MAC-300

MULTI-AXIS MOTION CONTROLLER

USER'S MANUAL

email SANDEEP@nyden.com

NYDEN CORPORATION
A Subsidiary of Mycom Inc.
Rev. 3.3

Table of Contents

Chapter 1

Welcome to Mycom's MAC-300 Controller ! This Controller is a modular Multi-Axis Controller used to control Stepping Motors and Digital Servo Motors. As you will see throughout this manual the many features of the MAC-300 Controller allow it to be programmed and used as a Stand Alone Controller or as a Smart Peripheral for a Computer system that controls complex machinery.

This section covers the following topics:

About This Manual

Main Features

Specifications

Enclosure Dimensions

Nyden Product Support Services

Updates

1.1 About This Manual

This manual is your guide to utilizing the full capabilities of the MAC-300 Motion Controller. It is organized into four main sections:

- Getting Started (Chapter 2)
- Hardware Configuration (Chapter 3, 4, 15)
- Software (Chapter 5)
- Programming (Chapter 6 - 12)
- Software Development (Chapter 13 & 14)
- Wiring and Special Configurations (Chapter 15 - Appendix A)

You will find many helpful examples throughout this manual to help you quickly understand the capabilities of the MAC-300. Programming examples in Appendix B can be used to give a broader understanding of the capabilities of this Controller.

1.2 Main Features

The MAC-300 Controller uses a modular configuration to provide maximum flexibility. The basic configuration includes 1 Axis of Motion Control and 8 General Purpose Inputs and 8 General Purpose Outputs (I/O) as well as Special Inputs and Outputs to be used with Joysticks, Analog Inputs, Program selection Inputs and Start & Stop Inputs. Additional Motion Control cards and Input/Out cards with 8 Inputs and 8 Outputs are also available. These Motion and I/O cards can be added in any combination to occupy the three expansion channels. Therefore, the MAC-300 Controller is expandable up to four axis of Motion Control or up to 32 I/O or any combination of I/O or Motion modules.

The block diagram below, **Figure 1.1**, shows the internal architecture of the MAC-300.

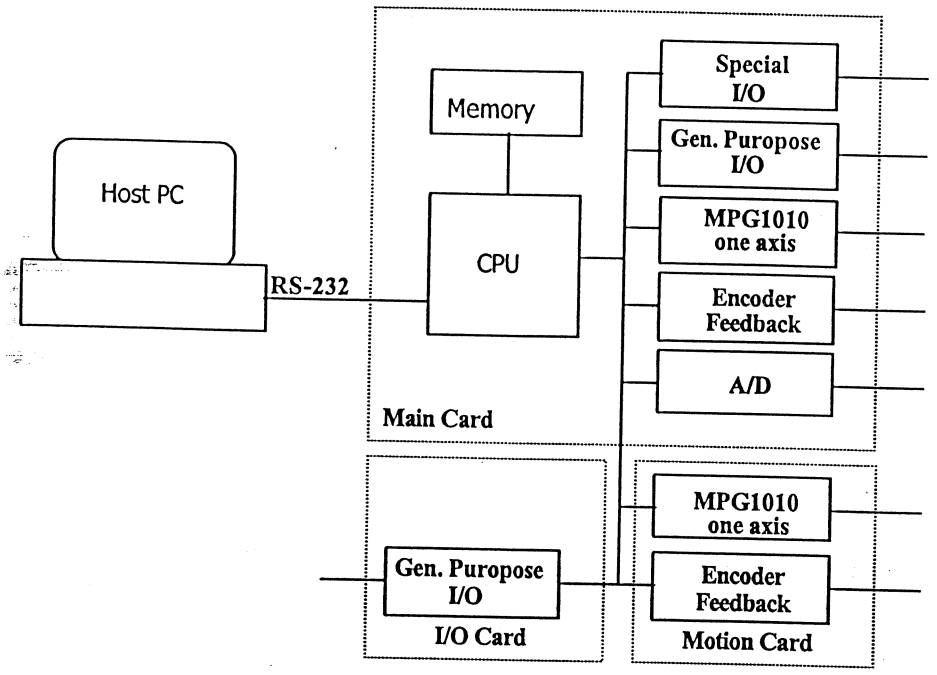


Figure 1.1 Internal Architecture of the MAC-300

The MAC-300 utilizes one of Mycom's MPG1010 pulse generator IC's for each Axis of Motion. The MPG1010 handles all the complex calculations for pulse train generation which gives the MAC-300's main CPU more time to deal with other processing tasks such as I/O control and interrupt handling.

1.3 Accessories included with the Controller

The following items should be included with your MAC-300 Controller:

MYEDIT interface software(DOS Version).

RS232 cable, Communication Cable

Power connector (4-pin)

25-pin connector(s)

50-pin connector (with or without cable)

1.4 Specifications

Number of axis	1 Motion Axis expandable to 4 Axes of Motion
Number of I/O	8 Inputs & 8 outputs expandable to 32 Inputs & 32 Outputs
Position range	± 2,147,483,647 (±2e31-1)
Speed range	Definable in eight ranges, maximum 500 k pps
Acceleration Profiles/Acceleration Range	Trapezoidal or triangular, 10-200,000 pps/s
Scaling Factor	1 to 65,535 steps/unit
Memory	EEPROM or Nonvolatile SRAM, 32 K
Storage Registers	64 with one byte, 32 with two bytes,and 32 with four bytes
Position Data Registers	64 Stored Position/Data points per Axis
Program Capacity	64 Programs, ~ 5000 lines total capacity
Programming Language	BASIC-like, meaningful English language
Movements	Absolute , incremental positioning, homing, and velocity moves
Home Return Sequences	One, two or three sensor sequences
Software Position Limits	Minimum and maximum definable for each axis
Execution Command Source	Host Computer (PC) or Special Start Input signal for Stand Alone operations
Main Operational Modes: Idle	For Downloading Programs & Position Data points and Register data via RS-232, wait for execution commands.
Manual/teach	For Axis Movement Control via Manual/Teach Inputs or Software.
Run	A program is running.
Second task : On-Line function	Single Line commands can be sent & executed in idle/run mode
Toggle function	Continuous back & forth move between two points in run mode
Real time status reading	Error, Position & Encoder data, A/D, timers, Register, I/O, Sensors Status
Power requirement	+5Vdc @ 350 mA max., +24 Vdc (I/O power)

Main Card:

One axis of motion: drive interface	Bi-clock, Step and Direction, and Differential Signal provided
sensor inputs	Reverse & Forward limit sensors, Near Home , Z-phase/ Home
special inputs	Alarm, Deviation Counter Overflow, In-Position
special outputs	Motor Free, Counter Reset, 50 mA max.
encoder feedback	Differential or TTL , A phase, B phase and Z phase inputs
General purpose I/O's	8 inputs – opto isolated, 10 mA 8 outputs – opto isolated, 150 mA max.
A/D inputs	4 Channels with 8 Bit resolution , 0 - 5 V
Special purpose inputs	Prog# /Position Data #: 7 bits , Axis #: 2 bits, Teach, Store, Start, EStop
Special purpose outputs	Error & Busy, 50mA max.
Manual/teach operation inputs	Analog /Contact Joystick Inputs, Fast Input, ± Jog Inputs
Communication: Interface	RS-232C serial, 3 wire (TX, DX, GND)
Format	9600 baud, 8 data bits, no parity, 1 stop bit, ASCII characters
Multi-Axis	Daisy chain up to 16 Controllers from a single Host RS-232 port

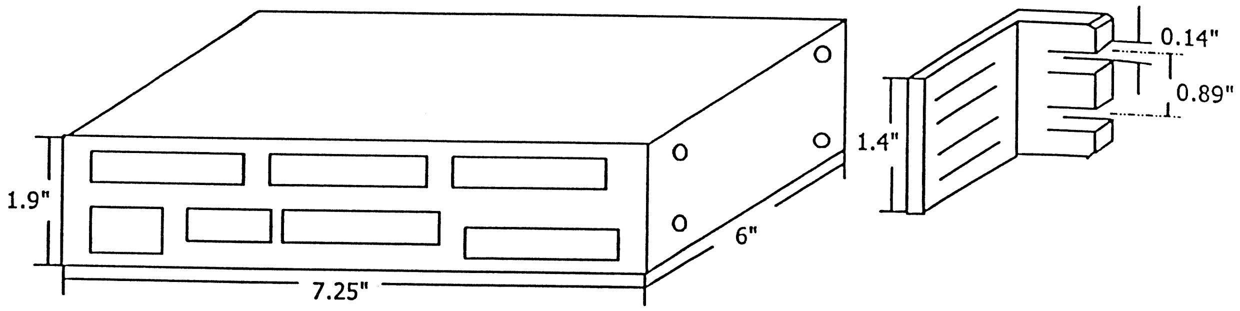
Motion Card:

One axis of motion: drive interface	Bi-clock, Step and Direction, and Differential Signal provided
sensor inputs	Reverse & Forward limit sensors, Near Home, Z-phase/ Home
special inputs	Alarm, Deviation Counter Overflow, In-Position
special outputs	Motor Free, Counter Reset, 50 mA max.
encoder feedback	Differential or TTL , A phase, B phase and Z phase inputs

I/O Card:

General purpose user I/O's	8 inputs – Opto Isolated, 10 mA 8 outputs – Opto Isolated, 500 mA max.

1.5 Enclosure Dimensions



Note: For board level dimensioning, refer to Appendix A.

1.6 Nyden Product Support Services

We want you to be happy with your choice of the MAC-300 so we fully support this Controller as well as all other Mycom products. Our application Engineers are knowledgeable and dedicated to answering any questions you may have in getting your system up and running as soon as possible. In addition, our R&D engineers are willing to work with you for special requirements or OEM applications. If questions arise, please feel free to call Nydens applications Engineers at 408-232-7700 (San Jose , California) .

1.7 Updates

Our R&D department is continuously upgrading and adding to the MAC-300's capabilities by implementing new features and developing new applications. Contact Nyden for the latest list of features.

Chapter 2

The purpose of this section is to take you through the steps required to set up one Axis of Motion on your MAC-300.

What you will need to get started:

- MAC-300 Controller and the supplied accessories
- Computer (IBM compatible PC)
- Power Supply (24 VDC and 5 VDC)
- Motor (Stepping or Digital Servo)
- Motor Driver (Stepping or Digital Servo)

Step 1. To Install and Start the Software (DOS version)

In order to install the interface software, simply copy all files from the provided disk onto your hard drive. We recommend that you create a directory (i.e. mkdir MAC-300) to place these files in. The executable file is named "MYEDIT.EXE", therefore type **MYEDIT** at your working directory to start this program.

Running MYEDIT will put you in the main window. The Function keys *F1* through *F4* activate the pull-down menus. Within these menus you can move up and down with the arrow keys. Pressing the ESC key deactivates or cancels the previous pull down command. Sections 5.3 - 5.7 explain each of the software functions in greater detail.

Step 2. Applying Power to the Controller

The MAC-300 requires power inputs of 24 VDC for the Inputs & Outputs and 5 VDC for the Controllers control circuitry. Power is applied by connecting an external power supply to the 4 pin power connector as shown in Figure 3.2. The pin assignments for this connector are listed in Table 3.1.

Two green LED's on the face of the Controller will illuminate when both power inputs are connected properly. The yellow (Busy) LED should momentarily illuminate as power is applied and the Controller automatically runs a self diagnostic test.

Step 3. Connecting the PC to the Controller

Using Com.Port #1 from your PC connect the 9 pin RS-232 cable provided to the MAC300 Controller. For longer cabling requirements see Section 3.3.

Step 4. Checking the Communication Link

To check if the software is communicating correctly with to the Controller, you will need to check the communication link. This is done by first assigning a Controller number to the Controller.

- Press *F3* - this will activate the **Commands** pull-down menu

- Select ***Assign Ctr_num*** with the arrow keys and press Enter or Return. A window asking which Controller number you wish to use will appear.

- Enter **0** and press Return.

  Note: This Controller # will used as the Calling Command in later examples.

  If the pop-up window disappears quickly, the link is good.

  If nothing occurs for a long period of time, the link is faulty. In this case, check that:

  1. the RS232 cable and its connections are correct
  2. the Communication Port is set correctly (Sec. 5.7) and no other device uses this port (i.e. Mouse Drivers)

Step 5. To Down Load a System Configuration File

The System Configuration File defines the hardware structure of the Controller and all the necessary Default Motion Parameters (Sec. 6.2). A System Configuration File must be down loaded to a new Controller or when the System Configuration Parameters need to be changed (i.e. when hardware changes are made to the system). A sample System Configuration File named **defaulta.sys** that configures Axis A **only,** is located in the software package. In order to down load this file, follow the steps below.

- If you are still under the ***Assign Ctr #*** press the ESC key on your keyboard as many times as it takes to cancel all of the windows.

- Press *F1*, select ***Load File*** and press return

- A listing of all files within the current directory will appear. Select DEFAULTA.SYS and press return. The DEFAULTA.SYS program will appear within the Editor window.

- Press *F3,* select ***Down Load File*** and press return. This operation will always write over the previously loaded System Configuration File and save these Parameters in Non-Volatile Memory.

Step 6. To Connect a Motor Driver to the Controller

The MAC-300 Controller is designed to accommodate three different types of drivers: Bi-Clock, Pulse/Step & Direction (Gate), and Differential Signal. Connect the Controller's Axis A connector to your Motor Driver as outlined in Table 3.3 and Figure 3.5 Consult your Motor & Driver Specifications for the proper wiring of these items.

Step 7. To Run a Sample Program

Now that all software and hardware requirements have been met, we are ready to run the sample program. This program will Index (increment) the motor 400 steps in one direction and then Index in the reverse direction 400 steps back to the original position. The running speed will be the default maximum speed for Axis A , which was setup in the System Configuration file.

NOTE: FOR THE ABOVE EXAMPLE, THE MOTOR SHAFT MUST NOT BE CONNECTED TO YOUR INTENDED LOAD OR DAMAGE MAY OCCUR IF THERE ARE PHYSICAL LIMITS TO YOUR SYSTEM.

Loading a File:

- Select *Load File* from the **F1** menu

- Load the **sample0.s00** file

The following is a listing of the file that will appear in the editor window.

```
@0                      ; Call Controller #0
$WR.PROG0               ; Write Program #0 to Memory
INDEX.A = 400           ; Set Index Distance to 400 pulses for axis A
MOV.INDEX.A             ; Move axis A forward for 400 pulses
MOV.INDEX.-A            ; Move axis A backward for 400 pulses
END                     ; End of the program execution.
```

- Select ***Down Load*** from the ***F3 Commands*** menu

If an Error occurs immediately, the cause may be that Program #0 (in this example) has been already loaded inside the Controller. You will have to delete the existing Program #0 inside the Controller before you can down load this new one. To do that, hit ***Esc***, and select ***Delete a Prog***. from the ***F3*** menu. When the pop-up window asks you for the Program number to delete, type **0** and hit ***Enter***. The Busy LED(Yellow) on the MAC-300 will turn on for a short time. Wait until the Busy LED turns off, then select ***Down Load*** from the ***F3*** menu again. This time you should be able to down load it.

- Select ***Run Program*** from the ***F3*** menu
- Enter **0** when the pop-up window asks you for the Program number. At this point your Motor should begin running.

Chapter 3

3.1 Controller Overview

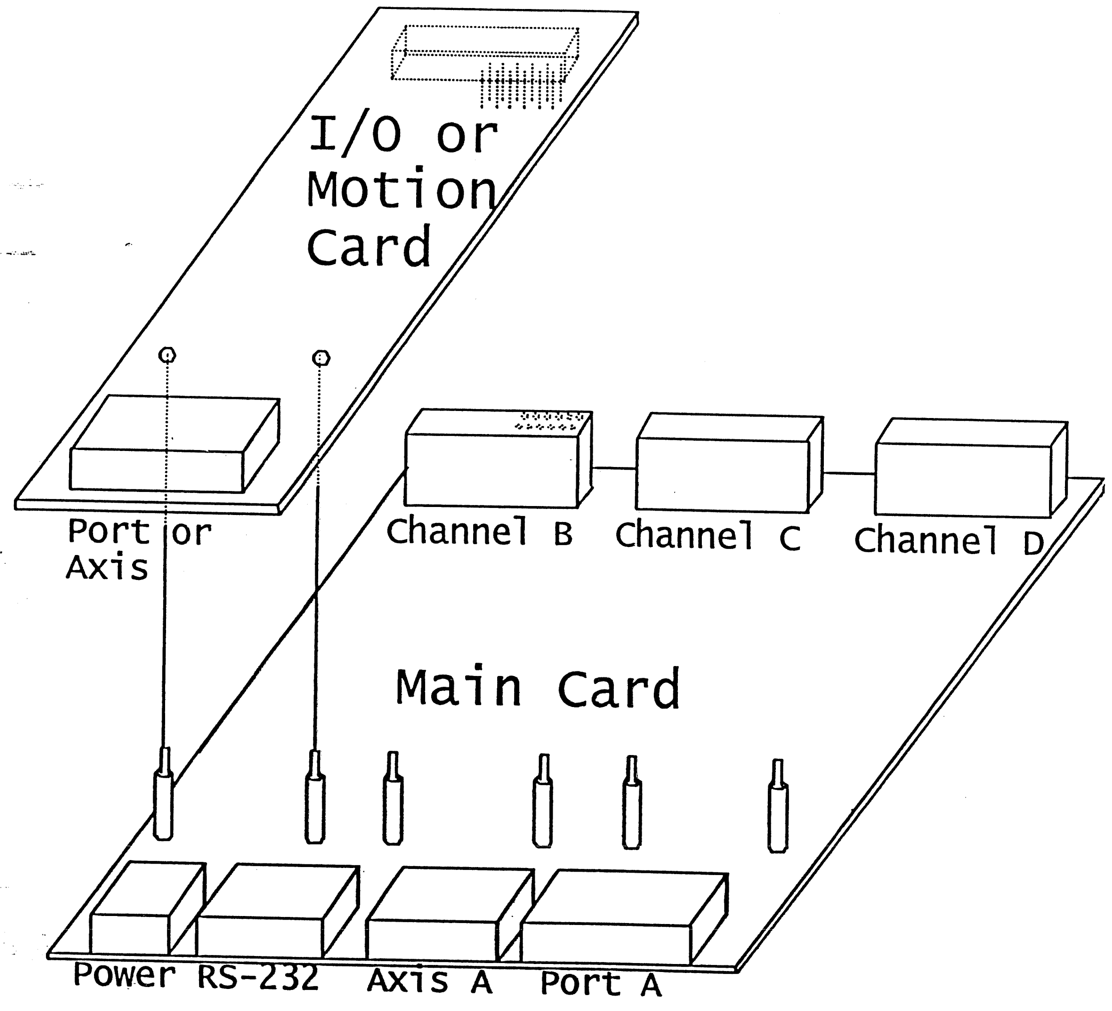


Figure 3.1 MAC-300

Figure 3.1 shows the modular configuration of the MAC-300. The modular configuration allows each of the three modular channels to be designated as I/O or Motion cards. If a Motion card(MAC-310) is added, this channel is called an **Axis**. If an I/O card(MAC-320) is added, this channel is called a **Port.**

The Main card(Mother board) of the MAC-300 includes both an Axis of Motion and 8 Inputs and 8 Outputs, these are called **Axis.A** and **Port.A**, respectively.

3.2 Connectors

The purpose of this section is to define all pin assignments for the connectors that are provided by the MAC-300's main controller card.

1. Power Input Connector

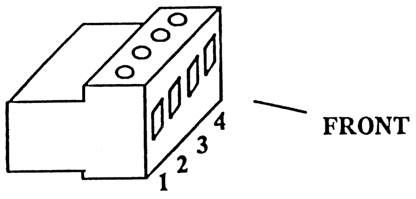


Figure 3.2 Pin Locations of Power Input Connector

Pin #	Name	Explanation
1	GND	Common of +5VDC
2	+5 VDC	5 ± 5% VDC @0.35A for Logic & Control signals.
3	IO.COM	Common of +24VDC
4	+24 VDC	+24 VDC for I/O control. Current depends on I/O devices

Table 3.1 Power Input Connector Pin Assignment

2. RS-232C Serial Communication Connector

Pin #	Name	Explanation
1	DRXD	Daisy RXD
2	TXD	Transmit
3	RXD	Receive
4	DTXD	Daisy TXD
5	GND	Ground
6-9	NC	Not used

Table 3.2 RS-232C Connector Pin Assignment

3. Motion Axis Connectors, 25 Pin DSub

	Pin #	Name	Description	See Circuit #
Driver	1	+5V	Output for +CW/CLK and +CCW/DIR Inputs of the Motor Drivers.	
	25	-CW/CLK	Pulse(Step) / CW Output signal	Output Circuit 4
	13	-CCW/DIR	Direction / CCW Output signal	"
	19	+RSCW	Phase Differential Pulse Output	"
	20	-RSCW	"	"
	3	+RSCCW	"	"
	4	-RSCCW	"	"
Inputs	18	NEAR.HOME	Near Home Sensor Input	Input Circuit 1
	6	ALARM	Alarm Input from Motor driver	"
	7	FOR.LIMIT	Forward Position Limit Sensor Input	"
	8	REV.LIMIT	Reverse Position Limit Sensor Input	"
Outputs	12	-CO	Motor Free(Current OFF) Output	Output Circuit 1
	24	-CR	Counter Reset Output	"
	2	O.COM	Output Common for CO and CR	"
Encoder Inputs	9	A	Encoder Feedback Input	Input Circuit 3
	21	$\overline{A}$	"	"
	10	B	"	"
	22	$\overline{B}$	"	"
	11	Z (HOME)	"	"
	23	$\overline{Z}$ (HOME)	"	"
Power Output	14	+24V	+24VDC Output for I/0 Devices	
	16	IO.COM	Common/Return of +24V Input	
	15	GND	Common/Return of +5V Input	

Table 3.3 Motion Axis Pin Assignments

4. Port.A Connector

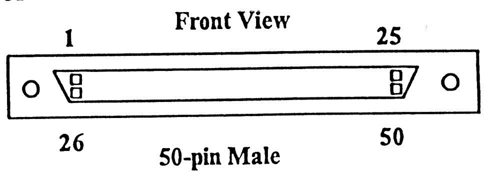


Figure 3.3 Port.A Cable Connector Pin Locations

	Pin #	Name	Description	See Circuit #
Stand Alone and Manual / Teach Operation Inputs	2	START	Start Program or Resume(Restart) Program Input See Chapter 15	Input Circuit 1
	9	ESTOP	Emergency Stop Input	"
	34	#0	Program # Selector Switch and Manual/Teach Data point #	Input Circuit 1
	6	#1	"	
	32	#2	"	
	5	#3	"	
	27	#4	BCD Inputs represent #: 0-63	
	26	#5	(refer to Ch.15 Fig15.1 for	
	1	#6	wiring details)	
	30	AXIS.0	Set Motion Axis # 0,1,2,3 (A,B,C,D)	Input Circuit 1
	4	AXIS.1	2-Bit Binary Data represents 0-3	
	28	MANUAL/ TEACH	Activate Manual/Teach mode Input	Input Circuit 1
	8	FAST	Fast/Scan Input	"
	36	+JOG	Positive Jog Single Step Input	"
	7	-JOG	Negative Jog Single Step Input	"
	3	Store/Halt	Store Data Point or Halt(Pause) Program Input . See Chapter 15	"

Table 3.4 Port A Pin Assignments (continued on next page)

Table 3.4 Port.A Pin Assignments continued

Inputs	38	IN.A0	General Purpose Inputs	Input Circuit 1
	40	IN.A1	"	"
	42	IN.A2	"	"
	23	IN.A3	"	"
	45	IN.A4	"	"
	25	IN.A5	"	"
	24	IN.A6	"	"
	44	IN.A7	"	"
Analog to Digital Inputs	22	A2D0	Analog to Digital Inputs	Input Circuit 2
	21	A2D1	"	"
	20	A2D2	"	"
	19	A2D3	"	"
Outputs	18	OUT.A0	General Purpose Outputs	Output Circuit 2
	17	OUT.A1	"	"
	16	OUT.A2	"	"
	15	OUT.A3	"	"
	14	OUT.A4	"	"
	13	OUT.A5	"	"
	12	OUT.A6	"	"
	11	OUT.A7	"	"
Reset Input	10	RESET	System Reset Input	TTL input
Special Outputs	29	ERROR	Error Signal Output	Output Circuit 1
	31	BUSY	Busy Signal Output	"
Power Output	49	+5V	+5V DC Output	
	50	GND	Common of +5V	
	39	+24V	+24V DC Output	
	41	+24V	"	
	33	IO.COM	Common of +24V	
	35	IO.COM	"	

Table 3.4 Port A Pin Assignments

5. MAC-320 I/O Modular Connector

	Pin #	Name	Description	See Circuit #
Inputs	8	IN.x0	General Purpose Inputs	Input Circuit 1
	9	IN.x1	"	"
	10	IN.x2	"	"
	11	IN.x3	"	"
	12	IN.x4	"	"
	13	IN.x5	"	"
	25	IN.x6	"	"
	24	IN.x7	"	"
Outputs	7	OUT.x0	General Purpose Outputs	Output Circuit 3
	6	OUT.x1	"	"
	5	OUT.x2	"	"
	4	OUT.x3	"	"
	3	OUT.x4	"	"
	2	OUT.x5	"	"
	1	OUT.x6	"	"
	14	OUT.x7	"	"
Power	15	+24V	+24V DC Output	
	16	+24V	"	
	17	+24V	"	
	18	+24V	"	
	19	IO.COM	Common/Return of +24V Supply	
	20	IO.COM	"	
	21	IO.COM	"	
	22	IO.COM	"	
	23	IO.COM	"	

Table 3.5 Input / Output Card (MAC-320) Pin Assignments
(x = B,C or D if an I/O is installed.)

