly taking place.
Related Registers:	RTU

RTV Retrieve Variable from Nonvolatile Memory to RAM *jr*

Syntax:	RTV
Restrictions:	Allowed only in programs.
Use:	The RTV command retrieves integer variables 1 through 1,024 and floating point variables 1 through 512 from nonvolatile memory (flash) to RAM.
Remarks:	The RTV command will fill variables with 1s bits if executed after the flash is erased until the SVV command is successfully executed. Units ship from the factory with all variables initialized to zero.
Related Registers:	SVV, VI, VF
Related Commands:	SAVE, SVV, RETRIEVE

RVF Runs to Velocity Forward

Class:	Motion Command	
Syntax:		
I, jr	RVF	
🗎	RVF*p1* (e.g., RVF4 RVF235 RVFVI6)	
Parameters:	*allowed values*	*description*
🗎 *p1*	1 through 8 **or** list of numbers 1 through 8 **or** VI*n*	axis number

Use: This command runs the axis, or axes, in the forward direction.

Remarks: The run commands override each other unless they are used in a motion block.

Examples:

IMC/IMJ	**Target**	
MVL=10	MVL1=10	(* set motion velocity)
MAC=50	MAC1=50	(* set motion acceleration)
RVF	RVF1	(* run forward)

What will happen: Loading the velocity and acceleration and issuing the RVF command will cause the axis to run in the forward direction until another motion command is issued.

Related Commands: RVR, RPA, RPI, RPO

Registers Used:

When MT=VEL	MVL, MAC, MDC, MJK, MFP, MFA, MFD
When MT=TIME	Command cannot be used
When MT=PULSE	MPS, MPL, MVP

Motion Templates: Run reverse until torque limit; velocity-based continuous move; run forward until torque limit; run reverse at torque limit; single-axis run forward until input

Utility Templates: Jog using OIP; jog using single-pole, double-throw switch

RVR Runs to Velocity Reverse

Class:	Motion Command	
Syntax:		
I, jr	RVR	
🗎	RVR*p1* (e.g., RVR8 RVR57 RVRVI2)	
Parameters:	*allowed values*	*description*
🗎 *p1*	1 through 8 **or** list of numbers 1 through 8 **or** VI*n*	axis number

Use: This command runs the axis, or axes, in the reverse direction.

Remarks: The run commands override each other unless they are used in a motion block.

Examples:

IMC/IMJ	**Target**	
MVL=10	MVL1=10	(* set motion velocity)
MAC=50	MAC1=50	(* set motion acceleration)
RVR	RVR1	(* run forward)

What will happen: Setting the velocity and acceleration and issuing the RVR command will cause the axis to run in the reverse direction until another motion command is issued.

Related Commands: RVF, RPA, RPI, RPO

Registers Used:

When MT=VEL	MVL, MAC, MDC, MJK, MFP, MFA, MFD
When MT=TIME	Command cannot be used
When MT=PULSE	MPS, MPL, MVP

Motion Templates: Run reverse until torque limit; velocity-based continuous move; run forward until torque limit; run reverse at torque limit; single-axis run forward until input

Utility Templates: Jog using OIP; jog using single-pole, double-throw switch

SAVE Saves User Memory

Class:	System Command
Syntax:	SAVE
Restrictions:	Not allowed in programs or motion blocks.
Use:	This command is used to save user memory from RAM to nonvolatile memory.
Remarks:	This command will execute only when the controller or system and all axes are faulted and no programs or motion blocks are executing.
jr	In IMJ controllers, executing the SAVE command automatically executes the AUTORET command.
Related Commands:	RETRIEVE, SVL, AUTORET

SCAN Maximum Scan Time

Syntax:	SCAN
Range:	
units	seconds
default	0
minimum	0.00
maximum	1.00
Restrictions:	Not allowed in programs, motion blocks, or expressions.
Use:	Use to define the maximum time allowed between updates of the I/O connection of the network. If the I/O connection is not updated in time, then the system will fault due to Network Communication Error and the FCN register will have bit 11 set to indicate I/O Scan Time-Out. If SCAN is set to zero, then no check of the update time is performed.
Example:	
SCAN=.05	(* set maximum scan time to 50 milliseconds)

SCRD Screen Data

Class:	Input/Output Register	
Syntax:	SCRD*p1.p2* (e.g., SCRD1.1 SCRDVI1.2 SCRDVI5.VI7)	
Parameters:	*allowed values*	*description*
p1	1 through 50 **or** VI*n*	screen number
p2	1 through 4 **or** VI*n*	line number
Restrictions:	Not allowed in expressions.	
Use:	This register is used to define screen data for line *p2* of screen number *p1*.	

Example:

SCRD1.1=FTS(VLA,5,2)	(* set screen data for screen 1, line 1 to axis velocity, field width of 5 and 2 decimal places)
SCRDVI1.2="Jogging"	(* set screen data for screen VI1, line 2 to "Jogging")

Related Registers: SCRP, SCRL, UPS

SCRL Screen Line

Class:	Input/Output Register	
Type:	String	
Syntax:	SCRL*p1.p2* (e.g., SCRL1.1 SCRLVI2.3 SCRLVI4.VI9)	
Parameters:	*allowed values*	*description*
p1	1 through 50 **or** VI*n*	screen number
p2	1 through 4 **or** VI*n*	line number
Range:		
default	“”	
allowed values	any string, 0 through 40 characters long	
Use:	This register is used to define a line of characters for line number *p2* of screen *p1*.	

Example:

```
SCRL1.1=“Axis velocity:”            (* set screen line 1 of screen 1 to “Axis velocity:”)
SCRLVI2.3=“Motion Parameters”  (* set screen line 3 of screen VI2 to “Motion
                                    Parameters”)
```

Related Registers: SCRD, SCRL, UPS

SCRP Screen Position of Data

Class:	Input/Output Register	
Type:	Integer	
Syntax:	SCRP*p1.p2* (e.g., SCRP1.1 SCRPVI2.3 SCRPVI3.VI6)	
Parameters:	*allowed values*	*description*
p1	1 through 50 **or** VI*n*	screen number
p2	1 through 4 **or** VI*n*	line number
Range:		
default	1	
minimum	1	
maximum	40	
Use:	This register is used to define the column position where the screen data, SCRD, is placed on the screen line.	
Example:		
SCRP1.1=15	(* set screen position of data for screen 1, line 1 to column 15)	
SCRPVI2.3=20	(* set screen position of data for screen VI2, line 3 to column 20)	
Related Registers:	SCRD, SCRL, UPS	

SECURE Secures User Memory

Class:	System Command
Syntax:	SECURE
Restrictions:	Not allowed in programs or motion blocks.
Use:	This command secures user memory space and protects user's intellectual property. It disables the PROGRAM, FAULT, and MOTION commands and prohibits programs or motion blocks from being uploaded to the controller. To re-enable these commands, you must execute the CLM command to clear the memory.
Related Commands:	PASSWORD, CHANGEPW

SM Servo Module Assignment

Class:	System Register	
Syntax:	SM*p1* (e.g., SM2 SM5)	
Parameters:	*allowed values*	*description*
p1	1 through 8	axis number

Range:

default	0
allowed values	0 **or** list of up to 8 rack slots separated by commas, where the rack slots are 11 through 18; 21 through 28; 31 through 38

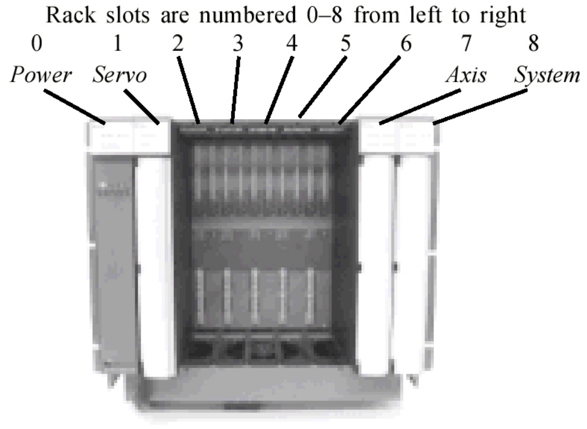
Restrictions: Not allowed in programs, motion blocks, or expressions.

Use: The servo module rack slot assignment is used to define which servo module(s) are assigned to an axis. The servo module assignment consists of a list of up to eight rack slots, where each rack slot consists of two digits. The first digit is the rack number and the second digit is the slot number. If SM*p1* is equal to 0, it means that no servo modules are assigned to axis *p1*.

Example:

SM1=13	(* set axis one servo module assignment to the servo module in rack one, slot three)
SM5=23,24,25	(* set axis 5 servo module assignment to the servo modules in rack two, slots three, four, and five)
SM7?	(* report axis 7 servo module assignment)

Target® Rack with Required Modules Installed



SME Servo Module Assignment Error

Class:	System Register
Type:	Integer, Boolean
Syntax:	SME*p1* (e.g., SME SME8 SMEVI2)
Parameters:	*allowed values* / *description*
p1	none **or** 0 through 23 **or** VI*n* / servo module assignment error register bit number
Range:	
allowed values	0 through $FFFFFF_{16}$ **or** 0 and 1
Restrictions:	Read only.
Use:	The servo module assignment error register is used to determine if any of the servo modules are not properly assigned by the system.

Remarks:

1. When the SME? command is executed, the module assignment error register value will be given as an English statement. If all servo module assignments are correct, the message given is *All module assignments are correct*.
2. If the computer interface format is enabled, and the SME? command is executed, the module assignment error register value will be given as an integer number. If all servo module assignments are correct, the module assignment error register will be set to 0. The possibilities are listed below:

bit	*message*
0	Module in rack one, slot one did not respond to assignment
1	Module in rack one, slot two did not respond to assignment
2	Module in rack one, slot three did not respond to assignment
3	Module in rack one, slot four did not respond to assignment
4	Module in rack one, slot five did not respond to assignment
5	Module in rack one, slot six did not respond to assignment
6	Module in rack one, slot seven did not respond to assignment
7	Module in rack one, slot eight did not respond to assignment
8	Module in rack two, slot one did not respond to assignment
...	...
...	...
22	Module in rack three, slot seven did not respond to assignment
23	Module in rack three, slot eight did not respond to assignment

SP*p1p2*B Set Point Begin *I*

Class: Input/Output Register

Type: Floating point

Syntax:

I	SP*p1p2*B (e.g., SPA1B SPFVI4B)
	SP*p1p2*B*p3* (e.g., SPA1B2 SPA1BVI2 SPAVI2B1)

Parameters:

		allowed values	*description*
I	*p1*	A through F	set point
	p2	1 through 8 **or** VI*n*	set point pair number
	p1	A	set point
	p2	1 through 8 **or** VI*n*	set point pair number
	p3	1 through 8 **or** VI*n*	axis number

Range:

units	axis units
default	OFF
minimum	-2,000,000,000 pulses
maximum	2,000,000,000 pulses

Use: This register is used to define a set point pair begin position for set points A through F for the IMC and set point A for Target. Up to eight pairs can be defined for each set point with *p2* being the pair designation. The pairs are defined by a begin point and an end point. These points are the position at which the set point will turn on and turn off, respectively. If a set point pair is set to OFF, it is not defined.

Remarks:

I IMC set point outputs are assigned to the following digital outputs: A=DO11,B=DO12, C=DO7, D=DO8, E=DO9, F=DO10.

Examples:

IMC	**Target**	
PSA=0	PSA1=0	(* set axis position)
MPA=20	MPA1=20	(* set absolute move position)
SPA1B=5	SPA1B1=5	(* set set point A pair one begin)
SPA1E=10	SPA1E1=10	(* set set point A pair one end)
SPA2B=12	SPA2B1=12	(* set set point A pair two begin)
SPA2E=20	SPA2E1=20	(* set set point A pair two end)
RPA	RPA1	(* run to absolute position)

What will happen: First, the axis position register is set and absolute move position is loaded. Next, the set point A pair one begin at 5 units and set point A pair one end at 10 units are loaded. Then, set point A pair two begin at 12 units and set point A pair two end at 20 units are loaded. Issuing the RPA command will cause the axis to move 20 units in the forward direction. Set point A will turn on at 5 units, turn off at 10 units, turn on again at 12 units, and finally turn off at 20 units.

Related Registers: URA, SP*p1p2*E, SPIA, SPOA

SP*p1p2*E Set Point End *I* 🖹

Class: Input/Output Register

Type: Floating point

Syntax:

I	SP*p1p2*E (e.g., SPA1E SPFVI4E)
🖹	SP*p1p2*E*p3* (e.g., SPA1E2 SPA1EVI2 SPAVI2E1 SPAVI1EVI2)

Parameters:

		allowed values	*description*
I	*p1*	A through F	set point
	p2	1 through 8 **or** Vi*n*	set point pair number
🖹	*p1*	A	set point
	p2	1 through 8 **or** VI*n*	set point pair number
	p3	1 through 8 **or** VI*n*	axis number

Range:

units	axis units
default	OFF
minimum	-2,000,000,000 pulses
maximum	2,000,000,000 pulses

Use: This register is used to define a set point pair end position for set points A through F for the IMC and set point A for Target. Up to eight pairs can be defined for each set point with *p2* being the pair designation. The pairs are defined by a begin point and an end point. These points are the position at which the set point will turn on and turn off, respectively. If a set point pair is set to OFF, it is not defined.

Remarks:

I IMC set point outputs are assigned to the following digital outputs: A=DO11,B=DO12, C=DO7, D=DO8, E=DO9, F=DO10.

Examples:

IMC	**Target**	
PSA=0	PSA1=0	(* set axis position)
MPA=20	MPA1=20	(* set absolute move position)
SPA1B=5	SPA1B1=5	(* set set point A pair one begin)
SPA1E=10	SPA1E1=10	(* set set point A pair one end)
SPA2B=12	SPA2B1=12	(* set set point A pair two begin)
SPA2E=20	SPA2E1=20	(* set set point A pair two end)
RPA	RPA1	(* run to absolute position)

What will happen: First, the axis position register is set and absolute move position is loaded. Next, the set point A pair one begin at 5 units and set point A pair one end at 10 units are loaded. Then, set point A pair two begin at 12 units and set point A pair two end at 20 units are loaded. Issuing the RPA command will cause the axis to move 20 units in the forward direction. Set point A will turn on at 5 units, turn off at 10 units, turn on again at 12 units, and finally turn off at 20 units.

Related Registers: URA, SP*p1p2*B, SPIA, SPOA

SP*p1*I Set Point Input *I*

Class:	Input/Output Register	
Syntax:	SP*p1*I (e.g., SPAI SPFI)	
Parameters:	*allowed values*	*description*
p1	A through F	set point
Range:		
default	PSA	
allowed values	PSA (axis position) PSX (auxiliary position)	
Restrictions:	Not allowed in programs or motion blocks. For IMCs, this function available only with the extended command set.	
Use:	This register sets the set point input for set points A through F.	
Related Registers:	PSA, PSX, SP*p1*T	

SP*p1*T Set Point Turn-off Time *I*

Class:	Input/Output Register	
Type:	Floating point	
Syntax:	SP*p1*T (e.g., SPAT SPFT)	
Parameters:	*allowed values*	*description*
p1	A through F	set point

Range:

units	seconds
default	0
minimum	0 seconds
maximum	2.0000 seconds

Use: This register is used to set an output pulse width in seconds for set points A through F.

Remarks: The SP*p1*T register overrides all output functions—return SP*p1*T to its default setting of zero to disable pulse output operation.

Example:

```
SPAT=1.075          (* set output pulse width of setpoint A to 1.075 seconds)
```

Related Registers: SP*p1p2*B, SP*p1p2*E

SPIA Axis Set Point Input

Class:	Input/Output Register	
Type:	Boolean	
Syntax:	SPIA*p1* (e.g., SPIA1 SPIAVI2)	
Parameters:	*allowed values*	*description*
p1	1 through 8 **or** VI*n*	axis number
Range:		
allowed values	0, 1	
Restrictions:	Read only.	
Use:	This register is used to determine the state of an axis set point input.	
Related Registers:	SPOA, SPAB	

SPIS System Set Point Input

Class:	Input/Output Register
Type:	Boolean
Syntax:	SPIS
Range:	
allowed values	0, 1
Restrictions:	Read only.
Use:	This register is used to determine the state of the system set point input.
Related Registers:	SPOS, SPS

SPOA Axis Set Point Output

Class:	Input/Output Register	
Type:	Boolean	
Syntax:	SPOA*p1* (e.g., SPOA1 SPOAVI2)	
Parameters:	*allowed values*	*description*
p1	1 through 8 **or** VI*n*	axis number
Range:		
default	0	
allowed values	0, 1	
Use:	The axis set point output is used to force the set point on. If SPOA*p1* is equal to 1, the set point output is forced on. If SPOA*p1* is equal to 0, the set point output is controlled by the axis set point pairs defined by SPAB and SPAE.	
Related Registers:	SPIA, SPAB	

SPOS System Set Point Output

Class:	Input/Output Register
Type:	Boolean
Syntax:	SPOS
Range:	
default	0
allowed values	0, 1
Use:	The system set point output is used to force the set point on. If SPOS is equal to 1, the set point output is forced on. If SPOS is equal to 0, the set point output is controlled by the system set point sets defined by SPS.
Related Registers:	SPIS, SPS

SPS System Set Point

Class:	Input/Output Register	
Syntax:	SPS*p1* (e.g., SPS2 SPSVI3)	
Parameters:	*allowed values*	*description*
p1	1 through 8 **or** VI*n*	set point set number
Range:		
default	OFF	
allowed values	set of up to 8 axis numbers, where each axis number can be 1 through 8	
Restrictions:	Not allowed in expressions.	
Use:	The system set point is an output that turns on whenever all of the axis set points in one of the defined sets are on simultaneously. It turns off whenever none of the defined sets have all of their axis set points on simultaneously. If a system set point set is set to OFF, it is not defined.	
Example:		
SPS1=134	(* set system set point set one to axis one, axis three and axis four set points)	
SPSVI2?	(* report system set point VI2)	
Related Registers:	SPIS, SPOS, SPAB	

SRA Axis Status

Class: System Register

Type: Integer, Boolean

Syntax:

I, jr	SRA*p1*	(e.g., SRA SRA4 SRAVI3)
🖹	SRA*p1.p2*	(e.g., SRA2 SRAVI1.3 SRAVI1.V12)

Parameters:

		allowed values	*description*
I, jr	*p1*	none **or** 0 through 15 **or** Vi*n*	axis status register bit number
🖹	*p1*	1 through 8 **or** VI*n*	axis number
	p2	none **or** 0 through 15 **or** VI*n*	axis status register bit number

Range:

allowed values 0 through FFFF$_{16}$ **or** 0 and 1

Restrictions: Read only.

Use: The axis status register is used to determine the status of the axis.

Remarks: 1. When the SRA? command is executed, the axis status register value will be given as an English statement.
2. If the computer interface format is enabled, and the SRA? command is executed, the axis status register value will be given as an integer number equal to the decimal equivalent of the register's binary value. Note that if the axis direction is reverse, bit 7 will be set to 0, and the associated message is *Axis direction reverse*. The possibilities are listed below:

bit	*message*		*bit*	*message*
0	Motion generator enabled		8	Axis in position
1	Gearing enabled		9	Axis at torque limit
2	Phase locked loop enabled		10	Axis at overtravel
3	Motion block executing		11	Axis at software overtravel
4	Phase error captured		12	Motion suspended
5	Phase error past bound		13	AXIS FAULT
6	Axis accel/decel		14	Cam enabled
7	Axis direction forward	***I, jr***	15	Reserved
		🖹	15	Play/record enabled

SRAM Analog Module Status

Class:	System Register
Type:	Integer, Boolean
Syntax:	SRAM*p1.p2* (e.g., SRAM1 SRAM2.VI3 SRAMVI1.VI2)

Parameters:

	allowed values	*description*
p1	1 through 4 **or** VI*n*	analog module number
p2	none **or** 0 through 15 **or** VI*n*	analog module status register bit number

Range:

allowed values	0 through $FFFF_{16}$ **or** 0 and 1

Restrictions: Read only.

Use: The analog module status register is used to determine the status of one of the analog modules.

Remarks: 1. When the SRAM*p1*? command is executed, the analog module status register value will be given as an English statement.
2. If the computer interface format is enabled, and the SRAM*p1*? command is executed, the analog module status register value will be given as an integer number. This number is the sum of all the powers of two associated with each analog module status register bit equal to 1. Note that if there is no module fault, bit 12 will be set to 0 and the associated message is *Module Functional*. The possibilities are listed below:

bit	*message*	*bit*	*message*
0	Reserved	7	System Communication Error
1	Reserved	8	Reserved
2	Reserved	9	Reserved
3	Reserved	10	Reserved
4	Reserved	11	Module Enabled
5	Reserved	12	MODULE FAULT
6	Reserved		

SRC Communication Status

Class:	System Register	
Type:	Integer, Boolean	
Syntax:	SRC*p1* (e.g., SRC SRC8 SRCVI2)	
Parameters:	*allowed values*	*description*
p1	none **or** 0 through 23 **or** VI*n*	communication status register bit number
Range:		
allowed values	0 through FFFFFF$_{16}$ **or** 0 and 1	
Restrictions:	Read only.	
Use:	The communication status register is used to determine if any communication between the modules is bad.	

Remarks:

1. When the SRC? command is executed, the communication status register value will be given as an English statement. If all communication is OK, the message given is *All communication is ok*.
2. If the computer interface format is enabled, and the SRC? command is executed, the communication status register value will be given as an integer number. If all communication is OK, the communication status register is set to 0. The possibilities are listed below:

bit	*message*	*bit*	*message*
0	Axis one communication is bad	12	Reserved
1	Axis two communication is bad	13	Reserved
2	Axis three communication is bad	14	Reserved
3	Axis four communication is bad	15	Reserved
4	Axis five communication is bad	16	Digital module one communication is bad
5	Axis six communication is bad	17	Digital module two communication is bad
6	Axis seven communication is bad	18	Digital module three communication is bad
7	Axis eight communication is bad	19	Digital module four communication is bad
8	Analog module one communication is bad	20	Digital module five communication is bad
9	Analog module two communication is bad	21	Digital module six communication is bad
10	Analog module three communication is bad	22	Digital module seven communication is bad
11	Analog module four communication is bad	23	Digital module eight communication is bad

SRDM Digital Module Status

Class:	System Register	
Type:	Integer, Boolean	
Syntax:	SRDM*p1.p2* (e.g., SRDM1 SRDM2.VI3 SRDMVI1.VI2)	
Parameters:	*allowed values*	*description*
p1	1 through 8 **or** VI*n*	digital module number
p2	none **or** 0 through 15 **or** VI*n*	digital module status register bit number
Range:		
allowed values	0 through $FFFF_{16}$ **or** 0 and 1	
Restrictions:	Read only.	
Use:	The digital module status register is used to determine the status of one of the digital modules.	

Remarks:

1. When the SRDM*p1*? command is executed, the digital module status register value will be given as an English statement.
2. If the computer interface format is enabled and the SRDM*p1*? command is executed, the digital module status register value will be given as an integer number. Note that if there is no module fault, bit 12 will be set to 0, and the associated message is *Module Functional*. The table below lists the possibilities:

bit	*message*
0	Reserved
1	Output Fault
2	Reserved
3	Reserved
4	Reserved
5	Reserved
6	24 Volt Supply Fault
7	System Communication Error
8	Reserved
9	Reserved
10	Reserved
11	Module Enabled
12	MODULE FAULT

SRP Program Status

Class: System Register

Type: Integer, Boolean

Syntax: SRP*p1.p2* (e.g., SRP1 SRPVI1.3 SRP2.VI3 SRPVI1.VI2)

Parameters:

	allowed values	*description*
p1	1 through 4 **or** VI*n*	program number
p2	none **or** 0 through 15 **or** VI*n*	program status register bit number

Range:

allowed values	0 through $FFFF_{16}$ **or** 0 and 1

Restrictions: Read only.

Use: The program status register is used to determine the status of a program.

Remarks: 1. When the SRP*p1*? command is executed, the program status will be given as an English statement. If the program is not executing, the message given is *Program not executing.*
2. If the computer interface format is enabled and the SRP*p1*? command is executed, the program status will be given as an integer number equal to the decimal equivalent of the register's binary value. Note that if the program is not executing, bit 0 will be set to 0, and the associated message is *Program not executing*. The possibilities are listed below:

	bit	*message*		*bit*	*message*
	0	Program executing	***I***	9	Screen lines save failure
	1	Program locked out	***jr***	9	Variable save failure
	2	Reserved	🗎	9	Reserved
	3	Reserved		10	Reserved
	4	Invalid digit in string		11	Reserved
	5	String value out of range		12	Reserved
	6	Floating point value out of range		13	Reserved
I 🗎	7	Invalid time/date		14	Reserved
jr	7	Reserved		15	PROGRAM FAULT
	8	Invalid command acknowledgment			

SRS System Status

Class: System Register

Type: Integer, Boolean

Syntax: SRS*p1* (e.g., SRS SRS8 SRSVI2)

Parameters:	*allowed values*	*description*
p1	none **or** 0 through 15 **or** VI*n*	system status register bit number

Range:

allowed values 0 through $FFFF_{16}$ **or** 0 and 1

Restrictions: Read only.

Use: The system status register is used to determine the status of the system.

Remarks: 1. When the SRS? command is executed, the system status register value will be given as an English statement.
2. If the computer interface format is enabled and the SRS? command is executed, the system status register value will be given as an integer number. Note that if no program is executing, bit 0 will be set to 0, and the associated message is *No program executing*. The possibilities are listed below:

	bit	*message*		*bit*	*message*
	0	Program executing	***I***, ***jr***	9	Reserved
	1	Program locked out	🗎	9	Variable save failure
	2	Reserved	***I***, ***jr***	10	Reserved
	3	Motion block executing	🗎	10	Axis at overtravel
I, ***jr***	4	Key buffer empty	***I***, ***jr***	11	Reserved
🗎	4	User receive buffer empty	🗎	11	Axis at software overtravel
I, ***jr***	5	Transmit buffer empty		12	I/O FAULT
🗎	5	User transmit buffer empty		13	AXIS FAULT
	6	Network connection available		14	SYSTEM FAULT
	7	Network on-line		15	MEMORY FAULT
I, ***jr***	8	Reserved			
🗎	8	All axes in position			

SRSM Servo Module Status

Class: System Register

Type: Integer, Boolean

Syntax: SRSM*p1.p2.p3* (e.g., SRSM1.1 SRSM2.1.5 SRSMVI1.1.VI2)

Parameters:

	allowed values	*description*
p1	1 through 8 **or** VIn	axis number
p2	1 through 8	servo module number
p3	none **or** 0 through 15 **or** VI*n*	servo module status register bit number

Range:

allowed values 0 through $FFFF_{16}$ or 0 and 1

Restrictions: Read only.

Use: The servo module status register is used to determine the status of one of the servo modules.

Remarks: 1. When the *SRSM*p1.p2? command is executed, the servo module status register value will be given as an English statement.
2. If the computer interface format is enabled, and the SRSM*p1*.p2? command is executed, the servo module status register value will be given as an integer number. If there is no module fault, bit 12 will be set to 0; the associated message *is Module Funct*ional. The possibilities are listed belo*w:*

bit	*message*	*bit*	*message*
0	Under-Voltage	7	Axis Communication Error
1	Over-Voltage	8	Servo Module Communication Error
2	Clamp Excessive Duty Cycle	9	Reserved
3	Clamp Current Fault	10	Reserved
4	Current Fault	11	Module Enabled
5	Over-Temperature	12	MODULE FAULT
6	Power Module Over-Temperature		

SRT Tertiary Status

Class:	System Register	
Type:	Integer, Boolean	
Syntax:	*S*RTp1 (e.g., SRT SRT2 SRTVI3)	
Parameters:	*allowed values*	*description*
p1	none **or** 0 through 15 **or** VI*n*	tertiary status register bit number
Range:		
*allowed va*lues	0 through FFFF$_{16}$ **or** 0 and 1	
Restrictions:	Read only.	
Use:	The tertiary status register is used to determine the status of the tertiary and operator interface buffers.	

Remarks:

1. When the SRT? command is executed, the tertiary status register value will be given as an English statement.
2. If the computer interface format is enabled and the SRT? command is executed, the tertiary status register value will be given as an integer number. Note that if no status bits are set, the associated messag*e* is *No tertiary status a*ctive. The possibilities are listed below:

bit	*message*
0	Reserved
1	Reserved
2	Key buffer empty
3	Program transmit buffer empty
4	Tertiary receive buffer empty
5	Tertiary transmit buffer empty
6–15	Reserved
16	No tertiary status active

ST Stops Motion

Class: Motion Command

Syntax:

I, jr	ST
🗎	ST*p1* (e.g., ST ST3 ST567 STVI2)

Parameters:

	allowed values	*description*
🗎 *p1*	none **or** 1 through 8 **or** list of numbers 1 through 8 **or** VI*n*	axis number

Restrictions:

🗎 Not allowed in motion blocks without a specified axis.

Use: This command stops all motion.

Remarks: This command, once executed, will immediately decelerate the axis at the deceleration loaded. For the Target, if *p1* is not specified, all axes will stop.

Examples:

IMC/IMJ	**Target**	
MVL=10	MVL1=10	(* set motion velocity)
MAC=20	MAC1=20	(* set motion acceleration)
MDC=40	MDC1=40	(* set motion deceleration)
RVF	RVF1	(* run forward)
ST	ST1	(* stop all motion)

What will happen: Setting the velocity, acceleration, and deceleration and issuing the RVF command will cause the axis to run forward. Issuing the ST command will decelerate the axis at 40 units/sec^2 and stop all motion.

Related Commands: HT, STT

Registers Used:

When MT=VEL	MDC, MJK
When MT=TIME	No registers used
When MT=PULSE	MPS, MPL

Motion Template: Single-axis run forward until input

Utility Templates: Jog using OIP; jog using single-pole, double-throw switch

STEP Step Input

Class:	Motion Command	
Syntax:		
I*, *jr	STEP*p1* (e.g., STEP100 STEPVI1)	
🖹	STEP*p1.p2* (e.g., STEP3.100 STEP45.100 STEPVI1.VI2)	
Parameters:	*allowed values*	*description*
I, jr *p1*	-16,000 through 16,000 **or** VI*n*	number of pulses
🖹 *p1*	1 through 8 **or** list of numbers 1 through 8 **or** Vi*n*	axis number
p2	-16,000 through 16,000 **or** VI*n*	number of pulses
Restrictions:	Servo only; not allowed in motion blocks.	
Use:	This command applies a step input to the axis or axes.	
Remarks:	The step input cannot be larger than the following error bound, FEB.	
Related Registers:	FEB	

STF Sets Fault *I jr*

Class:	System Command
Syntax:	STF
Restrictions:	Not allowed in motion blocks.
Use:	This command faults the controller.
Remarks:	If this command is in a program, executing STF will fault the program, which will stop program execution.
Related Commands:	RSF
Related Registers:	FC

STFA Sets Axis Fault

Class:	System Command	
Syntax:	STFA*p1* (e.g., STFA3 STFA145 STFAVI6)	
Parameters:	*allowed values*	*description*
p1	1 through 8 **or** list of numbers 1 through 8 **or** VI*n*	axis number
Use:	This command sets axis *p1* fault.	
Related Commands:	RSFA, RSFALL, RSFS, STFALL, STFS	

STFALL Sets System and All Axes' Fault

Class:	System Command
Syntax:	STFALL
Use:	This command sets system and all axes' fault.
Remarks:	If this command is in a program, executing STFALL will fault the program, which will stop program execution.
Related Commands:	RSFA, RSFALL, RSFS, STFA, STFS

STFS Sets System Fault

Class:	System Command
Syntax:	STFS
Use:	This command sets the system fault.
Remarks:	If this command is in a program, executing STFS will fault the program, which will stop program execution.
Related Commands:	RSFA, RSFALL, RSFS, STFA, STFALL

STM Start Time of Timer

Class:	System Register	
Type:	Floating point	
Syntax:	STM*p1* (e.g., STM2 STMVI3)	
Parameters:	*allowed values*	*description*
I, jr *p1*	1 through 8 **or** VI*n*	timer number
🖹 *p1*	1 through 16 **or** VI*n*	timer number

Range:

units	seconds
default	2,000,000.000
minimum	.001
maximum	2,000,000.000

Use: This register is used to define the starting time from which a timer will count down continuously to zero seconds. Once a timer is set, it will immediately start counting. For example, after you enter **STM1=7**, timer one would be set to seven seconds and would immediately start to count down to zero seconds. Once it has reached zero seconds, it would start again at seven seconds, count down to zero seconds, and so on.

Examples:

STMVI2=5	(* set start time of timer VI2 to five seconds)
STM3?	(* report start time of timer three)

Related Registers**:** TMR, TM

STT Stops Trajectory Motion

Class:	Motion Command
Syntax:	STT
Use:	This command stops trajectory motion.
Remarks:	This command, once executed, will immediately decelerate all trajectory axes at the trajectory feedrate deceleration loaded.

Example:

```
TVL=5          (* set trajectory velocity)
TFA=500        (* set trajectory feedrate acceleration)
MPI1=10        (* set incremental position)
MPI2=20        (* set incremental position)
RLI12          (* run incremental linear)
TFD=1000       (* set trajectory feedrate deceleration)
STT            (* stop trajectory motion)
```

What will happen:	Setting the trajectory velocity, trajectory feedrate acceleration, and incremental positions and issuing the RLI command will cause axes one and two to move in a line. Issuing the STT command will cause the axes to decelerate to a stop.
Related Commands:	HTT, ST
Registers Used:	TFD

STVB…GOTO Sets Boolean Variable; "Gotos" Label

Class:	Program Command
Syntax:	STVB*p1* GOTO*p2* (e.g., STVB1 GOTO30 STVBVI1 GOTOVI2)

Parameters:

	allowed values	*description*
p1	1 through 256 **or** VI*n*	Boolean variable number
p2	1 through 999 **or** VI*n*	label number

Restrictions: Allowed only in programs.

Use: This command sets Boolean variable *p1* and then checks to see if it was previously set. If Boolean variable *p1* was not set, this command will cause the program to go to label *p2*.

Example:

```
   PROGRAM1                PROGRAM2                (* edit program)
10 VI1=VI1+1            10 VI2=VI2+1               (* set integer variable)
   IF VI1<1000 GOTO10      IF VI2<996 GOTO10       (* conditionally goto 10)
   STVB1 GOTO20            STVB1 GOTO20            (* set Boolean variable 1 and if
                                                   (* Boolean variable 1 wasn't set, goto 20)
   GOTO10                  GOTO10                  (* unconditionally goto 10)
20 OUT"VI1="            20 OUT"VI2="               (* output string expression to the serial port)
   OUT ITS(VI1,5)+"$N"     OUT ITS(VI2,5)+"$N"     (* output string expression to the serial port)
   VI1=0                   VI2=0                   (* load integer variable)
   VB1=0                   VB1=0                   (* reset Boolean variable 1)
   GOTO10                  GOTO10                  (* unconditionally goto 10)
   END                     END                     (* end program and exit editor)
```

What will happen: These two programs, when executed, will increment integer variables 1 and 2 until they reach 1,000 and 996 respectively. The first program to finish this task will set *p1* equal to 1; and, since it was not previously set, it will go to the statement at label 20, which outputs the value to the display, loads 0 into the integer variable, and resets Boolean variable 1. If one program finishes this task while the other is outputting the value to the display, the program will go back to label 10, increment the integer variable, and check again for Boolean variable 1 to be reset.

Related Commands: IF...GOTO

SUP Suspends Motion

Class:	Program Command	
Syntax:		
I, jr	SUP	
🗎	SUP*p1* (e.g., SUP5 SUP346 SUPVI2)	
Parameters:	*allowed values*	*description*
🗎 *p1*	1 through 8 **or** list of numbers 1 through 8 **or** VI*n*	axis numbers
Restrictions:	Not allowed in motion blocks.	
Use:	This command suspends axis motion.	
Remarks:	Motion will continue to be suspended until the RSM or RSMALL command is executed, which resumes the motion. If, however, a motion command is issued while motion is suspended, the suspended motion will be eliminated.	
Related Commands:	RSM, RSMALL, SUPALL	

SUPALL Suspends All Motion

Class:	Program Command
Syntax:	SUPALL
Restrictions:	Not allowed in motion blocks.
Use:	This command suspends all motion.
Remarks:	All motion will continue to be suspended until the RSM or RSMALL command is executed, which resumes the motion. If, however, a motion command is issued while motion is suspended, the suspended motion will be eliminated.
Related Commands:	RSM, RSMALL, SUP

SVF Saves Firmware in Nonvolatile Memory

Class:	System Command
Syntax:	SVF
Use:	This command saves firmware in the flash memory card from code memory. It can be executed only after RTF. The SVF command enables all other commands.
Related Commands:	RTF

SVL Saves Screen Lines *I jr*

Class:	System Command
Syntax:	SVL
Restrictions:	Not allowed in motion blocks.
Use:	This command is used to save the screen lines from RAM to nonvolatile memory.
Remarks:	If the screen lines are not saved correctly, then bit 9 in the program status register will be set to 1, which means *Screen Lines Save Failure*.
Related Commands:	RETRIEVE, SAVE

SVV Save Variables from RAM to Nonvolatile Memory *jr*

Class:	System Command
Syntax:	SVV
Restrictions:	Allowed only in programs. SVV will execute only when the profile generator is not running (i.e., SRA bits 0, 1, 2 and 14 are false).
Use:	The SVV command saves integer variables 1 through 1,024 and floating point variables 1 through 512 from RAM to nonvolatile memory. SVV will execute only when the profile generator is not running, i.e., SRA bits 0–2 and SRA bit 14 are all false.
Remarks:	Test the state of the variable save failure bit (bit 9) in the Program Status (SRP) register after each SVV command in a program to ensure SVV completed successfully. If the variables are not saved correctly, then bit 9 in the Program Status Register (SRP) will be set to 1, which means *Variable Save Failur*e. **Caution: The controller flash memory can support a finite number of write cycles before the flash memory will fail. Although the typical limit for this type of flash is +100,000 write cycles, it is easy to exceed this limit by executing frequent SVV commands from within a program.**
Related Registers:	VI, VF
Related Commands:	SAVE, RETRIEVE, RTV

TAD Trajectory Arc Direction

Class:	Motion Register
Syntax:	TAD
Range:	
default	CW
allowed values	CW (clockwise)
	CCW (counterclockwise)
Use:	This register determines the direction of arc moves.

TBA Trajectory Motion Buffer Available

Class:	Motion Register
Type:	Boolean
Syntax:	TBA
Range:	
allowed values	0, 1
Restrictions:	Read only.
Use:	The TBA register reports a value of 1 when there is buffer space available for a trajectory move to be buffered; it reports a value of 0 when the buffer is full.

TFA Trajectory Feedrate Acceleration/Deceleration

Class:	Motion Register
Type:	Integer
Syntax:	TFA
Range:	
units	percent/second
default	1,000
minimum	1
maximum	200,000

Use: This register is used to define both an acceleration and a deceleration rate for the trajectory feedrate percentage. Define the deceleration rate separately with TFD. In cases where the acceleration rate differs from the deceleration rate, you must set TFA first and TFD second.

Example:

```
PSA1=0        (* set axis one position)
PSA2=0        (* set axis two position)
MPA1=3        (* set axis one absolute position)
MPA2=5        (* set axis two absolute position)
TVL=4         (* set trajectory velocity)
TFA=500       (* set trajectory feedrate acceleration)
TFD=200       (* set trajectory feedrate deceleration)
TFP=80        (* set trajectory feedrate percentage)
RLA12         (* run linear trajectory motion)
```

What will happen: Setting axis position, absolute move position, trajectory velocity, trajectory feedrate acceleration, and trajectory feedrate deceleration and issuing the run linear motion command will cause axes one and two to ramp up to 80 percent of 4 units/second (i.e., 3.2) of trajectory velocity at 500 percent/second and then ramp down to a stop at 3 units and 5 units at 200 percent/second.

Related Registers: TFD, TFP

TFD Trajectory Feedrate Deceleration

Class:	Motion Register
Type:	Integer
Syntax:	TFD
Range:	
units	percent/second
default	1,000
minimum	1
maximum	200,000

Use: This register is used to determine a deceleration rate for the trajectory feedrate percentage. In cases where the acceleration rate differs from the deceleration rate, you must set TFA first and TFD second.

Example:

```
PSA1=0        (* set axis one position)
PSA2=0        (* set axis two position)
MPA1=3        (* set axis one absolute position)
MPA2=5        (* set axis two absolute position)
TVL=4         (* set trajectory velocity)
TFA=500       (* set trajectory feedrate acceleration)
TFD=200       (* set trajectory feedrate deceleration)
TFP=80        (* set trajectory feedrate percentage)
RLA12         (* run linear trajectory motion)
```

What will happen: Setting axis position, absolute move position, trajectory velocity, trajectory feedrate acceleration, and trajectory feedrate deceleration and issuing the run linear motion command will cause axes one and two to ramp up to 80 percent of 4 units/second (i.e., 3.2) of trajectory velocity at 500 percent/second and then ramp down to a stop at 3 units and 5 units at 200 percent/second.

Related Registers: TFA, TFP

TFP Trajectory Feedrate Percentage

Class:	Motion Register
Type:	Floating Point
Syntax:	TFP
Range:	
units	percent
default	100.00
minimum	0.00
maximum	100.00

Use: This register is used to determine a feedrate percentage for the trajectory motion. The feedrate percentage causes the motion to run at a velocity that is a percentage of the trajectory velocity specified when the motion command was executed.

Remarks: This register is set to its default value on power-up.

Example:

```
PSA1=0          (* set axis one position)
PSA2=0          (* set axis two position)
MPA1=3          (* set axis one absolute position)
MPA2=5          (* set axis two absolute position)
TVL=4           (* set trajectory velocity)
TFA=500         (* set trajectory feedrate acceleration)
TFD=200         (* set trajectory feedrate deceleration)
TFP=80          (* set trajectory feedrate percentage)
RLA12           (* run linear trajectory motion)
```

What will happen: Setting axis position, absolute move position, trajectory velocity, trajectory feedrate acceleration, and trajectory feedrate deceleration and issuing the run linear motion command will cause axes one and two to ramp up to 80 percent of 4 units/second (i.e., 3.2) of trajectory velocity at 500 percent/second and then ramp down to a stop at 3 units and 5 units at 200 percent/second.

Related Registers: TFA, TFD, TVL

TIME Time Of Day *I* 🗎

Class:	System Register
Type:	String
Syntax:	TIME
Range:	
allowed values	00:00:00 through 23:59:59
Restrictions:	Cannot be assigned in motion blocks.
Use:	The time of day register is used to keep track of the time of day in 24-hour format. For example, if you wanted to set the time of day to 2:30 P.M., the command would be **TIME="14:30:00"**.

Examples:

```
TIME="20:40:15"         (* set time of day to 15 seconds after 8:40 P.M.)
TIME?                   (* report time of day)
```

***Related Registers*:** DATE, DAY, MONTH

TL Axis at Torque Limit

Class:	System Register	
Type:	Boolean	
Syntax:		
I*, *jr	TL	
🗎	TL*p1* (e.g., TL3 TLVI4)	
Parameters:	*allowed values*	*description*
🗎 *p1*	1 through 8 **or** VI*n*	axis number
Range:		
allowed values	0, 1	
Restrictions:	Servo only; read only.	
Use:	This register is used to determine whether the axis is at its torque limit. If the axis is at its torque limit, then TL is equal to 1; and when it is not at its torque limit, then TL is equal to 0.	
***Related Registers*:**	TLANY, TLC, TLE, SRA	

TLANY Any Axis at Torque Limit

Class:	System Register
Type:	Boolean
Syntax:	TLANY
Range:	
allowed values	0, 1
Restrictions:	Servo only; read only.
Use:	This register is used to determine whether any of the axes are at torque limit. If any of the axes are at torque limit, then TLANY is equal to 1; if none of the axes are at torque limit, then TLANY is equal to 0.
***Related Registers*:**	TL, TLC, TLE, SRS

TLC Torque Limit Current

Class:	Axis Register	
Type:	Floating point	
Syntax:		
I, jr	TLC	
🗎	TLC*p1* (e.g., TLC1 TLCVI3)	
Parameters:	*allowed values*	*description*
🗎 *p1*	1 through 8 **or** VI*n*	axis number
Range:		
units	%	
default	100.0	
minimum	0.1	
maximum	100.0	
Restrictions:	Servo only.	
Use:	This command loads the torque limit current as a percentage of the continuous current, CURC.	
Remarks:	The torque limit is enabled by the TLE command.	
Related Registers:	TLE, CURC	

TLE Torque Limit Enable

Class:	Axis Register	
Type:	Boolean	
Syntax:		
I, jr	TLE	
🗎	TLE*p1* (e.g., TLE1 TLEVI3)	
Parameters:	*allowed values*	*description*
🗎 *p1*	1 through 8 **or** VI*n*	axis number
Range:		
default	0	
allowed values	0, 1	
Restrictions:	Servo only.	
Use:	This command is used to enable the torque limit. If TLE is set to 1, then torque limit is enabled; and if TLE is set to 0, it is disabled.	
Registers Used:	TLC, TL	
Motion Templates:	Run reverse until torque limit; run forward until torque limit; run reverse at torque limit.	

TM Timer Timed Out Flag

Class:	System Register	
Type:	Boolean	
Syntax:	TM*p1* (e.g., TM1 TMVI3)	
Parameters:	*allowed values*	*description*
I, jr *p1*	1 through 8 **or** VI*n*	timer number
🖹 *p1*	1 through 16 **or** VI*n*	timer number
Range:		
allowed values	0, 1	
Restrictions:	Read only.	
Use:	This register is used to tell whether one of the timers timed out (i.e., was equal to 0). If TM*p1* is set to 1, then the timer timed out; if TM*p1* is set to 0, it did not time out. After the state of the timed out flag is read, the flag is set to zero until the timer times out again. It is then set to 1 and will stay at 1 until it is read again.	
Related Registers:	TMR, STM	

TMI Interval Timer

Class:	Input/Output Register	
Type:	Floating point	
Syntax:	TMI*p1.p2* (e.g., TMI1.3 TMI5.VI1 TMIVI2.1 TMIVI1.VI2)	
Parameters:	*allowed values*	*description*
p1	1 through 8 **or** VI*n*	digital module number
p2	1 through 4 **or** VI*n*	interval timer number
Range:		
units	seconds	
minimum	0.0000	
maximum	200,000.0000	
Restrictions:	Read only.	
Use:	The interval timer register is used to store the time between two successive activations of a digital input. Each of the interval timers takes its input from one of the first four inputs of a digital I/O module. For example, interval timer one takes its input from digital input one, interval timer two from digital input two, etc.	
Example:		
TMI5.VI1?	(* report interval timer VI1 of digital module five)	
Related Registers:	CTR, TMP	

TMP Pulse Timer

Class:	Input/Output Register	
Type:	Floating point	
Syntax:	TMP*p1.p2* (e.g., TMP1.3 TMP5.VI1 TMPVI2.1 TMPVI1.VI2)	
Parameters:	*allowed values*	*description*
p1	1 through 8 **or** VI*n*	digital module number
p2	1 through 4 **or** VI*n*	pulse timer number
Range:		
units	seconds	
maximum	0.0000	
maximum	200,000.0000	
Restrictions:	Read only.	
Use:	The pulse timer register is used to store the time during which a digital input stays active. Each of the pulse timers takes its input from one of the first four inputs of a digital I/O module. For example, pulse timer one takes its input from digital input one, pulse timer two from digital input two, etc.	
Example:		
TMP5.VI1?	(* report pulse timer VI1 of digital module five)	
Related Registers:	CTR, TMI	

TMR Timer

Class:	System Register	
Type:	Floating point	
Syntax:	TMR*p1* (e.g., TMR2 TMRVI3)	
Parameters:	*allowed values*	*description*
I, jr *p1*	1 through 8 **or** VI*n*	timer number
🖹 *p1*	1 through 16 **or** VI*n*	timer number
Range:		
units	seconds	
minimum	0.000	
maximum	2,000,000.000	
Restrictions:	Read only.	
Use:	The timer register is used to determine the current value of timer *p1*.	
Example:		
TMRVI2?	(* report timer VI2)	
Related Registers:	STM, TM	

TP Test Point Output

Class:	Axis Register	
Type:	Floating point	
Syntax:	TP*p1* (e.g., TP2 TPVI3)	
Parameters:	*allowed values*	*description*
p1	1 through 8 **or** VI*n*	axis number

Range:

units	volts
default	VLA
allowed values	-10.000 through 10.000 **or** VLA (axis velocity) CMD (control output) FE (following error)

Use: The test point output is an analog output that can be used either as a general purpose output or to output a test signal, which can be one of the following:

VLA (axis velocity)	10 volts = 20 Krpm
CMD (control output)	10 volts = maximum combined peak rating of assigned modules
FE (following error)	10 volts = 2,048 pulses of following error

Example:

```
TP1=VLA         (* set test point of axis one to axis velocity)
TPVI2=4         (* set test point of axis VI2 to 4 volts)
```

TVL Trajectory Velocity

Class:	Motion Register
Type:	Floating point
Syntax:	TVL
Range:	
units	trajectory units
default	.000001
minimum	.000001
maximum	16,000,000
Use:	This register is used to determine the trajectory velocity of a trajectory motion.
Remarks:	1. The trajectory units are determined by the axis units of the axes involved in the trajectory motion. 2. The trajectory velocity register is used only when trajectory motion is first started from a stop.

