

	IB-11B021	
--	-----------	--

MACROPRO™ FOR WINDOWS™	DECEMBER 1996
------------------------	---------------



INSTRUCTION BOOK

INDUSTRIAL INDEXING SYSTEMS, Inc.

Revision - 0	
--------------	--

Approved By:	
--------------	--

Proprietary information of Industrial Indexing Systems, Inc. furnished for customer use only.
No other uses are authorized without the prior written permission of
Industrial Indexing Systems, Inc.

ER-6026

ERRATA SHEET, IB-11B021

JUNE 1997

Date	Rev.	ECN No.	DR	CHK	CHK
1/24/97	A	ECN-97-025 (See Note 1)			
6/11/97	B	ECN-97-204 (See Note 2)	TB		

Notes:

- 1) Section 5 - Instruction References Pages 144, 163, 179 added January 1997.
- 2) Section 3 - Control and Motion Concepts have been updated.
Section 5 - Instruction References Pages 231-236 have been updated.

INDUSTRIAL INDEXING SYSTEMS, Inc.

Tel: (585) 924-9181

626 Fishers Run
Victor, New York 14564

Fax: (585) 924-2169

Proprietary information of Industrial Indexing Systems, Inc. furnished for customer use only.
No other uses are authorized without the prior written permission of
Industrial Indexing Systems, Inc.

TABLE OF CONTENTS

Introduction _____ **Section 1**

Programming Language Concepts _____ **Section 2**

Control and Motion Concepts _____ **Section 3**

Analyzer _____ **Section 4**

Instruction Reference _____ **Section 5**

MACROPROTM

FOR WINDOWSTM

INDUSTRIAL INDEXING SYSTEMS

Introduction



Industrial Indexing Systems, Inc.

Table of Contents

<i>Introduction</i>	3
Overview	3
Highlights	4
Manual Conventions	5
<i>Getting Started</i>	6
System Requirements	6
Installing MacroPro	7
Application Development	7
Selecting a function using the IDE	7
Creating a Motion Control Program	7
Compiling Your Program	8
Testing Your Program	9
Running Your Program	9

This page is intentionally left blank.

Introduction

This document is one in a series of technical documents supporting Industrial Indexing Systems family of Multi-axis Servo Controllers (MSC). This particular document provides information about the MacroPro Development System for Windows, which serves as a tool to assist the user in the development of motion control programs.

Overview

The MacroPro Development System is intended to be used with the following products in the MSC series of programmable motion controllers:

1. The **MSC-800**, an 8-axis controller which simultaneously monitors and controls discrete I/O signals, resolver based servo axes and analog I/O axes.
2. The **MSC-850**, an enhanced version of MSC-800. This is also an 8-axis controller which simultaneously monitors and controls discrete I/O signals, resolver based servo axes, encoder based servo axes and analog I/O axes. Fiber optic communication, dual master angle bus configuration and programmable limit switch control are also provided.
3. The **MSC-850/32**, an enhanced version of MSC-850. This controller provides all the functionality of the MSC-850 with the added benefits of a 32-bit processor. A significant increase in application program size, from 16K to 128K was added.
4. The **MSC-250**, a 2 1/2 axis version of the MSC-850/32, providing basically the same features in a scaled down package. Application programs as large as 32K are supported.

The MacroPro Development System runs under the Microsoft Windows environment, providing users a very powerful, flexible and intuitive software approach to program generation and testing.

This manual was written with the assumption that you are somewhat familiar with Personal Computers, MS-DOS and the Microsoft Windows environment.

Highlights

1. The Integrated Development Environment (IDE) provides access to the major MacroPro features.
2. The MacroPro Help System allows:
 - a. Access to on-line documentation whenever needed.
 - b. Search for topic on keyword, history, next/previous.
 - c. The ability to cut and paste from examples or the documentation into your program.
3. The MacroPro Editor offers:
 - a. A multi-window, multi-file editing environment.
 - b. Access to the MacroPro Compiler while editing your program.
 - c. Automatic syntax verification of your program lines.
 - d. Unique color combinations for instructions, comments and errors.
 - e. Standard Windows based file management features.
 - f. On-line documentation.
4. The MacroPro Compiler provides:
 - a. Program compiler errors that are clear and concise.
 - b. Compiler errors are easily identified and can be located within your program by "double clicking" on the error description.
 - c. The ability to abort compiling and the ability to resume compiling.
5. The MacroPro Analyzer provides a real time test and analysis facility, including:
 - a. Program trace features.
 - b. The ability to select data variables and flags from a sorted list.
 - c. The ability to monitor data variables only, flags only or mixed data and flags.
 - d. Simultaneous reading and writing of data variables and flags.
 - e. Axis controller status flags can be monitored in real time.

- f. A controller information screen provides information on the hardware/firmware in use.
- g. The ability to view your source and symbol files.
- h. The ability to copy program and data space from the MSC Controller to an EPROM.

Manual Conventions

Throughout this manual, the following typeface conventions are used:

- 1. Program instruction names appear in **bold** print.
- 2. Optional instruction labels appear in *italics*.

Example 1

The following program line defines a timer (called MONITOR) with a time of 100.

```
wait_1_sec                set_tmr                MONITOR,100
```

The label *wait_1_sec* is an optional label. Although it may be needed because the user intends for the program to branch to that label during program execution, it is not required for the instruction **set_tmr** to be valid.

Example 2

The first line of the following group of program instructions, is an example of an instruction which requires an instruction label.

```
speed_table                begin_data  
                            data                100,150,200,600,1200  
                            end_data
```

The label **speed_table** is a required label. This label identifies a place in the program data area, where an array of data can be retrieved. Without the label, we have no way of accessing the data.

Getting Started

The first step to developing and testing your programs is to install the MacroPro Development System on your computer. While this is a relatively simple task, it is important to understand the recommended and minimum hardware configuration of your Personal Computer. Once your software has been successfully installed, take a few minutes to read the section which follows regarding the program development cycle. This should give you some insight into the major components of the MacroPro Development System and how the pieces fit together.

System Requirements

System

A 386 class machine is required minimally. A 33MHz 486 class machine with at least 4MB of RAM is recommended. Performance will also vary depending on the video adapter, driver and video mode you have selected.

Memory

MacroPro requires 2MB of memory to operate. Additional memory may be required depending on the size of the programs being developed.

Disk Space

When fully installed, MacroPro uses approximately 5MB of disk space. Additional disk space may be required depending on your application(s).

Windows

You must be running Microsoft Windows version 3.1 or greater in enhanced mode.

Installing MacroPro

Like many Windows based applications, a program called SETUP.EXE resides on the MacroPro Installation Disk and will guide you through the installation process. Basically, you will do the following steps:

1. Begin Windows by typing win at the MS-DOS prompt and pressing the Enter key. You should be running Windows version 3.1 or greater.
2. Insert the disk labeled Disk 1 in your floppy disk drive.
3. Choose the Run command from the File menu in Windows Program Manager. Type a:setup and press the Enter key (or b:setup if using the drive b:).
4. Follow the instructions on the screen and provide the appropriate responses to the installation options as needed.

Once the installation process is complete, you should be able to run the MacroPro Development System by selecting the corresponding icon or running from the Program Manager.

Application Development

The MacroPro Development System provides you with the necessary tools for creating, compiling and testing your motion control programs. A description of the major components of the MacroPro Development System follows.

Selecting a function using the IDE

The MacroPro IDE (Integrated Development Environment) provides easy access to the MacroPro Editor, Analyzer and Help System. Selections are made by pressing the desired button on the IDE screen. Selecting the Exit button will close down the MacroPro Development System.

Creating a Motion Control Program

Application programs are created and/or changed using the MacroPro Editor. The Editor has a wealth of features that will help streamline the editing of your application programs. The actual text that you enter is often referred to as the **source** file.

Compiling Your Program

The program **source** file will need to be compiled before it can be run on your MSC Controller. The process of compiling must be initiated by you, although the actual process of compiling is transparent to you.

The Compiler is initiated from the Editor. By pressing the Compile button from the Toolbox, the compile process will begin. The **source** file currently being edited, will be saved. Then, the Compiler checks for syntax errors in your **source** file and build other files that will be required to run your program.

Typically, the Compiler will build a **symbol** file and an **error** file. These files are built if errors are built regardless of whether an error was or was not encountered. An optional **listing** file might also be created.

A **trace** file and **executable** file will also be created if no compile errors were discovered. The Compiler will always indicate whether a compile was successful, unsuccessful or was aborted by the programmer.

Since the **executable** file will only be created if there were no Compiler errors, it follows that only compiled programs without errors can be run on the MSC Controller.

Testing Your Program

Application programs are tested using the MacroPro Analyzer. The Analyzer is used to transmit, start, stop and autostart your application program at the MSC. The Analyzer also provides the tools to watch program variables, trace program execution and monitor the status of the MSC Controller and related axes, I/O and other events.

If the Analyzer is selected from the Editor, the currently selected file will be used **provided that file has been successfully compiled and is error free**. Other application programs to be run can be selected while in the Analyzer assuming that they are error free.

While in the Analyzer, you will have easy access to all program **source** and **symbol** file information which will prove useful as your program is being tested.

Using the Analyzer, you can also "burn" a copy of the program resident on your MSC Controller onto an erasable PROM (EPROM) for archive or backup purposes.

Running Your Program

Once your program has been successfully tested, you are ready to setup your MSC Controller so that the program will run automatically on system power-up. There are two ways to do this.

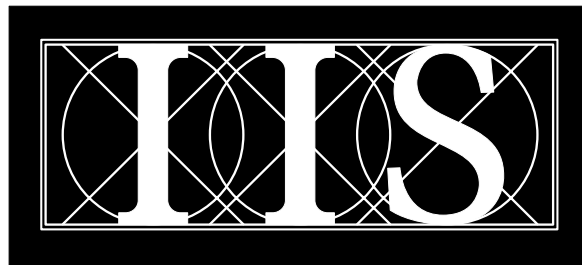
The most commonly used method is, through the Analyzer, to set the MSC Controllers' AUTOSTART flag to be active. By doing this, your program will automatically run each time power to the MSC Controller is enabled.

The other method would be to enable power to the MSC Controller while an EPROM containing your application program is inserted into the Controllers' EPROM POCKET. The MSC Controller **always** looks to the EPROM POCKET first. If an EPROM is inserted into the POCKET with a valid program, that program will be loaded from the EPROM and automatically run. The MSC Controller will also enable the AUTOSTART flag for you, as well.

This method is not recommended usually, however, as your program data area will be constantly reset to that of the image on the EPROM. This may not be desirable, depending on your application.

MACROPROTM **FOR WINDOWSTM** **INDUSTRIAL INDEXING SYSTEMS**

Programming Language
Concepts



Industrial Indexing Systems, Inc.

Table of Contents

<i>Program Language Concepts</i>	2
Program Statement Format	3
Comments	4
Blank Lines	4
Label Lines	5
<i>Program Compiler</i>	6
Configuring the Compiler	6
The MACROPRO.INI File	6
Compile Time Problems	7
Aborting the Compiler	7
Compiling is Complete	8
Compiler Output	8
<i>Compiler Directives</i>	9

This page is intentionally left blank.

Program Language Concepts

A program is an organized group of statements that can be executed by an MSC Controller. Application programs reside in a section of non-volatile memory in the controller.

In the MSC-850/32, there are 64,000 bytes allocated for the program area and an additional 64,000 bytes allocated for the data area.

In the MSC-250, 32,000 bytes of storage are available for the combined program and data areas.

In an MSC-800 and MSC-850, 16,000 bytes of storage are available for the combined program and data areas. This memory space is dynamically allocated as the program is compiled. The size of each area may vary, so long as the sum of the program and data areas do not exceed 16,000 bytes.

Program Statement Format

Program *statements* are, in many ways, similar to the statements of the BASIC computer language. Each statement may consist of up to 4 parts; a *label*, an *instruction*, a list of *parameters*, and an optional *comment*. This format is illustrated in Figure 2.1.

1. *label* - A label can be from one (1) to twelve (12) characters in length, and may consist of both upper and lower case letters, numbers and the underscore character. Labels must begin in column 1 of the program line. Labels are optional. If no label is used, at least one blank must be at the beginning of the line.
2. *instruction* - This part of the line consists of any valid MacroPro Programming Language command. Instructions must be typed in lower case letters. At least one blank must precede the instruction and at least one blank must follow the instruction.
3. *parameters* - This part of the program line consists of the information, if any, processed by the instruction. It must be separated from the instruction by one or more spaces.
4. *comment* - This field can contain any explanatory information about the program line. It must be separated from the preceding field by at least one blank. Program lines which contain only comments must have an exclamation point (!) in column 1.

label *instruction* *parameter list* *comment*

Figure 2.1 - Program Instruction Format

The label portion of the statement is not always required, depending on the type of instruction used. It is required for instructions which define variables, text strings, arrays, or equated values.

There are instructions which do not require a list of parameters. However, there are usually one or more parameters for an instruction. In the example in Figure 2.2, two parameters are required.

label *set_speed* *axis_1,300* *comment*

Figure 2.2 - Instruction With Two Parameters

Comments

Comments provide a means for a programmer to describe the functions being performed by a particular instruction or group of instructions. It is important to note that comments do not use any of the storage within the program memory space. The liberal use of comments is highly recommended. There are two ways to implement comments in a program:

1. An entire line of text can be dedicated as a comment line by placing an exclamation point in the first character position of the line.
2. Comments can be placed at the end of an instruction line by leaving at least one space between the last parameter in the instruction and the beginning of the comment.

Comments of both formats are included in the sample program shown in Figure 2.3.

Blank Lines

Blank lines may be entered in the program to make it more readable. For example, a subroutine might be separated from the main body of the program by one or more blank lines. As with comments, blank lines do not use any storage within the program.

Label Lines

Program lines containing only a label are allowed. It is often handy to place a line containing only a label just ahead of an instruction it references. In this way, it is easier to insert new instructions in the program if necessary during the testing process. Lines containing only a label use no storage within the program.

<u>LABEL</u>	<u>OPERATIONPARAMETERS</u>	<u>COMMENT</u>	
	msc_type	850	
	declare	ON	
POS_1	equ	81920	two turns CW
POS_2	equ	-81920	two turns CCW
LIFT	equ	1	use axis 1
LIFT_DOWN	equ	93	down flag
LIFT_BUSY	equ	94	busy flag
NO_FAULT	equ	0	I/O zero
! do initialization functions			
	turn_on	NO_FAULT	show no fault
	drive_on	LIFT	enable drive
	set_speed	LIFT,50	500 rpm speed
	set_ac_dc	LIFT,20	rev/sec/sec
! set a local zero at current position			
	set_local	LIFT	
! begin main program loop			
restart	position	LIFT,POS_1	go 2 turns CW
loop1	if_stat_on	LIFT_DOWN,fault	check error
	if_stat_on	LIFT_BUSY,loop1	wait for done
	position	LIFT,POS_2	go 2 turns CCW
loop2	if_stat_on	LIFT_DOWN,fault	check error
	if_stat_on	LIFT_BUSY,loop2	wait for done
	goto	restart	do it all again
fault	turn_off	NO_FAULT	signal error
	sys_return		

Figure 2.3 - Sample Application Program

Program Compiler

The MacroPro Compiler converts MSC application programs from human readable instructions (**source files**) to numeric codes (**executable files**) which can be interpreted and acted upon by the MSC.

In addition, the Compiler produces information which is used by the MacroPro Editor and MacroPro Analyzer to simplify the programming and testing processes. The user can specify that an optional listing file be created when desired.

The Compiler is invoked by the user from the MacroPro Editor, by pressing the Compile button.

Configuring the Compiler

The user may request that a listing file be created during the compile process. This is typically a rather large file that contains more information than most care to know about the details of a compiled file. It is, however, available to those users with a need to know.

A listing file will be created, if the "List file=ON" string is read from the "[Compiler]" section of the file MACROPRO.INI. This file will be found in your WINDOWS directory.

A listing file will not be created if the "List file=OFF" string is read from the "[Compiler]" section of the file MACROPRO.INI.

By default, a listing file will not be created.

The MACROPRO.INI File

The Compiler will attempt to read the MACROPRO.INI file from the WINDOWS directory at the start of the compile process. This file should minimally contain lines similar to those listed below.

The following lines indicate the directory containing the file MACROPRO.OPC.

```
[Environment]
Exec dir=c:\directory
```

The following lines indicate the path and name of the last file used.

```
[Last File]
Last file=c:\progdire\name.prg
```

The following lines indicate whether a listing file should be produced at compile time.

```
[Compiler]
```

List file=ON

Compile Time Problems

Initiating the Compiler is a very simple task. However, there are a number of conditions which might cause the Compiler to abort. The Compiler will attempt to read the MACROPRO.INI file from the WINDOWS directory at the start of the compile process. The process will be aborted, with an error message, for any of the following reasons:

1. The MACROPRO.INI file was not found in the WINDOWS directory.
2. The "[Environment]" section was not found.
3. The "Exec dir=" string was not found within the "[Environment]" section.
4. The MACROPRO.OPC file was not found in the directory listed in the "[Environment]" section.
5. The path and filename passed to the Compiler is illegal or does not exist.

Should you encounter any of these problems, verify that your MACROPRO.INI file is located in the proper directory and has the correct syntax as described above.

Aborting the Compiler

Once the compile process has begun, you may abort that process if necessary. A dialog box will be displayed, indicating the current pass of the Compiler and the percent complete for that pass (the Compiler is a four pass process, where passes 2 and 4 do most of the work). An ABORT button will also be shown.

Press this button to abort the compile process at any time. The Compiler will display a message box, requesting that you confirm whether to abort or not. Press the YES button to confirm the abort or NO to resume compiling. The Compiler will pick up where it left off, if you decide to resume.

Compiling is Complete

When the compile process is complete, a message box will be displayed indicating that the compile was successful or not successful.

If successful, the Program Size (in bytes) and the Data Size (in bytes) will be displayed. If unsuccessful, a message will be displayed indicating the number of errors detected.

In either case, you must press the OK button in the message box in order to continue.

Compiler Output

The MacroPro Compiler will generate the following files:

1. The **listing file** is an optional file. It will be created if specified in the MACROPRO.INI file. The listing file has the program file extension **.lst**. It will be created even if program errors were detected.

The **listing file** consists of:

- a. An EQUATE Table - This table lists all symbols defined in **equ** statements. Its contents are symbol name, decimal symbol value, and hexadecimal symbol value.
 - b. A LABEL Table - This table contains all program statement labels and their equivalent addresses.
 - c. A CONSTANTS Table - This consists of a list of all constants used. This table consists only of address and values.
 - d. A DATA Table - This table contains all the data variables used in the program. Each entry in this table consists of the variable name, followed by its address in both decimal and hexadecimal formats.
 - e. A PROGRAM Listing - This portion consists of a listing of each program line together with the MSC numerical equivalent of the program line. Any errors detected in a program line will be listed just after the line containing the error.
2. An **error file** is always created and always includes some header information. When errors are detected, they are written to this file. The information regarding each error includes; program source file, line number and error description. The error file has the program file extension **.err**. It is used by the MacroPro Editor to assist in locating and correcting errors.
 3. A **symbol file** is always created. This file contains all symbol, label, and data definitions. The symbol file has the program file extension **.sym**. It is used by the Analyzer to assist in the testing and analysis process.

4. The **executable file** is only created when no compiler errors are detected. This file contains the compiled program in MSC machine instruction format. The executable file has the program file extension **.mcm**. The Analyzer transmits this file to the MSC Controller where it will be executed.
5. The **trace file** is only created when no compiler errors are detected. This file contains the information needed by the Analyzer to translate trace information into source program lines. This file has the program file extension **.mcm**. The trace file is only used by the Analyzer.

Compiler Directives

Compiler directives are a class of instructions which simplify such operations as defining constants, data arrays and cams, or which cause the MacroPro Compiler to function in a particular way. These instructions have no direct effect on the MSC Controller. Their purpose is to simplify the programmer's task.

A list of compiler directive instructions and their formats is shown in Figure 2.4.

Instruction	Format		
begin_data	label	begin_data	
begin_cam	label	begin_cam	
cam		cam	value,value,etc.
data		data	value,value,etc.
declare		declare	mode
dim	label	dim	controller#,size
end_cam		end_cam	
end_data		end_data	
equ	label	equ	value
integer	label	integer	
msc_type		msc_type	system_type
text	label	text	"ASCII string"

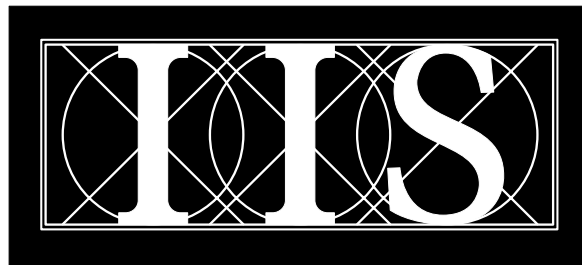
Figure 2.4 - Compiler Directives

MACROPROTM

FOR WINDOWSTM

INDUSTRIAL INDEXING SYSTEMS

Control and Motion Concepts



Industrial Indexing Systems, Inc.

Table of Contents

Flags	6
Flag Categories	6
MSC-800, MSC-850, MSC-850/32 Internal Status Flags	6
MSC-250 Internal Status Flags	7
Timers	7
Controller Flags	8
ACR-850 Controller Status Flags	8
ACE-850 Controller Status Flags	9
MCF-850 Multi-Function Controller Status Flags	9
HPL-850 Controller Status Flags	10
ACM-850 Analog Controller Status Flags	10
MSC-250 Controller Status Flags	11
MSC-250 Pseudo Axis Controller Status Flags	11
Flag Descriptions	12
Flag Instruction Summary	15
Arithmetic Instructions	16
Integer Arithmetic	16
Array Manipulation	17
Byte Operations	17
Bit Oriented Operations	18
Arithmetic Functions	18
Arithmetic Instruction Summary	18
Program Flow Instructions	19
Branching Instructions	19
Subroutine Control	19
The Select Statement	20
Program Branching Instructions	21
Motion Instructions	22
Position Data	22
Speed (Velocity) Data	22
Acceleration Data	22
Global And Local Zeroes	23

Motion Preparation Instructions	24
Digital Compensation	25
The P Term (Proportional Gain)	26
The I Term (Integral)	26
The D Term (Differential)	26
Velocity Gain	27
Velocity Control Instructions	27
Positioning Instructions	28
<i>Piecewise Profiles</i>	29
Building Profile Data Tables	29
Piecewise Profiles And Master Slave	32
Reading Controller Position	32
<i>Master Slave Concepts</i>	33
Simple Lock (Electronic Gearbox)	34
Useful Facts About Simple Lock Mode	34
Lock Methods For Simple Lock	36
Lock Method 1	36
Lock Method 4	39
Lock Method 6	39
Electronic Cams	40
Master Scaling	40
Data Scaling	41
Important Notes Regarding Electronic Cams	41
Calculating Electronic Cams	42
Electronic Cam Lock Methods	45
Lock Method 0	47
Lock Method 5	47
Lock Method 8	47
Lock Method 9	47
Lock Method 10	48
Sample Electronic Cam Application	48
Piecewise Lock	50
Master Angle Bus	50
Master Angle Bus Cautions	52

Fiber Optic Network	53
Master/Slave Instructions	58
<i>Programmable Limit Switches</i>	59
MSC-850/MCF-850 PLS Functions	59
MSC-250 PLS Functions	59
Programming using PLS's	60
Processing of programs using PLS's	60
Execution of programs using PLS's	60
MSC-250 Programming Example using PLS's	61
High Performance Programmable Limit Switch (MSC-850/HPL-850)	63
Theory Of Operation of PLS's	63
Programming Considerations for PLS's	64
HPL-850 Programming Example #1 using PLS's	64
HPL-850 Programming Example #2 using PLS's	65
Programmable Limit Switch Instructions	67
<i>Interrupts</i>	68
Software Interrupts	68
Hardware Interrupts	69
Interrupt Instructions	70
<i>Extended Memory Operations</i>	71
Extended RAM	71
Extended RAM Programming	71
Extended Memory Limitations	72
EPROM Memory	72
Automatic Program Load From EPROM	72
EPROM Status Codes	73
EPROM Manager Instructions	73
<i>Analog Input/Output</i>	74
Capabilities of Analog Input/Output	74
ACM-850 Functional Description	74
Power-Up States for Analog I/O Channels	75
ACM-850 Instructions	75
<i>User Serial Ports</i>	76
Serial Port Initialization	77

Important Notes Regarding Serial Ports	78
Serial Instructions	79

This page is intentionally left blank.

Flags

A flag is a bit in memory representing the current state of a timer, axis status, I/O or other condition. The MSC has reserved a storage area in memory for 256 flags. A flag can be either on (1) or off (0).

There are a number of program instructions dealing with flags. In fact, taken as a group, flag instructions comprise the largest subset of program instructions.

There are flag instructions to read inputs, to turn outputs on and off, to start and stop timers, to read motor activity and fault conditions and to read, set and clear user program switches.

Figures 3.1 through 3.9 list the flags used for each MSC Controller.

Flag Categories

The following are the major categories of flags:

MSC-800, MSC-850, MSC-850/32 Internal Status Flags

MSC-250 Internal Status Flags

MSC-800, MSC-850, MSC-850/32 Internal Status Flags

FLAG #	FUNCTION	CATEGORY
0 - 7	ON BOARD I/O, PERMANENTLY ASSIGNED	INPUT/ OUTPUT
8 - 23	I/O EXPANDER ADDRESS CODE "A"	
24 - 39	I/O EXPANDER ADDRESS CODE "B"	
40 - 55	I/O EXPANDER ADDRESS CODE "C"	
56 - 71	I/O EXPANDER ADDRESS CODE "D"	
72 - 79	PROGRAMMABLE TIMERS	TIMER
80 - 95	CONTROLLER #1 STATUS FLAGS	CONTROLLER STATUS
96 - 111	CONTROLLER #2 STATUS FLAGS	
112 - 127	CONTROLLER #3 STATUS FLAGS	
128 - 143	CONTROLLER #4 STATUS FLAGS	
144 - 159	CONTROLLER #5 STATUS FLAGS	
160 - 175	CONTROLLER #6 STATUS FLAGS	
176 - 191	CONTROLLER #7 STATUS FLAGS	
192 - 207	CONTROLLER #8 STATUS FLAGS	
208 - 255	GENERAL PURPOSE USER FLAGS	USER

Figure 3.1 - MSC-800, MSC-850, MSC-850/32 Internal Status Flags

MSC-250 Internal Status Flags

FLAG #	FUNCTION	CATEGORY
0 - 15	ON BOARD I/O, PERMANENTLY ASSIGNED	INPUT/ OUTPUT
16 - 31	I/O EXPANDER ADDRESS CODE "A"	
32 - 47	I/O EXPANDER ADDRESS CODE "B"	
48 - 63	NOT USED	
64 - 71	SOFTWARE PLS FLAGS	
72 - 79	PROGRAMMABLE TIMERS	TIMER
80 - 95	CONTROLLER #1 STATUS FLAGS	CONTROLLER STATUS
96 - 111	CONTROLLER #2 STATUS FLAGS	
112 - 127	CONTROLLER #3 STATUS FLAGS	
128 - 143	NOT USED	
144 - 159	NOT USED	
160 - 175	NOT USED	
176 - 191	NOT USED	
192 - 207	NOT USED	
208 - 255	GENERAL PURPOSE USER FLAGS	USER

Figure 3.2 - MSC-250 Internal Status Flags

Timers

MSC Controllers provide 8 user programmable timers. These timers have a resolution of 10 milliseconds per "tick".