3.3 Controller / PC Connection

To make your own cable, you must follow the wiring diagram below to link one MAC-300 controller to a host computer via the RS-232C port.

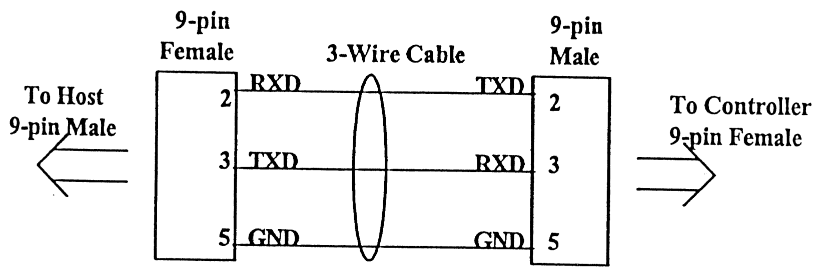


Figure 3.4 RS-232C Cable Wiring Diagram

Note: If a 25 Pin Connector is used by the Host PC then use Pin 7 of the 25 Pin Connector as the GND (Logic Ground) signal instead of Pin 5 . Pins 2 & 3 will remain the same for both a 9 pin and 25 pin serial port connection.

3.4 Motor Driver, Limit Switch and Home Sensor Connections

1. Motor Driver Wiring

The following diagram shows the Step & Direction(CLK or CW & CCW = DIR) signal connections for Stepping Motor Drivers or Digital Servo Drivers. The diagram below shows a three wire connection for Open Loop control systems that do not require any Encoder Feedback.

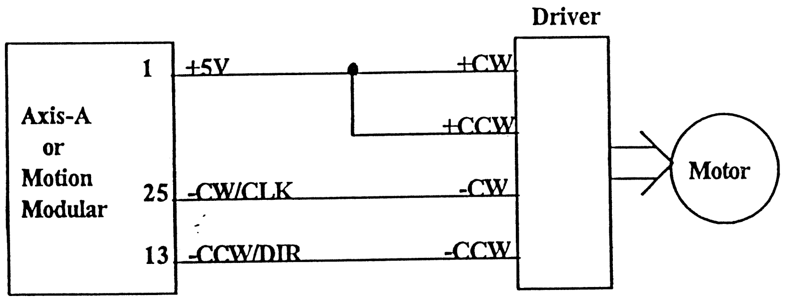


Figure 3.5 Stepper Motor Driver Wiring Diagram

The MAC-300 controller is designed to accommodate three different types of drivers: Bi-Clock, Pulse/Step and Direction (gate), and Differential Signal. **The factory default setting is Pulse and Direction.** If your Driver is different, jumper JP3 on the Main Motion Card or the modular Motion Card must be simply changed according to Appendix A.

Definitions:
Bi-Clock , is a control signal method in which Step pulses for Clockwise rotation are applied to the -CW Input while for Counter Clockwise rotation Step pulses are applied to the -CCW Input.

Gate , Step & Direction or Pulse & Direction , is a control signal method in which the Step pulses are only applied to the -CW(CLK) Input of the Driver while a High or Low signal applied to the Direction Input controls the CW or CCW rotation.

2. Forward & Reverse Limit Switches and Near Home Sensor Wiring

When a particular application requires Position Limits or a Starting Origin(Home Position), the Sensors shown in figure 3.6 can be used. All of these Sensor Inputs are Opto-Isolated. The MAC-300 Control has the capability of setting the ON/OFF control logic of the Sensor Inputs to either Normally Open or Normally Closed. **The factory default setting is Normally Open for all Sensor Inputs.** If the Sensor logic needs to be changed, refer to Appendix A. The wiring of the Limit Switches and Near Home Sensor are shown in the figure below. Maximum current for each sensor input is 10ma @ 24VDC.

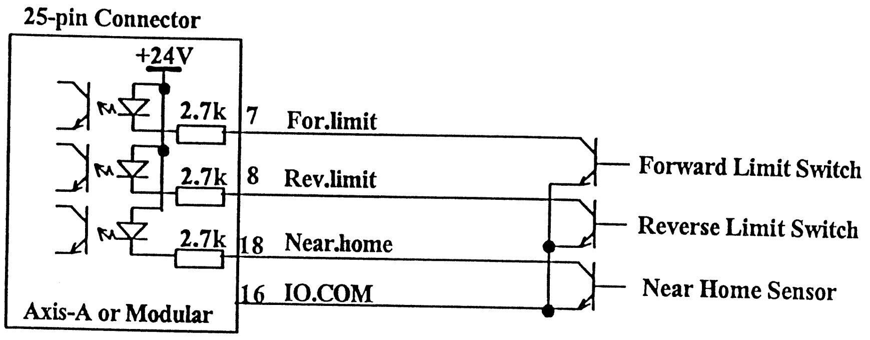


Figure 3.6 Forward & Reverse Limit Switches and Near Home Sensor Wiring Diagram

3. Motor & Driver with Encoder Feedback Wiring

When your application requires Position Compensation or Information on the exact Rotation of the Motors, Encoders can be attached to the Shafts of the Motors. With this Controller there are two types of Encoder signals that the MAC-300 will accept: TTL and Differential signals. **Differential signal is the factory default.** If your Encoder signals are TTL, please refer to Appendix A for the correct jumper settings. Figure 3.7 shows how to connect these signals to the 25 pin Motion connector.

As mentioned in section 3.4.1, the MAC-300 controller is designed to accommodate three different types of drivers: Bi-Clock, Pulse & Direction (gate), and Differential signal. **The factory default setting is Pulse and Direction.** If your Driver is different, jumper JP3 on the Main Motion Card or the modular Motion Card must be simply changed according to Appendix A.

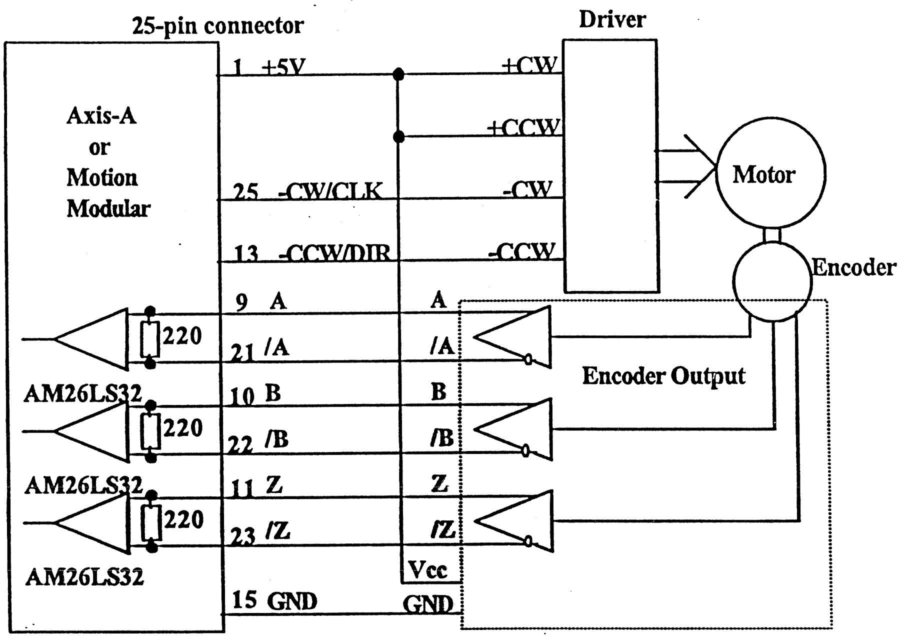


Figure 3.7 Motor & Driver with Encoder Feedback Wiring

3.5 General Purpose & Special Input / Output Wiring

The following diagrams are examples of wiring any of the General Purpose Input & Outputs, and dedicated Inputs & Outputs that this Controller provides. Refer to Tables 3.3, 3.4 and Table 3.5 when wiring these signals. With this Controller all the Digital Inputs and Outputs are Opto-Isolated. To use these Inputs follow the wiring diagrams shown below. Maximum current for each Input is 10ma @ 24VDC.

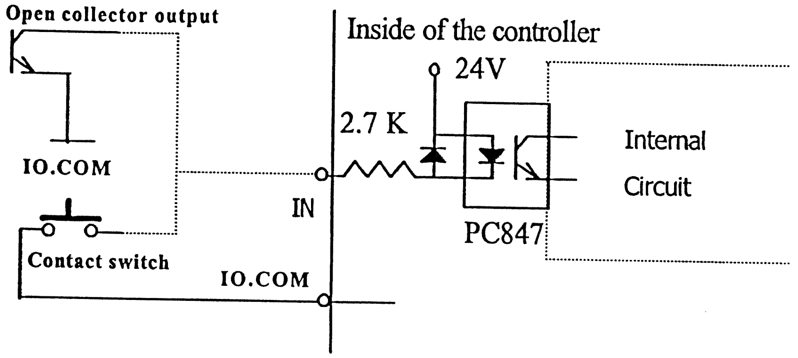


Figure 3.8 General Purpose or Special Input Wiring

Note: Since IO.Common is the common of the 24V supply and is internally connected within the MAC300, the user can connect one side of the switch to the Input pin and the other side of the switch to either the common(return) of the port connector or the common(return) of the 24V power supply.

Figure 3.9 shows a wiring example of how to drive a Relay by using one of the General Purpose Outputs.

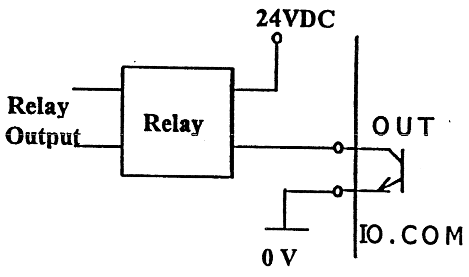


Figure 3.9 Wiring Example for General Purpose or Special Output Circuits

3.6 Circuit Diagrams

Input Circuit #1: For General Purpose & Special Inputs

IN.A0 - IN.A7, Near Home, Reverse & Forward Limits, Start Program , ESTOP etc.

Input Circuit 1 is shown in **Fig. 3.10**. It is Opto-Isolated by the Opto-Coupler PC847. The Input Current is approximately 10 mA. See **section 3.5, Fig. 3.8 for the suggested connections.** Maximum current for each Input is 10ma @ 24VDC.

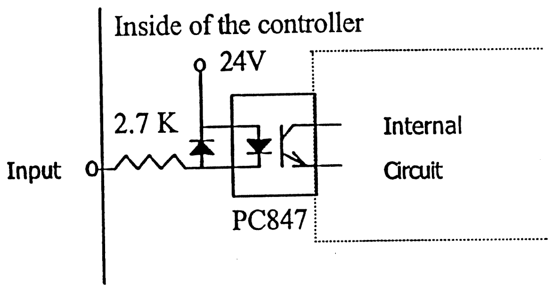


Figure 3.10 Input Circuit #1 (PC847 Equivalent)

When Proximity Sensors that produce Analog Voltage outputs or similar devices are used within your system this Controller has the capability of monitoring four of these 0 to +5 Volt signals. These Analog Inputs are also used for Analog JoySticks as outlined in Chapter 15. Figure 3.11 shows the connections required for these Inputs.

Input Circuit #2: For Analog Inputs: 0-5 VDC.

A2D0, A2D1, A2D2 and A2D3

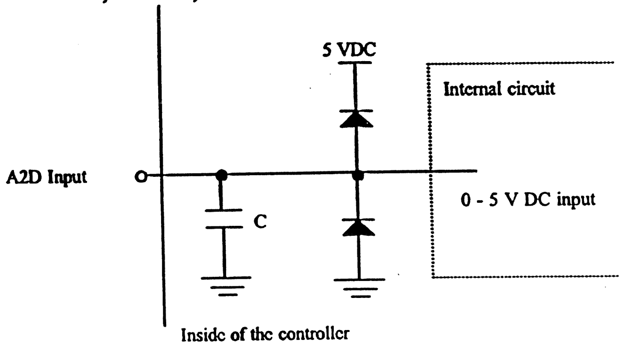


Figure 3.11 Analog Input Circuit #2

Note: The Common/Return of this A2D Signal should be the +5V Common/Return which is referred to as GND within this Manual.

Input Circuit #3: Encoder Input Signals

The following diagram is a small segment of the Encoder circuit shown in figure 3.7. Each Encoder Feedback signal is connected to an Input circuit as shown in the figure below. The Factory Default settings for the Encoders is for Differential Signal Inputs. If TTL Logic inputs are required then see Appendix A for the appropriate modifications.

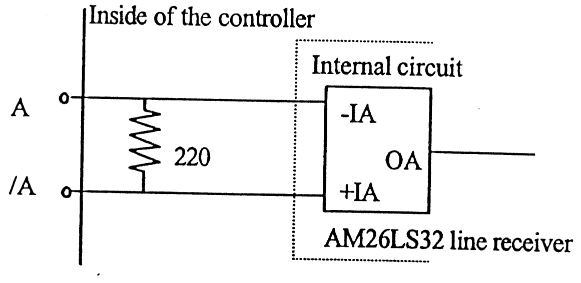


Figure 3.12 Input Circuit #3

Output Circuit #1 : For Busy, Error Signal, Motor Free and Counter Reset Outputs

When your system requires indications of when a Program is running(BUSY) or if an Error has occurred(ERROR) as well as Turn-OFF the current to the Motors(Motor Free) these Outputs as shown in Output Circuit #1 can be used. The Outputs are Opto-Isolated by the Opto-Coupler PS2501. The maximum sinking current of these Outputs is 50 mA. Therefore, if the driven load is larger than 50 mA, an external relay or other driver should be used as shown in figure 3.9.

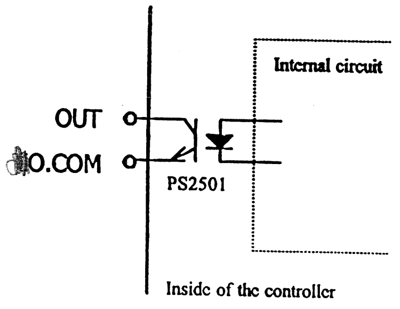


Figure 3.13 Output Circuit #1 (PS2501)

Output Circuit #2 : For Port A General Purpose Outputs , OUT.A0 - OUT.A7

When your application requires the MAC-300 to control other devices external to the Controller, the Outputs as shown in Output Circuit #2 below are available. These Outputs are Opto-Isolated by the Opto-Coupler PS2502. The maximum sinking current is for these Outputs is 150 mA. Therefore, if the driven load is bigger than 150 mA, an external relay or other driver should be used as shown in figure 3.9.

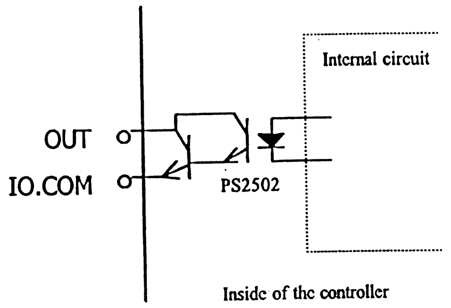


Figure 3.14 Output Circuit #2 (Darlington Transistor Drive)

Output Circuit #3 : For MAC-320(Port Module) General Purpose Outputs, OUT.x0 - OUT.x7

When your application requires more than the standard 8 Inputs & 8 Outputs that are provided by the MAC-300's main card, additional I/O modules(MAC-320) can be added to the Controller. Output Circuit #3 as shown in the figure below represent the Outputs of these I/O modules. These Outputs are Opto-Isolated by the Opto-Coupler PS2501. The NPN Darlington transistor array is used to enhance the drive current capability. The maximum sinking current for this Output is 500 mA. Therefore, if the driven load is larger than 500 mA, an external relay or other driver should be used as shown in figure 3.9.

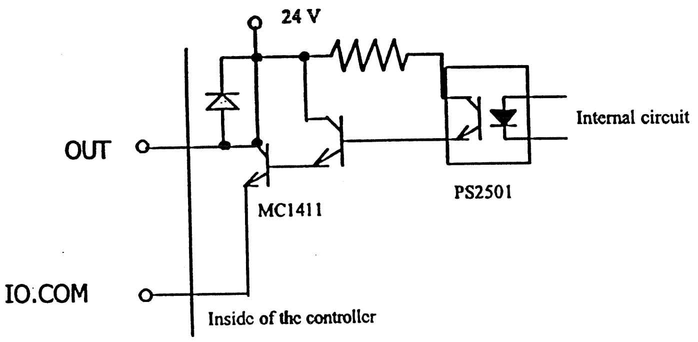


Figure 3.15 Output Circuit #3 for MAC320(Port Module) (Darlington Transistor Drive)

Output Circuit #4 : For Motion Control Signals

The main control signals that are applied to a Stepping Motor Driver or a Digital Servo Motor are shown below in Output Circuit #4. The MAC-300 Controller is designed to control Drivers that require Step & Direction, Bi-Clock(CW&CCW) or Differential Input Signals. The default setting at the factory is Step & Directions Signals. The Step & Direction as well as the Bi-Clock outputs from this Controller are TTL sinking outputs (see Section 3.4 for TTL wiring diagram). The Outputs +RSCW, -RSCW (Clockwise), +RSCCW and -RSCCW(Counter Clockwise) are Differential Signal pulse outputs.

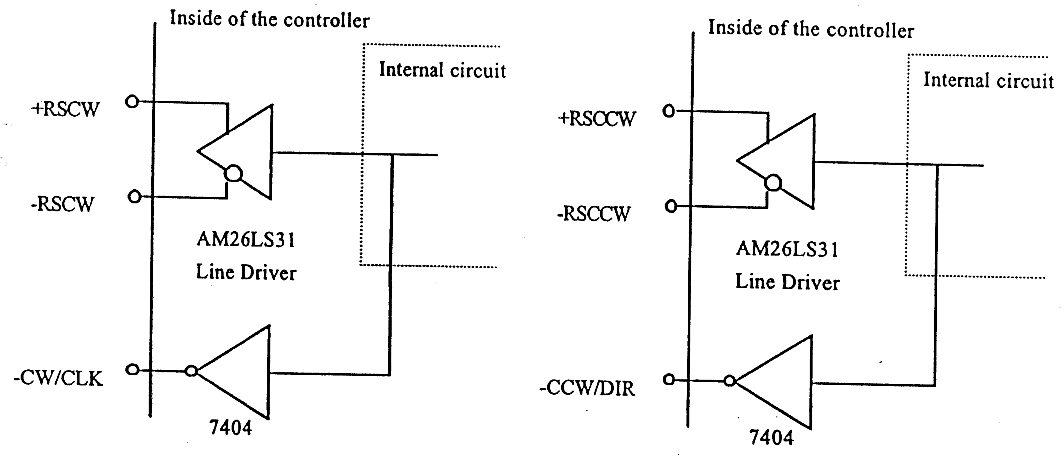


Figure 3.16 Motion Control Driver Output Circuit #4

Definitions:

Bi-Clock , is a control signal method in which Step pulses for Clockwise rotation are applied to the -CW Input while for Counter Clockwise rotation Step pulses are applied to the -CCW Input.

Gate , Step & Direction or Pulse & Direction , is a control signal method in which the Step pulses are only applied to the -CW(CLK) Input of the Driver while a High or Low signal applied to the Direction Input controls the CW or CCW rotation.

Chapter 4

The MAC-300 Controllers are designed so that Multiple Controllers can be linked to a single Host Computer(PC) by using a Serial Daisy-chain via the RS-232 port. There are two different types of Daisy-Chain connections. Figure 4.1 shows the first connection which is ideal when the number of Controllers is less than 16. In this configuration the Host PC can simultaneously call all of the Controllers. Even if one of the Controllers is out of order, such as in the case of a power failure, the command from the Host PC can be sent to all of the other Controllers.

Note: Before Multiple Controllers are set up, each Controller must be designated a Controller number (see Sec. 5.6). This *cannot* be done after they are Daisy-Chained.

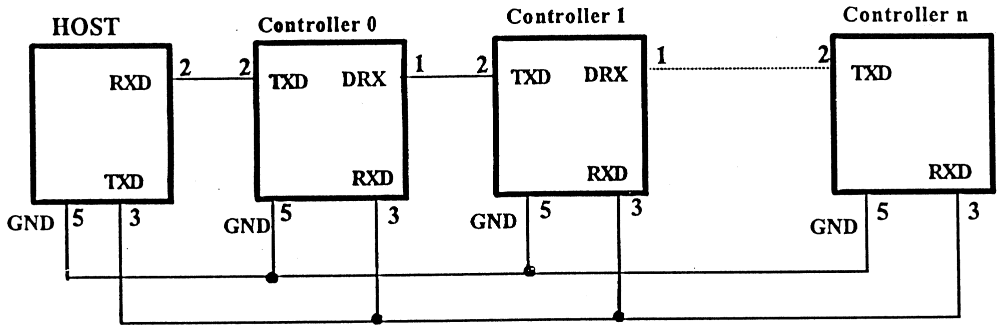


Figure 4.1 Daisy-Chain Connection A

Figure 4.2 shows the second connection which is ideal when the number of Controllers is more than 16. In this setting, the command from the Host PC will be sent to the #1 Controller first, then the #2 Controller, and so on. The delay time on each Controller is about 1 uS. Although this method allows you to connect many Controllers to the same Daisy-chain, if one of the Controllers is not functioning, the command from the Host PC will stop at the failed Controller.

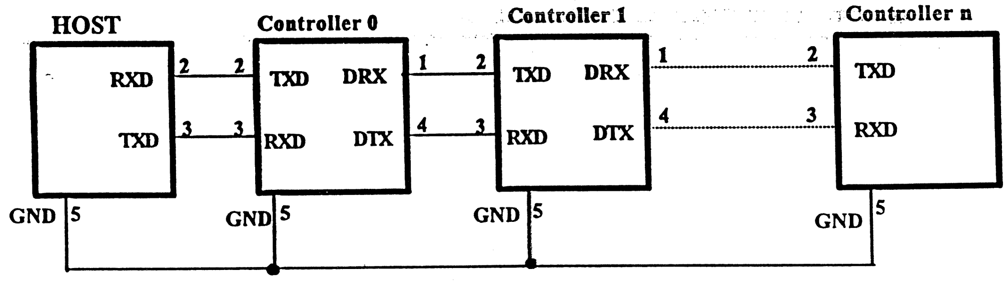


Figure 4.2 Daisy-chain Connection B

Chapter 5 MYEDIT Software (DOS Version)

The purpose of this chapter is to show you how to use the MAC-300 Editor(MYEDIT) and its Communication Link(Read & DownLoad features) to the Controller.

5.1 Installing the MYEDIT program

In order to install the Editor, simply copy all files from the provided Disk onto your Hardrive. We recommend that you create a new directory (i.e. mkdir MAC-300) to place these files in. The executable file is named "Myedit.exe". Type " **MYEDIT** " from your new working directory to start this Motion Control Program Editor.

5.2 Introduction to Software

MYEDIT is a basic Text Editor and Communication Software Package for the MAC-300 Controller. At the top of the MYEDIT screen you will see six 'F" keys(Function Keys) that activate pull-down menus which allow you select many of the MYEDIT options.

Editor Function Keys

Function keys *F1* thru *F4* activate the pull-down menus. Within each of these menus, you can select one of the many options by using the up and down the arrow keys until your desired option is highlighted. Once the option is highlighted press the 'ENTER" key to start that operation. Pressing the "ESC" key cancels or closes the pull down menu window.

Function keys *F6* Slow Stop(with deceleration) and *F7* Emergency Stop will Stop all Axes of Motion respectively and any Program of the active Controller. If the Controller is executing a Home Return command, the Motors will not Slow Stop until the Home Return command is finished. Only the ESTOP input and the RESET input can Emergency Stop Motion and End the Program immediately under any situation.

5.3 Editor Window and Commands

To create Motion Control Program, System Configuration, Position Data point, and Variable Storage Register files, command lines are entered at the left of MYEDIT Editor window while comments are entered to the right of the window. The comments area should start with a semicolon (;) and can be placed to the right of the functional commands or at the beginning of lines used only as comment lines. Each line can be up to 70 characters. Blank lines are also allowed in any of the files. When you finish a command line, you must press ENTER to move the cursor to the next line. Line Numbers are automatically added at the left of each line. The Line Number of the current Line and the total number of lines for the whole file are displayed on the bottom of the screen window.

The following is a list of control keys you can use with MYEDIT Editor:

BACKSPACE	Deletes the Character to the left of the cursor.
DEL	Deletes the Character at the cursor.
INS	Inserts a Line.
CTRL_Y	Deletes a Line.
Arrow keys	Move the cursor one character or one line.
HOME	Moves the cursor to the beginning of the line.
END	Moves the cursor to the end of the line.
TAB	Moves the cursor five spaces to the left.
PAGE UP	Scrolls up one screen.
PAGE DOWN	Scrolls down one screen.

5.4 *F1* - File menu

- **New**
  Opens a new file for editing and clears the screen of any existing file.

- **Load File**
  Loads an existing file from the your MAC-300 directory into the Editor. A listing of all programs with the extension * . S ** (Ex. Linear.sys) within the current directory will be shown.

- **Insert File**
  This command will insert a file into the existing file that is in the Editor window at the insertion point (the point of the cursor).

- **Save**
  This command will save the Current file in the Editor window into the current working directory.

- **Save As**
  This command will save the current file under a user defined name (it must be in the format of *.S**). It is recommended that System Configuration file be saved as *.SYS, while Motion Control Programs should be saved as *.S## (##: 00-63 indicates Program Number). For example, **defaulta.sys** is the System Configuration file; **sample.s00** is the Motion Control Program #0; **robot.s04** can be used as the name for the Motion Control Program #4.