Example:

```
PSA1=0        (* set axis one position)
PSA2=0        (* set axis two position)
MPA1=3        (* set axis one absolute move position)
MPA2=5        (* set axis two absolute move position)
TVL=4         (* set trajectory velocity)
TFP=100       (* set trajectory feedrate to 100 percent
TFA=500       (* set trajectory feedrate acceleration)
RLA12         (* run linear trajectory motion)
```

What will happen:	Setting axis position, absolute move position, trajectory velocity, and trajectory feedrate acceleration and issuing the run linear motion command will cause axes one and two to move to 3 units and 5 units, running along a line at 4 units/second.
Related Registers:	TFA, TFP, TFD
Related Commands:	RLA

UNLOCK Unlocks Interpreter from Program

Class:	Program Command
Syntax:	UNLOCK
Restrictions:	Allowed only in programs.
Use:	This command unlocks the interpreter from the program, which lets other currently suspended programs execute concurrently.

Example:

```
        PROGRAM1        (* edit program 1)
        STM1=0.01       (* load start time of timer 1 and start timer 1)
1       WAIT TM1        (* wait for expression to be true)
        LOCK            (* lock interpreter to program)
        IF KEY GOTO2    (* conditionally goto 2)
        UNLOCK          (* unlock interpreter from program)
        GOTO1           (* unconditionally goto 1)
2       END             (* end program and exit editor)
```

What will happen: This program, once executed, will first wait for 10 ms. Then it locks the interpreter and checks for KEY to be true (i.e., for a character to be entered into the key buffer). If KEY is true, then the program goes to the statement at label 2, which ends the program. If KEY is not true, it unlocks the interpreter and goes to the statement at label 1, which waits for 10 ms, etc.

Related Commands: LOCK

UPS Update Screen

Class:	Input/Output Register
Type:	Integer
Syntax:	UPS
Range:	
default	0
minimum	0
maximum	50
Use:	This register is used to determine which screen is updated. The screen data, SCRD, for the screen specified in UPS is updated every 1/4 second.
Remarks:	This register is set to 0 upon power-up.
Registers Used:	SCRD, SCRP

URA Axis Unit Ratio

Class:	Axis Register	
Type:	Integer	
Syntax:		
I, jr	URA	
🖹	URA*p1* (e.g., URA1 URAVI4)	
Parameters:	*allowed values*	*description*
🖹 *p1*	1 through 8 **or** VI*n*	axis number
Range:		
units	pulses/axis unit	
default	1	
minimum	1	
maximum	1,000,000	

Restrictions: Not allowed in programs or motion blocks.

Use: The axis unit ratio scales the axis programming units from the default "pulses" to the desired engineering units. A similar register, URX, is used to scale the auxiliary encoder feedback. URA scales controller registers that represent axis position, velocity, acceleration or jerk.

Remarks: 1. This register can be set only after the memory has been cleared using the CLM command and before any programs or motion blocks are defined.
2. The numerical values for the default, minimum, and maximum of all registers with axis units are assuming that the axis unit ratio, URA, is set at its default value of 1. If the axis unit ratio is set to a value other than 1, the maximum and minimum values will be divided by the axis unit ratio. For example, if the maximum value of a register is 2,000,000,000 pulses and the axis unit ratio is set to 4,096, the new maximum of that parameter will be (2,000,000,000 pulses)/(4,096 pulses/axis unit) = 488,281.25 axis units.

Related Registers: URX, PLA, PSA, PZA, OFA, VLA

URX Auxiliary Unit Ratio

Class:	Axis Register	
Type:	Integer	
Syntax:		
I, jr	URX	
🖹	URX*p1* (e.g., URX1 URXVI4)	
Parameters:	*allowed values*	*description*
🖹 *p1*	1 through 8 **or** VI*n*	axis number
Range:		
units	auxiliary encoder pulses/auxiliary unit	
default	1	
minimum	1	
maximum	1,000,000	

Restrictions: For IMCs, this function available only with the extended command set; not allowed in programs or motion blocks.

Use: The auxiliary unit ratio is used to define auxiliary units (engineering units for the auxiliary encoder input) for the PLX, PSX, PZX, OFX and VLX registers.

Remarks:
1. This register can be set only after the memory has been cleared using the CLM command and before any programs or motion blocks are defined.
2. The numerical values for the default, minimum, and maximum of all registers with auxiliary units are assuming that the auxiliary unit ratio, URX, is set at its default value of 1. If the auxiliary unit ratio is set to a value other than 1, the maximum and minimum values will be divided by the auxiliary unit ratio. For example, if the maximum value of a register is 2,000,000,000 pulses and the auxiliary unit ratio is set to 4,096, the new maximum of that parameter will be (2,000,000,000 pulses)/(4,096 pulses/auxiliary unit) = 488,281.25 auxiliary units.

Related Registers: PLX, PSX, PZX, OFX, VLX, URA

VB Boolean Variable

Class:	Variable Register	
Type:	Boolean	
Syntax:	VB*p1* (e.g., VB1 VBVI2)	
Parameters:	*allowed values*	*description*
p1	1 through 256 **or** VI*n*	Boolean variable number
Range:		
default	0	
allowed values	0, 1	

Use: Boolean variables are used mainly in conditional statements of programs, such as IF...GOTO (conditional goto) and WAIT (wait for expression to be true). They can also be used to load register values.

Example:

VB1=VI1>0	(* set Boolean variable one to 1 if integer variable one is greater than zero)
VB3=VB1 AND VB2	(* set Boolean variable three to 1 if both Boolean variable one and Boolean variable two are set)
VBVI2=VI1<5	(* set Boolean variable VI2 to 1 if integer variable one is less than five)
VBVI2?	(* report Boolean variable VI2)

VF Floating Point Variable

Class:	Variable Register	
Type:	Floating point	
Syntax:	VF*p1* (e.g., VF1 VFVI2)	
Parameters:	*allowed values*	*description*
p1	1 through 2,048 **or** VI*n*	floating point variable number
Range:		
default	0.0	
minimum	1.5 x 10-^{39} (absolute value)	
maximum	1.7 x 10^{38} (absolute value)	
Use:	Floating point variables are used in variable expressions and to load register values.	
Remarks:	1. The numerical value for the maximum of parameter *p1* shown above is assuming that the floating point variable allocation, VFA, is set to 2,048. If VFA is set to a value other than 2,048, the maximum of *p1* will change. 2. To access the extended floating point variables, use the indirect addressing scheme (i.e., VFVI*n*).	

Example:

VF1=5.776	(* set floating point variable one to 5.776)
VI1=2000	(* set integer variable to 2,000)
VFVI1=SQR(2.*VF1)	(* set floating point variable VI1 [i.e., 2,000] to square root of 2 times 5.776)
VF2=PSA/5.	(* set floating point variable two to axis position divided by 5)
VFVI1?	(* report floating point variable VI1)

Related Registers: VFA, VFEA

VFA Floating Point Variable Allocation

Class:	System Register
Type:	Integer
Syntax:	VFA
Range:	
default	1,024
minimum	0
maximum	2,048
Restrictions:	Cannot be assigned in programs or motion blocks.
Use:	The floating point variable allocation register is used to define how many floating point variables can reside in memory.
Remarks:	1. This register can be set only after the memory has been cleared using the CLM command and before any programs or motion blocks are defined. 2. Setting the register will overwrite part of the memory space normally allocated for integer variables. One floating point variable will take over the space that two integer variables previously occupied. For example, if VFA is set to 200, the integer variables will range from 1 to 3,696 [4,096 - (2*200)].
Related Registers:	VFEA, VF, VI

VFEA Floating Point Variable Extended Allocation *I* 🖹

Class:	System Register
Type:	Integer
Syntax:	VFEA
Range:	
default	2,048
minimum	2,048
I *maximum*	14,336
🖹 *maximum*	131,072
Restrictions:	Cannot be assigned in programs or motion blocks.
Use:	The floating point variable extended allocation register is used to define how many floating point variables can reside in extended memory.
Remarks:	1. This register can be set only after the memory has been cleared using the CLM command and before any programs or motion blocks are defined. 2. Setting the register overwrites part of the memory space normally allocated for integer variables. One floating point variable will take over the space that two integer variables previously occupied. For example, if VFEA is set to 4,000, the extended integer variables will range from 4,097 to 258,240 (262,144 - 2*(4,000 - 2,048)); **or** from 4,097 to 10,432 (14,336 - 2*(4,000 - 2,048)).
Related Registers:	VFA, VF, VI

VI Integer Variable

Class:	Variable Register	
Type:	Integer, Boolean	
Syntax:	VI*p1.p2* (e.g., VI1 VIVI2 VI1.5 VI1.VI3 VIVI2.VI6)	
Parameters:	*allowed values*	*description*
p1	1 through 4,096 **or** VI*n*	integer variable number
p2	none **or** 0 through 31 **or** VI*n*	integer variable bit number
Range:		
default	0	
minimum	-2,147,483,648	
maximum	2,147,483,647	
Use:	Integer variables are used in variable expressions and to load register values.	
Remarks:	1. The numerical value for the maximum of parameter *p1* shown above is assuming that the floating point variable allocation, VFA, is set to 0. If VFA is set to a value other than 0, the maximum of *p1* will change. 2. To access the extended integer variables, the indirect addressing scheme (i.e., VIVI*n*) must be used.	

Examples:

VI1=3000	(* set integer variable one to 3,000)
VI2=-330	(* set integer variable two to -330)
VIVI1=VI1+VI2	(* set integer variable VI1 [i.e., integer variable 3,000] to 3,000 plus -330)
VI3=PSR*2	(* set integer variable three to PSR times 2 [i.e., resolver position times 2])
VI2?	(* report integer variable two)
VI1.4=1	(* set bit four of integer variable one)
VI5.17=0	(* clear bit 17 of integer variable five)
VI2.3=VI4.2 OR VI5.7	(* set bit three of VI2 if bit two of VI4 or bit seven of VI5 is set)

Related Registers: VFA, VFEA

VLA Axis Velocity

Class:	Axis Register	
Type:	Floating point	
Syntax:		
I, jr	VLA	
🗎	VLA*p1* (e.g., VLA1 VLAVI3)	
Parameters:	*allowed values*	*description*
🗎 *p1*	1 through 8 **or** VI*n*	axis number
Range:		
units	axis units/sec	
minimum	-16,000,000 pulses/sec	
maximum	16,000,000 pulses/sec	
Restrictions:	Read only.	
Use:	This register is used to determine the current velocity of the axis.	
Remarks:	The numerical values for the default, minimum, and maximum of this register are assuming that the axis unit ratio, URA, is set at its default value of 1. If the axis unit ratio is set to a value other than 1, the default, maximum, and minimum values will change appropriately (see URA).	
Related Registers:	URA, VLAT	

VLAT Axis Velocity Filter Time Constant

Class:	Axis Register		
Type:	Floating point		
Syntax:			
I, jr	VLAT		
🖹	VLAT*p1* (e.g., VLAT1, VLATVI3)		
Parameters:	*allowed values*	*description*	
🖹 *p1*	1 through 8 **or** VI*n*	axis number	
Range:			
units	seconds		
default	0.01		
minimum	0.002		
maximum	0.1		

Use: The axis velocity filter time constant represents the length of the time window that is used to filter the axis velocity, VLA. This time window is applied to previous values of VLA in order to calculate the current filtered value of VLA. This happens every 2 msec.

Remarks: VLAT can be set only in 2 msec increments (i.e., 0.002, 0.004, 0.006, ...). This corresponds to the number of previous values of VLA that are being filtered. For example, setting VLAT = 0.01 means that the previous 5 values of VLA will be filtered, since 0.002*5 = 0.01.

Example:

IMC/IMJ	**Target**	
VLAT=0.008	VLAT1=0.008	(* set axis velocity filter time (* constant to 0.008 sec)
VLAT?	VLAT1?	(* report value of axis velocity (* filter time constant)

Related Registers: VLA, VLXT

VLT Trajectory Velocity

Class:	Axis Register
Type:	Floating point
Syntax:	VLT
Range:	
units	trajectory units/sec
minimum	-16,000,000 units/sec
maximum	16,000,000 units/sec
Restrictions:	Read only.
Use:	This register is used to determine the current trajectory velocity of a trajectory move.

VLX Auxiliary Velocity

Class:	Axis Register	
Type:	Floating point	
Syntax:		
I, jr	VLX	
🖹	VLX*p1* (e.g., VLX1 VLXVI3)	
Parameters:	*allowed values*	*description*
🖹 *p1*	1 through 8 **or** VI*n*	axis number
Range:		
units	auxiliary units/sec	
minimum	-16,000,000 pulses/sec	
maximum	16,000,000 pulses/sec	

Restrictions: For IMCs, this function available only with the extended command set; read only.

Use: This register is used to determine the current auxiliary velocity of the axis.

Remarks: The numerical values for the default, minimum, and maximum of this register are assuming that the auxiliary unit ratio, URX, is set at its default value of 1. If the auxiliary unit ratio is set to a value other than 1, the default, minimum, and maximum values will change appropriately (see URX).

Related Registers: URX, VLXT

VLXT Auxiliary Velocity Filter Time Constant

Class:	Axis Register	
Type:	Floating point	
Syntax:		
I, jr	VLXT	
🖹	VLXT*p1* (e.g., VLXT1 VLXTVI3)	
Parameters:	*allowed values*	*description*
🖹 *p1*	1 through 8 **or** VI*n*	axis number
Range:		
units	seconds	
default	0.01	
minimum	0.002	
maximum	0.1	

Restrictions: For IMCs, this function available only with the extended command set.

Use: The auxiliary velocity filter time constant is used to represent the length of the time window that is used to filter the auxiliary velocity, VLX. This time window is applied to previous values of VLX in order to calculate the current filtered value of VLX. This happens every 2 msec.

Remarks: VLXT can be set only in 2 msec increments (i.e., 0.002, 0.004, 0.006, ...) This corresponds to the number of previous values of VLX that are being filtered. For example, setting VLXT = 0.01 means that the previous 5 values of VLX will be filtered, since 0.002*5 = 0.01.

Example:

IMC/IMJ	**Target**	
VLXT=0.008	VLXT1=0.008	(* set auxiliary velocity filter)
VLXT?	VLXT1?	(* report value of auxiliary
		(* velocity filter time constant)

Related Registers: VLX, VLAT

VS String Variable

Class:	Variable Register	
Type:	String	
Syntax:	VS*p1* (e.g., VS1 VSVI2)	
Parameters:	*allowed values*	*description*
p1	1 through 144 **or** VI*n*	string variable number
Range:		
default	“”	
allowed values	any string, 0 through 127 characters long, enclosed in quotes	
Use:	String variables are used mainly to load strings and in input/output commands such as GET, PUT, IN, and OUT as a means of user interface.	
Remarks:	If the extended memory card is available, then *p1* can be up to 272 for a Target.	
Examples:		
VS1=“$20”+“$R”	(* set string variable one to a space followed by a carriage return)	
VI1=2	(* set integer variable one to 2)	
VSVI1=“Done”+VS1	(* set string variable VI1 [i.e., string variable two] to “Done” followed by a space and a carriage return)	
VSVI2?	(* report string variable VI2)	
Related Commands:	EXVS, GET, PUT, IN, OUT	

WAIT Waits for Expression to be True

Class:	Program Command	
Syntax:	WAIT*p1* (e.g., WAIT VB1 WAIT KEY)	
Parameters:	*allowed values*	*description*
p1	any Boolean expression	Boolean expression

Restrictions: Allowed only in programs or motion blocks.

Use: This command causes the program or motion block to wait for Boolean expression *p1* to be true (i.e., evaluate to 1). Once *p1* is true, the next program or motion block statement will be executed.

Example:

```
IMC/IMJ                    Target
PROGRAM1                   PROGRAM1     (* edit program 1)
PSA=0                      PSA1=0       (* set axis position register)
MVL=10                     MVL1=10      (* set motion velocity)
MAC=40                     MAC1=40      (* set motion acceleration)
MPA=0                      MPA1=0       (* set absolute move position)
MPI=10                     MPI1=10      (* set incremental move position)
RPI                        RPI1         (* run to incremental move position)
WAIT IP                    WAIT IP1     (* wait for expression to be true)
STM1=1                     STM1=1       (* set start time of timer 1)
WAIT TM1                   WAIT TM1     (* wait for expression to be true)
RPA                        RPA1         (* run to absolute move position)
WAIT IP                    WAIT IP1     (* wait for expression to be true)
OUT "Motion completed"     OUTW "Motion completed"
                                        (* output string expression to display)
END                        END          (* end program 1 and exit editor)
```

What will happen: This program sets the axis position register, velocity, acceleration, absolute move position, and incremental move position. Then it issues the RPI command, which runs the axis 10 units in the forward direction. It then waits until the axis is in position. Next, it loads timer 1 with a start time of 1 second. The timer will then count down from 1 second to 0. Once it reaches 0, the RPA command is issued, which runs the axis 10 units in the reverse direction. It waits until the axis is in position, and then it prints *Motion completed* to the display.

Related Commands: WAIT...WHEN...GOTO

WAIT…WHEN…GOTO

Class: Program Command

Syntax: WAIT *p1* WHEN *p2* GOTO*p3*

Parameters:

	allowed values	*description*
p1	any Boolean expression	Boolean expression
p2	any Boolean expression	Boolean expression
p3	1 through 999 **or** VI*n*	label number

Restrictions: Allowed only in programs.

Use: This statement causes the program either to wait for *p1* to become true (i.e., evaluate to 1) or to go conditionally to label *p3* if *p2* is true (i.e., evaluates to 1).

Examples:

```
IMC/IMJ                     Target
PROGRAM1                    PROGRAM1   (* edit program 1)
MVL=5                       MVL1=5     (* set motion velocity)
MAC=40                      MAC1=40    (* set motion acceleration)
MPI=25                      MPI1=25    (* set incremental move position)
RPI                         RPI1       (* run to incremental move position)
WAIT IP WHEN KEY GOTO5      WAIT IP1 WHEN KEYW GOTO5
                                       (* wait for expression to be true or
                                       (* when condition becomes true goto 5)
OUT "Motion complete$N"     OUTW "Motion complete$N"
                                       (* output string expression to display)
GOTO10                      GOTO10     (* unconditionally goto 10)
5 ST                        5 ST1      (* stop axis)
WAIT IP                     WAIT IP1   (* wait for expression to be true)
OUT "Motion interrupted$N"  OUTW "Motion interrupted$N"
                                       (* output string expression to display)
10 END                      10 END     (* end program 1 and exit editor)
```

What will happen: This program, once executed, sets the velocity, acceleration, and incremental move position. It then issues the RPI command, which runs the axis 25 units in the forward direction. If a character goes into the key buffer (KEY) before the axis is in position (IP), the program execution will go to the statement at label 5, which stops the axis. It then prints *Motion interrupted* to the display and ends. If a character does not go into the key buffer before the axis is in position, the program will continue to the next statement, which prints *Motion complete*. It then goes to the statement at label 10, which ends the program.

Related Commands: WAIT

WKY Puts Character into Key Buffer

Class:	Input/Output Command	
Syntax:	WKY*p1* (e.g., WKY1 WKYB)	
Parameters:	*allowed values*	*description*
p1	any ASCII character	ASCII character
Restrictions:	Not allowed in motion blocks.	
Use:	This command puts one character into the key buffer.	

Example:

```
WKYE        (* put "E" into key buffer)
WKY1        (* put "1" into key buffer)
```

Related Commands: GETW, INW

X Steps through Program/Motion Block

Class:	Program Command	
Syntax:	X*p1* (e.g., X X3 X10)	
Parameters:	*allowed values*	*description*
p1	1 through 65,000	step size

Use: This command steps *p1* lines through a program or motion block while in the line editor; **or** it steps through the execution of a program if not in the line editor and single step mode is enabled (i.e., DGE is set to 1 and DGS is set to the program you wish to step through [see DGS]).

Remarks: Note that *p1* is optional. If *p1* is not specified, a value of 1 will be assumed.

Examples:

```
IMC/IMJ      Target
PROGRAM1     PROGRAM1     (* edit program 1)
*  MVL=5     *  MVL1=5
X            X            (* step through program)
*  MAC=40    *  MAC1=40
X            X            (* step through program)
*  MPI=25    *  MPI1=25
!            !            (* exit line editor)
*            *
```

Related Commands: DGE, DGS, PROGRAM, MOTION, L, LABEL, !

Appendix B Operators

This appendix provides details on the following types of operators:

- Relational

>	greater than operator
>=	greater than or equal to operator
=	equal to operator
<>	not equal to operator
<=	less than or equal to operator
<	less than operator

- Logical

NOT	not operator
AND	and operator
OR	or operator
XOR	exclusive or operator
ROL	rotate left operator
ROR	rotate right operator
SHL	arithmetic shift left operator
SHR	arithmetic shift right operator

- Arithmetic

+	add operator
-	subtract operator
*	multiply operator
/	divide operator
**	exponentiate operator

- Math

ABS	absolute value operator
CRC	cyclical redundancy check calculation operator
EXP	exponential operator
LGN	natural log operator
SQR	square root operator

- Trigonometric

SIN	sine of angle in degrees operator
COS	cosine of angle in degrees operator
TAN	tangent of angle in degrees operator
ATN	arctangent to angle in degrees operator

- String

+	concatenation operator
LEN	length of string operator
LFT	leftmost characters of string operator
MID	middle characters of string operator
RGT	rightmost characters of string operator
FIN	find string in string operator
INS	insert characters into string
DEL	delete characters from string
LWR	convert string to lower case operator
UPR	convert string to upper case operator
ASC	convert from character to ASCII code operator
CHR	convert from ASCII code to character operator

- Convert to Floating Point

ITF	convert integer to floating point operator
STF	convert string to floating point operator

- Convert to Integer

DTI	convert date to integer operator
TTI	convert time to integer operator
FTI	convert floating point to integer operator
TRC	truncate floating point to integer operator
STI	convert string to integer operator

- Convert to String

FTS	convert floating point to string operator
ITB	convert integer to binary string operator
ITH	convert integer to hexadecimal string operator
ITS	convert integer to string operator

- Convert to Time/Date

ITD	convert integer to date operator
ITT	convert integer to time operator

>, >=, =, <>, <=, < Relational Operators

Type: Boolean

Syntax: p1 > p2, p1 >= p2, p1 = p2, p1 <> p2, p1 <= p2, p1 < p2

Parameters: *allowed values*

p1	any integer, floating point, or string operand
p2	any integer, floating point, or string operand

Use: These operators are used to compare the two operands *p1* and *p2*. Note that *p1* and *p2* must be of the same type. If the relation is false, its value is 0; and if the relation is true, its value is 1. A *relation* is two operands with a relational operator between them. The operators are described below:

p1 > p2	*p1* greater than *p2*
p1 >= p2	*p1* greater than or equal to *p2*
p1 = p2	*p1* equal to *p2*
p1 <> p2	*p1* not equal to *p2*
p1 <= p2	*p1* less than or equal to *p2*
p1 < p2	*p1* less than *p2*

Remarks: Note that for string operands, the relational operators compare the two strings character by character. The ASCII values of each character are compared one by one from left to right.

Example:

```
VF1=12.5            (* set floating point variable 1 to 12.5)
VF2=12.0            (* set floating point variable 2 to 12.0)
VB1= VF1<=VF2       (* set boolean variable 1 to VF1<=VF2)
VB1?                (* report value of boolean variable 1)
*  0
VS1="Hello"         (* set string variable 1 to "Hello")
VS2="AB"            (* set string variable 2 to "AB")
VS3="AC"            (* set string variable 3 to "AC")
VS4="ABC"           (* set string variable 4 to "ABC")
VB1= VS1<>VS2       (* set boolean variable 1 to VS1<>VS2)
VB1?                (* report value of boolean variable 1)
*  1
VB1= VS2<VS3        (* set boolean variable 1 to VS2<VS3)
VB1?                (* report value of boolean variable 1)
*  1
VB1=VS2>VS4         (* set boolean variable 1 to VS2>VS4)
VB1?                (* report value of boolean variable 1)
*  0
```

NOT, AND, OR, XOR   Logical Operators

Type: Boolean, integer

Syntax: NOT *p2*   *p1* AND *p2*   *p1* OR *p2*   *p1* XOR *p2*

Parameters:	*allowed values*
p1	any boolean or integer operand
p2	any boolean or integer operand

Use: These operators are used to perform logical operations on *p1* and *p2*. Note that *p1* and *p2* must be of the same type. If *p1* and *p2* are integer operands, the logical operators perform bitwise logical operations. The operations are described below:

NOT *p2*	not operator
p1 AND *p2*	and operator
p1 OR *p2*	or operator
p1 XOR *p2*	exclusive or operator

ROL, ROR Rotate Operators

Type:	Integer
Syntax:	*p1* ROL *p2* *p1* ROR *p2*
Parameters:	*allowed values*
p1	any integer operand
p2	any integer operand
Use:	These operators are used to rotate the bits of *p1* by the number of places specified by *p2*.

Example:

```
VI1=2#11101001          (* set integer variable 1 to 2#11101001)
VI2=VI1 ROL 2           (* set integer variable 2 to VI1 rotated left by 2 places)
VS1=ITB(VI2,12)         (* set string variable 1 to VI2 converted to binary string)
VS1?                    (* report value of string variable 1)
*"2#1110100100"
VI3=VI1 ROR 3           (* set integer variable 3 to VI1 rotated right by 3 places)
VS2=ITB(VI3,32)         (* set string variable 2 to VI3 converted to binary string)
VS2?                    (* report value of string variable 2)
*"2#100000000000000000000000011101"
```

Related Operators: SHL, SHR

SHL, SHR Arithmetic Shift Operators

Type:	Integer
Syntax:	*p1* SHL *p2* *p1* SHR *p2*
Parameters:	*allowed values*
p1	any integer operand
p2	any integer operand
Use:	These operators are used to perform an arithmetic shift of *p1* by the number of places specified by *p2*.

Example:

```
VI1=2#11101001       (* set integer variable 1 to 2#01101001)
VI2=VI1 SHL 3        (* set integer variable 2 to VI1 shifted left by 3 places)
VS1=ITB(VI2,13)      (* set string variable 1 to VI2 converted to binary string)
VS1?                 (* report value of string variable 1)
*"2#11101001000"
VI3=VI1 SHR 2        (* set integer variable 3 to VI1 shifted right by 2 places)
VS2=ITB(VI3,8)       (* set string variable 2 to VI3 converted to binary string)
VS2?                 (* report value of string variable 2)
*"2#111010"
```

Related Operators: ROL, ROR

+, -, *, /, ** Arithmetic Operators

Type:	Floating point, integer
Syntax:	*p1 + p2, p1 - p2, - p1, p1 * p2, p1 / p2, p1 ** p2*
Parameters:	*allowed values*
p1	any integer or floating point operand
p2	any integer or floating point operand

Use: These operators are used to perform arithmetic operations on *p1* and *p2*. Note that *p1* and *p2* must be of the same type. The operations are described below:

p1 + p2	add
p1 - p2	subtract
- p1	negate
*p1 * p2*	multiply
p1 / p2	divide
*p1 ** p2*	exponentiate (i.e., raise *p1* to the *p2* power)

ABS Absolute Value Operator

Type:	Floating point, integer
Syntax:	ABS(*p1*)
Parameters:	*allowed values*
p1	any integer or floating point operand
Use:	This operator is used to take the absolute value of *p1*.

CRC Cyclical Redundancy Check Calculation Operator

Type:	Integer
Syntax:	CRC(*p1*)
Parameters:	*allowed values*
p1	any integer operand
Use:	This operator is used to do a 16-bit cyclical redundancy check calculation on 8-bit data.

Example:

```
VI1=65535              (* initialize CRC register)
VI1=CRC(VI1 XOR 23)    (* calculate CRC of  23)
VI1=CRC(VI1 XOR 215)   (* calculate CRC of 23 and 215)
```

EXP Exponential Operator

Type:	Floating point
Syntax:	EXP(*p1*)
Parameters:	*allowed values*
p1	any floating point operand
Use:	This operator is used to take the exponential of *p1* (i.e., raise the number e to the power *p1)*.

LGN Natural Log Operator

Type:	Floating point
Syntax:	LGN(*p1*)
Parameters:	*allowed values*
p1	any positive floating point operand
Use:	This operator is used to take the natural log of *p1* (i.e., the logarithm base e of *p1*).

SQR Square Root Operator

Type:	Floating point, integer
Syntax:	SQR(*p1*)
Parameters:	*allowed values*
p1	any positive integer or floating point operand
Use:	This operator is used to take the square root of *p1*.

SIN, COS, TAN, ATN Trigonometric Function Operators

Type:	Floating point
Syntax:	SIN(*p1*) COS(*p1*) TAN(*p1*) ATN(*p1*)
Parameters:	*allowed values*
p1	any floating point operand
Use:	These operators are used to perform trigonometric functions on *p1*. When using SIN, COS, or TAN, *p1* must be in degrees. The operations are described below:

SIN(*p1*)	sine of angle in degrees
COS(*p1*)	cosine of angle in degrees
TAN(*p1*)	tangent of angle in degrees
ATN(*p1*)	arctangent to angle in degrees

+ Concatenation Operator

Type:	String
Syntax:	*p1 + p2*
Parameters:	*allowed values*
p1	any string operand
p2	any string operand
Use:	This operator is used to concatenate strings *p1* and *p2*.

Example:

```
VS1="Hello"            (* set string variable 1 to "Hello")
VS2=VS1+ " There"      (* set string variable 2 to the concatenation of VS1 and "
                         There")
VS2?                   (* report value of string variable 2)
*"Hello There"
```

LEN Length Of String Operator

Type:	Integer
Syntax:	LEN(*p1*)
Parameters:	*allowed values*
p1	any string operand
Use:	This operator is used to compute the length of the string in *p1*.

Example:

```
VI1=LEN(“Hello”)        (* set integer variable 1 to length of string “Hello”)
VI1?                    (* report value of integer variable 1)
*5
```

LFT, MID, RGT — Select Characters Of String Operators

Type: String

Syntax: LFT(*p1*,*p2*) MID(*p1*,*p2*,*p3*) RGT(*p1*,*p2*)

Parameters:

	allowed values	*description*
p1	any string operand	string
p2	any integer operand >= 0	number of characters
p3	any integer operand >= 1	location of characters

Use: These operators are used to select characters of string *p1*. The operations are described below:

LFT	leftmost characters of string—takes the leftmost *p2* characters of string operand *p1*.
MID	middle characters of string—takes the middle *p2* characters of string operand *p1* from character number *p3*.
RGT	rightmost characters of string—takes the rightmost *p2* characters of string operand *p1*.

Example:

```
VS1="Jogging axis forward"
                       (* set string variable 1 to "Jogging axis forward")
VS2=LFT(VS1,7)         (* set string variable 2 to leftmost 7 characters of VS1)
VS2?                   (* report value of string variable 2)
*"Jogging"
VS3=MID(VS1,4,9)       (* set string variable 3 to the middle 4 characters of VS1 from
                       character 9)
VS3?                   (* report value of string variable 3)
*"axis"
VS4=RGT(VS1,7)         (* set string variable 4 to leftmost 7 characters of VS1)
VS4?                   (* report value of string variable 4)
*"forward"
```

FIN Find String In String Operator

Type:	Integer
Syntax:	FIN(*p1*,*p2*)
Parameters:	*allowed values*
p1	any string operand
p2	any string operand
Use:	This operator is used to find string *p2* in string *p1*. If *p2* is found in *p1*, the value returned is the location of the first character of *p2* in the string *p1*. If *p2* is not in *p1*, the value returned is 0.

Example:

VS1="Jogging"	* set string variable 1 to "Jogging")
VI1=FIN(VS1,"Jog")	(* set integer variable 1 to location of first character of "Jog" in VS1)
VI1?	(* report value of integer variable 1)
*1	
VI2=FIN(VS1,"in")	(* set integer variable 2 to location of first character of "in" in VS1)
VI2?	(* report value of integer variable 2)
*5	
VI3=FIN(VS1,"Hello")	(* set integer variable 3 to location of first character of "Hello" in VS1)
VI3?	(* report value of integer variable 3)
*0	

INS, DEL Edit String Operators

Type: String

Syntax: INS(*p1*,*p2*,*p4*) DEL(*p1*,*p3*,*p4*)

Parameters:

	allowed values	*description*
p1	any string operand	string to be edited
p2	any string operand	string to be inserted
p3	any integer operand >= 0	number of characters to delete
p4	any integer operand >= 1	location in *p1* to insert/delete

Use: These operators are used to edit string operand *p1*. The operations are described below:

INS	insert characters into string—inserts string operand *p2* into string operand *p1* at location *p4*.
DEL	delete characters from string—deletes *p3* characters starting at location *p4* of string operand *p1*.

Example:

```
VS1="Drill operation"     (* set string variable 1 to "Drill operation")
VS2=INS(VS1,"in ",7)      (* set string variable 2 to SV1 with "in " inserted at location 7)
VS2?                      (* report value of string variable 2)
*"Drill in operation"
VS3=DEL(VS2,3,7)          (* set string variable 3 to SV2 with 3 characters deleted starting
                            at location 7)
VS3?                      (* report value of string variable 3)
*"Drill operation"
```

LWR, UPR Case Conversion Operators

Type:	String
Syntax:	LWR(*p1*) UPR(*p1*)
Parameters:	*allowed values*
p1	any string operand
Use:	These operators are used to convert string operand *p1* to either lower (LWR) or upper (UPR) case.

Example:

```
VS1="Hello"          (* set string variable 1 to "Hello")
VS2=UPR(VS1)         (* set string variable 2 to upper case of VS1)
VS2?                 (* report value of string variable 2)
*"HELLO"
VS3=LWR(VS1)         (* set string variable 3 to lower case of VS1)
VS3?                 (* report value of string variable 3)
*"hello"
```

ASC Convert from Character to ASCII Operator

Type:	Integer
Syntax:	ASC(*p1*)
Parameters:	*allowed values*
p1	any string operand
Use:	This operator is used to convert the first character in string operand *p1* to the ASCII code that represents this character.

Example:

```
VI1=ASC("Hello")     (* set integer variable 1 to the ASCII code of the first character
                       of "Hello")
VI1?                 (* report value of integer variable 1)
*72                  (* note that 72 is the ASCII code for "H")
```

Related Operators: CHR

CHR Convert from ASCII Code to Character Operator

Type:	String
Syntax:	CHR(*p1*)
Parameters:	*allowed values*
p1	any integer operand
Use:	This operator is used to convert the ASCII code *p1* to the character represented by ASCII code *p1*.

Example:

VS1=CHR(65)	(* set string variable 1 to the character represented by ASCII code 65)
VS1?	(* report value of string variable 1)
*"A"	

Related Operators: ASC

ITF Convert Integer to Floating Point Operator

Type:	Floating point
Syntax:	ITF(*p1*)
Parameters:	*allowed values*
p1	any integer operand
Use:	This operator is used to convert integer operand *p1* to a floating point number.

Example:

```
VI1=12          (* set integer variable 1 to 12)
VF1=ITF(VI1)    (* set floating point variable 1 to VI1 converted to floating
                  point)
VF1?            (* report value of floating point variable 1)
*12.
```

STF Convert String to Floating Point Operator

Type:	Floating point
Syntax:	STF(*p1*)
Parameters:	*allowed values*
p1	any string operand
Use:	This operator is used to convert the string operand *p1* to a floating point number.
Remarks:	1. If the string operand contains an invalid digit for a floating point number, then this operator will return a result of zero and set the "Invalid digit in string" bit (i.e., bit 4) of the program status register.
	2. If the converted value is too large to be represented by a floating point number, then this operator will return a result of zero and set the "String value out of range" bit (i.e., bit 5) of the program status register.

Example:

```
VS1="12.95"       (* set string variable 1 to "12.95")
VF1=STF(VS1)      (* set floating point variable 1 to VS1 converted to floating
                    point)
VF1?              (* report value of floating point variable 1)
*12.95
```

DTI, TTI Convert Time/Date to Integer Operators

Type:	Integer
Syntax:	TTI(*p1*) DTI(*p1*)
Parameters:	*allowed values*
p1	any string operand
Use:	These operators are used to convert a time or date string *p1* to integer. The operations are described below:
	TTI convert time to integer—converts 24-hour format time string to number of seconds.
	DTI convert date to integer—converts date string to number of seconds from January 1, 1994.
Remarks:	If the string operand is not a valid representation of a time or date, then these operators will return a result of zero and set the "Invalid time/date" bit (i.e., bit 7) of the program status register.

Example:

```
VS1="09:30:30"        (* set string variable 1 to "09:30:30")
VI1=TTI(VS1)          (* set integer variable 1 to VS1 converted to seconds)
VI1?                  (* report value of integer variable 1)
*34230
VS2="1996-02-01"      (* set string variable 2 to "1996-02-01")
VI2=DTI(VS2)          (* set integer variable 2 to VS2 converted to seconds)
VI2?                  (* report value of integer variable 2)
*65750400
```

FTI, TRC Convert Floating Point to Integer Operators

Type: Integer

Syntax: FTI(*p1*) TRC(*p1*)

Parameters: *allowed values*

p1 any floating point operand

Use: These operators are used to convert floating point operand *p1* to an integer:

FTI	Convert floating point to integer—rounds *p1* to the nearest integer.
TRC	Truncate floating point to integer—truncates *p1*.

Remarks: If the floating point number is too large to be represented by an integer, then these operators will return a result of zero and set the *Floating point value out of range* bit (i.e., bit 6) of the program status register.

Example:

```
VF1=12.9505        (* set floating point variable 1 to 12.9505)
VI1=FTI(VF1)       (* set integer variable 1 to VF1 converted to integer by
                     rounding)
VI1?               (* report value of integer variable 1)
*13
VI2=TRC(VF1)       (* set integer variable 2 to VF1 converted to integer by
                     truncation)
VI2?
*12
```

STI Convert String to Integer Operator

Type:	Integer
Syntax:	STI(*p1*)
Parameters:	*allowed values*
p1	any string operand
Use:	This operator is used to convert the string operand *p1* to an integer.
Remarks:	1. If the string operand contains an invalid digit for an integer, then this operator will return a result of zero and set the “Invalid digit in string” bit (i.e., bit 4) of the program status register.
	2. If the converted value is too large to be represented by an integer, then this operator will return a result of zero and set the “String value out of range” bit (i.e., bit 5) of the program status register.

Example:

```
VS1="1204"        (* set string variable 1 to "1204")
VI1=STI(VS1)      (* set integer variable 1 to VS1 converted to integer)
VI1?              (* report value of integer variable 1)
*1024
```

FTS Convert Floating Point to String Operator

Type: String

Syntax: FTS(*p1,p2,p3*)

Parameters:	*allowed values*	*description*
p1	any floating point operand	floating point operand
p2	any integer operand in	field width range 0 through 40
p3	any integer operand in	number of decimal places range 0 through 10

Use: This operator is used to convert floating point operand *p1* to a string with field width *p2* and *p3* decimal places.

Remarks:

1. If the floating point number cannot be contained in the field width specified, then the result is a string of asterisks of length equal to the field width.

2. If the field width is set to 0, then the result is the string representation of the floating point number, which has the minimum field width.

Example:

```
VF1=12.9505          (* set floating point variable 1 to 12.9505)
VS1=FTS(VF1,6,2)     (* set string variable 1 to VF1 converted to string with field
                       width 6 and 2 decimal places)
VS1?                 (* report value of string variable 1)
*"  12.95"
```

ITB, ITH, ITS Convert Integer to String Operators

Type: String

Syntax: ITS(*p1*,*p2*) ITB(*p1*,*p2*) ITH(*p1*,*p2*)

Parameters:	*allowed values*	*description*
p1	any integer operand	integer
p2	any integer operand in	field width range 0 through 40

Use: These operators are used to convert the integer operand *p1* to a string. The operations are described below:

ITB	convert integer to binary string—converts *p1* to a binary string.
ITH	convert integer to hex string—converts *p1* to a hexadecimal string.
ITS	convert integer to string—converts *p1* to a string.

Remarks:

1. If the integer cannot be contained in the field width specified, then the result is a string of asterisks of length equal to the field width.

2. If the field width is set to 0, then the result is the string representation of the integer, which has the minimum field width.

Example:

```
VI1=2282              (* set integer variable 1 to 2282)
VS1=ITS(VI1,6)        (* set string variable 1 to VI1 converted to string with field
                        width of 6)
VS1?                  (* report value of string variable 1)
*"  2282"
VS2=ITB(VI1,14)       (* set string variable 2 to VI1 converted to binary string with
                        field width of 14)
VS2?                  (* report value of string variable 2)
*"2#100011101010"
VS3=ITH(VI1,4)        (* set string variable 3 to VI1 converted to hex string with field
                        width of 4)
VS3?                  (* report value of string variable 3)
*"****"
VS3=ITH(VI1,0)        (* set string variable 3 to VI1 converted to hex string with
                        minimum field width)
VS3?                  (* report value of string variable 3)
*"16#8EA"
```

ITD, ITT Convert Integer to Time/Date Operators

Type:	String
Syntax:	ITT(*p1*) ITD(*p1*)
Parameters:	*allowed values*
p1	any integer operand in range 0 through 2,114,380,799 for date and 0 through 86,399 for time

Use: These operators are used to convert the integer operator *p1* to a date or time string. The operations are described below:

ITD	convert integer to date—converts seconds from January 1, 1994 to date.
ITT	convert integer to time—converts seconds to 24-hour time format.

Example:

```
VI1=34230          (* set integer variable 1 to 34230)
VS1=ITT(VI1)       (* set string variable 1 to VI1 converted to time string)
VS1?               (* report value of string variable 1)
*“09:30:30”
VI2=65750400       (* set integer variable 2 to 65,750,400)
VS2=ITD(VI2)       (* set string variable 2 to VI2 converted to date string)
VS2?               (* report value of string variable 2)
*“1996-02-01”
```

Appendix C Operands

FALSE, OFF, ON, *p1*, TRUE **BOOLEAN OPERANDS**

Type:	Boolean	
Syntax:	TRUE, FALSE, ON, OFF, *p1*, *p2*	
Parameters:	*allowed values*	*range*
p1	any boolean	0, 1
p2	any boolean register	

Use: These operands are used as boolean numbers. TRUE and ON are equivalent to the boolean number 1, and FALSE and OFF are equivalent to the boolean number 0.

Example:

```
VB1=TRUE        (* set boolean variable 1 to TRUE [i.e, one])
POE1=ON         (* set power output stage enable of axis one to ON [i.e., one])
DO1.8=ON        (* set digital output 8 of module 1 to one)
VB2=0           (* set boolean variable 2 to zero)
```

p1, p2 Floating Point Operands

Type:	Floating point	
Syntax:	*p1*, *p2*	
Parameters:	*allowed values*	*range*
p1	any floating point	+/- 1.5E-39 through 1.7E38
p2	any floating point register	
Use:	These operands are used as floating point numbers. Note that floating point numbers must always have a decimal point in them.	

Example:

VF1=105.	(* set floating point variable 1 to 105.)
MPA1=20.2	(* set axis one absolute move position to 20.2)
VF2=FEB1	(* set floating point variable 2 to axis one following error bound)

p1, 2#*p2*, 16#*p3*, *p4* Integer Operands

Type: Integer

Syntax: *p1*, 2#*p2*, 16#*p3*, *p4*

Parameters:	*allowed values*	*range*
p1	any integer	-2,147,483,648 through 2,147,483,647
p2	any base 2 integer	0 through 11111111111111111111111111111111 (2^32 -1)
p3	any base 16 integer	00000000 through FFFFFFFF
p4	any integer register	

Use: These operands are used as integer numbers.

Example:

```
MAP1=45              (* set axis one motion acceleration percent to 45%)
VI1=2#10101111       (* set integer variable 1 to 10101111₂  [i.e., 175₁₀])
URA1=4096            (* set axis one unit ratio to 4,096 pulses/rev)
VI2=423234           (* set integer variable 2 to 423,234)
VI3=16#40E8          (* set integer variable 3 to 40E8₁₆  [i.e., 16616₁₀])
```

"p1", p2, $p3 String Operands

Type: String

Syntax: "*p1*", *p2*, $*p3*

Parameters: *allowed values*

p1	any string, 0 through 127 characters long
p2	any string register
p3	any nonstring register

Use: These operands are used as strings.

Remarks: 1. When a string contains a dollar sign, the character immediately after it is treated in a special manner. The possibilities are:

Character after $	***Interpretation when sent to serial port***
$	dollar sign
"	quote
0 through FF	ASCII code (in hexadecimal)
T	tab
L	line feed
R	carriage return
N	new line (carriage return and line feed)

2. Using the dollar sign followed by a register converts the value of the register into the appropriate string. Floating point, integer, and boolean register values will be converted into strings containing the number value of the register. In the special case of bit-valued registers (e.g., SRS, SRP1, FCS), the register value will be converted into a string containing the hexadecimal (base 16) value of the registers. If the bit is specified (e.g., SRS1, SRP1.5, FCS2), the string will be "0" if the bit is zero; or it will be the assigned message of the bit.

Example:

```
VS1="Energy cost: $$50" (* set string variable 1 to "Energy cost: $50")
VS2="$"Hello$""         (* set string variable 2 to ""Hello"")
OUT VS2                 (* output string expression to serial port)
*""Hello""
VS3=$SRA1               (* set string variable 3 to axis one status register converted to
                          hex string)
VS3?                    (* report value of string variable 3)
*"16#0100"
VS4=$PSA1               (* set string variable 4 to axis one position converted to string)
VS4?                    (* report value of string variable 4)
*"2.563924"
VS5=$SRA1.8             (* set string variable 5 to bit 8 of axis one status register)
VS5?                    (* report value of string variable 5)
* "Axis in position"
```

Appendix D IMC and Target Command Fault and Status Messages

Command Messages—IMC and Target			
Number	**Command Message**	**Possible Cause(s)**	**Possible Solution(s)**
6	RECEIVE ERROR	A character that was entered was not received correctly by the controller.	Check to make sure that the serial/program port settings (baud, parity, databits) are correct. Check the connection to the serial/program port.
7	NETWORK ADDRESS OUT OF RANGE	The network address entered is less than 0 or greater than 63.	Re-enter the address making sure that it is a number 0 through 63.
8	LABEL OUT OF RANGE	The program label entered as part of a program statement is less than 1 or greater than 999.	Re-enter the label making sure that it is a number 1 through 999.
9	INVALID COMMAND	The system or controller did not recognize the command entered.	The command was misspelled. Re-enter correctly spelled command. The command was invalid. Re-enter valid command.
10	INVALID DIGIT	The number entered as a parameter for the command contained an invalid digit.	Re-enter the command making sure that the parameter does not contain an invalid digit.
11	INVALID ASSIGNMENT	The assignment entered was not valid for the command entered.	The assignment was misspelled. Re-enter correctly spelled command. The assignment was invalid. Re-enter the command with valid assignment.
12	TOO MANY DECIMAL PLACES	Value entered as a parameter had more decimal places than allowed for the command entered.	Re-enter the command making sure that the parameter does not have too many decimal places.
13	SYNTAX ERROR - POSSIBLY MISMATCHED OPERAND AND OPERATOR TYPE	The operands of an operator in an expression are not of the correct type for the operator.	Make sure to use the appropriate conversion operators to achieve the correct types of operands for the operator.
14	SYNTAX ERROR - POSSIBLY TOO MANY OPERANDS	There are more operands in an expression than the operators require.	Review the syntax of the operators used. Check the parentheses used for proper placement.
15	SYNTAX ERROR - POSSIBLY TOO FEW OPERANDS	There are fewer operands in an expression than the operators require.	Review the syntax of the operators used. Check the parentheses used for proper placement.
16	SYNTAX ERROR - POSSIBLY UNBALANCED PARENTHESES	There are not the same number of left parentheses as there are right parentheses.	Be sure to use the same number of left and right parentheses.