Timers are used by first setting the timer to the desired number of counts or ticks. This causes the flag associated with the timer to turn on and to remain on until the specified time has passed.

User timers are decremented on every other tick of the system's 5 millisecond clock. As a result of this, these timers should be considered accurate to +/- 10 milliseconds.

Controller Flags

The following are categories of various controller flags:

ACR-850 Controller Status Flags
 ACE-850 Controller Status Flags
 MCF-850 Multi-Function Controller Status Flags
 HPL-850 Controller Status Flags
 ACM-850 Analog Controller Status Flags
 MSC-250 Controller Status Flags
 MSC-250 Pseudo Axis Controller Status Flags

ACR-850 Controller Status Flags

#1	#2	#3	#4	#5	#6	#7	#8	DESCRIPTION
80	96	112	128	144	160	176	192	MDU FAIL
81	97	113	129	145	161	177	193	LOCK PENDING
82	98	114	130	146	162	178	194	FOLLOWING ERROR
83	99	115	131	147	163	179	195	MASTER/TEST MODE
84	100	116	132	148	164	180	196	CAM/PW PROFILE SIZE EXCEEDED
85	101	117	133	149	165	181	197	COMMAND INVALID IN THIS STATE
86	102	118	134	150	166	182	198	LOSS OF RESOLVER
87	103	119	135	151	167	183	199	TIME-OUT
88	104	120	136	152	168	184	200	JOGGING
89	105	121	137	153	169	185	201	INDEXING
90	106	122	138	154	170	186	202	CALCULATION IN PROGRESS
91	107	123	139	155	171	187	203	HARDWARE INTERRUPT ARMED
92	108	124	140	156	172	188	204	FORCED DECEL IN PROGRESS
93	109	125	141	157	173	189	205	DOWN
94	110	126	142	158	174	190	206	BUSY
95	111	127	143	159	175	191	207	MASTER/SLAVE LOCK

Figure 3.3 - ACR-850 Controller Status Flags

ACE-850 Controller Status Flags

#1	#2	#3	#4	#5	#6	#7	#8	DESCRIPTION
80	96	112	128	144	160	176	192	MDU FAIL
81	97	113	129	145	161	177	193	LOCK PENDING
82	98	114	130	146	162	178	194	FOLLOWING ERROR
83	99	115	131	147	163	179	195	MASTER/TEST MODE
84	100	116	132	148	164	180	196	CAM/PW PROFILE SIZE EXCEEDED
85	101	117	133	149	165	181	197	COMMAND INVALID IN THIS STATE
86	102	118	134	150	166	182	198	MARKER OUT OF LIMIT
87	103	119	135	151	167	183	199	TIME-OUT
88	104	120	136	152	168	184	200	JOGGING
89	105	121	137	153	169	185	201	INDEXING
90	106	122	138	154	170	186	202	CALCULATION IN PROGRESS
91	107	123	139	155	171	187	203	HARDWARE INTERRUPT ARMED
92	108	124	140	156	172	188	204	FORCED DECEL IN PROGRESS
93	109	125	141	157	173	189	205	DOWN
94	110	126	142	158	174	190	206	BUSY
95	111	127	143	159	175	191	207	MASTER/SLAVE LOCK

Figure 3.4 - ACE-850 Controller Status Flags

MCF-850 Multi-Function Controller Status Flags

#1	#2	#3	#4	#5	#6	#7	#8	DESCRIPTION
80	96	112	128	144	160	176	192	MDU FAIL
81	97	113	129	145	161	177	193	PLS FLAG 16
82	98	114	130	146	162	178	194	NOT USED
83	99	115	131	147	163	179	195	PLS FLAG 17
84	100	116	132	148	164	180	196	PLS FLAG 19
85	101	117	133	149	165	181	197	COMMAND INVALID IN THIS STATE
86	102	118	134	150	166	182	198	PLS FLAG 18
87	103	119	135	151	167	183	199	TIME-OUT
88	104	120	136	152	168	184	200	JOGGING
89	105	121	137	153	169	185	201	INDEXING
90	106	122	138	154	170	186	202	CALCULATION IN PROGRESS
91	107	123	139	155	171	187	203	NOT USED
92	108	124	140	156	172	188	204	FORCED DECEL IN PROGRESS
93	109	125	141	157	173	189	205	DOWN
94	110	126	142	158	174	190	206	BUSY
95	111	127	143	159	175	191	207	MASTER/SLAVE LOCK

Figure 3.5 - MCF-850 Multi-Function Controller Status Flags

HPL-850 Controller Status Flags

#1	#2	#3	#4	#5	#6	#7	#8	DESCRIPTION
80	96	112	128	144	160	176	192	MDU FAIL
81	97	113	129	145	161	177	193	PLS FLAG 16
82	98	114	130	146	162	178	194	NOT USED
83	99	115	131	147	163	179	195	PLS FLAG 17
84	100	116	132	148	164	180	196	PLS FLAG 19
85	101	117	133	149	165	181	197	COMMAND INVALID IN THIS STATE
86	102	118	134	150	166	182	198	PLS FLAG 18
87	103	119	135	151	167	183	199	TIME-OUT
88	104	120	136	152	168	184	200	NOT USED
89	105	121	137	153	169	185	201	NOT USED
90	106	122	138	154	170	186	202	CALCULATING PLS DATA
91	107	123	139	155	171	187	203	NOT USED
92	108	124	140	156	172	188	204	NOT USED
93	109	125	141	157	173	189	205	DOWN
94	110	126	142	158	174	190	206	NOT USED
95	111	127	143	159	175	191	207	NOT USED

Figure 3.6 - HPL-850 Controller Status Flags

ACM-850 Analog Controller Status Flags

#1	#2	#3	#4	#5	#6	#7	#8	DESCRIPTION
80	96	112	128	144	160	176	192	NOT USED
81	97	113	129	145	161	177	193	NOT USED
82	98	114	130	146	162	178	194	NOT USED
83	99	115	131	147	163	179	195	MASTER TEST MODE
84	100	116	132	148	164	180	196	NOT USED
85	101	117	133	149	165	181	197	COMMAND INVALID IN THIS STATE
86	102	118	134	150	166	182	198	NOT USED
87	103	119	135	151	167	183	199	TIME-OUT
88	104	120	136	152	168	184	200	NOT USED
89	105	121	137	153	169	185	201	NOT USED
90	106	122	138	154	170	186	202	NOT USED
91	107	123	139	155	171	187	203	NOT USED
92	108	124	140	156	172	188	204	NOT USED
93	109	125	141	157	173	189	205	DOWN
94	110	126	142	158	174	190	206	NOT USED
95	111	127	143	159	175	191	207	NOT USED

Figure 3.7 - ACM-850 Analog Controller Status Flags

MSC-250 Controller Status Flags

#1	#2	DESCRIPTION
80	96	NOT USED
81	97	LOCK PENDING
82	98	FOLLOWING ERROR
83	99	MASTER/TEST MODE
84	100	CAM/PW PROFILE
85	101	COMMAND INVALID IN THIS STATE
86	102	NOT USED
87	103	NOT USED
88	104	JOGGING
89	105	INDEXING
90	106	CALCULATION IN PROGRESS
91	107	HARDWARE INTERRUPT ARMED
92	108	FORCED DECEL IN PROGRESS
93	109	DOWN
94	110	BUSY
95	111	MASTER/SLAVE LOCK

Figure 3.8 - MSC-250 Controller Status Flags

MSC-250 Pseudo Axis Controller Status Flags

#3	DESCRIPTION
112	NOT USED
113	NOT USED
114	NOT USED
115	NOT USED
116	CAM/PW PROFILE SIZE EXCEEDED
117	COMMAND INVALID IN THIS STATE
118	NOT USED
119	NOT USED
120	JOGGING
121	INDEXING
122	CALCULATION IN PROGRESS
123	NOT USED
124	FORCED DECEL IN PROGRESS
125	NOT USED
126	BUSY
127	NOT USED

Figure 3.9 - MSC-250 Pseudo Axis Controller Status Flags

Flag Descriptions

MDU FAIL

A test of the MDU (Multiply/Divide Unit) is made on system power up. The purpose of the test is to exercise the arithmetic functions of the MDU (shift, rotate, add, subtract etc.). This flag will be set if an error occurs while testing the unit. This flag is not used in the MSC-250 as there is no Multiply/Divide chip.

POSITION LOOP FAIL

A test of the Resolver to Digital Converter is made on system power up. This flag will be set if an error is encountered with the unit. This flag is not used in the MSC-250.

FOLLOWING ERROR

A following error occurs when the difference between the actual transducer angle and the expected transducer angle is outside the allowable range. The angles are compared every 10 ms. The allowable error is +/- 17 degrees if the motor shaft is motionless and +/- 180 degrees if the motor shaft is moving. **NOTE:** If digital compensation is being used, the allowable error while the motor shaft is moving becomes $\pm 180 \times 16 / \text{PGAIN}$ degrees, where PGAIN is the proportional gain setting being used.

MASTER/TEST MODE

This flag will be set if the axis has been put into the Master/Test mode. All motor fault conditions will have been reset. This is the default, power on state for an axis controller. For the ACR-850, ACE-850 and MSC-250 Axis Controllers, the servo amplifier will have been disabled and no checks for servo following errors will be made. The analog position output will be set to be proportional to the transducer position as shown in Figure 3.10.

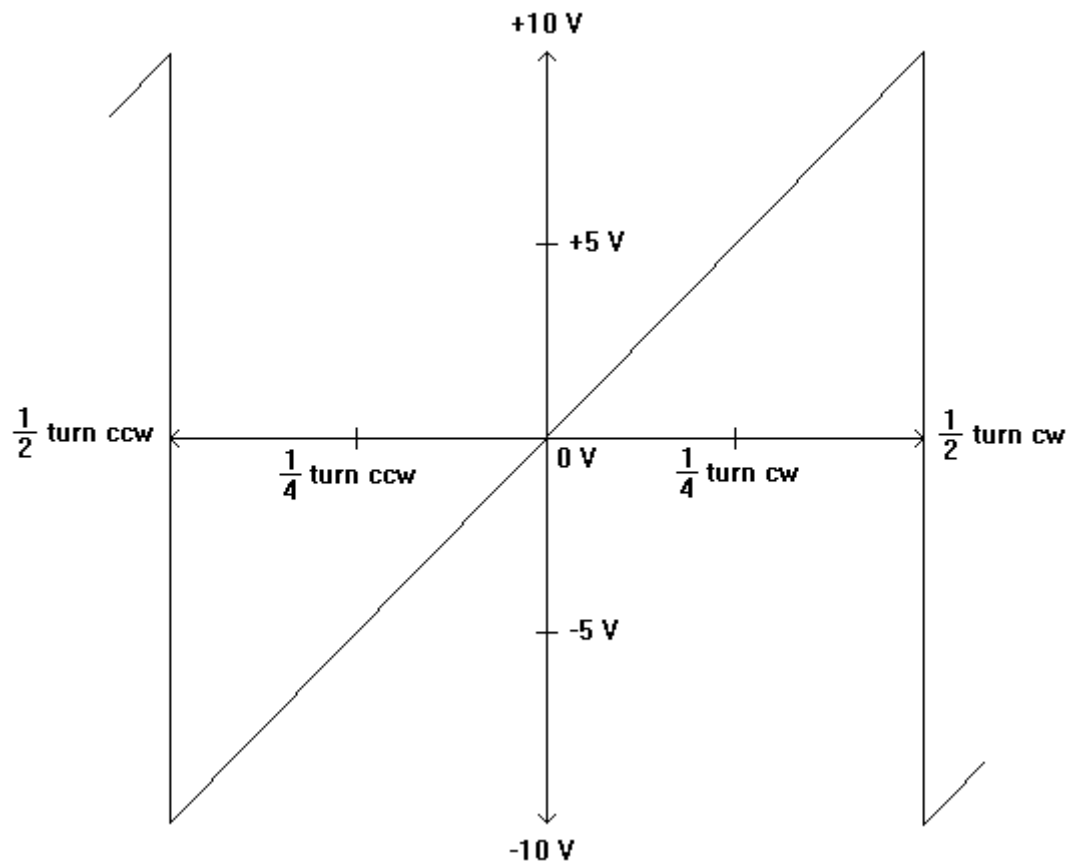


Figure 3.10 - Position Output vs. Transducer Angle

CAM SIZE EXCEEDED/PIECEWISE PROFILE ARRAY FULL

This flag will be set if the number of cam array data elements exceeds 28K bytes, or if the number of piecewise profile segments exceeds 99.

COMMAND INVALID IN THIS STATE

This flag will be set whenever the axis is requested to execute a command that it is not capable of at that moment. This typically occurs when program steps are out of logical order. An example would be the situation where the axis is currently jogging and is then commanded to index without first completing a **f_decel** instruction and waiting for the motor to stop.

MARKER OUT OF LIMIT

This flag, specific to the MSC-850/ACE-850 and MSC-250 Axis Controllers, indicates that the controller observed a marker pulse from the encoder that was more than +/- 5 bits from the expected position, and that the marker correction function has been suspended. The flag will be cleared if the marker pulse is subsequently observed within the +/- 5 bit window.

LOSS OF RESOLVER

This flag, specific to the ACR-850C (SFO-8060) Axis Controller card, indicates that the resolver converter has lost one or more of the required signals for conversion or the converter has lost its ability to track the angle due to speeds in excess of 7200 RPM.

TIME-OUT

On system power up each controller card slot in an MSC Controller is checked for the existence of a functional controller card. This flag will be set for each axis that does not have a functional card installed or for those slots experiencing communications problems. This flag is not used in the MSC-250.

JOGGING

This flag will be set when a **jog_cw**, **jog_ccw**, **track_spd**, **l_track_spd**, **vel_cw** or **vel_ccw** command is executed. This flag will be cleared after completing a **f_decel** operation.

INDEXING

This flag will be set when an index or position command is executed. This flag will be cleared after completing the selected move.

CALCULATING PIECEWISE PROFILE

This flag will be set while a Piecewise profile is being calculated. It will be cleared otherwise.

CALCULATING PLS DATA

This flag will be set while Programmable Limit Switch data is being calculated.

HARDWARE INTERRUPT ARMED

The axis controller has received a hardware interrupt instruction and is monitoring its respective hardware interrupt input line.

FORCED DECEL IN PROGRESS

This flag will be set when an **f_decel** or **unlock** command is executed and the axis is still in motion.

DOWN

This flag will be set if a FOLLOWING ERROR or TIME-OUT is encountered (FOLLOWING ERRORS and TIME-OUTs are described above).

BUSY

This flag will be set when the axis is busy performing a motion related operation such as **jog_cw**, **jog_ccw**, **index**, **position**, **vel_cw**, **vel_ccw**, **track_spd**, **l_track_spd**, or **exec_profile**. If an axis card experiences a **DOWN** condition while the **BUSY** flag is set, **BUSY** will remain set.

MASTER/SLAVE LOCK

This flag will be set when a **lock** command is executed. It will be cleared when a **f_decel**, **unlock**, or **enable** command is executed.

BUSY PROCESSING

This flag is set when the axis processor is executing a lengthy command to prevent the main processor from sending another program instruction for the specified axis controller. The following program Instructions cause this flag to be set: **calc_cam_sum**, **calc_unit_cam**, **drive_on**, **drive_off**, **find_mrk_ccw**, **find_mrk_cw**, **over_draw**, **prep_profile**, and **test_mode**.

Flag Instruction Summary

Instruction	Format
blk_io_in	<i>label</i> blk_io_in input flag#,variable
blk_io_out	<i>label</i> blk_io_out output flag#,variable
clr_flag	<i>label</i> clr_flag user flag#
get_status	<i>label</i> get_status controller#
if_flag_off	<i>label</i> if_flag_off user_flag#,address_label
if_flag_on	<i>label</i> if_flag_on user_flag#,address_label
if_io_off	<i>label</i> if_io_off I/O flag#,address_label
if_io_on	<i>label</i> if_io_on I/O flag#,address_label
if_stat_off	<i>label</i> if_stat_off status_flag#,address_label
if_stat_on	<i>label</i> if_stat_on status_flag#,address_label
if_tmr_off	<i>label</i> if_tmr_off timer_flag#,address_label
if_tmr_on	<i>label</i> if_tmr_on timer_flag#,address_label
set_flag	<i>label</i> set_flag user_flag
set_tmr <i>label</i>	set_tmrtimer_flag#,ticks
turn_off	<i>label</i> turn_off I/O flag#
turn_on	<i>label</i> turn_on I/O flag#

Figure 3.11 - Flag Instructions

Arithmetic Instructions

Several types of arithmetic functions are available, including:

1. 32 bit integer arithmetic
2. Array manipulation
3. Byte operations
4. Bit set and bit clear operations
5. Built in arithmetic functions

Each of these items will be explained in more detail below.

Integer Arithmetic

Program variables normally occupy 32 bits (4 bytes) of storage. Arithmetic statements operate on these 4 byte variables. The format for arithmetic instructions is similar to that of the **LET** statement used in BASIC:

label **let** **result = variable1 + variable2**

where **result** is the result of the operation, **variable1** and **variable2** are the variables to be operated on. The symbol for addition "+" is used for purposes of illustration. Note that multiple operations in a single **let** statement are *not allowed*.

For example,

label **let** **a = b + c**

is acceptable, but

label **let** **a = b + c / d**

is not.

Calculations requiring multiple operations must be performed using multiple **let** statements. Spaces between variable names and operators should not be used.

Array Manipulation

A data array is a group of 32 bit variables which can be referenced by the same variable name. Data arrays are usually defined using a **dim** instruction or by use of the **begin_data**, **data**, and **end_data** instructions. The **INSTRUCTION SUMMARY** provides additional information on these instructions.

For example, to set aside ten 32 bit storage locations to contain a table of positions, the instruction

```
positions      dim      10
```

could be used. Within the program, the first position in the table would be referred to as **positions[0]**. The value within brackets is known as the array subscript. Note that for a table of ten values, the array subscript ranges from zero to nine. Array subscripts can be a constant, an expression, or a variable.

Special forms of the **let** instruction serve to store information in and retrieve information from data arrays. In our position table example, the instruction

```
label      let      p=position[3]
```

would retrieve the fourth position from the array and store that value in the variable **p**. To store a value in a data array, the instruction

```
label      let      position[3]=x
```

would be used.

To use a value from an array in arithmetic instructions, it is necessary to retrieve the value into a non-array variable, perform the arithmetic operation, and then store the result back in the array.

Byte Operations

Special instructions for the manipulation of byte oriented data such as electronic cams (see Chapter 14) or strings of characters. This special instruction, the **let_byte**, is similar to the special case of the **let** statement for handling data arrays. For example, to access the fourth character of the text string **f_name**, the instruction

```
label      let_byte  a=f_name[3]
```

could be used. To store data into a byte oriented data array, the instruction format

```
label      let_byte  f_name[3]=a
```

would be used.

The **let_byte** instruction does not support any arithmetic operations. To perform arithmetic on byte oriented data, it is necessary to retrieve the byte into a conventional variable, perform the arithmetic, and store the result back into the byte array. The **let_byte** instruction treats byte information as unsigned.

Bit Oriented Operations

A set of instructions are available for setting (turning on) and clearing (turning off) individual bits within a 32 bit data value. These instructions can be useful when setting up variables for use with the **set_map** and **set_mcf** instructions. A complementary set of bit testing instructions (see Chapter 18) allow program branching to take place depending on the state of a single bit in a 32 bit data value.

Arithmetic Functions

A select group of arithmetic functions are included for commonly required calculations. These functions are listed in Table 9.1. The format of these functions is:

label **let** **result = function(x)**

where **function** represents the function name.

Function	Description
abs(x)	Returns the absolute value of x.
sqr(x)	Returns the square root of x. NOTE: If x is negative or zero, zero is returned.
neg(x)	Returns the two's complement of x.

Figure 3.12 - Arithmetic Functions

Arithmetic Instruction Summary

Instruction	Format
clr_bit	<i>label</i> clr_bit bit#,variable
let_byte	<i>label</i> let_byte destination=source
let	<i>label</i> let variable=op1 operation op1
set_bit	<i>label</i> set_bit bit#,variable

Figure 3.13 - Arithmetic Instructions

Program Flow Instructions

Normally, program instructions execute in an orderly, sequential fashion. Sometimes, it is desirable to alter this sequential order based on the status of an I/O module, the result of a calculation, the detection of a motor fault, or some other condition. Program flow instructions provide this capability.

There are three general classes of program flow instructions - branching, subroutine control, and the select statement. A special type of program flow instruction based on interrupts is described separately in Chapter 12.

Branching Instructions

There are two types of branching instructions - unconditional and conditional. The **goto** and **restart_at** instructions are the only unconditional branch instructions other than subroutine control instructions, which are covered separately.

Conditional branching instructions are handled as follows:

1. Test the specified condition.
2. If the condition is true, transfer program control to the specified address.
3. If the condition is false, continue by executing the next sequential instruction.

Conditional instructions can test the status of MSC flags, the result of an arithmetic comparison, the status of bits in a variable, or whether characters are present at a serial port.

Subroutine Control

When designing a program, it is often desirable to place a group of commonly used instructions into a unit called a subroutine. The subroutine can then be called from anywhere within the program. Subroutines must begin with a labeled statement. The label on this initial statement is often referred to as the subroutine name. Subroutines must end with a **return_sub** instruction.

The **gosub** instruction is then used to transfer program control to the subroutine. The subroutine instructions are then executed until the **return_sub** instruction is encountered. At that point, program control returns to the instruction immediately following the **gosub** instruction.

The Select Statement

Often, the process of controlling program flow is more complex than testing a flag and branching if the flag is set. For example, it may be necessary to choose among several alternatives based on a user entered menu selection. In cases like these, the **select** instruction group provides an effective means of controlling program flow.

Figure 3.14 illustrates how the **select** instruction might be used.

```
label      select      keynum
case      1
      .
      (statements to do when keynum = 1)
      .
exit_select

case      2
      .
      (statements to do when keynum = 2)
      .
exit_select

default
      .
      (statements to do when keynum does not match a
      case option)
      .
exit_select
end_select
```

Figure 3.14 - Select Statement Usage

A program may contain up to 32 **select** statement groups. Each group may contain up to 100 **case** instructions.

Program Branching Instructions

Instruction	Format
case	<i>label</i> case num
default	<i>label</i> default
end_select	<i>label</i> end_select
exit_select	<i>label</i> exit_select
gosub	<i>label</i> gosub subroutine label
goto	<i>label</i> goto address_label
if	<i>label</i> if var1 op var2,address_label
if_bit_clr	<i>label</i> if_bit_clr bit#,variable,address_label
if_bit_set	<i>label</i> if_bit_set bit#,variable,address_label
if_char	<i>label</i> if_char port#,address_label
if_flag_off	<i>label</i> if_flag_off user_flag#,address_label
if_flag_on	<i>label</i> if_flag_on user_flag#,address_label
if_io_off	<i>label</i> if_io_off I/O flag#,address_label
if_io_on	<i>label</i> if_io_on I/O flag#,address_label
if_no_char	<i>label</i> if_no_char port#,address_label
if_stat_off	<i>label</i> f_stat_off status_flag#,address_label
if_stat_on	<i>label</i> if_stat_on status_flag#,address_label
if_tmr_off	<i>label</i> if_tmr_off timer_flag#,address_label
if_tmr_on	<i>label</i> if_tmr_on timer_flag#,address_label
restart_at	<i>label</i> restart_at address_label
return_sub	<i>label</i> return_sub
select	<i>label</i> select variable

Figure 3.15 - Program Branching Instructions

Motion Instructions

Motion instructions are divided into five classes:

1. Pre-motion Preparation
2. Velocity Control
3. Incremental and Absolute Positioning
4. Piecewise Profiles
5. Master/Slave Operations

Position Data

MSC-850 and MSC-850/32 Controllers maintain position data as a signed 24 bit number. The least significant 12 bits of this number represent the position of the feedback transducer. The most significant 12 bits are used as a signed turns counter. The range of valid positions is from -2048 to +2047 turns.

MSC-250 Controllers maintain position data as a 32 bit number. The least significant 12 bits of this number represent the position of the feedback transducer. The most significant 20 bits are used as a signed turns counter. The range of valid positions is from -524,287 to +524,287 turns. Positive positions represent a displacement from zero in the clockwise direction.

Incremental distances are expressed in a manner similar to position data. For example, an incremental distance of two turns would be expressed as 8192 (2×4096). Positive distances represent clockwise motion.

Speed (Velocity) Data

MSC Controllers process speed data in RPM. Speeds may range from 0 to 3600 RPM (7200 RPM if using an MSC-250) in whole number increments. In special cases, speed values can be scaled down by a factor of 256 to provide fractional speed control.

Acceleration Data

Acceleration (accel/decel) data is expressed in revolutions/second². Accelerations may range from 2 to 800 revolutions/second² (1600 revolutions/second² if using an MSC-250). In special cases, acceleration values can be scaled down by a factor of 256 to provide very slow acceleration rates.

Global And Local Zeroes

MSC Controllers can maintain both a global and a local zero. Global zero usually refers to a fixed machine home position, and is often established by moving a load until it contacts a sensor. Local zero can be thought of as a floating home position. Local zero is often used when moving a known distance from the global zero, setting the local zero, and then performing a series of motions relative to the new zero position. The local zero may then be cleared.

Motion Preparation Instructions

Motion preparation instructions are used to condition a controller prior to executing any motion. They are used to turn on the drive (amplifier), to set speeds and acceleration rates, and to set and clear zero positions.

On power up, the ACR-850 Axis Controller sets its turns counter to zero. The low order 12 bits of its position will reflect the current resolver reading. The default settings for speed, acceleration, and digital compensation are activated.

On power up, ACE-850 Axis Controllers and MSC-250 Axis Controllers also set their turn counters to zero. The low order 12 bits of its position are also set to zero. For an ACE-850 Axis Controller, absolute position of the feedback transducer is not known until the controller is instructed to find the encoder marker pulse.