- Print

  This command prints the current file to the default printer.

- Exit

  Exits MYEDIT ***without*** saving the current document.

5.5 *F2* - Read menu

These functions are used to observe the operation and status of the system. The values displayed are automatically updated at one second intervals.

- **Error Code**
  Checks the current Error Code (refer to Chapter 13). When the system is OK, the Error Code will indicate a zero.

- **Current Pos.**
  With this command, you can check the Current Position of any Axis. The Current Position is defined as the number of Output Pulses sent from the Controller. In order to calculate the value in the units times the Scaling Factor used (Section 6.3), you must divide the number of pulses by the Scaling Factor.

- **Encoder Data**
  With this command, you can read the exact number of Pulses produced by the Encoder. If there is no Position Errors then this value should be the same as the Current Position.

- **Scaling Factor**
  This command returns the value of Scaling factor (Sec. 6.3) for a given Axis.

- **Input Status**
  This command returns an 8-Bit Binary Number. Each Bit refers to the status of one Input (0 = Closed/Active, 1 = Open/Not Active). The order from left to right refers to IN.x7, IN.x6, ... , IN.x0. x = the A,B,C or D Ports

- **Output Status**
  This command returns an 8-Bit Binary Number. Each Bit refers to the status of one Output (0 = Closed/Active, 1 = Open/Not Active). The order from left to right refers to OUT.x7, OUT.x6, ... , OUT.x0 x = the A,B,C or D Ports

- **Sensor Status**
  This command returns an 8-Bit Binary Number. Each Bit refers to the status of one of the Motion Axis's Sensor Inputs (1 = Not Active, 0 = Active). The order from left to right is Bit 7, Bit 6, ... , Bit 0.

Bit #	Function
7	Not used
6	Not used
5	Near Home Sensor Input
4	Not used
3	Reverse limit switch Input
2	Forward Limit switch Input
1	Alarm Input
0	Emergency Stop Input

Note: The Status of the Near Home Sensor Input is Only available in the Idle mode.

- **A/D Input**
  This command returns an Integer value from 0 to 255. This value corresponds to the Analog Voltage Input of 0-5V for one of the four A/D Inputs.

- **Timer Value**
  This command returns the value of one of the eight Software Timers in units of 10 milliseconds. Example: When Timer0 = 100 this is equivalent to 1 second.

- **Short Register**
  This command returns the value of one of the Short Registers (SREG#, #:0~63), which is an Integer value from 0 to 255.

- **Norm. Register**
  This command returns the value of one of the Normal Register (REG#, #:0~31), which is a numerical Integer value from 0 to 65535.

- **Long Register**
  This command returns the value of one of the Long Registers (LREG#, #:0~31), which is a numerical Integer value from -2,147,483,647 to +2,147,483,647.

- **Position Data**

  This Information Uploading operation is only available with the Windows 3.1 to 95 software.

- **Register Table**

  This Information Uploading operation is only available with the Windows 3.1 to 95 software.

5.6 *F3* - Commands menu

This section describes the operational Communication Link between the Editor and the Controller.

- **Down Load File**
  This command Down Loads the Current Editor file into the Controller. If there is already a Program with the same Program Number inside the Controller, the Controller will not allow you to Down Load this file. You can Delete the Program that already exists inside the Controller or you can Assign a New Number to this Program, then you can Down Load this Program. If there is a Line in the Program that the Controller can not understand, the Controller will not allow the Program to be Down Loaded either and an Error will occur. Therefore you must review the proper command syntax for the Controller.

- **Run Program**
  When this command is selected, the Controller will ask for a Program Number (0-63). After receiving a Program Number, the Controller will execute that Program.

- **Delete a Program**
  This command will Delete a specified Program from within the Controller. If there is a large number of Programs loaded in the Controller, this can take some time.

- **Delete All**
  This command Deletes all Programs in the MAC-300 Controller. This command will not delete the System Configuration File.

- **Error Reset**
  This command Resets the Errors that may have stopped the system (refer to Ch. 13 for detailed Error Code information).

- **Assign Ctr_num**
  When Multiple Controllers are used, it is necessary to Assign a Number to each Controller. Controller Numbers **must be Assigned Individually** before they are Daisy-Chained. The Host Computer(PC) must be connected to the Controller which is being Assigned a Controller Number. **The default Controller Number is zero, 0.**

- **Step-by-Step**
  In this mode, you can execute a Program Line by Line simply using the down arrow key. The PC will down load Command Lines in the Editor window one by one starting from Line number 000. Blank lines, comment lines and communication protocol lines (lines beginning with a $ symbol, i.e $WR.PROG0) will be skipped. Every time you press the down arrow key, the next program line will be sent to the MAC-300 and executed. When the END line is found or ESC key is pressed, the Step-by-Step execution is terminated. This function can be used as a Debugging Tool. All commands except for flow control commands (L#, GOTO, CALL, RET and IF...THEN..., refer to Sec. 9.1) can be used in this mode.

- **On-line Mode**
  In this mode you can Type in a Command Line and the Controller will execute that line. All commands except for Program Flow Commands (L#, GOTO, CALL, RET and IF...THEN..., refer to Sec. 9.1) can be used for this mode.

- **Enable Joystick**
  When the Controller is in Idle-mode, this command can Turn-On the Controller's Manual/Teach mode. In Manual/Teach mode, you are able to use the Joysticks to control one or more Axes of Motion and Store Position Data points if required.

- **Disable Joystick**
  This command deactivates the Manual/Teach mode if the hardware MANUAL/TEACH Input is not active.

5.7 *F4* - System menu

- **Controller #**
  In order for the Software to communicate with the correct Controller, the Controller Number listed at the bottom of the Editor screen must match the intended Controller. The Controller Number must be set correctly in order to use the Read ***(F2)*** and Commands ***(F3)*** functions. The only exceptions are the Down Load and the Step by Step commands, where the first line of any program (which designates the Controller, Sec. 6.1) overrides the current Controller number displayed in the MYEDIT main window.

Chapter 6

6.1 Introduction to Programming

Motion Control Programs, System Configuration, Position Data Point , Variable Storage Register files and the On-Line mode all must start and end with the following Communication Lines:

@(0, 1, 2, ...)	; Tells which Controller will Receive the Information.
$(Header Line)	; Specifies which Type of File to be down loaded to Memory.
...	;
...	; File contents
...	; File contents
...	
End	; End of file

The second line of any file is the Communication Protocol (Header Line) which specifies the type of file you are down loading to the Controllers memory. These four headers are:

$WR.SYS	; Write a System Configuration File to Memory (Ch. 6)
$WR.PROG(0-63)	; Write a Motion Program(Prog. Num., 0-63) (Ch.7 - 12)
$WR.DATA	; Write a Position Data point file to Memory(Ch.8)
$WR.VAR	; Write a Variable/Storage Register file to Memory (Ch. 8)
$ON.LINE	; On-Line Mode -- also used for Step-by-Step operation. (Ch. 5)

With the MAC-300 Controller you will typically write the System Configuration file that defines the Default Motion & Control Parameters and you will write Motion Control Programs that define your machine movements. For advanced programmers Position Data Point and Variable Storage Registers files are available so you can write general purpose Programs that use this Position & Register data. With this feature your main Programs would always remain the same while the Position Data point and Variable Storage Register files can be changed. When Position Data point and Variable Storage Register files are Down Loaded to the Controller this information is saved in Non-Volatile Memory and can only be accomplished in the Idle mode where a Program is not running and the Manual/Teach Mode is not active. The On-Line Mode can be used to change the values of the Storage Registers when a Program is running however, these values will only be retained until they are changed again or as long as your Programs are running, they will not be saved to Non-Volatile Memory in the On-Line Mode. The Manual/Teach Mode can be used to save Position Data points one at a time to Non-Volatile Memory which is a good way of Teaching the Controller your important machine positions however this can be a long process. Therefore, once your important Position Data points have been defined you may wish to include them in your Position Data file.

6.2 Introduction to the System Configuration file

The System Configuration defines the hardware structure of the Controller and all the necessary Default Motion and Control Parameters. A System Configuration file must be down loaded for a new Controller or when the System Configuration Parameters need to be changed. A sample file named **defaulta.sys** is included with this Software package. This file is for one Axis of Motion (Axis A). Below is a listing of the file which can easily be modified to meet your specifications.

```
@0                  ; Call Controller 0 (Section 6.1)
$WR.SYS             ; Denotes System Configuration File (Section 6.1)
CH.B = N            ; Set Channel B to "Not Used" (Section 6.4)
CH.C = N            ; Set Channel C to "Not Used" (Section 6.4)
CH.D = N            ; Set Channel D to "Not Used" (Section 6.4)
```

Note: The following 5 System definitions may not be required for all applications.

```
JOY = 0             ; Set the Joystick Type to Analog Input (Section 6.7)
AJOY = A            ; Define the available Analog Joysticks (Section 6.7)
AWINDOW = 0         ; Define the Analog Joysticks Voltage Window ( Section 6.7 )
IN.JOG = 0          ; Set Port A Inputs as Not Used for Jogging ( Section 6.7 )
ENA.HL = 0          ; Disable Hardware Limit Error Indications ( Section 6.9 )
```

Note1: The Scaling Factor must be set before all Default Parameters and Software Position Limits.
Note2: If the MAC-300 Controller you are using has more than 1 Axis of Motion then duplicate the following information and change the reference for Axis A to the Motion Axis you are using.

```
SCA.FAC.A=1         ; Set the Scaling Factor of Axis A to 1 (Section 6.3)
ENC.ENB.A=0         ; Set the Encoder Feedback "OFF" (Section 6.5)
TRA.AUT.A=0         ; Set Auto-Tracking/Auto Correction to "OFF" (Section 6.5)
HOM.DIR.A=0         ; Set the starting Homing Direction to Reverse (Section 6.6)
ORI.SEQ.A=1         ; Set the Home Return Sequence to Mode 1 (Section 6.6)
VEL.RNG.A=8         ; Set the Velocity Range to Range 8 (Section 6.8)
DEF.MAX.A=3000      ; Set Default Maximum Velocity to 3000 units/sec (Section 6.9)
DEF.MIN.A=1000      ; Set Default Minimum Velocity to 1000 units/sec (Section 6.9)
DEF.ACC.A=2000      ; Set Default Acceleration Rate to 2000 units/sec² (Section 6.9)
DEF.DEC.A=2000      ; Set Default Deceleration Rate to 2000 units/sec² (Section 6.9)
LIM.FOR.A=N         ;Set Software Forward Position Limit to "Not Used" (Section 6.10)
LIM.REV.A=N         ; Set Software Reverse Position Limit to "Not Used" (Section 6.10)
END                 ; End of file (Section 6.1), go back to the Idle Mode.
```

Note: The Scaling Factor must be set before all Default Parameters and Software Position Limits. When the Scaling Factor = 1 then all of the Units are referred to as Pulses or Steps.

6.3 Setting the Scaling Factor for each Axis

- **SCA.FAC.(Axis)= integer value (0-65,000)**

For any Mechanical System connected to a Motor, there is a relationship between the number of Pulses (Steps) the Motor takes and the Distance Unit the Mechanical System has moved(i.e. lead screw). This relationship is called the Scaling Factor (Steps per Distance Unit). The Unit that is used (i.e. Inch, mm, Pulse, Angular Unit, etc.) is defined by the User.

Example:
If the Distance Unit desired for Axis A is millimeters(mm) and a Stepping Motor with 2000 steps/rev is used along with a Lead Screw that has a Lead of 5 mm/rev, the Scaling Factor used for Axis can be calculated as follows:

2000 (steps/rev) ÷ 5 (mm/rev) = 400 (steps/mm)

Therefore, you can set the Scaling Factor for Axis A as: **SCA.FAC.A = 400**

Advantages to using a Scaling Factor:
By setting the Scaling Factor, all Distances, Velocities, Accelerations and Deceleration's, can be entered as convenient values instead of numbers of Pulses or Steps. For example, in the above application you could program an Index Move to be 32.25 mm at 5.5 mm/sec with an Acceleration and Deceleration rate of 12 mm/sec^2 by using the numbers 32.25, 5.5, and 12 respectively in the program. If the Scaling Factor were set to 1, all Distances, Velocities, Accelerations and Deceleration's would have to be entered in terms of Pulses or Steps (12900, 2200, and 4800).

Note 1. The Controller will only accept numbers with up to 4 Decimal Point Accuracy (i.e. 23.0001). Therefore, if the Scaling Factor is too large, the accuracy of the system will not be maximized. In order to minimize error, the Scaling Factor should be made less than 10000. This will keep the error to plus or minus one Pulse.

Note 2. Table 6.1 shows which Parameters are affected by the Scaling Factor.

Affected Parameter	Reference Section	Affected Parameter	Reference Section
DEF.MAX DEF.MIN DEF.ACC DEF.DEC	Section 6.7	FIN.POS INC.DIS INDEX.	Section 7.2
ACC.RATE DEC.RATE MIN.VEL RUN.VEL	Section 7.1	DATA(Axis)(#)	Section 8.2

Table 6.1 Parameters Affected by the Scaling Factor

6.4 Modular Channel Configurations

To Define the Modular Channel, Motion Axis, I/O Port or Not Used
Since the MAC-300 Controller is a modular Controller that can be configured as a 1 to 4 Axis Motion Controller or Controller with addition I/O modules, each additional Channel (B-D) must be designated with one of the following in the System Configuration file:

- **CH.(Axis or I/O Port name) = M , I , or N**
  Axis or I/O Port name : B, C, or D

This command designates each of the 3 Expansion ports (B-D) as :

M = Motion Axis
I = I/O Port
N = Not used

Examples:

CH.B=I	; Channel B is a General Purpose I/O Port
CH.C=M	; Channel C is a Motion Axis
CH.D=N	; Channel D is Not Used

6.5 Defining Encoders parameters

- **ENC.FAC.(Axis) = 1, 2, 4, 8, 1/2, 1/4 or 1/8**

When your application requires Encoder Feedback ,this command defines the Scaling Factor for the Encoder Feedback data. The Encoder Feedback data is equal to the Encoder Scaling Factor times the number of Encoder Feedback counts. With this Controller the number of Encoder Feedback counts are equal to Four Times the Number of Lines on the Optical Encoder. For example, when an Optical Encoder is directly mounted on the Shaft of a Stepping Motor that has 1000 Steps/Revolution, to change the Encoder Feedback data to 1000 Counts/Revolution when the Encoder has 250 Lines per Revolution, the Encoder Scaling Factor should be 1. If the Encoder has 125 lines/rev and is attached to the same Motor, the Encoder Scaling Factor should be set to 2. If the Encoder has 500 lines/rev , then the Encoder Scaling Factor should set to 1/2.

Examples:

ENC.FAC.A=1	; Encoder Feedback data is equal to Feedback counts
ENC.FAC.C=2	; Encoder Feedback data is equal to 2 times Feedback counts
ENC.FAC.D=1/4	; Encoder Feedback data is equal to 1/4 times Feedback counts

To Enable or Disable Encoder Feedback

When an Encoder is attached to any of your Motors, you have the option of Enabling or Disabling this feature in the System Configuration file.

- **ENC.ENB.(Axis) = 0 or 1**

  0: Disables the Encoder Feedback function for the specified Axis

  1: Enables the Encoder Feedback function for the specified Axis

Examples:

ENC.ENB.A=1	; Enable the Encoder Feedback function for Axis A
ENC.ENB.C=0	; Disable the Encoder Feedback function for Axis C

To Enable or Disable Auto-Tracking/Auto Correction Function

If Auto Tracking or Auto Correction features are required in order to compensate for Position inaccuracies, the following command can be used to bring the Motor Position to within your defined limits(see LIM.ERR).

- **TRA.AUT.(Axis) = 0 or 1**

  0: Disables the Auto-Tracking/Auto Correction function for the specified Axis

  1: Enables the Auto-Tracking/Auto Correction function for the specified Axis

Examples:

TRA.AUT.A=1	; Enable the Auto-Tracking function for Axis A
TRA.AUT.C=0	; Disable the Auto-Tracking function for Axis C.

If the Encoder Feedback function is Enabled, the Controller will continuously track the Encoder signal. The Controller will not compensate for any Position Error after the Motor has stopped unless the Auto-Tracking function is Enabled. For example, if an external interference causes a Position Error and the Auto-Tracking function is Enabled, the Controller will automatically attempt to move the Motor in order to correct the Position Error. The Controller will try a given number of times to do the compensation (See In-Position Error Limit Setting). If the Controller still cannot correct the Position Error, the ERROR LED(RED LED) will turn on and the Controller will stop operation until the Error and the Controller is Reset/Cleared.

Setting the In-Position Error Limit for Auto Correction

- **LIM.ERR.(Axis) = value (pulse)**

This command defines the Allowable Position Error value for each Axis. The Controller records the Output Pulse number and the Encoder Feedback Pulse number. If the difference between the Output Pulse number and the Encoder Feedback Pulse number is greater than the In-Position Error Limit value, the Controller will try to compensate for this Error by sending more pulses accordingly. The In-Position Error Limit value can be 1 to 127 in units of Pulses.

Example: Set the In-Position Error Limit for Axis C to be 5 Pulses or Steps:

```
LIM.ERR.C= 5
```

Setting the Number of Repeat Times for Error Compensation

- **REP.TIM.(Axis) = value**

This command defines the number of times the Controller will try to compensate for a Position Error. If compensation is not successful in the specified number of Repeat Times, the Controller will stop sending Pulses. The ERROR LED(RED LED) will also turn on and an ERROR signal will be generated at the ERROR Output pin (Ch. 13). The Controller will wait for an Error Reset command from the Host Computer(PC) or an External Reset input. The compensation Repeat Time limit can be from 1 to 127 times.

Example: Set the compensation Repeat Time limit for Axis C to 10 times:

```
REP.TIM.C = 10
```

6.6 Setting the Home Return Configuration

If your mechanical system requires Home or Origin Sensors that define the starting zero position of your movements, this Controller allows you to define the Direction to start the Home search process and allows you to choose among the three different Home Return Sequences.

Home Return Direction

The following command in the System Configuration file defines the starting Direction for the Home search process.

- **HOM.DIR.(Axis) = 0 or 1**

  0 = Start the Home search process in the Reverse direction.

  1 = Start the Home search process in the Forward direction.

Home Return Sequences , Mode 0, Mode 1, Mode 2, or Mode 3

- **ORI.SEQ.(Axis) = 0 , 1 , 2 or 3**

One of the following four Home search modes can be defined for each control Axis.

Mode 0: With this Home Return Sequence , 1 Home Sensor is required.

In this process the Motor will Run at the Default Minimum Velocity (DEF.MIN) in the Home Return Direction and Stop when the Near Home Sensor is ON/Active.

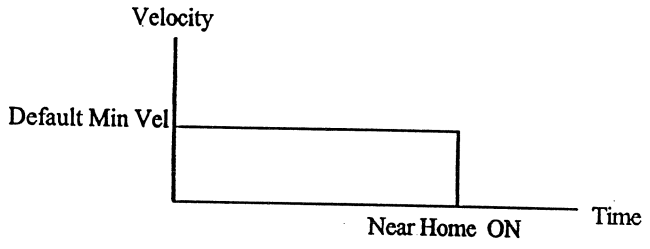


Figure 6.1 Home Return Sequence in Mode 0

Mode 1: With this Home Return Sequence 1 , 2 or 3 Sensors are required (Near Home and Forward or Reverse Limits or both limits).

In this process the Motor will first Run at the Default Maximum Velocity (DEF.MAX) in the Home Return Direction until the Near Home Sensor is ON/Active. At that time, the Motor will begin to decelerate. Once the Motor decelerates to the Default Minimum Velocity (DEF.MIN), it will change directions and Run at the Default Minimum Velocity until the Near Home Sensor is detected again. At this point the Motor will Stop when the Near Home Sensor switches from ON/Active to OFF/Not Active. In the Figures 6.2 and 6.3, the Home Return Direction is defined as the Reverse Direction.

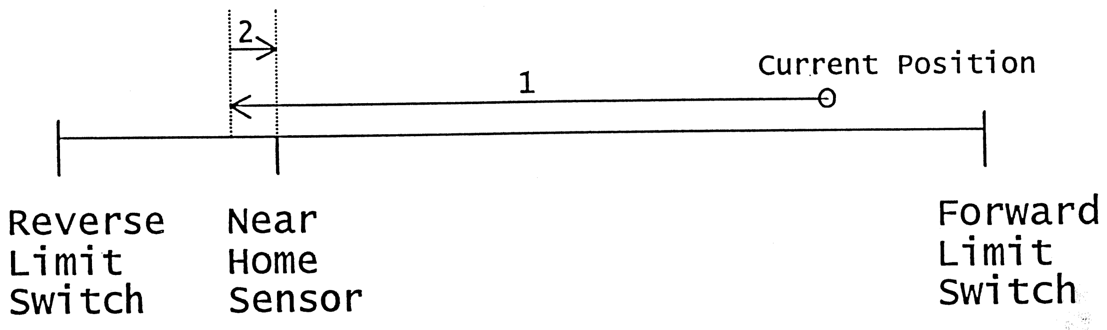


Figure 6.2 Mode 1: Near Home Sensor Triggered First

If the Reverse Limit or Forward Limit Sensor is triggered before the Near Home Sensor, which depends on the Home Return Direction, the Motor will Reverse Direction and Run at the Default Minimum Velocity until the Near Home Sensor is ON. The Motor will Stop when the Near Home Sensor switches from ON to OFF.

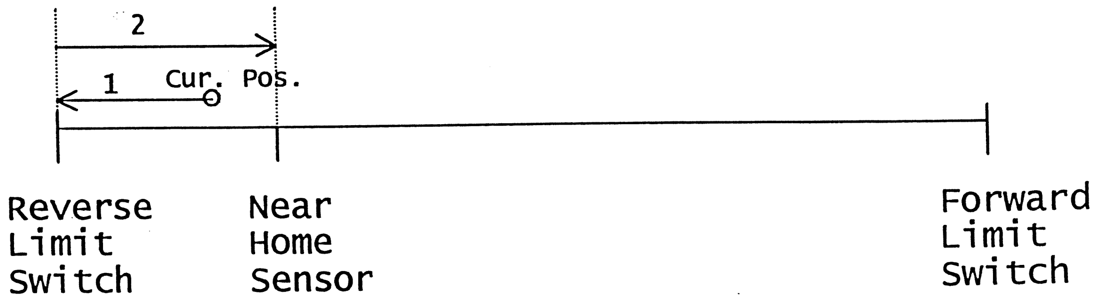


Figure 6.3 Mode 1: Limit Sensor Triggered First

Mode 2: With this Home Return Sequence 2, 3 or 4 Sensors are require (Home(Z-phase, Near Home and Forward & Reverse Limits or both limits)

In this process the motor will Run at the Default Maximum Velocity (DEF.MAX) in the Home Return Direction until the Near Home Sensor is ON/Active. At that time, the Motor will begin to decelerate. Once the Motor decelerates to the Default Minimum Velocity (DEF.MIN), it will change directions and Run at the Default Minimum Velocity. When the Near Home Sensor switches from ON to OFF, the Motor will change directions again and Run at the Default Minimum Velocity (DEF.MIN). The Motor will Stop when the Home(Z-phase) Sensor is turned ON. In Figures 6.4 and 6.5, the Home Return Direction is defined as Reverse.

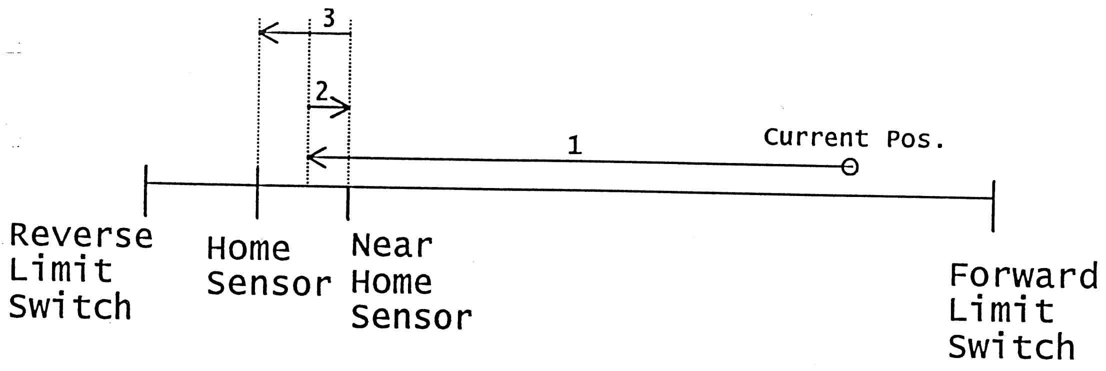


Figure 6.4 Mode 2: Near Home Sensor Triggered First

If the Forward Limit or Reverse Limit Sensor is triggered before the Near Home Sensor, which depends on the Home Return Direction, the Motor will Reverse its direction and Run at the Default Minimum Velocity. When the Near Home Sensor switches from ON to OFF, the motor will change directions again and Run at the Default Minimum Velocity (DEF.MIN). The Motor will Stop when the Home(Z-phase) Sensor is triggered ON.