Command Messages—IMC and Target			
Number	**Command Message**	**Possible Cause(s)**	**Possible Solution(s)**
17	EXPRESSION TOO LONG	The expression entered is longer than the register or command will accept.	Simplify the expression. Break the expression into two or more parts.
18	EXPRESSION NOT BOOLEAN	The command expects an expression with a Boolean result and the expression entered evaluated to an integer, floating point, or string.	Review the expression for correct form. Consider using one of the comparison operators.
19	EXPRESSION NOT INTEGER	The command expects an expression with an integer result and the expression entered evaluated to a Boolean, floating point, or string.	Review the expression for correct form. Consider using one of the conversion operators.
20	EXPRESSION NOT FLOATING POINT	The command expects an expression with a floating point result and the expression entered evaluated to a Boolean, integer, or string.	Review the expression for correct form. Consider using one of the conversion operators.
21	EXPRESSION NOT STRING	The command expects an expression with a string result and the expression entered evaluated to a Boolean, integer, or floating point.	Review the expression for correct form. Consider using one of the conversion operators.
22	COMMAND NOT ALLOWED	The command entered in the program/motion block editor is not allowed in a program/motion block, or the command entered in immediate mode is allowed only in a program and/or motion block.	For specific information about the command you are using, see the *Restrictions* information in Appendix A.
23	NOT READY FOR COMMAND	The system was not ready to accept the command entered because it was executing an operation that cannot be interrupted by that command.	Wait until the operation finishes or stop it completely; kill programs with the KLP or KLALL command, and stop any motion with the ST or HT command. See the *Remarks* information in Appendix A.
24	OUT OF PROGRAM MEMORY	The system has run out of memory available for programs and motion blocks.	Delete any programs or motion blocks that are not currently being used.
25	NO PROGRAM FAULT	The FAULT command was entered without there being a program fault.	If the controller is faulted, the FC?, FCS?, or FCAa? command can be used to show what fault has occurred.
26	INVALID COMMAND IN STRING	An attempt was made to execute the EXVS command, but the command stored in the string variable was not recognized by the system.	The command is misspelled. Check spelling and re-enter the command. The command is invalid. Re-enter valid command.
27	TRANSMIT BUFFER OVERFLOW	The program has sent more characters to the transmit buffer than the communications port can handle.	The PUT, OUT, or OUTS commands have been executed multiple times — they are in a loop. Change the program accordingly.
28	RESOURCE NOT AVAILABLE	The addressed network controller is not online. 🗎 An attempt was made to execute a command specific to a module, but the module is not available.	Check network connections. Check network address and baud rate. 🗎 Check system configuration.

Command Messages—IMC and Target

Number	Command Message	Possible Cause(s)	Possible Solution(s)
29	INVALID VARIABLE POINTER	The pointer loaded in an integer variable was out of the range of registers available.	Re-enter the pointer making sure that it is in the range of the type of register accessed.
30	MATHEMATICAL OVERFLOW	The result of the expression entered was outside the allowed bounds of the type of expression.	Re-enter the expression making sure that the operation will never go outside the allowed bounds of the type of expression. If using an integer expression, consider using a floating point expression instead.
31	MATHEMATICAL DATA ERROR	The result of the expression entered cannot be represented as a number.	Make sure that the SQR and LGN operators never have negative operands. Make sure that a divide-by-zero operation will never occur.
32	VALUE OUT OF RANGE	The value entered as a parameter was out of the range specified for the command entered.	Re-enter the command making sure that the parameter is within the range specified for the command. See the ***Parameters*** information in Appendix A for the allowed range.
33	STRING TOO LONG	The string entered was longer than 127 characters.	Re-enter the command/string using 127 or fewer characters.
34	NONEXISTENT LABEL	The LABEL command was entered in program editor with a nonexistent label.	Re-enter the LABEL command making sure that the label exists in the program.
35	DUPLICATE LABEL	The program label entered as part of a program statement was already in the current program.	Re-enter the program statement making sure that the label does not already exist in the program. Remove the label from the existing program statement first.
36	MISSING QUOTATION MARK	A string was entered as part of a command without being enclosed in quotes.	Re-enter the command with the string enclosed in quotation marks.
37	INVALID MOTION	The combination of motion parameters defines a motion that cannot be executed, or a motion command or motion block was executed when the system was faulted.	Make sure that the motion parameters define a motion that can be executed. For specific information about the parameters you are using, see Appendix A. Make sure the system is not faulted when executing a motion command or motion block.
38	***I***, ***jr*** Reserved 🗎 INCONSISTENT AXIS GROUPING	 🗎 Two motion commands with intersecting axis sets were executed together.	 🗎 Ensure that coordinated motion commands running together are consistent.
39	SWITCH MOTOR LEADS	The MOTORSET command was entered and the system decided from its calculations that two motor leads should be switched.	Switch two of the motor leads.
40	BAD POLES RATIO	The MOTORSET command was entered, and the system calculated the motor poles to resolver poles ratio to be less than 1 or greater than 16.	Use a different resolver.
41	***I***, ***jr*** Reserved 🗎 BAD RESOLVER AMPLITUDE	🗎 The MOTORSET command was entered, and the system could not set the resolver amplitude correctly.	🗎 Use a different motor or a different resolver.

Command Messages—IMC and Target			
Number	**Command Message**	**Possible Cause(s)**	**Possible Solution(s)**
42	TORQUE TO INERTIA RATIO TOO LOW	The AUTOTUNE command was entered and the system calculated the torque to inertia ratio of the axis to be less than 125 radians/sec^2.	Autotuning with the AUTOTUNE command will not work. Use the expressions for KA, KD, KI, KP, and KT to calculate values if possible. See Appendix A.
43	TORQUE TO INERTIA RATIO TOO HIGH	The AUTOTUNE command was entered and the system calculated the torque to inertia ratio of the axis to be greater than 125,000 radians/sec^2.	Autotuning with the AUTOTUNE command will not work. Use the expressions for KA, KD, KI, KP, and KT to calculate values if possible. See Appendix A.
44	TORQUE RESPONSE NON-LINEAR	The AUTOTUNE command was entered, and the system could not calculate the control constants because the motor did not respond linearly.	Autotuning with the AUTOTUNE command will not work. Use the expressions for KA, KD, KI, KP, and KT to calculate values if possible. See Appendix A.
45	Enter password:	The PASSWORD command has been entered, and the system is waiting for the password to be entered.	Enter the password.
46	Password accepted	The PASSWORD command and the correct password have been entered, or the CHANGEPW command and the new password have been entered correctly.	Continue with normal operation.
47	Invalid password - access denied	The PASSWORD or CHANGEPW command has been entered, and the password entered is incorrect.	Enter the PASSWORD command again, and enter the correct password.
48	Enter old password:	The CHANGEPW command has been entered, and the system is waiting for the old password to be entered.	Enter the old password.
49	Enter new password:	The CHANGEPW command has been entered, and the system is waiting for the new password to be entered.	Enter the new password.
50	Enter new password again to verify:	The CHANGEPW command has been entered, and the system is waiting for the new password to be entered and verified.	Enter the new password again.
51	Invalid password - Password unchanged	The CHANGEPW command has been entered, and either the new password entered is invalid, or the new password entered the second time does not match the one entered the first time.	Enter the CHANGEPW command again to start over. Make sure that the new password is at least 4 characters and no longer than 10 characters.
52	Retrieving user memory...	The RETRIEVE command has been entered, and the system is in the process of retrieving user memory.	Wait for user memory to be retrieved.
53	User memory retrieved	The RETRIEVE command has been entered, and the system has retrieved user memory.	Continue with normal operation.
54	Saving user memory...	The SAVE command has been entered; the system is in the process of saving user memory.	Wait for user memory to be saved.

Command Messages—IMC and Target			
Number	**Command Message**	**Possible Cause(s)**	**Possible Solution(s)**
55	User memory saved	The SAVE command has been entered, and the system has saved user memory.	Continue with normal operation.
56	FLASH MEMORY PROGRAM FAILURE	The SVF or SAVE command was entered, and the flash memory could not be erased.	Try a different flash memory card. Replace the System Module.
57	FLASH MEMORY PROGRAM FAILURE	The SVF or SAVE command was entered, and the program could not be written to the flash memory card.	Try a different flash memory card. Replace the System Module.
58	STORED PROGRAM DOES NOT CHECKSUM	The RETRIEVE command was entered, and the program stored in the flash card does not checksum.	Download the program, and save the program again using the SAVE command. Replace the flash card.
59	Are you sure you want to clear all the user memory and reset the registers to their default values?	The CLM command has been entered, and the system is waiting for the user to respond.	If you are sure that you want to do this, type **Y** or **y**. The system will clear all memory and reset the registers to their default values. If you are not sure, type **N** or **n**. The system will continue with normal operation.
60	User memory cleared	The user memory has been cleared using the CLM command.	Continue with normal operation.
61	Are you sure you want to erase the current firmware and load a new firmware version?	The FIRMWARE command has been entered, and the system is waiting for the user to respond.	If you are sure that you want to do this, type **Y** or **y**. The system will erase the current firmware and load the new firmware. If you are not sure, type **N** or **n**. The system will continue with normal operation.
62	🗎 FLASH CARD NOT INSERTED	The flash memory card is not inserted in the System Module.	Insert the flash memory card into the System Module. Make sure that the flash memory card is properly seated.
63	🗎 FLASH CARD WRITE PROTECTED	The flash memory card in the System Module is write protected.	Remove write protection from the flash memory card by moving the switch to the correct position.
64	🗎 COMMAND CAN BE EXECUTED ONLY AFTER RTF COMMAND	The SVF command was executed before the RTF command was executed.	Wait until you have executed the RTF command before executing the SVF command.
67	! TERTIARY TRANSMIT BUFFER OVERFLOW	The program has sent more characters to the transmit buffer than the tertiary port can handle.	The PUTT or OUTT commands have been executed multiple times; they are in a loop. Change the program accordingly.
68	! PROGRAM TRANSMIT BUFFER OVERFLOW	The program has sent more characters to the transmit buffer than the program port can handle.	The PUTW, OUTW, or OUTS commands have been executed multiple times; they are in a loop. Change the program accordingly.
69	! EXTENDED MEMORY CARD NOT INSERTED	The extended memory card is not inserted in the System Module.	Insert the extended memory card into the System Module. Ensure the extended memory card is properly seated.
70	! EXTENDED MEMORY CARD WRITE PROTECTED	The extended memory card in the System Module is write protected.	Remove write protection from the memory card by moving the switch to the correct position.
71	! RAM CARD NOT INSERTED	The RAM memory card is not inserted in the System Module.	Insert the RAM memory card into the System Module. Ensure the RAM memory card is properly seated.

Command Messages—IMC and Target			
Number	**Command Message**	**Possible Cause(s)**	**Possible Solution(s)**
72	⧫ RAM CARD WRITE PROTECTED	The RAM memory card in the System Module is write protected.	Remove write protection from the RAM memory card by moving the switch to the correct position.
73	⧫ COPYING EXTENDED MEMORY CARD...	The COPYRAM or COPYFLASH command has been entered, and the system is in the process of copying the extended memory card.	Wait for the card to be copied.
74	🗎 EXTENDED MEMORY CARD COPIED	The COPYRAM or COPYFLASH command has been entered, and the system has copied the extended memory card.	Continue with normal operation.
75	⧫ Are you sure you want to clear the extended memory card?	The CLX command has been entered, and the system is waiting for the user to respond.	If you are sure that you want to do this, type **Y** or **y**. The system will clear the extended memory card. If you are not sure, type **N** or **n**. The system will continue with normal operation.
76	⧫ Extended memory card cleared.	The extended memory card has been cleared using the CLX command.	Continue with normal operation.

Appendix
E

IMC Fault and Status Register Messages

FC—IMC/IMJ System Fault Code Register

Bit	System Fault Code Message	Possible Cause(s)	Possible Solution(s)
0	Power Failure	A power failure has occurred. This fault always occurs when the system is powered-up.	Use the RSF command to reset the fault condition.
1	Reserved		
2	Software Fault	The STF command was executed.	Use the RSF command to reset the fault condition.
3	Lost Enable	The enable input was deactivated.	Reactivate the enable input; use the RSF command to reset the fault condition.
4	Digital Output Fault	The digital input associated with the digital output did not detect a change of state in the output after setting the digital output register, DO, and the digital output fault enable, DOE, is enabled.	Check that the output common is connected to power return and the input common is connected to power supply or vice versa, depending on whether you have a sinking or sourcing configuration. Check that the output is not shorted.
5	Invalid Command in String	The program attempted to execute the EXVS command, but the command stored in the string variable was not recognized by the system. The program attempted to execute the OUTN command, but the command sent over the network was not recognized by the recipient.	The command is misspelled. Re-enter the correctly spelled command in the program editor. The command is invalid. Re-enter the valid command in the program editor.
6	Transmit Buffer Overflow	The program has sent more characters to the transmit buffer than the COM port can handle.	The PUT, OUT, or OUTS commands have been executed multiple times—they are in a loop. Change the program accordingly.
7	Resource Not Available	The addressed network controller is not online.	Check network connections. Check network address and baud rate.
8	Invalid Variable Pointer	The pointer loaded in an integer variable in a program/motion block was out of the range of registers available.	Re-enter the pointer in the program editor making sure that it is in the range of the type of register accessed.

FC—IMC/IMJ System Fault Code Register			
Bit	**System Fault Code Message**	**Possible Cause(s)**	**Possible Solution(s)**
9	Mathematical Overflow	The result of an expression in the program or motion block was outside the allowed bounds of the type of expression.	Re-enter the expression in the program/motion block editor making sure that the operation will never go outside the allowed bounds of the type of expression. If using an integer expression, consider using a floating point expression instead.
10	Mathematical Data Error	The result of an expression in the program or motion block cannot be represented as a number.	Make sure that the SQR and LGN operators in the program/motion block never have negative operands. Make sure that a divide by zero operation will never occur in the program/motion block.
11	Value Out of Range	The value of a parameter obtained from a variable or expression was out of the range specified for the register or command in the program or motion block.	Make sure that the variable or expression stays within the range of the register or parameter of the command. See the *Parameter* and *Range* information in Appendix A.
12	String Too Long	The result of a string variable operation in the program/ motion block was longer than 127 characters.	Re-enter the string variable operation in the program/motion block editor making sure that the result is not more than 127 characters.
13	Nonexistent Label	One of these commands was in the program with a label that does not exist in the program: GOTO, GOSUB, IF...GOTO, IF...GOSUB, WAIT...ON...GOTO, STVB...GOTO, FUNCTION.	Re-enter the command in the program editor making sure that the label exists in the program. Add the label number to the appropriate statement in the program.
14	Gosub Stack Underflow	The RETURN command was executed without a corresponding GOSUB.	Make sure the program will execute a gosub the same number of times it will execute a return. Check for program flow through a subroutine without a gosub call.
15	Gosub Stack Overflow	There were more than 32 nested gosubs in the program.	Make sure the program will execute a return the same number of times it will execute a gosub. If a program leaves a subroutine without using a RETURN command, use the POP gosub stack command to remove the return address from the gosub stack.
16	Invalid Motion	The combination of motion parameters defines a motion that cannot be executed, or a motion command or motion block was executed when the system was faulted.	Make sure that the motion parameters define a motion that can be executed. For specific information about the parameters you are using, see the commands information in Appendix A. Make sure the system is not faulted when executing a motion command or motion block.
17	Reserved		

FC—IMC/IMJ System Fault Code Register			
Bit	**System Fault Code Message**	**Possible Cause(s)**	**Possible Solution(s)**
18	Reserved		
19	Network Power Failure	The network connector is disconnected, or the network power is below the minimum voltage.	Reconnect the network connector. Inspect the network power source and replace if required.
20	Duplicate Network Address	More than one device at the same MAC ID.	Assign each device a unique address.
21	Excessive Following Error	The following error, FE, was greater than the following error bound, FEB.	Make sure that the control constants are set up properly. Make sure that the position feedback wiring is correct. Make sure that the motor has sufficient torque.
22	Excessive Command Increment	Too many motions were simultaneously executed by the program.	Make sure that the program does not execute too many motions simultaneously.
23	Position Register Overflow	The axis has moved past +/-2,000,000,000 pulses and position register wrap, PWE, is disabled.	If the axis is to move constantly in one direction for long periods of time, PWE should be enabled. Make sure that the motion parameters define a motion that does not cause position register overflow. For specific information about the parameters you are using, see Appendix A.
24	Position Feedback Lost	The position feedback became disconnected.	Check the position feedback connection.
25	Motor Power Over-Voltage	The bus voltage was greater than 475 V.	The clamp did not function correctly. Make sure that the wiring is correct.
26 *(3 amp IMJ)*	Motor Power Clamp Excessive Duty Cycle–Under-Voltage	The internal clamp was operated past its rating of 25 W continuous.	Increase cycle time.
26 *(7 amp IMJ)*	Motor Power Clamp Excessive Duty Cycle–Under-Voltage	The internal clamp was operated past its rating of 25 W continuous, or the motor power is off.	Try using an external clamp resistor instead. Turn motor power on. Replace blown fuse(s).
26 *(3 & 6 amp IMC)*	Motor Power Clamp Excessive Duty Cycle–Under-Voltage	The internal clamp was operated past its rating of 50 W continuous, or the motor power is off.	Try using an external clamp resistor instead. Turn motor power on. Replace blown fuse(s).
26 *(12–28 amp)*	Motor Power Under-Voltage	The motor power is off.	Turn motor power on. Replace blown fuse(s).

FC—IMC/IMJ System Fault Code Register			
Bit	**System Fault Code Message**	**Possible Cause(s)**	**Possible Solution(s)**
27 *(3 amp IMJ)*	Reserved		
27 *(3 & 6 amp IMC; 7 amp IMJ)*	Motor Power Clamp Over-Current Fault	The external clamp resistance was less than 50 ohms.	Make sure that the resistor value is equal to 50 ohms. Make sure that the resistor is correctly wired.
27 *(12–28 amp)*	Motor Power Clamp Excessive Duty Cycle	The internal clamp was operated past its rating of 50 W continuous.	Try using an external clamp resistor.
28 *(3–7 amp)*	Motor Over-Current Fault	The controller was putting out excessive current through the motor leads.	Check the wiring of the motor leads. Ensure motor leads are not shorted.
28 *(12–28 amp)*	Motor Over-Current Fault	The external clamp resistor is shorted; or the controller was putting out excessive current through the motor leads.	Make sure the clamp leads are not shorted. Check the wiring of the motor leads. Ensure motor leads are not shorted.
29	Motor Over-Temperature	The temperature sensor in the motor sensed the motor going over its maximum allowed temperature.	Check for a broken wire in motor feedback cable. If motor is hot, it is improperly sized.
30	Control Over-Temperature	The temperature of the controller heat sink was greater than 80 degrees Celsius.	Check the controller for adequate air flow. A fan may be needed, or through-wall heat sink mounting can be used to allow adequate air flow.
31	Network Communication Error	Network is not properly configured.	Check network configuration.
All bits set to 0	Controller Functional	The controller is not faulted.	Continue with normal operation.

FC—IMC/IMJ System Fault Code Register			
Bit	**System Fault Code Message**	**Possible Cause(s)**	**Possible Solution(s)**

FI—IMC/IMJ System Fault Input Register		
Bit	**Fault Input Message**	**Possible Solution(s)**
0	Position feedback lost input active	The position feedback is disconnected.
1	Motor power over-voltage input active	The bus voltage is greater than 475 V.
2 *(3 amp IMJ)*	Motor power clamp input active	The internal clamp is on.
2 *(3 & 6 amp IMC; 7 amp IMJ)*	Motor power clamp or under-voltage input active	The internal clamp is on, or the motor power is off.
2 *(12–28 amp)*	Motor power under-voltage input active	The motor power is off.
3 *(3 amp IMJ)*	Reserved	
3 *(3 & 6 amp IMC; 7 amp IMJ)*	Motor power clamp over-current input active	The external clamp resistance is less than 50 ohms.
3 *(12–28 amp)*	Motor power clamp input active	The internal clamp is on.
4	Motor over-current input active	The controller was putting out excessive current through the motor leads.
5	Motor over-temperature input active	The temperature sensor in the motor is sensing the motor temperature is over its allowed maximum, or the motor feedback cable is not connected correctly.
6	Controller over-temperature input active	The temperature of the controller heat sink is greater than 80 degrees Celsius.
7	Network power failure input active	The network is disconnected or the network power source is below the minimum voltage.
All bits set to 0	No fault input is active	There are no currently active fault inputs.

IO—IMC/IMJ General I/O Register		
Bit	**General I/O Message**	**Description**
0 *(IMJ)*	Reserved	
0 *(IMC)*	Capture input 2 active	The position capture input 2 is active.
1 *(IMJ)*	Reserved	
1 *(IMC)*	Capture input 2 edge	A positive edge was sensed on position capture input 2.
2	Axis channel A input active	Channel A of the axis encoder is active.
3	Axis channel B input active	Channel B of the axis encoder is active.
4	Auxiliary channel A input active	Channel A of the auxiliary encoder is active.
5	Auxiliary channel B input active	Channel B of the auxiliary encoder is active.
6	Auxiliary index input active	The index input of the auxiliary encoder is active.
7	Marker input active	The resolver of a resolver feedback unit is at 0, or the index input of an encoder feedback unit is active.
8	Home input active	The home input is active.
9	Forward overtravel input active	The forward overtravel input is active.
10	Reverse overtravel input active	The reverse overtravel input is active.
11	Enable input active	The enable input is active.
12	Capture input active	The position capture input is active.
13	Capture input edge	A positive edge was sensed on the position capture input.
14	Reserved	
15	OK output active	The OK output is active.
All bits set to 0	No I/O is active	None of the above I/O is active.

SRA—IMC/IMJ Axis Status Register		
Bit	**Axis Status Message**	**Description**
0	Motion generator enabled	The motion generator is enabled.
1	Gearing enabled	Electronic gearing is enabled.
2	Phase-locked loop enabled	The phase-locked loop is enabled.
3	Motion block executing	A motion block is executing.
4	Phase error captured	The phase error, PHR, has been captured by the position capture input.
5	Phase error past bound	The phase error is past the phase error bound, PHB.
6	Axis accel/decel	The axis is either accelerating or decelerating.
7	Axis direction forward	The axis is moving or has last moved in the forward direction.
8	Axis in position	The axis is stopped and within the position band, IPB, of the command position, PSC.
9	Axis at torque limit	The torque limit enable, TLE, is enabled, and the axis is at the torque limit set by the torque limit current, TLC.
10	Axis at overtravel	The axis is either at a hardware overtravel input or a software overtravel limit.
11	Axis at software overtravel	The axis is at a software overtravel limit.
12	Motion suspended	The motion of the axis has been suspended.
13	AXIS FAULT	A fault specific to the axis has occurred.
14	Cam enabled	Cam following is enabled.
15	Reserved	
Bits 7set to 0	Axis direction reverse	The axis is moving or has last moved in the reverse direction.

SRP—IMC/IMJ Program Status Register		
Bit	**Program Status Message**	**Description**
0	Program executing	The program is executing.
1	Program locked out	The program is being locked out by another program.
2	Reserved	
3	Reserved	
4	Invalid digit in string	The program specified a string variable to floating point or integer variable conversion, and the string variable contained an invalid digit; or the floating point or integer variable input by the IN command contained an invalid digit.
5	String value out of range	The program specified a string variable to floating point or integer variable conversion, and the string variable contained a number out of the range of the variable; or the floating point or integer variable input by the IN command was out of the range of the variable.
6	Floating point value out of range	The program specified a floating point to integer variable conversion and the floating point variable contained a number out of the range of the integer variable.
7 *(IMJ)*	Reserved	
7 *(IMC)*	Invalid time/date	The program specified a time/date to integer variable conversion and the string variable contained an invalid time/date.
8	Invalid command acknowledgment	The OUSN command was executed, and the responding device didn't accept the command as valid.
9 *(IMJ)*	Variable save failure	The SVV command was executed and variables couldn't be saved in flash memory.
9 *(IMC)*	Screen lines save failure	The SVL command was executed, and the screen lines could not be saved in flash memory.
10-14	Reserved	
15	PROGRAM FAULT	The program specified caused the system to fault.
All bits set to 0	Program not executing	The program specified is not executing.

SRS—IMC/IMJ System Status Register		
Bit	**System Status Message**	**Description**
0	Program executing	One of the programs is executing.
1	Program locked out	One of the executing programs is being locked out by another program.
2	Reserved	
3	Motion block executing	One of the motion blocks is executing.
4	Key buffer empty	The key buffer contains no characters to be input by the GET or IN command.
5	Transmit buffer empty	The transmit buffer of the controller is empty.
6	Network connection available	There is a connection available for communication.
7	Network on-line	The network is ready to communicate.
8	Reserved	
9	Reserved	
10	Reserved	
11	Reserved	
12	I/O FAULT	A digital output fault has occurred. See the *Digital Output Fault* message for the FC register in Appendix E for more information.
13	AXIS FAULT	A fault specific to the axis has occurred.
14	SYSTEM FAULT	A fault has occurred. This could be any fault possible in the system.
15	MEMORY FAULT	A memory fault has occurred due to the user program memory not checksumming.
Bits 0 set to 0	No program executing	None of the programs is executing.
Bits 4 set to 0	Character in key buffer	A character is available to be input by the GET or IN command.

Appendix F

Target Fault and Status Register Messages

AME, DME, SME—Target Module Assignment Error Register

Bit	Module Assignment Error Message	Possible Cause(s)	Possible Solution(s)
0	Module in rack one, slot one did not respond to assignment	The module assigned to rack one, slot one is not in the rack with the System Module or is not functional.	Make sure that the module is in the correct slot. Replace the module if not functional.
1	Module in rack one, slot two did not respond to assignment	The module assigned to rack one, slot two is not in the rack with the System Module or is not functional.	Make sure that the module is in the correct slot. Replace the module if not functional.
2	Module in rack one, slot three did not respond to assignment	The module assigned to rack one, slot three is not in the rack with the System Module or is not functional.	Make sure that the module is in the correct slot. Replace the module if not functional.
3	Module in rack one, slot four did not respond to assignment	The module assigned to rack one, slot four is not in the rack with the System Module or is not functional.	Make sure that the module is in the correct slot. Replace the module if not functional.
4	Module in rack one, slot five did not respond to assignment	The module assigned to rack one, slot five is not in the rack with the System Module or is not functional.	Make sure that the module is in the correct slot. Replace the module if not functional.
5	Module in rack one, slot six did not respond to assignment	The module assigned to rack one, slot six is not in the rack with the System Module or is not functional.	Make sure that the module is in the correct slot. Replace the module if not functional.
6	Module in rack one, slot seven did not respond to assignment	The module assigned to rack one, slot seven is not in the rack with the System Module or is not functional.	Make sure that the module is in the correct slot. Replace the module if not functional.
7	Module in rack one, slot eight did not respond to assignment	The module assigned to rack one, slot eight is not in the rack with the System Module or is not functional.	Make sure that the module is in the correct slot. Replace the module if not functional.

AME, DME, SME—Target Module Assignment Error Register			
Bit	**Module Assignment Error Message**	**Possible Cause(s)**	**Possible Solution(s)**
8	Module in rack two, slot one did not respond to assignment	The module assigned to rack two, slot one is not in the rack with the expansion module (which is connected directly to the System Module) or is not functional.	Check the rack expansion cables to see if they are correctly installed. The expansion module in rack two is either not installed or not functional. Make sure that the module is in the correct slot. Replace the module if not functional.
9	Module in rack two, slot two did not respond to assignment	The module assigned to rack two, slot two is not in the rack with the expansion module (which is connected directly to the System Module) or is not functional.	Check the rack expansion cables to see if they are correctly installed. The expansion module in rack two is either not installed or not functional. Make sure that the module is in the correct slot. Replace the module if not functional.
10	Module in rack two, slot three did not respond to assignment	The module assigned to rack two, slot three is not in the rack with the expansion module (which is connected directly to the System Module) or is not functional.	Check the rack expansion cables to see if they are correctly installed. The expansion module in rack two is either not installed or not functional. Make sure that the module is in the correct slot. Replace the module if not functional.
11	Module in rack two, slot four did not respond to assignment	The module assigned to rack two, slot four is not in the rack with the expansion module (which is connected directly to the System Module) or is not functional.	Check the rack expansion cables to see if they are correctly installed. The expansion module in rack two is either not installed or not functional. Make sure that the module is in the correct slot. Replace the module if not functional.
12	Module in rack two, slot five did not respond to assignment	The module assigned to rack two, slot five is not in the rack with the expansion module (which is connected directly to the System Module) or is not functional.	Check the rack expansion cables to see if they are correctly installed. The expansion module in rack two is either not installed or not functional. Make sure that the module is in the correct slot. Replace the module if not functional.
13	Module in rack two, slot six did not respond to assignment	The module assigned to rack two, slot six is not in the rack with the expansion module (which is connected directly to the System Module) or is not functional.	Check the rack expansion cables to see if they are correctly installed. The expansion module in rack two is either not installed or not functional. Make sure that the module is in the correct slot. Replace the module if not functional.

AME, DME, SME—Target Module Assignment Error Register			
Bit	**Module Assignment Error Message**	**Possible Cause(s)**	**Possible Solution(s)**
14	Module in rack two, slot seven did not respond to assignment	The module assigned to rack two, slot seven is not in the rack with the expansion module (which is connected directly to the System Module) or is not functional.	Check the rack expansion cables to see if they are correctly installed. The expansion module in rack two is either not installed or not functional. Make sure that the module is in the correct slot. Replace the module if not functional.
15	Module in rack two, slot eight did not respond to assignment	The module assigned to rack two, slot eight is not in the rack with the expansion module (which is connected directly to the System Module) or is not functional.	Check the rack expansion cables to see if they are correctly installed. The expansion module in rack two is either not installed or not functional. Make sure that the module is in the correct slot. Replace the module if not functional.
16	Module in rack three, slot one did not respond to assignment	The module assigned to rack three, slot one is not in the rack with the expansion module (which is connected to rack two by the expansion module) or is not functional.	Check the rack expansion cables to see if they are correctly installed. The expansion module in rack three is either not installed or not functional. Make sure that the module is in the correct slot. Replace the module if not functional.
17	Module in rack three, slot two did not respond to assignment	The module assigned to rack three, slot two is not in the rack with the expansion module (which is connected to rack two by the expansion module) or is not functional.	Check the rack expansion cables to see if they are correctly installed. The expansion module in rack three is either not installed or not functional. Make sure that the module is in the correct slot. Replace the module if not functional.
18	Module in rack three, slot three did not respond to assignment	The module assigned to rack three, slot three is not in the rack with the expansion module (which is connected to rack two by the expansion module) or is not functional.	Check the rack expansion cables to see if they are correctly installed. The expansion module in rack three is either not installed or not functional. Make sure that the module is in the correct slot. Replace the module if not functional.
19	Module in rack three, slot four did not respond to assignment	The module assigned to rack three, slot four is not in the rack with the expansion module (which is connected to rack two by the expansion module) or is not functional.	Check the rack expansion cables to see if they are correctly installed. The expansion module in rack three is either not installed or not functional. Make sure that the module is in the correct slot. Replace the module if not functional.

AME, DME, SME—Target Module Assignment Error Register			
Bit	**Module Assignment Error Message**	**Possible Cause(s)**	**Possible Solution(s)**
20	Module in rack three, slot five did not respond to assignment	The module assigned to rack three, slot five is not in the rack with the expansion module (which is connected to rack two by the expansion module) or is not functional.	Check the rack expansion cables to see if they are correctly installed. The expansion module in rack three is either not installed or not functional. Make sure that the module is in the correct slot. Replace the module if not functional.
21	Module in rack three, slot six did not respond to assignment	The module assigned to rack three, slot six is not in the rack with the expansion module (which is connected to rack two by the expansion module) or is not functional.	Check the rack expansion cables to see if they are correctly installed. The expansion module in rack three is either not installed or not functional. Make sure that the module is in the correct slot. Replace the module if not functional.
22	Module in rack three, slot seven did not respond to assignment	The module assigned to rack three, slot seven is not in the rack with the expansion module (which is connected to rack two by the expansion module) or is not functional.	Check the rack expansion cables to see if they are correctly installed. The expansion module in rack three is either not installed or not functional. Make sure that the module is in the correct slot.
23	Module in rack three, slot eight did not respond to assignment	The module assigned to rack three, slot eight is not in the rack with the expansion module (which is connected to rack two by the expansion module) or is not functional.	Replace the module if not functional. Check the rack expansion cables to see if they are correctly installed. The expansion module in rack three is either not installed or not functional. Make sure that the module is in the correct slot. Replace the module if not functional.
All bits set to 0	All module assignments are correct	All modules are properly installed and functional.	Continue with normal operation.

AXE—Target Axis Assignment Error Register			
Bit	**Axis Assignment Error Message**	**Possible Cause(s)**	**Possible Solution(s)**
0	Axis one did not respond to assignment	An Axis Module is not in the rack with the System Module, or it is not functional.	Make sure that the Axis Module is in the rack with the System Module. Replace the module if not functional.
1	Axis two did not respond to assignment	An Axis Module is not in the rack with the System Module, or it is not functional.	Make sure that the Axis Module is in the rack with the System Module. Replace the module if not functional.
2	Axis three did not respond to assignment	A Four-Axis Module is not in the rack with the System Module, or it is not functional.	Make sure that a Four-Axis Module is in the rack with the System Module. Replace the module if not functional.
3	Axis four did not respond to assignment	A Four-Axis Module is not in the rack with the System Module, or it is not functional.	Make sure that a Four-Axis Module is in the rack with the System Module. Replace the module if not functional.
4	Axis five did not respond to assignment	An Axis Module is not in the rack with the expansion module, or it is not functional.	Check the rack expansion cables to see if they are correctly installed. The expansion module is either not installed or not functional. Make sure that the Axis Module is in the rack with the expansion module. Replace the module if not functional.
5	Axis six did not respond to assignment	An Axis Module is not in the rack with the expansion module, or it is not functional.	Check the rack expansion cables to see if they are correctly installed. The expansion module is either not installed or not functional. Make sure that the Axis Module is in the rack with the expansion module. Replace the module if not functional.
6	Axis seven did not respond to assignment	A Four-Axis Module is not in the rack with the expansion module, or it is not functional.	Check the rack expansion cables to see if they are correctly installed. The expansion module is either not installed or not functional. Make sure that a Four-Axis Module is in the rack with the expansion module. Replace the module if not functional.
7	Axis eight did not respond to assignment	A Four-Axis Module is not in the rack with the expansion module, or it is not functional.	Check the rack expansion cables to see if they are correctly installed. The expansion module is either not installed or not functional. Make sure that a Four-Axis Module is in the rack with the expansion module. Replace the module if not functional.
All bits set to 0	All axis assignments are correct	All Axis Modules are properly installed and functional.	Continue with normal operation.

FCA—Target Axis Fault Code Register			
Bit	**Axis Fault Code Message**	**Possible Cause(s)**	**Possible Solution(s)**
0	Power Failure	A power failure has occurred. This fault always occurs when the system is powered-up.	Use the RSFALL command to reset the fault condition.
1	Encoder Supply Fault	The encoder power supply of an Axis Module is not functioning correctly.	Check for short on power supply. Replace Power Module.
2	Software Fault	The STFA command was executed.	Use the RSFA command to reset the fault condition.
3	Lost Enable	The enable input was deactivated.	Reactivate the enable input and use the RSFA command to reset the fault condition.
4	Excessive Following Error	The following error, FE, was greater than the following error bound, FEB.	Make sure that the control constants are set up properly. Make sure that the position feedback wiring is correct. Make sure that the motor has sufficient torque.
5	Excessive Command Increment	Too many motions were simultaneously executed by the program.	Make sure that the program does not execute too many motions simultaneously.
6	Position Register Overflow	The axis has moved past +/-2,000,000,000 pulses, and position register wrap, PWE, is disabled.	If the axis is to move constantly in one direction for long periods of time, PWE should be enabled. Make sure that the motion parameters define a motion that does not cause position register overflow. For specific information about the parameters you are using, see Appendix A.
7	Position Feedback Lost	The position feedback became disconnected.	Check the position feedback connection.
8	Motor Power Under-Voltage	The motor power is off.	Turn motor power on. Replace blown fuse(s).
9	Motor Power Over-Voltage	The bus voltage was greater than 475 V.	The clamp did not function correctly. Make sure that the wiring is correct.
10	Motor Power Clamp Excessive Duty Cycle	The internal clamp was operated past its rating of 50 W continuous.	Try using an external clamp.
11	Motor Power Clamp Current Fault	The external clamp resistance was less than 12 ohms.	Make sure that the resistor value is equal to 12 ohms. Make sure that the resistor is correctly wired.
12	Servo Module Current Fault	The module was not able to control the current to the motor correctly.	Check the wiring of the motor leads. Make sure that the motor leads are not shorted.
13	Servo Module Over-Temperature	The temperature of the Servo Module heat sink was greater than 80 degrees Celsius.	Check the Servo Module for adequate air flow.
14	Power Module Over-Temperature	The temperature of the Power Module heat sink was greater than 80 degrees Celsius.	Check the Power Module for adequate air flow.
15	Motor Over-Temperature	The temperature sensor in the motor sensed the motor going over its maximum allowed temperature.	Check for a broken wire in the motor feedback cable. If motor is hot, it is improperly sized.
16	Reserved		
17	Reserved		

FCA—Target Axis Fault Code Register			
Bit	**Axis Fault Code Message**	**Possible Cause(s)**	**Possible Solution(s)**
18	Reserved		
19	Reserved		
20	Set Point Output Fault	The axis set point input did not detect a change of state in the axis set point output after executing the SPOA or SPA command.	Check that the output common is connected to power return and the input common is connected to power supply or vice versa, depending on whether you have a sinking or sourcing configuration. Check that the output is not shorted.
21	Reserved		
22	Reserved		
23	Reserved		
24	System Communication Error	The System Module and the Axis Module are not communicating properly.	Replace System Module and/or Axis Module if fault will not go away with RSFA command.
25	Servo Module Communication Error	The Servo Module and the Axis Module are not communicating properly.	Try reassigning modules with the SM command. Replace Servo Module and/or Axis Module if fault will not go away with RSFA command.
26	Reserved		
27	Reserved		
28	Reserved		
29	Reserved		
30	Reserved		
31	Servo Module Assignment Error	A Servo Module is not in the correct rack slot or it is not functional.	Type SME? to find which module is causing the error.
All bits set to 0	Axis Functional	The axis is not faulted.	Continue with normal operation.

FCS—Target System Fault Code Register			
Bit	**System Fault Code Message**	**Possible Cause(s)**	**Possible Solution(s)**
0	Power Failure	A power failure has occurred. This fault always occurs when the system is powered-up.	Use the RSFALL command to reset the fault condition.
1	24 Volt Supply Fault	The 24 V power supply of the System Module is not functioning correctly.	Check for short on power supply. Replace Power Module.
2	Software Fault	The STFS command was executed.	Use the RSFS command to reset the fault condition.
3	Lost Enable	The enable input was deactivated.	Reactivate the enable input and use the RSFS command to reset the fault condition.
4	Digital Output Fault	The digital input associated with the digital output did not detect a change of state in the output after setting the digital output register, DO; and the digital output fault enable, DOE, is enabled.	Check that the output common is connected to power return and the input common is connected to power supply or vice versa, depending on whether you have a sinking or sourcing configuration. Check that the output is not shorted.
5	Invalid Command in String	An attempt was made by the program to execute the EXVS command, but the command stored in the string variable was not recognized by the system.	The command is misspelled. Re-enter the correctly spelled command in the program editor. The command is invalid. Re-enter the valid command in the program editor.
6	User Transmit Buffer Overflow	The program has sent more characters to the transmit buffer than the user serial port can handle.	The PUT or OUT commands have been executed multiple times—they are in a loop. Change the program accordingly.
7	Resource Not Available	An attempt was made to execute a command specific to a module, but the module is not available. The addressed network controller is not online.	Double check the configuration of the system. Check the network connections, network address, and baud rate.
8	Invalid Variable Pointer	The pointer loaded in an integer variable in a program/motion block was out of the range of registers available.	Re-enter the pointer in the program editor making sure that it is in the range of the type of register accessed.
9	Mathematical Overflow	The result of an expression in the program or motion block was outside the allowed bounds of the type of expression.	Re-enter the expression in the program/motion block editor making sure that the operation will never go outside the allowed bounds of the type of expression. If using an integer expression, consider using a floating point expression instead.
10	Mathematical Data Error	The result of an expression in the program or motion block cannot be represented as a number.	Make sure that the SQR and LGN operators in the program/motion block never have negative operands. Make sure that a divide by zero operation will never occur in the program/motion block.
11	Value Out of Range	The value of a parameter obtained from a variable or expression was out of the range specified for the register or command in the program or motion block.	Make sure that the variable or expression stays within the range of the register or parameter of the command. See the *Parameter* and *Range* information in Appendix A.
12	String Too Long	The result of a string variable operation in the program/motion block was longer than 127 characters.	Re-enter the string variable operation in the program/motion block editor making sure that the result is not more than 127 characters.

FCS—Target System Fault Code Register

Bit	System Fault Code Message	Possible Cause(s)	Possible Solution(s)
13	Nonexistent Label	One of these commands was in the program with a label that does not exist in the program: GOTO, GOSUB, IF...GOTO, IF...GOSUB, WAIT...ON...GOTO, STVB...GOTO, FUNCTION.	Re-enter the command making sure that the label exists in the program. Add the label number to the appropriate statement in the program.
14	Gosub Stack Underflow	The RETURN command was executed without there being a corresponding GOSUB.	Make sure that the program will execute a gosub the same number of times that it will execute a return. Check for program flow through a subroutine without a gosub call.
15	Gosub Stack Overflow	There were more than 32 nested gosubs in the program.	Make sure that the program will execute a return the same number of times that it will execute a gosub. If a program leaves a subroutine without using a RETURN command, use the POP gosub stack command to remove the return address from the gosub stack.
16	Invalid Motion	The combination of motion parameters defines a motion that cannot be executed, or a motion command or motion block was executed when the system was faulted.	Make sure that the motion parameters define a motion that can be executed. For specific information about the parameters you are using, see the command summary. Make sure the system is not faulted when executing a motion command or motion block.
17	Inconsistent Axis Groupings	Two motion commands with intersecting axis sets were executed together.	Make sure that coordinated motion commands running together are consistent.
18	Duplicate Network Address	More than one device at the same MAC ID.	Assign each device a unique address.
19	Network Power Failure	The network connector is disconnected, or the network power is below the minimum voltage.	Reconnect the network connector. Inspect the network power source and replace if required.
20	Set Point Output Fault	The system set point input did not detect a change of state in the system set point output after executing the SPOS or SPS command.	Check that the output common is connected to power return and the input common is connected to power supply or vice versa, depending on whether you have a sinking or sourcing configuration. Check that the output is not shorted.
21	Tertiary Transmit Buffer Overflow	The program has sent more characters to the transmit buffer than the tertiary port can handle.	The PUTT or OUTT commands have been executed multiple times; they are in a loop. Change the program accordingly.
22	Program Transmit Buffer Overflow	The program has sent more characters to the transmit buffer than the program port can handle.	The PUTW, OUTW, or OUTS commands have been executed multiple times; they are in a loop. Change the program accordingly.
23	Firmware Load Error	The firmware did not load correctly.	Try to reload the firmware.
24	Axis Communication Error	The System Module is not communicating properly with one of the axes.	Type SRC? to determine the specific axis that is causing the error.
25	I/O Communication Error	The System Module is not communicating properly with one of the I/O Modules.	Type SRC? to determine the specific I/O Module that is causing the error.

FCS—Target System Fault Code Register			
Bit	**System Fault Code Message**	**Possible Cause(s)**	**Possible Solution(s)**
26	User Port Communication Error	The System Module and the device connected to the user port are not communicating properly.	Check to make sure that the user serial port settings (baud, parity, databits) are correct. Check the connection to the user port.
27	Network Communication Error	Network is not configured properly.	Check network configuration.
28	Axis Assignment Error	An Axis Module is not in the rack, or it is not functional.	Type AXE? to find which axis is causing the error.
29	Analog Module Assignment Error	An Analog Module is not in the correct rack slot, or it is not functional.	Type AME? to find which module is causing the error.
30	Digital Module Assignment Error	A Digital Module is not in the correct rack slot, or it is not functional.	Type DME? to find which module is causing the error.
31	Servo Module Assignment Error	A Servo Module is not in the correct rack slot, or it is not functional.	Type SME? to find which module is causing the error.
All bits set to 0	System Functional	The system is not faulted.	Continue with normal operation.

IOA—Target Axis I/O Register		
Bit	**Axis I/O Message**	**Description**
0	Set point output active	The set point output is active.
1	Set point input active	The set point input is active.
2	Axis channel A input active	Channel A of the axis encoder is active.
3	Axis channel B input active	Channel B of the axis encoder is active.
4	Auxiliary channel A input active	Channel A of the auxiliary encoder is active.
5	Auxiliary channel B input active	Channel B of the auxiliary encoder is active.
6	Position feedback lost input active	The position feedback is disconnected.
7	Marker input active	The resolver of a resolver feedback unit is at 0 or the index input of an encoder feedback unit is active.
8	Home input active	The home input is active.
9	Forward overtravel input active	The forward overtravel is active.
10	Reverse overtravel input active	The reverse overtravel input is active.
11	Enable input active	The enable input is active.
12	Capture input active	The position capture input is active.
13	Capture input edge	A positive edge was sensed on the position capture input.
14	Motor over-temperature input active	The temperature sensor in the motor is sensing the motor temperature over its allowed maximum, or the motor feedback cable is not connected properly.
15	OK output active	The OK output is active.
All bits set to 0	No I/O is active	None of the above I/O is active.

IOS—Target System I/O Register		
Bit	**System I/O Message**	**Description**
0	Set point output active	The set point output is active.
1	Set point input active	The set point input is active.
2	Flash memory card inserted	The flash memory card is inserted.
3	Flash memory card write protected	The flash memory card is write protected.
4	Extended memory card inserted	The extended memory card is inserted.
5	Extended memory card write protected	The extended memory card is write protected.
6	Extended memory card battery low	The battery in the extended memory card is low.
7	Extended memory card battery dead	The battery in the extended memory card is dead.
8	Teach pendant available	The teach pendant is available for use.
9	Suspend input active	The suspend input is active.
10	Resume input active	The resume input is active.
11	Enable input active	The enable input is active.
12	Network power failure input active	The network is disconnected, or the network power source is below the minimum voltage.
13	Reserved	
14	Ready output active	The ready output is active.
15	OK output active	The OK output is active.
All bits set to 0	No I/O is active	None of the above I/O is active.