Instruction	Format
clr_local	<i>label</i> clr_local controller#
digi_comp	<i>label</i> digi_comp controller#,gain,integral,damp
drive_off	<i>label</i> drive_off controller#
drive_on	<i>label</i> drive_on controller#
f_decel	<i>label</i> f_decel controller#
find_mrk_ccw	<i>label</i> find_mrk_ccw controller#,counts
find_mrk_cw	<i>label</i> find_mrk_cw controller#,counts
find_tm_ccw	<i>label</i> find_tm_ccw controller#,counts
find_tm_cw	<i>label</i> find_tm_cw controller#,counts
master	<i>label</i> master controller#
set_ac_dc	<i>label</i> set_ac_dc controller#,rate
set_acy_cnt	<i>label</i> set_acy_cnt controller#,count
set_home	<i>label</i> set_home controller#,offset
set_gl_ccw	<i>label</i> set_gl_ccw controller#
set_gl_cw	<i>label</i> set_gl_cw controller#
set_local	<i>label</i> set_local controller#
set_speed	<i>label</i> set_speed controller#,speed
set_vgain	<i>label</i> set_vgain controller#,vel_gain

Figure 3.16 - Motion Preparation Instructions

Digital Compensation

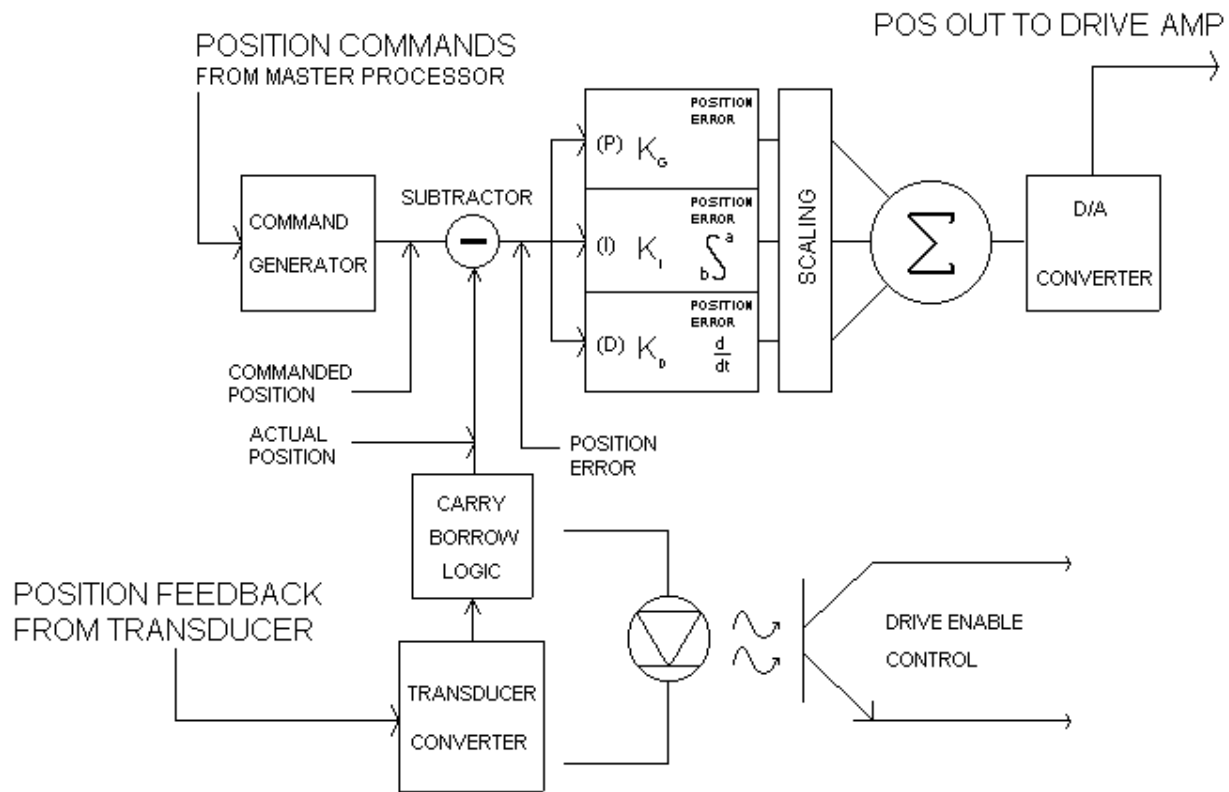


Figure 3.17 - PID Block Diagram

ACR-850, ACE-850 and MSC-250 Axis Controllers provide a means of digital compensation for their position loop. The programmer can override the default gain (P) term of 16, integral (I) term of 0, and differential (D) term of 0. These terms can be adjusted to customize system response, stability, and stiffness to perform in particular applications. Figure 3.17 illustrates the integration of PID into these controllers.

These terms are used in the following paragraphs:

- Stiffness:** the ability of the servo system to keep its position under loaded conditions.
- Response:** how fast the servo system will respond to changes in the load and/or commands.
- Stability:** the ability of the servo system to control the load under changing conditions and commands.

The P Term (Proportional Gain)

On power up the axis controller is configured as a proportional gain controller. The default gain value of 16 provides an overall system gain of 1 which will produce a POS OUT signal of 20 volts per revolution of the encoder shaft. This relationship can be varied by changing the P term.

$P = 256$ will produce a POS OUT signal of 20 volts per $1/16$ revolution of the encoder shaft.

$P = 1$ will produce a POS OUT signal of 20 volts per 16 turns of the encoder shaft.

This relationship can be used to increase/decrease the servo systems' response and stiffness.

The I Term (Integral)

The integral term is introduced into a servo control system when the proportional gain cannot compensate for steady-state errors. The actual amount of integral to be added is system dependent; the valid range is ± 127 . At power-up, the integral term is 0.

The I term introduces an anticipated error. During steady-state operation, it reduces the position error, providing more accurate position tracking. During acceleration/deceleration ramps, it can increase system responsiveness by tending to overshoot the commanded value.

The D Term (Differential)

The differential term is introduced into a servo control system to control the rate of change of the error signal. The D term has the effect of rounding the acceleration/deceleration ramps, reducing overshoot of the commanded value. The actual amount of D to be added is system dependent; the valid range is ± 127 . At power-up, the differential term is 0.

The D term benefits the system by reducing overshoot and providing more stability and better response by damping oscillations. Because the D term suppresses oscillations, the proportional gain term can be increased to provide a more responsive system.

Velocity Gain

The **set_vgain** command sets the velocity feed forward gain for the specified axis. During calculation of the position output value for the specified axis, the velocity feed forward gain term is multiplied by the current commanded velocity. The resultant value is added to the other digital compensation terms for that axis.

An approximate starting value for the velocity gain term may be calculated as follows:

$$Vg = \frac{K * 402,650}{BPR}$$

where K is the velocity scale factor for the motor/drive system, in volts per 100 RPM, and BPR is the number of transducer bits per revolution, after quadrature, of the motor shaft. For example, BPR would be 4096 for a 1024 line encoder.

Velocity Control Instructions

Velocity control instructions are used in applications where control of motor speed is the objective. Velocity control instructions are listed in Table 11.2.

MSC Controllers provide several different types of velocity control instructions. Choice of instruction is usually determined by the requirements of the application. Each of the different types is described below.

1. Jog instructions cause the axis controller to run its motor at a constant velocity until commanded to stop. Direction of rotation is determined by the instruction -- **jog_cw** or **jog_ccw**. Velocity may not be changed while the motor shaft is moving.
2. Track speed instructions allow on-the-fly speed changes. Unlike jog instructions, direction of rotation is determined by the sign of the speed variable, with positive values indicating clockwise rotation. Two forms of the track speed instruction are provided. The **track_spd** instruction covers speeds in the range of -3600 to 3600 RPM with a resolution of 1 RPM. The **l_track_spd** instruction covers speeds from -128 to +127.99 RPM in increments of 1/256 RPM.
3. Velocity instructions provide speed control with very low acceleration rates. Like the jog instructions, direction of rotation is determined by the instruction syntax. When a controller receives a **vel_cw** or **vel_ccw** command, it divides its present acceleration rate by 256 before using it. Speed, but not direction, may be changed while in velocity mode by issuing another velocity instruction with the new speed.
4. To stop a motor shaft, the **f_decel** instruction is used. This instruction causes the motor to decelerate to zero speed at the currently programmed deceleration rate.

Instruction	Format
f_decel	<i>label</i> f_decel controller#
jog_ccw	<i>label</i> jog_ccw controller#
jog_cw	<i>label</i> jog_cw controller#
l_track_spd	<i>label</i> l_track_spd controller#,speed
track_spd	<i>label</i> track_spd controller#,speed
vel_ccw	<i>label</i> vel_ccw controller#
vel_cw	<i>label</i> vel_cw controller#

Figure 3.18 - Velocity Control Instructions

Positioning Instructions

MSC Controllers support both absolute and incremental positioning. Absolute motions are made relative to the zero or home position established by the controller. Incremental motions are made relative to the current motor position. An absolute move is referred to as a **position**. An incremental move is called an **index**.

Instruction	Format
index	<i>label</i> index controller#,distance
position	<i>label</i> position controller#,abs_position

Figure 3.19 - Positioning Instructions

Piecewise Profiles

Piecewise Profiles provide a simple means of defining complex motion profiles as a series of speeds, accel/decel rates and distances.

MSC Controllers perform Piecewise Profiles according to the following guidelines:

- A. If the present motor speed is LESS than the speed specified by the next profile segment, then:
 1. Accelerate to the new speed.
 2. Continue at the new speed until the distance specified has been moved. NOTE - some of the distance is used during acceleration.
- B. If the present motor speed is GREATER than the speed specified by the next profile segment, then:
 1. Continue at the old speed until the specified distance less the distance needed to decelerate has been traveled.
 2. Decelerate to the new speed.

Building Profile Data Tables

Profile data tables are organized as a series of profile segments. Each profile segment consists of a target speed (in RPM), an acceleration value (in revs/second²) and a distance (in bits). The last profile segment in a profile must contain a zero speed. All profile segments in a given profile must contain direction values of the same sign.

Probably the most convenient method of creating Piecewise Profile tables would be to use the **data** statement. The following defines a three segment profile.

prodata	begin_data	
	data	500,100,6.2*4096
	data	1000,200,8.6*4096
	data	0,150,15.2*4096
	end_data	

The profile above can be described as follows:

1. Accelerate to 500 RPM at an acceleration rate of 100 revs/sec², continue until the motor has turned 6.2 revolutions (including acceleration distance).
2. Accelerate to 1000 RPM at an acceleration rate of 200 revs/sec². Continue at that speed until the motor has traveled 8.6 additional turns (including acceleration distance).
3. Continue at 1000 RPM as long as necessary, then decelerate to zero speed at a deceleration rate of 150 revs/sec². The Total distance traveled in this segment is 15.2 motor revolutions, including deceleration distance.

4. The total distance traveled in the profile is 30 revolutions.

The following program statements could be used to execute the segments discussed above.

Program Example

```

!
! ----- FLAG DEFINITION -----
!
CALCULATING    equ        90
BUSY           equ        94

!
! ----- SEGMENT DATA -----
!
pw_data        begin_data
               data        500,100,6.2*4096
               data        1000,200,8.6*4096
               data        0,150,15.2*4096
               end_data
               .
               .

!
! ----- CALCULATE THE PROFILE -----
!
ck_calc        prep_profile 1,prodata          ! send profile
               if_stat_on  CALCULATING,ck_calc ! wait for calcs
               get_pstat   1,status
               if          status<>0,calc_error ! error routine
               .

!
! ----- BEGIN RUNNING THE PROFILE -----
!
wt_busy        exec_profile 1                  do the profile
               if_stat_on  BUSY,wt_busy        wait til done
               .
               (instructions performed when execution is ok)
               .

!
! ----- ERROR HANDLER -----
!
calc_error     if          status=0,profile_ok    all zero = OK
               let_byte    segment=status[3]     get segment in error
               let_byte    error=status[2]       get error code

```

.
(process errors here)
.
.

Piecewise Profiles may also be calculated in the program and stored in an array in the appropriate format. This format is always:

speed, accel, distance	Segment #1
speed, accel, distance	Segment #2
.	.
.	.
.	.
speed, accel, distance	Segment #x

The maximum number of profile segments allowed in a Piecewise Profile is 96. Note that the last segment in a Piecewise Profile must have a zero speed.

In the MSC-850/32, MSC-850, MSC-800 and MSC-250 Controllers, Piecewise Profile data shares the same memory area as cam data. Piecewise profile data is always placed in the axis controllers memory starting at location zero.

Piecewise Profiles And Master Slave

It is possible to send a Piecewise Profile to an axis controller card and to instruct the controller card to execute the profile whenever another axis controller reaches a specified angular position. Refer to the section on **LOCK METHOD 3** in **MASTER SLAVE CONCEPTS** for a discussion of this technique.

Instruction	Format
exec_profile	<i>label</i> exec_profile controller#
get_pstat	<i>label</i> get_pstat controller#,status
prep_profile	<i>label</i> prep_profile controller#,array_label
set_trig_pw	<i>label</i> set_trig_pw controller#,master_angle

Figure 3.20 - Piecewise Profile Instructions

Reading Controller Position

MSC Controllers provide instructions for reading the current transducer position of the axis controllers, as well as the position of the "pseudo axis." It is possible to read two types of position data from an axis controller:

1. Actual Position - The **get_pos** instruction returns the actual position of the corresponding transducer in the format described in section 11.2.1.
2. Commanded Position - The **get_com** instruction returns the commanded position for the specified controller. Commanded position is usually slightly different than actual position. The difference is referred to as following error.

NOTE:

The "pseudo axis" commanded position is always equal to actual position.

Instruction	Format
get_com	<i>label</i> get_com controller#,variable
get_fol_err	<i>label</i> get_fol_err controller#,variable
get_pos	<i>label</i> get_pos controller#,variable

Figure 3.21 - Instructions to Read Controller Position

Master Slave Concepts

MSC-850/32, MSC-850 and MSC-250 Controllers provide a mode of operation whereby the motion of one axis, called a master axis, can be used to control the motion of one or more slave axes. Two data paths are provided, each allowing for a single master axis and multiple slave axes. The data paths can be extended over multiple MSC Controller using the Fiber Optic Network feature (See Section 13.9).

Slave Axes use the Master Axis position to determine their own position and speed according to various methods described in subsequent sections of this chapter.

In the MSC-850/32 and MSC-850, each Master Axis transmits its current position transducer reading onto the designated data bus every millisecond. In the MSC-250, this occurs every 488 microseconds. Each Slave Axis listening on that bus will retrieve this position and use it to determine its own position. This high speed data rate allows very accurate tracking and smooth operation.

There are currently three modes of operation for the slave axes. In simple lock, or "electronic gearbox" mode, a simple ratio is applied to the master position/speed data and the results used to directly determine the proper slave axis position and speed. In electronic cam mode, the slave axis moves through a pre-programmed path (the electronic cam) dependent on the rotation of the master axis. In Piecewise lock, the slave executes a Piecewise Profile each time the master axis reaches a specified angle. Each of these modes is explained in detail below.

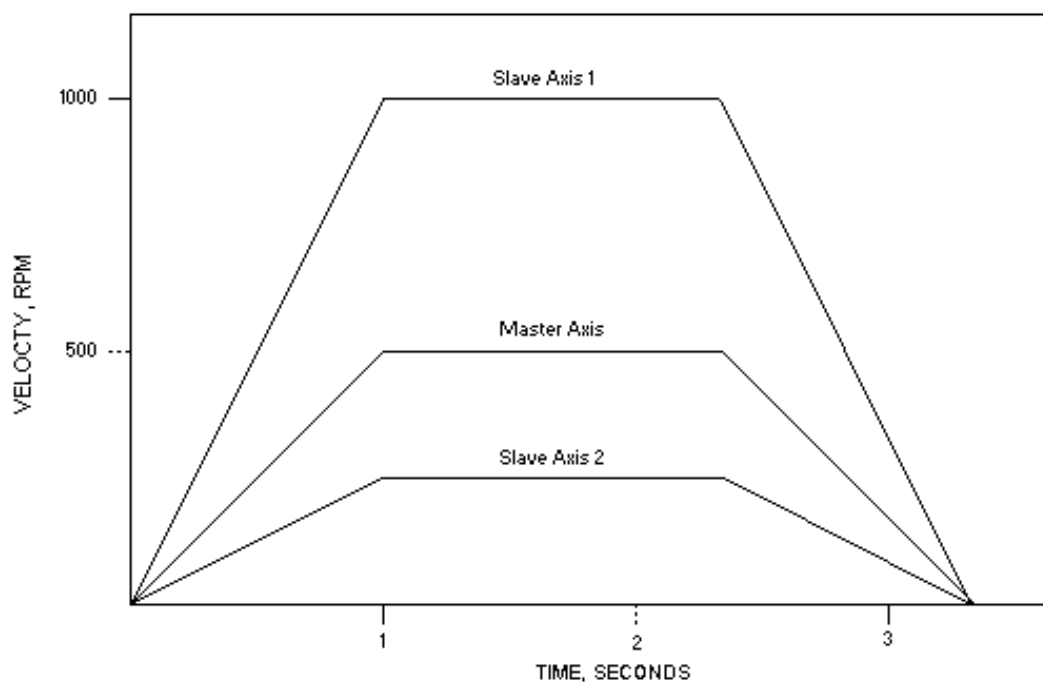


Figure 3.22 - Simple Lock Illustration

Simple Lock (Electronic Gearbox)

In simple lock mode, the position of the master axis is used directly to determine the proper slave axis position and speed. Figure 3.22 illustrates simple lock mode. In this example, the master axis and slave axes were positioned to zero before entering master slave lock. Slave axis 1 has a ratio of 2, and slave axis 2 has a ratio of 0.5.

NOTE:

Note the motion profile of each slave in relationship to the master. Slave axis 1 always travels at twice the speed of the master. The distance traveled by slave 1 is twice that of the master. Slave 2 travels at half the speed of the master and moves half the distance.

Useful Facts About Simple Lock Mode

1. Ratios are treated as 12 bit fractions. To express a ratio of 1.0, the actual argument in the ratio instruction would be 4096.
2. Allowed values for ratio are from -8.000 to +7.9999 (-32768 to +32767 bits). Positive ratios cause a slave axis to turn in the same direction as the master; negative ratios cause counter rotation.
3. When a program is started, the default ratio value for each axis is to one-to-one (+4096 bits).
4. The recommended sequencing for establishing master/slave lock is as follows:
 - A. If necessary, position the master and/or slaves to the desired locations.
 - B. Establish the accel/decel rates for the slaves. For certain lock methods, a high acceleration rate is needed to insure that the slave axis closely maintains the configured ratio between itself and the master axis.
 - C. Issue a **set_map** instruction to start the information flow between the master and the slave.
 - D. Lock the slave axis using the simple lock mode (mode 1).
 - E. Set the desired ratio for the slave axis.
 - F. Proceed with the desired motion for the master axis.Steps D through E can be used to lock on to a master axis already in motion.
5. A typical sequence for ending simple lock is:
 - A. Unlock the slave axis using the **unlock** instruction.

- B. Set the ratio for the slave axis to zero.
 - C. Terminate the master angle passing process by issuing a **set_map** instruction with no source axis.
 - D. Determine that the slave axis has stopped by testing its **BUSY** status flag.
- 6. Ratio may be changed at any time during simple lock mode. The slave axis will simply break lock, accelerate or decelerate at the currently programmed rate to match the master speed at the new ratio, and re-enter lock.
 - 7. While in master/slave lock, a slave axis will ignore all motion instructions except forced deceleration. Issuing a **f_decel** instruction will break lock and cause the slave axis to stop.

Lock Methods For Simple Lock

There are four lock methods which can be used in simple lock applications. Each method and its function are outlined in Figure 3.23.

Lock Method	Description
1	Simple lock with acceleration limit. Slave tracks master position as long as currently set accel/decel rate is not exceeded.
2	Velocity Lock. Slave tracks master velocity, with accel/decel rate limit.
4	Simple lock with no acceleration limit.
6	Keyway to Keyway lock. Ratio is forced to be 1.0. On receipt of lock command, slave axis aligns its position transducer to match that of master. Accel/decel rate limit is used.

Figure 3.23 - Lock Methods for Simple Lock

Details for each of these lock methods are presented below.

Lock Method 1

Lock Method 1 provides the electronic equivalent of a gearbox. The electronic gearbox or ratio is driven by the master angle from one of the master angle buses. The master angle is processed by the previously specified **ratio** instruction and an offset is added, resulting in an effective master used to drive the slave command position.

When the **lock** instruction is executed, with a lock method of 1, the axis controller calculates the instantaneous offset between the master angle processed by the ratio and the slave command position. Once the offset is calculated, master slave lock is accomplished. Once per millisecond (MSC-850) or once every 488 microseconds (MSC-250) thereafter, the slave axis position is updated based on the new master position.

The slave motion is limited by the previously executed **set_ac_dc** instruction and further limited to a maximum speed of 3600 RPM. The limiting of the slave acceleration rate can be an advantage in systems that have rough or rapidly changing master speeds. The limited acceleration on the slave dampens the slave motor and smooths out the operation.

The **ratio** instruction can be executed while in Lock Method 1 Simple Ratio. When the **ratio** instruction is executed, the slave controller stops executing the equation above and reverts to simply slewing at the **set_ac_dc** rate until the slave reaches the new slave speed as if lock were in place. When the speed is matched, a new offset is calculated and the equation above is executed.

It is important to note that after a **ratio** instruction is executed, a new offset is used. The **JOGGING** status flag is on while the slave motor is changing from one speed to another. This flag goes off when the ratio lock equation starts executing.

The acceleration rate may be changed during the slew from one speed to another caused by a ratio change. A **set_ac_dc** prior to the **ratio** instruction will cause the slew acceleration rate to change.

The limiting of slave speed and acceleration result in some interesting situations. For example, if the master is turning when the **lock** instruction is executed, the slave motor will accelerate at the specified acceleration rate until the equation above is satisfied. In actual practice the slave motor will accelerate above what would appear to be the final speed in order to make up the angular phase difference lost during acceleration. The resulting motion profile is shown in Figure 3.24.

Since the slave is limited in both speed and acceleration, a situation may occur where the slave can never reach final lock speed. The same situation can occur if the effective master is changing speed faster than the slave can change speed because of the acceleration limit. As a rule of thumb, the slave acceleration rate should be set to at least 5 times the expected rate of change of the effective master.

Moving Master Profile

Lock Method 1

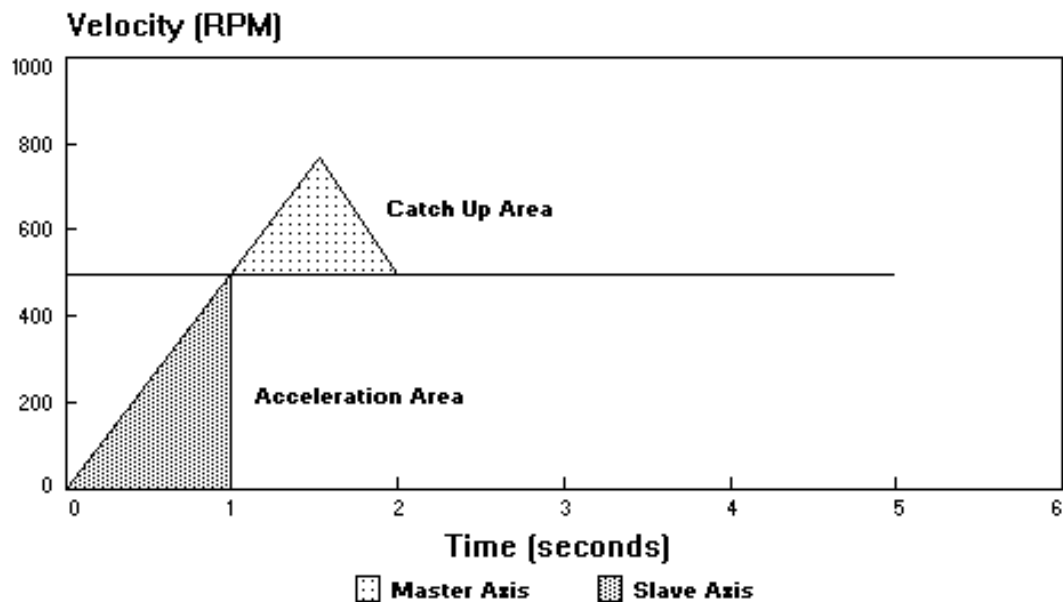


Figure 3.24 - Lock on a Moving Master

Lock Method 4

Lock method 4 is identical to lock method 1, except that the slave is not limited by the specified slave acceleration rate. This means that during lock, the slave is commanded directly by the effective master position.

NOTE:

Any perturbations or roughness in the master will be passed onto the slave with no rate limiting.

The **ratio** instruction can be executed during Lock Method 4. When a new **ratio** is executed, the slave controller breaks lock, slews to new lock speed at the specified acceleration rate and relocks using the above equation. The **JOGGING** status flag is on during the slew period. Note that after the slew period a new offset is calculated. The acceleration rate may be changed during slew by executing a **set_ac_dc** instruction prior to the **ratio** instruction.

Lock Method 6

Lock method 6 allows the user to align the absolute position of the slave with the absolute position of the master. The ratio in Lock Method 6 is fixed at 1:1 and may not be changed. The slave controller acceleration rate is limited by the previously executed **set_ac_dc** instruction. The slave top speed is also limited to 3600 RPM.

When the **lock** instruction is executed, the slave controller executes the following equation every 1 millisecond.

$$\text{Master Angle} = \text{Slave Command Position}$$

This equation is executed modulo 1 turn (4096 bits) of the master. This means that when the **lock** instruction is executed, the slave will move, at the acceleration rate specified, in the shortest direction to bring the master and slave absolute angles into alignment. Note that Lock Method 6 can cause movement when **lock** is executed even if the master is at rest.

The slave acceleration and speed are limited. Hunting can occur if the master rate of change is greater than the specified slave acceleration rate. The rule of thumb of having the slave acceleration rate at least 5 times the expected master rate of change is applicable. The limited acceleration rate can be used to smooth a master with fast perturbations (roughness).

Electronic Cams

The MSC multi-axis controllers provide a mode of operation that effectively emulates mechanical cams. In the electronic cam mode, slave axes follow digital cams based on the rotation of a master axis.

Electronic cams are tables of incremental motor moves. Each incremental value occupies one byte of memory in the MSC and has an allowable range of -127 to +127. (A value of 128 is used to signal the end of the cam). As the master axis turns, its position is continually transmitted onto the data bus. The slave controller uses the current master position to determine the proper index into the electronic cam array. As the index moves through the cam table, the incremental distances are added together to form the proper slave axis position. Linear interpolation is performed between elements in the cam array. When the end of the cam table is reached, the process begins again at the beginning of the table.

The key MSC macro instruction for electronic cam mode is the **cam_data** instruction.

This instruction transmits the electronic cam from the MSC master processor to the specified axis controller and establishes the scaling factors for master position data and the cam array data. Master and data scaling are explained in the paragraphs below.

Master Scaling

Master scaling provides a mechanism to control the rate at which a slave axis proceeds through its cam array relative to the rate at which the master axis turns.

Positional information within the MSC is treated as a 24 bit binary number. The least significant half (12 bits) of the number refers to the position within the current motor revolution, and the most significant half serves as a turn counter.