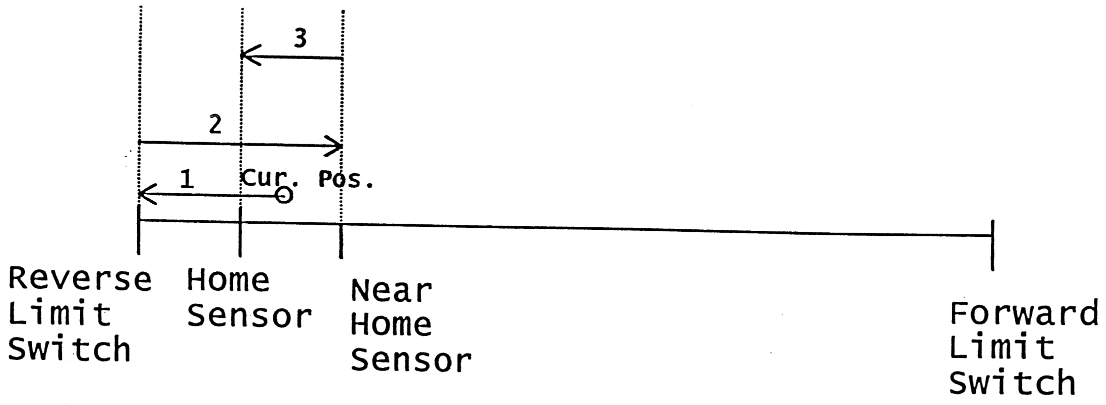


Figure 6.5 Mode 2: Limit Sensor Triggered First

Mode 3: With this Home Return Sequence 1 Sensor is required (Home or Z-phase(Index) of the Encoder).

In this process the Motor will Run at the Default Minimum Velocity (DEF.MIN) in the Homing Return Direction and Stop Immediately as soon as the Home(Z-phase) Sensor of the Encoder is triggered ON.

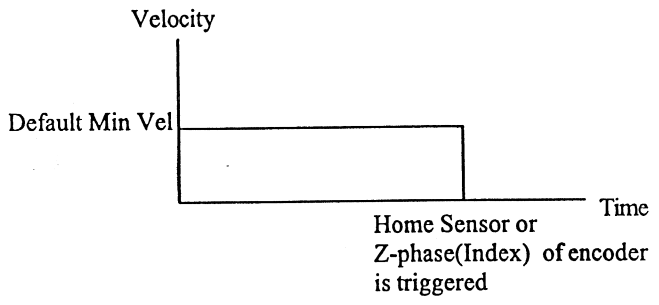


Figure 6.7 Homing Sequence in Mode 3

6.7 Joystick Configurations

If your application at certain times requires Manual operations, the MAC-300 Controller has the capability of using either Analog or Digital Joysticks which can be used to Manually Teach a machine important Position Data points or just move the Motors to specific positions.

To Define the Joystick Type

Use the following command in the System Configuration file to define the type of Joystick your application will require.

- JOY = 0 or 1

  0 = Define the Joysticks as Analog Joysticks (0 to +5 VDC)

  1 = Define the Joysticks as Digital Contact Joystick (Push Button)

Note: If a Digital Contact Joystick is used, you can move only one Axis at a time in the Teach/Manual mode. If more than Axis A is required you must use and Axis Selector switch to change Axes (See Section 15.2.1) .

To Define the Voltage Window for the Analog Joystick

- **AWINDOW = integer value (0 ~ 3)**

If an Analog Joystick is used , the Maximum Output Voltage from the Joystick to the Controller must range from 0 V to 5 V. Since some Analog Joysticks can vary in the amount of the actual Voltage sent to the Controller this system command allows you to define the range of Voltages that the Controller will sense. For example, a Joystick may except a Source Voltage of 0 to +5VDC however, when the Joystick is moved from its maximum forward point to its maximum reverse point the Voltage sent to the Controller may only be in the range of 1.88 to 3.12 Volts , this is when you may need to change the Voltage Window that the Controller senses.

System Command	Voltage Window	8 Bit A2D Reading
AWINDOW = 0	0.00 to 5.00 V	0 to 255
AWINDOW = 1	1.25 to 3.75 V	0 to 255
AWINDOW = 2	1.56 to 3.43 V	0 to 255
AWINDOW = 3	1.88 to 3.12 V	0 to 255

To Define the Threshold for an Analog Joystick

- **THR = integer value (0 ~ 128)**

Since Analog signals tend to fluctuate by small amounts when Joysticks remain in one position, this system command allows you to define the width of center point of the Analog Joystick and at what Voltage (A2D) settings the Motors will start to rotate in the Forward or Reverse directions with an Analog Joystick . As shown in Figure 6.6, the selected Axis will move in the Forward Direction if the Input Voltage is greater than 2.5V plus the Threshold Voltage, where Threshold Voltage = THR*2.5/128 Volts. The Axis will move in the Reverse Direction if the Input Voltage is less than 2.5V minus the Threshold Voltage. The Axis will Stop if the Input Voltage is between 2.5V minus the Threshold Voltage and 2.5V plus the Threshold Voltage.

Assuming that an Input Voltage of 0 to +5VDC is used :

If | Input Voltage - 2.5V | > Threshold Voltage:
Speed = (M - N) * (| I - 2.5 | - T) / (2.5 - T) + N

where M is Default Maximum speed, N is Default Minimum speed (refer to Sec. 6.9 for Default Motion Parameters), I is Input Voltage and T is Threshold Voltage value.

If | Input Voltage - 2.5V | <= Threshold Voltage: Speed = 0.

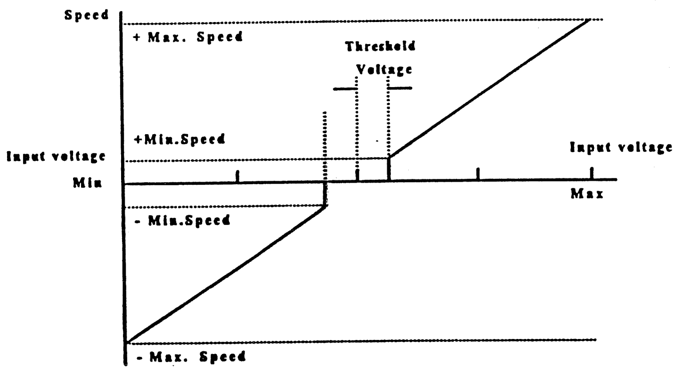


Figure 6.6 Analog Joy Stick Input vs. Motor Speed

To Specify which Axes will be controlled by Analog Joysticks

With the MAC-300 Controller up to 4 Axes can be simultaneously controlled by Analog Joysticks as described below.

- **AJOY = W or {A}{B}{C}{D}**

  W is used for a Single Analog Joystick which can control all 4 Motors with an Axis selector switch.

  {A}{B}{C}{D} is used to specify the Axis(es) that are controlled by a Multi-Axis Analog Joystick.

If you use a Single Axis Analog Joystick, you can move only one Axis at a time in Teach/Manual mode. If you use an Analog Joystick with 2 to 4 Analog Outputs (Multi-Axis Analog Joystick), you can move 2 to 4 Axes simultaneously in Teach/Manual mode. A2D0, A2D1, A2D2 and A2D3 will accept Analog Signals(0 to +5VDC) from your Joystick and control Axis A, B, C and D respectively. For example, if you use a 3 Axes Analog Joystick to control Axis A, C and D, you need to wire the 3 Analog Outputs from your Joystick to A2D0, A2D2 and A2D3 respectively and use the command AJOY = ACD in System Configuration file.

To Configure Inputs of Port A as JOG Inputs

When Analog Joysticks are required and small incremental steps (Jogging) is also required the following System Configuration command can be used.

- **IN.JOG = 0 , 1 or 2**

  0 = Do not use Input Port A's Input pins as JOG Inputs

  1 = Use Input Port A's Input pins as JOG Inputs, Edge Trigger --- a transition from High/Not Active to Low/Active sends out 1 Pulse to the Driver.

  2 = Use Input Port A's Input pins as JOG inputs, Level Trigger --- when an Input stays Low/Active, a Pulse train will be sent out continuously at the speed of 1 pulse per second to the Driver. The Pulse train stops when the Input turns High.

This function is valuable only when a Multi-Axes Analog Joystick is used and Jog movements are desired. By using IN.JOG = 1 or 2, you can have separate +JOG and -JOG Inputs for each Axis. When the Inputs of Port A are used as JOG Inputs, the definition for each Bit of Input Port A is listed below:

IN.A7	IN.A6	IN.A5	IN.A4	IN.A3	IN.A2	IN.A1	IN.A0
-JOG.D	+JOG.D	-JOG.C	+JOG.C	-JOG.B	+JOG.B	-JOG.A	+JOG.A

Table 6.2 Configure Input Port A's Inputs pins as Jog Inputs

If you use a 2 Axes Analog Joystick, you will only need to use 4 Inputs as JOG inputs for each Axis. In this case, you still have 4 Inputs of Port A that can be used as General Purpose Inputs. If a 3 Axes Analog Joystick is used, you will have 2 Inputs of Port A remaining that can be used as General Purpose Inputs. The command **AJOY** = **{A}{B}{C}{D}** also defines which Inputs of Port A can be used as JOG Inputs and the rest are General Purpose Inputs. For example, if you use the commands **AJOY=ACD** and **IN.JOG=1** in the System Configuration file, you have defined a 3 Axes Analog Joystick to control Axis A, C and D that will have +JOG, -JOG inputs for each Axis. Input pins A2 and A3 of Input Port A will now be General Purpose Inputs.

6.8 Setting the Velocity Range for each Axis

Since the MAC-300 Controller can work with many different types of Motor Drivers, it is capable or providing 8 Velocity/Speed ranges. **Note**: The terms Velocity, Speed and Pulse Rate are often used interchangeably

- **VEL.RNG.(Axis) = value**

The Velocity Range setting determines the upper and lower limits of the Velocities which may be used. There are 8 different settings (Table 6.3). The Speed setting resolution decreases as your Speed range increases. The minimum increment of the Speed setting is equal to the minimum value of the Speed range.

Example:

If the Maximum Speed of your system is 200,000 pps, the necessary Speed Range setting is 2 (4 - 250,000 pps). With this setting the Minimum Speed incremental value is 4 pps. Therefore, if the Speed which you enter is not divisible by 4, the Controller will automatically round the Speed to the nearest value divisible by 4.

Special Note: **Any time the Velocity Range is changed in the System Configuration File , it is important to Re-Load any Motion Control Programs that define the Motors Speeds with the MIN.VEL.x = (Number) and the RUN.VEL.x = (number) commands.**

Value	Velocity Range (PPS)
1	8 - 499,995
2	4 - 250,000
3	2.5 -166,664
4	2 - 125,000
5	1.5 - 100,000
6	1.2 - 83,000
7	1.1 - 71,000
8	1 - 65,535

Table 6.3 Velocity Range Values

6.9 Setting the Default Motion Parameters , Accel. , Decel., Max. & Min. Velocities

The Default Motion Parameters define the Default Acceleration Rate, Deceleration Rate, the Maximum Velocity and the Minimum Velocities for the specific Axis(Motor). These Motion Parameters are in effect if no Parameter setting commands are used before a Move Command. These Parameters will remain in effect until they are redefined within a Program(Sec. 7.1). The Joystick, Manual/Teach and the Home Return operations will always use the Default Parameters.

• **DEF.ACC.(Axis) = value**	; Default Acceleration Rate (units/sec^2)
• **DEF.DEC.(Axis) = value**	; Default Deceleration Rate (units/sec^2)
• **DEF.MAX.(Axis) = value**	; Default Max. Velocity (units/sec)
• **DEF.MIN.(Axis) = value**	; Default Min. Velocity (units/sec)

Note: **For 2 Phase Stepping Motors the Minimum Velocity is typically set to 500 pps while for 5 Phase Stepping Motors it is set to 1000 pps. This will prevent any resonance at rotational speeds of less than 1 rev/sec.**

6.10 Setting the Software Position Limits and Hardware Limit Errors

If you do not plan to use Forward and Reverse Limit Sensors in your application the following Control Parameters provide another method for setting Limits.

• **LIM.FOR.(Axis) = value or N**	; Forward Position Limit (unit)
• **LIM.REV.(Axis) = value or N**	; Reverse Position Limit (unit)

These commands define the Software Position Limits (Internal Limits) for each Axis. If Limits are assigned to an Axis, the Controller will check before each movement if the desired movement will hit the software limits. If this is the case, the Controller will Stop the Program and produce an ERROR signal. When no software limit is required for an Axis, an 'N" must be assigned to this command. In this case, the Controller will not check the Position Limit before each movement.

NOTE: The Forward and Reverse limits must be specified both as a Position value or both as "N" (not used).

To Specify if a Hardware Limit Sensor Error is necessary

If you do plan to use Hardware Sensor Limits and you wish to activate an Error condition when any of these limits have been reached, the following command will allow the machine to stop all operations when these limits have be reach.

• **ENA.HL = 0**	; Disable Hardware Position Limit Errors
• **ENA.HL = 1**	; Enable Hardware Position Limit Errors

When this feature is activated if any Axis of motion hits a Hardware Limit then the current Running Program will stop and the Red Error LED will turn on. The Error Codes for these limits can be found in Chapter 13 .

Chapter 7

This chapter describes all of the Movement commands that can be used in your Motion Control Programs. Their relationship is summarized in Table 7.1 (Sec. 7.6). See Appendix B for Programming Examples.

7.1 Movement Parameter Commands

The Movement parameter commands specify the movement profile characteristics of Acceleration, Deceleration, Start / Stop Speed, and Run/Max. Speed. Unless these values are specified within a Motion Control Program , the Default Motion Parameters (Sec. 6.9) in the System Configuration will be used. Once these values are set within a Program, they will become the effective values for the remainder of the Program or until they are changed again. Any one of these four may be specified at any point in the Motion Control Program. The next Program executed will Default back to the System Configuration Default Parameters until they are changed.

The following commands define the new Acceleration and Deceleration values within a Program.

- **Acceleration / Deceleration Rate Setting**

  ACC.RATE.(axis) = value

  DEC.RATE.(axis) = value

Where **value** is the Acceleration and Deceleration rate setting in units/sec^2. If the Scaling Factor for the specified Axis has been set to 1, the Units in this case will be in Pulses/sec^2.

Example: Set Axis A's Acceleration to 500 units/sec^2, Deceleration to be 300 units/sec^2.

ACC.RATE.A = 500
DEC.RATE.A = 300

The following command defines the new Start/Stop Speed within a Program.

- **Minimum Speed (Start / Stop Speed) Setting**

  MIN.VEL.(axis) = value

Where **value** equals the minimum speed (start / stop speed) in units/sec.

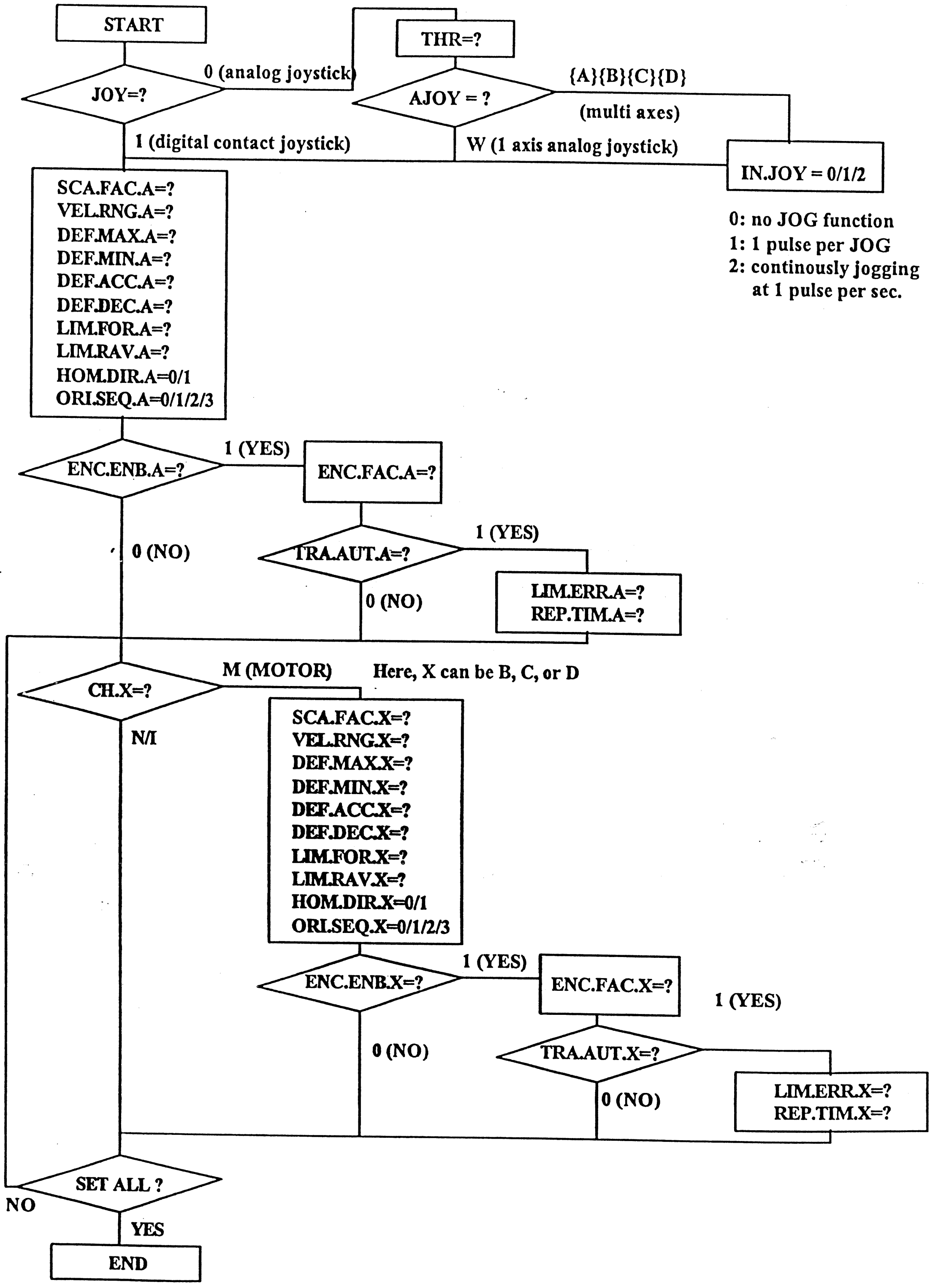
Example: Set Axis D's Start / Stop Speed to 500 units/sec:

MIN.VEL.D = 500

Note: **For 2 Phase Stepping Motors the Minimum Velocity is typically set to 500 pulses/sec while for 5 Phase Stepping Motors it is set to 1000 pulses/sec. This will prevent any resonance at rotational speeds of less than 1 rev/sec.**

6.11 System Configuration Setting Flow Chart

The following flow chart will help you determine which settings must be established and which are not necessary for your application.



The following command defines the Running or Maximum Speed of the Motor. The Maximum Speed of your mechanical system and Torque vs. Speed graphs for your Motor should be reviewed before you set this value.

- **Run Speed Setting**

  RUN.VEL.(axis) = value

Where value is the desired Run/Max. Speed in units/sec.

Example: Set axis B run speed to be 3000 unit/sec:

```
RUN.VEL.B = 3000
```

7.2 Distance and Position Setting Commands

The following commands can be used in your programs in order to specify Movement Distances and Final Positions.

- **Absolute Final Position Setting**

  FIN.POS.(axis) = value

This command defines the Absolute Final Position for the specified axis and must precede a Position Move command(POS.MOV.(axis)). Absolute Positions are with respect to the Current Zero Position. The Zero Position is defined by following four methods:

1. When the **Controller is Turned On** the Current Motor Position is defined as Zero.
2. When the **Controller is Reset** (Ch. 15) the Current Motor Positions are defined as Zero.
3. After a **Home Return Command** (Sec. 6.6) is completed, the Final Position is defined as Zero.
4. When a **Reset Position Counter Command** is executed (Sec. 10.2).

Example: Set Axis B's absolute Final Position to be the 80,000 unit position and Axis A's absolute Final Position to be the -15000 unit.

Note: If Scaling Factor is set to 1 then the unit value will be pulses or steps.

```
FIN.POS.B = 80000
FIN.POS.A = -15000
POS.MOV.AB                  ; Move Axis A& B to the Defined Positions.
```

- **Increment Final Position Distance**
  INC.DIS.(axis) = value

This command increments the Final Position distance for each Axis relative to its Final Position setting. This value must be positive and less than 16,777,215. This command overwrites the current Final Position.

Example: The current Final Position of Axis B is 1000 and the Current Position = 0.

```
        FIN.POS.B=1000        ; Assign the Final Position as 1000
        INC.DIS.B=500         ; Add 500 to the current Final Position, this
                              ; overwrites the current final position to 1500.
        POS.MOV.B             ; The Current Position will be 1500 after this move
                              ; is completed.
```

Note: The Final Position can not be changed once the Position Move has started.

- **Index Distance Setting**
  INDEX.(axis) = value

This command sets the Index/Incremental Move Distance for the specified Axis and must precede a Index Move command(MOV.INDEX.(axis)). This value must be positive and less than 16,777,215.

Example: Set Axis C's Index Distance to 5,000 units.

INDEX.C = 5000

7.3 Movement Commands

The following commands, used within your Programs, initiate the movements defined by the Final Position command and the Index Distance command mention above .

- **Move to a Final Position**
  POS.MOV.(axis)(axis)(axis)(axis)

This command can be used to simultaneously move each of the specified Axes to their previously assigned Final Positions or this command can be used to move each Axis separately.

<Ex> Move Axes A , B & C to their respective final positions

POS.MOV.AB

POS.MOV.C

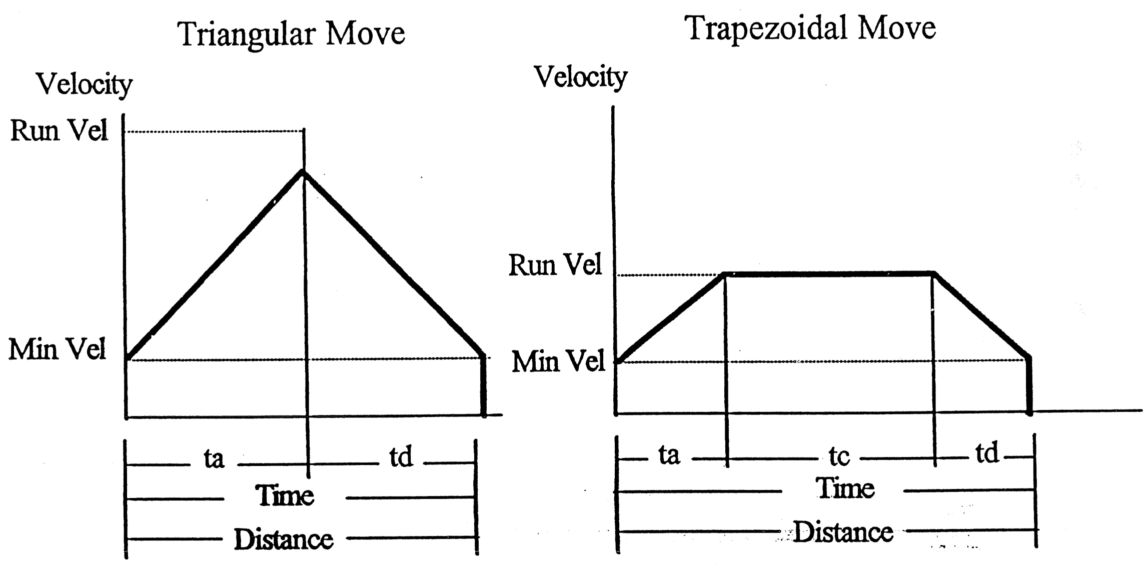
By using this command, a Triangular move or Trapezoidal move may be executed. The Controller determines the appropriate move profile automatically.

Triangular Move:

1. The Motor Accelerates from Start / Stop Speed at the ACC.RATE
2. The Motor does not reach the RUN.VEL(Max.Vel)
3. At the half way point of the move, Deceleration begins at the DEC.RATE. The Deceleration position is automatically calculated by the controller.
4. The Motor Decelerates to Start / Stop Speed and Stops.

Trapezoidal Move:

1. The Motor Accelerates from Start / Stop Speed at the ACC.RATE
2. The Motor achieves RUN.VEL(Max. Vel) and Acceleration ends.
3. The Motor runs at RUN.VEL until it reaches the Deceleration position which is automatically calculated by the Controller.
4. The Motor Decelerates at the DEC.RATE to Start / Stop Speed and Stops.



ta = acceleration time
tc = constant velocity time
td = deceleration time

Figure 7.1 Move Profiles

- **Index/Incremental Movements**

  MOV.INDEX.(±axis)(±axis)(±axis)(±axis)

This command, used within your Programs, Indexes / Increments the Motor(s) a distance specified by the INDEX.(axis) command (Sec. 7.2). The Index movement profiles are also calculated automatically by the Controller. Refer to the Position Move command for move profiles.

Example: Move Axis A forward 5000 units and Axis B backward 2000 units.

```
INDEX.A = 5000
MOV.INDEX .A      ; Move Axis A first

INDEX.B=2000
MOV.INDEX.-B      ; Move Axis B second
WAIT.STOP.AB      ; Wait for Motors to Stop

MOV.INDEX.A-B     ; Move both Motors simultaneously
```

- **Constant Velocity Movements**

  VEL.MOV.(±axis)(±axis)(±axis)(±axis)

This command moves the specified Axes at a Constant Velocity defined by either the Default Maximum Velocity set in the System Configuration file or by the previously defined RUN.VEL command. This command will initiate the change of the Current Velocity to the new RUN.VEL without stopping the Motor (the Acceleration and Deceleration Rates will apply). The Motor(s) will continue to Run until a NOR.STOP or EME.STOP command is executed. Refer to Example 1 in Appendix B for more details.

Example: Move Axis A, B, and C at the desired velocity with C moving in the reverse direction.

```
VEL.MOV.AB-C
```

- **Home Return Command**
  HOME.(axis)(axis)(axis)(axis)

This command, used within your Programs, will return the specified Axes to their Home Positions. This function is used in conjunction with the Home Return Sequences described in the System Configuration file settings , Section 6.6.