SRA—Target Axis Status Register		
Bit	**Axis Status Message**	**Description**
0	Motion generator enabled	The motion generator is enabled.
1	Gearing enabled	Electronic gearing is enabled.
2	Phase-locked loop enabled	The phase-locked loop is enabled.
3	Motion block executing	A motion block specifying the axis is executing.
4	Phase error captured	The phase error, PHR, has been captured by the position capture input.
5	Phase error past bound	The phase error is past the phase error bound, PHB.
6	Axis accel/decel	The axis is either accelerating or decelerating.
7	Axis direction forward	The axis is moving or has last moved in the forward direction.
8	Axis in position	The axis is stopped and within the position band, IPB, of the command position, PSC.
9	Axis at torque limit	The torque limit enable, TLE, is enabled, and the axis is at the torque limit set by the torque limit current, TLC.
10	Axis at overtravel	The axis is either at a hardware overtravel input or a software overtravel limit.
11	Axis at software overtravel	The axis is at a software overtravel limit.
12	Motion suspended	The motion of the axis has been suspended.
13	AXIS FAULT	A fault specific to the axis has occurred.
14	Cam enabled	Cam following is enabled.
15	Play/record enabled	Playback or record of positions is enabled.
Bit 7 set to 0	Axis direction reverse	The axis is moving or has last moved in the reverse direction.

SRAM—Target Analog Module Status Register		
Bit	**Analog Module Status Message**	**Description**
0	Reserved	
1	Reserved	
2	Reserved	
3	Reserved	
4	Reserved	
5	Reserved	
6	Reserved	
7	System Communication Error	The Analog Module specified is not communicating properly with the System Module.
8	Reserved	
9	Reserved	
10	Reserved	
11	Module enabled	The Analog Module specified is enabled.
12	MODULE FAULT	The Analog Module specified is faulted.
Bit 12 set to 0	Module Functional	The Analog Module specified is not faulted.

SRC—Target Communication Status Register			
Bit	**Communication Status Message**	**Possible Cause(s)**	**Possible Solution(s)**
0	Axis one communication is bad	The System Module is not communicating properly with axis one.	Replace System Module and/or Axis Module if fault will not go away with the RSFALL command.
1	Axis two communication is bad	The System Module is not communicating properly with axis two.	Replace System Module and/or Axis Module if fault will not go away with the RSFALL command.
2	Axis three communication is bad	The System Module is not communicating properly with axis three.	Replace System Module and/or Axis Module if fault will not go away with the RSFALL command.
3	Axis four communication is bad	The System Module is not communicating properly with axis four.	Replace System Module and/or Axis Module if fault will not go away with the RSFALL command.
4	Axis five communication is bad	The System Module is not communicating properly with axis five.	Replace System Module and/or Axis Module if fault will not go away with the RSFALL command.
5	Axis six communication is bad	The System Module is not communicating properly with axis six.	Replace System Module and/or Axis Module if fault will not go away with the RSFALL command.
6	Axis seven communication is bad	The System Module is not communicating properly with axis seven.	Replace System Module and/or Axis Module if fault will not go away with the RSFALL command.
7	Axis eight communication is bad	The System Module is not communicating properly with axis eight.	Replace System Module and/or Axis Module if fault will not go away with the RSFALL command.
8	Analog Module one communication is bad	The System Module is not communicating properly with Analog Module one.	Try reassigning the module with the AM1 command. Replace System Module and/or Analog Module if fault will not go away with the RSFALL command.
9	Analog Module two communication is bad	The System Module is not communicating properly with Analog Module two.	Try reassigning the module with the AM2 command. Replace System Module and/or Analog Module if fault will not go away with the RSFALL command.
10	Analog Module three communication is bad	The System Module is not communicating properly with Analog Module three.	Try reassigning the module with the AM3 command. Replace System Module and/or Analog Module if fault will not go away with the RSFALL command.
11	Analog Module four communication is bad	The System Module is not communicating properly with Analog Module four.	Try reassigning the module with the AM4 command. Replace System Module and/or Analog Module if fault will not go away with the RSFALL command.
12	Reserved		
13	Reserved		
14	Reserved		
15	Reserved		

SRC—Target Communication Status Register			
Bit	**Communication Status Message**	**Possible Cause(s)**	**Possible Solution(s)**
16	Digital Module one communication is bad	The System Module is not communicating properly with Digital Module one.	Try reassigning the module with the DM1 command. Replace System Module and/or Digital Module if fault will not go away with the RSFALL command.
17	Digital Module two communication is bad	The System Module is not communicating properly with Digital Module two.	Try reassigning the module with the DM2 command. Replace System Module and/or Digital Module if fault will not go away with the RSFALL command.
18	Digital Module three communication is bad	The System Module is not communicating properly with Digital Module three.	Try reassigning the module with the DM3 command. Replace System Module and/or Digital Module if fault will not go away with the RSFALL command.
19	Digital Module four communication is bad	The System Module is not communicating properly with Digital Module four.	Try reassigning the module with the DM4 command. Replace System Module and/or Digital Module if fault will not go away with the RSFALL command.
20	Digital Module five communication is bad	The System Module is not communicating properly with Digital Module five.	Try reassigning the module with the DM5 command. Replace System Module and/or Digital Module if fault will not go away with the RSFALL command.
21	Digital Module six communication is bad	The System Module is not communicating properly with Digital Module six.	Try reassigning the module with the DM6 command. Replace System Module and/or Digital Module if fault will not go away with the RSFALL command.
22	Digital Module seven communication is bad	The System Module is not communicating properly with Digital Module seven.	Try reassigning the module with the DM7 command. Replace System Module and/or Digital Module if fault will not go away with the RSFALL command.
23	Digital Module eight communication is bad	The System Module is not communicating properly with Digital Module eight.	Try reassigning the module with the DM8 command. Replace System Module and/or Digital Module if fault will not go away with the RSFALL command.
All bits set to 0	All communication is ok	All modules are communicating properly with each other.	Continue with normal operation.

SRDM—Target Digital Module Status Register		
Bit	**Digital Module Status Message**	**Description**
0	Reserved	
1	Output Fault	The digital input associated with the digital output did not detect the output in the correct state as specified by the DO register.
2	Reserved	
3	Reserved	
4	Reserved	
5	Reserved	
6	24 Volt Supply Fault	The 24 V power supply of the specified module is not functioning correctly.
7	System Communication Error	The Digital Module specified is not communicating properly with the System Module.
8	Reserved	
9	Reserved	
10	Reserved	
11	Module Enabled	The Digital Module specified is enabled.
12	MODULE FAULT	The Digital Module specified is faulted.
Bit 12 set to 0	Module Functional	The Digital Module specified is not faulted.

SRP—Target Program Status Register		
Bit	**Program Status Message**	**Description**
0	Program executing	The program is executing.
1	Program locked out	The program is being locked out by another program.
2	Reserved	
3	Reserved	
4	Invalid digit in string	The program specified a string variable to floating point or integer variable conversion, and the string variable contained an invalid digit; or the floating point or integer variable input by the IN command contained an invalid digit.
5	String value out of range	The program specified a string variable to floating point or integer variable conversion, and the string variable contained a number out of the range of the variable; or the floating point or integer variable input by the IN command was out of the range of the variable.
6	Floating point value out of range	The program specified a floating point to integer variable conversion, and the floating point variable contained a number out of the range of the integer variable.
7	Invalid time/date	The program specified a time/date to integer variable conversion and the string variable contained an invalid time/date.
8	Invalid command acknowledgment	The OUSN command was executed, and the responding device didn't accept the command as valid.
9	Reserved	
10	Reserved	
11	Reserved	
12	Reserved	
13	Reserved	
14	Reserved	
15	PROGRAM FAULT	The program specified caused the system to fault.
Bit 0 set to 0	Program not executing	The program specified is not executing.

SRS—Target System Status Register		
Bit	**System Status Message**	**Description**
0	Program executing	One of the programs is executing.
1	Program locked out	One of the executing programs is being locked out by another program.
2	Reserved	
3	Motion block executing	One of the motion blocks is executing.
4	User receive buffer empty	The user serial port receive buffer contains no characters to be input by the GET or IN command.
5	User transit buffer empty	The user serial port transmit buffer contains no characters to be output.
6	Network connection available	There is a connection available for communication.
7	Network on-line	The network is ready to communicate.
8	All axes in position	All axes are stopped and within the position band, IPB, of the command position, PSC.
9	Axis at torque limit	An axis is at the torque limit set by the torque limit current, TLC.
10	Axis at overtravel	An axis is either at a hardware overtravel input or a software overtravel limit.
11	Axis at software overtravel	An axis is at a software overtravel limit.
12	I/O FAULT	A digital or an analog I/O Module is faulted.
13	AXIS FAULT	An axis is faulted.
14	SYSTEM FAULT	Any fault possible in the system has occurred.
15	MEMORY FAULT	A memory fault has occurred due to the user program memory not checksumming.
Bit 0 set to 0	No program executing	None of the programs is executing.
Bit 4 set to 0	Character in use buffer	A character is available to be input by the GRT or IN command.

SRSM—Target Servo Module Status Register		
Bit	**Servo Module Status Message**	**Description**
0	Under-Voltage	The motor power is off.
1	Over-Voltage	The bus voltage was greater than 475 V.
2	Clamp Excessive Duty Cycle	The internal clamp was operated past its rating of 50 W continuous.
3	Clamp Current Fault	The external clamp resistance was less than 12 ohms.
4	Current Fault	The Servo Module was not able to control the current to the motor correctly.
5	Over-Temperature	The temperature of the Servo Module heat sink was greater than 80 degrees Celsius.
6	Power Module Over-Temperature	The temperature of the Power Module heat sink was greater than 80 degrees Celsius.
7	Axis Communication Error	The Servo Module and the Axis Module are not communicating properly.
8	Servo Module Communication Error	The Servo Module and the Axis Module are not communicating properly.
9	Reserved	
10	Reserved	
11	Module Enabled	The Servo Module specified is enabled.
12	MODULE FAULT	The Servo Module specified is faulted.
Bit 12 set to 0	Module Functional	The Servo Module specified is not faulted.

SRT—Target Tertiary Status Register		
Bit	**Tertiary Status Message**	**Description**
0	Reserved	
2	Key buffer empty	The key buffer contains no characters to be input by the GETW or INW command.
3	Program transmit buffer empty	The program transmit buffer contains no characters to be output.
4	Tertiary receive buffer empty	The tertiary receive buffer contains no characters to be input by the GETT or INT command.
5	Tertiary transmit buffer empty	The tertiary transmit buffer contains no characters to be output.
16	No tertiary status active	No tertiary status is active.
Bit 2 set to 0	Character in key buffer	The key buffer contains a character to be input by the GETW or INW command.
Bit 4 set to 0	Character in tertiary buffer	A character is available to be input by the GETT or INT command.

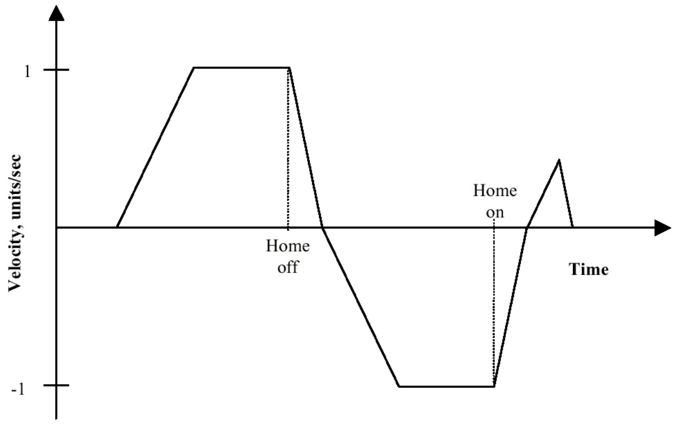
Appendix G Motion Templates

This appendix provides details on the following types of Motion templates:

- Homing Routines
- Velocity-Based Moves
- Time-Based Moves
- Pulse-Based Moves
- Torque-Limited Moves
- Synchronized Moves
- Trajectory Moves

Homing Routines

Run Reverse until Home Input



```
(* Notes:
(* 1- Registers that have been previously loaded with appropriate values do not have to be
(*     reloaded for this motion.
(* 2- Loading the MAC register also loads the MDC register with the same value.  To set
(*     MDC to a value different from MAC, load MDC after loading the MAC register.
(* 3- This example assumes the MFP (Motion Feedrate Percentage) is set to its default value
(*     of 100.  See IMC & IMJ files I_MVABSF.TXT and I_MT1ABF.TXT of Target files
(*     T_MVABSF.TXT and T_MT1ABF.TXT for examples using the MFP register.
```

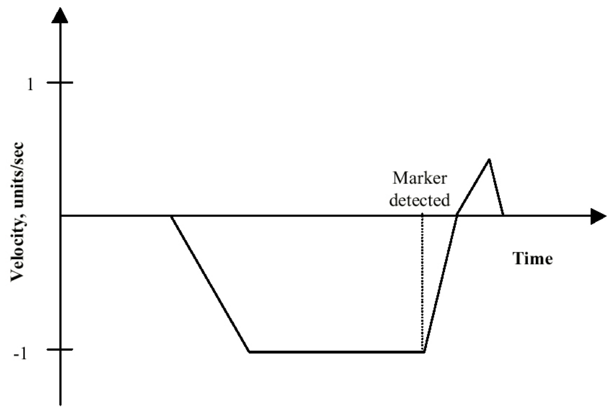
IMC & IMJ Template

```
(* Motion Template:        I_RHHM.TXT
(* Revision Log:           REV 098OCT02
(* DspMotion Series:       IMC & IMJ
(* Move Type:              Run reverse until home input
(* Engineering Units:      Motor revolutions: i.e., URA = position feedback resolution

(* Motion:                 Run forward until the home input is off.  Run reverse until
(*                         home input, then stop and run back to the position where the
(*                         home input was detected.  Set position to zero.
(* MT = VEL                This register cannot be loaded if motion is in progress.
(*                         MT does not need to be set unless it is set to a MT setting other
(*                         than VEL.  The default value for MT is VEL.


    MAC = 50.0             (* set motion acceleration, units/sec^2
    MDC = 75.0             (* set motion deceleration, units/sec^2
    MJK = 0                 (* set motion jerk percentage, % of accel & decel interval
    MVL = 1.0               (* set motion velocity, units/sec
    RVF                     (* run forward
    WAIT NOT IO8            (* wait for home input to be off
    RHR                     (* run reverse until home input
    WAIT IP                 (* wait for axis to be in position
    PSA = 0.0               (* set axis position, units
```

Target Template

```
(* Motion Template:         T_RHHM.TXT
(* Revision Log:            REV 098OCT02
(* DspMotion Series:        Target ARS
(* Move Type:               Run reverse until home input
(* Engineering Units:       Motor revolutions: i.e., URA1 = position feedback resolution

(* Motion:                  Run axis 1 forward until the home input is off.  Run reverse
(*                          until home input, then stop and run back to the position where
(*                          the home input was detected.  Set position to zero.
(* MT1 = VEL                This register cannot be loaded if motion is in progress.
(*                          MT does not need to be set unless it is set to a MT setting other
(*                          than VEL.  The default value for MT is VEL.


    MJK1 = 0                (* set motion jerk percentage, % of accel & decel interval
    MAC1 = 50.0             (* set motion acceleration, units/sec^2
    MDC1 = 75.0             (* set motion deceleration, units/sec^2
    MVL1 = 1.0              (* set motion velocity, units/sec
    RVF1                    (* run axis 1 forward
    WAIT NOT IOA1.8         (* wait for home input to be off
    RHR1                    (* run axis 1 reverse until home input
    WAIT IP1                (* wait for axis 1 to be in position
    PSA1 = 0.0              (* set axis 1 position, units
```

Run Reverse until Marker Input



```
(* Notes:
(* 1- Registers that have been previously loaded with appropriate values do not have to be
(*     reloaded for this motion.
(* 2- Loading the MAC register also loads the MDC register with the same value.  To set
(*     MDC to a value different from MAC, load MDC after loading the MAC register.
(* 3-  This example assumes the MFP (Motion Feedrate Percentage) is set to its default value
(*     of 100.  See IMC & IMJ files I_MVABSF.TXT and I_MT1ABF.TXT or Target files
(*     T_MVABSF.TXT and T_MT1ABF.TXT for examples using the MFP register.
```

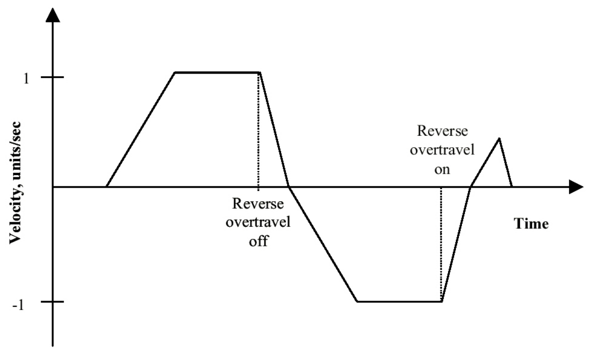
IMC & IMJ Template

```
(* Motion Template:          I_RHMK.TXT
(* Revision Log:             REV 098OCT02
(* DspMotion Series:         IMC & IMJ
(* Move Type:                Run reverse until marker input
(* Engineering Units:        Motor revolutions: i.e., URA = position feedback resolution

(* Motion:                   Run reverse until marker input, then stop and run back to the
(*                           position where the marker input was detected.  Set position to
(*                           zero.
(* MT = VEL                  This register cannot be loaded if motion is in progress.
(*                           MT does not need to be set unless it is set to a MT setting other
(*                           than VEL.  The default value for MT is VEL.


    MAC = 50.0               (* set motion acceleration, units/sec^2
    MDC = 75.0               (* set motion deceleration, units/sec^2
    MJK = 0                  (* set motion jerk percentage, % of accel & decel interval
    MVM = 1.0                (* set motion velocity for run to marker, units/sec
    RMR                      (* run reverse until marker input
    WAIT IP                 (* wait for axis to be in position
    PSA = 0.0               (* set axis position, units
```

Target Template

```
(* Motion Template:     T_RHMK.TXT
(* Revision Log:        REV 098OCT02
(* DspMotion Series:    Target ARS
(* Move Type:           Run reverse until marker input
(* Engineering Units:   Motor revolutions: i.e., URA1 = position feedback resolution

(* Motion:              Run axis 1 reverse until marker input, then stop and run back to
(*                      the position where the marker input was detected.  Set position
(*                      to zero.
(* MT1 = VEL            This register cannot be loaded if motion is in progress.
(*                      MT does not need to be set unless it is set to a MT setting other
(*                      than VEL.  The default value for MT is VEL.


    MJK1 = 0                (* set motion jerk percentage, % of accel & decel
    MAC1 = 50.0             (* set motion acceleration, units/sec^2
    MDC1 = 75.0             (* set motion deceleration, units/sec^2
    MVM1 = 1.0              (* set motion velocity for run to marker, units/sec
    RMR1                    (* run axis 1 reverse until marker input
    WAIT IP1                (* wait for axis 1 to be in position
    PSA1 = 0.0              (* set axis 1 position, units
```

Run Reverse until Overtravel Input



(* Notes:
(* 1- Registers that have been previously loaded with appropriate values do not have to be
(*   reloaded for this motion.
(* 2- Loading the MAC register also loads the MDC register with the same value. To set
(*   MDC to a value different from MAC, load MDC after loading the MAC register.
(* 3- This example assumes the MFP (Motion Feedrate Percentage) is set to its default value of
(*   100. See IMC & IMJ files I_MVABSF.TXT and I_MT1ABF.TXT or Target files
(*   T_MVABSF.TXT and T_MT1ABF.TXT for examples using the MFP register.

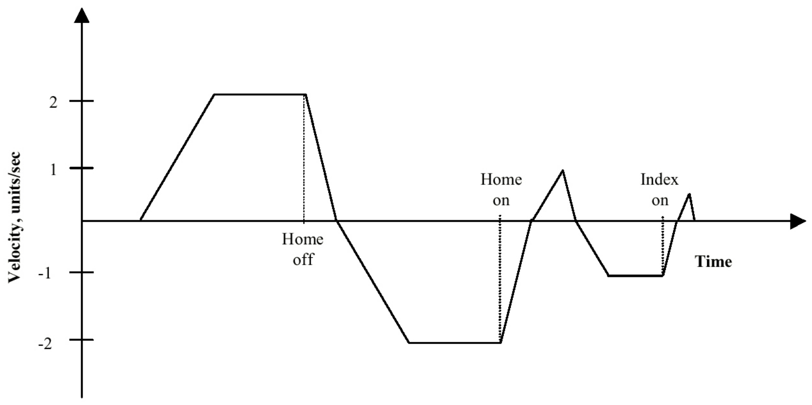
IMC & IMJ Template

```
(* Motion Template:        I_RHOT.TXT
(* Revision Log:           REV 098OCT02
(* DspMotion Series:       IMC & IMJ
(* Move Type:              Run reverse until overtravel input
(* Engineering Units:      Motor revolutions: i.e., URA = position feedback resolution

(* Motion:                 Run forward until reverse overtravel input is off.  Run reverse
(*                         until overtravel input, then stop and run back to the position
(*                         where the overtravel input was detected.  Set position to zero.
(* MT = VEL                This register cannot be loaded if motion is in progress.
(*                         MT does not need to be set unless it is set to a MT setting other
(*                         than VEL.  The default value for MT is VEL.


    MAC = 50.0             (* set motion acceleration, units/sec^2
    MDC = 75.0             (* set motion deceleration, units/sec^2
    MJK = 0                (* set motion jerk percentage, % of accel & decel interval
    MVL = 1.0              (* set motion velocity, units/sec
    RVF                    (* run forward
    WAIT NOT IO10          (* wait for reverse overtravel input to be off
    ROR                    (* run reverse until overtravel input
    WAIT IP                  (* wait for axis to be in position
    PSA = 0.0                (* set axis position, units
```

Target Template

```
(* Motion Template:      T_RHOT.TXT
(* Revision Log:         REV 098OCT02
(* DspMotion Series:     Target ARS
(* Move Type:            Run reverse until overtravel input
(* Engineering Units:    Motor revolutions: i.e.,  URA1 = position feedback resolution

(* Motion:               Run axis 1 forward until reverse overtravel input is off. Run
(*                       reverse until overtravel input, then stop and run back to the
(*                       position where the overtravel input was detected.  Set position
(*                       to zero.
(* MT1 = VEL             This register cannot be loaded if motion is in progress.
(*                       MT does not need to be set unless it is set to a MT setting other
(*                       than VEL.  The default value for MT is VEL.


    MJK1 = 0             (* set motion jerk percentage, % of accel & decel
    MAC1 = 50.0          (* set motion acceleration, units/sec^2
    MDC1 = 75.0          (* set motion deceleration, units/sec^2
    MVL1 = 1.0           (* set motion velocity, units/sec
    RVF1                 (* run axis 1 forward
    WAIT NOT IOA1.10     (* wait for reverse overtravel input to be off
    ROR1                 (* run axis 1 reverse until overtravel input
    WAIT IP1             (* wait for axis 1 to be in position
    PSA1 = 0.0           (* set axis 1 position, units
```

Run Reverse until Home and Marker Inputs



(* Notes:
(* 1- Registers that have been previously loaded with appropriate values
(*   do not have to be reloaded for this motion.
(* 2- Loading the MAC register also loads the MDC register with the same value. To set
(*   MDC to a value different from MAC, load MDC after loading the MAC register.
(* 3- This example assumes the MFP (Motion Feedrate Percentage) is set to its default value of
(*   100. See IMC & IMJ files I_MVABSF.TXT and I_MT1ABF.TXT or Target files
(*   T_MVABSF.TXT and T_MT1ABF.TXT for examples using the MFP register.

IMC & IMJ Template

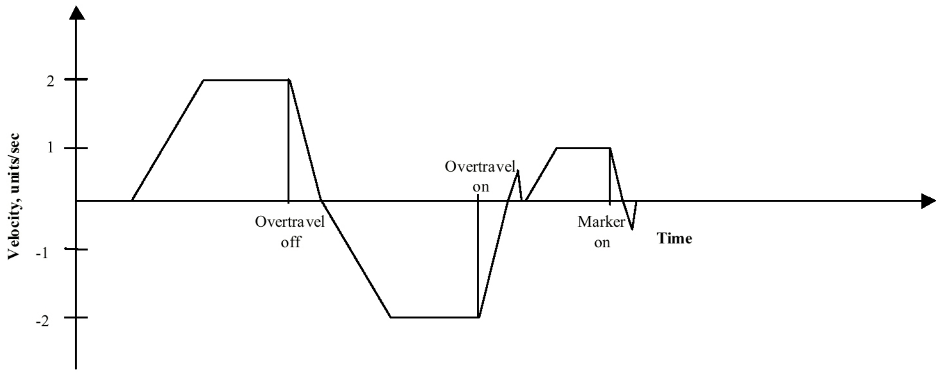
(* Motion Template:	I_RHHMMK.TXT
(* Revision Log:	REV 098OCT02
(* DspMotion Series:	IMC & IMJ
(* Move Type:	Run reverse until home and marker inputs
(* Engineering Units:	Motor revolutions: i.e., URA = position feedback resolution
(* Motion:	Run forward until home input is off. Run reverse until home
(*	input is on. Run reverse until the marker input, then stop and
(*	run back to that position. Wait until axis is in position and set
(*	position to zero.
(* MT = VEL	This register cannot be loaded if motion is in progress.
(*	MT does not need to be set unless it is set to a MT setting other
(*	than VEL. The default value for MT is VEL.

```
MAC = 50.0          (* set motion acceleration, units/sec^2
MDC = 75.0          (* set motion deceleration, units/sec^2
MJK = 0             (* set motion jerk percentage, % of accel & decel interval
MVL = 2.0           (* set motion velocity, units/sec
MVM = 1.0           (* set motion velocity for move to marker,  units/sec
RVF                 (* run forward
WAIT NOT IO8        (* wait for home input to be off
RHR                 (* run reverse until home input
WAIT IP                    (* wait for axis to be in position
RMR                        (* run reverse until index input
WAIT IP                    (* wait for axis to be in position
PSA = 0.0                  (* set axis position, units
```

Target Template

```
(* Motion Template:        T_RHHMMK.TXT
(* Revision Log:           REV 098OCT02
(* DspMotion Series:       Target ARS
(* Move Type:              Run reverse until home and marker inputs
(* Engineering Units:      Motor revolutions: i.e., URA1 = position feedback resolution

(* Motion:                 Run axis 1 forward until home input is off.  Run reverse until
(*                         home input is on.  Run reverse until the marker input, then stop
(*                         and run back to that position.  Wait until axis is in position and
(*                         set position to zero.
(* MT1 = VEL               This register cannot be loaded if motion is in progress.
(*                         MT does not need to be set unless it is set to a MT setting other
(*                         than VEL.  The default value for MT is VEL.


    MJK1 = 0               (* set motion jerk percentage, % of accel & decel
    MAC1 = 50.0            (* set motion acceleration, units/sec^2
    MDC1 = 75.0            (* set motion deceleration, units/sec^2
    MVL1 = 2.0             (* set motion velocity, units/sec
    MVM1 = 1.0             (* set motion velocity for move to marker, units/sec
    RVF1                   (* run axis 1forward
    WAIT NOT IOA1.8        (* wait for home input to be off
    RHR1                   (* run axis 1 reverse until home input
    WAIT IP1               (* wait for axis 1 to be in position
    RMR1                   (* run axis 1 reverse until index input
    WAIT IP1               (* wait for axis 1 to be in position
    PSA1 = 0.0             (* set axis 1 position, units
```

Run Reverse until Overtravel and Marker Inputs



```
(* Notes:
(* 1- Registers that have been previously loaded with appropriate values
(*    do not have to be reloaded for this motion.
(* 2- Loading the MAC register also loads the MDC register with the same value.  To set
(*    MDC to a value different from MAC, load MDC after loading the MAC register.
(* 3- This example assumes the MFP (Motion Feedrate Percentage) is set to its default value
(*    of 100.  See IMC & IMJ files I_MVABSF.TXT and I_MT1ABF.TXT or Target files
(*    T_MVABSF.TXT and T_MT1ABF.TXT for examples using the MFP register.
```

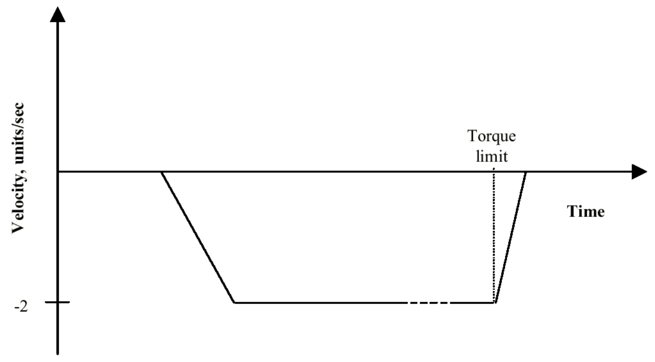
IMC & IMJ Template

```
(* Motion Template:        I_RHOTMK.TXT
(* Revision Log:           REV 098OCT02
(* DspMotion Series:       IMC & IMJ
(* Move Type:              Run reverse until overtravel and marker inputs
(* Engineering Units:      Motor revolutions: i.e., URA = position feedback resolution

(* Motion:                 Run forward until reverse overtravel input is off.  Run reverse
(*                         until overtravel input.  Run forward until marker input, then
(*                         stop and run back to the position where the marker input was
(*                         detected.  Set position to zero.
(* MT = VEL                This register cannot be loaded if motion is in progress.
(*                         MT does not need to be set unless it is set to a MT setting other
(*                         than VEL.  The default value for MT is VEL.


    MAC = 50.0             (* set motion acceleration, units/sec^2
    MDC = 75.0             (* set motion deceleration, units/sec^2
    MJK = 0                (* set motion jerk percentage,  % of accel & decel interval
    MVL = 2.0              (* set motion velocity, units/sec
    MVM = 1.0              (* set motion velocity for run to marker, units/sec
    RVF                    (* run forward
    WAIT NOT IO10          (* wait for reverse overtravel input to be off
    ROR                    (* run reverse until overtravel input
    WAIT IP                (* wait for axis to be in position
    RMF                    (* run forward until index input
    WAIT IP                (* wait for axis to be in position
    PSA = 0.0              (* set axis position, units
```

Target Template

```
(* Motion Template:        T_RHOTMK.TXT
(* Revision Log:           REV 098OCT02
(* DspMotion Series:       IMC
(* Move Type:              Run reverse until overtravel and marker inputs
(* Engineering Units:      Motor revolutions: i.e., URA1 = position feedback resolution

(* Motion:                 Run axis 1 forward until reverse overtravel input is off.  Run
(*                         reverse until overtravel input. Run forward until marker input,
(*                         then stop and run back to the position where the marker input
(*                         was detected.  Set position to zero.
(* MT1 = VEL               This register cannot be loaded if motion is in progress.
(*                         MT does not need to be set unless it is set to a MT setting other
(*                         than VEL.  The default value for MT is VEL.


    MAC1 = 50.0            (* set motion acceleration, units/sec^2
    MDC1 = 75.0            (* set motion deceleration, units/sec^2
    MJK1 = 0               (* set motion jerk percentage,  % of accel & decel interval
    MVL1 = 2.0             (* set motion velocity, units/sec
    MVM1 = 1.0             (* set motion velocity for run to marker, units/sec
    RVF1                   (* run axis 1 forward
    WAIT NOT IOA1.10       (* wait for reverse overtravel input to be off
    ROR1                   (* run axis 1 reverse until overtravel input
    WAIT IP1               (* wait for axis 1 to be in position
    RMF1                   (* run axis 1 forward until index input
    WAIT IP1               (* wait for axis 1 to be in position
    PSA1 = 0.0             (* set axis 1 position, units
```

Run Reverse until Torque Limit



```
(* Notes:
(* 1- Registers that have been previously loaded with appropriate values
(*    do not have to be reloaded for this motion.
(* 2- Loading the MAC register also loads the MDC register with the same value.  To set
(*    MDC to a value different from MAC, load MDC after loading the MAC register.
(* 3- This example assumes the MFP (Motion Feedrate Percentage) is set to its default value
(*    of 100.  See IMC & IMJ files I_MVABSF.TXT and I_MT1ABF.TXT or Target files
(*    T_MVABSF.TXT and T_MT1ABF.TXT for examples using the MFP register.
```

IMC & IMJ Template

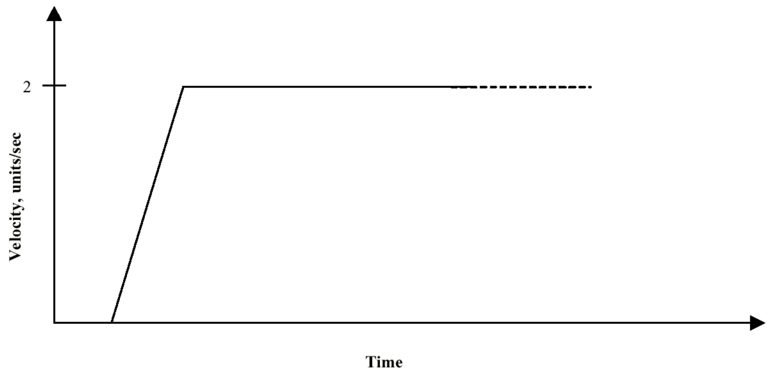
```
(* Motion Template:        I_RHTQLT.TXT
(* Revision Log:           REV 098OCT02
(* DspMotion Series:       IMC & IMJ
(* Move Type:              Run reverse until torque limit
(* Engineering Units:      Motor revolutions: i.e., URA = position feedback resolution

(* Motion:                 Run reverse until torque limit reached.  Disable torque limit
(*                         and set position to zero once axis is in position.
(* MT = VEL                This register cannot be loaded if motion is in progress.
(*                         MT does not need to be set unless it is set to a MT setting other than
(*                         VEL.  The default value for MT is VEL.


    MAC = 50.0             (* set motion acceleration, units/sec^2
    MDC = 75.0             (* set motion deceleration, units/sec^2
    MJK = 0                (* set motion jerk percentage, % of accel & decel interval
    MVL = 2.0              (* set motion velocity, units/sec
    TLC = 10.0             (* set torque limit current, % of continuous current
    TLE = ON               (* enable torque limit
    RVR                    (* run reverse
    WAIT TL                (* wait axis to be at torque limit
    HT                     (* halt all motion
    TLE = OFF                (* disable torque limit
    PSA = 0.0                (* set axis position, units
```

Target Template

```
(* Motion Template:      T_RHTQLT.TXT
(* Revision Log:         REV 098OCT02
(* DspMotion Series:     Target ARS
(* Move Type:            Run reverse until torque limit
(* Engineering Units:    Motor revolutions: i.e., URA1 = position feedback resolution

(* Motion:               Run axis 1 reverse until torque limit reached.  Disable torque
(*                       limit and set position to zero once axis is in position.
(* MT1 = VEL             This register cannot be loaded if motion is in progress.
(*                       MT does not need to be set unless it is set to a MT setting other
(*                       than VEL.  The default value for MT is VEL.


    MJK1 = 0             (* set motion jerk percentage, % of accel & decel
    MAC1 = 50.0          (* set motion acceleration, units/sec^2
    MDC1 = 75.0          (* set motion deceleration, units/sec^2
    MVL1 = 2.0           (* set motion velocity, units/sec
    TLC1 = 10.0          (* set torque limit current, % of continuous current
    TLE1 = ON            (* enable torque limit
    RVR1                 (* run axis 1 reverse
    WAIT TL1             (* wait axis 1 to be at torque limit
    HT1                  (* halt all motion
    TLE1 = OFF           (* disable torque limit
    PSA1 = 0.0           (* set axis 1 position, units
```

Velocity-Based Moves

Velocity-Based, Continuous Move



```
(* Notes:
(* 1- Registers that have been previously loaded with appropriate values
(*    do not have to be reloaded for this motion.
(* 2- Loading the MAC register also loads the MDC register with the same value.  To set
(*    MDC to a value different from MAC, load MDC after loading the MAC register.
(* 3- This example assumes the MFP (Motion Feedrate Percentage) is set to its default value
(*    of 100.  See IMC & IMJ files I_MVABSF.TXT and I_MT1ABF.TXT or Target files
(*    T_MVABSF.TXT and T_MT1ABF.TXT for examples using the MFP register.
```

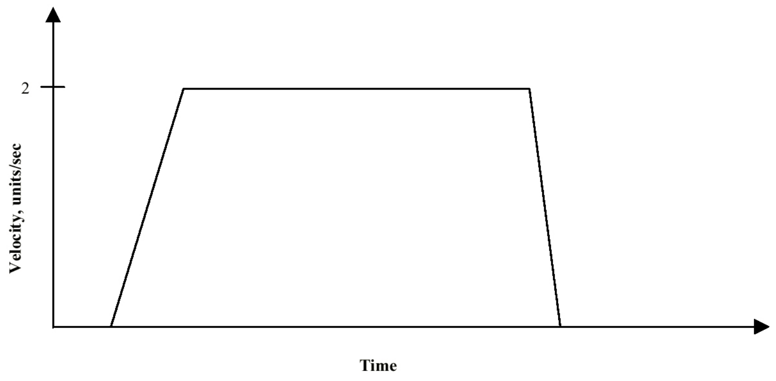
IMC & IMJ Template

```
(* Motion Template:        I_MVCON.TXT
(* Revision Log:           REV 098OCT01
(* DspMotion Series:       IMC & IMJ
(* Move Type:              Velocity-based, continuous move
(* Engineering Units:      Motor revolutions: i.e., URA = position feedback resolution

(* Motion:                 Move forward with a velocity of 2 units/sec.
(* MT = VEL                This register cannot be loaded if motion is in progress.
(*                         MT does not need to be set unless it is set to a MT setting other than
(*                         VEL.  The default value for MT is VEL.


    MAC = 50.0             (* set motion acceleration, units/sec^2
    MDC = 75.0             (* set motion deceleration, units/sec^2
    MJK = 0                (* set motion jerk percentage, % of accel & decel interval
    MVL = 2.0              (* set motion velocity, units/sec
    RVF                    (* run forward
```

Target Template

```
(* Motion Template:        T_MVCON.TXT
(* Revision Log:           REV 098OCT01
(* DspMotion Series:       Target ARS
(* Move Type:              Velocity-based, continuous move
(* Engineering Units:      Motor revolutions: i.e., URA1 = position feedback
(*                         resolution

(* Motion:                 Move axis 1 forward with a velocity of 2 units/sec.
(* MT1 = VEL               This register cannot be loaded if motion is in progress.
(*                         MT does not need to be set unless it is set to a MT setting other
(*                         than VEL.  The default value for MT is VEL.


    MAC1 = 50.0            (* set motion acceleration, units/sec^2
    MDC1 = 75.0            (* set motion deceleration, units/sec^2
    MJK1 = 0               (* set motion jerk percentage, % of accel & decel interval
    MVL1 = 2.0             (* set motion velocity, units/sec
    RVF1                   (* run axis 1 forward
```

Velocity-Based, Incremental Move



```
(* Notes:
(* 1- Registers that have been previously loaded with appropriate values
(*    do not have to be reloaded for this motion.
(* 2- Loading the MAC register also loads the MDC register with the same value.  To set
(*    MDC to a value different from MAC, load MDC after loading the MAC register.
(* 3- This example assumes the MFP (Motion Feedrate Percentage) is set to its default value
(*    of 100.  See IMC & IMJ files I_MVABSF.TXT and I_MT1ABF.TXT or Target files
(*    T_MVABSF.TXT and T_MT1ABF.TXT for examples using the MFP register.
```

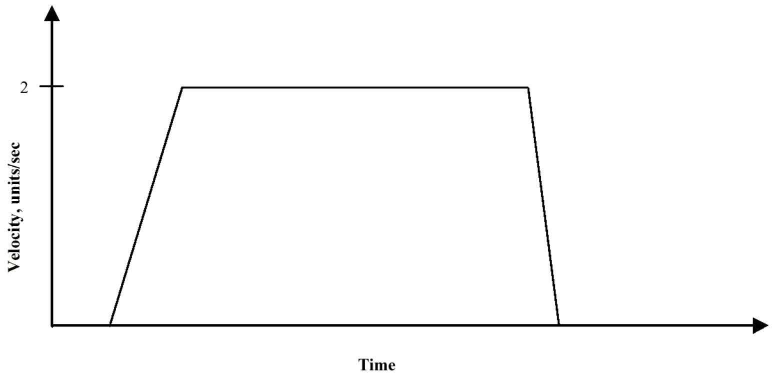
IMC & IMJ Template

```
(* Motion Template:        I_MVINC.TXT
(* Revision Log:           REV 098OCT01
(* DspMotion Series:       IMC & IMJ
(* Move Type:              Velocity-based, incremental move
(* Engineering Units:      Motor revolutions: i.e., URA = position feedback resolution

(* Motion:                 Move forward 10 units.
(* MT = VEL                This register cannot be loaded if motion is in progress.
(*                         MT does not need to be set unless it is set to a MT setting other
(*                         than VEL.  The default value for MT is VEL.


    MAC = 50.0             (* set motion acceleration, units/sec^2
    MDC = 75.0             (* set motion deceleration, units/sec^2
    MJK = 0                (* set motion jerk percentage, % of accel & decel interval
    MVL = 2.0              (* set motion velocity, units/sec
    MPI = 10.0             (* set incremental move position, units
    RPI                    (* run to incremental move position
```

Target Template

```
(* Motion Template:        T_MVINC.TXT
(* Revision Log:           REV 098OCT01
(* DspMotion Series:       Target ARS
(* Move Type:              Velocity-based, incremental move
(* Engineering Units:      Motor revolutions: i.e., URA1 = axis 1 position feedback
(*                         resolution

(* Motion:                 Move axis 1 forward 10 units.
(* MT1 = VEL               This register cannot be loaded if motion is in progress.
(*                         MT does not need to be set unless it is set to a MT setting other
(*                         than VEL.  The default value for MT is VEL.


    MAC1 = 50.0            (* set motion acceleration, units/sec^2
    MDC1 = 75.0            (* set motion deceleration, units/sec^2
    MJK1 = 0               (* set motion jerk percentage, % of accel & decel interval
    MVL1 = 2.0             (* set motion velocity, units/sec
    MPI1 = 10.0            (* set axis 1 incremental move position, units
    RPI1                   (* run axis 1 to incremental move position
```

Velocity-Based, Absolute Move



```
(* Notes:
(* 1- Registers that have been previously loaded with appropriate values
(*    do not have to be reloaded for this motion.
(* 2- Loading the MAC register also loads the MDC register with the same value.  To set
(*    MDC to a value different from MAC, load MDC after loading the MAC register.
(* 3- This example assumes the MFP (Motion Feedrate Percentage) is set to its default value
(*    of 100.  See IMC & IMJ files I_MVABSF.TXT and I_MT1ABF.TXT or Target files
(*    T_MVABSF.TXT and T_MT1ABF.TXT for examples using the MFP register.
(* 4- RPA moves the axis from its present position to the absolute position specified in the
(*    MPA register.  This example begins by loading the absolute position register, PSA, with
(*    0 for the purpose of accurately graphing the subsequent motion.  In general, applications
(*    will only load PSA at the end of a homing motion.
```

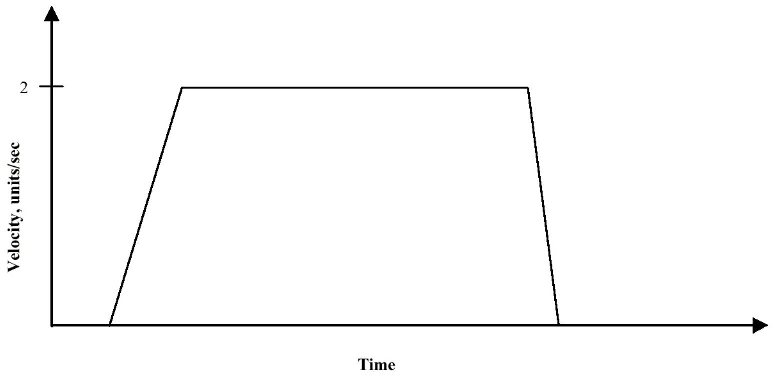
IMC & IMJ Template

```
(* Motion Template:        I_MVABS.TXT
(* Revision Log:           REV 098OCT02
(* DspMotion Series:       IMC & IMJ
(* Move Type:              Velocity-based, absolute move
(* Engineering Units:      Motor revolutions: i.e., URA = position feedback resolution

(* Motion:                 Move to absolute position of 10 units.
(* MT = VEL                This register cannot be loaded if motion is in progress.
(*                         MT does not need to be set unless it is set to a MT setting other than
(*                         VEL.  The default value for MT is VEL.

(* Initialize absolute position register to 0
    PSA = 0.0              (* set absolute position, units

(* Move axis to absolute position 10 with the accelerations and velocities shown.
    MAC = 50.0             (* set motion acceleration, units/sec^2
    MDC = 75.0             (* set motion deceleration, units/sec^2
    MJK = 0                (* set motion jerk percentage, % of accel & decel interval
     MVL = 2.0              (* set motion velocity, units/sec
     MPA = 10.0             (* set absolute move position, units
     RPA                    (* run to absolute move position
```

Target Template

```
(* Motion Template:         T_MVABS.TXT
(* Revision Log:            REV 098OCT02
(* DspMotion Series:        Target ARS
(* Move Type:               Velocity-based, absolute move
(* Engineering Units:       Motor revolutions: i.e., URA1 = axis 1 position feedback
(*                          resolution

(* Motion:                  Move axis 1 to absolute position of 10 units.
(* MT1 = VEL                This register cannot be loaded if motion is in progress.
(*                          MT does not need to be set unless it is set to a MT setting other
(*                          than VEL.  The default value for MT is VEL.

(* Initialize axis 1 absolute position register to 0
     PSA1 = 0.0             (* set axis 1 absolute position, units

(* Move axis 1 to absolute position 10 with the accelerations and velocities shown.
     MAC1 = 50.0            (* set motion acceleration, units/sec^2
     MDC1 = 75.0            (* set motion deceleration, units/sec^2
     MJK1 = 0               (* set motion jerk percentage, % of accel & decel interval
     MVL1 = 2.0             (* set motion velocity, units/sec
     MPA1 = 10.0            (* set axis 1 absolute move position, units
     RPA1                   (* run axis 1 to absolute move position
```

Velocity-Based, Offset Move



```
(* Notes:
(* 1- Registers that have been previously loaded with appropriate values
(*    do not have to be reloaded for this motion.
(* 2- Loading the MAC register also loads the MDC register with the same value.  To set
(*    MDC to a value different from MAC, load MDC after loading the MAC register.
(* 3- This example assumes the MFP (Motion Feedrate Percentage) is set to its default value
(*    of 100.  See IMC IMJ files I_MVABSF.TXT and I_MT1ABF.TXT or Target files
(*    T_MVABSF.TXT and T_MT1ABF.TXT for examples using the MFP register.
(* 4- RPO moves the axis from its present position to the offset position specified in the MPO
(*    register.  This example begins by loading the offset position register, PSO, with 0 for the
(*    purpose of accurately graphing the subsequent motion.  Applications may require other
(*    offset position register values.
```