In electronic cam mode, the master scale factor refers to the number of times the master position value is divided by 2 (i.e. shifted right) by the slave processor. For example, if a master scale factor of 12 were used, the slave processor would divide the master position value by 4096 (2^{12}). The slave processor then compares the least significant 12 bits of the scaled value to the previous scaled master position to determine if it has changed. If the value has changed, the cam array index is changed accordingly.

Figure 3.25 summarizes the effect of master scaling.

Master Scale Factor	Cam Array Advances Every
12	full turn
11	1/2 turn
10	1/4 turn
9	1/8 turn
8	1/16 turn
7	1/32 turn
6	1/64 turn
5	1/128 turn
4	1/256 turn
3	1/512 turn
2	1/1024 turn
1	1/2048 turn
0	1/4096 turn

Figure 3.25 - Master Scaling

Data Scaling

In order to insure that each element of an electronic cam is in the range of -127 to +127, it is often necessary to scale the cam data. This is accomplished by dividing the data by 2^n where n is selected to cause the largest element of the cam array to be less than -127 to +127.

For example, if the largest element in a cam array is 864, we would need to divide by 8 (2^3) to reduce the number below 128. The data scale factor in this case would be 3. Data scale factors from zero to 7 are allowed.

Important Notes Regarding Electronic Cams

The following items should be considered when using electronic cams (Note - These examples assume a positive ratio value is being used):

1. Positive cam data indicates to the slave axis that the slave should travel in the same direction as the master axis.
2. The cam data pointer will always start at the top of the cam table, unless the **set_cam_ptr** instruction and lock method 5, 8 or 9 are used.
3. The cam data pointer will move toward the cam data table terminator when the master axis is traveling in the clockwise (CW) direction. The cam data pointer will move away from the cam data table terminator when the master axis is traveling in the counter-clockwise (CCW) direction.

Calculating Electronic Cams

There are several ways that electronic cam data can be transferred to the appropriate axis controller:

1. The cam data can be precalculated and placed in cam tables using the instructions **begin_cam**, **cam** and **end_cam**. When the **cam_data** instruction is executed, the data defined in the cam table will be transferred to the axis controller along with the 'master scale' and 'data scale' values. The cam table terminator character (hex value 0x80) will be automatically transferred by the **cam_data** instruction, as well.
2. The cam data can be calculated by the program, or off-line by another computing device. This data can then be transferred to the axis controller using a series of **let_byte** instructions. The programmer in this instance is responsible for sending the cam table terminator character (hex value 0x80) as the last cam data table element. When the **cam_data** instruction is executed, only the 'master scale' and 'data scale' values are transferred to the axis controller, since it is expected that the data has been previously transmitted.
3. The cam data table can be calculated **by the axis controller** using the **calc_unit_cam** instruction. When using this approach, a data table defining the 'shape' of the profile is transmitted to the axis controller. The programmer then issues a **calc_unit_cam** instruction which will calculate the cam data table. When the **cam_data** instruction is executed, only the 'master scale' and 'data scale' values are transferred to the axis controller, since it is expected that the data has been previously calculated.

In applications where it might be desirable to compute an electronic cam based on a predetermined shape, such as a modified sine or trapezoidal type motion profile, the **calc_unit_cam** instruction can be used to greatly simplify this process.

The general process for using the **calc_unit_cam** instruction is as follows:

DEFINE THE MOTION SHAPE BY CREATING A 'UNIT CAM' TABLE

1. Either graphically or mathematically describe the motion profile desired. The profile should be a function of distance versus machine angle or other appropriate X-axis value.
2. Measure and tabulate the value of distance at 128 equally spaced X-axis intervals over the duration of the profile.
3. Normalize the distance values by dividing each tabulated value by the largest value in the table.
4. Scale each table value by multiplying it by 65,536 (2^{16}).
5. Add a zero element to the beginning of the table and replicate the last element in the table, so that the resulting table has 130 elements.

These steps result in a 'unit cam' table, which can then be used to generate cams of various lengths and number of elements.

TRANSFER THE 'UNIT CAM' TABLE TO THE CONTROLLER

6. Dimension data storage areas on the axis controller of interest as follows:

cam_area	dim	axis#,6750
table_area	dim	axis#,130

The labels 'cam_area' and 'table_area' may be named whatever you desire. The dimension numbers must be as shown. It would also be acceptable to have more than one 'cam_area' defined, as long as the total is equal to 6750 (this is 6750 4-byte elements totaling 27000 bytes). This is due to the fact that the axis controller expects to find the start of the 'table_area' at memory location 6750.

7. Load the 'unit cam' table into the axis controllers 'table_area' using the **let** instruction (the 130 entries in the 'unit cam' table are 4-byte values).

CALCULATE THE CAM

8. Determine the total distance to be represented by the cam, and the total number of cam elements required.
9. Issue the appropriate **calc_unit_cam** instruction. While the axis controller is busy calculating the cam, its status flag for **CALCULATING PW PROFILE** is set.

NOTE:

It is not necessary to issue a **drive_on** instruction before executing the **calc_unit_cam** instruction.

10. Insert an end of cam character (hex value 0x80) at the appropriate place in the cam array.
11. Issue a **cam_data** instruction so that the appropriate 'master scale' and 'data scale' are used.

At this point, the axis controller contains a valid electronic cam. Processing can now follow normal cam procedures.

The instruction format for **calc_unit_cam** is as follows:

label calc_unit_cam axis#,distance,elements

where 'axis#' is the axis controller number, 'distance' is the total distance in counts to be traveled and 'elements' is the size of the cam to be generated.

The following example illustrates the **calc_unit_cam** instruction and the steps taken to load the predetermined 'unit cam' table named 'trap_1_3' into the axis controllers memory:

Program Example

```

                                msc_type      850/32
                                declare      off

!
! ----- FLAG DEFINITION -----
!
CALC          equ          74
DOWN          equ          77
BUSY          equ          78
LOCK          equ          79
AXIS_1        equ          1
BUSY_1        equ          (AXIS_1*16)+BUSY
DOWN_1        equ          (AXIS_1*16)+DOWN
CALC_1        equ          (AXIS_1*16)+CALC
LOCK_1        equ          (AXIS_1*16)+LOCK

cam           dim          1,6750
table         dim          1,130

!
! ----- TABLE FOR UNIT TRAPEZOIDAL CURVE 1/3,1/3,1/3 -----
!
trap_1_3      begin_data
              data          0,9,36,81,144,225,324
              data          441,576,729,900,1089,1296,1521
              data          1764,2025,2304,2601,2916,3249,3600
              data          3969,4356,4761,5184,5625,6084,6561
              data          7056,7569,8100,8649,9216,9801,10404
              data          11025,11664,12321,12996,13689,14400,15129
              data          15876,16640,17408,18176,18944,19712,20480
              data          21248,22016,22784,23552,24320,25088,25856
              data          26624,27392,28160,28928,29696,30464,31232
              data          32000,32768,33536,34304,35072,35840,36608
              data          37376,38144,38912,39680,40448,41216,41984
              data          42752,43520,44288,45056,45824,46592,47360
              data          48128,48896,49660,50407,51136,51847,52540
              data          53215,53872,54511,55132,55735,56320,56887
              data          57436,57967,58480,58975,59452,59911,60352
              data          60775,61180,61567,61936,62287,62620,62935
              data          63232,63511,63772,64015,64240,64447,64636
              data          64807,64960,65095,65212,65311,65392,65455
              data          65500,65527,65536,65536
              end_data

```

```
!  
! ----- LOAD THE UNIT CAM TABLE INTO AXIS CONTROLLER #1 -----  
!  
load_table    let          ctr=0  
load_loop     let          t=trap_1_3[ctr]  
              let          table[ctr]=t  
              let          ctr=ctr+1  
              if           ctr<130,load_loop  
  
!  
! ----- CALCULATE THE CAM BASED ON UNIT CAM TABLE -----  
!  
              let          distance=40960  
              let          elements=2048  
              let          start_of_cam=0  
  
              calc_unit_cam AXIS_1,distance,elements,start_of_cam  
calc_busy     if_stat_on    CALC_1,calc_busy  
  
!  
! ----- CALCULATE THE CAM SUM FOR THE CAM -----  
!  
              let          end_of_cam=elements-1  
  
              calc_cam_sum AXIS_1,start_of_cam,end_of_cam  
sum_busy     if_stat_on    CALC_1,sum_busy  
  
!  
! ----- GET SUM, SHOULD EQUAL REQUESTED CAM DISTANCE -----  
!  
              get_cam_sum  AXIS_1,sum  
              if           sum<>distance,calc_err  
  
!  
! ----- INDICATE THE CAM TO USE, MASTER SCALE AND DATA SCALE -----  
!  
              cam_data     AXIS_1,cam,5,0  
              ratio        AXIS_1,4096
```

Electronic Cam Lock Methods

Lock Method Description

0	Cam lock at beginning of cam data table.
5	Cam lock at current cam pointer position. Used in conjunction with the set_cam_ptr instruction to lock at other than the beginning of the cam data table.
8	Cam Lock at current cam pointer position when specified master angle is crossed. (See set_trig_cam)
9	Same as Lock type 8 except execution of the cam is terminated at the last element in the cam.
10	Velocity cam lock.

Figure 3.26 - Lock Methods for Cam Lock

A detailed explanation of each of these lock methods follows.

Lock Method 0

When the **lock** instruction is executed with a lock method of 0, the axis controller puts the cam pointer at the very top of the cam array, creates an instantaneous offset between the absolute master angle, on the selected bus, and 0.00 effective master, then locks the axis to the master. Once per millisecond after the **lock** instruction, the following equation is executed to drive the cam pointer.

$$\frac{(\text{Absolute master angle} * \text{ratio})}{(2^{\text{master scale}})} + \text{offset} = \text{effective master}$$

As this equation executes, the integer part of the effective master causes the cam pointer to move and the fractional part is used to linearly interpolate cam elements. The offset value in the equation is set at **lock** to yield a 0.00 effective master and is modified each time the cam pointer wraps around to maintain the cam pointer within the cam data array.

If the ratio is changed during **lock** in Lock Method 0, the offset is instantaneously changed to yield the same effective master just before as just after the **ratio** instruction.

Changing the ratio on the fly during Lock Method 0 does not cause a jump in the cam pointer, but does cause a change in rate or direction of cam pointer movement. Since the slave speed and acceleration rate are not limited, the slave servo may change speed abruptly. Caution should be used when changing **ratio** when executing a cam.

During Cam Lock Method 0, the pointer position in the cam data table can be captured using the **get_cam_ptr** instruction.

Lock Method 5

Lock Method 5 is identical to Lock Method 0, except that the cam pointer is not moved to the top of the cam array when the **lock** instruction is executed. The cam pointer is started at the beginning of the cam element wherever the cam pointer was previously left by either previous execution of the cam or a **set_cam_ptr** instruction execution.

Lock Method 8

Lock Method 8 causes execution of a cam to be executed when a specified master angle is crossed. Execution begins at the current cam pointer (see **set_cam_ptr**).

Lock Method 9

Lock Method 9 is identical to lock method 8 except execution of the cam is terminated when the last element in the cam table is executed.

Lock Method 10

Lock Method 10 allows switching from the normal cam lock to velocity cam lock, but only during cam execution. This position output is no longer based on distance but varies with speed changes in the cam table, based on a constant calculated at the time the lock command is issued. This lock mode can only be terminated by executing an **unlock** command in mode 0 or by doing a **f_decel** command. Executing the **lock** command in mode 10 will disable the position loop, following error test, and digital compensation. The subsequent **unlock** command in mode 0 (or **f_decel**) will zero the position output and re-enable the position loop, following error test and digital compensation.

Sample Electronic Cam Application

An application requires a slave axis to follow a motion described by the mechanical cam shown in Figure 3.27. It has been determined that 16 cam data points will provide sufficient resolution for this application. The master axis is known to make 4 complete rotations for one cycle of the cam.

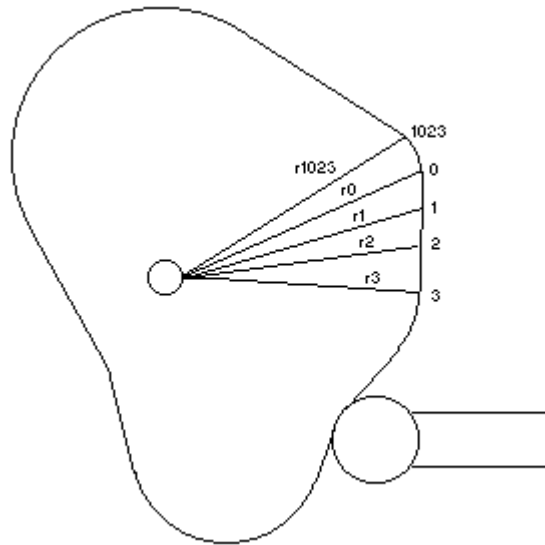


Figure 3.27 - Mechanical Cam

To implement this application, the following procedure could be used:

1. Measure the radius of the cam at each of 16 equal angle increments around the cam.
2. Convert data to incremental form by subtracting each data point from its successor.
3. Convert the data from inches to position transducer units by applying the proper scaling factor (4096 in this case.)
4. Examine each cam array to find the largest value. Use this value to determine the data scale factor needed to scale the data to the range of -127 to +127.

5. Divide each cam array element by the appropriate value (2^n , where n is the data scale factor).
6. Use a master scale factor of 10 (1/4 turn of master = 1 cam element).

The resulting arrays can then be entered into a program to produce the desired motion. Figure 3.28 shows a graphic representation of the resulting cam motion.

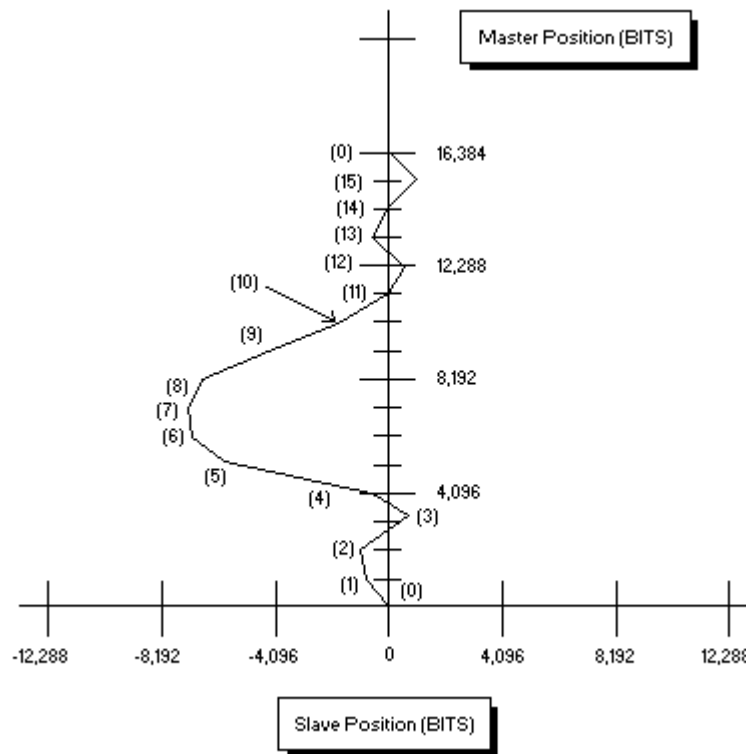


Figure 3.28 - Resulting Master/Slave Motion

Piecewise Lock

Piecewise lock, lock method 3, provides a means of triggering execution of a Piecewise profile in a slave axis based on the master reaching a certain angular position. Refer to Section 11.7 for further details on Piecewise profiles.

To establish Piecewise lock, follow the steps outlined below.

1. Build the profile as described in Section 11.7.
2. Establish appropriate master angle passing.
3. Issue the **prep_profile** command to transfer the profile data to the slave axis card.
4. Set up the trigger angle with the **set_trig_pw** instruction.
5. Issue the **lock** command with lock method set to 3.

When the specified master angle position is reached, the Piecewise profile will be executed.

Master Angle Bus

The master angle bus is a high speed data highway which links the controllers in an MSC-850, MSC-850/32 or MSC-250 Controller. Each controller has two independent buses, designated as Master Bus A and Master Bus B. The **set_map** program instruction provides a means for the programmer to specify which controller talks as a master on a given bus and which controller(s) listen as slaves.

There can only be one controller as master on a bus. Any controller may be configured as a slave. One slave controller may not be configured to listen to both masters, except for the MCF-850 card which can listen to both.

The **set_map** instruction uses a 4 byte word (32 bits) to define the total configuration. In the 32 bit variable, two (2) bytes are used for each bus. One byte configures which controller is the master and the other byte configures which controllers are the slaves. The **set_bit** instruction provides a convenient means of setting the proper bits in the **set_map** variable.

Figure 3.29 outlines the configuration of the **set_map** variable for the MSC-850/32 and MSC-850. Figure 3.30 shows the configuration of the **set_map** variable for the MSC-250.

BIT	AXIS	FUNCTION	MASTER ANGLE BUS
31	1	MASTER/TALKER	A
30	2		
29	3		
28	4		
27	5		
26	6		
25	7		
24	8		
23	1	SLAVE/LISTENER	
22	2		
21	3		
20	4		
19	5		
18	6		
17	7		
16	8		
15	1	MASTER/TALKER	B
14	2		
13	3		
12	4		
11	5		
10	6		
9	7		
8	8		
7	1	SLAVE/LISTENER	
6	2		
5	3		
4	4		
3	5		
2	6		
1	7		
0	8		

Figure 3.29 - MSC-850 and MSC-850/32 Master Angle Bus Configuration

BIT	AXIS	FUNCTION	MASTER ANGLE BUS
31	1	MASTER/TALKER	A
30	2		
29	3		
28			
27			
26			
25			
24			
23	1	SLAVE/LISTENER	
22	2		
21	3		
20			
19			
18			
17			
16			
15	1	MASTER/TALKER	B
14	2		
13	3		
12			
11			
10			
9			
8			
7	1	SLAVE/LISTENER	
6	2		
5	3		
4			
3			
2			
1			
0			

Figure 3.30 - MSC-250 Master Angle Bus Configuration

Consider an example where the controller in slot 3 is to be the master axis, and controllers in slots

1, 4, and 5 are to be slaves. Master Angle Bus A will be used for the data path. The following program instructions could be used to configure the master angle bus:

Program Example

```

!
! ----- DEFINE BITS USED TO CONFIGURE THE MASTER ANGLE BUS  STRUCTURE
-----
!
MASTER_3  equ          29                Slot 3 talker on Bus A
SLAVE_1   equ          23                Slot 1 listen on Bus A
SLAVE_4   equ          20                Slot 4 listen on Bus A
SLAVE_5   equ          19                Slot 5 listen on Bus A

!
! ----- INITIALIZE THE MASTER ANGLE BUSES - NO ACTIVITY  -----
!
                let          mapvar=0
                set_map      mapvar

!
! ----- TURN ON THE NECESSARY BITS IN THE VARIABLE mapvar -----
!
                set_bit      MASTER_3,mapvar
                set_bit      SLAVE_1,mapvar
                set_bit      SLAVE_4,mapvar
                set_bit      SLAVE_5,mapvar

!
! ----- CONFIGURE THE BUS STRUCTURE USING THE VARIABLE mapvar -----
!
                set_map      mapvar

```

Master Angle Bus Cautions

It is important to note that the data being transmitted on a master angle bus is the current master **position transducer reading**. Program instructions such as **set_local** or **set_offset** DO NOT affect the data being transmitted on a master angle bus, but DO impact the logical position of an axis. Thus, it is possible to read a position from a master controller using a **get_pos** instruction and get a position value which does not match the value being transmitted on the master angle bus.

Another precaution regarding master angle bus data concerns ACE-850 controllers. Because encoders are incremental devices, the ACE-850 sets its position transducer reading to zero on power up. Zero would be transmitted on the master angle bus if the ACE-850 were commanded to be a master under these circumstances. If a **find marker** instruction was then issued to the ACE-850, data being sent on the master angle bus would undergo a step change as the ACE-850 located

the encoder marker pulse. It is recommended that the **find marker** instruction be done for an encoder based master controller before the controller is instructed to transmit on the master angle bus.

Fiber Optic Network

The Fiber Optic Network feature of the MSC-850 provides a means of extending Master Angle Bus communications to multiple MSC Controllers. This function is provided by the MCF-850 Axis Controller.

The MSC-250 has one fiber optic receiver. This allows a link between an MSC-850/MCF-850 and an MSC-250.

BIT	SOURCE	DESTINATION
23	FIBER OPTIC NETWORK B	Pseudo Axis
22	FIBER OPTIC NETWORK A	
21	MASTER ANGLE BUS B	
20	MASTER ANGLE BUS A	
19	FIBER OPTIC NETWORK B	Programmable Limit Switch
18	FIBER OPTIC NETWORK A	
17	MASTER ANGLE BUS B	
16	MASTER ANGLE BUS A	
15		Master Angle Bus B
14	PSEUDO AXIS	
13	FIBER OPTIC NETWORK B	
12	FIBER OPTIC NETWORK A	
11		Master Angle Bus A
10	PSEUDO AXIS	
9	FIBER OPTIC NETWORK B	
8	FIBER OPTIC NETWORK A	
7	PSEUDO AXIS	Fiber Optic Network B
6	FIBER OPTIC FEED THRU B	
5	MASTER ANGLE BUS B	
4	MASTER ANGLE BUS A	
3	PSEUDO AXIS	Fiber Optic Network A
2	FIBER OPTIC FEED THRU A	
1	MASTER ANGLE BUS B	
0	MASTER ANGLE BUS A	

Figure 3.31 - MCF-850 Multi-Function Controller Configuration

BIT	SOURCE	DESTINATION
31	PLS IS EXPANDER #2	
30	PLS IS EXPANDER #1	
29	PLS IS ON BOARD	
28		
27		
26		
25		
24		
23		
22		
21		
20		
19		Programmable Limit Switch
18	FIBER OPTIC RECEIVER	
17	MASTER ANGLE BUS B	
16	MASTER ANGLE BUS A	
15		Master Angle Bus B
14	PSEUDO AXIS	
13		
12	FIBER OPTIC RECEIVER	
11		Master Angle Bus A
10	PSEUDO AXIS	
9		
8	FIBER OPTIC RECEIVER	
7		
6		
5		
4		
3		
2		
1		
0		

Figure 3.32 - MSC-250 Multi-Function Controller Configuration

Each MCF-850 Axis Controller has two fiber optic communications ports. Each port consists of a transmitter and a receiver. The MCF-850 must be configured, using the **set_mcf** instruction, to complete the desired data paths. Figure 3.31 shows the bit assignments used to define the network data paths.

Each MSC-250 Axis Controller has a single fiber optic communications port. This port contains a single receiver. There is port must be configured similar to the way the MCF-850 is configured using the **set_mcf** instruction. Figure 3.32 shows the bit assignments used to define the network data paths.

Consider the example diagrammed in Figure 3.33. In this example, three MSC-850 Controllers are linked in a daisy chain fashion. The location of master axes and communication interrelationships are shown in the figure. The following program segments could be used to establish the desired network configuration.

Program Example - MSC Controller #1

```

!
! ----- DEFINE BITS USED TO CONFIGURE THE MASTER ANGLE BUS
STRUCTURE -----
!
MASTER_3  equ      29                Card 3 talker on Bus A
SLAVE_1   equ      23                Card 1 listen on Bus A
SLAVE_4   equ      20                Card 4 listen on Bus A
SLAVE_5   equ      19                Card 5 listen on Bus A
MABA_FOCA equ      0                MAB A is the source for FOC A
MCF_CARD  equ      6                Card 6 is a MCF-850

!
! ----- DEFINE THE MULTI-FUNCTION CONTROLLER CARD -----
!
          let      mcf_var=0
          set_bit   MABA_FOCA,mcf_var
          set_mcf   MCF_CARD,mcf_var

!
! ----- INITIALIZE THE MASTER ANGLE BUSES - NO ACTIVITY -----
!
          let      mapvar=0
          set_map   mapvar

!
! ----- TURN ON THE NECESSARY BITS IN THE VARIABLE mapvar -----
!

```

```

set_bit    MASTER_3,mapvar
set_bit    SLAVE_1,mapvar
set_bit    SLAVE_4,mapvar
set_bit    SLAVE_5,mapvar

```

```

!
! ----- CONFIGURE THE BUS STRUCTURE USING THE VARIABLE mapvar -----
!
      set_map      mapvar

```

Program Example - MSC Controller #2

```

!
! ----- DEFINE BITS USED TO CONFIGURE THE MASTER ANGLE BUS STRUCTURE

```

```

!
MASTER_1      equ      15      Card 1 talker on Bus B
SLAVE_2       equ      6       Card 2 listen on Bus B
SLAVE_3       equ      5       Card 3 listen on Bus B
FOCA_FTHRU    equ      2       FOC A feedthru
MABB_FOCBequ  5           MAB B is the source for FOC B
MCF_CARD      equ      4

```

```

!
! ----- DEFINE THE MULTI-FUNCTION CONTROLLER CARD -----
!

```

```

      let      mcf_var=0
      set_bit  FOCA_FTHRU,mcf_var
      set_bit  MABB_FOCB,mcf_var
      set_mcf  MCF_CARD,mcf_var

```

```

!
! ----- INITIALIZE THE MASTER ANGLE BUSES - NO ACTIVITY -----
!

```

```

      let      mapvar=0
      set_map  mapvar

```

```

!
! ----- TURN ON THE NECESSARY BITS IN THE VARIABLE mapvar -----
!