Example: Return Axis A and Axis B to their Home Positions.
  HOME .AB

7.4 Stopping and Waiting Commands

The following commands can be used within your Programs in order the Stop a Motor with deceleration, Stop a Motor immediately or Wait until a Motor has finished its move before proceeding within a Program.

- **Normal Stop the Motor(s) with Deceleration**
  NOR.STOP.(axis)(axis)(axis)(axis)

This command Decelerates and Stops for all specified Axes. This command will not Stop a Motor at a defined position like the Position Move command mention earlier.

Example: Stop Axis A & B with Deceleration
  NOR.STOP.AB

- **Emergency Stop without Deceleration**
  EME.STOP.(axis)(axis)(axis)(axis)

This command Immediately Stops the specified Axes without Deceleration. This command will not Stop a Motor at a defined position like the Position Move command mention earlier.

Example: Stop Axis A & C Immediately.
  EME.STOP.AC

- **Wait Until Specified Moving Axes Stop**
  WAIT.STOP.(axis)(axis)(axis)(axis)

When this command is executed, the Controllers program flow will halt and wait until the specified Axes have stopped. After the Motor(s) have stopped, the Controller executes the next Motion Control Program command line.

Example: The following Program moves Axis B to Position 0 then turns on Output A0.

```
FIN.POS.B = DATA.B0      : Set the desired Final Position as Position Data 0
POS.MOV.B                : Start the Position movement
WAIT.STOP.B              : Wait until the Position movement is complete.
OUT.A0 = 0               : Turn-On Output A0
```

7.5 Toggle movements

The following commands allow you to Toggling motors back and forth between two points. Toggling is a continuous reverse then forward Index movement between two points.Toggling starts with a reverse Index move from the starting point and then ends at the starting point. The Toggling function can be accomplished in two ways:

1. Using the EN.TOG and DIS.TOG commands.
2. Using the EN.TIN and DIS.TIN commands.

- **Start Toggle Movements**
  EN.TOG.(axis)(axis)(axis)(axis)

This command starts the Toggle movement for the specified Axes. The Toggle Distance, Toggle Speed and Acceleration & Deceleration Rates must be preset by **INDEX.(axis), MIN.VEL.(axis), RUN.VEL.(axis), ACC.RATE.(axis)** and **DEC.RATE.(axis).**

Example: Start Toggle movement for Axis A and C

```
EN.TOG.AC
```

- **Stop Toggle Movements Activated by EN.TOG Command**
  DIS.TOG.(axis)(axis)(axis)(axis)

When this command is executed, the specified Axes will Stop Toggling after the last toggle movement is completed. This command has no effect on those Toggle movements activated by onboard Input pins used with the EN.TIN command.

Example: Stop Toggle movements for Axis A and C

```
DIS.TOG.AC
```

- **Enable Toggle Trigger Input**
  EN.TIN.(axis)(axis)(axis)(axis)

This command Enables the Input Trigger Toggle function for the specified Axes. The Input pins A0 - A3 of Port A are used to Trigger the Toggle function. There is a one to one relationship between the Input pins and the specified Axes: Input A0-A3=> Axes A-D. Therefore, each Axis (A-D) can independently begin Toggling and End Toggling by controlling the status of Inputs A0 to A3.

Example: Enable Toggle trigger input for axis B and D

EN.TIN.BD

After this command has been executed and Input Pin A1 of Port A is turned ON, Axis B will begin to Toggle. When pin A1 is turned OFF, Axis B will Stop after the last toggle movement is completed. Input pin A3 of port A starts and stops the Toggle movements for Axis D in the same manner. For more details on I/O functions, see Chapter 10.

- **Disable Toggle Trigger Input**
  DIS.TIN.(axis)(axis)(axis)(axis)

This command Disables the Input Trigger Toggle function for the specified Axes. This command will Stop the Toggle movements for those activated by Input pins A0-A3 of Port A after the last toggle movements are completed. In addition, the status of the Input pins of Port A will no longer effect the Toggle function for those specified Axes.

Example: Disable Toggle Trigger Inputs for Axis B and D

DIS.TIN.BD

This command will Stop the Toggling of Axis B and D ignoring the status of Input pins A1 and A3 on Port A.

7.6 Summary

This table summarizes the commands in this chapter that affect each other. Any of the variables set in Column 1 will affect those in Column 2.

Column 1	Column 2
ACC.RATE DEC.RATE MIN.VEL RUN.VEL	POS.MOV VEL.MOV MOV.INDEX EN.TOG EN.TIN
FIN.POS	POS.MOV
INC.DIS	POS.MOV
INDEX.	MOV.INDEX EN.TOG EN.TIN

Table 7.1 Command Relationships

Chapter 8

There are two different data types available for the MAC-300 Controller: Data Storage Registers and stored Position Data points. Each data type has different characteristics which will be looked at in this chapter.

8.1 Data Storage Registers

There are three types of Data Storage Registers available with this Controller:

- SREG(0-63) - Short registers (any ***integer*** between 0 and 255)
- REG(0-31) - Regular registers (any ***integer*** between 0 and 65535)
- LREG(0-31) - Long registers (any ***integer*** between -2.14 E 9 and +2.14 E 9)

Storage Data Registers are similar to variables in Computer Programming and they have three distinct advantages over using simple Integer numbers:

1. Data Storage Registers can be down loaded into the Controllers Non-Volatile memory during idle mode. This allows you to change data parameters without modifying the program itself.

2. Data Storage Registers can be changed within a program. Arithmetic operations (Addition, Subtraction, Multiplication and Division) are available for Data Storage Registers only (Sec. 10.5).

3. Data type conversions are possible with the use of Data Storage Registers.

   The Data length can be extended using the following conversions:

   LREG# = REG#
   LREG# = SREG#
   REG# = SREG#.

   Zeros are added to the upper byte of the larger Register before the shorter Register value is added.

Example: Application of extending the data length:
Final position is the potential of the A/D input.

```
SREG0 = A2D0              ; Save the A2D0 value to SREG #0
LREG0 = SREG0             ; Extend a 1-byte Register to 4-byte Register
LREG0 = LREG0 * 980       ; Multiply by the constant, 980
FIN.POS.A = LREG1         ;Assign the calculation result to final position
POS.MOV.A                 ; Move to the desired Final Position
```

Important Notes:

1. **The Scaling Factor has no effect on Storage Register values.** Therefore, when a Register value is used with Move parameters (Sec. 7.1) or Distance settings (Sec. 7.2), they will be read directly as Pulses.

2. Whenever the Controller is first turn on, it will use the initial Register values that were saved during Down loading a Register file($WR.VAR). After a Program has started via th START Input or Host PC Run command any changes made to the Registers prior the Start signal will be used within the Program. When one Program jumps to another Program all present Register values are maintained. When a Program is stopped and restart , during the restart the Register values from the previous Program cycle will remain as long as power has not been removed or the Reset Input has not be activated.

There are only three times when the Registers are reset to their original down loaded condition automatically :

a) During Power Up.
b) After the Reset Input has been activated.
c) After the same Variable File has been down loaded again in the Idle mode.

8.2 Position Data points

Position Data points can only be defined by using the Store Position Input during the Manual/Teach operation (Ch. 15) or by down loading a Position Data file using the Host Computer(PC). <u>This Data Cannot be Modified within a Program</u>. The format for Position Data points is as follows.

DATA.(Axis)(0-63)= value

Although these Position Data points values can not be change within a Program , they can be used to move the Motors to Absolute or Final Positions.

Example: Use Data point A5 to setup the next Final Position value.

FIN.POS.A=DATA.A5

8.3 Down Loading Storage Register values and Position Data points

Storage Registers or Position Data points can be used in any of your Motion Control Programs. To define these values prior to running your Programs you can down load Data Storage Register and Position Data point files. These files can only be down loaded during the Idle mode. In order to down load these data files, use the following formats shown in the examples below. The down load command is located under the **F3 Commands** menu.

Example 1: Data Storage Register/Variable down loading format

```
@(0,1,2...)          ; Call  Controller (# 0,1 or 2...) command
$WR.VAR              ; Write Register Values to memory
LREG6 = 110          ; Set Long Register #6 = 110
LREG12 = 12362       ; Set Long Register #12 = 12362
REG28 = 32123        ; Set Register #28 = 32123
SREG30 = 2132        ; Set Short Register #30 = 2132
END                  ; End the Down Loading procedure.
```

Example 2: Position Data point down loading format

```
@(0,1,2...)          ; Call Controller (# 0, 1 or 2.... ) command
$WR.DATA             ; Write a Position Data point file to memory
DATA.A5 = -1233      ; Set Axis A Position Data point  #5 = -1233
DATA.B10 = 301.2     ; Set Axis B Position Data point #10 = 301.2
DATA.D50 = 12.321    ; Set Axis D Position Data point #50 = 12.321
END                  ; End the Down Loading procedure.
```

The following is a brief example of a Program which will use some of the Data previously down loaded.

Example 3:

```
@0                   ; Call Controller #0
$WR.PROG7            ; Write Program #7 to Memory

FIN.POS.A=DATA.A5    ; Define Final Position as Data Ponit A5
POS.MOV.A            ; Move to the Final Position
WAIT.STOP.A          ; Wait until Motor Stops

INDEX.A=LREG6        ; Define the Index Distance as LREG6
MOV.INDEX.A          ; Move the Index Distance amount
WAIT.STOP.A          ; Wait until Motor Stops

END                  ; End Program Execution, Return to Idle mode
```

Note: The Current Position of Axis A after Program #7 will be ... -1123

Chapter 9

This chapter descibes the commands that can be used to Control the flow of your Programs. The following commands allow you to check if certain conditions have or have not be met before proceeding within a Program. These commands also allow you to branch to another Program , Subroutine or Program location when desired.

9.1 Labels or Markers used for Program Jumping

L(#) : # = **0 - 31**

Labels are used as Program markers in order perform Loops or Jumps within a Program. They are used in conjunction with **IF ... THEN, GOTO,** and **CALL** commands. A maximum of 32 labels may be used in a program.

Example: Use Label #0 to form a Time Delay Loop

```
TIMER0=0                ; Set the Initial value of Software Timer#0 to 0
L0                      ; Assign Label #0
IF TIMER0<100 THEN L0   ; Wait Loop,wait for 100 x 10 ms = 1 sec
```

9.2 Goto Statements

GOTO L#

This command will force the Program Flow to Jump to the command line specified by label L#.

Example: Define Label #20 and then use the GOTO L20 to form an Infinite Loop

```
L20
••••••
GOTO L20
```

- **Goto Program Statements**
  GOTO PROG#

This command will force the Program Flow of the present Program to the first command line of the specified new Program. All Storage Registers and Motion Parameters will remain the same assigned values during this transitions from one Program to the other.

Example: Use GOTO PROG21 as the last command before END in Program 20, so once Program 20 is complete, the Controller will automatically run Program 21.

```
@0               ;Call Controller #0
$WR.PROG20       ; Writer Program #20 to Memory
......
GOTO PROG21      ; Go to and Start Program 21
END              ; End Write/Down Loading operation
```

9.3 Call Statements

CALL L#

This command, when added to your Programs, will force the Program Flow to Jump to the command line specified by **L#**. When the return command **RET** is executed the Program Flow will return to the command line following the **CALL L#**. This command is typically is referred to as a Subroutine Call Command.

Example: Call a Subroutine Labeled as L10

```
CALL L10         : Call Subroutine L10
......
L10              : Beginning of Subroutine L10
CALL L20         : Call Subroutine L20 , example: Delay Subroutine
......
RET              : End of Subroutine L10
```

Note: When using Call Commands you may have as many additional Call Commands within the first subroutine as you required however , when branching from one subroutine to the next the number of branches is limited to 3 before you must use the Return Command.

- **Return from a Subroutine**
  RET

When this command is executed, the Program Flow will complete the Subroutine and Return to the command line following the last **CALL L#** command.

9.4 IF ... THEN Conditional Statements

IF *expression* THEN L# or IF *expression* THEN PROG#

or

IF *expression* THEN CALL L#

This Command will control which Label (L#) or Program # or Subroutine the present Program will go to if the expression is true. If the expression is false, the Program Flow will go to the next command line. The ***expression*** must be in the following format: **Variable ~ Operator ~ Variable.** The first **Variable** must be a variable from Table 9.1. The **Operator** is a Conditional Operator from the list below. The second **Variable** can either be a variable from Table 9.1 or a number. **NOTE: The two variables must be of the same data type(1 , 2 or 4 bytes long).**

Example 1: Compare the Current Position to a Long Register.

```
L18                                   ; Assign Label #18
IF CUR.POS.A<LREG25 THEN L18              ; Check Axis A's Position, If less
                                          ; than Long Register #25 then Wait.
                                      ; Note: Each Data Type is 4 bytes
                                      ; long.
```

Example 2: Check for Input A4 of Port A and go to Program 10 if active.

```
IF IN.A4 = = 0 THEN PROG10        ; Check the status of  Input A4 , If active
                                  ; then goto Program #10 ,otherwise proceed.
```

Note: Unlike the Call Command mentioned in section 9.3 , once the IF ... THEN L# or IF...Then Prog# statement does branch it will not return to the command line below the IF ...THEN statement. If you wish to return to the command line below the IF ... THEN you must use the

IF *expression* THEN CALL L# command in order to return to the line below the IF...THEN statement after the Subroutine has been performed . See Section 9.3 for the proper use of Call commands.

The following are valid Conditional Operators that can be used in the IF...THEN statements mention above.

- **Valid Conditional Operators**

= =	**: equal**
! =	**: not equal**
<	**: less than**
>	**: greater than**
< =	**: less than or equal**
> =	**: greater than or equal**

Variable	Read/Write	Data Types	Descriptions
A2D(0-3)	R	1-Byte decimal	A/D Input value
TIMER(0-7)	R/W	4-Byte decimal	Software timer counter value
SREG(0-63)	R/W	1-Byte decimal	Short Register value
REG(0-32)	R/W	2-Byte decimal	Normal Register value
LREG(0-32)	R/W	4-Byte decimal	Long Register value
MOV.STA.(axis)	R	0/1 , Bit	Motion Status, 0:stop, 1:runing
IN.(port)(0-7)	R	0/1 , Bit	Input Bit status of the Input Port
IN.(port)	R	1-Byte Hex	Status of the Input port
OUT.(port)(0-7)	R/W	0/1 , Bit	Output Bit status of the Output Port
OUT.(port)	R/W	1-Byte Hex	Status of the Output port
ACT.POS.(axis)	R	4-Byte decimal, signed	Encoder Position Counter value
CUR.POS.(axis)	R	4-Byte decimal, signed	Current Position / Output Pulse Counter value
DATA.(axis)	R	4-Byte decimal, signed	Stored Position Data point value

Table 9.1 Variable List

Note:

1-Byte Decimal = Integer numbers in the range of 0 to 255.
Example: SREG7=A2D0=255

2-Byte Decimal = Integer numbers in the range of 0 to 64,535.
Example: REG2=60,000

4-Byte Decimal = Integer numbers in the range of 0 to 4,294,967,295.
Example: TIMER7 = 5,700,000

4-Byte decimal, signed = Integer numbers in the range of ±2,147,483,647.
Example: LREG15=CUR.POS.A

All Data Types with 1-Byte Hex can also be represented by 8-Bit Binary numbers.
Example: OUT.A = 051H = 01010001B

Chapter 10

10.1 Assignment Operators

To assign values to Registers, Motion Parameters, Index Distances, Final Distances or Timers you must use the following Operator.

Assignment Operator: = (Equal Sign)

Variable1 = Variable2 or a Number

Examples:

LREG1 = CUR.POS.A	; Save the Current Position of Axis A in Long Register #1
SREG2= SREG2 + 1	; Increment Short Register #2 , the Parts Counter.
LREG3=13500	; Save an Index Distance in Long Register #3

If Variable 1 is a FIN.POS , LREG or SREG variable then Variable 2 must be of the same Data type. If Variable 2 is a CUR.POS or ACT. POS variable then Variable 1 must be a LREG Variable which is of the same data type(4 bytes). Each Variable Data Type must match in your statements. See the next page for the absolute lengths of these variables and Table 9.1 .

Examples:

TIMER2 = 0	; Set Timer #2 to Zero, Clear Timer #2.
LREG7 = CUR.POS.A	; Save the Current Position of Axis A to 4 byte register.
SREG10 = IN.A	; Save the Input Port A's status to 1 byte register

Valid Assignments

Parameter Assignments examples

ACC.RATE.A=1500	Integer Value 3	ACC.RATE.A=LREG7	Long Register
DEC.RATE.A=2000	Integer Value 3	DEC.RATE.A=LREG9	Long Register
MIN.VEL.A=1500	Integer Value 3	MIN.VEL.A=LREG17	Long Register
RUN.VEL.A=2000	Integer Value 3	RUN.VEL.A=LREG19	Long Register
INC.DIS.A=15000	Integer Value 3	INC.DIS.A=LREG27	Long Register
INDEX.A=20000	Integer Value 3	INDEX.A=LREG29	Long Register
FIN.POS.A=15000	Integer Value 4	FIN.POS.A=LREG20	Long Register
TIMER# =20000	Integer Value 3	TIMER# =LREG21	Long Register

Variable Storage Register Assignments

SREG23=15000	Integer Value 1 **SREG23=SREG7**	Short Registers
REG20=1550	Integer Value 2 **REG20=REG2**	Word Registers
LREG20=2000	Integer Value 4 **LREG6=LREG3**	Long Registers

Note:

Integer Value 1 = 0 to 255	; Short Register
Integer Value 2 = 0 to 65535	; Word Register
Integer Value 3 = 0 to 4,294,967,295	; Motion Parameters & Timers
Integer Value 4 = -2,147,483,647 to +2,147,483,647	; Long Registers

Examples using Storage Registers:

SREG17=A2D3	;Set Short Register to Analog to Digital Decimal Value
SREG55=IN.A	;Set Short Register to Input Port Decimal Value
REG20=A2D1*10	;Set Word Register to Analog to Digital Decimal Value x10
LREG4=CUR.POS.A	;Set Long Register to Current Position. Positive or Negative

10.2 Reset Positions Counters to Zero

The following command can be used within a Program in order to Reset the Current Position Counters to zero.

- **Reset Position Counters to Zero**
  RESET.(axis)(axis)(axis)(axis)

If the Motors are not in motion the Encoder and Position Counters will be set to Zero . If the Motors are in motion then this command will Wait until all specified Axes Stop Running, then set both the Encoder Counters (ACT.POS) and the Current Position Counters (CUR.POS) to zero for the specified Axes. Anytime a Reset command is given for a specified Axis at the same time, a pulse is sent out on the -CR pin, (Counter Reset Output for the Driver) of the specified Axis.

Example: Set Axis A and C's Current Position to Zero.
RESET.AC

10.3 Assign Initial Value to Software Timers.

To Reset any of the 8 software Timers to zero is a simple matter of assigning that Timer a zero value. The following shows examples of setting the Timers to any value.

Note: These Counter will count from 0 to 4,294,967,295
in increments of 10 milliseconds

```
TIMER# = value            ; TIMER0 thru TIMER7 can be used.
```

The above command sets the initial value for the specified software timer.

Example: Use Software Timer#1 for a 10 seconds Delay.

```
TIMER1=0                     ; Initialize Software Timer#1 to Zero
L1                           ; Assign a Label for returning
IF TIMER1 < 1000 THEN L1     ; Loop for 1000 x 10 ms = 10 sec
```

10.4 Conditional Operators

The following Operators are valid for the conditional IF...THEN statements used in Programming the MAC-300 Controller.

```
= =       ; Equivalent
! =       ; Not Equal
<         ; Less Than
>         ; Greater Than
<=        ; Less Than or Equal
>=        ; Greater Than or Equal
```

Example:

```
IF CUR.POS.A<30000 THEN L7     ; If the Current Position of Axis A is less
                               ; than 3000 then Jump to Label L7
IF IN.A ! = 54H THEN L1        ; If the Input Port A is not equal to 54
                         ; Hexadecimal then Jump to Label L1
IF OUT.A2 = = 0 THEN PROG2     ; If Output A2 is ON then Jump to Prog#2
```

10.5 Mathematical Operations

To perform mathematical operations using Storage Registers any statement including an assignment operator must follow one of the following general formats:

Variable1 = Variable2 Arithmetic Operator Variable3

In the above mathematical statement ,**Variable 1** can only be one of the following Storage Registers: SREG, REG or LREG. Each of these Register contain Integer values as shown below.

Variable 2 should be another Storage Register or a Parameter (See Table 9.1). If you are using a Scaling Factor with your motion parameters (Current Position, Encoder Position etc.) then these values will be saved in the Storage Registers as Pulse Counts without any Scaling Factor.

Variable 3 can only be another Storage Register or a Number.

Before assigning values or using the Registers for data manipulation make sure that the you use the correct data types as shown in following table. Also see Table 9.1

Register Type	Data Type	# of Registers	Data Range
SREG#	1 Byte	0 to 63	0 to 255
REG#	2 Bytes	0 to 31	0 to 65535
LREG#	4 Bytes	0 to 31	-2,147,483,647 to +2,147,483,647

Valid Arithmetic Operators:

+ ; **Addition**

- ; **Subtraction**

* ; **Multiplication**

/ ; **Division**

Note: **When performing Calculations if the value calculated is beyond the range of the Data Type then an Error will occur and the Motion Control Program will stop.**

When performing Division if remainder of the value calculated is less than 0.5 then the result will be rounded thus dropping the remainder, while if the remainder is greater than or equal to 0.5 then the result will be rounded up by 1.

When performing Multiplication or Division, Variable 3 should never less than -65535 or greater than 65535.

• Mathematical Calculations

The following are some examples of the valid calculations that can be performed and some invalid commands.

Valid commands	Invalid commands	Problem
SREG4 = A2D0	SREG4 = ACT.POS.A	; Data Types do not match. ; See Table 9.1
SREG2 = IN.A	SREG55 = REG1+REG2	; Data Types do not match.
SREG6=SREG5 + SREG1	SREG5 = REG10	; Data Types do not match.
SREG7 = SREG2 /10	SREG3 = SREG6 / 300	; Short Registers can not be ; divided by numbers greater ; than 255
SREG5 = SREG2+2	SREG3 = SREG6 * 300	; Short Registers Product can ; not be greater than 255
REG10 = A2D3*10	REG11 = CUR.POS.A	; Data Types do not match.
REG22 = REG2 + REG2	REG5 = LREG1 + LREG2	; Data Types do not match.
REG1 = REG9 + REG10	REG15 = LREG12	; Data Types do not match.
REG4 = REG4 /REG1	REG3 = REG9 / 68000	; Word Registers can not be ; divided by numbers greater ; than 65535
REG7 = REG20+200	REG2 = REG11 * 68000	; Word Registers Product can ; not be greater than 65535
LREG10 = CUR.POS.A	LREG20 = IN.A	; Data Types do not match.
LREG10 = CUR.POS.A* -1	LREG20 = ACC.RATE.A	; Data Types do not match.
LREG10 = ACT.POS.A+100	LREG20 = RUN.VEL.A	; Data Types do not match.
LREG0 = LREG2 / 500	LREG4 = LREG6 / 70000	; Long Registers can not be ; divided by numbers less ; than or greater than +/- 65535
LREG10 = LREG5 * 500	LREG7 = LREG2 * -70000	; Long Registers can not be ; multiplied by numbers less ; than or greater than +/- 65535
LREG0 = LREG2 /LREG10		
LREG0 = LREG2 * LREG10		
LREG0 = LREG2 / -10		
LREG0 = LREG2 * -1		
FIN.POS.B = LREG1	FIN.POS.B = LREG1 * 5	; Value must be calculated first.
INDEX.A = LREG15	INDEX.A= LREG1 + 500	; Value must be calculated first.
OUT.B = 10111001B	OUT.B = IN.A	; (use: SREG2=IN.A then ; OUT.B=SREG2 instead)
OUT.A0 = 1	OUT.A=2	; Data Types do not match.

Common uses of Registers

LREG0 = CUR.POS.A * -1	; Save the Current Position as a Negative value
LREG7 = LREG2*2	; Double the Value of Long Register #2
SREG1= SREG1+1	; Increment Counter, Increment Short Register #1
LREG25=LREG7 / LREG10	; Divide one value by another and place result in Long Register #25

Chapter 11

11.1 General Purpose Inputs

The general purpose Inputs have the names of IN.(Port)0, IN.(Port)1, ... , IN.(Port)7 which correspond to the Input pin described in Section 3.2. The Inputs can be used in two ways:

1. They can be monitored using the **IF ... THEN** statements within the Program.
2. Their status can be saved in Short Registers using **SREG# = IN.(Port)** statements within the Program.
3. They can be monitored using a Host Computer, PC (Ch. 5).

Example:

```
L12                              ; Assign Label #12 for this Loop
IF IN.A2 = = 1 THEN L12          ; Check if Input A2 is OFF.
                                 ; If true then goto Label 12, L12
IF IN.A3 = = 0 THEN PROG7        ; Check if Input A3 is ON.
                                 ; If true then goto Program # 7
IF IN.A = = 00101010B THEN L5    ; Check if Input Port A = Binary
                                 ; value 00101010
IF IN.A = = 0AFH THEN L5         ; Check if Input Port A = Hex , AF
```

11.2 Dedicated Inputs

Home, Near Home, Reverse Limit, Forward Limit

The Home, Near Home, Forward Limit, and Reverse Limit functions are used in conjunction with the Home Return command (Sec. 7.3) while the Limits always provide mechanical safety to the system. Wiring diagrams for these Dedicated Inputs are located in Section 3.4. Unless a Home command is currently being performed, an Input to Reverse Limit or Forward Limit will cause the Controller to perform an Emergency Stop.