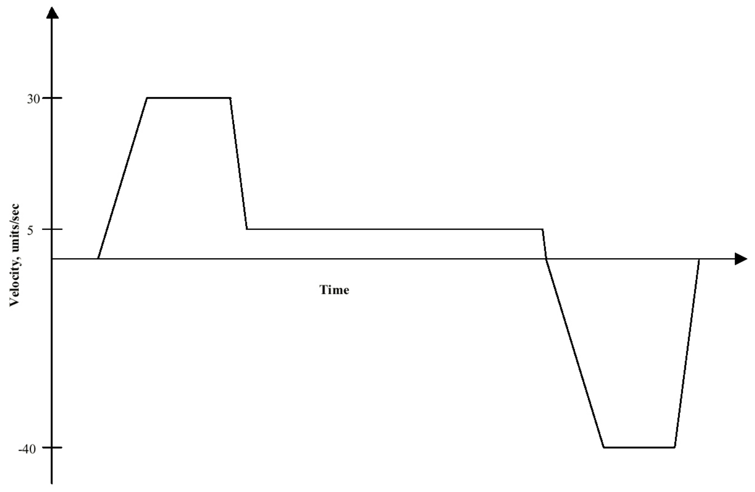
IMC & IMJ Template

```
(* Motion Template:        I_MVOFF.TXT
(* Revision Log:           REV 098OCT02
(* DspMotion Series:       IMC & IMJ
(* Move Type:              Velocity-based, offset move
(* Engineering Units:      Motor revolutions: i.e., URA = position feedback resolution

(* Motion:                 Move to offset position of 10 units.
(* MT = VEL                This register cannot be loaded if motion is in progress.
(*                         MT does not need to be set unless it is set to a MT setting other
(*                         than VEL.  The default value for MT is VEL.

(* Initialize offset position register to 0
    PSO = 0.0              (* set offset position, units

(* Move axis to offset position 10 with the accelerations and velocities shown.
    MAC = 50.0             (* set motion acceleration, units/sec^2
    MDC = 75.0             (* set motion deceleration, units/sec^2
    MJK = 0                (* set motion jerk percentage, % of accel & decel interval
    MVL = 2.0              (* set motion velocity, units/sec
MPO = 10.0              (* set offset move position, units
RPO                     (* run to offset move position
```

Target Template

```
(* Motion Template:        T_MVOFF.TXT
(* Revision Log:           REV 098OCT02
(* DspMotion Series:       Target ARS
(* Move Type:              Velocity-based, offset move
(* Engineering Units:      Motor revolutions: i.e., URA1 = axis 1 position feedback
(*                         resolution

(* Motion:                 Move axis 1 to offset position of 10 units.
(* MT1 = VEL               This register cannot be loaded if motion is in progress.
(*                         MT does not need to be set unless it is set to a MT setting other than
(*                         VEL.  The default value for MT is VEL.

(* Initialize axis 1 offset position register to 0
    PSO1 = 0.0             (* set axis 1 offset position, units

(* Move axis 1 to offset position 10 with the accelerations and velocities shown.
    MAC1 = 50.0            (* set motion acceleration, units/sec^2
    MDC1 = 75.0            (* set motion deceleration, units/sec^2
    MJK1 = 0               (* set motion jerk percentage, % of accel & decel interval
    MVL1 = 2.0             (* set motion velocity, units/sec
    MPO1 = 10.0            (* set axis 1 offset move position, units
    RPO1                   (* run axis 1 to offset move position
```

Velocity-Based, Blended Move



```
(* Notes:
(* 1- Registers that have been previously loaded with appropriate values need not be reloaded
(*    for this motion.
(* 2- Loading the MAC register also loads the MDC register with the same value.  To set
(*    MDC to a value different from MAC, load MDC after loading the MAC register.
(* 3- This example assumes the MFP (Motion Feedrate Percentage) is set to its default value
(*    of 100.  See IMC & IMJ files I_MVABSF.TXT and I_MT1ABF.TXT or Target files
(*    T_MVABSF.TXT and T_MT1ABF.TXT for examples using the MFP register.
(* 4- RPA moves the axis from its present position to the absolute position specified in the
(*    MPA register. This example begins by loading absolute position register PSA with 0 for
(*    the purpose of accurately graphing the subsequent motion.  In general, applications will
(*    only load PSA at the end of a homing motion.
(* 5- Blended moves are specified by setting a new velocity in the instruction immediately
(*    following a run command AND CAN BE DONE ONLY IN MOTION BLOCKS!
```

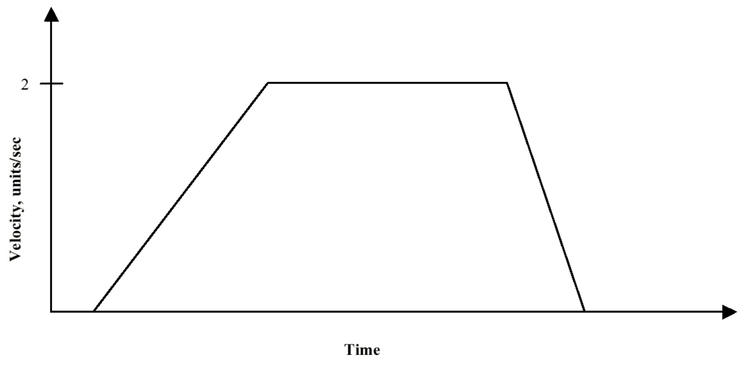
IMC & IMJ Template

```
(* Motion Template:        I_MVBLN.TXT
(* Revision Log:           REV 098OCT02
(* DspMotion Series:       IMC & IMJ
(* Move Type:              Velocity-based, blended move
(* Engineering Units:      Motor revolutions: i.e., URA = position feedback resolution

(* Motion:                 Move to 100 units at 30 units/sec, then decelerate to 5 units/sec
(*                         and move to 110 units. Finally, move back to position 0 at
(*                         40 units/sec.
(* MT = VEL                This register cannot be loaded if motion is in progress.
(*                         MT does not need to be set unless it is set to a MT setting other than
(*                         VEL.  The default value for MT is VEL.
(* Initialize absolute position register to 0
      PSA = 0.0                   (* set absolute position, units

(* Execute blended move with the accelerations and velocities shown.
      MAC = 50.0                  (* set motion acceleration, units/sec^2
      MDC = 75.0                  (* set motion deceleration, units/sec^2
      MJK = 0                     (* set motion jerk percentage, % of accel & decel interval
      MVL = 30.0                  (* set motion velocity, units/sec
      MPA = 100.0                 (* set absolute move position, units
      RPA                         (* run to absolute move position
      MVL = 5.0                   (* set motion velocity, units/sec
      MPA = 110.0                 (* set absolute move position, units
      RPA                         (* run to absolute move position
      MPA = 0.0                   (* set absolute move position, units
      MVL = 40.0                  (* set motion velocity, units/sec
      RPA                         (* run to absolute move position
```

Target Template

```
(* Motion Template:           T_MVBLN.TXT
(* Revision Log:              REV 098OCT02
(* DspMotion Series:          Target ARS
(* Move Type:                 Velocity-based, blended move
(* Engineering Units:         Motor revolutions: i.e., URA1 = position feedback resolution

(* Motion:                    Move axis 1 to 100 units at 30 units/sec, then decelerate to
(*                            5 units/sec and move to 110 units. Finally, move back to
(*                            position 0 at 40 units/sec.
(* MT1 = VEL                  This register cannot be loaded if motion is in progress.
(*                            MT does not need to be set unless it is set to a MT setting other than
(*                            VEL.  The default value for MT is VEL.

(* Initialize axis 1 absolute position register to 0
      PSA1 = 0.0                  (* set axis 1 absolute position, units

(* Execute blended move with the accelerations and velocities shown.
      MAC1 = 50.0                 (* set motion acceleration, units/sec^2
      MDC1 = 75.0                 (* set motion deceleration, units/sec^2
      MJK1 = 0                    (* set motion jerk percentage, % of accel & decel interval
      MVL1 = 30.0                 (* set motion velocity, units/sec
      MPA1 = 100.0                (* set axis 1 absolute move position, units
      RPA1                        (* run axis 1 to absolute move position
      MVL1 = 5.0                  (* set motion velocity, units/sec
      MPA1 = 110.0                (* set axis 1 absolute move position, units
      RPA1                        (* run axis 1 to absolute move position
      MPA1 = 0.0                  (* set axis 1 absolute move position, units
      MVL1 = 40.0                 (* set motion velocity, units/sec
      RPA1                        (* run axis 1 to absolute move position
```

Velocity-Based, Absolute Move with Feedrate Override



```
(* Notes:
(* 1- Registers that have been previously loaded with appropriate values
(*    do not have to be reloaded for this motion.
(* 2- Loading the MAC register also loads the MDC register with the same value.  To set
(*    MDC to a value different from MAC, load MDC after loading the MAC register.
(* 3- Loading the MFA register also loads the MFD register with the same value.  To set MFD
(*    to a value different from MFA, load MFD after loading the MFA register.
(* 4- The Motion Feedrate Percentage register, MFP, slows time by the % specified.  Thus the
(*    velocity is scaled by MFP.  Since acceleration is proportional to 1/(t^2), the acceleration
(*    is scaled by (MFP)^2.
(* 5- RPA moves the axis from its present position to the absolute position specified in the
(*    MPA register.  This example begins by loading the absolute position register, PSA, with
(*    0 for the purpose of accurately graphing the subsequent motion.  In general, applications
(*    will only load PSA at the end of a homing motion.
```

IMC & IMJ Template

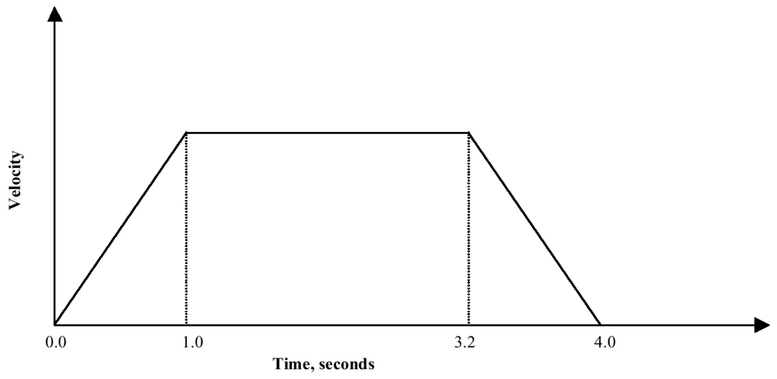
```
(* Motion Template:        I_MVABSF.TXT
(* Revision Log:           REV 098OCT02
(* DspMotion Series:       IMC & IMJ
(* Move Type:              Velocity-based, absolute move with feedrate override
(* Engineering Units:      Motor revolutions: i.e., URA = position feedback resolution

(* Motion:                 Move to 10 units at 20% of 10 units/sec, i.e., 2 units/sec.
(* MT = VEL                This register cannot be loaded if motion is in progress.
(*                         MT does not need to be set unless it is set to a MT setting other
(*                         than VEL.  The default value for MT is VEL.
(* Initialize absolute position register to 0
      PSA = 0.0                    (* set absolute position, units

(* Move axis to absolute position 10 with the accelerations and velocities shown.
      MAC = 50.0                   (* set motion acceleration, units/sec^2
      MDC = 75.0                   (* set motion deceleration, units/sec^2
      MJK = 0                      (* set motion jerk percentage, % of accel & decel interval
      MVL = 10.0                   (* set motion velocity, units/sec
      MFA = 500                    (* set motion feedrate acceleration, feedrate % / sec
      MFD = 650                    (* set motion feedrate deceleration, feedrate % / sec
      MFP = 20.0                   (* set motion feedrate percentage, % of velocity
      MPA = 10.0                   (* set absolute move position, units
      WAIT MFP <= 20.0             (* wait for feedrate to decrease to 20.0
      RPA                          (* run to absolute move position
```

Target Template

```
(* Motion Template:          T_MVABSF.TXT
(* Revision Log:             REV 098OCT02
(* DspMotion Series:         Target ARS
(* Move Type:                Velocity-based, absolute move with feedrate override
(* Engineering Units:        Motor revolutions: i.e., URA1 = position feedback resolution

(* Motion:                   Move axis 1 to 10 units at 20% of 10 units/sec, i.e., 2 units/sec.
(* MT1 = VEL                 This register cannot be loaded if motion is in progress.
(*                           MT does not need to be set unless it is set to a MT setting other
(*                           than VEL.  The default value for MT is VEL.

(* Initialize axis 1 absolute position register to 0
      PSA1 = 0.0                   (* set axis 1 absolute position, units

(* Move axis 1 to absolute position 10 with the accelerations and velocities shown.
      MAC1 = 50.0                  (* set motion acceleration, units/sec^2
      MDC1 = 75.0                  (* set motion deceleration, units/sec^2
      MJK1 = 0                     (* set motion jerk percentage, % of accel & decel interval
      MVL1 = 10.0                  (* set motion velocity, units/sec
      MFA1 = 500                   (* set motion feedrate acceleration, feedrate % / sec
      MFD1 = 650                   (* set motion feedrate deceleration, feedrate % / sec
      MFP1 = 20.0                  (* set motion feedrate percentage, % of velocity
      MPA1 = 10.0                  (* set axis 1 absolute move position, units
      WAIT MFP1 <= 20.0            (* wait for feedrate to decrease to 20.0
      RPA1                         (* run axis 1 to absolute move position
```

Timed-Based Moves

Time-Based, Single-Axis Incremental Move



(* Notes:
(* 1- Registers that have been previously loaded with appropriate values
(* do not have to be reloaded for this motion.
(* 2- Loading the MAP register also loads the MDP register with the same value. To set MDP
(* to a value different from MAP, load MDP after loading the MAP register.
(* 3- This example assumes the MFP (Motion Feedrate Percentage) is set to its default value
(* of 100. See IMC & IMJ files I_MVABSF.TXT and I_MT1ABF.TXT or Target files
(* T_MVABSF.TXT and T_MT1ABF.TXT for examples using the MFP register.

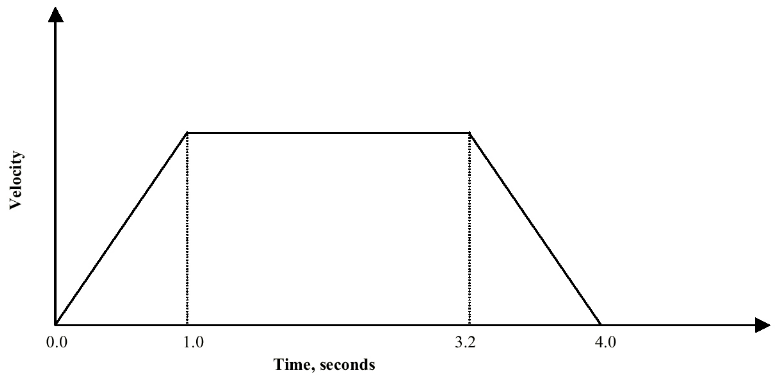
IMC & IMJ Template

```
(* Motion Template:        I_MT1INC.TXT
(* Revision Log:           REV 098OCT01
(* DspMotion Series:       IMC & IMJ
(* Move Type:              Time-based, single-axis incremental move
(* Engineering Units:      Motor revolutions: i.e., URA = position feedback resolution

(* Motion:                 Move 5 units forward in 4.0 seconds.
(* MT = TIME               This register cannot be loaded if motion is in progress.
(*                         MT does not need to be set unless it is set to a MT setting other
(*                         than TIME. The default value for MT is VEL.


   MAP = 25                (* set motion acceleration percentage, % of move time
   MDP = 20                (* set motion deceleration percentage, % of move time
   MJK = 0                 (* set motion jerk percentage, % of accel & decel interval
   MTM = 4.0               (* set move time, seconds
   MPI = 5.0               (* set incremental move position, units
   RPI                     (* run to incremental move position
```

Target Template

```
(* Motion Template:          T_MT1INC.TXT
(* Revision Log:             REV 098OCT02
(* DspMotion Series:         Target ARS
(* Move Type:                Time-based, single-axis incremental move
(* Engineering Units:        Motor revolutions: i.e., URA1 = position feedback resolution

(* Motion:                   Move axis 1 5 units forward in 4.0 seconds.
(* MT1 = TIME                This register cannot be loaded if motion is in progress.
(*                           MT does not need to be set unless it is set to a MT setting other than
(*                           TIME.  The default value for MT is VEL.


    MAP1 = 25                (* set motion acceleration percentage, % of move time
    MDP1 = 20                (* set motion deceleration percentage,  % of move time
    MJK1 = 0                 (* set motion jerk percentage, % of accel & decel interval
    MTM1 = 4.0               (* set move time, seconds
    MPI1 = 5.0               (* set axis 1 incremental move position, units
    RPI1                     (* run axis 1 to incremental move position
```

Time-Based, Single-Axis Absolute Move



```
(* Notes:
(* 1- Registers that have been previously loaded with appropriate values
(*    do not have to be reloaded for this motion.
(* 2- Loading the MAP register also loads the MDP register with the same value.  To set MDP
(*    to a value different from MAP, load MDP after loading the MAP register.
(* 3- This example assumes the MFP (Motion Feedrate Percentage) is set to its default value
(*    of 100.  See IMC & IMJ files I_MVABSF.TXT and I_MT1ABF.TXT and Target files
(*    T_MVABSF.TXT and T_MT1ABF.TXT for examples using the MFP register.
(* 4- RPA moves the axis from its present position to the absolute position specified in the
(*    MPA register.  This example begins by loading the absolute position register, PSA, with
(*    0 for the purpose of accurately graphing the subsequent motion.  In general, applications
(*    will only load PSA at the end of a homing motion.
```

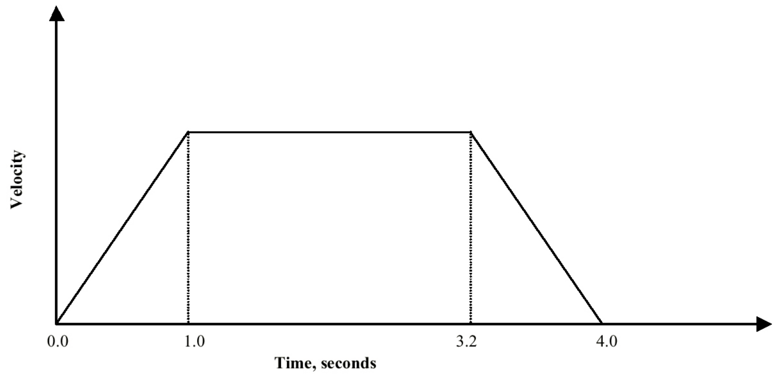
IMC & IMJ Template

```
(* Motion Template:       I_MT1ABS.TXT
(* Revision Log:          REV 098OCT02
(* DspMotion Series:      IMC & IMJ
(* Move Type:             Time-based, single-axis absolute move
(* Engineering Units:     Motor revolutions: i.e., URA = position feedback resolution

(* Motion:                Move to absolute position of 5 units in 4.0 seconds.
(* MT = TIME              This register cannot be loaded if motion is in progress.
(*                        MT does not need to be set unless it is set to a MT setting other
(*                        than TIME.  The default value for MT is VEL.

(* Initialize absolute position register to 0
    PSA = 0.0             (* set absolute position, units

(* Move axis to absolute position 5 with the accelerations and move times shown.
    MAP = 25              (* set motion acceleration percentage, % of move time
    MDP = 20              (* set motion deceleration percentage, % of move time
    MJK = 0               (* set motion jerk percentage, % of accel & decel interval
    MTM = 4.0            (* set move time, seconds
    MPA = 5.0            (* set absolute move position, units
    RPA                  (* run to absolute move position
```

Target Template

```
(* Motion Template:      T_MT1ABS.TXT
(* Revision Log:         REV 098OCT02
(* DspMotion Series:     Target ARS
(* Move Type:            Time-based, single-axis absolute move
(* Engineering Units:    Motor revolutions: i.e., URA1 = position feedback resolution

(* Motion:               Move axis 1 to absolute position of 5 units in 4.0 seconds.
(* MT1 = TIME            This register cannot be loaded if motion is in progress.
(*                       MT does not need to be set unless it is set to a MT setting other than
(*                       TIME.  The default value for MT is VEL.

(* Initialize axis 1 absolute position register to 0
    PSA1 = 0.0           (* set axis 1 absolute position, units

(* Move axis 1 to absolute position 5 with the accelerations and move times shown.
    MAP1 = 25            (* set motion acceleration percentage, % of move time
    MDP1 = 20            (* set motion deceleration percentage, % of move time
    MJK1 = 0             (* set motion jerk percentage, % of accel & decel interval
    MTM1 = 4.0           (* set move time, seconds
    MPA1 = 5.0           (* set axis 1 absolute move position, units
    RPA1                 (* run axis 1 to absolute move position
```

Time-Based, Single-Axis Offset Move



```
(* Notes:
(* 1- Registers that have been previously loaded with appropriate values
(*    do not have to be reloaded for this motion.
(* 2- Loading the MAP register also loads the MDP register with the same value.  To set MDP
(*    to a value different from MAP, load MDP after loading the MAP register.
(* 3- This example assumes the MFP (Motion Feedrate Percentage) is set to its default value
(*    of 100.  See files I_MVABSF.TXT and I_MT1ABF.TXT for examples using the MFP
(*    register.
(* 4- RPO moves the axis from its present position to the offset position specified in the MPO
(*    register.  This example begins by loading the offset position register, PSO, with 0 for the
(*    purpose of accurately graphing the subsequent motion.  Applications may require other
(*    offset position register values.
```

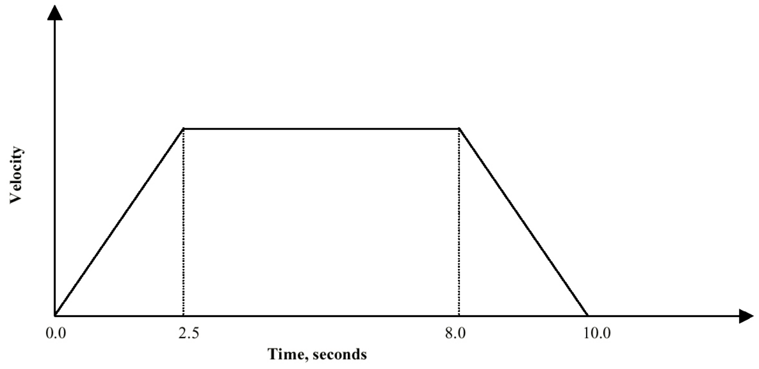
IMC & IMJ Template

```
(* Motion Template:          I_MT1OFF.TXT
(* Revision Log:             REV 098OCT02
(* DspMotion Series:         IMC & IMJ
(* Move Type:                Time-based, single-axis offset move
(* Engineering Units:        Motor revolutions: i.e., URA = position feedback resolution

(* Motion:                   Move to offset position of 5 units in 4.0 seconds.

(* MT = TIME                 This register cannot be loaded if motion is in progress.
(*                           MT does not need to be set unless it is set to a MT setting other
(*                           than TIME.  The default value for MT is VEL.

(* Initialize offset position register to 0
     PSO = 0.0               (* set offset position, units

(* Move axis to offset position 5 with the accelerations and move times shown.
     MAP = 25                (* set motion acceleration percentage, % of move time
     MDP = 20                (* set motion deceleration percentage, % of move time
     MJK = 0                 (* set motion jerk percentage, % of accel & decel interval
     MTM = 4.0               (* set move time, seconds
     MPO = 5.0                (* set offset move position, units
     RPO                      (* run to offset move position
```

Target Template

```
(* Motion Template:         T_MT1OFF.TXT
(* Revision Log:            REV 098OCT02
(* DspMotion Series:        Target ARS
(* Move Type:               Time-based, single-axis offset move
(* Engineering Units:       Motor revolutions: i.e., URA1 = position feedback resolution

(* Motion:                  Move axis 1 to offset position of 5 units in 4.0 seconds.
(* MT1 = TIME               This register cannot be loaded if motion is in progress.
(*                          MT does not need to be set unless it is set to a MT setting other
(*                          than TIME.  The default value for MT is VEL.

(* Initialize axis 1 offset position register to 0
    PSO1 = 0.0              (* set axis 1 offset position, units

(* Move axis 1 to offset position 5 with the accelerations and move times shown.
    MAP1 = 25               (* set motion acceleration percentage, % of move time
    MDP1 = 20               (* set motion deceleration percentage, % of move time
    MJK1 = 0                (* set motion jerk percentage, % of accel & decel interval
    MTM1 = 4.0              (* set move time, seconds
    MPO1 = 5.0              (* set axis 1 offset move position, units
    RPO1                    (* run axis 1 to offset move position
```

Time-Based, Single-Axis Absolute Move with Feedrate Override



```
(* Notes:
(* 1- Registers that have been previously loaded with appropriate values
(*    do not have to be reloaded for this motion.
(* 2- Loading the MAP register also loads the MDP register with the same value.  To set MDP
(*    to a value different from MAP, load MDP after loading the MAP register.
(* 3- Loading the MFA register also loads the MFD register with the same value.  To set MFD
(*    to a value different from MFA, load MFD after loading the MFA register.
(* 4- The Motion Feedrate Percentage register, MFP, slows time by the % specified.  Thus the
(*    move time and the accel and decel times are increased by the reciprocal of the %
(*    specified in MFP.
(* 5- RPA moves the axis from its present position to the absolute position specified in the
(*    MPA register.  This example begins by loading the absolute position register, PSA, with
(*    0 for the purpose of accurately graphing the subsequent motion.  In general, applications
(*    will only load PSA at the end of a homing motion.
```

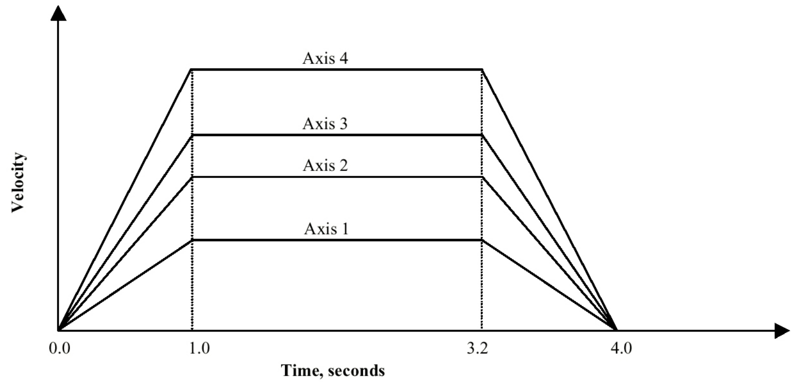
IMC & IMJ Template

```
(* Motion Template:       I_MT1ABF.TXT
(* Revision Log:          REV 098OCT02
(* DspMotion Series:      IMC & IMJ
(* Move Type:             Time-based, single-axis absolute move with feedrate override
(* Engineering Units:     Motor revolutions: i.e., URA = position feedback resolution

(* Motion:                Move to 5 units in 10 seconds.
(*                        (note that 40% of 10 seconds = 4 seconds)
(* MT = TIME              This register cannot be loaded if motion is in progress.
(*                        MT does not need to be set unless it is set to a MT setting other
(*                        than TIME.  The default value for MT is VEL.
(* Initialize absolute position register to 0
      PSA = 0.0                (* set absolute position, units

(* Move axis to absolute position 5 with the accelerations and move times shown.
      MAP = 25                 (* set motion acceleration percentage, % of move time
      MDP = 20                 (* set motion deceleration percentage, % of move time
      MJK = 0                  (* set motion jerk percentage, % of accel & decel interval
      MTM = 4.0                (* set move time, seconds
      MPA = 5.0                (* set absolute move position, units
      MFA = 500                (* set motion feedrate acceleration, feedrate % / sec
      MFD = 650                (* set motion feedrate deceleration, feedrate % / sec
      MFP = 40.0               (* set motion feedrate percentage, % of velocity
      WAIT MFP <= 40.0         (* wait for feedrate to decrease to 40.0
      RPA                      (* run to absolute move position
```

Target Template

```
(* Motion Template:            T_MT1ABF.txt
(* Revision Log:               REV 098OCT02
(* DspMotion Series:           Target ARS
(* Move Type:                  Time-based, single-axis absolute move with feedrate override
(* Engineering Units:          Motor revolutions: i.e., URA1 = position feedback resolution

(* Motion:                     Move axis 1 to 5 units in 10 seconds
(*                             (note that 40% of 10 seconds = 4 seconds)
(* MT1 = TIME                  This register cannot be loaded if motion is in progress.
(*                             MT does not need to be set unless it is set to a MT setting other
(*                             than TIME.  The default value for MT is VEL.

(* Initialize axis 1 absolute position register to 0
      PSA = 0.0                (* set absolute position, units

(* Move axis 1 to absolute position 5 with the accelerations and move times shown.
      MAP1 = 25                (* set motion acceleration percentage, % of move time
      MDP1 = 20                (* set motion deceleration percentage, % of move time
      MJK1 = 0                 (* set motion jerk percentage, % of accel & decel interval
      MTM1 = 4.0               (* set move time, seconds
      MPA1 = 5.0               (* set absolute move position, units
      MFA1= 500                (* set motion feedrate acceleration, feedrate % / sec
      MFD1 = 650               (* set motion feedrate deceleration, feedrate % / sec
      MFP1 = 40.0              (* set motion feedrate percentage, % of velocity
      WAIT MFP1 <= 40.0        (* wait for feedrate to decrease to 40.0
      RPA1                     (* run axis 1 to absolute move position
```

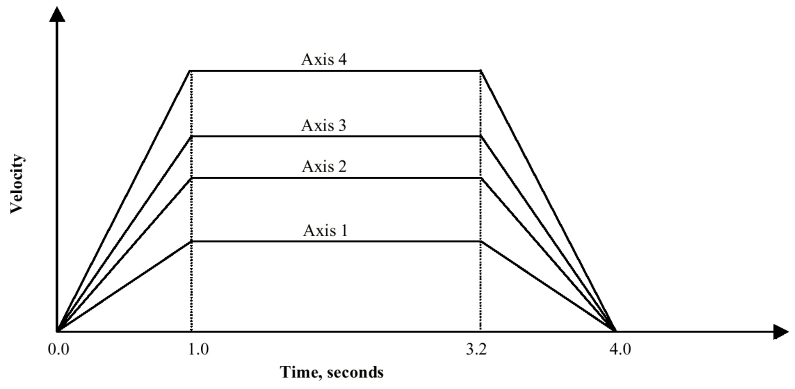
Time-Based, Multi-axis Incremental Move ⊙



Target Template

```
(* Motion Template:          T_MT4INC.TXT
(* Revision Log:             REV 098OCT02
(* DspMotion Series:         Target ARS
(* Move Type:                Time-based, multi-axis incremental move
(* Engineering Units:        Motor revolutions: i.e., URA1 = axis 1 feedback resolution
(*                           URA2 = axis 2 feedback resolution, etc.

(* Motion:                   Move axes 1, 2, 3 and 4 forward by 3, 5, 6 and 8 units in
(*                           4.0 seconds.

(* Notes:
(* 1- Registers that have been previously loaded with appropriate values
(*    do not have to be reloaded for this motion.
(* 2-  Loading the MAP register also loads the MDP register with the same value.  To set
(*    MDP to a value different from MAP, load MDP after loading the MAP register.
(* 3- This example assumes the MFP (Motion Feedrate Percentage) is set to its default value
(*    of 100.  See files T_MVABSF.TXT and T_MT1ABF.TXT for examples using the MFP
(*    register.

(* MT1 = TIME
(* MT2 = TIME
(* MT3 = TIME
(* MT4 = TIME                The MT register cannot be loaded if motion is in progress.
(*                           MT does not need to be set unless it is set to a MT setting other
(*                           than TIME.  The default value for MT is VEL.


   MAP1 = 25                 (* set motion accel percentage, % of move time
   MAP2 = 25                 (* set motion accel percentage, % of move time
   MAP3 = 25                 (* set motion accel percentage, % of move time
   MAP4 = 25                 (* set motion accel percentage, % of move time
   MDP1 = 20                 (* set motion decel percentage, % of move time
   MDP2 = 20                 (* set motion decel percentage, % of move time
   MDP3 = 20          (* set motion decel percentage, % of move time
   MDP4 = 20          (* set motion decel percentage, % of move time
   MJK1 = 0           (* set motion jerk percentage, % of accel & decel interval
   MJK2 = 0           (* set motion jerk percentage, % of accel & decel interval
   MJK3 = 0           (* set motion jerk percentage, % of accel & decel interval
   MJK4 = 0           (* set motion jerk percentage, % of accel & decel interval
   MTM1 = 4.0         (* set move time, seconds
   MTM2 = 4.0         (* set move time, seconds
   MTM3 = 4.0         (* set move time, seconds
   MTM4 = 4.0         (* set move time, seconds
   MPI1 = 3.0         (* set axis 1 incremental move position, units
   MPI2 = 5.0         (* set axis 2 incremental move position, units
   MPI3 = 6.0         (* set axis 3 incremental move position, units
   MPI4 = 8.0         (* set axis 4 incremental move position, units
   RPI1234            (* run axes 1, 2, 3 and 4 to incremental move positions
```

Time-Based, Multi-axis Absolute Move ⊙



Target Template

(* Motion Template:	T_MT4ABS.TXT
(* Revision Log:	REV 098OCT02
(* DspMotion Series:	Target ARS
(* Move Type:	Time-based, multi-axis absolute move
(* Engineering Units:	Motor revolutions: i.e., URA1 = axis 1 feedback resolution
(*	URA2 = axis 2 feedback resolution, etc.

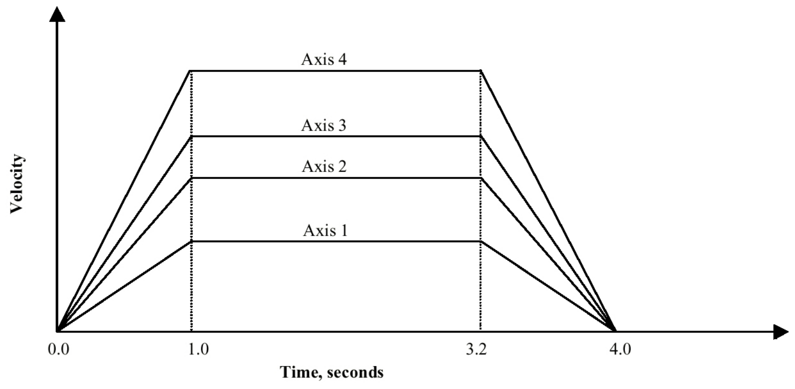
(* Motion:	Move axes 1, 2, 3 and 4 to absolute positions of 3, 5, 6 and
(*	8 units in 4.0 seconds.

(* Notes:
(* 1- Registers that have been previously loaded with appropriate values
(* do not have to be reloaded for this motion.
(* 2- Loading the MAP register also loads the MDP register with the same value. To set MDP
(* to a value different from MAP, load MDP after loading the MAP register.
(* 3- This example assumes the MFP (Motion Feedrate Percentage) is set to its default value
(* of 100. See files T_MVABSF.TXT and T_MT1ABF.TXT for examples using the MFP
(* register.
(* 4- RPA moves the axes from their present positions to the absolute positions specified in the
(* MPA registers. This example begins by loading the absolute position registers, PSA1
(* through PSA4, with 0 for the purpose of accurately graphing the subsequent motion. In
(* general, applications will only load PSA at the end of a homing motion.

(* MT1 = TIME	
(* MT2 = TIME	
(* MT3 = TIME	
(* MT4 = TIME	The MT register cannot be loaded if motion is in progress.
(*	MT does not need to be set unless it is set to a MT setting other
(*	than TIME. The default value for MT is VEL.

```
(* Initialize absolute position registers to 0
    PSA1 = 0.0              (* set axis 1 absolute position, units
    PSA2 = 0.0              (* set axis 2 absolute position, units
    PSA3 = 0.0              (* set axis 3 absolute position, units
    PSA4 = 0.0              (* set axis 4 absolute position, units

(* Move axes to absolute positions 3, 5, 6 and 8 with the accelerations and move times shown.
    MAP1 = 25               (* set motion accel percentage, % of move time
    MAP2 = 25               (* set motion accel percentage, % of move time
    MAP3 = 25               (* set motion accel percentage, % of move time
    MAP4 = 25               (* set motion accel percentage, % of move time
    MDP1 = 20               (* set motion decel percentage, % of move time
    MDP2 = 20               (* set motion decel percentage, % of move time
    MDP3 = 20               (* set motion decel percentage, % of move time
    MDP4 = 20               (* set motion decel percentage, % of move time
    MJK1 = 0                (* set motion jerk percentage, % of accel & decel interval
    MJK2 = 0                (* set motion jerk percentage, % of accel & decel interval
    MJK3 = 0                (* set motion jerk percentage, % of accel & decel interval
    MJK4 = 0                (* set motion jerk percentage, % of accel & decel interval
    MTM1 = 4.0              (* set move time, seconds
    MTM2 = 4.0              (* set move time, seconds
    MTM3 = 4.0              (* set move time, seconds
    MTM4 = 4.0              (* set move time, seconds
    MPA1 = 3.0              (* set axis 1 absolute move position, units
    MPA2 = 5.0              (* set axis 2 absolute move position, units
    MPA3 = 6.0              (* set axis 3 absolute move position, units
    MPA4 = 8.0              (* set axis 4 absolute move position, units
    RPA1234                 (* run axes 1, 2, 3 and 4 to absolute move positions
```

Time-Based, Multi-axis Offset Move ⊙



Target Template

(* Motion Template:	T_MT4OFF.TXT
(* Revision Log:	REV 098OCT02
(* DspMotion Series:	Target ARS
(* Move Type:	Time-based, multi-axis absolute move
(* Engineering Units:	Motor revolutions: i.e., URA1 = axis 1 feedback resolution
(*	URA2 = axis 2 feedback resolution, etc.
(* Motion:	Move axes 1, 2, 3 and 4 to offset positions of 3, 5, 6 and 8 units
(*	in 4.0 seconds.

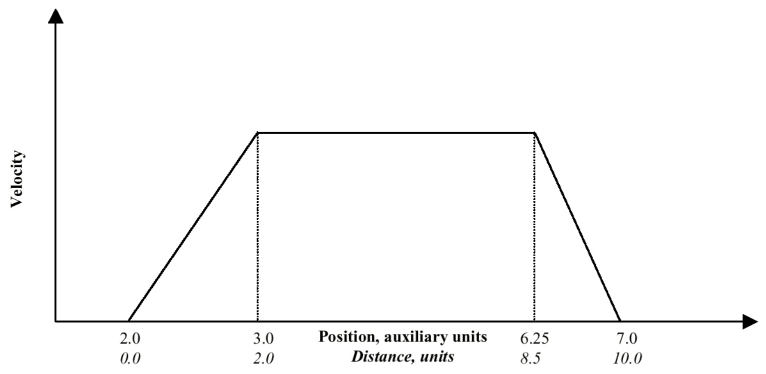
(* Notes:
(* 1- Registers that have been previously loaded with appropriate values
(*   do not have to be reloaded for this motion.
(* 2- Loading the MAP register also loads the MDP register with the same value. To set MDP
(*   to a value different from MAP, load MDP after loading the MAP register.
(* 3- This example assumes the MFP (Motion Feedrate Percentage) is set to its default value
(*   of 100. See files T_MVABSF.TXT and T_MT1ABF.TXT for examples using the MFP
(*   register.
(* 4- RPO moves the axes from their present positions to the offset positions specified in the
(*   MPO registers. This example begins by loading the offset position registers, PSO1
(*   through PSO4, with 0 for the purpose of accurately graphing the subsequent motion.
(*   Applications may require other offset position register values.

(* MT1 = TIME	
(* MT2 = TIME	
(* MT3 = TIME	
(* MT4 = TIME	The MT register cannot be loaded if motion is in progress.
(*	MT does not need to be set unless it is set to a MT setting other
(*	than TIME. The default value for MT is VEL.

```
(* Initialize axis 1 offset position registers to 0
    PSO1 = 0.0              (* set axis 1 offset position, units
    PSO2 = 0.0              (* set axis 2 offset position, units
    PSO3 = 0.0              (* set axis 3 offset position, units
    PSO4 = 0.0              (* set axis 4 offset position, units

(* Move axes to offset positions 3, 5, 6 and 8 with the accelerations and move times shown.
    MAP1 = 25               (* set motion accel percentage, % of move time
    MAP2 = 25               (* set motion accel percentage, % of move time
    MAP3 = 25               (* set motion accel percentage, % of move time
    MAP4 = 25               (* set motion accel percentage, % of move time
    MDP1 = 20               (* set motion decel percentage, % of move time
    MDP2 = 20               (* set motion decel percentage, % of move time
    MDP3 = 20               (* set motion decel percentage, % of move time
    MDP4 = 20               (* set motion decel percentage, % of move time
    MJK1 = 0                (* set motion jerk percentage, % of accel & decel interval
    MJK2 = 0                (* set motion jerk percentage, % of accel & decel interval
    MJK3 = 0                (* set motion jerk percentage, % of accel & decel interval
    MJK4 = 0                (* set motion jerk percentage, % of accel & decel interval
    MTM1 = 4.0              (* set move time, seconds
    MTM2 = 4.0              (* set move time, seconds
    MTM3 = 4.0              (* set move time, seconds
    MTM4 = 4.0              (* set move time, seconds
    MPO1 = 3.0              (* set axis 1 offset move position, units
    MPO2 = 5.0              (* set axis 2 offset move position, units
    MPO3 = 6.0              (* set axis 3 offset move position, units
    MPO4 = 8.0              (* set axis 4 offset move position, units
    RPO1234                 (* run axes 1, 2, 3 and 4 to offset move positions
```

Pulse-Based Moves

Pulse-Based, Single-Axis Incremental Move



```
(* Notes:
(* 1-  Registers that have been previously loaded with appropriate values
(*     do not have to be reloaded for this motion.
(* 2- Loading the MAP register also loads the MDP register with the same value.  To set MDP
(*     to a value different from MAP, load MDP after loading the MAP register.
(* 3-  This example assumes the MFP (Motion Feedrate Percentage) is set to its default value
(*     of 100. See IMC & IMJ files I_MVABSF.TXT and I_MT1ABF.TXT or Target files
(*     T_MVABSF.TXT and T_MT1ABF.TXT for examples using the MFP register.
(* 4- This example begins by loading the auxiliary position register (PSX) with 0 for the
(*     purpose of accurately depicting the motion.  In general, applications will load MPS with
(*     the appropriate starting position.
```

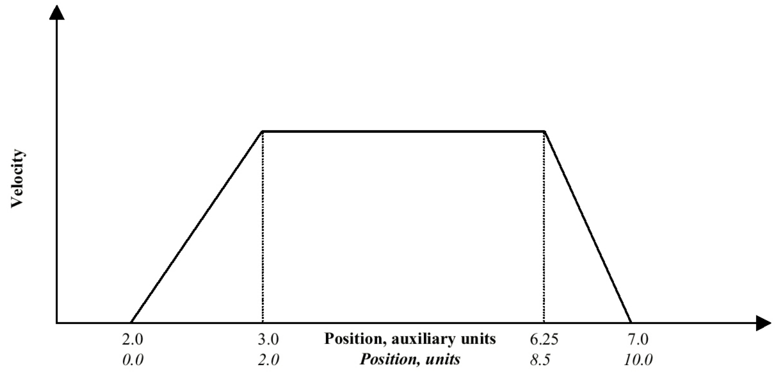
IMC & IMJ Template

```
(* Motion Template:        I_MP1INC.TXT
(* Revision Log:           REV 098OCT02
(* DspMotion Series:       IMC & IMJ
(* Move Type:              Pulse-based, single-axis incremental move
(* Engineering Units:      Motor revolutions: i.e., URA = position feedback resolution
(*                         URX = auxiliary feedback resolution

(* Motion:                 The axis will remain in position until the auxiliary position
(*                         increases to 2 aux units.  Then, as the aux position increases
(*                         from 2 to 7 aux units, the axis will run forward 10 axis units.
(* MT = PULSE              This register cannot be loaded if motion is in progress.
(*                         MT does not need to be set unless it is set to a MT setting other
(*                         than PULSE.  The default value for MT is VEL.

(* Initialize auxiliary position register to 0
     PSX = 0               (* set auxiliary position, aux units
(* Move axis 10 units as the auxiliary position goes from 2 to 7 units
     MAP = 20                  (* set motion acceleration percentage, % of move pulses
     MDP = 15                  (* set motion deceleration percentage, % of move pulses
     MPS = 2.0                 (* set motion start position, aux units
     MPL = 5.0                 (* set move pulses, aux units
     MPI = 10.0                (* set incremental move position, units
     RPI                       (* run to incremental move position
```

Target Template

```
(* Motion Template:         T_MP1INC.TXT
(* Revision Log:            REV 098OCT02
(* DspMotion Series:        Target ARS
(* Move Type:               Pulse-based, single-axis incremental move
(* Engineering Units:       Motor revolutions: i.e., URA1 = position feedback resolution
(*                          URX1 = auxiliary feedback resolution

(* Motion:                  Axis 1 will remain in position until the auxiliary position
(*                          increases to 2 aux units.  Then, as the aux position increases
(*                          from 2 to 7 aux units, the axis will run forward 10 axis units.
(* MT1 = PULSE              This register cannot be loaded if motion is in progress.
(*                          MT does not need to be set unless it is set to a MT setting other
(*                          than PULSE.  The default value for MT is VEL.

(* Initialize auxiliary position register to 0
      PSX1 = 0                  (* set auxiliary position, aux units

(* Move axis 1 10 units as the auxiliary position goes from 2 to 7 units
      MI1 = PSX1                (* set motion pulse input to axis 1 aux input
      MAP1 = 20                 (* set motion acceleration percentage, % of move pulses
      MDP1 = 15                 (* set motion deceleration percentage, % of move pulses
      MPS1 = 2.0                (* set motion start position, aux units
      MPL1 = 5.0                (* set move pulses, aux units
      MPI1 = 10.0               (* set incremental move position, units
      RPI1                      (* run axis 1 to incremental move position
```

Pulse-Based, Single-Axis Absolute Move



(* Notes:
(* 1- Registers that have been previously loaded with appropriate values
(*   do not have to be reloaded for this motion.
(* 2- Loading the MAP register also loads the MDP register with the same value. To set MDP
(*   to a value different from MAP, load MDP after loading the MAP register.
(* 3- This example assumes the MFP (Motion Feedrate Percentage) is set to its default value
(*   of 100. See IMC & IMJ files I_MVABSF.TXT and I_MT1ABF.TXT or Target files
(*   T_MVABSF.TXT and T_MT1ABF.TXT for examples using the MFP register.
(* 4- RPA moves the axis from its present position to the absolute position specified in the
(*   MPA register. This example begins by loading the absolute position register, PSA, with
(*   0 for the purpose of accurately graphing the subsequent motion. In general, applications
(*   will only load PSA at the end of a homing motion.
(*5- This example loads the auxiliary position register (PSX) with 0 for the purpose of
(*   accurately depicting the motion. In general, applications will load MPS with the
(*   appropriate starting position.