```

```

      set_bit  MASTER_1,mapvar
      set_bit  SLAVE_2,mapvar
      set_bit  SLAVE_3,mapvar

```

```

!
! ----- CONFIGURE THE BUS STRUCTURE USING THE VARIABLE mapvar -----

```



```
!
      set_map      mapvar
```

Program Example - MSC Controller #3

```
!
! ----- DEFINE BITS USED TO CONFIGURE THE MASTER ANGLE BUS STRUCTURE
-----
!
SLAVE_1      equ      15      Card 1 listen on Bus B
SLAVE_2      equ      14      Card 2 listen on Bus B
SLAVE_3      equ      13      Card 3 listen on Bus B
FOCA_MABA    equ      8       FOC A is the source for MAB A
FOCB_MABB    equ      13      FOC B is the source for MAB B
MCF_CARD     equ      4       Card 6 is a MCF-850

!
! ----- DEFINE THE MULTI-FUNCTION CONTROLLER CARD -----
!
      let          mcf_var=0
      set_bit      FOCA_MABA,mcf_var
      set_bit      FOCB_MABB,mcf_var
      set_mcf      MCF_CARD,mcf_var

!
! ----- INITIALIZE THE MASTER ANGLE BUSES - NO ACTIVITY -----
!
      let          mapvar=0
      set_map      mapvar

!
! ----- TURN ON THE NECESSARY BITS IN THE VARIABLE mapvar -----
!
      set_bit      SLAVE_1,mapvar
      set_bit      SLAVE_2,mapvar
      set_bit      SLAVE_3,mapvar

!
! ----- CONFIGURE THE BUS STRUCTURE USING THE VARIABLE mapvar -----
!
      set_map      mapvar
```

Master/Slave Instructions

Instruction	Format
calc_cam_sum	<i>label</i> calc_cam_sum controller#,start_element,ending_element
calc_unit_cam	<i>label</i> calc_unit_cam controller#,distance,# elements,start_element
cam_data	<i>label</i> cam_data controller#,label,ms,ds
get_cam_cnt	<i>label</i> get_cam_cnt controller#,variable
get_cam_end	<i>label</i> get_cam_end controller#,variable
get_cam_ptr	<i>label</i> get_cam_ptr controller#,variable
get_cam_strt	<i>label</i> get_cam_strt controller#,variable
get_map	<i>label</i> get_map variable
get_map_stat	<i>label</i> get_map_stat variable
get_mcf	<i>label</i> get_mcf controller#,variable
incr_com	<i>label</i> incr_com controller#,bits,interrupts
lock	<i>label</i> lock controller#,lock#
ratio	<i>label</i> ratio controller#,ratio
set_cam_ptr	<i>label</i> set_cam_ptr controller#,value
set_map	<i>label</i> set_map variable
set_mcf	<i>label</i> set_mcf controller#,variable
set_trig_cam	<i>label</i> set_trig_cam controller#,master_angle
switch_cam	<i>label</i> switch_cam controller#,start_element
unlock	<i>label</i> unlock controller#,mode#

Figure 3.33 - Master/Slave Instruction Summary

Programmable Limit Switches

MSC-850/32, MSC-850 and MSC-250 Controllers provide a means of switching output modules on and off in relationship to the position of a master axis. This function, called Programmable Limit Switch (PLS), is similar to that provided by mechanical cam or drum switches. The MCF-850 Multi-Function Controller and the MSC-250 Controller provides basic PLS capabilities. For applications with high performance requirements, the HPL-850 High Speed Programmable Limit Switch controller is better suited. Each of these options is described in detail below.

MSC-850/MCF-850 PLS Functions

One of the features of the PLS-850 Controller and the MSC-250 is the Programmable Limit Switch feature. There are 24 PLS output flags available. These flags can be switched on and off, depending on the position of the master angle being used to drive the PLS function. These flags consist of 16 hardware modules and 8 software outputs. In the MCF-850, the 16 hardware modules are located on a PLS-850 rack. Four of the eight software outputs are flags, and can be used to trigger software interrupts. The state of all 24 flags can be monitored using the **get_pls_out** instruction.

The users' program specifies set points (angles) of the master angle bus where the 24 flags should be turned ON and OFF. During program execution, this angular position is monitored, and the outputs are turned ON and OFF according to the programmed set points.

MSC-250 PLS Functions

There are sixteen PLS output flags available on the MSC-250. All sixteen can be used as hardware outputs but only the last eight have software flags associated with them.

Usage: `set_pls_ang 3,on_angle,off_angle,module #` where module # is in the range of 0-15.

If module # is specified between 0 and 7, then it is a hardware output only - no software flags are associated with it.

If module # is specified between 8 and 15, then it is a hardware/software output. Eight software flags are associated with these 8 module #s and the I/O that it covers cannot be used for I/O. An output module is not needed if it is to be used only as a software PLS.

It is important to perform a `set_mcf` instruction before using the `set_pls_ang` command. In byte [0] of the `set_mcf` variable, set only one bit designating which set of I/O the 16 possible PLS module #s will be associated with.

If you use the `set_pls_mask` instruction, keep in mind it only masks on/off a pls module # that was previously set; it does NOT clear that module #. This means after issuing a `set_pls_ang` for a given module #, that I/O associated with it cannot be used as an I/O, even if you mask it off. Cycling power is the only way to free up that I/O.

Also, after each `set_pls_ang` instruction, you must have a loop to wait for the calculating flag to go off.

Programming using PLS's

Before attempting to write a program using the Programmable Limit Switch feature of the MSC, it is important to understand the following:

1. Only one ON and OFF angle may be programmed for each output. If the controller receives a second set of data for an output, it will replace the existing data with new data.
2. The ON and OFF set points are interpreted assuming a clockwise direction of rotation. For example, ON at 1000, OFF at 2000, means that an output will be on whenever the master angle position is between 1000 and 2000 bits. However, ON at 2000, OFF at 1000, means that an output will be off between 1000 and 2000 bits, and on for the remainder of the master rotation.
3. The switching of individual output modules may be masked off without reprogramming the PLS function.

Processing of programs using PLS's

When the controller receives a `set_pls_ang` instruction, it searches the programmed data for an existing record for the specified output. If a matching record is found, it will be replaced by the new data.

While this instruction is being processed, the CALCULATION IN PROGRESS flag of the axis controller will be activated. The program must verify that this flag is no longer active before executing subsequent `set_pls_ang` instructions. The axis processor will ignore any `set_pls_ang` instructions issued while the CALCULATION IN PROGRESS flag is activated.

Execution of programs using PLS's

The controller monitors data from a master angle bus and turns ON and OFF the outputs at programmed set points. No change in state from OFF to ON will occur if a particular switch output is disabled through the use of the `set_pls_mask` instruction. Changes in state from ON to OFF will still occur at the programmed set point even if a particular limit switch is disabled.

NOTE:

The angular position can come from one of several sources, as programmed by the `set_mcf` instruction.

MSC-850 / MCF-850

- A. Pseudo Axis
- B. Master Angle Bus A
- C. Master Angle Bus B
- D. Fiber Optic Channel A
- E. Fiber Optic Channel B

MSC-250

- A. Pseudo Axis
- B. Master Angle Bus A
- C. Master Angle Bus B
- D. Fiber Optic Receiver

When the MSC is powered down, all of the PLS (Programmable Limit Switch) data is lost. This data must be reset on power up by the program.

MSC-250 Programming Example using PLS's

We will use expander #1 by setting bit 30 in the set_mcf variable. Five hardware PLS outputs and six software PLS outputs will be used. Also various I/O will be used in different places to show the flexibility when using PLS outputs in the MSC-250. The I/O will be set as follows:

0-15	Onboard I/O
16	Hardware PLS
17	Hardware PLS
18	Hardware PLS
19	I/O
20	Hardware PLS
21	Hardware PLS
22	I/O
23	I/O
24	Software PLS
25	Software PLS
26	Software PLS
27	I/O
28	I/O
29	Hardware/Software PLS
30	Hardware/Software PLS
31	Software PLS

The macrolanguage instructions necessary to accomplish the above example would be:

```

!
!-----CONSTANT DEFINITION-----
!
CALC      equ          122              ! calculating flag on pseudo axis

          let          mcf-var = 0
          set_bit      30,mcf_var
          set_mcf      3,mcf_var

!

```

```

!-----SET THE ON/OFF ANGLES-----
!
      set_pls_ang      3,on_ang,off_ang,0      ! hardware pls
wait_1  if_stat_on      CALC,wait_1

      set_pls_ang      3,on_ang,off_ang,1      ! hardware pls
wait_2  if_stat_on      CALC,wait_2

      set_pls_ang      3,on_ang,off_ang,2      ! hardware pls
wait_3  if_stat_on      CALC,wait_3

      set_pls_ang      3,on_ang,off_ang,4      ! hardware pls
wait_4  if_stat_on      CALC,wait_4

      set_pls_ang      3,on_ang,off_ang,5      ! hardware pls
wait_5  if_stat_on      CALC,wait_5

      set_pls_ang      3,on_ang,off_ang,8      ! software pls
wait_6  if_stat_on      CALC,wait_6

      set_pls_ang      3,on_ang,off_ang,9      ! software pls
wait_7  if_stat_on      CALC,wait_7

      set_pls_ang      3,on_ang,off_ang,10     ! software pls
wait_8  if_stat_on      CALC,wait_8

      set_pls_ang      3,on_ang,off_ang,13     ! hardware/software pls
wait_9  if_stat_on      CALC,wait_9

      set_pls_ang      3,on_ang,off_ang,14     ! hardware/software pls
wait_10 if_stat_on      CALC,wait_10

      set_pls_ang      3,on_ang,off_ang,15     ! software pls
wait_11 if_stat_on      CALC,wait_11

```

High Performance Programmable Limit Switch (MSC-850/HPL-850)

The High Performance Programmable Limit Switch controller (HPL-850) is intended as a replacement for mechanical limit switches. Its function is to monitor angular position received on one of two master angle data buses, and to turn on and off outputs at programmed set points. The HPL-850 was designed for systems requiring multiple master turns per 360 degree cycle. The maximum number of turns per 360 cycle is 256. The HPL-850 is also equipped with a time advance feature. This allows the programmer to advance the on and off settings for an output by a specified time.

There are 24 outputs which may be turned on and off at set points specified in a program. Sixteen of these are outputs residing on PLS-850 output rack. The remaining 8 switches are software flags. The first four of these may be programmed to generate software interrupts in the program.

The application program specifies the source of the master angle data (master angle bus A or B), establishes the set points of the master angle where switches are to be turned ON and OFF, and sets any time advances to be used. During program execution, the HPL-850 monitors master angle position and turns on and off switches according to the programmed set points.

Theory Of Operation of PLS's

The HPL-850 has no direct position sensor. The input angle data must be supplied by one of two Master Angle Data Buses.

The programmer specifies a rollover point for the HPL. This value represents the point at which the HPL's accumulator will reset. For example, if a particular process completes one cycle in 4.5 turns of a master resolver, the rollover point could be specified as 4.5×4096 , or 18432. The accumulator value may be initialized to a value between zero and the rollover point set by the **preset** instruction.

An internal data table holds a series of 24 bit values representing the state of the 24 output flags at each of 8192 possible accumulator values. This table is created from the **set_pls_ang** instructions issued in the application program.

The HPL-850 uses its accumulator value, appropriately scaled, to create a pointer into this data table. The data table contents are then sent to the HPL-850 output functions (16 outputs and 8 user flags).

Programming Considerations for PLS's

Before attempting to write a program using the Programmable Limit Switch function of the HPL-850, it is important to understand the following:

1. Only one ON and OFF angle may be programmed for each output. If the HPL-850 receives a second set of data for an output, it will replace the existing data with new data.
2. The ON and OFF set points are entered assuming a CW direction of the angle.
3. The **set_pls_mask** instruction may be used to disable or enable output switching without reprogramming the PLS function.
4. The master angle bus data source (set_mcf) must be configured before issuing a **set_gl_cw**, **set_gl_ccw** or **set_local** instruction.
5. The time advance feature is meant to be used on machines with steady velocity inputs. Output results can be difficult to predict during accel/decel curves due to changing velocity.
6. The time advance parameters should never allow the output signal to advance into the next cycle.

HPL-850 Programming Example #1 using PLS's

The Master Angle Bus A is fed by a resolver rotating one revolution for one machine cycle. The system requires an output to be on between 160° and 200°. This output fires an air cylinder which has a 50 millisecond time delay, requiring the output signal to be advanced by 50 milliseconds.

Example Program

```
!
! ----- CONSTANT DEFINITIONS -----
!
HPL_CARD equ          3      ! HPL-850 card is in slot 3
HPL_CALC  equ          122    ! Indicates a calculation at HPL-850 card
PLS_CNT   equ          4096   ! 4096 counts per machine cycle

!
! ----- PLS ON/OFF ANGLES DEFINED -----
!
ON_ANGLE  equ          (4096*160)/360
OFF_ANGLE equ          (4096*200)/360

!
! ----- DEFINED I/O MODULES -----
!
```



```

AIR_CYL    equ        0      ! module 0 is air cylinder output
TIME_ADV   equ        5      ! advance 5 timer ticks = 50 ms.

.
.

        set_mcf        HPL_CARD,1
        set_pls_cnt    HPL_CARD,PLS_CNT

!
! ----- SET THE ON/OFF ANGLES -----
!
        set_pls_ang    HPL_CARD,ON_ANGLE,OFF_ANGLE,AIR_CYL
wait_1     if_stat_on   HPL_CALC,wait_1

!
! ----- SET THE TIME ADVANCE FOR THE AIR_CYL MODULE -----

        set_pls_time   HPL_CARD,TIME_ADV,AIR_CYL
wait_2     if_stat_on   HPL_CALC,wait_2

!
! ----- CLEAR THE HPL-850 ACCUMULATOR -----
!
        set_local      HPL_CARD

```

HPL-850 Programming Example #2 using PLS's

A system drives a lead screw 26 revolutions. An oil mist solenoid is to come on between 14 and 21 revolutions. Master Angle Bus data is on Bus B. The following program excerpt could be used for this application.

Example Program

```

!
! ----- CONSTANT DEFINITIONS -----
!
HPL_CARD    equ        3      ! HPL-850 card is in slot 3
HPL_CALC    equ        122    ! Indicates a calculation at HPL-850 card
PLS_CNT      equ       26*4096 ! counts per machine cycle

!
! ----- PLS ON/OFF ANGLES DEFINED -----
!
ON_ANGLE     equ       14*4096
OFF_ANGLE    equ       21*4096

```

```
!  
!----- DEFINED I/O MODULES -----  
!  
AIR_CYL          equ      0  ! module 0 is air cylinder output  
TIME_ADV         equ      5  ! advance 5 timer ticks = 50 ms.  
  
                .  
                .  
  
        set_mcf    HPL_CARD,2  
        set_pls_cnt HPL_CARD,PLS_CNT  
  
!  
!----- SET THE ON/OFF ANGLES -----  
!  
        set_pls_ang HPL_CARD,ON_ANGLE,OFF_ANGLE,AIR_CYL  
wait_1    if_stat_on HPL_CALC,wait_1  
  
!  
!----- SET THE TIME ADVANCE FOR THE AIR_CYL MODULE -----  
  
        set_pls_time HPL_CARD,TIME_ADV,AIR_CYL  
wait_2    if_stat_on HPL_CALC,wait_2  
  
!  
!----- CLEAR THE HPL-850 ACCUMULATOR -----  
!  
        set_local  HPL_CARD  
  
                .  
                .
```

Programmable Limit Switch Instructions

Instruction	Format
get_angle	<i>label</i> get_angle controller#,variable
get_for_ang	<i>label</i> get_for_ang controller#,channel#,variable
get_pls_mask	<i>label</i> get_pls_mask controller#,variable
get_pls_out	<i>label</i> get_pls_out controller#,variable
preset	<i>label</i> preset controller#,variable
set_gl_ccw	<i>label</i> set_gl_ccw controller#
set_gl_cw	<i>label</i> set_gl_cw controller#
set_local	<i>label</i> set_local controller#
set_mcf	<i>label</i> set_mcf controller#,variable
set_pls_ang	<i>label</i> set_pls_ang controller#,on,off,module#
set_pls_cnt	<i>label</i> set_pls_cnt controller#,count
set_pls_mask	<i>label</i> set_pls_mask controller#,variable
set_pls_time	<i>label</i> set_pls_time controller#,time,module#

Figure 3.34 - Programmable Limit Switch Instructions

Interrupts

It is often necessary for an MSC Controller (and program) to respond quickly to an external event, such as a switch closure or an operator input. The MSC programming language provides two methods for responding to this type of event.

Software Interrupts provide a means for a program to respond to changes in state of any MSC flag. Hardware Interrupts provide a means for an axis controller to perform a preprogrammed task immediately on receipt of an input signal.

Software Interrupts

The Software Interrupts feature of the MSC provides for automatic program response to the change in state of any MSC flag (I/O, timer, motor status or user).

This feature functions by allowing the programmer to associate the change of state of a flag with a program subroutine. Whenever the specified change occurs, the MSC makes a note of what it was doing, and then transfers control to the associated subroutine just as though a **gosub** instruction was performed. When the subroutine is completed, control passes back to the interrupted operation.

In the MSC-850/32 and MSC-850, Software Interrupts are checked between the execution of program instructions. Therefore, program activities such as downloading large cams can hold off the recognition of software interrupts.

In the MSC-250, Software Interrupts are checked once every millisecond.

Software Interrupt processing provides up to 32 prioritized interrupts ranging from priority 0 (highest) to 31 (lowest). If more than one interrupt occurs at the same time, the one with the LOWEST event number (highest priority) will be recognized first. The user may disable all or selected software interrupts during portions of the program which should not be "interrupted".

Software Interrupt processing is edge triggered -- the specified subroutine will be executed only on the leading or trailing edge of the transition, depending on the type of interrupt specified. For example, if a software interrupt is specified to occur whenever input module number 1 is ON and software interrupt processing is enabled while input module number 1 is already ON, no software interrupt will occur. The interrupt will only occur when the transition from the OFF to the ON state occurs.

Hardware Interrupts

In some cases, it is desirable for an axis controller to respond very rapidly to an external event. For these instances, MSC-850/32, MSC-850 and MSC-250 Controllers provide a feature called **Hardware Interrupts**.

Each axis controller is directly connected to the corresponding I/O Module Controller slot. For example,

Controller #1 is linked to I/O-1 (flag 0)

Controller #2 is linked to I/O-2 (flag 1)

•
•
•
•

Controller #8 is linked to I/O-8 (flag 7)

Hardware Interrupts can be processed much faster than conventional Input/Output handling since the Main Processor distributes the desired task to the axis controller before the interrupt occurs rather than waiting for the Input module to be activated, then distributing the task or instruction. Because of this, Hardware Interrupts are capable of 1 millisecond response times.

The following is a list of the tasks or instructions which can be executed by an axis controller when the corresponding hardware interrupt signal is detected:

trap_pos	lock
over_draw	ratio
index	exec_profile
position	f_decel

Each instruction above is used with the Hardware Interrupt feature by issuing an **enable_hwi** instruction, followed by the instruction to be executed when the interrupt is detected.

NOTE:

When using the **trap_pos** instruction with a controller using encoder for feedback (ACE-850, MSC-250), the marker correction feature is disabled until the hardware interrupt is recognized.

The following example would instruct controller #1 to index 2048 bits when input module I/O-1 changes from a low to high state:

label	enable_hwi	
	index	1,2048

Execution of this pair of instructions causes the Main Processor to instruct the appropriate controller card that it should now monitor its Hardware Interrupt signal and perform the designated operation when the interrupt signal is activated. The axis controller responds by enabling its **BUSY** and **HARDWARE INTERRUPT ARMED** flags. After the controller detects the interrupt

and completes the task, its **BUSY** and **HARDWARE INTERRUPT ARMED** flags will be disabled. It is the programmers responsibility to issue this pair of instructions again if it is desired to perform the task again.

The programmer can cancel an armed task by issuing the instruction **disable_hwi**. This instruction has 1 parameter - the slot # of the Controller card which is currently monitoring the hardware interrupt signal. On receipt of the **disable_hwi** instruction, the Controller's **BUSY** and **HARDWARE INTERRUPT ARMED** status flags will be turned off.

Only 1 task can be executed per interrupt. If more than 1 task is issued, the most recent will be used.

Interrupt Instructions

Instruction	Format
clr_swi	<i>label</i> clr_swi interrupt#
clr_all_swi	<i>label</i> clr_all_swi
disable_swi	<i>label</i> disable_swi
disable_hwi	<i>label</i> disable_hwi controller#
enable_hwi	<i>label</i> enable_hwi
enable_swi	<i>label</i> enable_swi
f_decel	<i>label</i> f_decel controller#
get_trap_pos	<i>label</i> get_trap_pos controller#,variable
over_draw	<i>label</i> over_draw controller#,speed,limit,distance
set_swi_mask	<i>label</i> swt_swi_mask variable
swi_if_off	<i>label</i> swi_if_off interrupt#,flag,sub_label
swi_if_on	<i>label</i> swi_if_on interrupt#,flag,sub_label
trap_pos	<i>label</i> trap_pos controller#

Figure 3.35 - Software and Hardware Interrupt Instructions

Extended Memory Operations

The MSC-850/32 and MSC-850 controllers provide three types of extended memory - volatile RAM, non-volatile RAM, and EPROM memory. The MSC-250 provides two types of extended memory - volatile RAM and EPROM memory.

EPROM memory is accessed through the PROM Pocket built into each MSC Controller.

Extended RAM

Regarding the MSC-850/32 or MSC-850 Controllers, the ACE-850, ACR-850 and ACY-850 Axis Controllers have 28K bytes of volatile data memory available for use by a program.

In the MSC-250, axes 1 and 2 have 28K bytes of volatile data memory each. This memory can be used to create very high resolution electronic cam data tables.

The MCF-850 Axis Controller provides 32K bytes of non-volatile data memory. This memory can be used to store data which must be retained even when power is shut off.

Extended RAM Programming

To create a data array in axis controller memory, a special form of the **dim** instruction is used:

SYNTAX:

label	dim	controller#,words
-------	-----	-------------------

PARAMETERS:

label	Name assigned to block of memory (array).
controller#	Axis ID #.
words	Length of data memory allocated to the label. Allocated in 4 byte (32 bit) words.

Once this special form of the **dim** instruction has been executed, it is not necessary to reference this area in memory by Axis ID#. Values in these arrays are accessed using the **let_byte** and **let** instructions in the same manner as any standard array.

WARNING:

Piecewise profiles and the conventional form of the **cam_data** instruction use memory from the 28K volatile RAM area. It is the responsibility of the programmer to ensure that arrays declared with the extended form of the **dim** statement do not overlap with piecewise profiles or conventional cams.

Extended Memory Limitations

1. The **prep_profile** instruction for Piecewise Profiles may not reference data stored in axis controller memory. Data for this type of profile must reside in the master controller.
2. The **cam_data** instruction can only reference data residing in the master controller or in axis controller memory which is the target of the **cam_data** instruction.
3. Depending upon the location of the data array, the **cam_data** instruction executes differently. If the array is in the master controller's memory, the **cam_data** instruction transmits the array to the specified axis controller, establishes pointers to the beginning and end of the cam array, and defines both the master scale and data scale factors. If the array is in the axis controller's memory, the **cam_data** instruction only establishes pointers to the beginning and end of the array, and the master and data scale factors. No data transfer is executed.

EPROM Memory

EPROM memory on the MSC is controlled through a series of program instructions called the EPROM MANAGER. These instructions are used to read from and write to an EPROM chip contained in the PROM POCKET. They are designed to simplify access to EPROM memory and are patterned after similar commands in the BASIC language.

In the MCF-850 Axis Controller, the use of non-volatile data storage greatly diminishes the need to have a programming device connected to the MSC-850 and can allow increased program size.

The PROM POCKET uses an INTEL D27256-1 UV erasable programmable ROM (EPROM) or compatible, and provides 32K bytes of memory. Each MCF-850 also provides 32K bytes of data storage.

Automatic Program Load From EPROM

On power-up, the MSC will check for the presence of a program PROM in the PROM socket. If one exists, that program will be loaded, run and the AUTOSTART flag will be set in the MSC Controller.

This sequence will take place even if there is a program currently running in the MSC and even if that program has been set to AUTOSTART.

EPROM Status Codes

Each EPROM Manager instruction returns a status code that indicates the result of the instruction.

CODE MEANING

0	Operation was successful
1	No EPROM present
2	End of file
3	File not found
4	Duplicate file name
5	Verify during write failed
6	EPROM Full
7	No file open/created
8	Bad unit identifier (<1 or >8)
9	Wrong mode (i.e. trying to write to an opened file)
10	Bad file type (i.e. trying to open a program file)
11	Unit busy - file already opened/created using this unit identifier
12	Attempt to create more than one file at a time
13	The sum of the 'program' and 'data' area exceeds 32,000 bytes

Figure 3.36 - EPROM Status Codes

EPROM Manager Instructions

Instruction	Format
close	<i>label</i> close unit,status
create	<i>label</i> create unit,f_n,status
get_space	<i>label</i> get_space unit,space,status
get_volume	<i>label</i> get_volume unit,d_a,status
load	<i>label</i> load unit,f_n,status
initialize	<i>label</i> initialize unit,d_a,status
open	<i>label</i> open unit,f_n,status
read	<i>label</i> read unit,d_a,l,status
save	<i>label</i> save unit,f_n,status
write	<i>label</i> write unit,d_a,l,status

Figure 3.37 - EPROM Manager Instructions

Analog Input/Output

MSC-850/32 and MSC-850

The ACM-850 Analog Control Module provides an MSC application with eight analog input and four analog output channels. The ACM-850 might be used in applications like a joystick interface, controlling DC motor drives, or monitoring analog sensors.

MSC-250

The MSC-250 has one analog input and one analog output channel.

Capabilities of Analog Input/Output

Data range - 12 bit conversion over a range of -10 to +10 volts is provided. The data range is -2048 (-10 volts) to +2047 (+10 volts).

Offset - Any of the 12 channels may be offset. The offset value is added to the real channel value at the time of conversion. The channel offsets range from -2048 to +2047. The default value is 0.

Slew rate - An analog rate of change limit may be set for each channel. This rate of change limit is "calibrated" in bits per 10 milliseconds. The default value is +2047.

ACM-850 Functional Description

Analog processing uses a 10 millisecond update cycle in which all analog inputs and outputs are updated. In the ACM-850 updates are done sequentially by channel number.

INPUT channels are read-only. Each input channel is read, and the raw voltage signal is added to the corresponding channel offset value. The result of this addition is compared to the previous reading for that channel, and if necessary, limited by the slew rate limit currently in effect.

OUTPUT channels are updated using the same algorithm as INPUT channels. That is, the offset is added and then the rate of change limit is applied. This limited result is then output.

Power-Up States for Analog I/O Channels

On power up, or after a **RESET** command from the MacroPro Analyzer, the analog I/O channels are in the following state:

- 1. Offsets are set to zero (0).
- 2. Slew rate limits are set to 2047.
- 3. Output channel values are set to zero.

ACM-850 Instructions

Instruction	Format
analog_in	<i>label</i> analog_in controller#,ch#,value
analog_out	<i>label</i> analog_out controller#,ch#,output
analog_rt	<i>label</i> analog_rt controller#,ch#,value
analog_zo	<i>label</i> analog_zo controller#,ch#,value

Figure 3.38 - ACM-850 Instructions

User Serial Ports

The MSC family of controllers support a number of serial communication ports. The following is a summary of the ports available for each controller and a short description of each:

<u>MSC TYPE</u>	<u>PORT NUMBER</u>	<u>DESCRIPTION</u>
MSC-800	0	User programmable active/passive current loop port.
MSC-800	1	RS-232C Executive serial port.
MSC-800	2	User programmable passive current loop port.
MSC-850	0	User programmable active/passive current loop port.
MSC-850	1	RS-232C Executive serial port.
MSC-850	2	User programmable passive current loop port.
MSC-250	1	RS-232C/RS-485 Executive serial port.
MSC-250	2	User programmable passive current loop.
MSC-250	3	User programmable RS-232C serial port.
MSC-850/32	0I	User programmable active/passive current loop port.
MSC-850/32	1	RS-232C/RS-485 Executive serial port.
MSC-850/32	2	User programmable passive current loop.
MSC-850/32	0R	User programmable RS-232C serial port.

An **active** current loop port indicates that this port powers the current loop.

A **passive** current loop port indicates that the user device will power the current loop.

The **executive** port is an RS-232C serial port that only supports the MSC Packet Protocol method of communication. This port is typically used by the MacroPro Development System for the loading and testing of MSC programs.