11.3 Analog to Digital Inputs

Depending on the Joystick configuration (refer to Appendix A), there are up to 4 Analog to Digital Inputs that can be utilized. They have the names A2D0, A2D1, A2D2, and A2D3.
Analog to Digital inputs can be used in two ways:

1. They can be monitored and used in **IF ...THEN** statements within the program.
2. Their values can be saved in registers using **SREG# = A2D#** statements within the program.
2. They can be monitored from the host PC (Ch. 5).

Example: Check the value of the Analog to Digital Input A2D0.

```
L1                        ; Assign Label #1 for Loop
IF A2D0 < 200 THEN L1     ; Conditional Statement checks value of
                          ; A2D0 when A2D0 is above 200,
OUT.A1= 1                 ; the Program turns off Output A1.
```

11.4 General Purpose Outputs

The general purpose Outputs have the following names: **OUT(Port)(0-7)** . These can be controlled from anywhere within the program by designating the Output ON = 0 or OFF=1.

Example:

```
OUT.A4=0          ; Turn On general purpose Output A4.
```

Since more than one Output can be controlled simultaneously, setting the command OUT.(Port) equal to a Eight-Bit Binary number (or a one-byte Hex) will control all Outputs with no delay time.
<Ex>

```
OUT.A=01010101B  ;Turn-On Outputs 1,3,5 and 7 and Turn-Off 0,2,4 and 6.
                 ;The B at the end signifies a Binary number.
OUT.B=55H        ;Turn-On Outputs 1,3,5 and 7 and Turn-Off 0,2,4 and 6.
                 ;The H at the end signifies a Hexadecimal number
OUT.B=0AAH       ;Turn-On Outputs 0,2,4 and 6 and Turn-Off 1,3,5 and 7.
OUT.C=SREG2      ;Send the value of Short Register #2 to Output Port C
```

Note: For Hexadecimal numbers, if the first digit is a letter (A-F) a zero must precede it.

11.5 Motor Free Output

MF.ON.(axis)(axis)(axis)(axis)

MF.OFF.(axis)(axis)(axis)(axis)

If your application requires turning off the Motor Winding Currents when the machine is not running, the Motor free command can be used in conjunction with Current Cut Off Input of the Motor Driver(see Sect. 3.6, Output Circuit #1). The **MF.ON.(axis)** command will internally connect the MF output with O.COM. The **MF.OFF.(axis)** command will disconnect the MF Output and O.COM. The default setting is MF.OFF for all Axes.

Chapter 12

This Chapter lists all the commands that can be used in Motion Control Programs.

12.1 Parameter Assignment Commands

ACC.RATE.{axis}=LREG#/Value	; Set Accel and Decel Rate for Axis ?
MIN.VEL.{axis}=LREG#/Value	; Set Start/Stop Speed for an Axis
RUN.VEL.{axis}=LREG#/Value	; Set Run/Max Speed for an Axis
INC.DIS.{axis}=LREG#/Value	; Set Incremental the Final Position
INDEX.{axis}=LREG#/Value	; Set Index Movement Distance
FIN.POS.{axis}=LREG#/DATA#/Value	; Set Desired Final Position for Axis
TIMER# = LREG#/Value	; Set Initial value for Software Timer

Examples:

ACC.RATE.A = LREG0	; Set Axis A's Accel Rate to LREG0
RUN.VEL.B = 2000	; Set the Run/Max Velocity as 2000
TIMER2 = 0	; Clear Timer #2

12.2 Assignment, Data Type Conversion and Calculation Commands

- **Variable Assignment**
  SREG# = SREG#/A2D#/IN.{port}/Value
  REG# = REG#/Value
  LREG# = LREG#/TIMER#/Value

- **Data Type Conversion**
  SREG# = IN.{Port} / OUT{Port}
  REG# = SREG#
  LREG# = SREG#/REG#

- **Calculation Commands**
  SREG# = SREG# Op. SREG#/Value
  REG# = REG# Op. REG#/Value
  LREG# = LREG# Op. LREG#/Value
  where **Op.** can be: +, -, * or /.

Symbol # is used in the above commands. For SREG, # is 0 to 63; for REG and LREG, # is 0 to 31; for A2D, # is 0 to 3; for TIMER, # is 0 to 7.

Examples:

SREG1 = A2D0
REG2 = 3000

```
LREG0 = SREG1
LREG0 = LREG1 / 500
```

12.3 Movement Control Commands

RESET.{axis}	; Reset Positions of one or more Axes to zero
POS.MOV.{axis}	; Position Move for one or more Axes
MOV.INDEX.{±axis}	; Index Move for one or more Axes
VEL.MOV.{±axis}	; Velocity Move for one or more Axes
HOME.{axis}	; Return to Home for one or more Axes
WAIT.STOP.{axis}	; Wait until the specified moving Axes have stopped.
NOR.STOP.{axis}	; Decelerate and Stop one or more Axes
EME.STOP. {axis}	; Immediately Stop one or more Axes

Examples:

RESET.ABCD	; Reset Axis A, B, C and D
MOV.IND.+C-D	; Index Move for Axis C and D in desired directions
NOR.STOP.B	; Decelerate and Stop Axis B

12.4 Toggle Control Commands

EN.TOG.{axis}	; Start Toggle movement for one or more Axes
DIS.TOG.{axis}	; Stop Toggle movement for one or more Axes
EN.TIN.{axis}	; Enable Toggle Trigger Input for one or more Axes
DIS.TIN.{axis}	; Disable Toggle Trigger Input for one or more Axes

Note:

Accel/Decel Rate, Start/Stop Speed, Run speed and Index Distance should be set using Parameter Assignment Commands before activating a Toggle Movement.

Examples:

EN.TOG.AB	; Start Toggle movement for Axis A and B
DIS.TIN.AD	; Disable Toggle trigger Input for Axis A and D

12.5 Output Control Commands

OUT.{port} = SREG#/Value	; Turn On/Off Output Port
OUT.{port}# = 0/1	; Turn On/Off Output Pin

Examples:

OUT.A = SREG7	;
OUT.B = 0FFH	;
OUT.C = 01010101B	;
OUT.D2=0	; Turn On pin D2 of I/O port D

12.6 Program Flow Control Commands

```
L#                               ; Define a Label, #: 0 to 31
GOTO L#                          ; Unconditional Jump within the Program
GOTO PROG#                       ; Unconditional Jump to another Program
CALL L#                          ; Call a Subroutine
RET                              ; End a Subroutine
END                              ; End a Program
IF var1 Op. var2 THEN L#         ; Conditional Statement & Jump commands
IF var1 Op. var2 THEN PROG#      ; Conditional Statement & Jump commands
```

Note: Op. is relation operator: = = (equal to) , != (not equal to), >=, >, <= and <.
var1 can be: SREG#, REG#, LREG#, TIMER#, A2D#, IN.{port}, IN.{port}#, MOV.STA.{axis}, OUT.{port}, OUT.{port}#, ACT.POS.{axis}, CUR.POS.{axis} and DATA.{axis}# . **var2** can be: any kind of op1 or proper type of value (decimal, binary, hex, or 0/1)

Examples:

```
L1                             ; Assign  Label #1
IF TIMER0 < = 100 THEN L2      ; If Timer 0 less or equal than 100, jump to L2
IF A2D2 > 10 THEN L4           ; If value of A2D2 is larger than 10, jump to L4
IF IN.A7 = = 0 THEN PROG5      ; If on-board Input pin A7 is low, jump to Prog#5
IF IN.A = = 55H THEN L20 ; If Port A = 55 Hex then goto L20
IF IN.A = = 01010101B THEN L25 ; If Port A = 01010101 Binary then goto L25
CALL L3                        ; Call a Subroutine L3
```

Chapter 13

Err #	Error Explanation	Solution
0	Normal status, No Error occurred	
1	Program Number already in use	Delete Program in Controller or renumber new program and reload
2	Insufficient Memory when down loading program	Delete unused programs
3	Syntax Error in Command line	Check the Command line
4	Data type does not match	Use the same type of data
5	Data value is out of range	Check the allowable data range
6	Integer or floating data format error	Use correct format in Command line
7	Binary or Hex data format error	Check the data format
8	Invalid Command used in On-Line mode	Check Command
9	Undefined Label is used in a program	Check program labels
10	The same program Label is defined twice.	Rename the Label.
11	Attempting to enter one mode while another mode is in use. Controller is busy.	Wait until Busy Signal (Yellow LED) is off.
12	Program Fails to Start	Check Program and reload
13	Software limit is reached	Check Software limits in System File
14	Error occurs during homing operation	Check Sensor Logic and Connections
15	Unexecutable program	Check Program and Re-load
16	Position compensation failure	Check Encoder Wiring, Motor Driver, & system load
17	Error occurs during manual operation	Check the Joystick Configuration in System Configuration file
18	ALARM Input is activated	Check the Alarm Source, Driver over heating.
19	System Configuration Error	Re-load System Configuration File
20	SRAM Memory Error	Call for Service and Repair
21	EEPROM Memory Error	Call for Service and Repair
22	Unexecutable program	Check Call & Return Commands as well as the Program and Re-load
23	Forward Limit Error	Check Program movement distances and system load
24	Reverse Limit Error	Check Program movement distances and system load

Table 13.1 Error Codes

Note:

1.Errors 1-10 are automatically cleared by the Controller after sending the Error Code to the host.

2.Errors 11-14 are automatically cleared during On-Line execution after the Error Code has been sent to the Host PC. If the Controller is not in the on-line mode, it will not automatically clear these errors (see note #4).

3.Errors 15-18 are not automatically cleared (see note #4).

4.When Errors are not automatically cleared, the ERROR LED(Red LED) turns on and the Controller stops the current task. When the ERROR LED is on, Port A's Pin 29 (ERROR Output) is closed. Use the $RD.ERR command to read the Error Code and then use the $RES.ERR command or reset the controller (cycle power or send an input to the RESET pin, Table 3.4) to clear the Error.

5.If you are designing your own communication package the Error Codes listed above will be sent to the Host PC in a single byte Hexadecimal Format for Communication method 1 (See Chapter 14).

Chapter 14

This chapter describes the Communication Protocols which you may use to develop Custom Communication Software for your Host Computer.

The Communication between a Host Computer and the MAC-300 Controller is done via the RS-232C communication port. The initialization of the serial port requires the following communication port parameters:

9600 baud rate; 8 data bits; no parity; 1 stop bit

There are two methods that can be used in order for the Host Computer to communicate with the MAC300. The first method uses the PC as a Smart Terminal while in the second method the Controller looks at the Interface device as a Dumb Terminal (i.e Handheld Terminal).

14 .1 Method 1 : Smart Terminal

In the first method the Host Computer is treated as a smart terminal which can convert hexadecimal and character information received from the Controller to a usable form. In this first method all of the Commands sent to the Controller must be in a **CAPITAL ASCII** character string format. The character strings sent to the MAC-300 must be single command lines as shown in the programming examples throughout this manual and the Commands listed in Table 14 .1. Following each character string, there should be a **true ASCII NULL (ASCII code = 0)** which terminates the character string. When the Command has been received correctly by the MAC-300, the Controller will then return another true ASCII Null (ASCII code = 0) back to the Host PC. If there is an Error in the Command line, then the MAC-300 will return an ASCII code/character that represents the Error Code instead of the true ASCII Null (refer to Chapter 13 , Table 13.1 , for the error code list). For example, if there is a syntax error in a given command line then the Controller will return an ASCII code/character = 3 instead of an ASCII Null.

The following examples require the first character string to be the Call Controller Command , @0 call Controller number zero for example , followed by the write/download command $WR.*** and the appropriate information , then to terminate the down loading operation the last command must be the END command.

Protocol Examples

Send the following character strings.

- **To Down Load the System Configuration File : Example**

@0	(plus an ASCII Null)	; Send Call Controller Command
$WR.SYS	(plus an ASCII Null)	; Start Write a System Configuration file.
SCA.FAC.A=1	(plus an ASCII Null)	; Set Scale Factor for Axis A = 1
CH.B=N	(plus an ASCII Null)	; Set Axis B to not used.
CH.C=M	(plus an ASCII Null)	; Set Axis C as a Motion Axis.
CH.D=N	(plus an ASCII Null)	; Set Axis D as not used.
JOY=0	(plus an ASCII Null)	; Set Joystick as an Analog type.
•		
etc.		
END	(plus an ASCII Null)	; End Write/DownLoad Command

- **To Down Load a Program File : Example**

@2	(plus an ASCII Null)	; Send Call Controller # 2 Command
$WR.PROG5	(plus an ASCII Null)	; Write/Download Program #5
RESET.A	(plus an ASCII Null)	; Reset Axis A to start at Position 0
INDEX.A=100	(plus an ASCII Null)	; Set Index Value for Driver A
MOV.IND.A	(plus an ASCII Null)	; Move Motor A 100 Steps
•		
etc.		
END	(plus an ASCII Null)	; End DownLoading Operation

- **To Down Load a Variable File : Example**

@1	(plus an ASCII Null)	; Call Controller #1 Command.
$WR.VAR	(plus an ASCII Null)	; Down load register value command
SREG6 = 110	(plus an ASCII Null)	; Set Short register #6 = 110
LREG12 = 12362	(plus an ASCII Null)	; Set Long register #12 = 12362
LREG15 = 52671	(plus an ASCII Null)	; Set Long register #15 = 52671
REG28 = 32123	(plus an ASCII Null)	; Set Normal register #28 = 32123
REG30 = 2142	(plus an ASCII Null)	; Set Normal register #30 = 214 2
END	(plus an ASCII Null)	; End Variable Down Loading

Protocol Examples continued

Send the following character strings.

- **To Down Load a Position Data File : Example**

```
@2                   ( plus an ASCII Null)     ; Call Controller #2
$WR.DATA             ( plus an ASCII Null)     ; Down load position data command
DATA.A5 = -1233      ( plus an ASCII Null)     ; A axis position data #5 = -1233
DATA.B10 = 301.2     ( plus an ASCII Null)     ; B axis position data #10 = 301.2
DATA.D50 = 12.321    ( plus an ASCII Null)     ; D axis position data #50 = 12.321
END                  ( plus an ASCII Null)     ; End Data Down Loading
```

The next example uses the ON.LINE mode and must also be terminated with an End command.

Note: In the On-Line Mode you will not be able to Read the Status of Storage Registers, Current Position etc. You must first exit the On-Line Mode before reading. In addition you can not use any IF..THEN , CALL or GOTO statements in the On-   Line Mode.

- **To Start the On-Line Mode of Controller #3 send following Strings**

```
@3                   ( plus an ASCII Null)     ; Send Call Controller #3 Command
$ON.LINE             ( plus an ASCII Null)     ; Start On-Line Mode

  .
INDEX.A=100          ( plus an ASCII Null)     ; Set Index Value for Driver A
MOV.INDEX.A          ( plus an ASCII Null)     ; Move Motor  A    100 Steps

  .
     etc.
END                  ( plus an ASCII Null)     ; End On-Line Mode
```

The remaining Commands available in the Protocol list do not require the END command in order to terminate the communication between the Host Computer and the Controller.

- **To Start a Program in Controller #2 send the following Strings**

```
@2                   ( plus an ASCII Null)     ; Call Controller #2
$GO.PROG5            ( plus an ASCII Null)     ; Start Program #5
```

- **To End a Program with Normal Deceleration send the following Strings**

```
@0                   ( plus an ASCII Null)     ; Call Controller #0
$NOR.STOP            ( plus an ASCII Null)     ; Stop Program that is running.
```

In order to read information from the internal registers of the MAC300 you must send the Call Controller command followed by the appropriate read command string. In the following example a read command , $RD.POS.A , is sent to the Controller which will read the current position of Axis A . Once the Controller has returned the true ASCII Null , an additional four bytes of hexadecimal information will be sent to the Host PC representing the position of Axis A. The order of the information returned will be first the Most Significant Byte followed by the remaining bytes. This information must be converted or processed by the Smart Terminal (Host PC) in order to be of any use.

To Read the Current Position of Controller #3 send the following Strings

```
@3                    ( plus an ASCII Null)        ; Call Controller #3
$RD.POS.A             ( plus an ASCII Null)        ; Read Current Position of Axis A
```

Note: When reading other registers refer to Table 14 .1 and Table 9.1 in order to determine the number of bytes of information return to the Host PC when a read command is used.

14 .2 Method 2 Dumb Terminal

In the second method the Host Computer is treated as a dumb terminal which is only required to display information received from the Controller. In this second method all of the Commands sent to the Controller must be in a **CAPITAL ASCII** character string format. The character strings sent to the MAC-300 must be single command lines as shown in the programming examples throughout this manual and the Commands listed in Table 14 .1. Following each character string, there should be a **Carriage Return (ASCII code = 13)** plus a **New Line** command **(ASCII code = 10)** which terminates the character string. When the Command has been received correctly by the MAC-300 , the Controller will then return a **true ASCII Null (ASCII code = 0)** back to the Host PC followed by a Carriage return and a New line command . If there is an Error in the Command line, then the MAC-300 will return an ASCII code/character that represents the Error Code(refer to Chapter 13 , Table 13.1 ,for the Error Code list) instead of the true ASCII Null. For example, if there is a syntax error in a given command line then the Controller will return an ASCII Code/Character = Chr$(3) instead of an ASCII Null.

Except for the Read Commands, the examples for Method 2 are the same as Method 1 however, as stated above, instead of an ASCII Null terminating a command line the dumb terminal must terminate a command line with a carriage return followed by a new line command.

- **Reading Information using a Dumb Terminal**

When reading information from the internal registers MAC300 you must send the Call Controller Command followed by the appropriate read command string. In the following example a read command , $RD.POS.A , is sent to the Controller which will read the current position of Axis A . Once the Controller has first returned the true ASCII Null , an additional unknown number of ASCII decimal bytes representing the position registers information will be sent to the Host PC followed by a Carriage Return and a New Line command. This information does not need to be converted or processed by the Dumb Terminal (Host PC) because it is already in a useful form. Although this is a simple method of receiving data, one disadvantage to this method is a slower processing speed. For example , when a large decimal number is returned to the PC the number of characters/bytes representing the actual value of this data will be greater for Method 2, thus with this method communication cycles will tend to take more PC processing time.

- **To Read the Current Position: Example**

```
@3          ( plus a Carriage Return & New Line command)      ; Call Controller #3
$RD.POS.A   ( plus a Carriage Return & New Line command)      ; Read Current
                                                                Position of Axis A
```

Protocol Command List

Command	Short form	Mode	Descriptions
#(0,1,2,...)		any	Assign a number to the Controller
@(0,1,2,...)		any	Controller CALL command
$DEL.ALL	$D.A	idle	Delete all control programs in the Controller
$DEL.PROG#	$D.P#	idle	Delete a program with number # = 0 ~ 63
$DIS.JOY	$D.J	any	Disable the joystick input
$EME.STOP	$E.S	any	Emergency stop for all axes and quit the running program if any
$EN.JOY	$E.J	idle	Enable the joystick input
$GO.PROG#	$G#	idle	(#=0 ~ 63) Start program #
$HALT	$HL	run	Halt Program Execution
$NOR.STOP	$N	any	Slow down stop with deceleration for all axes and quit the running program if any
$ON.LINE	$O	idle	Switch to on line mode
$RESUME	$RM	run	Resume Program Execution
$RD.A2D#	$R.A2#	any	(#=0~3) Read analog input of specified channel
$RD.COU.(axis)	$R.C.(axis)	any	Read encoder counter data of specified axis
$RD.CTR	$R.C	any	Read Controller status (1-byte, binary display)
$RD.DATA.(axis)#	$R.D.A#	any	Read Stored Position Data Point of specified axis
$RD.ERR	$R.E	any	Read error byte
$RD.FAC.(axis)	$R.F.(axis)	any	Read scaling factor of specified axis

$RD.IN.(port)	$R.I.(port)	any	Read input status of specified port
$RD.LREG#	$R.L#	any	Read a long register value (4-byte) (# = 0~31)
$RD.OUT.(port)	$R.O.(port)	any	Read output status of specified port
$RD.POS.(axis)	$R.P.(axis)	any	Read position data of specified axis
$RD.P#	$R.P#	any	Read the Current Program# , 1-byte , Hex value
$RD.REG#	$R.R#	any	Read a word register value (2-byte) (# = 0~31)
$RD.SREG#	$R.S#	any	Read a short register value (1-byte) (# = 0~63)
$RD.STA.(axis)	$R.ST.(axis)	any	Read sensor status of specified axis
$RD.TIMER#	$R.T#	any	(#=0~3) Read timer counter
$RES.ERR	$RE.E	any	Reset error
$SAVE.(axis)#	$S.(axis)#	idle teach	Save the current position of the specifed axis as the teaching position data (# = 0 ~63)
$UP.DATA.(axis)	$U.D.(axis)	idle	Header command for position-data up load
$UP.SYS	$U.S	idle	Header command for system data up load
$UP.VAR	$U.V	idle	Header command for variable up load
$WR.DATA	$W.D	idle	Header command for position-data down load
$WR.PROG#	$W.P#	idle	(#=0~63) Header command for program down load
$WR.SYS	$W.S	idle	Header command for system configuration
$WR.VAR	$W.V	idle	Header command for variable down load

Table 14 .1 Communication Protocol List

Note: When reading the Controller Status using a Dumb Terminal , a 8 bit Binary number is returned. The following Table lists the representation of each bit.

- **$RD.CTR**

This command returns an 8-Bit Binary Number. Each Bit refers to the Controller Status where 0 = Not Active and 1 = Active. The order from left to right is Bit 7, Bit 6,... ,Bit0.

Bit #	**Description**
7	Program Run Status
6	Manual Mode Status
5	Manual Input Status
4	Controller Error Status
3	Axis D Motion Status
2	Axis C Motion Status
1	Axis B Motion Status
0	Axis A Motion Status

Note: When you must know the number of the active Running Program you must use the status of Bit #7 listed above with the **$RD . P#** command to determine the Program Number.

14.3 Communication Software Examples

The following Communication Example for Down Loading Commands to the MAC-300 Controller was written with Visual Basic 4.0 for the **Windows** environment. In this example the MAC-300's Smart Terminal mode of communications is used.

```
Sub Down_Load(OutputCommand$)
Dim Duration!, CommBuffer%
Dim DataString$, DataValue$
'************************************************************
'Activate the Serial Communications
On Error Resume Next
Comm.CommPort = 1             ' Use Corn. Port 1 .... Change when Necessary
If Err Then
  Msgbox "Communication Port not available", 48
  Exit Sub
EndIf
Comm.Settings = "9600,N,8,1"   ' 9600 baud, no parity, 8 data bits, and I stop bit
Comm.Handshaking = 0          ' Set Communication Handshaking to Off.
Comm.InputLen = 0             ' Tell the PC to read All of bytes in the Input Buffer.
Comm.NullDiscard = False      ' Tell the PC to also Read Null Characters in the Input Buffer.
Comm.OutBufferSize = 32       ' Set Maximum Require Buffer Sizes
Comm.InBufferSize = 32
Comm.RTSEnable   False        ' Disable RTS and DTR Signals
Comm.DTREnable   False
'************************************************************
'************************************************************
Comm.PortOpen = True          ' Open the port, ' Open the Communication Port
'************************************************************
'************************************************************
If OutputCommand$ = "" Then Exit Sub          ' Make Sure Command has some length.
'************************************************************
'************************************************************
OutputCommand$ = UCase(OutputCommand$)       ' Insure that Command is Upper Case
'************************************************************
'************************************************************
Comm.InBufferCount = 0                       ' Flush the Input Buffer
Comm.OutBufferCount = 0                      ' Clear the Output Buffer
'************************************************************
'************************************************************
' Send Command to the MAC-300 using the Smart Mode: Method 1, Command + Null
Comm.Output = OutputCommand$ + Chr$(0)
'************************************************************
'************************************************************
  ' Important: Insure that the serial port output buffer is empty before proceeding
  Duration!  Timer + 0.5
     Do
       CommBuffer% = Comm.OutBufferCount
      If (Timer > Duration!) Then
       GoTo DownLoad_exit
      EndIf
     Loop Until CommBuffer% = 0
'************************************************************
```

Visual Basic 4.0 Software Example Continued

```
'*************************************************************
' Wait for the Data to come back ... If the Null is not the first character returned
' then there is an Error.
' Note: Delete Program, Wait, Home, Normal Stop and End Commands should have
' a wait time(Duration) as long as 20 seconds.

    Duration!  Timer + 0.5
    Do
        If Timer > Duration! Then
            ' Set Communication Error: 25
            ' Check_Data%= 25
            ' Go to Error Code Handier
            Call ErrorCode_Handier
            GoTo DownLoad_exit
        EndIf
    Loop Until Comm.InBufferCount >= 1

'*************************************************************
'*************************************************************
' Read the Null/Error Code Response in the serial port.
 DataString$ = Comm.Input
'*************************************************************
'*************************************************************
' Only Look at the Left most Hex value
 DataValue$ = Left$(DataString$, 1)

' If theres an Error then Check Data = ErrorCode
Check_ Data% = Asc(DataValue$)
'If there is no Null(Chr$=0) in the First Character then flag an Error Message
If Check_Data% > 0 Then
      Call ErrorCode_Handler
      GoTo DownLoad_exit
EndIf
'*************************************************************
DownLoad_exit:
'*************************************************************
'*************************************************************
Comm.PortOpen = False            ' Close the Communication Port
'*************************************************************

End Sub
```

The following Communication Example for Down Loading Commands to the MAC-300 Controller was written with ANSI C code for the DOS™ environment. In this example the MAC-300's Smart Terminal mode of communications is used.

```
#include "stdio.h"
#include "dos.h"
#include "ctype.h"
#include "stdlib.h"

/* Serial Port Communications Control Status
#define STBMT  0x20                            /* Mask for Transmitter Empty
#define SDAV    1                              /* Mask for Character Available
#defuie BUSYR  0x10                            /* External Device is Busy
 #define READY  0x20                           /* External Data not Ready

static int modctlr;                            /* Modem Control Registers
static int remdata;
static int remstat;
static int modstat;

int stopbit[ ] =  {0,4};                       /* One Stop Bit, Two Stop Bits
int parity[ ]= { 0,8,24};                      /* No Parity, Odd, Even Parity
int baud[ ] ={0,32,64,96,128,160,192,224};     /* Baud Rates
int databit[ ] ={2,3};                         /* Seven Data Bits, Eight Data bits

main()
{
unsigned char code;
union cc
        {
        char ch[2];
        int i;
        } ccc;

int     i, j, k, line_num;
char    end_cmd, string[30];

remdata   0x3f8;                               /* Comm Port 1 Address
remstat   0x3fd;                               /* Line Status Register
modctlr   0x3fc;                               /* Modem Control Register
modstat   0x3fe;                               /* Modem Status Register
code=baud[7]+parity[0]+stopbit[0l+databit[1]
int_port(0,code);
clear_port();

/* Line Numbers are Printed before commands are sent to the Controller*/
 line_num=0;
printf("Enter Commands ( Character Strings for the MAC-300 ) \n");
```

ANSI C code Software Example Continued

```
for(;;)
        {
        printf("%d: ", line_num);

        for(i=0;;)
{
/* Save the Characters that are Typed in the String
        while(!kbhit());
        string[i] = getch();
        printf("%c", string[i]);

/* When the User hits the Enter Key get out of this For Loop and send Commands to the
Controller */
                if (string[i] == 13)
                break;
                i++;
}

/* Add the Null to the end of the String
        string[i] = 00;

/* Determine the Length of the String
        k = strlen(string);
        k++;

/* Now, Send each Character to the MAC300 Controller
        for(j=0 ; j<k; j++)
        {
        while( ( inp(remstat) & STBMT)== 0)
         if( kbhit() ) exit(0);
        outp(remdata, string[j]);
        }
while((inp(remstat) & SDAV) == 0)
                if(kbhit())    exit(0);
                end_cmd=inp(remdata);

if(end_cmd!=0)
 {
printf("Command Error!  Err_Code=%d, Try Again.........", end_cmd);
line_num--;
  }
line_num++;
printf("\.n");

  }
}
```

ANSI C code Software Example Continued

```
/*          Comm Port Initialization Subroutines

clear_port()
   {
        if((inp(remstat) & SDAV) != 0)        /* Check for a Character
        inp(remdata);                         /* Read the Character
        return(0);
   }

int_port(int port, unsigned char code)
   {
        union REGS r;                         /* Serial Port Number
        r.x.dx = port;                        /* Initialize Port Function
        r.h.ah = 0;                           /* Initialization Code for Baud ... etc.
        r.h.al = code;
        int86(0x14,&r,&r);
        return(0);
   }
```

Chapter 15

After the System Configuration file has been down loaded to memory, the Controller is ready to accept Manual operation requests and externally Start/Stop Program Inputs.