IMC & IMJ Template

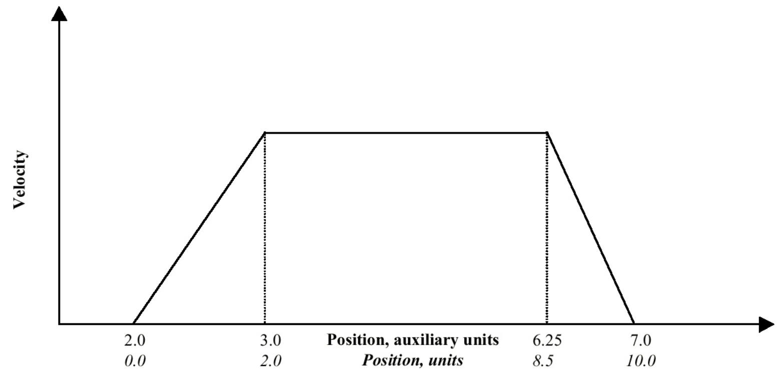
(* Motion Template:	I_MP1ABS.TXT
(* Revision Log:	REV 098OCT02
(* DspMotion Series:	IMC & IMJ
(* Move Type:	Pulse-based, single-axis absolute move
(* Engineering Units:	Motor revolutions: i.e., URA = position feedback resolution
(*	URX = auxiliary feedback resolution
(* Motion:	The axis will remain in position until the auxiliary position
(*	increases to 2 aux units. Then, as the aux position increases
(*	from 2 to 7 aux units, the axis will run to an absolute position
(*	of 10 units.
(* MT = PULSE	This register cannot be loaded if motion is in progress.
(*	MT does not need to be set unless it is set to a MT setting other
(*	than PULSE. The default value for MT is VEL.

```
(* Initialize absolute position register to 0
     PSA = 0.0                 (* set absolute position, units

(* Initialize auxiliary position register to 0
     PSX = 0                   (* set auxiliary position, aux units

(* Move axis to absolute position 10 with the accelerations and move pulses shown.
     MAP = 20                  (* set motion acceleration percentage, % of move pulses
     MDP = 15                  (* set motion deceleration percentage, % of move pulses
     MPS = 2.0                 (* set motion start position, aux units
     MPL = 5.0                 (* set move pulses, aux units
     MPA = 10.0                (* set absolute move position, units
     RPA                       (* run to absolute move position
```

Target Template

```
(* Motion Template:            T_MP1ABS.TXT
(* Revision Log:               REV 098OCT02
(* DspMotion Series:           Target ARS
(* Move Type:                  Pulse-based, single-axis absolute move
(* Engineering Units:          Motor revolutions: i.e., URA1 = position feedback resolution
(*                             URX1 = auxiliary feedback resolution

(* Motion:                     Axis 1 will remain in position until the auxiliary position
(*                             increases to 2 aux units.  Then, as the aux position increases
(*                             from 2 to 7 aux units, the axis will run to an absolute position
(*                             of 10 units.
(* MT1 = PULSE                 This register cannot be loaded if motion is in progress.
(*                             MT does not need to be set unless it is set to a MT setting other
(*                             than PULSE.  The default value for MT is VEL.

(* Initialize axis 1 absolute position register to 0
     PSA1 = 0.0                (* set axis 1 absolute position, units

(* Initialize auxiliary position register to 0
     PSX1 = 0                  (* set auxiliary position, aux units

(* Move axis 1 to absolute position 10 with the accelerations and move pulses shown
     MI1 = PSX1                (* set motion pulse input to axis 1 aux input
     MAP1 = 20                 (* set motion acceleration percentage, % of move pulses
     MDP1 = 15                 (* set motion deceleration percentage, % of move pulses
     MPS1 = 2.0                (* set motion start position, aux units
     MPL1 = 5.0                (* set move pulses, aux units
     MPA1 = 10.0               (* set axis 1 absolute move position, units
     RPA1                      (* run axis 1 to absolute move position
```

Pulse-Based, Single-Axis Offset Move



```
(* Notes:
(* 1- Registers that have been previously loaded with appropriate values
(*    do not have to be reloaded for this motion.
(* 2- Loading the MAP register also loads the MDP register with the same value.  To set MDP
(*    to a value different from MAP, load MDP after loading the MAP register.
(* 3- This example assumes the MFP (Motion Feedrate Percentage) is set to its default value
(*    of 100.  See IMC & IMJ files I_MVABSF.TXT and I_MT1ABF.TXT or Target files
(*    T_MVABSF.TXT and T_MT1ABF.TXT for examples using the MFP register.
(* 4- RPO moves the axis from its present position to the offset position specified in the
(*    MPO register.  This example begins by loading the offset position register, PSO, with
(*    0 for the purpose of accurately graphing the subsequent motion. Applications may
(*    require other offset position register values.
(*5-  This example begins by loading the auxiliary position register (PSX) with 0 for the
(*    purpose of accurately depicting the motion.  In general, applications will load MPS with
(*    the appropriate starting position.
```

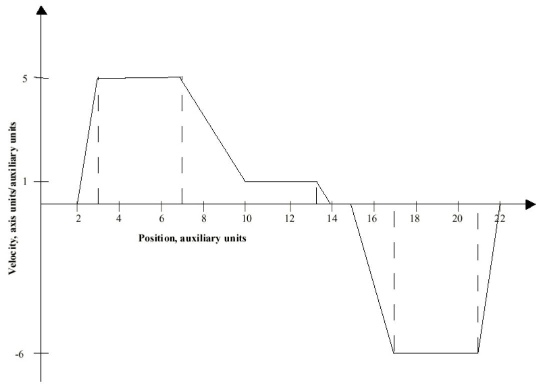
IMC & IMJ Template

```
(* Motion Template:        I_MP1OFF.TXT
(* Revision Log:           REV 098OCT02
(* DspMotion Series:       IMC & IMJ
(* Move Type:              Pulse-based, single-axis offset move
(* Engineering Units:      Motor revolutions: i.e., URA = position feedback resolution
(*                         URX = auxiliary feedback resolution

(* Motion:                 The axis will remain in position until the auxiliary position
(*                         increases to 2 aux units.  Then, as the aux position increases
(*                         from 2 to 7 aux units, the axis will run to an offset position
(*                         of 10 units.

(* MT = PULSE              This register cannot be loaded if motion is in progress.
(*                         MT does not need to be set unless it is set to a MT setting other
(*                         than PULSE.  The default value for MT is VEL.
(* Initialize offset position register to 0
     PSO = 0.0                (* set offset position, units

(* Initialize auxiliary position register to 0
     PSX = 0                  (* set auxiliary position, aux units

(* Move axis to offset position 10 with the accelerations and move pulses shown.
     MAP = 20                 (* set motion acceleration percentage, % of move pulses
     MDP = 15                 (* set motion deceleration percentage, % of move pulses
     MPS = 2.0                (* set motion start position, aux units
     MPL = 5.0                (* set move pulses, aux units
     MPO = 10.0               (* set offset move position, units
     RPO                      (* run to offset move position
```

Target Template

```
(* Motion Template:           T_MP1OFF.TXT
(* Revision Log:              REV 098OCT02
(* DspMotion Series:          Target ARS
(* Move Type:                 Pulse-based, single-axis offset move
(* Engineering Units:         Motor revolutions: i.e., URA1 = position feedback resolution
(*                            URX1 = auxiliary feedback resolution

(* Motion:                    Axis 1 will remain in position until the auxiliary position
(*                            increases to 2 aux units.  Then, as the aux position increases
(*                            from 2 to 7 aux units, the axis will run to an absolute position
(*                            of 10 units.
(* MT1 = PULSE                This register cannot be loaded if motion is in progress.
(*                            MT does not need to be set unless it is set to a MT setting other
(*                            than PULSE.  The default value for MT is VEL.

(* Initialize axis 1 offset position register to 0
     PSO1 = 0.0               (* set axis 1 offset position, units

(* Initialize auxiliary position register to 0
     PSX1 = 0                 (* set auxiliary position, aux units

(* Move axis 1 to offset position 10 with the accelerations and move pulses shown
     MI1 = PSX1               (* set motion pulse input to axis 1 aux input
     MAP1 = 20                (* set motion acceleration percentage, % of move pulses
     MDP1 = 15                (* set motion deceleration percentage, % of move pulses
     MPS1 = 2.0               (* set motion start position, aux units
     MPL1 = 5.0               (* set move pulses, aux units
     MPO1 = 10.0              (* set axis 1 offset move position, units
     RPO1                     (* run axis 1 to offset move position
```

Pulse-Based, Single-Axis Blended Move



```
(* Notes:
(* 1-  Registers that have been previously loaded with appropriate values need not be reloaded
(*     for this motion.
(* 2-  This example assumes the MFP (Motion Feedrate Percentage) is set to its default value
(*     of 100.  See  IMC & IMJ files I_MVABSF.TXT and I_MT1ABF.TXT and Target files
(*     T_MVABSF.TXT and T_MT1ABF.TXT for examples using the MFP register.
(* 3-  This example begins by loading the auxiliary position register with 0 for the purpose of
(*     accurately depicting the motion.  In general, applications will load MPS with the
(*     appropriate starting position.
(* 4-  RPA moves the axis from its present position to the absolute position specified in the
(*     MPA register.  This example begins by loading the absolute position register, PSA, with
(*     0 for the purpose of accurately graphing the subsequent motion.  In general, applications
(*     will load PSA only at the end of a homing motion.
(* 5-  Blended moves are specified by setting a new velocity in the instruction immediately
(*     following a run command AND CAN BE DONE ONLY IN MOTION  BLOCKS!
```

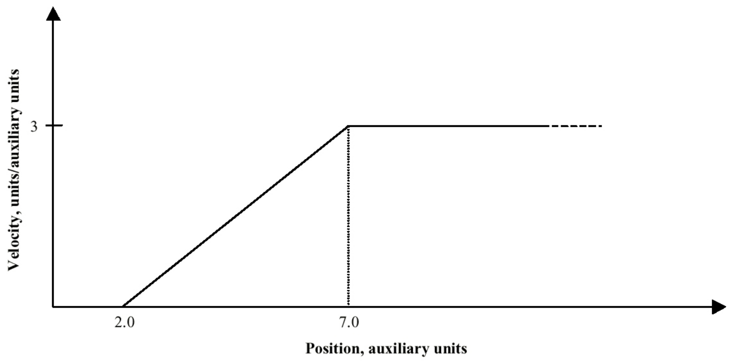
IMC & IMJ Template

```
(* Motion Template:        I_MP1BLN.TXT
(* Revision Log:           REV 098OCT02
(* DspMotion Series:       IMC & IMJ
(* Move Type:              Pulse-based blended move
(* Engineering Units:      Motor revolutions: i.e., URA = position feedback resolution
(*                         URX = auxiliary feedback resolution

(* Motion:                 The axis (axis 1 for Target) will remain in position until the
(*                         auxiliary position increases to 2 aux units.  Then, as the aux
(*                         position increases from 2 to 10 aux units, the axis will run
(*                         forward to 30 axis units.  As the aux position further increases
(*                         to 14 aux units, the axis will finish running forward to 34 axis
(*                         units. Finally, as the aux position increases from 15 to 22 aux units,
(*                         the axis will move back to position 0 axis units.
(* MT = PULSE              This register cannot be loaded if motion is in progress.
(*                         MT does not need to be set unless it is set to a MT setting other
(*                         than PULSE.  The default value for MT is VEL.

(* Initialize auxiliary position register to 0.
    PSX = 0                (* set auxiliary position, aux units

(* Initialize absolute position register to 0
    PSA = 0.0              (* set absolute position, units

(* Execute blended move with the accelerations and velocities shown.
    MAP = 20               (* set motion acceleration percentage, % of move pulses
    MDP = 15               (* set motion deceleration percentage, % of move pulses
    MPS = 2.0              (* set motion start position, aux units
    MPL = 8.0              (* set move pulses, aux units
    MPA = 30.0             (* set absolute move position, units
    RPA                    (* run to absolute move position
    MVP = 1.0              (* set motion velocity of pulse move, axis units/aux units
    MPS = MPS + 8.0        (* set motion start position, aux units
    MPL = 4.0              (* set move pulses, aux units
    MPA = 34.0             (* set absolute move position, units
    RPA                    (* run to absolute move position
    MPS = MPS + 5.0        (* set motion start position, aux units
    MPL = 7.0              (* set move pulses, aux units
    MPA = 0.0              (* set absolute move position, units
    RPA                    (* run to absolute move position
```

Target Template

```
(* Motion Template:        T_MP1BLN.TXT
(* Revision Log:           REV 098OCT02
(* DspMotion Series:       Target ARS
(* Move Type:              Pulse-based blended move
(* Engineering Units:      Motor revolutions: i.e., URA1 = position feedback resolution
(*                         URX1 = auxiliary feedback resolution

(* MT1 = PULSE             This register cannot be loaded if motion is in progress.
(*                         MT does not need to be set unless it is set to a MT setting other
(*                         than PULSE.  The default value for MT is VEL.

(* Initialize auxiliary position register to 0.
    PSX1 = 0               (* set auxiliary position, aux units

(* Initialize absolute position register to 0
    PSA1 = 0.0             (* set absolute position, units
(* Execute blended move with the accelerations and velocities shown.
    MI1 = PSX1              (* set motion pulse input to axis 1 aux input
    MAP1 = 20               (* set motion acceleration percentage, % of move pulses
    MDP1 = 15               (* set motion deceleration percentage, % of move pulses
    MPS1 = 2.0              (* set motion start position, aux units
    MPL1 = 8.0              (* set move pulses, aux units
    MPA1 = 30.0             (* set absolute move position, units
    RPA1                    (* run axis 1 to absolute move position
    MVP1 = 1.0              (* set motion velocity of pulse move, axis units/aux units
    MPS1 = MPS1 + 8.0       (* set motion start position, aux units
    MPL1 = 4.0              (* set move pulses, aux units
    MPA1 = 34.0             (* set absolute move position, units
    RPA1                    (* run axis 1 to absolute move position
    MPS1 = MPS1 + 5.0       (* set motion start position, aux units
    MPL1 = 7.0              (* set move pulses, aux units
    MPA1 = 0.0              (* set absolute move position, units
    RPA1                    (* run axis 1 to absolute move position
```

Pulse-Based, Single-Axis Continuous Move



(* Notes:
(* 1- Registers that have been previously loaded with appropriate values do not have to be
(*   reloaded for this motion.
(* 2- This example assumes the MFP (Motion Feedrate Percentage) is set to its default value
(*   of 100. See IMC & IMJ files I_MVABSF.TXT and I_MT1ABF.TXT or Target files
(*   T_MVABSF.TXT and T_MT1ABF.TXT for examples using the MFP register.
(* 3- This example begins by loading the auxiliary position register (PSX) with 0 for the
(*   purpose of accurately depicting the motion. In general, applications will load MPS with
(*   the appropriate starting position.

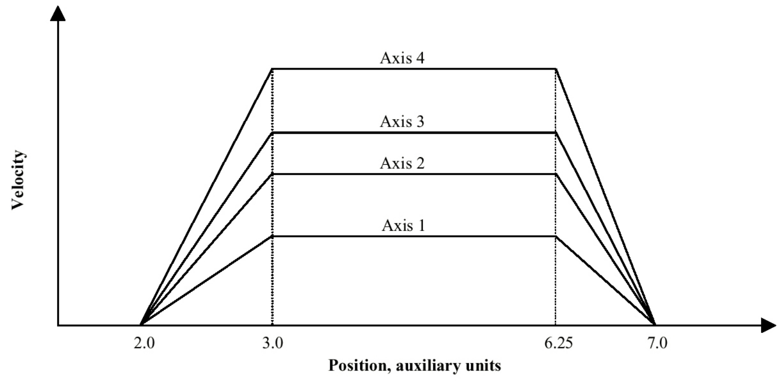
IMC & IMJ Template

```
(* Motion Template:        I_MP1CON.TXT
(* Revision Log:           REV 098OCT02
(* DspMotion Series:       IMC & IMJ
(* Move Type:              Pulse-based, single-axis continuous move
(* Engineering Units:      Motor revolutions: i.e., URA = position feedback resolution
(*                         URX = auxiliary feedback resolution

(* Motion:                 The axis will remain in position until the auxiliary position
(*                         increases to 2 aux units. Then, as the aux position increases
(*                         from 2 to 7 aux units, the axis will accelerate to 3 units/aux
(*                         units.
(* MT = PULSE              This register cannot be loaded if motion is in progress.
(*                         MT does not need to be set unless it is set to a MT setting other
(*                         than PULSE. The default value for MT is VEL.

(* Initialize auxiliary position register to 0
    PSX = 0                (* set auxiliary position, aux units
(* Execute single-axis continuous move with the accelerations and move pulses shown
    MPS = 2.0                 (* set motion start position, aux units
    MPL = 5.0                 (* set move pulses, aux units
    MVP = 3.0                 (* set pulse move velocity, units/aux units
    RVF                       (* run forward
```

Target Template

```
(* Motion Template:         T_MP1CON.TXT
(* Revision Log:            REV 098OCT02
(* DspMotion Series:        Target ARS
(* Move Type:               Pulse-based, single-axis continuous move
(* Engineering Units:       Motor revolutions: i.e., URA1 = position feedback resolution
(*                          URX1 = auxiliary feedback resolution

(* Motion:                  Axis 1 will remain in position until the auxiliary position
(*                          increases to 2 aux units.  Then, as the aux position increases
(*                          from 2 to 7 aux units, the axis will accelerate to 3 units/aux
(*                          units.
(* MT1 = PULSE              This register cannot be loaded if motion is in progress.
(*                          MT does not need to be set unless it is set to a MT setting other
(*                          than PULSE.  The default value for MT is VEL.

(* Initialize auxiliary position register to 0
      PSX1 = 0                  (* set auxiliary position, aux units

(* Execute single-axis continuous move with the accelerations and move pulses shown
      MI1 = PSX1                (* set motion pulse input to axis 1 aux input
      MPS1 =2.0                 (* set motion start position, aux units
      MPL1 = 5.0                (* set move pulses, aux units
      MVP1 = 3.0                (* set pulse move velocity, units/aux units
      RVF1                      (* run axis 1 forward
```

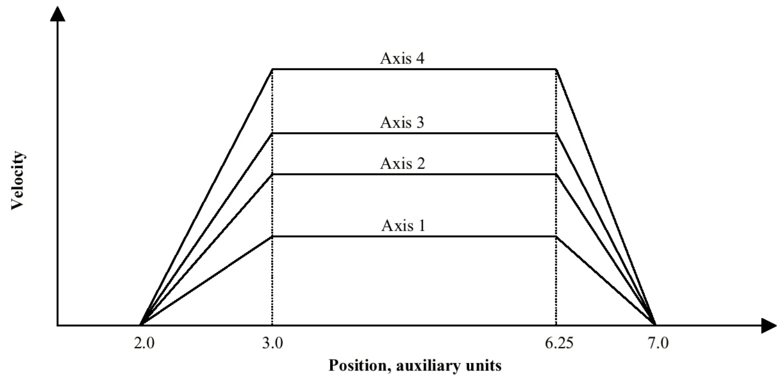
Pulse-Based, Multi-axis Incremental Move ⊙



Target Template

```
(* Motion Template:          T_MP4INC.TXT
(* Revision Log:             REV 098OCT02
(* DspMotion Series:         Target ARS
(* Move Type:                Pulse-based, multi-axis incremental move
(* Engineering Units:        Motor revolutions: i.e., URA1 = position feedback resolution
(*                           URX1 = auxiliary feedback resolution
(*                           URA2 = position feedback resolution, etc.

(* Motion:                   Each axis will remain in position until the auxiliary position of
(*                           axis 1 increases to 2 aux units.  Then, as the aux position
(*                           increases from 2 to 7 aux units, axes 1, 2, 3 and 4 will run
(*                           forward by 8, 10, 11 and 13 units.

(* Notes:
(* 1-  Registers that have been previously loaded with appropriate values
(*     do not have to be reloaded for this motion.
(* 2-  Loading the MAP register also loads the MDP register with the same value.  To set MDP
(*     to a value different from MAP, load MDP after loading the MAP register.
(* 3-  This example assumes the MFP (Motion Feedrate Percentage) is set to its default value
(*     of 100.  See files T_MVABSF.TXT and T_MT1ABF.TXT for examples using the MFP
(*     register.
(* 4-  This example begins by loading the auxiliary position register (PSX) with 0 for the
(*     purpose of accurately depicting the motion.  In general, applications will load MPS with
(*     the appropriate starting position.

(* MT1 = PULSE
(* MT2 = PULSE
(* MT3 = PULSE
(* MT4 = PULSE               The MT  register cannot be loaded if motion is in progress.
(*                           MT does not need to be set unless it is set to a MT setting other
(*                           than PULSE.  The default value for MT is VEL.
(* Initialize auxiliary position register to 0
    PSX1 = 0                (* set auxiliary position, aux units

(* Execute multi-axis incremental move with the accelerations and move pulses shown
    MI1 = PSX1              (* set motion pulse input to axis 1 aux input
    MI2 = PSX1              (* set motion pulse input to axis 1 aux input
    MI3 = PSX1              (* set motion pulse input to axis 1 aux input
    MI4 = PSX1              (* set motion pulse input to axis 1 aux input
    MAP1 = 20               (* set motion acceleration percentage, % of move pulses
    MAP2 = 20               (* set motion acceleration percentage, % of move pulses
    MAP3 = 20               (* set motion acceleration percentage, % of move pulses
    MAP4 = 20               (* set motion acceleration percentage, % of move pulses
    MDP1 = 15               (* set motion deceleration percentage, % of move pulses
    MDP2 = 15               (* set motion deceleration percentage, % of move pulses
    MDP3 = 15               (* set motion deceleration percentage, % of move pulses
    MDP4 = 15               (* set motion deceleration percentage, % of move pulses
    MPS1 = 2.0              (* set motion start position, aux units
    MPS2 = 2.0              (* set motion start position, aux units
    MPS3 = 2.0              (* set motion start position, aux units
    MPS4 = 2.0              (* set motion start position, aux units
    MPL1 = 5.0              (* set move pulses, aux units
    MPL2 = 5.0              (* set move pulses, aux units
    MPL3 = 5.0              (* set move pulses, aux units
    MPL4 = 5.0              (* set move pulses, aux units
    MPI1 = 8.0              (* set axis 1 incremental move position, units
    MPI2 = 10.0             (* set axis 2 incremental move position, units
    MPI3 = 11.0             (* set axis 3 incremental move position, units
    MPI4 = 13.0             (* set axis 4 incremental move position, units
    RPI1234                 (* run axes 1, 2, 3 and 4 to incremental move positions
```

Pulse-Based, Multi-axis Absolute Move ⊙



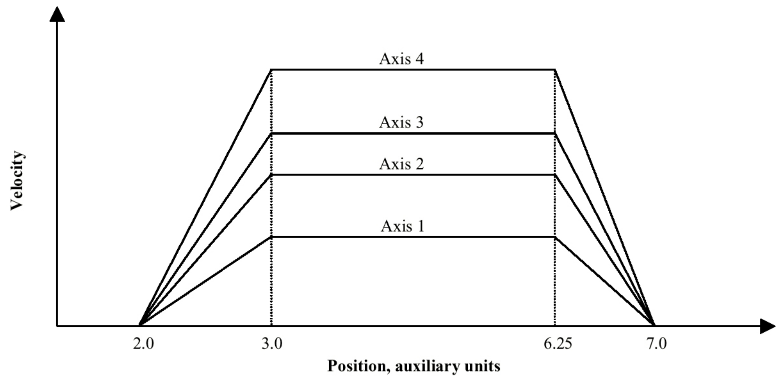
Target Template

(* Motion Template:	T_MP4ABS.TXT
(* Revision Log:	REV 098OCT02
(* DspMotion Series:	Target ARS
(* Move Type:	Pulse-based, multi-axis absolute move
(* Engineering Units:	Motor revolutions: i.e., URA1 = position feedback resolution
(*	URX1 = auxiliary feedback resolution,
(*	URA2 = position feedback resolution, etc.
(* Motion:	Each axis will remain in position until the auxiliary position of
(*	axis 1 increases to 2 aux units. Then, as the aux position
(*	increases from 2 to 7 aux units, axes 1, 2, 3 and 4 will run to
(*	absolute positions of 8, 10, 11 and 13 units.

(* Notes:
(* 1- Registers that have been previously loaded with appropriate values
(*   do not have to be reloaded for this motion.
(* 2- Loading the MAP register also loads the MDP register with the same value. To set MDP
(*   to a value different from MAP, load MDP after loading the MAP register.
(* 3- This example assumes the MFP (Motion Feedrate Percentage) is set to its default value
(*   of 100. See files T_MVABSF.TXT and T_MT1ABF.TXT for examples using the MFP
(*   register.
(* 4- RPA moves the axes from their present positions to the absolute positions specified in
(*   the MPA registers. This example begins by loading the absolute position registers, PSA1
(*   through PSA4, with 0 for the purpose of accurately graphing the subsequent motion. In
(*   general, applications will only load PSA at the end of a homing motion.
(* 5- This example begins by loading the auxiliary position register (PSX) with 0 for the
(*   purpose of accurately depicting the motion. In general, applications will load MPS with
(*   the appropriate starting position.

```
(* MT1 = PULSE
(* MT2 = PULSE
(* MT3 = PULSE
(* MT4 = PULSE             The MT register cannot be loaded if motion is in progress.
(*                         MT does not need to be set unless it is set to a MT setting other
(*                         than PULSE.  The default value for MT is VEL.

(* Initialize absolute position registers to 0
    PSA1 = 0.0             (* set axis 1 absolute position, units
    PSA2 = 0.0             (* set axis 2 absolute position, units
    PSA3 = 0.0             (* set axis 3 absolute position, units
    PSA4 = 0.0             (* set axis 4 absolute position, units

(* Initialize auxiliary position register to 0
    PSX1 = 0               (* set axis 1 auxiliary position, aux units

(* Move axes to absolute positions 8, 10, 11 and 13 with the accelerations and move pulses
(* shown.
    MI1 = PSX1             (* set motion pulse input to axis 1 aux input
    MI2 = PSX1             (* set motion pulse input to axis 1 aux input
    MI3 = PSX1             (* set motion pulse input to axis 1 aux input
    MI4 = PSX1             (* set motion pulse input to axis 1 aux input
    MAP1 = 20              (* set motion acceleration percentage, % of move pulses
    MAP2 = 20              (* set motion acceleration percentage, % of move pulses
    MAP3 = 20              (* set motion acceleration percentage, % of move pulses
    MAP4 = 20              (* set motion acceleration percentage, % of move pulses
    MDP1 = 15              (* set motion deceleration percentage, % of move pulses
    MDP2 = 15              (* set motion deceleration percentage, % of move pulses
    MDP3 = 15              (* set motion deceleration percentage, % of move pulses
    MDP4 = 15              (* set motion deceleration percentage, of move pulses
    MPS1 = 2.0             (* set motion start position, aux units
    MPS2 = 2.0             (* set motion start position, aux units
    MPS3 = 2.0             (* set motion start position, aux units
    MPS4 = 2.0             (* set motion start position, aux units
    MPL1 = 5.0             (* set move pulses, aux units
    MPL2 = 5.0             (* set move pulses, aux units
    MPL3 = 5.0             (* set move pulses, aux units
    MPL4 = 5.0             (* set move pulses, aux units
    MPA1 = 8.0             (* set axis 1 absolute move position, units
    MPA2 = 10.0            (* set axis 2 absolute move position, units
    MPA3 = 11.0            (* set axis 3 absolute move position, units
    MPA4 = 13.0            (* set axis 4 absolute move position, units
    RPA1234                (* run axes 1, 2, 3 and 4 to absolute move positions
```

Pulse-Based, Multi-axis Offset Move ⊙

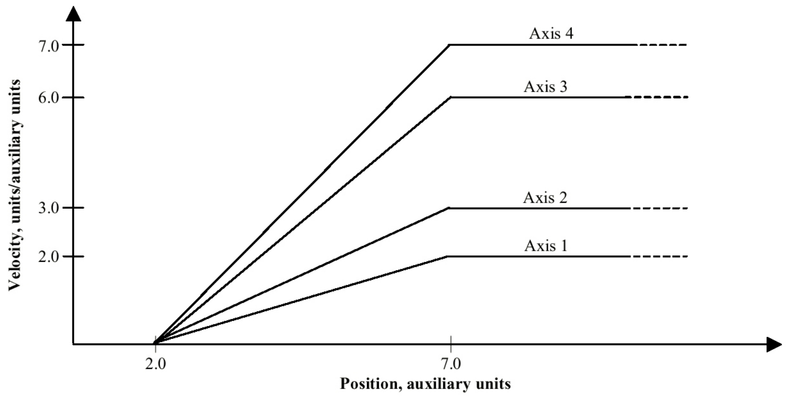


Target Template

(* Motion Template:	T_MP4OFF.TXT
(* Revision Log:	REV 098OCT02
(* DspMotion Series:	Target ARS
(* Move Type:	Pulse-based, multi-axis offset move
(* Engineering Units:	Motor revolutions: i.e., URA1 = position feedback resolution
(*	URX1 = auxiliary feedback resolution
(*	URA2 = position feedback resolution, etc.
(* Motion:	Each axis will remain in position until the auxiliary position of
(*	axis 1 increases to 2 aux units. Then, as the aux position
(*	increases from 2 to 7 aux units, axes 1, 2, 3 and 4 will run to
(*	offset positions of 8, 10, 11 and 13 units.

```
(* Notes:
(* 1- Registers that have been previously loaded with appropriate values
(*    do not have to be reloaded for this motion.
(* 2- Loading the MAP register also loads the MDP register with the same value. To set MDP
(*    to a value different from MAP, load MDP after loading the MAP register.
(* 3-  This example assumes the MFP (Motion Feedrate Percentage) is set to its default value
(*    of 100. See files T_MVABSF.TXT and T_MT1ABF.TXT for examples using the MFP
(*    register.
(* 4-  RPO moves the axes from their present positions to the offset positions specified in the
(*    MPO registers. This example begins by loading the offset position registers, PSO1
(*    through PSO4, with 0 for the purpose of accurately graphing the subsequent motion.
(*    Applications may require other offset position register values.
(* 5- This example begins by loading the auxiliary position register (PSX) with 0 for the
(*    purpose of accurately depicting the motion. In general, applications will load MPS with
(*    the appropriate starting position.
(* MT1 = PULSE
(* MT2 = PULSE
(* MT3 = PULSE
(* MT4 = PULSE            The MT register cannot be loaded if motion is in progress.
(*                        MT does not need to be set unless it is set to a MT setting other than
(*                        PULSE.  The default value for MT is VEL.

(* Initialize axis 1 offset position register to 0
    PSO1 = 0.0            (* set axis 1 offset position, units
    PSO2 = 0.0            (* set axis 2 offset position, units
    PSO3 = 0.0            (* set axis 3 offset position, units
    PSO4 = 0.0            (* set axis 4 offset position, units

(* Initialize auxiliary position register to 0
    PSX1 = 0              (* set auxiliary position, aux units

(* Move axes to offset positions 8, 10, 11 and 13 with the accelerations and move pulses
(* shown.
    MI1 = PSX1            (* set motion pulse input to axis 1 aux input
    MI2 = PSX1            (* set motion pulse input to axis 1 aux input
    MI3 = PSX1            (* set motion pulse input to axis 1 aux input
    MI4 = PSX1            (* set motion pulse input to axis 1 aux input
    MAP1 = 20             (* set motion acceleration percentage, % of move pulses
    MAP2 = 20             (* set motion acceleration percentage, % of move pulses
    MAP3 = 20             (* set motion acceleration percentage, % of move pulses
    MAP4 = 20             (* set motion acceleration percentage, % of move pulses
    MDP1 = 15             (* set motion deceleration percentage, % of move pulses
    MDP2 = 15             (* set motion deceleration percentage, % of move pulses
    MDP3 = 15             (* set motion deceleration percentage, % of move pulses
    MDP4 = 15             (* set motion deceleration percentage, % of move pulses
    MPS1 = 2.0            (* set motion start position, aux units
    MPS2 = 2.0            (* set motion start position, aux units
    MPS3 = 2.0            (* set motion start position, aux units
    MPS4 = 2.0            (* set motion start position, aux units
    MPL1 = 5.0            (* set move pulses, aux units
    MPL2 = 5.0            (* set move pulses, aux units
    MPL3 = 5.0            (* set move pulses, aux units
    MPL4 = 5.0            (* set move pulses, aux units
    MPO1 = 8.0            (* set axis 1 offset move position, units
    MPO2 = 10.0           (* set axis 2 offset move position, units
    MPO3 = 11.0           (* set axis 3 offset move position, units
    MPO4 = 13.0           (* set axis 4 offset move position, units
    RPO1234               (* run axes 1, 2, 3 and 4 to offset move positions
```

Pulse-Based, Multi-axis Continuous Move ⊙

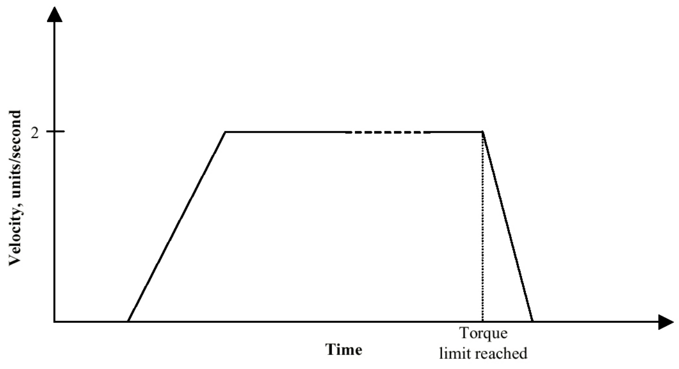


Target Template

```
(* Motion Template:         T_MP4CON.TXT
(* Revision Log:            REV 098OCT02
(* DspMotion Series:        Target ARS
(* Move Type:               Pulse-based, multi-axis continuous move
(* Engineering Units:       Motor revolutions: i.e., URA1 = position feedback resolution
(*                          URX1 = auxiliary feedback resolution
(*                          URA2 = position feedback resolution, etc.

(* Motion:                  Each axis will remain in position until the auxiliary position of
(*                          axis 1 increases to 2 aux units.  Then, as the aux position
(*                          increases from 2 to 7 aux units, axes 1, 2, 3 and 4 will
(*                          accelerate to 2, 3, 6 and 7 units/aux units.

(* Notes:
(* 1- Registers that have been previously loaded with appropriate values
(*    do not have to be reloaded for this motion.
(* 2- This example assumes the MFP (Motion Feedrate Percentage) is set to its default value
(*    of 100.  See files T_MVABSF.TXT and T_MT1ABF.TXT for examples using the MFP
(*    register.
(* 3- This example begins by loading the auxiliary position register (PSX) with 0 for the
(*    purpose of accurately depicting the motion.  In general, applications will load MPS with
(*    the appropriate starting position.
(* MT1 = PULSE
(* MT2 = PULSE
(* MT3 = PULSE
(* MT4 = PULSE              The  MT register cannot be loaded if motion is in progress.
(*                          MT does not need to be set unless it is set to a MT setting other
(*                          than PULSE. The default value for MT is VEL.

(* Initialize auxiliary position register to 0
    PSX1 = 0                (* set auxiliary position, aux units

(* Execute multi-axis continuous move with the accelerations and move pulses shown
    MI1 = PSX1              (* set motion pulse input to axis 1 aux input
    MI2 = PSX1              (* set motion pulse input to axis 1 aux input
    MI3 = PSX1              (* set motion pulse input to axis 1 aux input
    MI4 = PSX1              (* set motion pulse input to axis 1 aux input
    MPS1 = 2.0              (* set motion start position, aux units
    MPS2 = 2.0              (* set motion start position, aux units
    MPS3 = 2.0              (* set motion start position, aux units
    MPS4 = 2.0              (* set motion start position, aux units
    MPL1 = 5.0              (* set move pulses, aux units
    MPL2 = 5.0              (* set move pulses, aux units
    MPL3 = 5.0              (* set move pulses, aux units
    MPL4 = 5.0              (* set move pulses, aux units
    MVP1 = 2.0              (* set pulse move velocity, units/aux units
    MVP2 = 3.0              (* set pulse move velocity, units/aux units
    MVP3 = 6.0              (* set pulse move velocity, units/aux units
    MVP4 = 7.0              (* set pulse move velocity, units/aux units
    RVF1234                 (* run axis 1, 2, 3 and 4 forward
```

Torque-Limited Moves

Run Forward until Torque Limit



```
(* Notes:
(* 1- Registers that have been previously loaded with appropriate values
(*    do not have to be reloaded for this motion.
(* 2- Loading the MAC register also loads the MDC register with the same value.  To set
(*    MDC to a value different from MAC, load MDC after loading the MAC register.
(* 3- This example assumes the MFP (Motion Feedrate Percentage) is set to its default value
(*    of 100.  See IMC & IMJ files I_MVABSF.TXT and I_MT1ABF.TXT or Target files
(*    T_MVABSF.TXT and T_MT1ABF.TXT for examples using the MFP register.
(* 4- If this template is executed within a motion block, the WAIT TL command is not
(*    executed until the acceleration portion of the RVF command is complete.
```

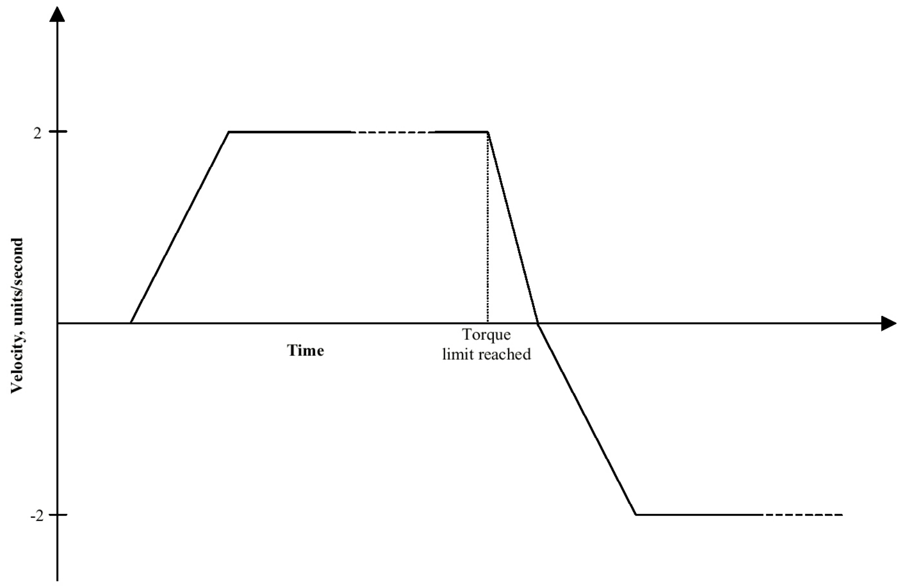
IMC & IMJ Template

```
(* Motion Template:       I_MTQFOR.TXT
(* Revision Log:          REV 098OCT02
(* DspMotion Series:      IMC & IMJ
(* Move Type:             Run forward until torque limit
(* Engineering Units:     Motor revolutions: i.e., URA = position feedback resolution

(* Motion:                Run forward until torque limit.
(* MT = VEL               This register cannot be loaded if motion is in progress.
(*                        MT does not need to be set unless it is set to a MT setting other
(*                        than VEL.  The default value for MT is VEL.

    MAC = 50.0            (* set motion acceleration, units/sec^2
    MDC = 75.0            (* set motion deceleration, units/sec^2
    MJK = 0               (* set motion jerk percentage, % of accel & decel interval
    MVL = 2.0             (* set motion velocity, units/sec
    TLC = 10.0            (* set torque limit current, % of continuous current
    TLE = ON              (* enable torque limit
    RVF                   (* run forward
    WAIT TL                 (* wait for axis to be at torque limit
    ST                      (* stop all motion
    TLE = OFF               (* disable torque limit
```

Target Template

```
(* Motion Template:         T_MTQFOR.TXT
(* Revision Log:            REV 098OCT02
(* DspMotion Series:        Target ARS
(* Move Type:               Run forward until torque limit
(* Engineering Units:       Motor revolutions: i.e., URA1 = position feedback resolution

(* Motion:                  Run axis 1 forward until torque limit.
(* MT1 = VEL                This register cannot be loaded if motion is in progress.
(*                          MT does not need to be set unless it is set to a MT setting other
(*                          than VEL.  The default value for MT is VEL.

    MAC1 = 50.0             (* set motion acceleration, units/sec^2
    MDC1 = 75.0             (* set motion deceleration, units/sec^2
    MJK1 = 0                (* set motion jerk percentage,  % of accel & decel interval
    MVL1 = 2.0              (* set motion velocity, units/sec
    TLC1 = 10.0             (* set torque limit current, % of continuous current
    TLE1 = ON               (* enable torque limit
    RVF1                    (* run axis 1 forward
    WAIT TL1                (* wait for axis 1 to be at torque limit
    ST1                     (* stop all motion
    TLE1 = OFF              (* disable torque limit
```

Run Reverse at Torque Limit



(* Notes:
(* 1- Registers that have been previously loaded with appropriate values
(*   do not have to be reloaded for this motion.
(* 2- Loading the MAC register also loads the MDC register with the same value. To set
(*   MDC to a value different from MAC, load MDC after loading the MAC register.
(* 3- This example assumes the MFP (Motion Feedrate Percentage) is set to its default value
(*   of 100. See IMC & IMJ files I_MVABSF.TXT and I_MT1ABF.TXT or Target files
(*   T_MVABSF.TXT and T_MT1ABF.TXT for examples using the MFP register.
(* 4- If this template is executed within a motion block, the WAIT TL1 and TLE1 = OFF
(*   commands are executed only after the acceleration portions of the preceding motion
(*   commands are complete.

IMC & IMJ Template

(* Motion Template:	I_MTQREV.TXT
(* Revision Log:	REV 098OCT02
(* DspMotion Series:	IMC & IMJ
(* Move Type:	Run reverse at torque limit
(* Engineering Units:	Motor revolutions: i.e., URA = position feedback resolution

(* Motion:	Run forward until torque limit, then run reverse.
(* MT = VEL	This register cannot be loaded if motion is in progress.
(*	MT does not need to be set unless it is set to a MT setting other
(*	than VEL. The default value for MT is VEL.

```
MAC = 50.0          (* set motion acceleration, units/sec^2
MDC = 75.0          (* set motion deceleration, units/sec^2
MJK = 0             (* set motion jerk percentage,  % of accel & decel interval
MVL = 2.0           (* set motion velocity, units/sec
TLC = 10.0          (* set torque limit current, % of continuous current
TLE = ON            (* enable torque limit
RVF                 (* run forward
WAIT TL             (* wait for axis to be at torque limit
RVR                 (* run reverse
TLE = OFF           (* disable torque limit
```

Target Template

```
(* Motion Template:        T_MTQREV.TXT
(* Revision Log:           REV 098OCT02
(* DspMotion Series:       Target ARS
(* Move Type:              Run reverse at torque limit
(* Engineering Units:      Motor revolutions: i.e., URA1 = position feedback resolution

(* Motion:                 Run axis 1 forward until torque limit, then run reverse.
(* MT1 = VEL               This register cannot be loaded if motion is in progress.
(*                         MT does not need to be set unless it is set to a MT setting other
(*                         than VEL.  The default value for MT is VEL.

    MAC1 = 50.0            (* set motion acceleration, units/sec^2
    MDC1 = 75.0            (* set motion deceleration, units/sec^2
    MJK1 = 0               (* set motion jerk percentage, % of accel & decel interval
    MVL1 = 2.0             (* set motion velocity, units/sec
    TLC1 = 10.0            (* set torque limit current, % of continuous current
    TLE1 = ON              (* enable torque limit
    RVF1                   (* run axis 1 forward
    WAIT TL1               (* wait for axis 1 to be at torque limit
    RVR1                   (* run axis 1 reverse
    TLE1 = OFF             (* disable torque limit
```

Synchronized Moves

Single-Axis Electronic Gearing

```
(* Notes:
(* 1- Registers that have been previously loaded with appropriate values do not have to be
(*    reloaded for this motion.
```

IMC & IMJ Template

```
(* Motion Template:        I_MS1EG.TXT
(* Revision Log:           REV 098OCT05
(* DspMotion Series:       IMC & IMJ
(* Move Type:              Single-axis electronic gearing
(* Engineering Units:      Motor revolutions: i.e., URA = position feedback resolution
(*                         URX = auxiliary feedback resolution

(* Motion:                 Move axis in relation to the auxiliary input.  Axis will follow
(*                         the auxiliary input based on the values of GRN and GRD, i.e.,
(*
(*                                       GRN
(*                         axis pulses = ——— * auxiliary pulses
(*                                       GRD

   GRN = 1                 (* set gearing numerator
   GRD = 1                 (* set gearing denominator
   GRB = 0                 (* set gearing bound
   GRF = 0                 (* set gearing filter constant
   GRE = ON                (* enable electronic gearing
```

Target Template

```
(* Motion Template:        T_MS1EG.TXT
(* Revision Log:           REV 098OCT05
(* DspMotion Series:       Target ARS
(* Move Type:              Single-axis electronic gearing
(* Engineering Units:      Motor revolutions: i.e., URA1 = position feedback resolution
(*                         URX1 = auxiliary feedback resolution

(* Motion:                 Move axis 1 in relation to the auxiliary input.  Axis 1 will
(*                         follow the auxiliary input based on the values of GRN1 and
(*                         GRD1, i.e.,

(*
(*                                       GRN1
(*                         axis pulses = ———— * auxiliary pulses
(*                                       GRD1
GRI1 = PSX1          (* set gearing input to aux input of axis 1
GRN1 = 1             (* set gearing numerator
GRD1 = 1             (* set gearing denominator
GRB1 = 0             (* set gearing bound
GRF1 = 0             (* set gearing filter constant
GRE1 = ON            (* enable electronic gearing for axis 1
```

Multi-axis Electronic Gearing⊙

Target Template

```
(* Motion Template:          T_MS4EG.TXT
(* Revision Log:             REV 098OCT05
(* DspMotion Series:         Target ARS
(* Move Type:                Multi-axis electronic gearing
(* Engineering Units:        Motor revolutions: i.e., URA1 = axis 1 position feedback
(*                           resolution
(*                           URX1 = axis 1 auxiliary feedback resolution
(*                           URA2 = axis 2 position feedback resolution, etc.

(* Motion:                   Move each axis in relation to axis 1  auxiliary input.  Each axis
(*                           will follow the auxiliary input based on the values of GRNp1
(*                           and GRDp1, i.e.,

(*                                        GRNp1
(*                           axis pulses = ——— * auxiliary pulses
(*                                        GRDp1
(*
(*                           where p1 is an axis number 1 through 4.  In this template, we
(*                           have set the gear ratios to be 1/1, 3/4, 5/6 and 2/1.

(* Notes:
(* 1 -Registers that have been previously loaded with appropriate values do not have to be
(*    reloaded for this motion.

    GRI1 = PSX1              (* set gearing input to aux input of axis 1
    GRI2 = PSX1              (* set gearing input to aux input of axis 1
    GRI3 = PSX1              (* set gearing input to aux input of axis 1
    GRI4 = PSX1              (* set gearing input to aux input of axis 1
    GRN1 = 1                 (* set axis 1 gearing numerator
    GRN2 = 3                 (* set axis 2 gearing numerator
    GRN3 = 5                 (* set axis 3 gearing numerator
    GRN4 = 2                 (* set axis 4 gearing numerator
    GRD1 = 1                 (* set axis 1 gearing denominator
    GRD2 = 4                 (* set axis 2 gearing denominator
    GRD3 = 6                 (* set axis 3 gearing denominator
    GRD4 = 1                 (* set axis 4 gearing denominator
    GRB1 = 0                 (* set gearing bound
    GRB2 = 0                 (* set gearing bound
    GRB3 = 0                 (* set gearing bound
    GRB4 = 0                 (* set gearing bound
    GRF1 = 0                 (* set gearing filter constant
    GRF2 = 0                 (* set gearing filter constant
    GRF3 = 0                 (* set gearing filter constant
    GRF4 = 0                 (* set gearing filter constant
    GRE1234 = ON             (* enable electronic gearing for axes 1, 2, 3 and 4
```

Single-Axis, Phase-Locked Loop

```
(* Notes:
(* 1- Registers that have been previously loaded with appropriate values do not have to be
(*    reloaded for this motion.
(* 2- The phase-locked loop becomes active whenever a position is captured.  The output of
(*    the phase-locked loop is calculated based on the phase error, PHR, which is the
(*    difference between the desired reference position, PHP, and the captured position.  The
(*    output of the PLL replaces the gearing numerator each time the position is captured,
(*    thereby changing the value of PHM.
```

IMC & IMJ Template

```
(* Motion Template:        I_MS1PLL.TXT
(* Revision Log:           REV 098OCT16
(* DspMotion Series:       IMC & IMJ
(* Move Type:              Single-axis, phase-locked loop
(* Engineering Units:      Motor revolutions: i.e., URA = position feedback resolution
(*                         URX = auxiliary feedback resolution

(* Motion:                 Move axis in relation to the auxiliary input.  The axis will
(*                         follow the auxiliary input based on the output of the
(*                         phase-locked loop, i.e.,
(*
(*                                 axis pulses = PHM * auxiliary pulses
(*
(*                         where PHM is the phase multiplier, which is equal to the output
(*                         of the phase-locked loop divided by the gearing denominator,
(*                         GRD.

    PHP = 0                (* set phase position, pulses
    PHL = 4000             (* set phase length, pulses
    PHO = 0                (* set phase offset, pulses
    PHB = 2000             (* set phase error bound, pulses
    PHG = 10               (* set phase gain
    PHZ = 245              (* set phase zero
    PHT = 0.05             (* set phase lockout time, seconds

    GRN = 1000             (* set gearing numerator
    GRD = 1000             (* set gearing denominator
    GRB = 0                (* set gearing bound
    GRF = 0                (* set gearing filter constant

    GRE = ON               (* enable electronic gearing
    PHE = ON               (* enable phase-locked loop
```