The **user programmable** ports are suitable for use with a variety of computer displays, hand-held terminals, strip displays, printers, data entry terminals, etc.

Serial Port Initialization

Before a serial port can be used, it is necessary to tell the MSC controller which port to use and the desired communication parameters. The **port_set** instruction is used for this purpose. The format of this instruction is:

label **port_set** **port,baud,protocol**

Depending on the MSC Controller being used; the **port** number will be either a 0, 1, 2, 3, 0I or 0R, the **baud** rate can be 110, 300, 600, 1200, 2400, 4800, 9600, 19200 or 38400.

The **protocol** variable is used to select the proper combination of parity, stop bits and XON/XOFF selection, according to Figure 3.39.

NOTE: If no parity is chosen, then the MSC assumes 8 bit data words. If parity is chosen, the MSC assumes 7 bit data words.

Protocol	Description
0	1 stop bit, no parity, XON/XOFF disabled
1	2 stop bits, no parity, XON/XOFF disabled
2	1 stop bit, odd parity, XON/XOFF disabled
3	2 stop bits, odd parity, XON/XOFF disabled
4	1 stop bit, even parity, XON/XOFF disabled
5	2 stop bits, even parity, XON/XOFF disabled
6	RESERVED
7	RESERVED
8	1 stop bit, no parity, XON/XOFF enabled
9	2 stop bits, no parity, XON/XOFF enabled
10	1 stop bit, odd parity, XON/XOFF enabled
11	2 stop bits, odd parity, XON/XOFF enabled
12	1 stop bit, even parity, XON/XOFF enabled
13	2 stop bits, even parity, XON/XOFF enabled

Figure 3.39 - Communication Protocol Selection

Once a **port_set** instruction has been executed, it remains in effect until another is issued.

Once a serial port has been initialized, **print**, **print_num**, and **input** instructions may be issued.

Important Notes Regarding Serial Ports

The following factors should be considered when using the MSC serial ports.

1. Serial port instructions have been implemented so that program execution does not need to be delayed while waiting for characters to be transmitted or received. This leads to the following considerations:

- a. **Print** and **print_num** instructions are queued up to be executed by a separate task. Several milliseconds may pass between the execution of the instruction and the actual transmission of characters. The program sequence:

```

label      let      x=10
           print_num 4,0,x
           let      x=25

```

could result in "25" being sent out the serial port instead of "10". It may be necessary to use temporary variables to yield the desired result.

- b. The print queue of the operating system has room for a limited number of entries. It is possible for the user to overrun this queue by rapidly issuing print commands with no time delay between them. If your application will issue several print commands in succession, it may be necessary to implement a software delay between print commands.
- c. The **input** instruction takes precedence over the **print** and **print_num** instructions. The program sequence:

```

NULL      text      ""          null string
MSG_PROMPT text      "Please Enter Speed: "
           print     MSG_PROMPT
           input     NULL,4,0,x,in_done

```

may result in the **input** instruction executing before **MSG_PROMPT** is sent out the port. A programmed wait of approximately 30 milliseconds between the **print** and the **input** statement may be necessary to achieve the desired result.

2. There are two special cases of the **input** command:
 - a. The first parameter of the **input** instruction is the address of a text string which is sent to the serial port as a prompt. If the prompt string exists, then the current value of the variable being input is sent to the port immediately following the prompt string. If the prompt string is a null string, i.e. consists of a single null character, then no prompt is displayed, and the current value of the variable is NOT sent to the port.
 - b. A special form of the **input** command is provided to handle single character input.

In this form, the prompt string is a null string and the length and width parameters are zero. In this situation, the decimal value of the next character received via the serial port will be placed in the input variable.

Serial Instructions

Instruction	Format
get_pq_space	<i>label</i> get_pq_space space
if_char	<i>label</i> if_char port#,address_label
if_no_char	<i>label</i> if_no_char port#,address_label
input	<i>label</i> input textlabel,len,dec,variable,userflag
port_set	<i>label</i> port_set port#,baud,protocol
print	<i>label</i> print textlabel
print_num	<i>label</i> print_num len,dec,var
stop_input	<i>label</i> stop_input

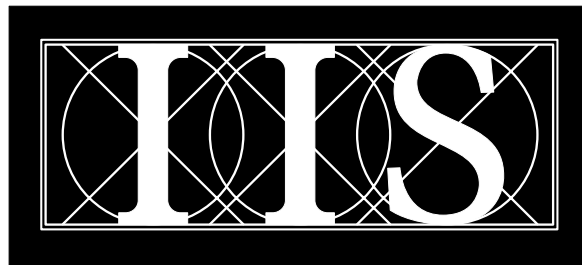
Figure 3.40 - Serial Port Instructions

MACROPROTM

FOR WINDOWSTM

INDUSTRIAL INDEXING SYSTEMS

Analyzer



Industrial Indexing Systems, Inc.

Table of Contents

<i>The Analyzer</i>	4
<i>File</i>	6
New Program	6
Program Info	6
Exit	6
<i>Actions</i>	7
Stop	7
Reset	7
Send	7
Run w/AutoStart	8
Send and Run	8
Run Only	8
Test Mode	8
<i>Status</i>	9
MSC Info	9
MSC Program Information	9
MSC System Status	9
<i>MSC Controller Information</i>	10
MSC Controller Status	10
Controller Selection	10
Status Indicators	11
Flag Description	11
<i>Read/Write</i>	12
Data And Flags	12
Writing Data	13
Writing Flags	13
Update Rate Button	13
Decimal/Hex Values Button	13
Stop/Start Read Button	13
Data/Flag List Button	13
Block Data	14
Decimal/Hex/ASCII Values Button	14

Starting Address Button	14
Data & Flag List - Select Data/Arrays	15
Data & Flag List - Select Flag Name	16
Data & Flag List - Select Flag Number	17
Add Button	17
Remove Button	17
Clear List Button	17
Use Button	18
Add Address Number Button	18
Arrays/Strings Button	18
Quick Find Option Box	18
<i>View</i>	19
Source	19
Equate Table	19
Label Table	20
Constant Table	20
Data Table	20
Close	20
Search	21
Search String	21
Ignore Case	21
Wrap at End	21
Goto Line #	21
<i>Trace</i>	22
Trace Window	22
Helpful Hints	23
Current Trace Type	23
Before Trace Type	23
About Trace Type	23
After Trace Type	24
Stop Trace	24
Close Command Button	24
Print Command Button	24
Reset Command Button	24
Start/Stop Trace Command Button	24

Trace Line	25
Enable Trace After Line	25
Valid Label Selection Window	26
Executable Source Line Labels	26
Read Trace	26
View Last Trace	26
<i>EPROM</i>	27
<i>Window</i>	27
<i>Help</i>	27
<i>Communication Status</i>	28
<i>The Status Bar</i>	28
Message Bar	28
Program in PC	28
Equality Indicator	28
Program in MSC	28

This page is intentionally left blank.

The Analyzer

The MacroPro Analyzer has many different features that allow you to analyze the operation of an Industrial Indexing Systems MSC (Multi-axis Servo Controller).

The following Figures 5.11 - 5.13 consist of the main pieces of the Analyzer screen. The following pages describe the functionality of each menu item or tool bar button seen below.

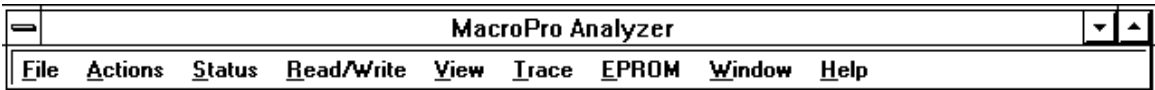


Figure 5.11 - Analyzer Menu Bar



Figure 5.12 - Analyzer Tool Bar



Figure 5.13 - Analyzer Status Bar

File

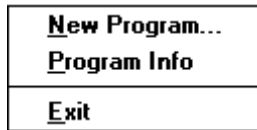


Figure 5.21 - Analyzer File Menu

New Program



New Program will display the standard file open dialog. The default display will show all of the “.prg” source files in the selected directory. The only files that can be loaded into the Analyzer are the “.prg” files. The other associated files will be loaded automatically as required.

Note: There are a number of associated files that must be present to allow full functionality of the Analyzer. These are: “.mcm” (the file that is loaded into the MSC), “.dbg” (analyzer information file), and “.sym” (symbol file, equates, labels, data, & constants).

Program Info



Program Info will display file information about the “Source” and “Executable” file that is resident in your PC. This information includes the file name, size, date and time. It is important to understand that the program loaded into your PC may or may not be the same as the program loaded into the MSC.

To view information about the program loaded into your MSC, use the MSC Info function.

Exit

Exit Analyzer will close all open Analyzer windows.

Actions

<u>S</u> top	F5
<u>R</u> eset	F6
<u>S</u> end	F7
Run w/ <u>A</u> utoStart	F8
<u>S</u> end and Run	F9
Run <u>O</u> nly	
<u>T</u> est Mode	

Figure 5.31 - Analyzer Actions Menu

Stop



Stop will cease the program running in the MSC, if one is running. An f_decel (forced deceleration) command is sent to each controller card.. The F5 function key will also execute this operation.

Reset



Reset will cause a reset of the MSC. The current program must be stopped before a Reset is allowed. The F6 function key will also execute this operation.

NOTE: When a reset is issued, the MSC program memory is cleared, all flags are cleared, and any existing program is erased. All controller cards are reset.

Send



Send will down-load the current program executable file (.mcm) from the PC to the MSC. A Reset must be performed prior to using the Send function. The F7 function key will also execute this operation.

Run w/AutoStart



Run w/AutoStart will turn on the AutoStart bit in the MSC's status word . The MSC will test its AutoStart bit on startup and, if set, the loaded program will start automatically . If the program is stopped, setting AutoStart will start the program in the MSC in addition to setting the AutoStart bit. The F8 function key will also execute this operation. The only way to clear the AutoStart is to issue a Reset.

Send and Run

Send and Run will issue the Stop, Reset, Send, and Run w/AutoStart commands to the MSC. The F9 function key will also execute this operation.

Run Only

Run will start or re-start the program currently resident in the MSC. If a program is not loaded, this function has no effect on the MSC.

Test Mode

Test Mode will place each controller into its test mode. Refer to the particular MSC manual, under the Test Procedure section. This function is typically used during the position loop test.

NOTE: Issuing this command will erase the current program in memory.

Status



Figure 5.41 - Analyzer Status Menu

MSC Info



MSC Info will display a window showing Program information, MSC System Status information and MSC Controllers being used and their part numbers.

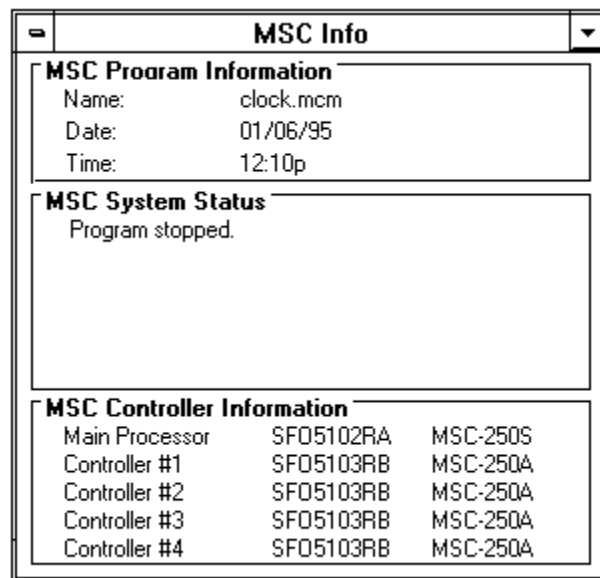


Figure 5.42 - MSC Info Sample Screen

MSC Program Information

This section of the window indicates the name of the program resident on the MSC along with the date and time that this program was compiled or created.

MSC System Status

This section of the window lists all System Status messages, such as “Program Stopped” or “Program Running”.

MSC Controller Information

This section of the window indicates the part number and revision of the firmware (operating system) for the Main Processor and each of the controller cards used. A description of that card follows. The firmware part numbers are shown as SFONNNNRX . “SFO” stands for “System Functional Operation, “NNNN” is a number from 0000 to 9999 and “RX” is a revision equal to “0” or a letter from “A” to “Z”.

MSC Controller Status



MSC Controller Status will display a window showing the status information for a single controller. This includes the flag number of the 16 associated status flags for that Controller as well as a description of each flag. Status flags that are active are indicated as such by a check in the corresponding check boxes.

MSC Controller Status

MSC-250A (Encoder Controller)

Controller	Status	Flag Description
<input checked="" type="radio"/> #1	<input type="checkbox"/> 80	Not Used
<input type="radio"/> #2	<input type="checkbox"/> 81	Lock Pending
<input type="radio"/> #3	<input type="checkbox"/> 82	Following Error
<input type="radio"/> #4	<input checked="" type="checkbox"/> 83	Master/Test Mode
	<input type="checkbox"/> 84	Cam / Piecewise Profile Exceeded
	<input type="checkbox"/> 85	Command Invalid in this State
	<input type="checkbox"/> 86	Not Used
	<input type="checkbox"/> 87	Not Used
	<input type="checkbox"/> 88	Jogging
	<input type="checkbox"/> 89	Indexing
	<input type="checkbox"/> 90	Calculation in Progress
	<input type="checkbox"/> 91	Hardware Interrupt Armed
	<input type="checkbox"/> 92	Forced Decel in Progress
	<input type="checkbox"/> 93	Down
	<input type="checkbox"/> 94	Busy
	<input type="checkbox"/> 95	Master / Slave Lock

Figure 5.43 - MSC Controller Status Sample Screen

Controller Selection

MSC Controller Status will display a window showing the status information for a single controller. This includes the flag number of the 16 associated status flags for that controller as well as a description of each flag. Active Status flags are indicated as such by a check in the corresponding check box.

The left section of the window labeled Controller lists the controllers that can be selected. For example, an MSC-250 is capable of displaying the controller status for 4 different controllers whereas an MSC-850/32 can display the controller status for up to 8 different controllers.

Status Indicators

The center section of the window displays the number of the Status Flag along with a check box which indicates whether or not the listed flag is active.

Flag Description

The right section of the window lists the Description of each Status Flag. The Flag Descriptions will change depending on the type of controller selected.

Read/Write

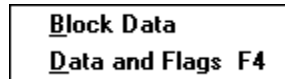


Figure 5.51 - Analyzer Read/Write Menu

Data And Flags



This window is used for reading individual data and/or flags from the MSC and also for writing data and/or flags to the MSC.

NOTE: When reading cam information(arrays), displaying in hexadecimal format is recommended.

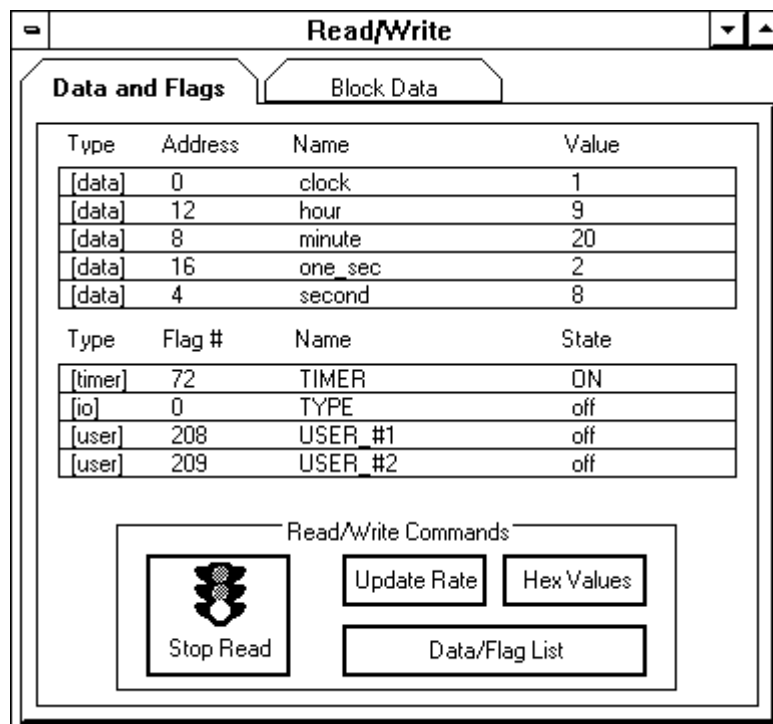


Figure 5.52 - Read/Write Sample Screen (Data and Flags)

Writing Data

To write data, click on the value of the data to write. An entry box and a Cancel button will appear. Type in the value, in decimal only, and hit the 'Enter' key to write the value. The entry box will stay up so that by simply typing in a new value and hitting the 'Enter' key, another write will be performed. Clicking on the Cancel button that appeared will end the data write mode.

Writing Flags

To change the state of a flag, double-click on the flag state. This will toggle the state of the flag. Timers and status flags are read-only.

NOTE: Changing the state of an I/O flag from this window assigns that flag as an output flag and will no longer respond to an external input.

Update Rate Button

This button brings up the update rate scroll bar which allows the user to vary the rate at which the items are read from the MSC. Choosing a faster rate reads from the MSC more often, but slows down the Windows operating system response. The update rate setting for Data and Flags is separate from that of Block Data.

Decimal/Hex Values Button

Toggling this button displays data in either decimal or hexadecimal format.

Stop/Start Read Button

Toggling this button will either stop or start reading items from the MSC. When stopped, the headings and the values are colored red. While reading, the headings are colored blue. The stop/start read setting for Data and Flags is separate from that of Block Data.

NOTE: If "Data and Flags" or "Block Data" are set for reading, they only read when they are the active tab, thus NOT slowing down Windows.

Data/Flag List Button

Clicking the Data/Flag List button will open the which is used to make up or edit the current list of items to read/write.

Block Data



This window is used for reading blocks of data from the MSC and also for writing data to the MSC. The

NOTE: When reading cam information(arrays), displaying in hexadecimal format is recommended.

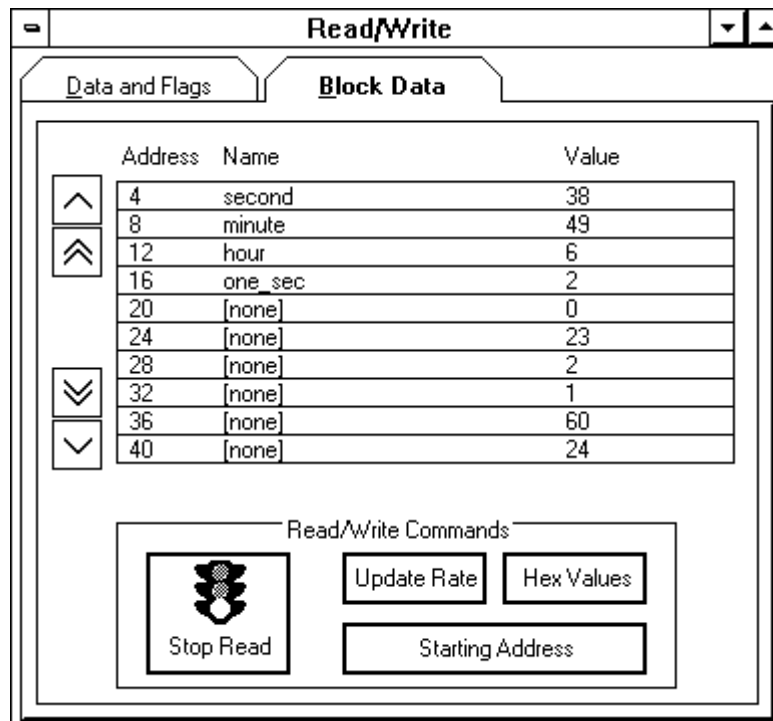


Figure 5.53 - Read/Write Sample Screen (Block Data)

Decimal/Hex/ASCII Values Button

Toggling this button displays data in either decimal, hexadecimal, or ASCII format.

Starting Address Button

Clicking this button to change the starting address of the block of 10 data items. At the prompt enter a data label name or an address number. For array elements, enter the name followed by the element number enclosed in square brackets. For example, to enter a starting address of the 3rd element in the array, Cam; simply enter: Cam[2] (since arrays start with zero).

Data & Flag List - Select Data/Arrays

The Data & Flag List window is used to make up a list of data and flags to read/write. A maximum of 10 items can be used and may be in any combination of data and/or flags. The Select Data list box is a list of variables defined in the current program (Source) and can be added to the read/write list. Just double-click on the data item to add or select the item and then click the Add button.

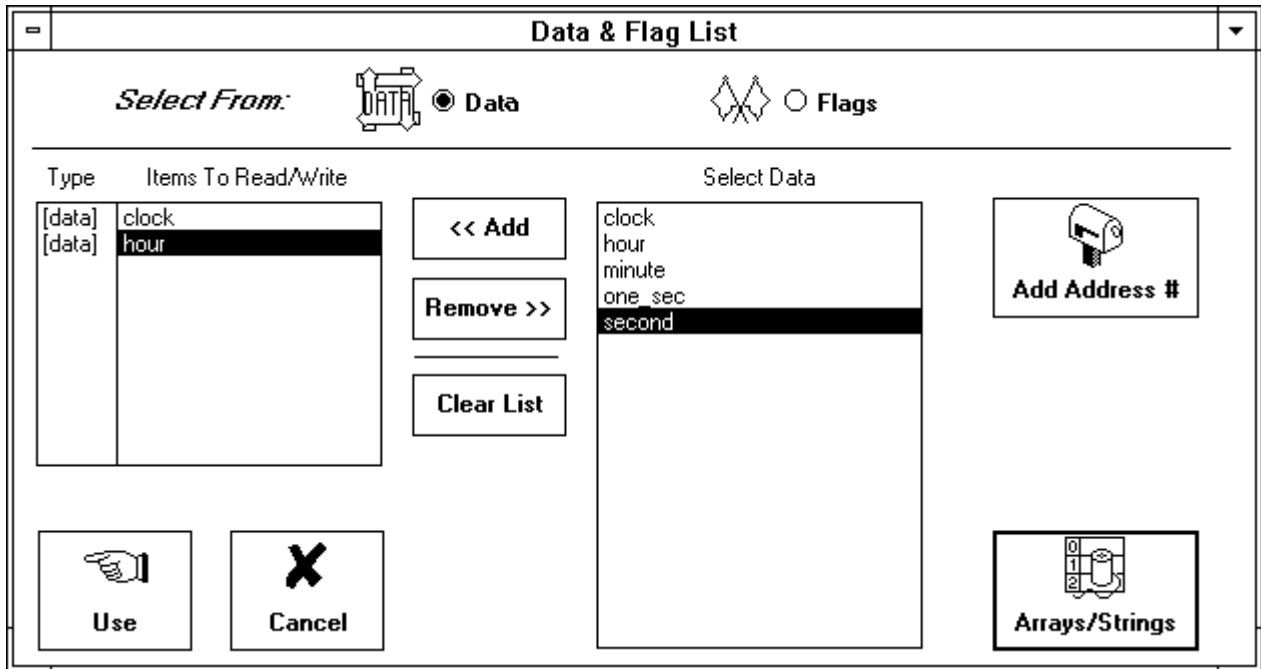



Figure 5.54 - Data & Flag List (Select Data)

Data & Flag List - Select Flag Name


The Data & Flag List window is used to make up a list of data and flags to read/write. A maximum of 10 items can be used and may be in any combination of data and/or flags. The Flag Name list box is a list of generic-named flag numbers which can be added to the read/write list. The Possible Flag Number is a generic-named flag number that is equal to the flag name selected in the Flag Number list box. The Possible Flag Number can NOT be added to the read/write list, it is displayed here to help the user determine if the selected flag name is the proper flag desired.

Data & Flag List

Select From:




☐ Data




☒ Flags


Type	Items To Read/Write		#	Flag Name	Possible Flag Number
[data]	clock	<div><< Add</div> <div>Remove >></div> <div>Clear List</div>	72	TIMER	<div>IO_#1</div>
[data]	hour		0	TYPE	
[data]	second				



Use



Cancel



Flag #'s

Figure 5.55- Data & Flag List (Select Flag Name)

Data & Flag List - Select Flag Number

The Data & Flag List window is used to make up a list of data and flags to read/write. A maximum of 10 items can be used and may be in any combination of data and/or flags. The Flag Number list box is a list of generic-named flag numbers which can be added to the read/write list. The Flag Name list box is a list of equates from the program that are equal to the flag number selected in the Flag Number list box. The Flag Name can also be added to the read/write list.

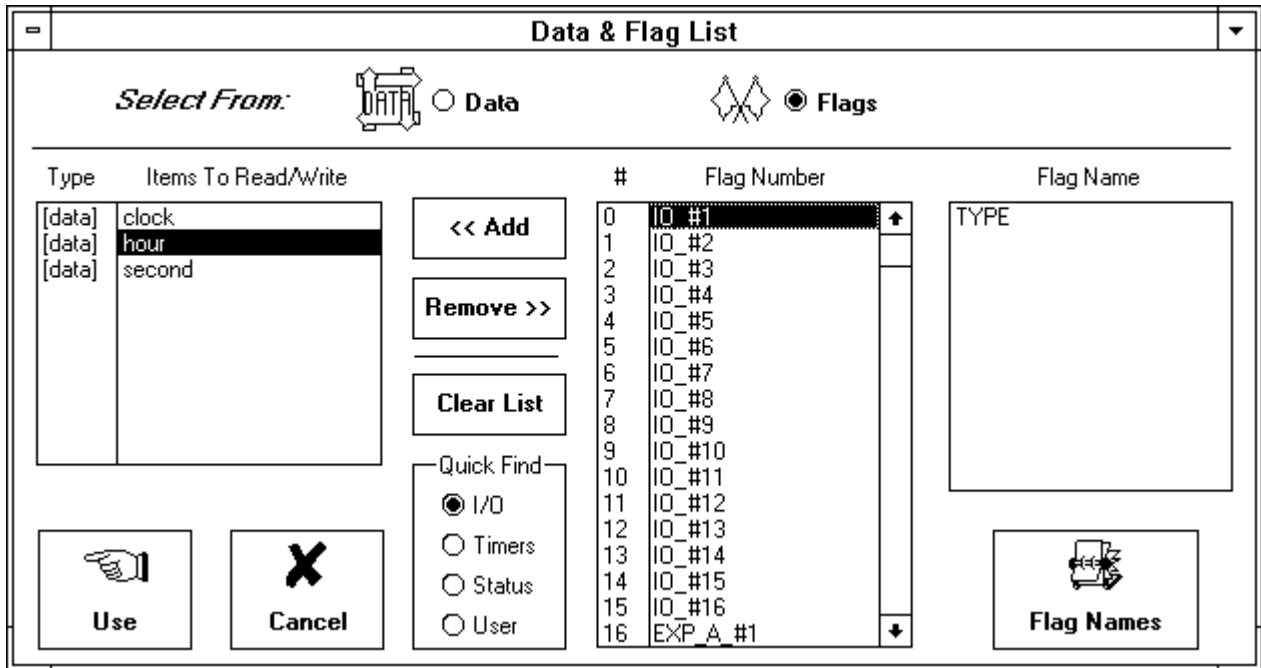


Figure 5.56 - Data & Flag List (Select Flag Number)

Add Button

Items may be added by highlighting the item in “Flag Number” or “Flag Name” list and using the Add button. Double-clicking the item will also select the item and add it to the list.