15.1 External Program Start and Stop Functions

Figure 15.1 shows how to connect the Store Position# & Program Number Inputs #0 - #6 and the START input. To Start a Program externally, first set the Program number through the Store Position/Program# Input pins #0 ~ #6. The number setting is in 7 bit BCD data format representing the numbers 0 ~ 63. Table 15.1 lists the BCD codes for the decimal data from 0 to 9. Input pins #0 ~ #3 represent lower digit of the decimal data (0-9), while input pins #4 ~ #6 represent upper digit of the decimal data (0-6 x10). For example, if the input is 010 1001 (in the order of #6 #5 #4 #3 #2 #1 #0), the program number is 2x10 + 9 = 29. Here, 1 represents a contact closure or a conduction path between the input pin and the IO.COM and 0 represents no contact or conduction path between the input pin and the IO.COM. For PLC users, IO.COM must be linked to the PLC common and the Store Position#/Program# Input pins #0 - #6 must be linked to the PLC output pins.

BCD Code	Decimal Data
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001	9
1010~1111	Invalid

Table 15.1 List of BCD Codes

After the Controller is in Idle mode (Not in running a Program or in the Manual or On-Line modes) and a valid Program number (0~63) is set, a contact closure between the START pin and the IO.COM pin will execute the specified Program. If an invalid Program number is set, the Controller will ignore the START input.

When the Controller is in the Execution mode, a contact closure between the ESTOP pin and the IO.COM pin will Stop the Program and immediately stop all Axes of Motion. The Controller has to be reset before it can run a Program again. To reset the Controller, you can cycle the +5VDC power input to the Controller or provide a momentary contact closure between the RESET input and GND.

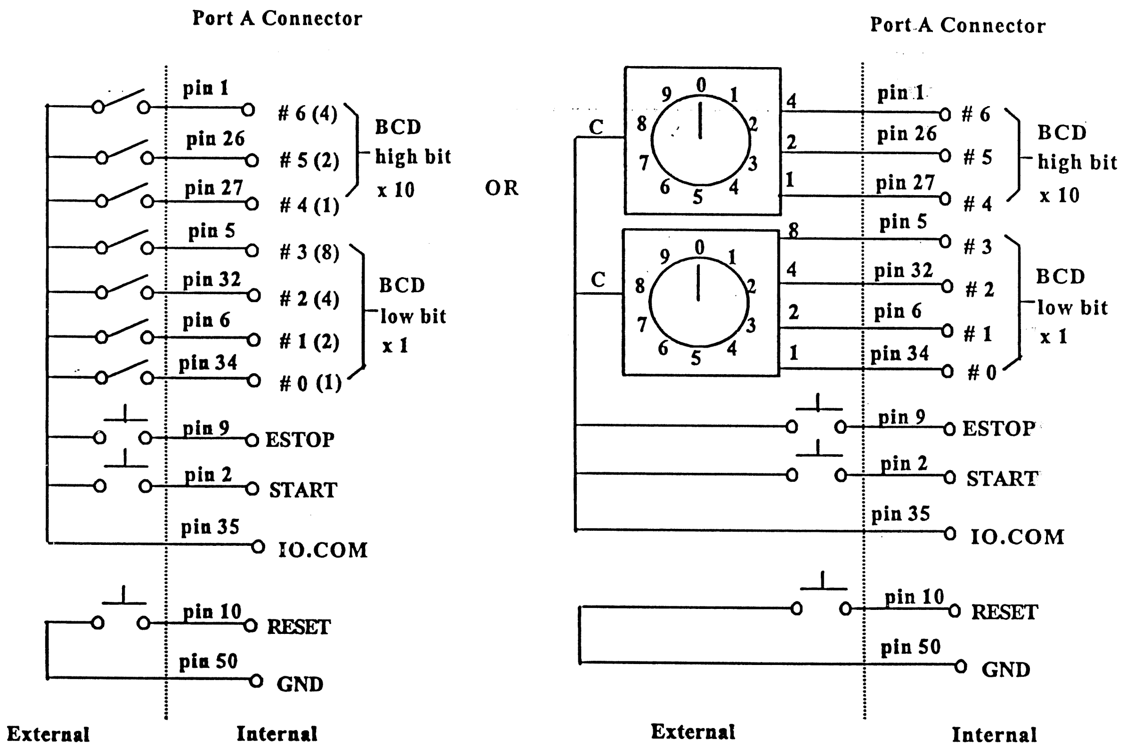


Fig 15.1 External Start/Stop Connection

15.2 Manual / Teach Operations

The Manual / Teach mode is used for two operations:

1. Manually control the Motors Motion,
2. Store (Teach) Position Data points.

In the Manual / Teach mode, one or more Axes can be controlled using the Joystick or the Jog Inputs (Manual operation). Also in this mode, Position Data points can be stored to the Non-Volatile memory (Teach operation).

The Joystick can be either an Analog Joystick or a Digital contact Joystick (push buttons). The Analog Joystick can be a Single Axis Joystick or a Multi-Axis Joystick. The Joystick type is defined in the System Configuration file(Sec. 6.7).

15.2.1 Using a Digital Contact Joystick

If a Digital Contact Joystick is used, in the Manual / Teach mode, only one Axis at a time can be controlled using the Joystick or Jog Inputs. In System Configuration file, make sure to use the command JOY=1 to set the joystick type to Digital contact Joystick (sec 6.7). Figure 15.2 shows the Digital Joystick connections.

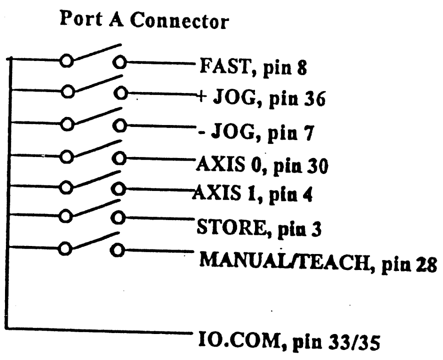


Fig. 15.2 Digital Joystick Connection

Step 1: To Activate the Manual / Teach Mode

When the Controller is in Idle mode, a closure between the MANUAL/TEACH pin and IO.COM will activate the Manual / Teach mode. When this mode is activated the BUSY LED(Yellow LED) will go on. To deactivate the Manual / Teach mode, disconnect the MANUAL/TEACH Input from IO.COM. The motor(s) will immediately stop, BUSY LED will go off and the Controller will be back to Idle mode.

The Manual / Teach mode can be also activated and deactivated through RS-232C communication commands. If you use MYEDIT.EXE software, under F3-Command menu, select Enable Joystick and Disable Joystick will activate and deactivate the Manual / Teach mode respectively (Sec. 5.5). If you develop your own communication software, please refer to Chapter 14 for the protocols to activate and deactivate the Manual / Teach mode.

Step 2: To Specify the Axis , Axis Number

In the Manual / Teach operation(Digital Joystick), one Axis can be moved at a time. The selection of the Axis number is done by setting the AXIS1 and AXIS0 inputs. This is a 2-Bit Binary data input (00 - 11) which correlates to Axis A - D (See Table 15.1).

AXIS1	AXIS0	Axis of Motion
0	0	Axis A
0	1	Axis B
1	0	Axis C
1	1	Axis D

Table 15.1 Specify Axis

A value of 0 means no contact or conduction between the AXIS input pin and the IO.COM. A value of 1 means there is contact or conduction between the AXIS input pin and the IO.COM. Switching the Axis number can be done at any time during the Manual / Teach operation. When the Axis number is changed during the Manual / Teach operation, the Controller will stop the last selected Axis and then begin the Manual / Teach operation on the currently selected Axis.

Step 3: To Use the Manual (Digital Joystick) Operation
The three Inputs **FAST, +JOG, -JOG** function in the following way:

- The selected Axis will move one step in the forward direction if the **+JOG** is activated or one step in the reverse direction if the **-JOG** is activated.

- If the **+JOG (or -JOG) and FAST** are activated at the same time, the selected Axis will move at the Default Maximum speed (Sec. 6.9) with the Default Acceleration and Deceleration Rates.

Step 4: To Use the Teach Operation
When the active Axis is in the desired position, you can Store the Position Data point into the memory by completing the following steps.

- Set the Teach Position data point number and Axis Number.
  The Teach Position data number Inputs are set in the same manner as the Program number Inputs from 0 to 63 (See Sec. 15.1) while the Axis Number is selected as described in the section above.

- Activate the STORE input by momentarily connecting the STORE pin with the IO.COM pin. The BUSY LED(Yellow LED) will switch off and then on to indicate that the Position has been successfully stored.

If the Encoder Feedback is not Enabled, the Stored position data will be the **Controller Output Pulse Counter value, the Current Position of the selected Axis.** If Encoder Feedback of the active Axis is Enabled, the stored position data point will be the **Encoder Feedback data value, the Actual Position of the Encoder (ACT.POS.(axis)).**

15.2.2 Using a Single Axis Analog Joystick

If a single Axis Analog Joystick is used in the Manual / Teach mode, only one Axis at a time can be controlled using the Joystick or Jog Inputs. In the System Configuration file, make sure to use the commands **JOY=0** and **AJOY=W** to set the Joystick type to single Axis Analog Joystick and use command **THR** = **#** to set a Threshold value for the Analog Joystick (sec 6.7). Figure 15.3 shows the connection for single Axis analog joystick.

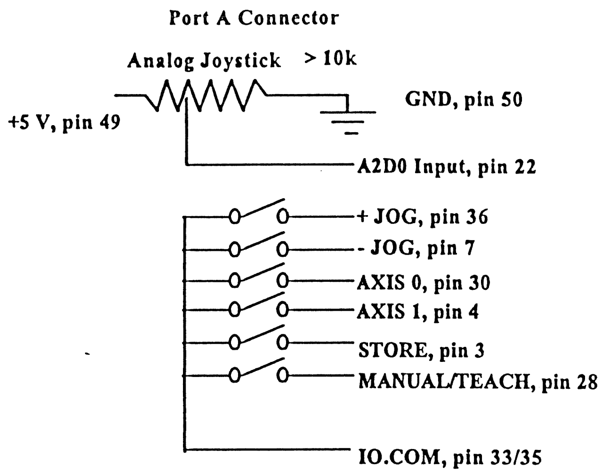


Fig. 15.3 Connection for Single Axis Analog Joystick

Step 1: To Activate the Manual / Teach Mode

When the Controller is in Idle mode, a closure between the MANUAL/TEACH pin and IO.COM will activate the Manual / Teach mode. Serial port communication commands can activate the Manual / Teach mode too(the same as Sec. 15.2.1 Step 1).

Step 2: To Specify the Axis

In the Manual / Teach operation, one Axis can be moved at a time. The selection of the Axis number is done by setting the AXIS1 and AXIS0 inputs (see Sec. 15.2.1 Step 2).

Step 3: To Use the Manual (Joystick) Operation

The Input Voltage to the A2D0 pin of the Controller controls the movement speed of the selected axis. The selected Axis will move in the forward direction if the Input Voltage is greater than 2.5V plus the threshold voltage. The Axis will move in the reverse direction if the Input voltage is less than 2.5V minus the threshold voltage. When the Input voltage is between 2.5V minus the threshold voltage and 2.5V plus the threshold voltage, the Jog function is Enabled. In this case, if +JOG is activated, the selected Axis will move one step in the forward direction; if -JOG is activated, the selected Axis will move one step in the reverse direction. If both +JOG and -JOG are activated at the same time both Inputs ignored and neither will be activated and the Motion Axis will stand still.

Step 4: To Use the Teach Operation

When the active Axis is in the desired position, you can set the Teach Position Data point number and activate the STORE input to store the position data into the Non-Volatile memory (see Sec. 15.2.1 Step 4).

15.2.3 Using a Multi-Axis Analog Joystick

When a Multi-Axis Analog Joystick is used, in the Manual / Teach mode, you can control Multi-Axes of Motions simultaneously using the Joystick or Jog inputs. In the System Configuration file, make sure to use the commands **JOY=0** and **AJOY={A}{B}{C}{D}** to define the Joystick type to Multi-Axis Analog Joystick and specify the controlled Axes for the Analog Joystick. Also use command **THR = #** to set a threshold value for the Analog Joystick and use command **IN.JOG = 0/1/2** to select the JOG operation mode (sec 6.7).
Figure 15.4 shows the connection for multi-Axis analog joystick.

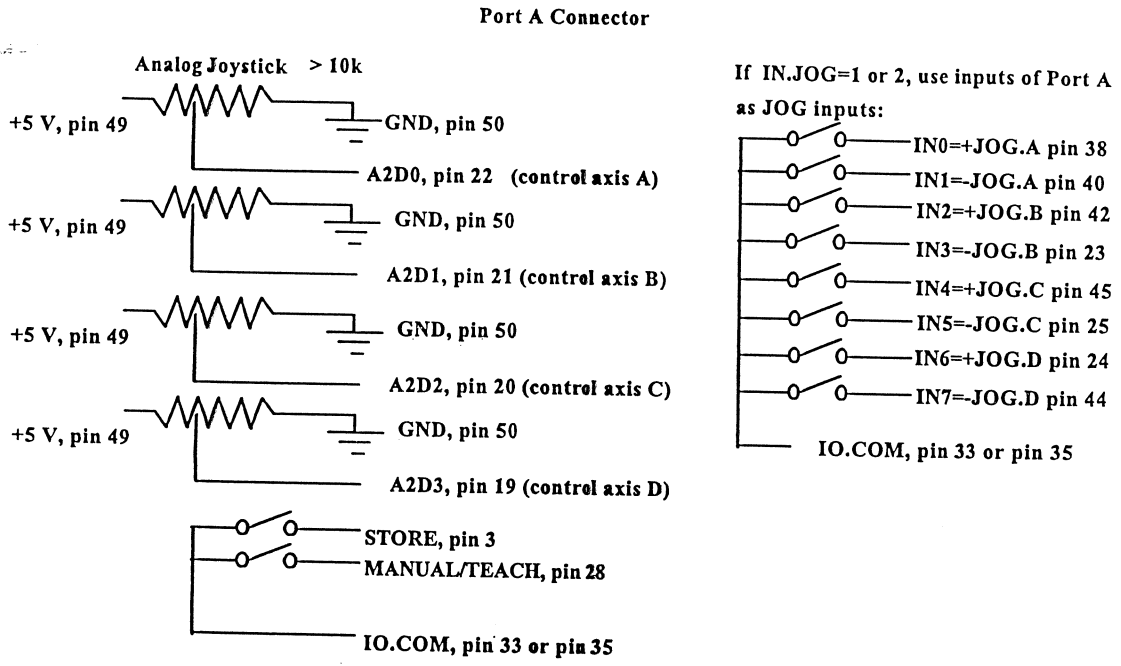


Fig. 15.4 Connection for Multi-Axis Analog Joystick

Step 1: To Activate the Manual / Teach Mode

When the Controller is in idle mode, a closure between the MANUAL/TEACH pin and IO.COM will activate the Manual / Teach mode. Serial port communication commands can activate the Manual / Teach mode too(the same as Sec. 15.2.1 Step 1).

Step 2: To Use the Manual (Joystick) Operation

The input voltage to A2D0, 1, 2 and 3 controls the movement speed of the Axis A, B, C and D respectively. The relationship between the movement speed of an Axis and its related analog input voltage is the same as that for single Axis analog joystick (Sec. 6.7). An Axis will move in the forward direction if its related input voltage is greater than 2.5V plus the threshold voltage. An Axis will move in the backward direction if its related input voltage is less than 2.5V minus the threshold voltage. You can control multi-Axis of motion simultaneously and independently through the multi-Axis analog joystick.

If the command IN.JOG = 0 is used in the System Configuration file, then there are no Jog functions for the Analog Joystick. If the command IN.JOG = 1 or 2 is used, then when all Analog Input voltages are between 2.5V minus the threshold voltage and 2.5V plus the threshold voltage, the Jog functions are activated. Since each Axis has its own dedicated +JOG and -JOG inputs, two or more axes can be Jogged simultaneously. By setting IN.JOG =1 or 2 in System Configuration file, you can select the Jog function as one pulse output at a time or continuos pulse train output at an one pulse per second frequency (Sec 6.7).

Step 3: To Use the Teach Operation

When all Axes are in the desired position, you can set the Teach Position Data point number and activate the STORE input to Store the Position data points into the Non-Volatile memory (see Sec. 15.2.1 Step 4). The Controller will store the Position Data points for All Axes at the same time with the same Position Data Number.

A.1 MAC-300 Main Card

Figure A.1 is a 1:1 drawing of MAC-300 main card with dimensions and jumpers' positions.

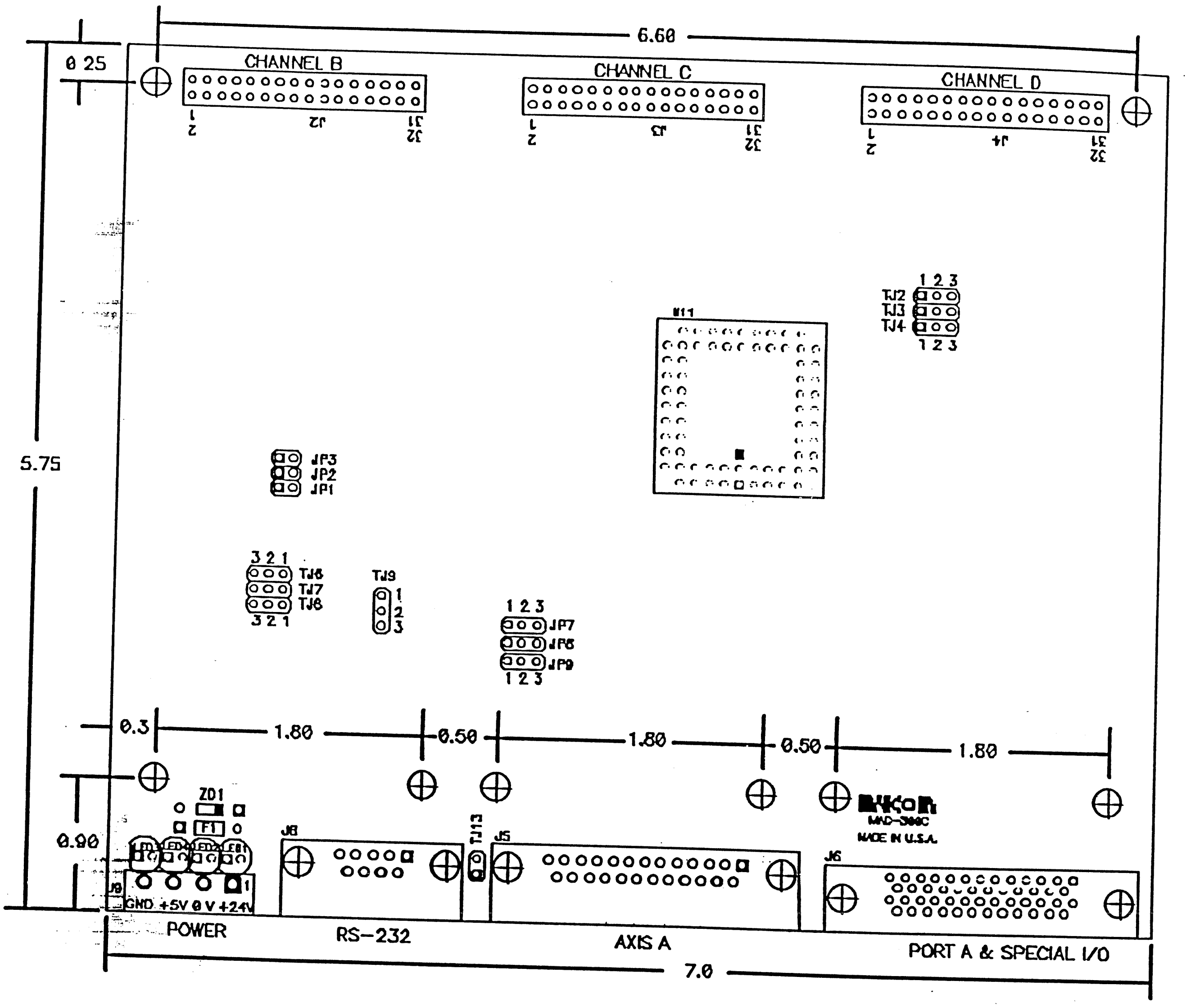


Fig. A.1 MAC-300 Main Card

TJ2-4: Manual Inputs / Analog to Digital Inputs for the Main Card

Three Jumpers are used to define whether the Digital Joystick is enabled or if the Analog A/D Inputs are enabled for general purpose use or for Analog Joysticks. When the three jumpers are in position 2-3, the manual operation functions (FAST, +JOG and -JOG) are enabled for Digital Joysticks. When these jumpers are put in position 1-2, the A/D pins are enabled for general purpose use or for Analog Joysticks . The relationship between the jumper locations and the activated functions are shown in Table A.1.

Note 1: All three jumpers must be in position 1-2 or 2-3 (**Default Setup**).

Note 2:When changing the Jumpers for the Joystick Type the System Configuration file of the MAC-300 should be set accordingly (See Chapter 6).

Jumper #	Pin 1-2	Pin 2-3
TJ2	A2D1	FAST
TJ3	A2D2	+JOG
TJ4	A2D3	-JOG

Table A.1 Jumper Location vs. Activated Function

TJ6-9: Limit Sensor Logic for the Main Card , Axis A

These Jumpers are used to select the Sensor Logic. When a Jumper is set on position 2-3, Negative Logic(Normaly Open) is selected (**Default Setup**). Negative Logic means that when the Sensor is active there is contact closure to the IO.COM (the IO supply ground) and this activates the Controllers Sensor Input. When a Jumper is set on position 1-2, Positive Logic(Normaly Closed) is selected. Positive Logic means that when the Sensor is active there is no contact closure to IO.COM and this activates the Sensor input. The Logic definition for each Sensor is independent. The relationship between Jumpers and Sensor Inputs are listed in Table A.2. See Table 3.3 for Sensor Input pin assignments.

Jumper #	Sensor Input Name	Pin 1-2	Pin 2-3
TJ6	NEAR.HOME	Normally Closed	Normally Open
TJ7	FOR.LIMIT	Normally Closed	Normally Open
TJ8	REV.LIMIT	Normally Closed	Normally Open
TJ9	Z-Phase Input	Active Low	Active High

Table A.2 Sensor Input Jumper Assignments

JP1 - JP3: Motor / Driver Variables for the Main Card , Axis A

These two & three pin jumpers are used to select various features of the motion setup including; type of Motor, Forward Direction of the Motor, type of Driver, and type of Encoder.