Target Template

```
(* Motion Template:        T_MS1PLL.TXT
(* Revision Log:           REV 098OCT16
(* DspMotion Series:       IMC
(* Move Type:              Single-axis, phase-locked loop
(* Engineering Units:      Motor revolutions: i.e., URA1 = position feedback resolution
(*                         URX1 = auxiliary feedback resolution

(* Motion:                 Move axis 1 in relation to the auxiliary input.  Axis 1 will
(*                         follow the auxiliary input based on the output of the
(*                         phase-locked loop, i.e.,
(*
(*                                 axis pulses = PHM * auxiliary pulses
(*
(*                         where PHM is the phase multiplier, which is equal to the output
(*                         of the phase-locked loop divided by the gearing denominator,
(*                         GRD.

    PHP1 = 0               (* set phase position, pulses
    PHL1 = 4000            (* set phase length, pulses
    PHO1 = 0               (* set phase offset, pulses
    PHB1 = 2000            (* set phase error bound, pulses
    PHG1 = 10              (* set phase gain
    PHZ1 = 245             (* set phase zero
    PHT1 = 0.05            (* set phase lockout time, seconds

    GRI1 = PSX1            (* set gearing input to aux input of axis 1
    GRN1 = 1000            (* set gearing numerator
    GRD1 = 1000            (* set gearing denominator
    GRB1 = 0               (* set gearing bound
    GRF1 = 0               (* set gearing filter constant

    GRE1 = ON              (* enable electronic gearing for axis 1
    PHE1 = ON              (* enable phase-locked loop for axis 1
```

Multi-axis, Phase-Locked Loop ⊙

Target Template

```
(* Motion Template:          T_MS4PLL.TXT
(* Revision Log:             REV 098OCT16
(* DspMotion Series:         Target ARS
(* Move Type:                Multi-axis phase-locked loop
(* Engineering Units:        Motor revolutions: i.e., URA1 = axis 1 position feedback
(*                           resolution
(*                           URX1 = axis 1 auxiliary feedback resolution
(*                           URA2 = axis 2 position feedback resolution
(*                           URX2 = axis 2 auxiliary feedback resolution, etc.

(* Motion:                   Move each axis in relation to its corresponding auxiliary input.
(*                           Each axis will follow each auxiliary input based on the output
(*                           of the phase-locked loop, i.e.,
(*
(*                                   axis pulses = PHMp1 * auxiliary pulses
(*
(*                           where p1 is an axis number 1 through 4.  PHM is the phase
(*                           multiplier, which is equal to the output of the phase-locked
(*                           loop divided by the gearing denominator, GRD.  Also note that
(*                           in this template, we have set the gear ratios to be 1/1, 3/4, 5/6
(*                           and 2/1.

(* Notes:
(* 1- Registers that have been previously loaded with appropriate values
(*    do not have to be reloaded for this motion.
(* 2- The phase-locked loop becomes active whenever a position is captured.  The output of
(*    the phase-locked loop is calculated based on the phase error, PHR, which is the
(*    difference between the desired reference position, PHP, and the captured position.  The
(*    output of the PLL replaces the gearing numerator each time the position is captured,
(*    thereby changing the value of PHM.

     PHP1 = 0                (* set phase position, pulses
     PHL1 = 4000             (* set phase length, pulses
     PHO1 = 0                (* set phase offset, pulses
     PHB1 = 2000             (* set phase error bound, pulses
     PHG1 = 10               (* set phase gain
     PHZ1 = 245              (* set phase zero
     PHT1 = 0.05             (* set phase lockout time, seconds

     PHP2 = 0                (* set phase position, pulses
     PHL2 = 4000             (* set phase length, pulses
     PHO2 = 0                (* set phase offset, pulses
     PHB2 = 2000             (* set phase error bound, pulses
     PHG2 = 10          (* set phase gain
     PHZ2 = 245         (* set phase zero
     PHT2 = 0.05        (* set phase lockout time, seconds

     PHP3 = 0           (* set phase position, pulses
     PHL3 = 4000        (* set phase length, pulses
     PHO3 = 0           (* set phase offset, pulses
     PHB3 = 2000        (* set phase error bound, pulses
     PHG3 = 10          (* set phase gain
     PHZ3 = 245         (* set phase zero
     PHT3 = 0.05        (* set phase lockout time, seconds

     PHP4 = 0           (* set phase position, pulses
     PHL4 = 4000        (* set phase length, pulses
     PHO4 = 0           (* set phase offset, pulses
     PHB4 = 2000        (* set phase error bound, pulses
     PHG4 = 10          (* set phase gain
     PHZ4 = 245         (* set phase zero
     PHT4 = 0.05        (* set phase lockout time, seconds

     GRI1 = PSX1        (* set gearing input to aux input of axis 1
     GRI2 = PSX2        (* set gearing input to aux input of axis 2
     GRI3 = PSX3        (* set gearing input to aux input of axis 3
     GRI4 = PSX4        (* set gearing input to aux input of axis 4
     GRN1 = 1000        (* set axis 1 gearing numerator
     GRN2 = 750         (* set axis 2 gearing numerator
     GRN3 = 833         (* set axis 3 gearing numerator
     GRN4 = 2000        (* set axis 4 gearing numerator
     GRD1 = 1000        (* set axis 1 gearing denominator
     GRD2 = 1000        (* set axis 2 gearing denominator
     GRD3 = 1000        (* set axis 3 gearing denominator
     GRD4 = 1000        (* set axis 4 gearing denominator
     GRB1 = 0           (* set gearing bound
     GRB2 = 0           (* set gearing bound
     GRB3 = 0           (* set gearing bound
     GRB4 = 0           (* set gearing bound
     GRF1 = 0           (* set gearing filter constant
     GRF2 = 0           (* set gearing filter constant
     GRF3 = 0           (* set gearing filter constant
     GRF4 = 0           (* set gearing filter constant

     GRE1234 = ON       (* enable electronic gearing for all axes
     PHE1234 = ON       (* enable phase-locked loop for all axes
```

Single-Axis, Electronic Camming

```
(* Notes:
(* 1- Registers that have been previously loaded with appropriate values
(*     do not have to be reloaded for this motion.
(* 2- Loading the MAP register also loads the MDP register with the same value.  To set MDP
(*     to a value different from MAP, load MDP after loading the MAP register.
(* 3- Loading the MAC register also loads the MDC register with the same value.  To set
(*     MDC to a value different from MAC, load MDC after loading the MAC register.
(* 4- This example assumes the MFP (Motion Feedrate Percentage) is set to its default value of
(*     100.  See IMC & IMJ files I_MVABSF.TXT and I_MT1ABF.TXT or Target files
(*     T_MVABSF.TXT and T_MT1ABF.TXT for examples using the MFP register.
(* 5- Since this template incorporates labels and commands that are not allowed in motion
(*     blocks (GOSUB, RETURN) it can only be used in a program.
```

IMC & IMJ Template

```
(* Motion Template:          I_MS1EC.TXT
(* Revision Log:             REV 098OCT19
(* DspMotion Series:         IMC & IMJ
(* Move Type:                Single-axis electronic camming
(* Engineering Units:        Motor revolutions: i.e., URA = position feedback resolution
(*                           URX = auxiliary feedback resolution

(* Motion:                   Move axis in relation to the auxiliary input.  The axis will
(*                           follow the auxiliary input based on the cam table which
(*                           contains the points for the cam motion.

(* Variables used:           VI10     initial cam location
(*                           VF10     calculated cam shaft offset, degrees

(*
(*  CAM TABLE SETUP
(*
 100CAZ                      (* zero cam table
(* compile 285 - 75 degree motion segment
     CCP = -1.0              (* set cam compile start position, axis units
     MPA = 1.0               (* set absolute move position, axis units
     CCB = 285.0             (* set cam compile beginning point, degrees
     CCE = 75.0              (* set cam compile ending point, degrees
     MAP = 25                (* set motion accel percentage, % of motion
     MDP = 20                (* set motion decel percentage, % of motion
     MJK = 100               (* set motion jerk percentage, % of accel/decel interval
     CCM                     (* compile axis motion segment
(* compile 75 - 105 degree dwell
     CCP = MPA               (* set cam compile start position, axis units
     CCB = CCE               (* set cam compile beginning point, degrees
     CCE = 105.0             (* set cam compile ending point, degrees
     CCM                     (* compile axis motion segment
(* compile 105 - 255 degree motion segment
     CCP = MPA               (* set cam compile start position, axis units
     MPA = -1.0              (* set absolute move position, axis units
     CCB = CCE               (* set cam compile beginning point, degrees
     CCE = 255.0             (* set cam compile ending point, degrees
     CCM                     (* compile axis motion segment
(* compile 255 - 285 degree dwell
     CCP = MPA               (* set cam compile start position, axis units
     CCB = CCE               (* set cam compile beginning point, degrees
     CCE = 285.0             (* set cam compile ending point, degrees
     CCM                     (* compile axis motion segment
     RETURN                  (* return from subroutine

(*
(*  RUN CAM MOTION
(*

(* MT = VEL                   This register cannot be loaded if motion is in progress.
(*                            MT does not need to be set unless it is set to a MT setting other
(*                            than VEL.  The default value for MT is VEL.

     CAT = PSX               (* set cam type to auxiliary input
     CAS = 2.5               (* set cam scale factor
     VI10 = 2700             (* define initial cam location
     VF10 = -CAP + ITF(VI10) / 10.     (* calculate the cam shaft offset
     IF VF10 > 180. THEN     (* bound offset to +/- 180 degrees
     VF10 = VF10 - 360.
     IF VF10 <= -180. THEN VF10 = VF10 + 360.
     CAO = VF10              (* set cam offset, degrees
     GOSUB 100               (* generate cam table
     MVL = 1.0               (* set motion velocity, units/sec
     MAC = 50.0              (* set motion acceleration, units/sec^2
     MDC = 75.0              (* set motion deceleration, units/sec^2
     MPA = CAS * CAMVI10 (* set absolute move position, units
     RPA                     (* run to initial cam follower position
     WAIT IP                 (* wait for axis to be in position
     CAF = 2                 (* set cam filter constant
     STM1 = 0.1              (* set start time of timer 1, seconds
     WAIT TM1                (* wait for filter to settle
     CAE = ON                (* enable electronic camming
```

Target Template

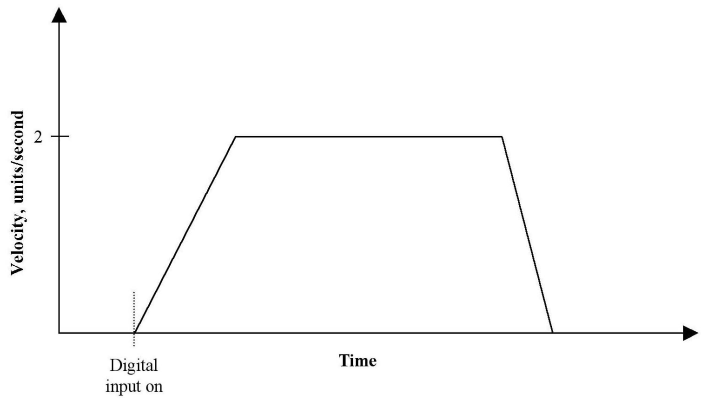
```
(* Motion Template:         T_MS1EC.TXT
(* Revision Log:            REV 098OCT19
(* DspMotion Series:        Target ARS
(* Move Type:               Single-axis electronic camming
(* Engineering Units:       Motor revolutions: i.e., URA1 = axis 1 position feedback
(*                          resolution
(*                          URA2 = axis 2 position feedback resolution

(* Motion:                  Move axis 2 in relation to axis 1.  Axis 2 will follow the axis 1
(*                          based on the cam table which contains the points for the cam
(*                          motion.

(* Variables used:          VI10    initial cam location
(*                          VF10    calculated cam shaft offset, degrees

(*
(*  CAM TABLE SETUP
(*
 100CAZ2                    (* zero cam table
(* compile 285 - 75 degree motion segment
    CCP2 = -1.0             (* set cam compile start position, axis units
    MPA2 = 1.0              (* set absolute move position, axis units
    CCB = 285.0             (* set cam compile beginning point, degrees
    CCE = 75.0              (* set cam compile ending point, degrees
    MAP2 = 25               (* set motion accel percentage, % of motion
    MDP2 = 20               (* set motion decel percentage, % of motion
    MJK2 = 100              (* set motion jerk percentage, % of accel/decel interval
    CCM2                    (* compile axis motion segment
(* compile 75 - 105 degree dwell
    CCP2 = MPA2             (* set cam compile start position, axis units
    CCB = CCE               (* set cam compile beginning point, degrees
    CCE = 105.0             (* set cam compile ending point, degrees
    CCM2                    (* compile axis motion segment
(* compile 105 - 255 degree motion segment
    CCP2 = MPA2             (* set cam compile start position, axis units
    MPA2 = -1.0             (* set absolute move position, axis units
    CCB = CCE               (* set cam compile beginning point, degrees
    CCE = 255.0             (* set cam compile ending point, degrees
    CCM2                    (* compile axis motion segment
(* compile 255 - 285 degree dwell
    CCP2 = MPA2             (* set cam compile start position, axis units
    CCB = CCE               (* set cam compile beginning point, degrees
    CCE = 285.0             (* set cam compile ending point, degrees
    CCM2                    (* compile axis motion segment
    RETURN                  (* return from subroutine
(*
(*  RUN CAM MOTION
(*
    CAT = PSR1                (* set cam type to axis 1 resolver input
    CAS2 = 2.5                (* set cam scale factor
    VI10 = 2700               (* define initial cam location
    VF10 = -CAP + ITF(VI10) / 10.
                              (* initialize the cam shaft offset
    IF VF10 > 180. THEN       (* bound offset to +/- 180 degrees
    VF10 = VF10 - 360. IF VF10 <= -180. THEN VF10 = VF10 + 360.
    CAO2 = VF10               (* set cam offset, degrees
    GOSUB 100                 (* generate cam table
    MVL2 = 1.0                (* set motion velocity, units/sec
    MAC2 = 50.0               (* set motion acceleration, units/sec^2
    MDC2 = 75.0               (* set motion deceleration, units/sec^2
    MPA2 = CAS2 * CAM2.VI10       (* set absolute move position, units
    RPA2                      (* run to initial cam follower position
    WAIT IP2                  (* wait for axis to be in position
    CAF = 2                   (* set cam filter constant
    STM1 = 0.1                (* set start time of timer 1, seconds
    WAIT TM1                  (* wait for filter to settle
    CAE2 = ON                 (* enable electronic camming
```

Multi-axis, Synchronized Electronic Camming ⊙

Target Template

```
(* Motion Template:          T_MS2EC.TXT
(* Revision Log:             REV 098OCT19
(* DspMotion Series:         Target ARS
(* Move Type:                Multi-axis synchronized electronic camming
(* Engineering Units:        Motor revolutions: i.e., URA1 = axis 1 position feedback
(*                           resolution URA2 = axis 2 position feedback resolution;
(*                           URA3 = axis 3 position feedback resolution

(* Motion:                   Move axes 2 and 3 in relation to axis 1.  Axes 2 and 3 will
(*                           follow the axis 1 based on the cam table that contains the points
(*                           for the cam motion.

(* Variables used:           VI10    initial cam location
(*                           VF10    calculated cam shaft offset, degrees

(* Notes:
(* 1- Registers that have been previously loaded with appropriate values do not have to be
reloaded for this motion.
(* 2- Loading the MAP register also loads the MDP register with the same value.  To set MDP
(*    to a value different from MAP, load MDP after loading the MAP register.
(* 3- Loading the MAC register also loads the MDC register with the same value.  To set
(*    MDC to a value different from MAC, load MDC after loading the MAC register.
(* 4- This example assumes the MFP (Motion Feedrate Percentage) is set to its default value
(*    of 100.  See files T_MVABSF.TXT and T_MT1ABF.TXT for examples using the MFP
(*    register.
(* 5- Since this template incorporates labels and commands that are not allowed in motion
(*    blocks (GOSUB, RETURN), it can only be used in a program.

(*  CAM TABLE SETUP
(*
 100CAZ23                    (* zero cam tables
(* compile axis 2 cam motion
(* compile 285 - 75 degree motion segment
     CCP2 = -1.0             (* set cam compile start position, axis units
     MPA2 = 1.0              (* set absolute move position, axis units
     CCB = 285.0             (* set cam compile beginning point, degrees
     CCE = 75.0              (* set cam compile ending point, degrees
     MAP2 = 25               (* set motion accel percentage, % of motion
     MDP2 = 20               (* set motion decel percentage, % of motion
     MJK2 = 100              (* set motion jerk percentage, % of accel/decel interval
     CCM2                    (* compile axis motion segment
(* compile 75 - 105 degree dwell
     CCP2 = MPA2             (* set cam compile start position, axis units
     CCB = CCE               (* set cam compile beginning point, degrees
     CCE = 105.0             (* set cam compile ending point, degrees
     CCM2                  (* compile axis motion segment


(* compile 105 - 255 degree motion segment
     CCP2 = MPA2           (* set cam compile start position, axis units
     MPA2 = -1.0           (* set absolute move position, axis units
     CCB = CCE             (* set cam compile beginning point, degrees
     CCE = 255.0           (* set cam compile ending point, degrees
     CCM2                  (* compile axis motion segment
(* compile 255 - 285 degree dwell
     CCP2 = MPA2           (* set cam compile start position, axis units
     CCB = CCE             (* set cam compile beginning point, degrees
     CCE = 285.0           (* set cam compile ending point, degrees
     CCM2                  (* compile axis motion segment
(* compile axis 3 cam motion
(* compile 310 - 50 degree motion segment
     CCP3 = 0.0            (* set cam compile start position, axis units
     MPA3 = 5.0            (* set absolute move position, axis units
     CCB = 310.0           (* set cam compile beginning point, degrees
     CCE = 50.0            (* set cam compile ending point, degrees
     CCM3                  (* compile axis motion segment
(* compile 50 - 130 degree dwell
     CCP3 = MPA3           (* set cam compile start position, axis units
     CCB = CCE             (* set cam compile beginning point, degrees
     CCE = 130.0           (* set cam compile ending point, degrees
     CCM3                  (* compile axis motion segment
(*  compile 130 - 230 degree motion segment
     CCP3 = MPA3           (* set cam compile start position, axis units
     MPA3 = 0.0            (* set absolute move position, axis units
     CCB = CCE             (* set cam compile beginning point, degrees
     CCE = 230.0           (* set cam compile ending point, degrees
     CCM3                  (* compile axis motion segment
(*  compile 230 - 310 degree dwell
     CCP3 = MPA3           (* set cam compile start position, axis units
     CCB = CCE             (* set cam compile beginning point, degrees
     CCE = 310.0           (* set cam compile ending point, degrees
     CCM3                  (* compile axis motion segment
     RETURN                (* return from subroutine

(* RUN CAM MOTION
(*
     CAT = PSR1            (* set cam type to axis 1 resolver input
     CAS2 = 2.5            (* set axis 2 cam scale factor
     CAS3 = 1              (* set axis 3 cam scale factor
     VI10 = 2700           (* define initial cam location
     VF10 = -CAP + ITF(VI10) / 10.
                           (* initialize the cam shaft offset
     IF VF10 > 180. THEN   (* bound offset to +/- 180 degrees
     VF10 = VF10 - 360.
     IF VF10 <= -180. THEN VF10 = VF10 + 360.
     CAO2 = VF10           (* set axis 2 cam offset, degrees
     CAO3 = VF10                 (* set axis 3 cam offset, degrees
     GOSUB 100                   (* generate cam table
     MVL2 = 1.0                  (* set motion velocity, units/sec
     MVL3 = 1.0                  (* set motion velocity, units/sec
     MAC2 = 50.0                 (* set motion acceleration, units/sec^2
     MAC3 = 50.0                 (* set motion acceleration, units/sec^2
     MDC2 = 75.0                 (* set motion deceleration, units/sec^2
     MDC3 = 75.0                 (* set motion deceleration, units/sec^2
     MPA2 = CAS2 * CAM2.VI10         (* set absolute move position, units
     MPA3 = CAS3 * CAM3.VI10         (* set absolute move position, units
     RPA23                       (* run axes to initial cam follower position
     WAIT (IP2 AND IP3)          (* wait for axes to be in position
     CAF = 2                     (* set cam filter constant
     STM1 = 0.1                  (* set start time of timer 1, seconds
     WAIT TM1                    (* wait for filter to settle
     CAE23 = ON                  (* enable electronic camming
```

Single-Axis, Index Move after Input



```
(* Notes:
(* 1- Registers that have been previously loaded with appropriate values
(*     do not have to be reloaded for this motion.
(* 2- Loading the MAC register also loads the MDC register with the same value.  To set
(*     MDC to a value different from MAC, load MDC after loading the MAC register.
(* 3- This example assumes the MFP (Motion Feedrate Percentage) is set to its default value of
(*     100.  See IMC & IMJ files I_MVABSF.TXT and I_MT1ABF.TXT or Target files
(*     T_MVABSF.TXT and T_MT1ABF.TXT for examples using the MFP register.
```

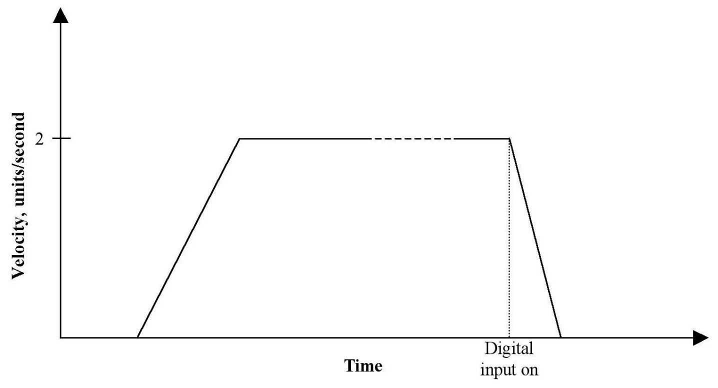
IMC & IMJ Template

```
(* Motion Template:        I_MS1AIN.TXT
(* Revision Log:           REV 098OCT08
(* DspMotion Series:       IMC & IMJ
(* Move Type:              Single-axis index move after input
(* Engineering Units:      Motor revolutions: i.e., URA = position feedback resolution

(* Motion:                 Run forward by 3 units after digital input 1 turns on.
(* MT = VEL                This register cannot be loaded if motion is in progress.
(*                         MT does not need to be set unless it is set to a MT setting other
(*                         than VEL.  The default value for MT is VEL.

    MAC = 50.0             (* set motion acceleration, units/sec^2
    MDC = 75.0             (* set motion deceleration, units/sec^2
    MJK = 0                (* set motion jerk percentage, % of accel & decel interval
    MVL = 2.0              (* set motion velocity, units/sec
    MPI = 3.0              (* set incremental move position, units
    WAIT DI1               (* wait for digital input 1 to be turned on
    RPI                    (* run to incremental move position
```

Target Template

```
(* Motion Template:        T_MS1AIN.TXT
(* Revision Log:           REV 098OCT08
(* DspMotion Series:       Target ARS
(* Move Type:              Single-axis index move after input
(* Engineering Units:      Motor revolutions: i.e., URA1 = position feedback resolution

(* Motion:                 Run axis 1 forward by 3 units after digital input 1 turns on.
(* MT1 = VEL               This register cannot be loaded if motion is in progress.
(*                         MT does not need to be set unless it is set to a MT setting other
(*                         than VEL.  The default value for MT is VEL.

    MAC1 = 50.0            (* set motion acceleration, units/sec^2
    MDC1 = 75.0            (* set motion deceleration, units/sec^2
    MJK1 = 0               (* set motion jerk percentage, % of accel & decel interval
    MVL1 = 2.0             (* set motion velocity, units/sec
    MPI1 = 3.0             (* set axis 1 incremental move position, units
    WAIT DI1.1             (* wait for digital input 1 to be turned on
    RPI1                   (* run axis 1 to incremental move position
```

Single-Axis, Run Forward until Input



```
(* Notes:
(* 1- Registers that have been previously loaded with appropriate values
(*     do not have to be reloaded for this motion.
(* 2- Loading the MAC register also loads the MDC register with the same value.  To set
(*     MDC to a value different from MAC, load MDC after loading the MAC register.
(* 3- This example assumes the MFP (Motion Feedrate Percentage) is set to its default value of
(*     100.  See IMC & IMJ files I_MVABSF.TXT and I_MT1ABF.TXT or Target files
(*     T_MVABSF.TXT and T_MT1ABF.TXT for examples using the MFP register.
```

IMC & IMJ Template

```
(* Motion Template:        I_MS1FIN.TXT
(* Revision Log:           REV 098OCT08
(* DspMotion Series:       IMC & IMJ
(* Move Type:              Run forward until input
(* Engineering Units:      Motor revolutions: i.e., URA = position feedback resolution

(* Motion:                 Run forward until input turned on.
(* MT = VEL                This register cannot be loaded if motion is in progress.
(*                         MT does not need to be set unless it is set to a MT setting other
(*                         than VEL.  The default value for MT is VEL.

    MAC = 50.0             (* set motion acceleration, units/sec^2
    MDC = 75.0             (* set motion deceleration, units/sec^2
    MJK = 0                (* set motion jerk percentage, % of accel & decel interval
    MVL = 2.0              (* set motion velocity, units/sec
    RVF                    (* run forward
    WAIT DI1               (* wait for digital input 1 to be turned on
    ST                     (* stop all motion
```

Target Template

```
(* Motion Template:        T_MS1FIN.TXT
(* Revision Log:           REV 098OCT08
(* DspMotion Series:       Target ARS
(* Move Type:              Run forward until input
(* Engineering Units:      Motor revolutions: i.e., URA1 = position feedback resolution

(* Motion:                 Run axis 1 forward until input turned on.
(* MT1 = VEL               This register cannot be loaded if motion is in progress.
(*                         MT does not need to be set unless it is set to a MT setting other
(*                         than VEL.  The default value for MT is VEL.

    MAC1 = 50.0            (* set motion acceleration, units/sec^2
    MDC1 = 75.0            (* set motion deceleration, units/sec^2
    MJK1 = 0               (* set motion jerk percentage,  % of accel & decel interval
    MVL1 = 2.0             (* set motion velocity, units/sec
    RVF1                   (* run axis 1 forward
    WAIT DI1.1             (* wait for digital input 1 to be turned on
    ST1                    (* stop all motion
```

Single-Axis Index Move at Predefined Auxiliary Position Reference

```
(* Notes:
(* 1- Registers that have been previously loaded with appropriate values
(*    do not have to be reloaded for this motion.
(* 2- Loading the MAP register also loads the MDP register with the same value.  To set MDP
(*    to a value different from MAP, load MDP after loading the MAP register.
(* 3- This example assumes the MFP (Motion Feedrate Percentage) is set to its default value
(*    of 100.  See IMC & IMJ files I_MVABSF.TXT and I_MT1ABF.TXT or Target files
(*    T_MVABSF.TXT and T_MT1ABF.TXT for examples using the MFP register.
(* 4- In order for this example to work properly, the position register wrap must be enabled,
(*    i.e., PWE must be set to ON.
(* 5- Since this template incorporates labels and commands that are not allowed in motion
(*    blocks (GOTO), it can be used only in a program.
```

IMC & IMJ Template

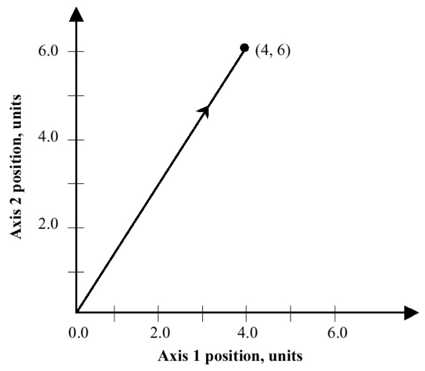
```
(* Motion Template:        I_MS1POS.TXT
(* Revision Log:           REV 098OCT12
(* DspMotion Series:       IMC & IMJ
(* Move Type:              Single-axis index move at predefined auxiliary position
(*                         reference
(* Engineering Units:      Motor revolutions: i.e., URA = position feedback resolution
(*                         URX = auxiliary feedback resolution

(* Motion:                 The axis will remain in position until the position capture input
(*                         edge is detected.  Then, as the aux position increases by 3 aux
(*                         units, the axis will run forward 6 axis units.
(* MT = PULSE              This register cannot be loaded if motion is in progress.
(*                         MT does not need to be set unless it is set to a MT setting other
(*                         than PULSE.  The default value for MT is VEL.

    MAP = 20               (* set motion acceleration percentage, % of move pulses
    MDP = 15               (* set motion deceleration percentage, % of move pulses
    MPL = 3.0              (* set move pulses, aux units
    MPI = 6.0              (* set incremental move position, units
    VF10 = 0.2             (* load distance between sensor and motion start
001 VF20 = PCA             (* reset position capture
    VF21 = PCX             (* reset aux position capture
    WAIT IO13              (* wait for position capture input edge
    VF11 = PCX + VF10      (* calculate motion start position
    IF VF11 < (PLX - (MPL + VF10)) - (1.0 / ITF(URX)) GOTO 10
                           (* if start position < max positive goto 10
    OFX = -(MPL + VF10)    (* offset aux position by move pulses  + distance between
                           (*sensor and motion start

    MPS = VF11 - (MPL + VF10)
                           (* set motion start position, aux units
    RPI                    (* run to incremental move position
     WAIT IP                (* wait until motion ends
     OFX = MPL + VF10       (* offset aux position back to original
     GOTO 1                 (* go back and wait for position capture
010MPS = VF11               (* set motion start position, aux units
     RPI                    (* run to incremental move position
     WAIT IP                (* wait until motion ends
     GOTO 1                 (* go back and wait for position capture
```

Target Template

```
(* Motion Template:         T_MS1POS.TXT
(* Revision Log:            REV 098OCT12
(* DspMotion Series:        Target ARS
(* Move Type:               Single-axis index move at predefined auxiliary position
(*                          reference
(* Engineering Units:       Motor revolutions: i.e., URA1 = position feedback resolution
(*                          URX1 = auxiliary feedback resolution

(* Motion:                  Axis 1 will remain in position until the position capture input
(*                          edge is detected.  Then, as the aux position increases by 3 aux
(*                          units, the axis will run forward 6 axis units.
(* MT1 = PULSE              This register cannot be loaded if motion is in progress.
(*                          MT does not need to be set unless it is set to a MT setting other
(*                          than PULSE.  The default value for MT is VEL.

     MI1 = PSX1             (* set motion pulse input to axis 1 aux input
     MAP1 = 20              (* set motion acceleration percentage, % of move pulses
     MDP1 = 15              (* set motion deceleration percentage, % of move pulses
     MPL1 = 3.0             (* set move pulses, aux units
     MPI1 = 6.0             (* set incremental move position, units
     VF10 = 0.2             (* load distance between sensor and motion start
001VF20 = PCA1              (* reset position capture
     VF21 = PCX1            (* reset aux position capture
     WAIT IOA1.13           (* wait for position capture input edge
     VF11 = PCX1 + VF10     (* calculate motion start position
     IF VF11 < (PLX1 - (MPL1 + VF10)) - (1.0 / ITF(URX1)) GOTO 10
                            (* if start position < max positive goto 10
     OFX1 = -(MPL1 + VF10)  (* offset aux position by move pulses  + distance between
                            (* sensor and motion start
     MPS1 = VF11 - (MPL1 + VF10)
                            (* set motion start position, aux units
     RPI1                   (* run axis 1 to incremental move position
     WAIT IP1               (* wait until motion ends
     OFX1 = MPL1 + VF10     (* offset aux position back to original
     GOTO 1                 (* go back and wait for position capture
010MPS1 = VF11          (* set motion start position, aux units
     RPI1               (* run axis 1 to incremental move position
     WAIT IP1           (* wait until motion ends
     GOTO 1             (* go back and wait for position capture
```

Trajectory Moves

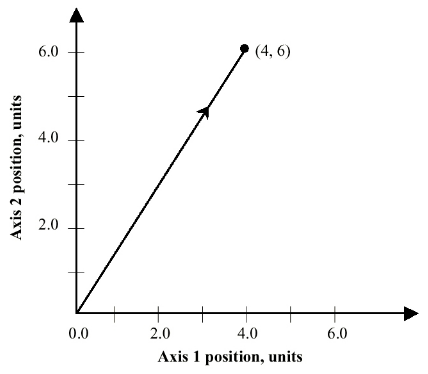
2-D Line Segment: Incremental Move ⊙



Target Template:

```
(* Motion Template:          T_MI2IL.TXT
(* Revision Log:             REV 098OCT05
(* DspMotion Series:         Target ARS
(* Move Type:                2-D line segment: incremental move
(* Engineering Units:        Motor revolutions: i.e., URA1 = axis 1 position feedback
(*                           resolution
(*                           URA2 = axis 2 position feedback resolution

(* Motion:                   Move axes 1 and 2 by 4 and 6 units in a line segment at a
(*                           trajectory velocity of 3 units/sec.

(* Notes:
(* 1- Registers that have been previously loaded with appropriate values
(*     do not have to be reloaded for this motion.
(* 2- Loading the TFA register also loads the TFD register with the same value.  To set TFD to
(*     a value different from TFA, load it after the TFA register.

    TVL = 3.0                (* set trajectory velocity, units/sec
    TFA = 500                (* set trajectory feedrate acceleration, feedrate % / sec
    TFD = 650                (* set trajectory feedrate deceleration, feedrate % / sec
    TFP = 100.0              (* set trajectory feedrate percentage, % of trajectory velocity
    MPI1 = 4.0               (* set axis 1 incremental move position, units
    MPI2 = 6.0               (* set axis 2 incremental move position, units
    RLI12                    (* run axes 1 and 2 to incremental move positions
```

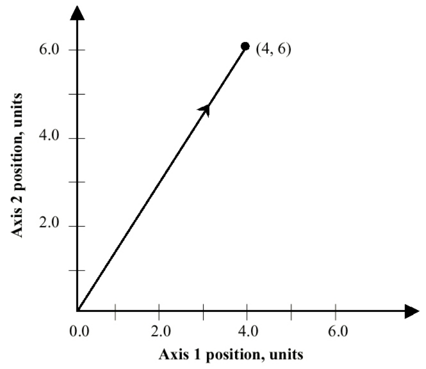
2-D Line Segment: Absolute Move ⊙



Target Template:

```
(* Motion Template:          T_MI2AL.TXT
(* Revision Log:             REV 098OCT05
(* DspMotion Series:         Target ARS
(* Move Type:                2-D line segment: absolute move
(* Engineering Units:        Motor revolutions: i.e., URA1 = axis 1 position feedback
(*                           resolution
(*                           URA2 = axis 2 position feedback resolution

(* Motion:                   Move axes 1 and 2 to absolute positions of 4 and 6 units in a
(*                           line segment at a trajectory velocity of 3 units/sec.

(* Notes:
(* 1- Registers that have been previously loaded with appropriate values
(*    do not have to be reloaded for this motion.
(* 2- Loading the TFA register also loads the TFD register with the same value.  To set TFD to
(*    a value different from TFA, load it after the TFA register.
(* 3- This example begins by loading the absolute position registers, PSA1 and PSA2, with 0
(*    for the purpose of accurately graphing the subsequent motion.  In general, applications
(*    will only load PSA1 and PSA2 at the end of a homing motion.

(* Initialize absolute position registers to (0,0)
    PSA1 = 0.0               (* set axis 1 absolute position register, units
    PSA2 = 0.0               (* set axis 2 absolute position register, units

(* Move axes 1 and 2 to absolute position (4,6) with the accelerations and velocities shown.
    TVL = 3.0                (* set trajectory velocity, units/sec
    TFA = 500                (* set trajectory feedrate acceleration, feedrate % / sec
    TFD = 650                (* set trajectory feedrate deceleration, feedrate % / sec
    TFP = 100.0              (* set trajectory feedrate percentage,  % of trajectory velocity
    MPA1 = 4.0               (* set axis 1 absolute move position, units
    MPA2 = 6.0               (* set axis 2 absolute move position, units
    RLA12                    (* run axes 1 and 2 to absolute move positions
```

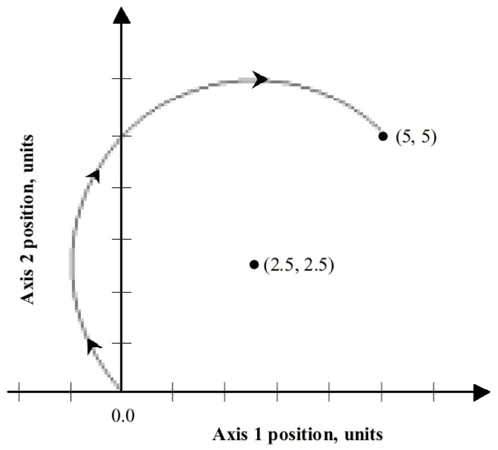
2-D Line Segment: Offset Move ⊙



Target Template:

```
(* Motion Template:          T_MI2OL.TXT
(* Revision Log:             REV 098OCT05
(* DspMotion Series:         Target ARS
(* Move Type:                2-D line segment: offset move
(* Engineering Units:        Motor revolutions: i.e., URA1 = axis 1 position feedback
(*                           resolution
(*                           URA2 = axis 2 position feedback resolution

(* Motion:                   Move axes 1 and 2 to offset positions of 4 and 6 units in a line
(*                           segment at a trajectory velocity of 3 units/sec.

(* Notes:
(* 1- Registers that have been previously loaded with appropriate values
(*    do not have to be reloaded for this motion.
(* 2- Loading the TFA register also loads the TFD register with the same value.  To set TFD to
(*    a value different from TFA, load it after the TFA register.
(* 3- This example begins by loading the offset position registers, PSO1 and PSO2, with 0 for
(*    the purpose of accurately graphing the subsequent motion.  Applications may require
(*    other offset position register values.

(* Initialize offset position registers to (0,0)
    PSO1 = 0.0               (* set axis 1 offset position register, units
    PSO2 = 0.0               (* set axis 2 offset position register, units

(* Move axes 1 and 2 to offset position (4,6) with the accelerations and velocities shown.
    TVL = 3.0                (* set trajectory velocity, units/sec
    TFA = 500                (* set trajectory feedrate acceleration, feedrate % / sec
    TFD = 650                (* set trajectory feedrate deceleration, feedrate % / sec
    TFP = 100.0              (* set trajectory feedrate percentage, % of trajectory velocity
    MPO1 = 4.0               (* set axis 1 offset move position, units
    MPO2 = 6.0               (* set axis 2 offset move position, units
    RLO12                    (* run axes 1 and 2 to offset move positions
```

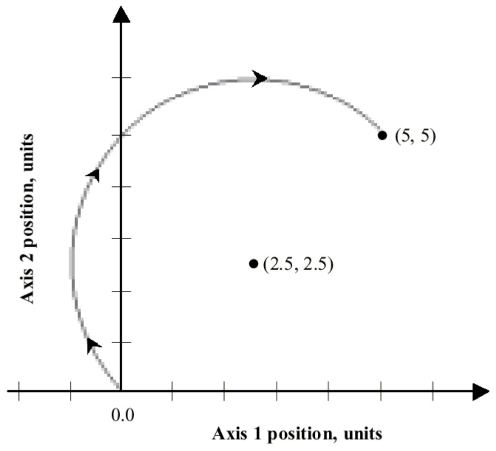
2-D Arc Segment Using Start, End, and Center Point: Incremental Move ⊙



Target Template:

```
(* Motion Template:          T_MI2ICC.TXT
(* Revision Log:             REV 098OCT05
(* DspMotion Series:         Target ARS
(* Move Type:                2-D arc segment using start, end, and center point:  incremental
(*                           move
(* Engineering Units:        Motor revolutions: i.e., URA1 = axis 1 position feedback
(*                           resolution; URA2 = axis 2 position feedback resolution

(* Motion:                   Move axes 1 and 2 in an arc with incremental center point at
(*                           (2.5, 2.5).  Move clockwise from starting position to
(*                           incremental position (5,5) at a trajectory velocity of 3 units/sec.

(* Notes:
(* 1- Registers that have been previously loaded with appropriate values
(*    do not have to be reloaded for this motion.
(* 2- Loading the TFA register also loads the TFD register with the same value.  To set TFD to
(*    a value different from TFA, load it after the TFA register.
(* 3- When using RCI to move the axes in arc, the incremental move distance registers, MDI1
(*    and MDI2, are used to define the center point of the arc.

    TAD = CW                 (* set arc direction
    TVL = 3.0                (* set trajectory velocity, units/sec
    TFA = 500                (* set trajectory feedrate acceleration, feedrate % / sec
    TFD = 650                (* set trajectory feedrate deceleration, feedrate % / sec
    TFP = 100.               (* set trajectory feedrate percentage, % of trajectory velocity
    MDI1 = 2.5               (* set axis 1 incremental move distance, units
    MDI2 = 2.5               (* set axis 2 incremental move distance, units
    MPI1 = 5.0               (* set axis 1 incremental move position, units
    MPI2 = 5.0               (* set axis 2 incremental move position, units
    RCI12                    (* run axes 1 and 2 to incremental move positions
```

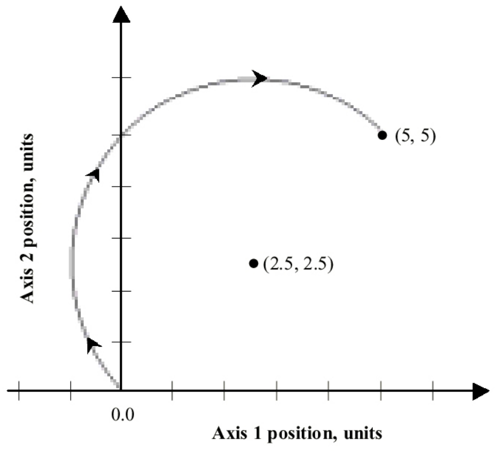
2-D Arc Segment Using Start, End, and ⊙ Center Point: Absolute Move



Target Template:

```
(* Motion Template:         T_MI2ACC.TXT
(* Revision Log:            REV 098OCT05
(* DspMotion Series:        Target ARS
(* Move Type:               2-D arc segment using start, end, and center point:  absolute
(*                          move
(* Engineering Units:       Motor revolutions: i.e., URA1 = axis 1 position feedback
(*                          resolution; URA2 = axis 2 position feedback resolution

(* Motion:                  Move axes 1 and 2 in an arc with center point at (2.5, 2.5)
(*                          clockwise from position (0,0) to position (5,5) at a trajectory
(*                          velocity of 3 units/sec.

(* Notes:
(* 1- Registers that have been previously loaded with appropriate values
(*    do not have to be reloaded for this motion.
(* 2- Loading the TFA register also loads the TFD register with the same value.  To set TFD to
(*    a value different from TFA, load it after the TFA register.
(* 3- The circularly interpolated move command, RCA, requires that both the starting and
(*    ending points of the motion lie on a circle with the specified center.  This example begins
(*    by moving the axes to absolute position (0,0), the starting point for the circle.
(* 4- When using RCA to move the axes in arc, the absolute move distance registers, MDA1
(*    and MDA2, are used to define the center point of the arc.

(* Initialize absolute position registers to (0,0)
    PSA1 = 0.0              (* set axis 1 absolute position register, units
    PSA2 = 0.0              (* set axis 2 absolute position register, units
(* Move axes 1 and 2 along circular path to absolute position (5,5) with the accelerations and
(* velocities shown.
    TAD = CW            (* set arc direction
    TVL = 3.0           (* set trajectory velocity, units/sec
    TFA = 500           (* set trajectory feedrate acceleration, feedrate % / sec
    TFD = 650           (* set trajectory feedrate deceleration, feedrate % / sec
    TFP = 100.          (* set trajectory feedrate percentage, % of trajectory velocity
    MDA1 = 2.5          (* set axis 1 absolute move distance, units
    MDA2 = 2.5          (* set axis 2 absolute move distance, units
    MPA1 = 5.0          (* set axis 1 absolute move position, units
    MPA2 = 5.0          (* set axis 2 absolute move position, units
    RCA12               (* run axes 1 and 2 to absolute move positions
```

2-D Arc Segment Using Start, End, and ⊙ Center Point: Offset Move



Target Template:

```
(* Motion Template:          T_MI2OCC.TXT
(* Revision Log:             REV 098OCT05
(* DspMotion Series:         Target ARS
(* Move Type:                2-D arc segment using start, end, and center point:  offset move
(* Engineering Units:        Motor revolutions: i.e., URA1 = axis 1 position feedback
(*                           resolution
(*                           URA2 = axis 2 position feedback resolution

(* Motion:                   Move axes 1 and 2 in an arc with center point at offset position
(*                           (2.5, 2.5).  Move clockwise from offset position (0,0) to offset
(*                           position (5,5) at a trajectory velocity of 3 units/sec.

(* Notes:
(* 1- Registers that have been previously loaded with appropriate values
(*    do not have to be reloaded for this motion.
(* 2- Loading the TFA register also loads the TFD register with the same value.  To set TFD to
(*    a value different from TFA, load it after the TFA register.
(* 3- The circularly interpolated move command, RCO, requires that both the starting and
(*    ending points of the motion lie on a circle with the specified center.  This example begins
(*    by setting the offset position to (0,0), the starting point for the circle.
(* 4- When using RCO to move the axes in arc, the offset move distance registers, MDO1 and
(*    MDO2, are used to define the center point of the arc.

(* Initialize offset position registers to (0,0)
    PSO1 = 0.0               (* set axis 1 offset position register, units
    PSO2 = 0.0               (* set axis 2 offset position register, units
(* Move axes 1 and 2 along circular path to offset position (5,5) with the accelerations and
(* velocities shown.
    TAD = CW              (* set arc direction
    TVL = 3.0             (* set trajectory velocity, units/sec
    TFA = 500             (* set trajectory feedrate acceleration, feedrate % / sec
    TFD = 650             (* set trajectory feedrate deceleration, feedrate % / sec
    TFP = 100.            (* set trajectory feedrate percentage, % of trajectory velocity
    MDO1 = 2.5            (* set axis 1 offset move distance, units
    MDO2 = 2.5            (* set axis 2 offset move distance, units
    MPO1 = 5.0            (* set axis 1 offset move position, units
    MPO2 = 5.0            (* set axis 2 offset move position, units
    RCO12                 (* run axes 1 and 2 to offset move positions
```

Appendix H Utility Templates

This appendix provides details on the following templates:

- First-In First-Out (FIFO) Buffer
- Display and Edit Time and Date on Operator Interface (OIP)
- Jogging Routines
  - Jog Using Analog Input
  - Jog Using Electronic Handwheel
  - Jog Using Single-Pole, Double-Throw Switch
  - Jog Using Operator Interface (OIP)
- Multi-axis Path Recording
- Report Controller Faults to OIP
- Retriggerable One-Shot
- Solve PID Algorithm
- Torque-Limited Pressing
- Two-Hand Anti-Tiedown