Remove Button

Items may be subtracted by highlighting the item in “Items to Read/Write” list and using the Remove button. Double-clicking the item will also select the item and remove it from the list.

Clear List Button

All of the items in “Items to Read/Write” list will be removed.

Use Button

Clicking this button will minimize the Data & Flags List window, show the Read /Write window, and begin reading the items from the read/write list.

Add Address Number Button

Click this button to add an address number to the read/write list. Address must be greater than or equal to zero and also divisible by 4 since MSC memory is organized in 4-byte words.

Arrays/Strings Button

To add an array element to the read/write list, click the Array/Strings button and the Data Select list box will display all arrays and text strings. Then, when adding an array/string item to the read/write list, you will be prompted to enter the element number.

NOTE: Every element contains 4-bytes.

Quick Find Option Box

The Quick Find Option box will allow you to quickly jump through the Flag Numbers list to reach the start of the particular section of interest. Flags start at #0 and go up to #255. There are four major flag sections, I/O flags #0 to #71, Timer flags #72 to #79, Status flags #80 to #207, and User Flags #208 to #255.

View

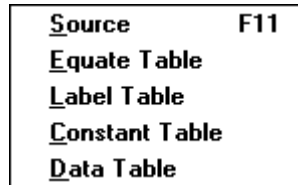


Figure 5.61 - Analyzer View Menu

Source



Source will display the Source for the program currently resident in the PC. Various controls allow you to view any or all program Source lines as needed. The F11 function key will also execute this operation.

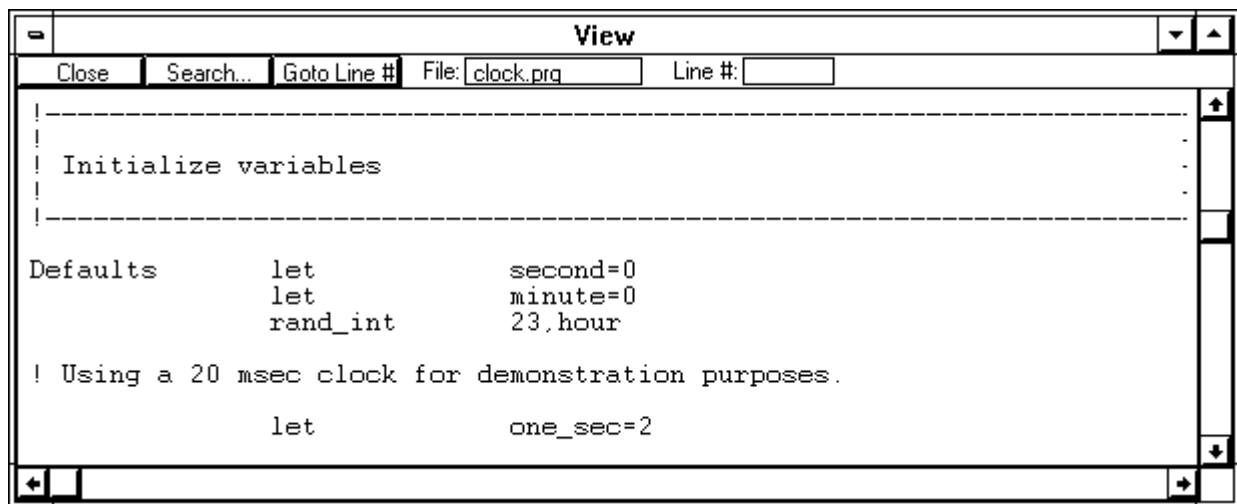


Figure 5.62 - View Sample Screen

Equate Table

Equate Table will move the file display pointer to the section in the symbol file (.sym) for the program currently resident in the PC, where all labels that were equated (using the *equ* instruction in the program) are listed. Various controls allow you to view any or all of the symbol file (.sym) as needed.

Label Table

Label Table will move the file display pointer to the section in the symbol file (.sym) for the program currently resident in the PC, where all program Labels are listed. Various controls allow you to view any or all of the symbol file (.sym) as needed.

Constant Table

Constant Table will move the file display pointer to the section in the symbol file (.sym) for the program currently resident in the PC, where all Constants used in the program are listed. Various controls allow you to view any or all of the symbol file (.sym) as needed.

Data Table

Data Table will move the file display pointer to the section in the symbol file (.sym) for the program currently resident in the PC, where all program Constants are listed. Various controls allow you to view any or all of the symbol file (.sym) as needed.

Close

Close will exit the View window.

Search

Search allows you to find a designated text string within the file being viewed. A list of strings previously searched is maintained for easier viewing. Like most Search or find functions, the F3 key will search for the next occurrence of the entered string.

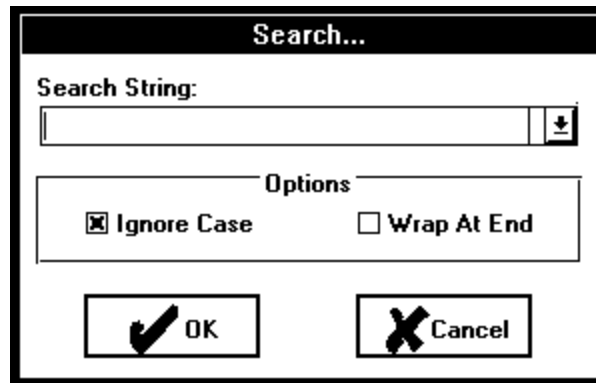


Figure 5.63 - Search Window

Search String

Enter the String to be searched, keeping in mind the Case option selected. A list of strings previously searched is maintained for easier viewing. Use the scroll bar to search on other strings as needed.

Ignore Case

When Ignore Case is checked, strings will be compared and the match will be made regardless of the case entered in the Search String *or* the file being searched.

Wrap at End

Wrap at End will cause the search to continue at the beginning of a file, when an end-of-file is encountered.

Goto Line

Goto Line # lets you position to a specific line within the file being viewed. The range of allowable lines is listed.

Trace



The Trace feature allows you to follow the program execution without effecting the operation of the program.

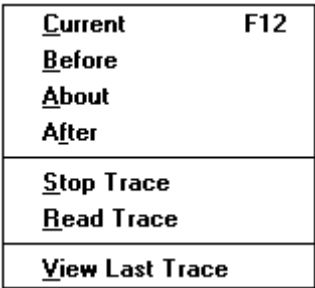


Figure 5.71 - Trace Menu Bar

Trace Window

The Trace feature is closely linked with the source file so that when the Trace window is opened, the Source view window is also opened. If the Source view window is already open, it may be re-sized so that the Trace and Source windows are equal in size.

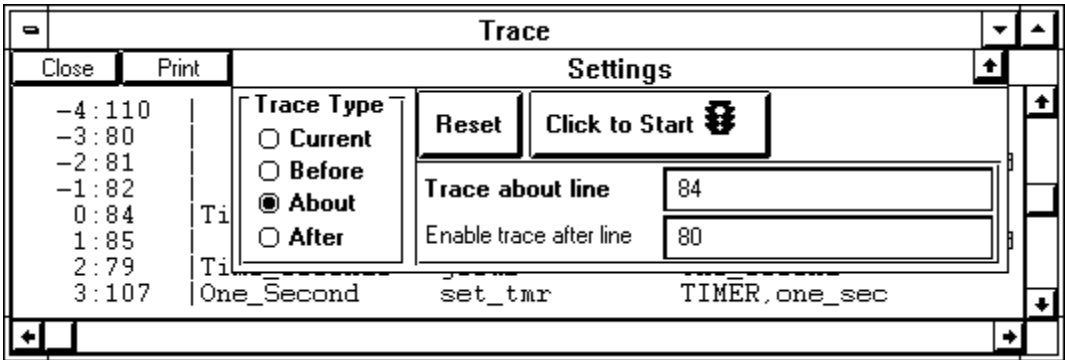


Figure 5.72 - Trace Window

The Trace Feature has the ability to capture up to 112 lines of program execution. This is not to say that each time the Trace is run that you should see 112 lines. The trace type, trace events, and program will effect the amount of lines that are displayed after a successful trace.

After a successful trace, the “Settings” window will roll-up to show the full trace listing. The trace listing shows the Trc. # (trace number) in the first column. This trace number indicates the order in which a program line was executed. A negative number indicates lines executed before the trace line, the trace line numbered zero is the line that was specified as the line to trace on. Positive numbered lines are lines that are executed after the trace line was executed. The next column in the trace listing is the

source program line number (Line #). The remaining columns are the actual source text for the associated line numbers.

There are two items in the menu bar Trace pull down box that are not available in the Trace window. These are Read Trace, and View Last Trace.

Helpful Hints

Clicking on a line in the Source window will automatically set the Trace Line or Enable Trace After Line. Repeated clicking on lines in the Source window will reset these trace events.

Highlighting a line in the Trace window will display the associated line in the Source window.

Entering any text string in the Trace Line or Enable Trace After Line entry boxes will bring up a Valid Label Selection window. This window will list all of the labels in the program that can be used as a trace parameter .

Current Trace Type

The Current trace type will initiate the trace process regardless of the settings for Trace Line, and Enable Trace After Line. This trace type is useful when trying to determine if the program is running, and what part of the program is being executed.

Before Trace Type

The Before trace type will allow you to see which lines are being executed before a particular program line. You must enter a Trace Line parameter to enable a trace of this type. The Enable Trace After Line is optional but will have an effect on the number of trace lines that might be displayed. The maximum number of lines that could be displayed before the “trace line” would be 112, with the trace line being Trc. # 0.

About Trace Type

The About trace type will allow you to see which lines are being executed before and after a particular program line. You must enter a Trace Line parameter to enable a trace of this type. The Enable Trace After Line is optional but will have an effect on the number of trace lines that might be displayed before the actual trace line. The maximum number of lines that could be displayed before the “trace line” would be 57 (negative Trc. #'s), with the trace line being Trc. # 0. The maximum number of trace lines that would be displayed after the trace line would be 55 (positive Trc. #'s).

After Trace Type

The After trace type will allow you to see which lines are being executed after a particular program line. You must enter a Trace Line parameter to enable a trace of this type. The Enable Trace After Line is optional. Your program will have an effect on the number of trace lines that might be displayed. The maximum number of lines that could be displayed after the “trace line” would be 112, with the trace line being Trc. # 0.

Stop Trace

This menu item will stop the currently active trace.

Close Command Button

The close command button will stop any active trace events, then close the trace window.

Print Command Button

The print command button will allow you to print the contents of the trace listing to your MS Windows print driver. This print uses the default settings that were last set with the Microsoft Windows Print Manager. It is recommended that you select a fixed space font (such as Courier), to give you the proper column alignment on the print out.

Reset Command Button

The reset command button will stop any active trace and reset the Trace Line and Enable Trace After Line parameters.

Start/Stop Trace Command Button

The Start/Stop trace command button will initiate the trace process based on the settings listed in the “Settings” window, or stop an active trace. When a trace is active the “Trace Active” message panel will flash next to the Start/Stop trace command button, as shown the following figure.

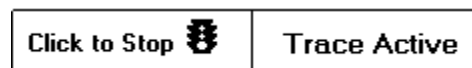


Figure 5.73 - “Trace Active” Panel

Trace Line

Purpose:

The trace line entry is the specifier which is the focal point of the trace process. This entry identifies the line that the MSC is looking for. This parameter is required when using the Before, About and After trace types.

Entry method:

The trace line entry box accepts either a source line number or an Executable Source Line Labels. Checks are made on the validity of those entries. To enter a line number you can enter the number manually in the box or you can click on a line in the Source window. When you click on a line in the Source window the associated line number will be automatically entered into the entry box. To enter a label you must enter the alpha string into the box manually. If an invalid label was entered a Valid Label Selection Window will be displayed to allow you to change your label selection.

Enable Trace After Line

Purpose:

The Enable trace after line entry is the specifier which tells the MSC to start looking for Trace Line. Only AFTER the Enable trace after line has been executed, will the MSC start to look for the Trace line.

Entry method:

The trace line entry box accepts either a source line number or an Executable Source Line Labels. Checks are made on the validity of those entries. To enter a line number you can enter the number manually in the box or you can click on a line in the Source window. When you click on a line in the Source window the associated line number will be automatically entered into the entry box. To enter a label you must enter the alpha string into the box manually. If an invalid label was entered a Valid Label Selection Window will be displayed to allow you to change your label selection.

Valid Label Selection Window

The following figure shows the Valid Label Selection window that would be displayed if an invalid label entry was entered into the Trace Line and Enable Trace After Line entry boxes. Valid labels are Executable Source Line Labels.

Entry method:

You can type in a label that is shown on the list, and then click on Accept. You can also double click on the label of choice.

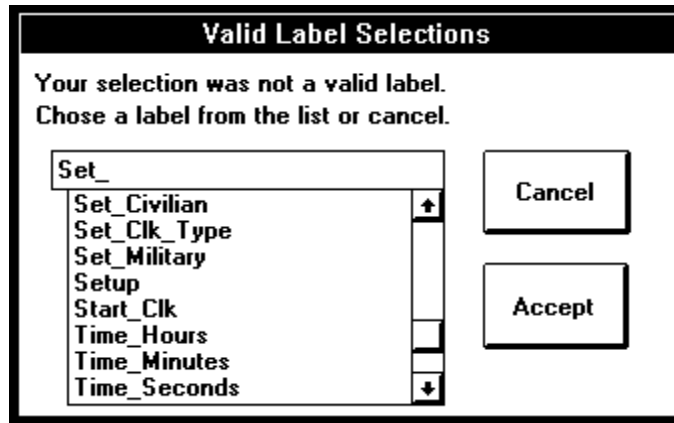


Figure 5.74 - Valid Label Selection Window

Executable Source Line Labels

This is a label that is tied to a line in the program that will be executed when the program runs. Labels for equates or data are not considered executable labels.

Read Trace

This menu item is not available on the Trace window.

In some cases a trace might not complete as expected. In the event this happens, but you still want to see what is currently in the MSC's trace memory, select Read Trace. This feature will stop the active trace then access the current state of the trace memory in the MSC. For this feature the trace "Settings" are ignored.

View Last Trace

This menu item is not available on the Trace window.

This feature will open the trace window if not already open and display the last trace results.

EPROM



You can save a copy of the program area and data area of the MSC onto a 256K bit PROM using EPROM.

The program in the MSC cannot be running when this is selected. If running, you will be prompted to first STOP the program.

The length of time required to save the program and data area is dependent on the size of these areas. Typically this operation takes less than 20 seconds.

Once completed, a message will be displayed indicating that the save was or was not successful. If not successful, the appropriate error message will be shown.

Window

Window lists the windows currently open in the Analyzer. Selecting any of these windows will re-open the window.

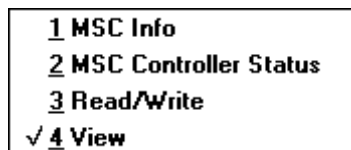


Figure 5.91 - Analyzer Window Menu

Help



Help gives you access to the Help facility for the Analyzer. The F2 function key will also execute this operation.

Communication Status



This is an indicator of the status of the connection between the PC and the MSC. If the connection line of the status indicator is "flowing", then there is a valid connection between the PC and the MSC.

The Status Bar

The following are the specific parts of the Analyzer status bar located at the bottom of the screen. Each of the items below are listed as seen from left to right on the status bar as seen in Figure 5.13.

Message Bar

This line is where the Analyzer notifies the user about the status of a current operation.

Program in PC

This is the name of the program that is currently in the PC.

Equality Indicator

This indicator will either be an equals sign or an unequals sign. An equals sign shows that the program in the PC is exactly the same as the program in the MSC, whereas an unequals sign shows that there is a difference between the two programs in the PC and the MSC (date and time stamps are included in the comparison).

Program in MSC

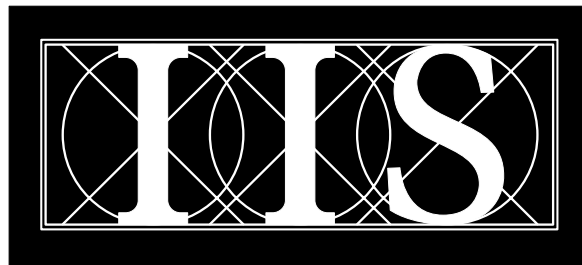
This is the name of the program that is currently loaded in the MSC.

MACROPROTM

FOR WINDOWSTM

INDUSTRIAL INDEXING SYSTEMS

Instruction Reference
& Examples



Industrial Indexing Systems, Inc.

Table of Contents (Instructions)

Instruction	Page
<i>analog_in</i>	8
<i>analog_out</i> (ACM-850)	9
<i>analog_out</i> (ACR-850/ACE-850/ACY-850)	10
<i>analog_out</i> (MSC-250 controllers 1 & 2)	11
<i>analog_out</i> (MSC-250 controller 4)	12
<i>analog_rt</i>	13
<i>analog_zo</i>	14
<i>begin_cam</i>	15
<i>begin_data</i>	16
<i>blk_io_in</i>	17
<i>blk_io_out</i>	18
<i>calc_cam_sum</i>	19
<i>calc_unit_cam</i>	20
<i>cam</i>	21
<i>cam_data</i>	22
<i>case</i>	24
<i>close</i>	25
<i>clr_all_swi</i>	26
<i>clr_bit</i>	27
<i>clr_flag</i>	28
<i>clr_hi_scan</i>	29
<i>clr_local</i>	30
<i>clr_swi</i>	31
<i>create</i>	32
<i>data</i>	33
<i>declare</i>	34
<i>default</i>	35
<i>digi_comp</i>	36
<i>dim</i>	37
<i>disable_hwi</i>	38

<i>disable_swi</i>	39
<i>drive_off</i>	40
<i>drive_on</i>	41
<i>enable_hwi</i>	42
<i>enable_swi</i>	44
<i>end_cam</i>	45
<i>end_data</i>	46
<i>end_select</i>	47
<i>equ</i>	48
<i>exec_profile</i>	49
<i>exit_select</i>	50
<i>f_decel</i>	51
<i>find_mrk_ccw</i>	52
<i>find_mrk_cw</i>	53
<i>find_tm_ccw</i>	54
<i>find_tm_cw</i>	55
<i>get_act_spd</i>	57
<i>get_angle</i>	58
<i>get_angle (MSC-850/HPL-850)</i>	59
<i>get_cam_cnt</i>	60
<i>get_cam_end</i>	61
<i>get_cam_ptr</i>	62
<i>get_cam_strt</i>	63
<i>get_cam_sum</i>	64
<i>get_com</i>	65
<i>get_fol_err</i>	66
<i>get_for_ang</i>	67
<i>get_for_spd</i>	68
<i>get_map</i>	69
<i>get_map_stat</i>	70
<i>get_mcf</i>	71
<i>get_pls_mask</i>	72
<i>get_pls_out</i>	73

<i>get_pq_space</i>	74
<i>get_pos</i>	75
<i>get_pstat</i>	76
<i>get_space</i>	77
<i>get_status</i>	78
<i>get_t_mark</i>	79
<i>get_time</i>	80
<i>get_trap_pos</i>	81
<i>get_volume</i>	82
<i>gosub</i>	83
<i>goto</i>	84
<i>if</i>	85
<i>if_bit_clr</i>	86
<i>if_bit_set</i>	87
<i>if_char</i>	88
<i>if_flag_off</i>	89
<i>if_flag_on</i>	90
<i>if_io_off</i>	91
<i>if_io_on</i>	92
<i>if_no_char</i>	93
<i>if_stat_on</i>	94
<i>if_stat_off</i>	95
<i>if_tmr_off</i>	96
<i>if_tmr_on</i>	97
<i>incr_com</i>	98
<i>index</i>	99
<i>initialize</i>	100
<i>input</i>	101
<i>integer</i>	103
<i>jog_ccw</i>	104
<i>jog_cw</i>	105
<i>l_track_spd</i>	106
<i>let</i>	107

<i>let_byte</i> _____	109
<i>load</i> _____	111
<i>lock</i> _____	112
<i>master</i> _____	117
<i>msc_type</i> _____	118
<i>no_op</i> _____	119
<i>offset_master</i> _____	120
<i>open</i> _____	121
<i>over_draw</i> _____	122
<i>port_set</i> _____	124
<i>position</i> _____	126
<i>prep_profile</i> _____	127
<i>preset</i> _____	128
<i>preset (MSC-850/HPL-850)</i> _____	129
<i>print</i> _____	130
<i>print_num</i> _____	131
<i>rand_int</i> _____	132
<i>ratio</i> _____	133
<i>read</i> _____	134
<i>read_offset</i> _____	135
<i>restart_at</i> _____	136
<i>return_sub</i> _____	137
<i>save</i> _____	138
<i>select</i> _____	139
<i>set_ac_dc</i> _____	140
<i>set_acy_cnt</i> _____	141
<i>set_bit</i> _____	142
<i>set_cam_ptr</i> _____	143
<i>set_com_limit</i> _____	144
<i>set_flag</i> _____	145
<i>set_gl_ccw</i> _____	146
<i>set_gl_cw</i> _____	147
<i>set_gl_ccw (HPL-850)</i> _____	148

<i>set_gl_cw (HPL-850)</i>	149
<i>set_hi_scan</i>	150
<i>set_home</i>	151
<i>set_local</i>	153
<i>set_local (HPL-850)</i>	154
<i>set_map</i>	155
<i>set_mcf (MCF-850)</i>	156
<i>set_mcf (ACR-850/ACE-850/ACY-850)</i>	157
<i>set_mcf (MSC-250 controllers 1 & 2)</i>	159
<i>set_mcf (MSC-250 controller 3)</i>	161
<i>set_mcf (HPL-850)</i>	162
<i>set_nar_ang</i>	163
<i>set_offset</i>	164
<i>set_ovd_mode</i>	165
<i>set_pls_ang (MCF-850)</i>	166
<i>set_pls_ang (HPL-850)</i>	167
<i>set_pls_ang (MSC-250)</i>	168
<i>set_pls_cnt (MCF-850)</i>	169
<i>set_pls_cnt (HPL-850)</i>	170
<i>set_pls_mask</i>	171
<i>set_pls_time</i>	172
<i>set_speed</i>	173
<i>set_swi_mask</i>	174
<i>set_tmr</i>	175
<i>set_trig_cam</i>	176
<i>set_trig_pw</i>	177
<i>set_vgain</i>	178
<i>set_wide_ang</i>	179
<i>stop_input</i>	180
<i>swi_if_off</i>	181
<i>swi_if_on</i>	182
<i>switch_cam</i>	183
<i>sys_fault</i>	186

<i>sys_return</i>	187
<i>test_mode</i>	188
<i>text</i>	189
<i>track_spd</i>	190
<i>trap_pos</i>	191
<i>turn_off</i>	192
<i>turn_on</i>	193
<i>unlock</i>	194
<i>vel_ccw</i>	195
<i>vel_cw</i>	196
<i>write</i>	197

Table of Contents (Examples)

Example	Page
<i>Analog Functions</i>	198
<i>CAMS with switch_cam</i>	201
<i>Clock Simulation</i>	209
<i>Data Storage Techniques onto EPROM with Operator Interface</i>	212
<i>Home Sequence</i>	231
<i>Index with over_draw</i>	237
<i>Programmable Limit Switch Functions</i>	242
<i>Piecewise Profile</i>	248
<i>Ratio Lock</i>	253
<i>CAMS with calc_unit_cam</i>	258
<i>I/O Functions with IOE</i>	264
<i>Offset Methods</i>	269
<i>Clear Software Interrupts</i>	276
<i>Enable Sequence for Absolute Encoder</i>	279

analog_in

SYNTAX:

label *analog_in* controller#,channel#,variable

PARAMETERS:

controller#	controller id#	
	Range: MSC-250	Always 4
	MSC-850/32	1 to 8
	MSC-850	1 to 8
	MSC-800	1 to 8
channel#	Channel to be used.	
	Range: MSC-250	Always 1
	MSC-850/32	1 to 8
	MSC-850	1 to 8
	MSC-800	1 to 8
variable	Returned reading.	

DESCRIPTION:

Perform a read from the specified channel of the analog controller. The data is placed in 'variable' after it is modified based on the channel's current slew rate limit and offset parameters. The modified data will have a range of -2048 to +2047 which is equivalent to -10 VDC to +10 VDC.

The value read may be delayed up to 11 milliseconds due to access time through the update loop.

RETURNS:

Returns the current reading of the specified analog input channel plus any currently set offset value and limited by current slew rate setting.

USAGE:

MSC-250: *analog_in*
 MSC-850/32: *analog_in*
 MSC-850: *analog_in*
 MSC-800: *analog_in*

analog_out (ACM-850)**SYNTAX:**

label *analog_out* controller#,channel#,value

PARAMETERS:

controller#	controller id#
	Range: MSC-850/32 1 to 8
	MSC-850 1 to 8
	MSC-800 1 to 8
channel#	Channel to be used.
	Range: MSC-850/32 9 to 12
	MSC-850 9 to 12
	MSC-800 9 to 12
value	Value to output.
	Range: -2048 to +2047

DESCRIPTION:

Modify the output value by the specified channel's current slew rate limit and offset parameters and then write to the specified channel. The specified channel update may be delayed for as long as 11 milliseconds due to the access time of the control loop.

Executing the **f_decel** macro instruction will slew all output channels to 0 volts.

The modified output data will have a range from -2048 to +2047 which is equivalent to -10 VDC to +10 VDC.

Outputs are initialized to 0.0 volts on power-up. Outputs will slew to 0.0 volts when program is stopped by the MSC Toolkit.

RETURNS:

None.

USAGE:

MSC-250:	analog_out
MSC-850/32:	analog_out
MSC-850:	analog_out
MSC-800:	analog_out

analog_out (ACR-850/ACE-850/ACY-850)

SYNTAX:

label analog_out controller#,channel#,value

PARAMETERS:

controller#	controller id# Range: MSC-850/32 1 to 8 MSC-850 1 to 8
channel#	Channel to be used. Range: MSC-850/32 Always 1 MSC-850 Always 1
value	Value to output. Range: -2048 to +2047

DESCRIPTION:

The **analog_out** instruction can be used with the ACR-850, ACE-850 and ACY-850 controller cards in order to implement an open loop mode of operation known as "analog mode".

In this mode of operation, the drive unit will be enabled by an external input source. A **drive_off** instruction followed by a **set_mcf** instruction to the ACE-850, ACR-850 or ACY-850 will put that axis controller into "analog mode". Once in "analog mode", the controller will be in an open loop mode where the analog output to the drive is not driven by the "position loop", but rather is controlled in the Macroprogram using the **analog_out** instruction.

When used with the ACR-850, ACE-850 or ACY-850 cards, the **analog_out** instruction will now function in the same manner as when it is used with the ACM-850 card. A voltage in the range of -10V to +10V, based on an **analog_out** value ranging from -2048 to +2047, will be generated by the ACE-850, ACR-850 or ACY-850 controller cards.