Default Setup is JP1 ON (shorted), JP2 & JP3 OFF (opened)

Jumper #	Contents	ON	OFF
JP1	Motor Type	Stepping	N.C
JP2	Forward Direction	CCW	CW
JP3	Clock Type	Bi-Clock(CW/CCW)	Gate (CLK/DIR)

Jumper #	Contents , Rev. C	Pins 1-2	Pins 2-3
JP7	Z-Phase Input Type	TTL(+Z) See Note 3	Differential (+Z, -Z)
JP8	A-Phase Input Type	TTL(+A) See Note 1	Differential (+A, -A)
JP9	B-Phase Input Type	TTL(+B) See Note 2	Differential (+B, -B)

Table A.3 Motor / Driver Variables

Note to TTL Encoder users for the Main Cards TTL modification :

The latest Jumper settings for MAC-300 Revision C (MAC-300C) are listed in the table above. If you are using a MAC-300 Revision B(MAC-300B) the following information applies if it is required to set the MAC-300 controller to recognize TTL Encoders.

If the Factory has not setup your Controller to except TTL Encoder Inputs (Default Configuration is Phase Differential) the following modification must be made. Although newer MAC-300 versions have IC sockets for easier modifications, if you want the Factory to make the modification call 408-232-7700 (San Jose , Ca.)

Note A: Remove Jumpers JP4-JP6

Note 1: With U28 removed, the AM26LS32 line driver chip, short pins 5&6 together.

Note 2: With U28 removed, the AM26LS32 line driver chip, short pins 10&11 together.

Note 3: With U28 removed, the AM26LS32 line driver chip, short pins 2&3 together.

A.2 MAC-310 Motion Axis Card

Figure A.2 is a 1:1 drawing of MAC-310 Motion Axis card with dimensions and jumpers' positions.

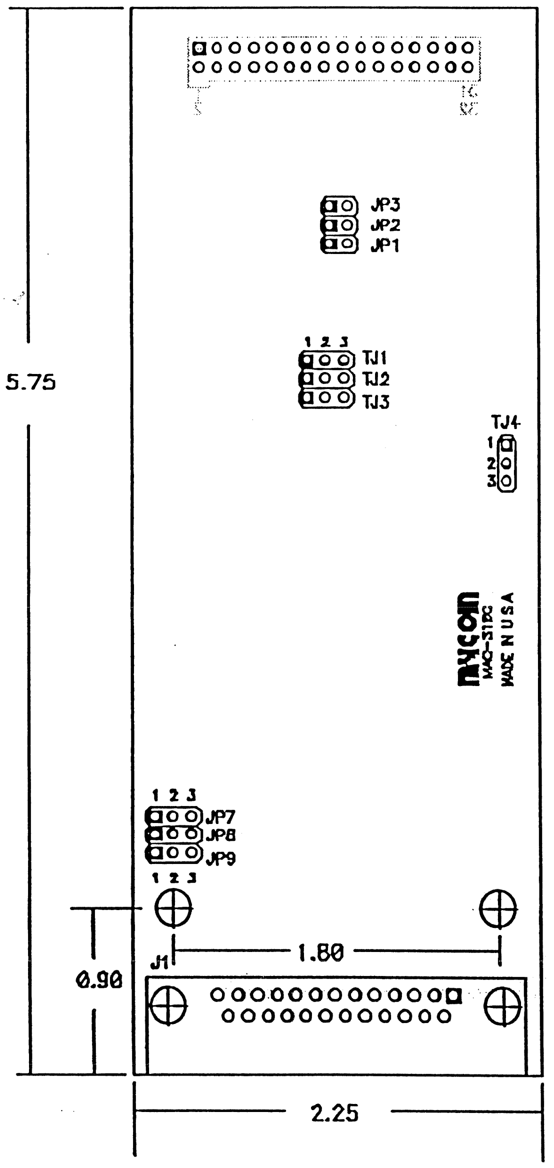


Fig. A.2 MAC-310 Motion Axis Card

TJ1-4 Limit Sensor Logic for the Motion Card , MAC-310

These jumpers act in the same manner as TJ6-TJ9 for the MAC-300 main card.

Default Setup : Negative Logic (Normally Open)

Jumper #	Sensor Input Name	Pin 1-2	Pin 2-3
TJ1	NEAR.HOME	Normally Closed	Normally Open
TJ2	FOR.LIMIT	Normally Closed	Normally Open
TJ3	REV.LIMIT	Normally Closed	Normally Open
TJ4	Z-Phase Input	Active Low	Active High

Table A.4 Sensor Input Jumper Assignments

JP1 - JP3: Motor / Driver Variables for the Motion Card , MAC-310

These jumpers are identical to those on the MAC-300 main card.

Default Setting is JP1 ON (shorted), JP2 & JP3 OFF (opened)

Jumper #	Contents	ON	OFF
JP1	Motor Type	Stepping	N.C
JP2	Forward Direction	CCW	CW
JP3	Clock Type	Bi-Clock(CW/CCW)	Gate (CLK/DIR)

Jumper #	**Contents , Rev. C**	Pins 1-2	Pins 2-3
JP7	Z-Phase Input Type	TTL(+Z) See Note 3	Differential (+Z, -Z)
JP8	A-Phase Input Type	TTL(+A) See Note 1	Differential (+A, -A)
JP9	B-Phase Input Type	TTL(+B) See Note 2	Differential (+B, -B)

Table A.5 Motor / Driver Variables

Note to TTL Encoder users for the Main Cards TTL modification :

The latest Jumper settings for MAC-310 Revision C (MAC-310C) are listed in the table above. If you are using a MAC-310 Revision B(MAC-310B) the following information applies if it is required to set the MAC-310 controller to recognize TTL Encoders.

If the Factory has not setup your Controller to except TTL Encoder Inputs (Default Configuration is Phase Differential) the following modification must be made. Although newer MAC-310 versions have IC sockets for easier modifications, if you want the Factory to make the modification call 408-232-7700 (San Jose , Ca.)

Note A: Remove Jumpers JP4-JP6

Note 1: With U28 removed, the AM26LS32 line driver chip, short pins 5&6 together.

Note 2: With U28 removed, the AM26LS32 line driver chip, short pins 10&11 together.

Note 3: With U28 removed, the AM26LS32 line driver chip, short pins 2&3 together.

A.3 MAC-320 Input/Output Port Card

Figure A.3 is a 1:1 drawing of MAC-320 I/O Port card with dimensions. There are no jumpers on MAC-320 I/O Port card.

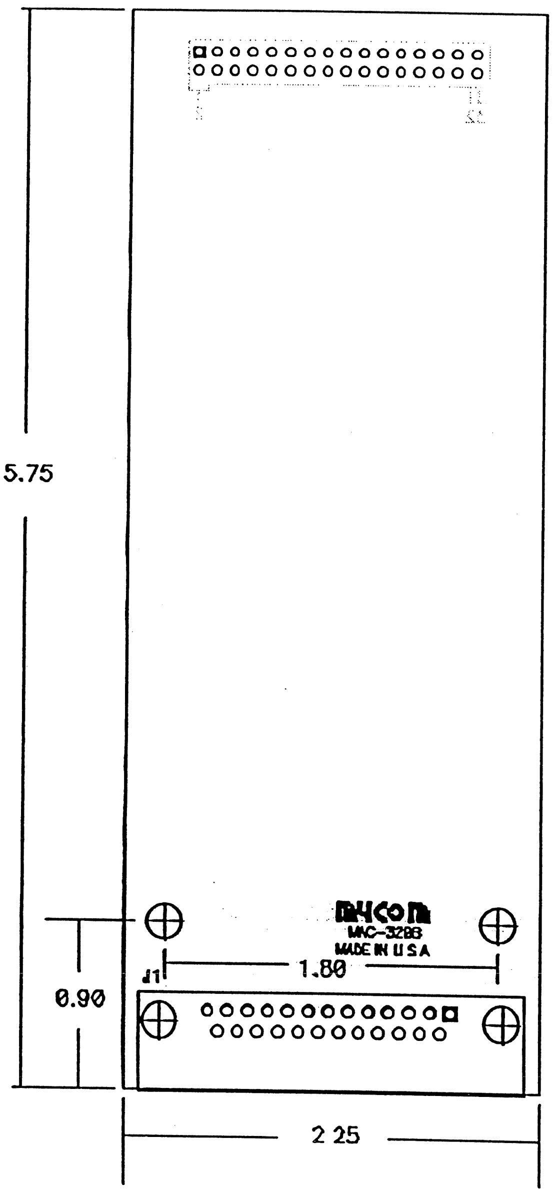
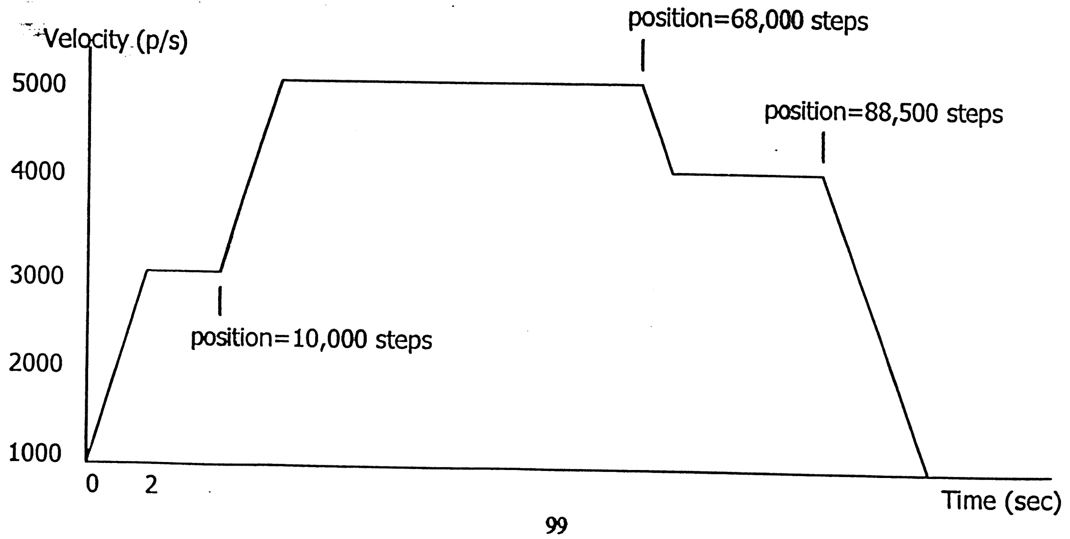


Fig. A.3 MAC320 I/O Port Card

Example 1

This Program produces a Multi-Level Velocity profile. Velocity changes are initiated based upon the present position of the motor. This Program is for Axis A and Controller #0 where the Scaling Factor of Axis A is 1, i.e. the Position unit is Pulse Counts.
Note: Scaling Factors are preset in the System Configuration File (Ch.6).

Program line	Comments
@0	; Call Controller #0.
$WR.PROG1	; Write Program Number 1 to memory.
ACC.RATE.A=1000	; Set Accel Rate to 1000 pulses/sec^2.
DEC.RATE.A=1000	; Set Decel Rate to 1000 pulses/sec^2.
MIN.VEL.A=1000	; Set Start/Stop Velocity to 1000 pulses/sec.
RUN.VEL.A=3000	; Set Run/Max. Speed to 3000 pulse/sec.
RESET.A	; Set Current Position to zero.
VEL.MOV.A	; Start to Accelerate to 3000 pulse/sec.
RUN.VEL.A=5000	; Define a new Run Speed as 5000 pulse/sec.
L0	; Define Label #0.
IF CUR.POS.A<10000 THEN L0	; Loop until Current Position = 10,000 steps.
VEL.MOV.A	; Accelerate to 5000 pulses/sec.
RUN.VEL.A=4000	; Define new Run speed as 4000 pulse/sec.
L1	; Define Label #1.
IF CUR.POS.A<68000 THEN L1	; Loop until 68,000 steps.
VEL.MOV.A	; Decelerate to 4000 pulses/sec.
L2	; Define Label #2
IF CUR.POS.A<88500 THEN L2	; Loop until 88,500 steps
NOR.STOP.A	; Decelerate and Stop
END	; End Program and return to the Idle mode.

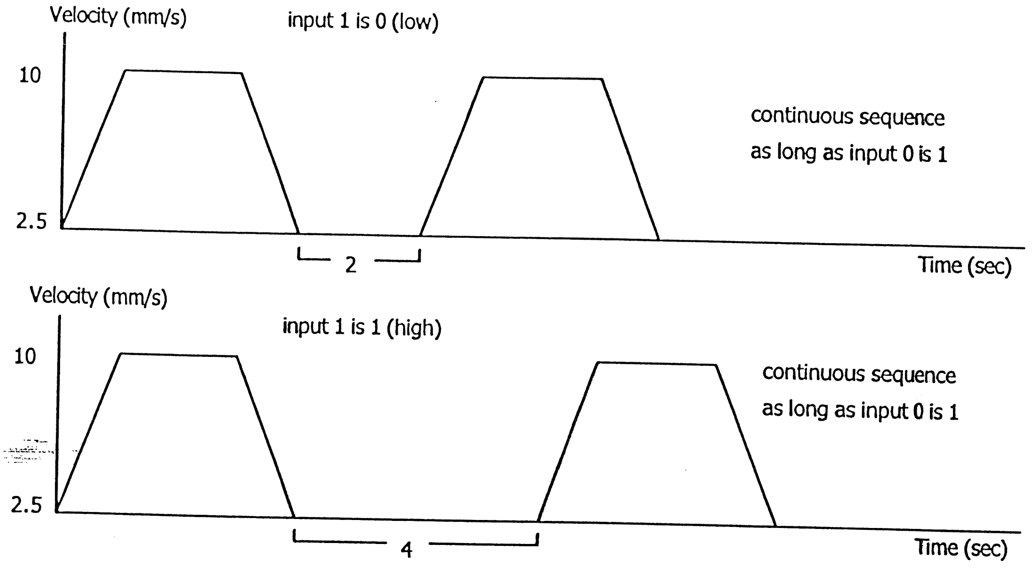


Example 2

Here, the IF...THEN command is used to Test Discrete Inputs and take action depending upon the state of the Inputs. The Jumps in the Program depend upon the state of Input A1 on the Main Controller card. Input A0 is used to control the termination of this Program. This Program also shows the application using Scaling Factor other than 1.

This Program is for axis C of Controller #1. The Position unit is in millimeters(mm).
A motor movement of 400 steps is equivalent to 1 mm of movement of the final Load. Therefore, the Scaling Factor has been set in the System Configuration file to 400 for Axis C.

Program line	Comments
@1	; Call Controller #1 on Daisy chain.
$WR.PROG2	; Write Program #2 to memory.
ACC.RATE.C=7.5	; Set Accel Rate to 7.5mm/s^2 = 3000p/s^2
DEC.RATE.C=7.5	; Set Decel Rate to 7.5mm/s^2 = 3000p/s^2
MIN.VEL.C=2.5	; Set Start/Stop Speed to 2.5 mm/s =1000p/s
RUN.VEL.C=10	; Set Run/Max. Speed to 10 mm/s = 4000p/s
INDEX.C=32.5	; Set Index Distance to 32.5 mm = 13000 p
L0	; Assign Label #0.
IF IN.A0 = = 0 THEN L3	; Check Input A0, if it is ON then exit.
MOV.INDEX.C	; Start Index Move
WAIT.STOP.C	; Wait until the above move is complete.
TIMER0=0	; Set Software Timer #0 to zero.
IF IN.A1 = = 1 THEN L2	; Check Input A1, if it is OFF then go to L2
L1	; Assign Label #1
IF TIMER0 <200 THEN L1	; Loop for 200 x 10ms = 2 sec
GOTO L0	; Go back to Label #0
L2	; Assign Label #2
IF TIMER0< 400 THEN L2	; Loop for 400 x 10ms = 4 sec
GOTO L0	; go back to L0
L3	; Assign Label #3
END	; End Program and return to Idle mode.



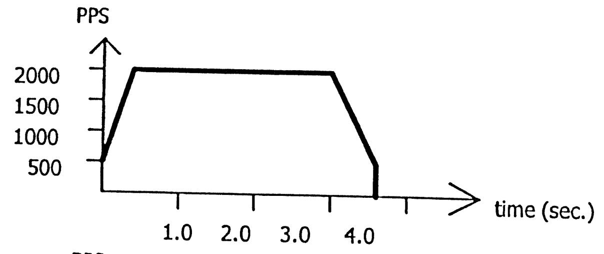
Running sequence for Program #2 based on Inputs A0 and A1

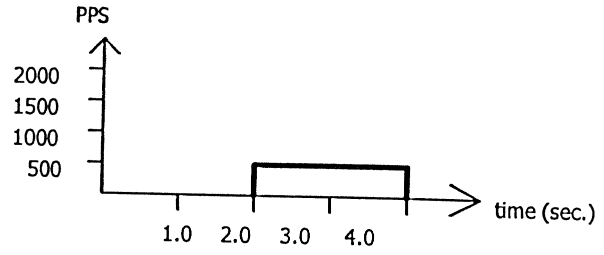
Example 3

In this application example, Motor A drives a Conveyor supplying material to be cut, and Motor B rotates a Cutting blade. IN.A0 is connected to a sensor that indicates if stock is in position, OUT.A0 is connected to an Alarm. The Program starts by initializing various Acceleration, Velocity, and Index parameters. Next, the Program enters Loop L0 and checks the status of IN.A0. If IN.A0 is closed(On), Motor A begins an Index Move of 6500 steps. Two seconds after Motor A starts, Motor B starts a one revolution Index Move. Motor A stops while Motor b is running, but before the cut begins. The Loop waits for Motor B to stop at its initial position, then returns to the top of the Loop. If IN.A0 is closed(On), the Loop continues. If IN.A0 is open(Off), the Program exits the Loop, an Alarm sounds for five seconds, and the Program terminates.

Program Line	Comments
@0	; Call Controller #0.
$WR.PROG3	; Write Program number 3 to memory.
ACC.RATE.A=3000	; Set Acceleration Rate of motor A to 3000 p/sec^2
DEC.RATE.A=3000	; Set Deceleration Rate of motor A to 3000 p/sec^2
MIN.VEL.A=500	; Set Start/Stop Speed of motor A to 500 pulse/sec
RUN.VEL.A= 2000	; Set Run/Max. Speed of motor A to 2000 pps
INDEX.A=6500	; Set Index Distance of motor A to 6500 steps
RUN.VEL.B=500	; Set Run/Max. Speed of motor B to 500 pps
MIN.VEL.B=500	; Set Start/Stop Speed of motor B to 500 pps
INDEX.B= 1000	; Set Index Distance of motor B to 1000 steps
L0	; Define Label #0
IF IN.A0 = = 1 THEN L0	; Waiting loop until IN.A0 closes (1=OFF, 0 = ON)
MOV.INDEX.A	; Index motor A to supply material
TIMER0=0	; Set TIMER #0 to 0, Timers increment by 10 ms
L1	; Define Label #1
IF TIMER0 < 200 THEN L1	; Loop for 2 seconds
MOV.INDEX.B	; Index motor B to cut the material
WAIT.STOP.B	; Wait until motor B stops
IF IN.A0 = = 0 THEN L0	; If Input A0 is closed (0=ON), go back to L0
TIMER0=0	; Sets Timer #0 to 0 after Program exits Loop L0
OUT.A0=0	; Set OUT.A0 to ON ,closed Output , Alarm On
L2	; Define Label #2
IF TIMER0 < 500 THEN L2	; Waiting loop for 5 seconds
OUT.A0=1	; Set OUT.A0 to OFF, open Output, Alarm Off
END	; End Program and return to Idle mode.

Velocity Profile





Running sequence for Program #3 based on Input A0

Example 4

This Program controls 4 Axes of movement. Axis A and B perform a position movement loop. In the meantime, axis C and D can start or finish toggling depending on the state of Input A2 and A3 respectively. Input A7 is used to control the termination of this Program.

The Scaling factors for Axis A, B, C and D are all equal to one.

```
Program line                  Comments
@0                            ; Call Controller #0
$WR.PROG4                     ; Write Program #3 to memory.
MIN.VEL.A=LREG0               ; 1. Parameter Settings for axis A-D
RUN.VEL.A=LREG1               ; They can be direct numbers or values of registers
ACC.RATE.A=LREG2              ;
MIN.VEL.B=500
RUN.VEL.B=2000
ACC.RATE.B=2000
MIN.VEL.C=500
RUN.VEL.C=2000
ACC.RATE.C=2000
INDEX.C=8000                  ; Movement distance for Toggle or Index move
MIN.VEL.D=800
RUN.VEL.D=3000
ACC.RATE.D=2000
INDEX.D=4000
HOME.ABCD                     ;2. Axes A, B, C and D return Home simultaneously.
EN.TIN.CD                     ;3. Enable Toggle Trigger Inputs for axes C and D.
                              ;   Now, Axes C and D can start or finish
                              ;   Toggling depending on the state of input A2
                              ;   and A3 respectively (regardless if axes A and B
                              ;   are moving or waiting).
L0                            ;4. Position Movement Loop for axes A and B
FIN.POS.A=DATA.A0
FIN.POS.B=DATA.B0
IF IN.A7= =0 THEN L3          ;   If Input A7 is low (0=ON), then Exit
POS.MOV.AB                    ;   else Axes A and B move to their Positions
WAIT.STOP.AB                  ;   Wait until axes A and B reach their Positions
LREG0=200                     ;   Input Parameter for Call Subroutine
CALL  L1                      ;   Wait 200x10ms=2sec
FIN.POS.A=DATA.A1
FIN.POS.B=DATA.B1
IF IN.A7= =0 THEN L3          ;   If Input A7 is low (0=ON), then Exit
POS.MOV.AB                    ;   Axes A and B move to their position 1
WAIT.STOP.AB                  ;   Wait until both axes A and B reach Position 1
```

```
LREG0=500
CALL L1                      ;  Wait 500x10ms=5sec
FIN.POS.A=0
FIN.POS.B=0
IF IN.A7= =0 THEN L3         ;  If Input A7 is low, then Exit
POS.MOV.AB                   ;  Axes A and B move back to Start Positions
WAIT.STOP.AB                 ;
LREG0=300
CALL L1                      ;  Wait 300x10ms=3sec
GOTO L0                      ;  Go back to Movement loop

L1                           ; 5. Subroutine to wait a certain time, input LREG0
TIMER0=0
L2
IF TIMER0<LREG0 THEN L2      ;  Wait LREG0 x 10ms
RET                          ;  End of subroutine

L3                           ; 6. Assign Label for Exiting segment
DIS.TIN.CD                   ;  Disable Toggle Inputs for Axes C and D
NOR.STOP.AB                  ;  Normal Stop Axes A and B
END                          ;  End Program and return to Idle mode.
```

File EX4-REG.S00 contains the variable Register Table for this example:

```
@0                           ; Call Controller # 0
$WR.VAR                      ; Down Load Register Values to memory.
LREG0=500                    ; Three Long Register values
LREG1=2000
LREG2=2000
END                          ; End down load and return to Idle mode.
```

File EX4-DATA.S00 contains the Position Data points for this example:

```
@0                    ; Call Controller # 0
$WR.DATA              ; Down Load Position Data points to memory.
DATA.A0=1000          ; Value stored in Position Data register, axis A, #0 = 1000
DATA.B0=2000          ; Value stored in Position Data register, axis B, #0 = 2000
DATA.A1=5000          ; Value stored in Position Data register, axis A, #1 = 5000
DATA.B1=8000          ; Value stored in Position Data register, axis B, #1 = 8000
END                   ; End down load and return to Idle mode.
```

See Section 8.3 for details on down loading Data to the Controller.

Example 5

This is a Programming example of how to use CALL commands IF...THEN statements as well as Mathematical operators.

```
Program line                      Comments
------------------------------------------------------------------------

@0                                ; Call Controller #0
$WR.PROG10                                ; Write/Download Program 10
RESET.AB                          ; Reset Current Positions to Zero
LREG1 = 1000                              ; Define Long Register LREG1
LREG2 = -3500                     ; Define Long Register LREG2
LREG3 = 10                        ; Define Long Register LREG4
LREG4 = LREG3 * LREG2             ; Multiply one Register by the other.

L1                                ; Define Label for Main Routine
CALL L10                          ; Call L10 ... Delay Subroutine
IF IN.A0 == 0 THEN L2             ; If Input A0 is active then skip two lines, goto L2
FIN.POS.B=LREG2                   ; Define Final Position for Axis B
CALL L12                          ; Call Subroutine L12
L2                                ; Define Label for Jumping
FIN.POS.A=LREG4                   ; Define Final Position for Axis A
CALL L13                          ; Call Subroutine L13
IF IN.A3 == 0 THEN PROG7                  ; If Input A3 is active then goto Program # 7
RESET.AB                          ; Reset Current Positions to Zero
GOTO L1                           ; Go back to the Start of Main Routine

L10                               ; Start of Delay Subroutine
TIMER0 = 0                        ; Clear Timer #0
L11
IF TIMER0 < 200 THEN L11                  ; If Timer is less than 2 sec then goto L11
RET                               ; Return to line below original Call command.

L12
POS.MOV.B                         ; Move Motor B to desired Position
WAIT.STOP.B                       ; Wait until Motor Stops
CALL L10                          ; Call L10 ... Delay Subroutine
RET                               ; Return to line below original Call command.

L13
POS.MOV.A                         ; Move Motor A to desired Position
WAIT.STOP.A                       ; Wait until Motor Stops
CALL L10                          ; Call L10 ... Delay Subroutine
RET                               ; Return to line below original Call command.

END                               ; End Program Execution
```