First-In First-Out Buffer

```
(* Utility Template:         A_FIFO.TXT
(* Revision Log:             REV 098APR29  Original release
(* DspMotion Series:         ALL
(* Function:                 First in first out buffer

(* Operation:
(*   Subroutine 300:         Increment stack input pointer and depth.
(*                                    Depth limited to value in VI21.
(*                                    Set VB20 if depth = VI21.
(*
(*   Subroutine 310:         If depth > 0, increment stack output pointer and
(*                           decrement stack depth.  VB21 set if depth = 0.
(*
(*  Subroutine 320:          Initialize FIFO stack input and output pointers to
(*                           value in VI20.  Initialize depth to 0.

(*  Global resources:
(*   VB20                    FIFO full flag
(*   VB21                    FIFO empty flag

(*  Module specific resources:
(*    Labels 300 through 320
(*    VFVI20 - VF(VI20+VI21)FIFO stack variables
(*    VI20                   FIFO start
(*    VI21                   maximum FIFO length
(*    VI22                   FIFO input pointer
(*    VI23                   FIFO output pointer
(*    VI24                   FIFO depth

(* Example of FIFO use:
(*
(*    VFVI22 = AI            (* load analog input into fifo (IMJ uses AIp1)
(*    GOSUB 300              (* increment input pointer
(*    ...
(*    IF VI24 = 0 GOTO 20    (* check for data available
(*    AO = VFVI23            (* load analog output from fifo
(*    GOSUB 310              (* increment output pointer
(*    20                     ...

(*  begin FIFO

(*  Subroutine: Increment FIFO stack input pointer. Call after loading
(*  variable pointed to by VI22 with new input value.
300 VI22 = VI22 + 1                (*  increment input pointer
    IF VI22 >= (VI20+VI21) THEN    (*  reset input pointer if
    VI22 = VI20                    (*    past top of buffer
    VI24 = VI24 + (NOT VB20)       (*  increment stack depth
    VB21 = FALSE                   (*  reset FIFO empty flag
    VB20 = (VI24 >= VI21)          (*  set state of FIFO full flag
    RETURN                         (*  return from subroutine

(*  Subroutine: Increment FIFO stack output pointer.  Call after
(*  retrieving value from variable pointed to by VI23.

310 IF VI24 = 0 GOTO 315           (*  If empty, return from subroutine
    VI23 = VI23 + 1                (*  increment output pointer
    IF VI23 >= (VI20+VI21) THEN    (*  reset output pointer if
    VI23 = VI20                    (*    end of buffer
    VI24 = VI24 - 1                (*  decrement stack depth
    VB21 = (VI24 = 0)              (*  set state of FIFO empty flag
315 RETURN                         (*  return from subroutine

(*  Subroutine: Initialize FIFO

320 VI22 = VI20                    (*  initialize input pointer
    VI23 = VI20                    (*  initialize output pointer
    VI24 = 0                       (*  initialize stack depth
    VB20 = FALSE                   (*  reset FIFO full flag
    VB21 = TRUE                    (*  set FIFO empty flag
    RETURN                         (*  return from subroutine
```

Display and Edit Time/Date on OIP I

```
(* Utility Template:           I_T&DATE.TXT
(* Revision Log:               REV 098OCT19    Original release
(* DspMotion Series:           IMC + OIP-DSP1-C
(* Function:                   Display and Edit Time and Date on Operator Interface
                               Display

(* Operation:                  Time and date displayed at cursor position when called.
(*                             Edit functions use display line in VI100 of 1 to 4.

(* Template includes:

(*  Label                      Function
(* ———————   ————————————————————————————————————————————————————————————
(*  600                        Display Date
(*  610                        Display Time as hh:mm:ss:AM or hh:mm:ss:PM
(*  620                        Edit Date
(*  640                        Edit 24 Time in 24 hour format
(*  650                        Edit Time in AM/PM format
(*  670                        Convert 24 hour time string in VS120 to AM/PM time format
(*  680                        Convert AM/PM time string in VS120 to 24 hour time format

(* Global resources:           Date and Time registers
(*                             VI100 - Pointer to display line to use for edit

(* Module specific resources:  Integer variables VI120, VI121, VI122
(*                             String variables  VS120, VS121

(* Example of Time & Date Display and Edit
program1
    cls
    vi100 = 3                  (* use line 3 for edit
    stm1 = 0.5                 (* update time each .5 sec
010 wait tm1
    crp1.1                     (* position cursor for date
    gosub 600                  (* display date
    crp1.30                    (* position cursor for time
    gosub 610                  (* display time
    crp2.1                     (* position cursor for message
    ekb                        (* empty key buffer
    out "Press A to edit date B or C to edit time"
(*  Key Press          A    B    C    D    E    F    G    H    I    J    K    L
015 function         020, 040, 050, 015, 015, 015, 015, 015, 015, 015, 015, 015
020 gosub 620                  (* edit date
    goto 10
040 gosub 640                  (* edit 24 hr time
    goto 10
050 gosub 650                  (* edit AM/PM time
    goto 10                    (* repeat forever
(* Subroutine: Display Date at present cursor position
(* Revision 1.0                    08/05/98  wbh

(* Format is month day, year i.e.  January 01, 2000

600 out month+" "+rgt(date,2)+", "+lft(date,4)
    return

(* Subroutine: Display Time as hh:mm:ss:AM or hh:mm:ss:PM
(* Revision 1.0                    08/05/98  wbh

(* VS120, VI120 used as scratchpads

610 vs120 = time                       (* capture time
    gosub 670                          (* convert to AM/PM
    if lft(vs120,1)="0" then
    vs120 = " " + rgt(vs120,10)
    out vs120                          (* display
    return

(* Subroutine: Edit date
(* Revision 1.0                    08/05/98  wbh

(* Format is month day, year i.e.  January 01, 2000

(* VI100 contains line number of display to use for edit of 1-4
(* vs121-vs123 and vi121-vi123 used as scratchpad

620 crpvi100.1                         (* move cursor to edit line
    cll
    vs121 = mid(date,2,6)+"-"+rgt(date,2)+"-"+lft(date,4)
    out "Type date, Enter to accept:  "
622 crpvi100.30
    out vs121                          (* write date: mo-da-yrrr
    crpvi100.30
    ekb                                (* flush key input buffer
    in vs122                           (* get new date
    if len(vs122) = 0 goto 632         (* quit if Enter
    if len(vs122) <> 10 goto 628       (* error if wrong # of char
    vi121 = sti(lft(vs122,2))          (* test month
    if (vi121 > 0) and (vi121 < 13) goto 624
    goto 628
624 vi122 = sti(mid(vs122,2,4))        (* test day
    if (vi122 > 0) and (vi122 < 32) goto 626
    goto 628
626 vi123 = sti(rgt(vs122,4))          (* test year
    if (vi123 > 1993) and (vi123 < 2050) goto 630

628 crpvi100.1                         (* error: clear line
    cll
    out "Please re-enter date:        "
    goto 622
```

```
630 vs123 = its(vi123,4)  + “-”             (* assemble year into date string
    vs123 = vs123 + its(vi121,2) + “-”
    vs123 = vs123 + its(vi122,2)
    if mid(vs123,1,6) = “ “ then
    vs123 = lft(vs123,5) + “0” + rgt(vs123,4)
    if mid(vs123,1,9) = “ “ then
    vs123 = lft(vs123,8) + “0” + rgt(vs123,1)
    date = vs123
632 crpvi100.1                             (* clear edit line
    cll
639 return

(* Subroutine: Edit Time in 24 hour format
(* Revision 1.0                 08/05/98  wbh

(* Format is hh:mm:ss
(* VI100 contains line number of display to use for edit of 1-4
(* VS120 used as scratchpad

640 crpvi100.1                             (* clear edit line
    cll
    out “Type 24 hr time as hh.mm.ss:  “
642 crpvi100.30
    vs120 = time                           (* capture time
    out lft(vs120,2)+”.”+mid(vs120,2,4)+”.”+rgt(vs120,2)
    crpvi100.30
    ekb                                    (* flush key input buffer
    in vs120                               (* get new time
    if len(vs120) = 0 goto 648             (* quit if Enter
    if len(vs120) <> 8 goto 644            (* error if wrong # of char
                                           (* check hour
    if sti(lft(vs120,2)) < 0 or sti(lft(vs120,2)) > 23 goto 644
                                           (* check minute
    if sti(mid(vs120,2,4)) < 0 or sti(mid(vs120,2,4)) > 59 goto 644
                                           (* check second
    if sti(rgt(vs120,2)) < 0 or sti(rgt(vs120,2)) > 59 goto 644
    if mid(vs120,1,3) <> “.” goto 644      (* check punctuation
    if mid(vs120,1,6) = “.” goto 646

644 crpvi100.1                             (* write error message
    cll
    out “Please re-enter time:”
    goto 642                               (* go get new input

646 time = lft(vs120,2)+”:”+mid(vs120,2,4)+”:”+rgt(vs120,2)
648 crpvi100.1                             (* clear edit line
    cll
    return                                 (* done

(* Subroutine: Edit Time in AM/PM format
(* Revision 1.0                 08/05/98  wbh
(* Format is hh:mm:ss:AM or hh:mm:ss:PM
(* VI100 contains line number of display to use for edit of 1-4
(* VS120 - VS122 used as scratchpad

650 crpvi100.1                          (* clear edit line
    cll
    out “Type new time as hh.mm.ss:  “
652 crpvi100.30
    vs120 = time                        (* capture time
    gosub 670                           (* convt to AM/PM
    if lft(vs120,1) = “ “ then
    vs120 = “0” + rgt(vs120,10)         (* strip AM/PM
    out lft(vs120,2)+”.”+mid(vs120,2,4)+”.”+mid(vs120,2,7)
    crpvi100.30
    ekb                                 (* flush key input buffer
    in vs120                            (* get new time
    if len(vs120) = 0 goto 668          (* quit if Enter
    if len(vs120) <> 8 goto 654         (* error if wrong # of char
                                        (* check hour
    if sti(lft(vs120,2)) < 1 or sti(lft(vs120,2)) > 12 goto 654
                                        (* check minute
    if sti(mid(vs120,2,4)) < 0 or sti(mid(vs120,2,4)) > 59 goto 654
                                        (* check second
    if sti(rgt(vs120,2)) < 0 or sti(rgt(vs120,2)) > 59 goto 654

    if mid(vs120,1,3) <> “.” goto 654   (* check punctuation
    if mid(vs120,1,6) = “.” goto 656

654 crpvi100.1                          (* write error message
    cll
    out “Please re-enter time:”
    goto 652                            (* go get new input
656 vs122 = lft(vs120,2)+”:”+mid(vs120,2,4)+”:”+rgt(vs120,2)
    crpvi100.1                          (* prompt for AM/PM
    cll
    out “Press 1 for AM, 2 for PM:  “
658 ekb                                 (* flush key input buffer
    get vs121                           (* get response
    out vs121                           (* echo response
    if vs121 = “1” goto 660             (* am
    if vs121 = “2” goto 662             (* pm
    crpvi100.1                          (* error
    cll                                 (* prompt for new entry
    out “Please re-enter: 1=AM, 2=PM:  “
    goto 658
```

```
660 vs120 = vs122 + “:AM”
    goto 664
662 vs120 = vs122 + “:PM”
664 gosub 680                    (* convert to 24 hour
    time = vs120                 (* save new time
668 crpvi100.1                   (* clear edit line
    cll
669 return

(* Subroutine: Convert 24 hour time string in VS120 to AM/PM time format
(* Rev 1.0                    08/05/98   wbh

(*  Input format is hh:mm:ss  Output format is hh:mm:ss:AM or hh:mm:ss:PM
(*  VI120 used as scratchpad

670 vi120 = sti(lft(vs120,2))                 (* convt hr to integer
    if vi120 < 12 goto 672                    (* if hr < 12 then AM
                                              (* time is PM
    if vi120 > 12 then                        (* if hour > 12 then
    vi120 = vi120 - 12                        (* hour = hour – 12
    vs120 = its(vi120,2)+rgt(vs120,6)+”:PM”   (* assemble time string
    goto 674
                                              (* time is AM
672 if vi120 < 1 then                         (* if hour < 1
    vi120 = 12                                (* then 12 am
    vs120 = its(vi120,2)+rgt(vs120,6)+”:AM”   (* assemble time string
674 return

(* Subroutine: Convert AM/PM time string in VS120 to 24 hour time format
(* Rev 1.0                    08/05/98   wbh

(*  Input format is hh:mm:ss:AM or hh:mm:ss:PM  Output format is hh:mm:ss
(*  VI120 used as scratchpad

680 vi120 = sti(lft(vs120,2))                 (* convt hour to integer
    if rgt(vs120,2) = “AM” goto 682           (* test for AM or PM
    if vi120 < 12 then                        (* map PM to 24 hour
    vi120 = vi120 + 12
    goto 684

682 if vi120 > 11 then                        (* map AM to 24 hour
    vi120 = vi120 – 12
684 vs120 = its(vi120,2)+mid(vs120,6,3)       (* assemble time string
    if lft(vs120,1) = “ “ then
    vs120 = “0” + rgt(vs120,7)
    return

end
```

Jog Using Analog Input

```
(* Notes:
(* 1- Registers that have been previously loaded with appropriate values
(*    do not have to be reloaded for this motion.
(* 2- Loading the MAC register also loads the MDC register with the same value. To set MDC to a value
(*    different from MAC, load MDC after loading the MAC register.
(* 3- Loading the MFA register also loads the MFD register with the same value. To set MFD to a value
(*    different from MFA, load MFD after loading the MFA register.
(* 4- The Motion Feedrate Percentage Register, MFP, slows time by the % specified. Thus the velocity is
(*    scaled by MFP.  Since acceleration is proportional to 1/(t^2), the acceleration is scaled by (MFP)^2.
(* 5- Since this template incorporates labels and commands that are not allowed in motion
(*    blocks (GOTO), it can be used only in a program.
```

IMC & IMJ Template

```
(* Motion Template:          I_RJANAI.TXT
(* Revision Log:             REV 098OCT08
(* DspMotion Series:         IMC & IMJ
(* Move Type:                Jog using analog input
(* Engineering Units:        Motor revolutions: i.e., URA = position feedback resolution

(* Motion:                   Jog axis in response to the analog input.  The axis will move at
(*                           a velocity that is proportional to the analog input.

(* Variables used:           VF10    velocity scale factor, (units/sec)/volt
(*                           VF11    computed feedrate percentage
(*                           VF12    maximum velocity, units/sec
(* MT = VEL                  This register cannot be loaded if motion is in progress.
(*                           MT does not need to be set unless it is set to a MT setting other
(*                           than VEL.  The default value for MT is VEL.

     AIB = 0.0               (* set analog input deadband, volts (IMJ requires AIBp1=0.0)
     AIO = 0.0               (* set analog input offset, volts (IMJ requires AIOp1)
     VF10 = 4.0              (* set velocity scale factor, (units/sec)/volt
     VF12 = 40.0             (* set maximum velocity, units/sec
     MFA = 500               (* set motion feedrate acceleration, feedrate % / sec
     MFD = 650               (* set motion feedrate deceleration, feedrate % / sec
     MFP = 0.0               (* set motion feedrate percentage, % of velocity
     MAC = 200000.0          (* set motion acceleration, units/sec^2
     MDC = 200000.0          (* set motion deceleration, units/sec^2
     MJK = 0                 (* set motion jerk percentage, % of accel & decel interval
     MVL = 40.0              (* set motion velocity, units/sec
     WAIT MFP = 0.0          (* wait for feedrate to decrease to 0.0
     RVF                     (* run forward
001  VF11 = ((AI * VF10) / VF12) * 100.
                             (* compute feedrate percentage (IMJ requires AIp1)
     IF VF11 < 0.5 THEN      (* if feedrate < minimum allowed then
     VF11 = 0.0                 (*  set feedrate to 0
     IF VF11 > 100.0 THEN       (* if feedrate > maximum allowed then
     VF11 = 100.0               (*  set feedrate to maximum
     MFP = VF11                 (* set motion feedrate percentage
     GOTO 1                     (* go back and compute new feedrate
```

Target Template

```
(* Motion Template:          T_RJANAI.TXT
(* Revision Log:             REV 098OCT08
(* DspMotion Series:         Target ARS
(* Move Type:                Jog using analog input
(* Engineering Units:        Motor revolutions: i.e., URA1 = position feedback resolution

(* Motion:                   Jog axis 1 in response to analog input 1 of analog module 1.
(*                           The axis will move at a velocity that is proportional to the
(*                           analog input.

(* Variables used:           VF10    velocity scale factor, (units/sec)/volt
(*                           VF11    computed feedrate percentage
(*                           VF12    maximum velocity, units/sec
(* MT1 = VEL                 This register cannot be loaded if motion is in progress.
(*                           MT does not need to be set unless it is set to a MT setting other
(*                           than VEL.  The default value for MT is VEL.

     AIB1.1 = 0.0               (* set analog input deadband, volts
     AIO1.1 = 0.0               (* set analog input offset, volts
     VF10 = 4.0                 (* set velocity scale factor, (units/sec)/volt
     VF12 = 40.0                (* set maximum velocity, units/sec
     MFA1 = 500                 (* set motion feedrate acceleration, feedrate % / sec
     MFD1 = 650                 (* set motion feedrate deceleration, feedrate % / sec
     MFP1 = 0.0                 (* set motion feedrate percentage, % of velocity
     MAC1 = 200000.0            (* set motion acceleration, units/sec^2
     MDC1 = 200000.0            (* set motion deceleration, units/sec^2
     MJK1 = 0                   (* set motion jerk percentage, % of accel & decel interval
     MVL1 = 40.0                (* set motion velocity, units/sec
     WAIT MFP1 = 0.0            (* wait for feedrate to decrease to 0.0
     RVF1                       (* run axis 1 forward
001  VF11 = ((AI1.1 * VF10) / VF12) * 100.
                                (* compute feedrate percentage
     IF VF11 < 0.5 THEN         (* if feedrate < minimum allowed then
     VF11 = 0.0                 (* set feedrate to 0
     IF VF11 > 100.0 THEN       (* if feedrate > maximum allowed then
     VF11 = 100.0               (* set feedrate to maximum
     MFP1 = VF11                (* set motion feedrate percentage
     GOTO 1                     (* go back and compute new feedrate
```

Jog Using Electronic Handwheel *I*

IMC Template

```
(* Motion Template:        I_RJELHW.TXT
(* Revision Log:           REV 098OCT08
(* DspMotion Series:       IMC
(* Move Type:              Jog using electronic handwheel
(* Engineering Units:      Motor revolutions: i.e., URA = position feedback resolution
(*                         URX = auxiliary feedback resolution

(* Motion:                 Move axis in relation to the electronic handwheel.  The
(*                         electronic handwheel is used in place of the auxiliary input as a
(*                         means of positioning for electronic gearing.  The axis will
(*                         follow the auxiliary input based on the values of GRN and
(*                         GRD, i.e.,
(*
(*                                                 GRN
(*                                 axis pulses = ——— * handwheel pulses
(*                                                 GRD

(* Notes:
(* 1- Registers that have been previously loaded with appropriate values
(*    do not have to be reloaded for this motion.
(* 2- The electronic handwheel input can be connected to the auxiliary input or to digital inputs
(*    5 and 6.  Setting HWE = ON enables digital inputs 5 and 6 as the handwheel inputs.

(* MT = VEL                This register cannot be loaded if motion is in progress.
(*                         MT does not need to be set unless it is set to a MT setting other
(*                         than VEL.  The default value for MT is VEL.

    GRN = 1                (* set gearing numerator
    GRD = 1                (* set gearing denominator
    GRB = 0                (* set gearing bound
    GRF = 0                (* set gearing filter constant
    GRE = ON               (* enable electronic gearing
    HWE = ON               (* enable digital inputs 5 and 6 electronic handwheel
```

Jog Using Single-Pole, Double-Throw Switch

```
(* Notes:
(* 1- Registers that have been previously loaded with appropriate values
(*     do not have to be reloaded for this motion.
(* 2- Loading the MAC register also loads the MDC register with the same value.  To set MDC to a value
(*    different from MAC, load MDC after loading the MAC register.
(* 3- This example assumes the MFP (Motion Feedrate Percentage) is set to its default value of 100.  See
(*    IMC files I_MVABSF.TXT and I_MT1ABF.TXT or Target files T_MVABSF.TXT and
(*    T_MT1ABF.TXT for examples using the MFP register.
(* 4- Since this template incorporates labels and commands that are not allowed in motion blocks
(*    (GOTO), it can only be used in a program.
```

IMC & IMJ Template

```
(* Motion Template:         I_RJSPOL.TXT
(* Revision Log:            REV 098OCT05
(* DspMotion Series:        IMC & IMJ
(* Move Type:               Jog using single-pole, double-throw switch
(* Engineering Units:       Motor revolutions: i.e., URA = position feedback resolution

(* Motion:                  Jog axis in response to a single-pole, double-throw switch.  Jog
(*                          axis forward while digital input 1 is true.  Jog axis reverse
(*                          while digital input 2 is true.
(* MT = VEL                 This register cannot be loaded if motion is in progress.
(*                          MT does not need to be set unless it is set to a MT setting other
(*                          than VEL.  The default value for MT is VEL.

    MAC = 50.0              (* set motion acceleration, units/sec^2
    MDC = 75.0              (* set motion deceleration, units/sec^2
    MJK = 0                 (* set motion jerk percentage,  % of accel & decel interval
    MVL = 1.0               (* set motion velocity, units/sec
001 WAIT DI1 OR DI2         (* wait for digital input 1 or 2 to turn on
    IF DI2 GOTO 20          (* goto label 20 if digital input 2 on
    RVF                     (* run forward
    WAIT NOT DI1            (* wait for digital input 1 to turn off
    ST                      (* stop all motion
    GOTO 1                  (* go back and wait for digital input
020 RVR                     (* run reverse
    WAIT NOT DI2            (* wait for digital input 2 to turn off
    ST                      (* stop all motion
    GOTO 1                  (* go back and wait for digital input
```

Target Template

```
(* Motion Template:          T_RJSPOL.TXT
(* Revision Log:             REV 098OCT05
(* DspMotion Series:         Target ARS
(* Move Type:                Jog using single-pole, double-throw switch
(* Engineering Units:        Motor revolutions: i.e., URA1 = position feedback resolution

(* Motion:                   Jog axis 1 in response to a single-pole, double-throw switch.
(*                           Jog axis 1 forward while digital input 1 of digital I/O module 1
(*                           is true.  Jog axis 1 reverse while digital input 2 of digital I/O
(*                           module 1 is true.
(* MT1 = VEL                 This register cannot be loaded if motion is in progress.
(*                           MT does not need to be set unless it is set to a MT setting other
(*                           than VEL.  The default value for MT is VEL.

    MAC1 = 50.0              (* set motion acceleration, units/sec^2
    MDC1 = 75.0              (* set motion deceleration, units/sec^2
    MJK1 = 0                 (* set motion jerk percentage,  % of accel & decel interval
    MVL1 = 1.0               (* set motion velocity, units/sec
001 WAIT DI1.1 OR DI1.2      (* wait for digital input 1 or 2 to turn on
    IF DI1.2 GOTO 20         (* goto label 20 if digital input 2 on
    RVF1                     (* run axis 1 forward
    WAIT NOT DI1.1           (* wait for digital input 1 to turn off
    ST1                      (* stop all motion
    GOTO 1                   (* go back and wait for digital input
020 RVR1                     (* run axis 1 reverse
    WAIT NOT DI1.2           (* wait for digital input 2 to turn off
    ST1                      (* stop all motion
    GOTO 1                   (* go back and wait for digital input
```

Jog Using Operator Interface (OIP)

```
(* Notes:
(* 1- Registers that have been previously loaded with appropriate values
(*    do not have to be reloaded for this motion.
(* 2- Loading the MAC register also loads the MDC register with the same value.  To set MDC to a value
(*    different from MAC, load MDC after loading the MAC register.
(* 3- This example assumes the MFP (Motion Feedrate Percentage) is set to its default value of 100.  See
(*    IMC & IMJ files I_MVABSF.TXT and I_MT1ABF.TXT or Target files T_MVABSF.TXT and
(*    T_MT1ABF.TXT for examples using the MFP register.
(* 4- Since this template incorporates labels and commands that are not allowed in motion blocks
(*    (GOTO, FUNCTION), it can be used only in a program.
(* 5- In order for this example to work, KYA1 and KYA2 (the key assignments for function keys A and
(*    B) must be set to DOUBLE in the initialization of the IMC and the Target.  This allows both the
(*    key press and the key release codes to be put in the key buffer.
```

IMC & IMJ Template

```
(* Motion Template:          I_RJOIP.TXT
(* Revision Log:             REV 098OCT08
(* DspMotion Series:         IMC & IMJ
(* Move Type:                Jog using OIP
(* Engineering Units:        Motor revolutions: i.e., URA = position feedback resolution

(* Motion:                   Jog axis while a key on the Operator Interface display
(*                           is pressed.  The twelve function keys on the left-hand side of
(*                           the OIP are programmable.  In this case, keys A and B
(*                           are programmed to jog the axis forward and reverse.


001 EKB                      (* empty key buffer
    FUNCTION 100,200,1,1,1,1,1,1,1,1,1,1
                             (* go to label associated with key pressed
    GOTO 1                   (* go back and check for key press
100 MAC = 50.0               (* set motion acceleration, units/sec^2
    MDC = 75.0               (* set motion deceleration, units/sec^2
    MJK = 0                  (* set motion jerk percentage, % of accel & decel interval
    MVL = 1.0                (* set motion velocity, units/sec
    RVF                      (* run forward
    WAIT KEY                 (* wait for character in key buffer
    ST                       (* stop all motion
    GOTO 1                   (* go back and check for key press
200 MAC = 50.0               (* set motion acceleration, units/sec^2
    MDC = 75.0               (* set motion deceleration, units/sec^2
    MJK = 0                  (* set motion jerk percentage, % of accel & decel interval
    MVL = 1.0                (* set motion velocity, units/sec
    RVR                      (* run reverse
    WAIT KEY                 (* wait for character in key buffer
          ST                          (* stop all motion
          GOTO 1                      (* go back and check for key press
```

Target Template

```
(* Motion Template:           T_RJOIP.TXT
(* Revision Log:              REV 098OCT08
(* DspMotion Series:          Target ARS
(* Move Type:                 Jog using OIP
(* Engineering Units:         Motor revolutions: i.e., URA = position feedback resolution

(* Motion:                    Jog axis 1 while a key on the Operator Interface
(*                            display is pressed.  The twelve function keys on the left-hand
(*                            side of the OIP are programmable.  In this case, keys
(*                            A and B are programmed to jog the axis forward and reverse.

001  EKB                      (* empty key buffer
     FUNCTION 100,200,1,1,1,1,1,1,1,1,1,1
                              (* go to label associated with key pressed
GOTO 1                        (* go back and check for key press
100  MAC1 = 50.0              (* set motion acceleration, units/sec^2
     MDC1 = 75.0              (* set motion deceleration, units/sec^2
     MJK1 = 0                 (* set motion jerk percentage,
                              (*  % of accel & decel interval
     MVL1 = 1.0               (* set motion velocity, units/sec
     RVF1                     (* run axis 1 forward
     WAIT KEYW                (* wait for character in key buffer
     ST1                      (* stop all motion
     GOTO 1                   (* go back and check for key press
200  MAC1 = 50.0              (* set motion acceleration, units/sec^2
     MDC1 = 75.0              (* set motion deceleration, units/sec^2
     MJK1 = 0                 (* set motion jerk percentage, % of accel & decel interval
     MVL1 = 1.0               (* set motion velocity, units/sec
     RVR1                     (* run axis 1 reverse
     WAIT KEYW                (* wait for character in key buffer
     ST1                      (* stop all motion
     GOTO 1                   (* go back and check for key press
```

Multi-axis Path Recording

```
(* Motion Template:          T_RT4PR.TXT
(* Revision Log:             REV 098OCT06
(* DspMotion Series:         Target ARS
(* Move Type:                Multi-axis path recording
(* Engineering Units:        Motor revolutions: i.e., URA1 = axis 1 position feedback resolution
(*                           URA2 = axis 2 position feedback resolution, etc.

(* Motion:                   Record and playback a sequence of positions that the axes
(*                           move.  During record, the axes will move in response to the
(*                           auxiliary inputs.  Digital input 1 of digital I/O module 1 will
(*                           start record, and digital input 2 will start playback.

(* Variables used:           VI1, VI3, VI5, VI7
(*                           axis 1-4 position pointer end
(*                           VI2, VI4, VI6, VI8
(*                           axis 1-4 position pointer begin

(* Notes:
(* 1- Registers that have been previously loaded with appropriate values
(*    do not have to be reloaded for this motion.
(* 2- The position pointers point to extended variable space.  Since this example uses integer
(*    variables up to 84999, the extended floating point variable allocation, VFEA, must be set
(*    to a value <= 90620.  See VFEA in the system registers for more information on extended
(*    integer and floating point variable allocation.
(* 3- This example assumes the MFP (Motion Feedrate Percentage) is set to its default value of
(*    100.  See files T_MVABSF.TXT and T_MT1ABF.TXT for examples using the MFP
(*    register.
(* 4- This example assumes the gearing registers GRN, GRD, GRF and GRB are set to their
(*    default values.  See files T_MS1EG.TXT and T_MS4EG.TXT for examples using these
(*    registers.
(* 5- Since this template incorporates labels and commands that are not allowed in motion
(*    blocks (GOTO) it can only be used in a program.

(* MT1 = VEL
(* MT2 = VEL
(* MT3 = VEL
(* MT4 = VEL                 The MT register cannot be loaded if motion is in progress.
(*                           MT does not need to be set unless it is set to a MT setting other
(*                           than VEL.  The default value for MT is VEL.

    GRI1 = PSX1              (* set axis 1 gearing input to axis 1 aux input
    GRI2 = PSX2              (* set axis 2 gearing input to axis 2 aux input
    GRI3 = PSX3              (* set axis 3 gearing input to axis 3 aux input
    GRI4 = PSX4              (* set axis 4 gearing input to axis 4 aux input
    IPB1 = 0.005             (* set in position band, units
    IPB2 = 0.005             (* set in position band, units
    IPB3 = 0.005             (* set in position band, units
    IPB4 = 0.005             (* set in position band, units
    VI1 = 5001               (* set axis 1 default position pointer end
    VI3 = 25001              (* set axis 2 default position pointer end
    VI5 = 45001              (* set axis 3 default position pointer end
    VI7 = 65001              (* set axis 4 default position pointer end
001  WAIT DI1.1 OR DI1.2      (* wait for record input or playback input
    IF DI1.2 GOTO 20         (* if playback input goto 20
    GRE1234 = ON             (* enable electronic gearing
    PPE1 = 24999             (* set axis 1 position pointer end
    PPB1 = 5000              (* set axis 1 position pointer begin
    PPE2 = 44999             (* set axis 2 position pointer end
    PPB2 = 25000             (* set axis 2 position pointer begin
    PPE3 = 64999             (* set axis 3 position pointer end
    PPB3 = 45000             (* set axis 3 position pointer begin
    PPE4 = 84999             (* set axis 4 position pointer end
    PPB4 = 65000             (* set axis 4 position pointer begin
    PPI1 = 1.0               (* set axis 1 position pointer interval
    PPI2 = 1.0               (* set axis 2 position pointer interval
    PPI3 = 1.0               (* set axis 3 position pointer interval
    PPI4 = 1.0               (* set axis 4 position pointer interval
    WAIT NOT IPALL           (* wait for one of the axes to start moving
    REC1234 = ON             (* enable record positions
    WAIT NOT DI1.1           (* wait for not record input
    REC1234 = OFF            (* disable record positions
    GRE1234 = OFF            (* disable electronic gearing
    VI1 = PP1 - 1            (* get last valid axis 1 position pointer
    VI3 = PP2 - 1            (* get last valid axis 2 position pointer
    VI5 = PP3 - 1            (* get last valid axis 3 position pointer
    VI7 = PP4 - 1            (* get last valid axis 4 position pointer
    GOTO 1                   (* go back and check for inputs
020  PPE1 = VI1               (* set axis 1 position pointer end
    PPB1 = 5000              (* set axis 1 position pointer begin
    PPE2 = VI3               (* set axis 2 position pointer end
    PPB2 = 25000             (* set axis 2 position pointer begin
    PPE3 = VI5               (* set axis 3 position pointer end
    PPB3 = 45000             (* set axis 3 position pointer begin
    PPE4 = VI7               (* set axis 4 position pointer end
    PPB4 = 65000             (* set axis 4 position pointer begin
    PPI1 = 1.0               (* set axis 1 position pointer interval
    PPI2 = 1.0               (* set axis 2 position pointer interval
    PPI3 = 1.0               (* set axis 3 position pointer interval
    PPI4 = 1.0               (* set axis 4 position pointer interval
    VI2 = PPB1               (* get axis 1 position pointer begin
    MPA1 = ITF(VIVI2)/ITF(URA1)
                             (* set axis 1 absolute move position, units
    VI4 = PPB2               (* get axis 2 position pointer begin
    MPA2 = ITF(VIVI4)/ITF(URA1)
                             (* set axis 2 absolute move position, units
    VI6 = PPB3               (* get axis 3 position pointer begin
    MPA3 = ITF(VIVI6)/ITF(URA1)
                             (* set axis 3 absolute move position, units
    VI8 = PPB4               (* get axis 4 position pointer begin
    MPA4 = ITF(VIVI8)/ITF(URA1)
```

```
                              (* set axis 4 absolute move position, units
MVL1 = 1.0                    (* set motion velocity, units/sec
MVL2 = 1.0                    (* set motion velocity, units/sec
MVL3 = 1.0                    (* set motion velocity, units/sec
MVL4 = 1.0                    (* set motion velocity, units/sec
RPA1234                       (* run axes to absolute position
WAIT IPALL                    (* wait for all axes to run back to beginning
PLY1234 = ON                  (* enable playback of recorded positions
WAIT PP1 = VI1                (* wait for end of run
WAIT NOT DI1.2                (* wait for not play input
GOTO 1                        (* go back and check for inputs
```

Report Controller Faults to OIP

I, jr

```
(* Utility Template:             I_FAULTS.TXT
(* Revision Log:                 REV 098OCT16     Original release
(* DspMotion Series:             IMC or IMJ and OIP-DSP1-C
(* Function:                     Report DspMotion Controller fault(s) to OIP

(* Operation:                    Write all currently active faults to line 4 of the OIP until OIP
(*                               key is pressed.  Active faults are scrolled at the rate of one per
(*                               2 seconds.

(* Global resources:             Serial communication port

(* Module specific resources:    VI101 is system fault bit counter
(*                               VI102 is network fault bit counter

(* Example of  use:
program4
(*    gosub 100                  (* display active faults
(*    function ....              (* decode key press

(* Begin IMC or IMJ fault reporting
100 crp4.1                              (* position cursor on fault
    cll                                 (*  display line & clear line
105 vi101 = 0                           (* initialize fault count
110  if fc <> 0 goto 115                (* if fault, go decode
     cll                                (* else clear line
     out "Controller Functional"        (*  and write ok message
     stm4 = 2                           (* start 2 sec timer
     wait tm4 when key goto 135         (* wait for timer or key press
     goto 105                           (*  repeat
115  if not fcvi101 goto 130            (* test for fault
     cll                                (* if fault, clear fault line
     if vi101 <> 2 goto 118             (* if not STF fault, go on
     out "Machine Fault"                (* else write new message
     goto 125                           (* go to pause
118  if vi101 <> 26 goto 120            (* if not excessive clamp, go on
     out "Excessive Motor Clamp - Under Voltage"            (* else write message
     goto 125                           (* go to pause
120  out $fcvi101                       (* report fault
125  stm4 = 2                           (* start 2 sec timer
     wait tm4 when key goto 135         (* wait for timer or key press
130  vi101 = vi101 + 1                  (* increment fault count
     if vi101 < 32 goto 110             (* test for faults done
     if fcn <> 0 gosub 140              (* report network faults
     goto 105                           (* repeat
135  RETURN
```

```
(* Begin IMC or IMJ Network fault reporting
140 vi102 = 0                        (* initialize fault count
145 if not fcnvi102 goto 150         (* test for fault
    cll                              (* if fault, clear fault line
    out $fcnvi102                    (* report fault
    stm4 = 2                         (* start 2 sec timer
    wait tm4 when key goto 155       (* wait for timer or key press
150 vi102 = vi102 + 1                (* increment fault count
    if vi102 < 11 goto 145           (* repeat if not faults done
155 return

(* End IMC or IMJ fault reporting
```

Retriggerable One-Shot

```
(* Utility Template:          I_1SHOT.TXT
(* Revision Log:              REV 097MAY16    Original release
(* DspMotion Series:          IMC/IMJ
(* Function:                  Retriggerable One Shot

(* Operation:                 Implement one-shot output on DO12 and DO13 with
(*                            programmable on-delay and programmable off-delay.
(*                            One-shots are triggered by VB27 and VB28.

(* NOTE: to maintain accurate timing, call this module every 10 ms.
(*
(* Global resources:
(*    vb27                    DO12 one shot input
(*    vb28                    DO13 one shot input
(*    vf40                    DO12 on-delay time, sec
(*    vf41                    DO12 off-delay time, sec
(*    vf42                    DO13 on-delay time, sec
(*    vf43                    DO13 off-delay time, sec

(* Module specific resources:
(*    Labels 500 through 549
(*    tm7 and tm8             One shot timers
(*    vb120                   DO12 output on_delay timer flag
(*    vb121                   tm12 timer state
(*    vb122                   DO13 output on_delay timer flag
(*    vb123                   tm8 timer state
(*    do7                     one shot output
(*    do8                     one shot output

(* Example of One-Shot use:
(*    do12 = off              (* initialize outputs to off
(*    do13 = off
(*    vb120 = off             (* initialize states to off
(*    vb121 = off
(*    vb122 = off
(*    vb123 = off
(*    stm1 = 0.01             (* initialize i/o scan timer
(* 005    wait tm1            (* wait for scan tick
(*    ...                     (* body of i/o scan
(*    gosub 500               (* execute ONE_SHOT
(*    goto 05                 (* repeat scan
(* Begin One_shot
(* Start DO7 One-Shot if vb27
500 if not vb27 goto 505          (* if no input go check timers
    vb27 = false                  (* reset input trigger
    if vf40 < 0.005 goto 510      (* no delay if time < 5 ms
    stm7 = vf40                   (* start on-delay timer
    vb120 = true                  (* set on-delay timer flag
505 vb121 = tm7                   (* capture timer state
    if (not vb120) or (not vb121) goto 515
                                  (* if timing, continue else start off-delay
510 if vf41 < 0.005 goto 516      (* no delay for short times
    stm7 = vf41                   (* start off-delay timer
    vb120 = false                 (* cancel on-delay flag
    do12 = on                     (* turn on output
    goto 520                      (* go check next input
515 if vb120 or (not vb121) or (not do12) goto 520
                                  (* if timing continue
516 do12 = off                    (* else turn off output

(* Start DO13 One-Shot if vb28
520 if not vb28 goto 525          (* if no input go check timers
    vb28 = false                  (* reset input trigger
    if vf42 < 0.005 goto 530      (* no delay if time < 5 ms
    stm8 = vf42                   (* start on-delay timer
    vb122 = true                  (* set on-delay timer flag
525 vb123 = tm8                   (* capture timer state
    if (not vb122) or (not vb123) goto 535
                                  (* if timing, continue else start off-delay
530 if vf43 < 0.005 goto 536      (* no delay for short times
    stm8 = vf43                   (* start off-delay timer
    vb122 = false                 (* cancel on-delay flag
    do13 = on                     (* turn on output
    goto 540                      (* go check next input
535 if vb122 or (not vb123) or (not do13) goto 540
                                  (* if timing continue
536 do13 = off                    (* else turn off output

540  return
(*  End One_shot
```

Solve PID Algorithm

```
(* Utility Template:          A_PID.TXT
(* Revision Log:              REV 097SEP12     Original release
(* DspMotion Series:          All
(* Function:                  Proportional, Integral, Derivative Controller with bounded integrator

(* Operation:                 Solve PID algorithm:
(*                            output(n) = KA*command(n) + KP*error(n) + KI*sum(error(N))
(*                             + KD*{0.2083646*[error(n-1) - error(n-2)]
(*                                       - .0285944*[error(n) - error(n-3)]}

(* Global resources:
(*  PID parameters
(*    VF20                    KP, proportional gain
(*    VF21                    KI, integral gain
(*    VF22                    KD, derivative gain
(*    VF23                    KF, feed forward gain
(*    VF24                    integrator bound
(*  Inputs
(*    VF100                   command(n)
(*    VF101                   error(n)
(*  Output
(*    VF102                   PID output(n)

(*  Module specific registers:
(*    VF103                   error(n-1)
(*    VF104                   error(n-2)
(*    VF105                   error(n-3)
(*    VF106                   integrator accumulator
(*    VF107                   derivative result

(* Example PID initialization:
(*    VF20 = 1.0              (* set proportional gain
(*    VF21 = .01              (* set integral gain
(*    VF22 = 10.0             (* set derivative gain
(*    VF23 = 0.0              (* set feed forward gain
(*    VF24 = 7.5              (* set integrator bound
(*    VF103 = 0.0             (* reset PID state to zero
(*    VF104 = 0.0
(*    VF105 = 0.0
(*    VF106 = 5.0             (* preset integrator with command

(* Example PID use: input is analog input, output is analog output
(*    STM2 = 0.01             (* initialize control loop timer
(* 005    WAIT TM2            (* wait for timer
(*    VF100 = 5.0             (* load command
(*    VF101 = VF100 - AI      (* compute error (IMJ requires AIp1)
(*    CALL 100                (* execute PID
(*    IF VF102 > 10. THEN     (* bound output
(*   VF102 = 10.
(*   IF VF102 < -10. THEN
(*   VF102 = -10.
(*   AO = VF102                    (* set output
(*   ...                           (* other control loop functions
(*   GOTO 05                       (* repeat

(* Begin PID
(* Compute integral term: accum = accum + (error * KI)
100 vf106 = vf106 + vf101 * vf21        (* add error to accumulator
    if vf106 < -vf24 then               (* lower integrator bound
    vf106 = -vf24
    if vf106 > vf24 then                (* upper intergrator bound
    vf106 = vf24
(* Compute derivative term using 4th order FIR filter
    vf107 = (vf101 - vf105) * -0.0285944 + (vf103 - vf104) * 0.2083646
    vf105 = vf104                       (* update history registers
    vf104 = vf103
    vf103 = vf101
(* Compute PID output
    vf102 = vf101*vf20 + vf106 + vf107*vf22 + vf100*vf23
    return
(* End PID
```

Torque-Limited Pressing

```
(* Utility Template:    I_TLPRESS.TXT
(* Revision Log:        REV 098OCT23     Original release
(* DspMotion Series:    IMC & IMJ
(* Function:            Torque limited pressing

(* Operation:           Run motor to press workpiece.  Pressing operation ends when specified torque
(*                      limit or maximum press travel is reached.  Set cycle complete and workpiece
(*                      accept outputs.

(*  Global resources:
(*    DI1               Input of start cycle
(*    DI2               Input of stop cycle
(*    DO12              Output of part accepted
(*    DO13              Output of cycle complete

(*    VB01              At cycle start flag
(*    VB02              Motion has stopped flag
(*    VB03              Press reached torque limit flag

(*    VF01              Press acceleration, units/sec^2
(*    VF02              Press deceleration, units/sec^2
(*    VF03              Press jerk, % of accel and decel interval
(*    VF04              Press velocity, units/sec
(*    VF05              Cycle start position, units
(*    VF06              Maximum press travel, units
(*    VF07              Press torque limit current, % maximum  continuous current
(*    VF08              Minimum acceptable part location, units
(*    VF09              Maximum acceptable part location, units
(*    VF10              Retract acceleration, units/sec^2
(*    VF11              Retract deceleration, units/sec^2
(*    VF12              Retract jerk, % of accel and decel interval
(*    VF13              Retract velocity, units/sec
(*    VF14              Press location at torque limit, units

(* Module specific resources:
(* Example of torque limited pressing invocation:
(*    do12 = off        (* cancel part accepted output
(*    do13 = off        (* cancel cycle done output
(*    goto 001

(* Begin torque limited pressing:
001 WAIT EG1 AND NOT DI2            (* wait for cycle start input
    DO13 = OFF                      (* turn off cycle complete output
    IF VB01 GOTO 005                (* if not at start position
    VB01 = FALSE                    (* then reset at cycle start flag
    EXM1                            (*  run to cycle start position
    WAIT VB01 WHEN EG2 GOTO 010             (* wait until at start position if cycle stop, go stop
005 IF EG2 GOTO 010                 (*  when cycle stop, go stop
    VB03 = FALSE                    (* reset reached torque limit flag
    MAC = VF01              (* set run acceleration, units/sec^2
    MDC = VF02              (* set deceleration, units/sce^2
    MJK = FTI(VF03)              (* set motion jerk percentage,  % of accel & decel interval
    MVL = VF04                   (* set run velocity
    MPA = VF06                   (* set maximum distance to run
    TLC = VF07                   (* set torque limit current, % continuous current
    TLE = ON                     (* enable torque limit
    RPA                          (* run to position with torque limit
    WAIT (IP OR TL OR EG2)       (* wait until in position or torque limit
    VB03 = TL                    (* save torque limit state
    VF14 = PSA                   (* save axis position
    DO12 = VB03 AND PSA>=VF08 AND PSA<= VF09       (* set accept output
    ST                           (* stop axis motion
    STM2 = 1                     (* pause at torque limit
    WAIT TM2
    TLE = OFF                    (* disable torque limit

    VB01 = FALSE                 (* reset at cycle start flag
    EXM1                         (* retract to home
    WAIT VB01 WHEN EG2 GOTO 010                (* wait for move complete
                                 (*  when cycle stop, go stop
    DO13 = ON                    (* turn on cycle complete output
    GOTO 001                     (* go to start of program

010  VB02 = FALSE                 (* reset stopped flag
    EXM2                         (* stop motion
    WAIT VB02                    (* wait for motion stopped
    GOTO 001                     (* go to start of program

(* Motion Blocks
    Motion1                      (* Run reverse to cycle start position
    MAC = VF10                   (* set acceleration, units/sec^2
    MDC = VF11                   (* set deceleration, units/sce^2
    MJK = FTI(VF12)              (* set motion jerk percentage,  % of accel & decel interval
    MVL = VF13                   (* set run velocity
    MPA = VF05                   (* set position
    RPA                          (* run to position
    WAIT IP                      (* wait for axis to be in position
    VB01 = TRUE                  (* set at cycle start flag
    End                          (* End motion block

Motion2                           (* Stop motion
    MDC = VF11                   (* set deceleration, units/sce^2
    MJK = FTI(VF12)              (* set motion jerk percentage, % of accel & decel interval
    ST                           (* stop
    VB01 = FALSE                 (* reset at start position
    VB02 = TRUE                  (* set motion stopped flag
    End
```

Two-Hand Anti-Tiedown

```
(* Utility Template:          I_2HAND.TXT
(* Revision Log:              REV 098OCT16  Original release
(* DspMotion Series:          IMC & IMJ
(* Function:                  Two hand anti_tiedown

(* Operation:                 Implement anti_tiedown on inputs DI1 and DI2.
(*                            VI110 = 30 while (DI1 AND DI2) if DI1 and DI2
(*                            occur within 0.5 seconds of one another.

(*  Global resources:
(*    DI1                     anti_tiedown input
(*    DI2                     anti_tiedown input
(*    VI110                   anti_tiedown state of
(*                            idle of 10              waiting for input
(*                            armed of 20             waiting for 2nd input
(*                            active of 30            both inputs active
(*                            relax of 40             waiting for no inputs

(* Module specific resources:
(*    TM03                    anti_tiedown timer

(* Example of anti_tiedown use:
(*    vi110 = 040             (* initialize state to relax
(*    do12 = off              (* “active” output
(*    stm1 = .025             (* initialize i/o scan timer
(* 005    wait tm1            (* wait for i/o scan timer
(*    gosub vi110             (* scan: goto state
(*    do12 = (vi110 = 30)     (* set/reset output
(*    ...                     (* rest of i/o scan
(*    goto 005                (* repeat

(* Begin anti_tiedown
(* state is idle - wait for either input
010 if not (di1 or di2) goto 015(* if no input return to scan
    stm3 = .5                 (* start timer
    vi110 = 20                (* state is armed
015  return                   (* return to scan

(* state is armed - wait for time-out or both inputs
020 if not tm3 goto 22        (* if timed out
    vi110 = 40                (*  state is relax
    goto 025                  (*  return to scan
022 if not (di1 and di2) goto 025         (* if both inputs are true
    vi110 = 30                (*  state is active
025  return                   (*  return to scan
(* state is active - wait for either input relaxed
030 if di1 and di2 goto 035      (* if either input is false
    vi110 = 40                   (*  state is relax
035  return                      (*  return to scan

(* state is relax - wait for both inputs relaxed
040 if (di1 or di2) goto 045     (* if both inputs false
    vi110 = 10                   (*  state is idle
045  return                      (*  return to scan

(* End of anti_tiedown
```

Index