A subsequent **drive_on** instruction will put the controller back into the normal "position loop mode" of operation.

RETURNS:

None.

USAGE:

MSC-250:	analog_out
MSC-850/32:	analog_out
MSC-850:	analog_out
MSC-800:	analog_out

analog_out (MSC-250 controllers 1 & 2)

SYNTAX:

label *analog_out* controller#,channel#,value

PARAMETERS:

controller#	controller id#	
	Range: MSC-250	1 or 2
channel#	Channel to be used.	
	Range: MSC-250	Always 1
value	Value to output.	
	Range: -2048 to +2047	

DESCRIPTION:

The **analog_out** instruction can be used with axis controller #1 and #2, in order to implement an open loop mode of motion control known as "analog mode".

In this mode of operation, the drive will be enabled by an external input source. A **drive_off** instruction followed by a **set_mcf** instruction (with a value of 1) to the axis controller, will put that axis controller into "analog mode". Once in "analog mode", the controller will be in an open loop mode. The analog output voltage from the controller can be driven using the **analog_out** instruction. The **analog_out** instruction will now function in the same manner as it is used with axis controller #4 of the MSC-250, which provides a single analog output channel. A voltage in the range of -10V to +10V, based on an *analog_out* value ranging from -2048 to +2047, will be generated by the axis controller.

A subsequent **set_mcf** instruction (with a value of 0) will put the axis controller back into the normal "position loop mode" of operation.

RETURNS:

None.

USAGE:

MSC-250:	<i>analog_out</i>
MSC-850/32:	<i>analog_out</i>
MSC-850:	<i>analog_out</i>
MSC-800:	<i>analog_out</i>

analog_out (MSC-250 controller 4)

SYNTAX:

label *analog_out* controller#,channel#,value

PARAMETERS:

controller#	controller id#	
	Range: MSC-250	Always 4
channel#	Channel to be used.	
	Range: MSC-250	Always 1
value	Value to output.	
	Range: -2048 to +2047	

DESCRIPTION:

Modify the output value by the specified channel's current slew rate limit and offset parameters and then write to the specified channel. The specified channel update may be delayed for as long as 11 milliseconds due to the access time of the control loop.

Executing the **f_decel** macro instruction will slew all output channels to 0 volts.

The modified output data will have a range from -2048 to +2047 which is equivalent to -10 VDC to +10 VDC.

Outputs are initialized to 0.0 volts on power-up. Outputs will slew to 0.0 volts when program is stopped by the MSC Toolkit.

RETURNS:

None.

USAGE:

MSC-250:	analog_out
MSC-850/32:	analog_out
MSC-850:	analog_out
MSC-800:	analog_out

analog_rt**SYNTAX:**

label *analog_rt* *controller#,channel#,value*

PARAMETERS:

<i>controller#</i>	controller id#	
	Range: MSC-250	Always 4
	MSC-850/32	1 to 8
	MSC-850	1 to 8
	MSC-800	1 to 8
<i>channel#</i>	Channel to be used.	
	Range: MSC-250	1 (for input channel) & 9 (for output channel)
	MSC-850/32	1 to 12
	MSC-850	1 to 12
	MSC-800	1 to 12
<i>value</i>	The slew rate limit.	
	Range: 0 to 2047	

DESCRIPTION:

Sets the specified channel slew rate limit in bytes per 10 milliseconds. The slew rate value limits the rate at which a particular channel (input or output) can change. A slew rate of 1 is equivalent to a rate of change of 4.88 mV per 10 milliseconds.

RETURNS:

None.

USAGE:

MSC-250: *analog_rt*
 MSC-850/32: *analog_rt*
 MSC-850: *analog_rt*
 MSC-800: *analog_rt*

analog_in

SYNTAX:

label analog_in controller#,channel#,variable

PARAMETERS:

controller#	controller id#
	Range: MSC-250 Always 4
	MSC-850/32 1 to 8
	MSC-850 1 to 8
	MSC-800 1 to 8
channel#	Channel to be used.
	Range: MSC-250 Always 1
	MSC-850/32 1 to 8
	MSC-850 1 to 8
	MSC-800 1 to 8
variable	Returned reading.

DESCRIPTION:

Perform a read from the specified channel of the analog controller. The data is placed in 'variable' after it is modified based on the channel's current slew rate limit and offset parameters. The modified data will have a range of -2048 to +2047 which is equivalent to -10 VDC to +10 VDC.

The value read may be delayed up to 11 milliseconds due to access time through the update loop.

RETURNS:

Returns the current reading of the specified analog input channel plus any currently set offset value and limited by current slew rate setting.

USAGE:

MSC-250: analog_in
MSC-850/32: analog_in
MSC-850: analog_in
MSC-800: analog_in

begin_cam

SYNTAX:

label begin_cam

PARAMETERS:

None.

DESCRIPTION:

Signals the start of a cam data area which will contain 8 bit (1 byte) values which represent incremental cam data. Each **begin_cam** instruction must have a corresponding **end_cam** instruction.

This instruction requires a 'label'.

EXAMPLE

```
cam_array    begin_cam
              cam    1,2,3,4,5
              cam    6,6,6,6,6
              cam    5,4,3,2,1
              end_cam
```

RETURNS:

None.

USAGE:

```
MSC-250:    begin_cam
MSC-850/32: begin_cam
MSC-850:    begin_cam
MSC-800:    begin_cam
```

begin_data

SYNTAX:

label begin_data

PARAMETERS:

None.

DESCRIPTION:

Signals the beginning of a data area which will contain 32-bit (4 byte) values.

This instruction requires a 'label'.

EXAMPLE

profile	begin_data	
	data	100,200,15*4096
	data	0,200,40*4096
	end_data	

RETURNS:

None.

USAGE:

MSC-250:	begin_data
MSC-850/32:	begin_data
MSC-850:	begin_data
MSC-800:	begin_data

blk_io_in

SYNTAX:

label blk_io_in input_flag#,variable

PARAMETERS:

input_flag#	Starting flag number to read. Must be a multiple of 8 (0,8,16 etc.).
variable	Value of I/O flags to be read.

DESCRIPTION:

The first eight bits in 'variable' are used to store the state of the I/O modules, starting with 'input_flag#'.

EXAMPLE

If 'variable' = 3, the first two inputs are ON.

If 'variable' = 255, all eight input modules are ON.

RETURN:

A four byte variable with the LS byte representing the ON and OFF states of the eight I/O read.
(Bit ON, I/O is ON)

USAGE:

MSC-250:	blk_io_in
MSC-850/32:	blk_io_in
MSC-850:	blk_io_in
MSC-800:	N/A

blk_io_out

SYNTAX:

label blk_io_out output_flag#,variable

PARAMETERS:

output_flag#	Starting flag number to modify. Must be a multiple of 8 (0,8,16 etc.).
variable	Value of I/O flags to set.

DESCRIPTION:

The first eight bits in 'variable' are used to set or clear eight outputs, starting with 'output_flag#'.

EXAMPLE

If 'variable' = 3, the first two output modules are ON.

If 'variable' = 255, all eight output modules are ON.

RETURNS:

None.

USAGE:

MSC-250:	blk_io_out
MSC-850/32:	blk_io_out
MSC-850:	blk_io_out
MSC-800:	blk_io_out

calc_cam_sum

SYNTAX:

label calc_cam_sum controller#,starting element,ending element

PARAMETERS:

controller#	controller id # Range: MSC-250 1 to 2 MSC-850/32 1 to 8 MSC-850 1 to 8
starting element	Element # relative to axis memory zero.
ending element	Element # relative to axis memory zero.

DESCRIPTION:

Sums the values of cam elements in axis controller memory starting with 'starting element' and including 'ending element'. Starting and ending element numbers are relative to axis controller memory location zero.

The axis status flag CALCULATION IN PROGRESS will be ON until calculations are complete

The calculated sum can be retrieved using the **get_cam_sum** instruction.

RETURNS:

None.

USAGE:

MSC-250:	calc_cam_sum
MSC-850/32:	calc_cam_sum
MSC-850:	calc_cam_sum
MSC-800:	N/A

calc_unit_cam

SYNTAX:

label calc_unit_cam controller#,distance, # of elements, starting element

PARAMETERS:

controller#	controller id# Range: MSC-250 1 to 2 MSC-850/32 1 to 8 MSC-850 1 to 8
distance	Total distance of the cam in bits.
# of elements	Number of elements which make up the cam.
starting element	The first element of the cam. This element is relative to axis memory location zero.

DESCRIPTION:

Allows the user to define the shape of a cam and to have the axis controller calculate the cam data array. A table of co-efficients is transmitted to the axis controller as well as the number of elements and starting element in the axis controller's 28K data block. The resolution of the table of co-efficients is 128 data points. The 128 data points used to define the shape of the cam must reside in the last 1K memory area (elements 6750 - 6879) of the axis controller's 28K memory array prior to execution of this instruction.

RETURNS:

None.

USAGE:

MSC-250: calc_unit_cam
MSC-850/32: calc_unit_cam
MSC-850: calc_unit_cam

cam**SYNTAX:**

label cam value,value,etc.

PARAMETERS:

value Incremental cam value.
Range: -127 to +127

DESCRIPTION:

Specifies one-byte incremental values for an electronic cam.

Expressions are allowed. More than one cam value may be contained in a **cam** statement. By placing the **end_cam** statement at the end of the cam table, a cam table terminator (128) is automatically generated.

EXAMPLE

```
cam_array begin_cam
cam 1,2,3,4,5,6,7,8,9,10
cam 10,10,10,10,10,10,10
cam 10,9,8,7,6,5,4,3,2,1
end_cam
```

RETURNS:

None.

USAGE:

MSC-250: cam
MSC-850/32: cam
MSC-850: cam
MSC-800: cam

cam_data**SYNTAX:**

label cam_data controller#,data_label,master_scale,data_scale

PARAMETERS:

controller#	controller id# Range: MSC-250 1 to 2 MSC-850/32 1 to 8 MSC-850 1 to 8 MSC-800 1 to 8
data_label	Program label of the cam data.
master_scale	Number of times to right shift the master position as it is received by the controller. Range: 0 to 12

Master Scale Factor Cam array advances every

12	1 full turn
11	1/2 turn
10	1/4 turn
9	1/8 turn
8	1/16 turn
7	1/32 turn
6	1/64 turn
5	1/128 turn
4	1/256 turn
3	1/512 turn
2	1/1024 turn
1	1/2048 turn
0	1/4096 turn

data_scale	Number of times to left shift each data element. Range: 0 to 7
------------	---

DESCRIPTION:

The **cam_data** instruction provides the specified controller with its' master scale, data scale and cam data array location.

When 'data_label' is dimensioned to reside within program memory, this instruction will also transfer the data array associated with 'data_label' to the specified controller.

When 'data_label' is dimensioned to reside directly on the volatile memory of the axis controller, no transfer of data will occur. Data can be transferred to the controller memory using the **let** or **let_byte** instructions before and/or after the **cam_data** instruction has been issued. This instruction

then serves as a pointer to that data array. More than 1 data array can be dimensioned per controller.

RETURNS:

None.

USAGE:

MSC-250: cam_data
MSC-850/32: cam_data
MSC-850: cam_data
MSC-800: cam_data

case**SYNTAX**

label case num

PARAMETERS:

num A integer value.
Range: -32768 to 32767

DESCRIPTION:

Used with the **select** instruction to designate a branch address. Each **case** value must be unique.

EXAMPLE

```
select      num

    case    1
    .
    .
    exit_select

    case    2
    .
    .
    exit_select

    default
    .
    .
    exit_select

end_select
```

RETURNS:

None.

USAGE:

MSC-250: case
MSC-850/32: case
MSC-850: case
MSC-800: case

close

SYNTAX:

label close unit,status

PARAMETERS:

unit	File identifier. Range: 1 to 8
status	Variable containing the result of the operation.

DESCRIPTION:

This instruction closes a file that was opened via the **create** or **open** instructions. This instruction frees the 'unit' for use with another file. If the file was opened via the **create** instruction, a directory entry is written at this time.

RETURNS:

The return variable will be zero if the operation is successful. Non-zero status codes are described in Section 15.3.2.

USAGE:

MSC-250:	close
MSC-850/32:	close
MSC-850:	close
MSC-800:	close

clr_all_swi**SYNTAX:**

label clr_all_swi

PARAMETERS:

None.

DESCRIPTION:

Disables and clears all 32 (0 through 31) software interrupts.

RETURNS:

None.

USAGE:

MSC-250: clr_all_swi
MSC-850/32: clr_all_swi
MSC-850: clr_all_swi
MSC-800: N/A

clr_bit

SYNTAX:

label clr_bit bit#,variable

PARAMETERS:

bit#	Number of the bit within the 4 byte variable to be cleared to off (logic 0). Range: 0 to 31
variable	The 4 byte area in memory affected by this instruction.

DESCRIPTION:

The specified bit will be cleared (logic 0). If that bit has been previously cleared, it will remain cleared. If that bit was previously set (logic 1), it will now be cleared.

This instruction has no effect on the remaining bits of this 4-byte variable.

This instruction will be ignored if 'bit#' is outside the range of 0 to 31.

RETURNS:

None.

USAGE:

MSC-250:	clr_bit
MSC-850/32:	clr_bit
MSC-850:	clr_bit
MSC-800:	N/A

clr_flag**SYNTAX:**

label clr_flag user_flag#

PARAMETERS:

user_flag# Number of the user flag to be cleared.
Range: 208 to 255

DESCRIPTION:

Clears the specified user flag.

RETURNS:

None.

USAGE:

MSC-250: clr_flag
MSC-850/32: clr_flag
MSC-850: clr_flag
MSC-800: clr_flag

clr_hi_scan**SYNTAX:**

label clr_hi_scan

PARAMETERS:

None.

DESCRIPTION:

Clears a previously executed 'set_hi_scan' instruction.

Sets the I/O Expander scan rate to every 12msec per expander.

i.e. (1 expander - scan rate is every 12msec
2 expanders - scan rate is every 24msec
3 expanders - scan rate is every 36msec
4 expanders - scan rate is every 48msec)

RETURN:

None.

USAGE:

MSC-250: N/A
MSC-850/32: clr_hi_scan
MSC-850: N/A
MSC-800: N/A

clr_local**SYNTAX:**

label clr_local controller#

PARAMETERS:

controller#	controller id #
	Range: MSC-250 1 to 3
	MSC-850/32 1 to 8
	MSC-850 1 to 8
	MSC-800 1 to 9

DESCRIPTION:

Clears the current local zero position and causes the specified controller card to use the current global zero as the absolute zero position.

RETURNS:

None.

USAGE:

MSC-250:	clr_local
MSC-850/32:	clr_local
MSC-850:	clr_local
MSC-800:	clr_local

clr_swi**SYNTAX:**

label clr_swi interrupt#

PARAMETERS:

interrupt# Interrupt number.
Range: 0 to 31

DESCRIPTION:

Purges a previously defined software interrupt.

RETURNS:

None.

USAGE:

MSC-250: clr_swi
MSC-850/32: clr_swi
MSC-850: clr_swi
MSC-800: clr_swi

begin_data

SYNTAX:

label begin_data

PARAMETERS:

None.

DESCRIPTION:

Signals the beginning of a data area which will contain 32-bit (4 byte) values.

This instruction requires a 'label'.

EXAMPLE

profile	begin_data	
	data	100,200,15*4096
	data	0,200,40*4096
	end_data	

RETURNS:

None.

USAGE:

MSC-250:	begin_data
MSC-850/32:	begin_data
MSC-850:	begin_data
MSC-800:	begin_data

begin_cam

SYNTAX:

label begin_cam

PARAMETERS:

None.

DESCRIPTION:

Signals the start of a cam data area which will contain 8 bit (1 byte) values which represent incremental cam data. Each **begin_cam** instruction must have a corresponding **end_cam** instruction.

This instruction requires a 'label'.

EXAMPLE

```
cam_array    begin_cam
              cam    1,2,3,4,5
              cam    6,6,6,6,6
              cam    5,4,3,2,1
              end_cam
```

RETURNS:

None.

USAGE:

```
MSC-250:    begin_cam
MSC-850/32: begin_cam
MSC-850:    begin_cam
MSC-800:    begin_cam
```

declare

SYNTAX:

label declare mode

PARAMETERS:

mode This entry can be only ON or OFF.

DESCRIPTION:

This is a directive to the MacroPro Compiler.

If 'mode' is ON, the Compiler will expect that all variables, text strings, equates etc. will be declared by the programmer. They will not be automatically created by the Compiler.

If 'mode' is OFF, the Compiler will automatically create data space for those variables used but not declared.

The Compiler will assume that 'mode' is OFF if this instruction is not included in the program.

RETURNS:

None.

USAGE:

MSC-250: declare
MSC-850/32: declare
MSC-850: declare
MSC-800: declare

default

SYNTAX:

label default

PARAMETERS:

None.

DESCRIPTION:

Used with the **select** instruction to designate a group of statements which are executed if none of the **case** statements produce a match.

EXAMPLE

```
select x
  case 1
    .
    .
  exit_select

  case 2
    .
    .
  exit_select

  default
  exit_select

end_select
```

RETURNS:

None.

USAGE:

MSC-250: default
MSC-850/32: default
MSC-850: default
MSC-800: default

digi_comp

SYNTAX:

label digi_comp controller#,gain,integral,damp

PARAMETERS:

controller#	controller id#
	Range: MSC-250 1 to 2
	MSC-850/32 1 to 8
	MSC-850 1 to 8
	MSC-800 1 to 8
gain	Range: 1 to 255
integral	Range: 0 to 255
damp	Range: -128 to +127

DESCRIPTION:

Set digital compensation values.

Default values for these parameters are:

gain = 16
integral = 0
damp = 0

It is not necessary to use this instruction if you are using conventional compensation methods.

RETURNS:

None.

USAGE:

MSC-250: digi_comp
MSC-850/32: digi_comp
MSC-850: digi_comp
MSC-800: digi_comp

dim

SYNTAX:

label	dim	size
	or	
label	dim	controller #,size

PARAMETERS:

size	Number of 32-bit storage locations to reserve.
controller#	controller id#
	Range: MSC-250 1 to 2
	MSC-850/32 1 to 8
	MSC-850 1 to 8

DESCRIPTION:

This statement allocates the indicated number of storage locations to the name 'label'. In the MSC-850 and MSC-250, this instruction may be used to allocate all or part of the 28K of controller memory.

This instruction requires a 'label'.

RETURNS:

None.

USAGE:

MSC-250:	dim
MSC-850/32:	dim
MSC-850:	dim
MSC-800:	dim

disable_hwi

SYNTAX:

label disable_hwi controller#

PARAMETERS:

controller#	controller id#	
	Range: MSC-250	1 to 2
	MSC-850/32	1 to 8
	MSC-850	1 to 8

DESCRIPTION:

Terminates scanning of the hardware interrupt signal in the specified axis controller. Also clears the appropriate **HARDWARE INTERRUPT ARMED** status in the controller.

RETURNS:

None.

USAGE:

MSC-250:	disable_hwi
MSC-850/32:	disable_hwi
MSC-850:	disable_hwi
MSC-800:	N/A

disable_swi**SYNTAX:**

label disable_swi

PARAMETERS:

None.

DESCRIPTION:

Disables software interrupt processing.

RETURNS:

None.

USAGE:

MSC-250: disable_swi
MSC-850/32: disable_swi
MSC-850: disable_swi
MSC-800: swi_off

drive_off

SYNTAX:

label drive_off controller#

PARAMETERS:

controller#	controller ID #
	Range: MSC-250 1 to 2
	MSC-850/32 1 to 8
	MSC-850 1 to 8

DESCRIPTION:

Resets all motor fault conditions and puts the specified axis controller into a passive position sensing mode. Turns the servo amplifier off and disables the following error check.

Sets the analog position output to be proportional to the feedback transducer position as seen in Figure 3.10.

NOTE:

For an MSC-850/ACE-850 controller or an MSC-250 controller, the above figure will not be accurate unless an appropriate find marker instruction had been done.

RETURNS:

None.

USAGE:

MSC-250:	drive_off
MSC-850/32:	drive_off
MSC-850:	drive_off
MSC-800:	master

drive_on

SYNTAX:

label drive_on controller#

PARAMETERS:

controller#	controller id#
	Range: MSC-250 1 to 2
	MSC-850/32 1 to 8
	MSC-850 1 to 8
	MSC-800 1 to 8

DESCRIPTION:

Reset all motor fault conditions and turns on the servo amplifier. All controllers must be enabled at least once prior to executing any motion instructions. A controller initializes to 'reset mode' on power-up.

Regarding MSC-850 and MSC-850/32 systems, if an axis controller card is currently in “analog mode” when a **drive_on** instruction is executed, “analog mode” will be disabled and “position loop mode” will be enabled.

RETURNS:

None.

USAGE:

MSC-250:	drive_on
MSC-850/32:	drive_on
MSC-850:	drive_on
MSC-800:	enable

enable_hwi

SYNTAX:

label enable_hwi

PARAMETERS:

The instruction immediately following **enable_hwi**.

DESCRIPTION:

This instruction, when used with supported 'hardware interrupt' instructions, will enable scanning at the specified motion controller for the 'hardware interrupt' signal to be activated.

When the 'hardware interrupt' signal is detected, the specified instruction will immediately be executed. The MacroPro Compiler will associate the instruction that follows 'enable_hwi' as the instruction to be executed.

The following instructions are currently supported:

enable_hwi
over_draw controller id#,speed,limit,distance

enable_hwi
trap_pos controller id#

enable_hwi
ratio controller id#,ratio

enable_hwi
lock controller id#,lock type

enable_hwi
position controller id#,abs_position

enable_hwi
index controller id#,distance

enable_hwi
exec_profile controller id#

enable_hwi
f_decel controller id#

The **HARDWARE INTERRUPT ARMED** status in the controller will be on after this instruction has been executed.

RETURNS:

None.

USAGE:

MSC-250: enable_hwi
MSC-850/32: enable_hwi
MSC-850: enable_hwi
MSC-800: N/A

enable_swi**SYNTAX:**

label enable_swi

PARAMETERS:

None.

DESCRIPTION:

Enables software interrupt processing.

RETURNS:

None.

USAGE:

MSC-250: enable_swi
MSC-850/32: enable_swi
MSC-850: enable_swi
MSC-800: swi_on

end_cam

SYNTAX:

label end_cam

PARAMETERS:

None.

DESCRIPTION:

Signals the end of a data area containing incremental cam data. Each **begin_cam** instruction must have a corresponding **end_cam** instruction.

Automatically places an end of cam value (80 hex) in the cam array.

EXAMPLE

cam_array	begin_cam	
	cam	1,2,3,4,5
	cam	6,6,6,6,6
	cam	5,4,3,2,1
	end_cam	

RETURNS:

None.

USAGE:

MSC-250:	end_cam
MSC-850/32:	end_cam
MSC-850:	end_cam
MSC-800:	end_cam

end_data

SYNTAX:

label end_data

PARAMETERS:

None.

DESCRIPTION:

Signals the end of a data array area. Each **begin_data** instruction must have a corresponding **end_data** instruction.

EXAMPLE

profile	begin_data	
	data	100,200,4096
	data	0,200,4096
	end_data	

RETURNS:

None.

USAGE:

MSC-250:	end_data
MSC-850/32:	end_data
MSC-850:	end_data
MSC-800:	end_data

end_select

SYNTAX:

label end_select

PARAMETERS:

None.

DESCRIPTION:

Used to end a group of **select/case** statements. Each **select** statement must have a corresponding **end_select** statement.

RETURNS:

None.

USAGE:

MSC-250: end_select
MSC-850/32: end_select
MSC-850: end_select
MSC-800: end_select

equ**SYNTAX:**

label equ constant_expression

PARAMETERS:

constant_expression A 32-bit number or expression.

DESCRIPTION:

The **equ** instruction assigns a symbol to a number or mathematic expression.

The MacroPro Compiler replaces each occurrence of 'label' with the assigned number. Fractional numbers are allowed in expressions but any resulting fraction will be truncated.

This instruction requires a 'label'.

RETURNS:

None.

USAGE:

MSC-250: equ
MSC-850/32: equ
MSC-850: equ
MSC-800: equ

exec_profile

SYNTAX:

label exec_profile controller#

PARAMETERS:

controller#	controller id#	
	Range: MSC-250	1 to 3
	MSC-850/32	1 to 8
	MSC-850	1 to 8
	MSC-800	1 to 9

DESCRIPTION:

Executes a previously defined piece-wise profile. While the profile is executing, the **get_pstat** instruction may be used to determine the profile segment currently running.

This instruction sets the controller status flags **BUSY** and **INDEXING**. These flags remain set until a **f_decel** instruction causes the motor to reach zero speed or the profile is completed.

This instruction will be ignored if no valid data has been transmitted to the controller via the **prep_profile** instruction.

This instruction will be ignored if the controller has not received a **drive_on** instruction, or if the controller is busy executing a motion instruction. If the instruction cannot be executed, the axis status flag **COMMAND INVALID IN THIS STATE** will be set. This flag will automatically be cleared by the controller at the next communication between the main processor and the axis controller. In the MSC-850 and the MSC-800, this occurs automatically every 100 milliseconds even if no program instruction directs communication to occur.

RETURNS:

None.

USAGE:

MSC-250:	exec_profile
MSC-850/32:	exec_profile
MSC-850:	exec_profile
MSC-800:	exec_profile

exit_select

SYNTAX:

label exit_select

PARAMETERS:

None.

DESCRIPTION:

Used to end a group of statements being executed within a particular **case** statement. Causes program control to transfer to the statement following the **end_select** statement. Each **case** statement must have a corresponding **exit_select** statement.

EXAMPLE

```
select
    case          1
        .
        exit_select
    case          2
        .
        exit_select
    default
        .
        exit_select
end_select
```

RETURNS:

None.

USAGE:

MSC-250: exit_select
MSC-850/32: exit_select
MSC-850: exit_select
MSC-800: exit_select

f_decel

SYNTAX:

label **f_decel** controller#

PARAMETERS:

controller#	controller id#
	Range: MSC-250 1 to 3
	MSC-850/32 1 to 8
	MSC-850 1 to 8
	MSC-800 1 to 9

DESCRIPTION:

The specified controller will be commanded to stop its motion at the last set accel/decel rate. If its motion is currently stopped, **f_decel** has no effect.

The controller status flag **FORCED DECEL IN PROGRESS** is set by this instruction. It remains set until the controller's motion reaches zero speed.

A mode zero unlock will occur if the controller is currently in a master/slave lock.

If this instruction is directed to an analog controller (i.e. ACM-850), all outputs will slew to zero.

A **f_decel** instruction is automatically sent to all controller cards whenever a **sys_fault** or a **sys_return** instruction is executed.

RETURNS:

None.

USAGE:

MSC-250:	f_decel
MSC-850/32:	f_decel
MSC-850:	f_decel
MSC-800:	f_decel

find_mrk_ccw**SYNTAX:**

label find_mrk_ccw controller#,counts

PARAMETERS:

controller#	controller id#
	Range: MSC-250 1 to 2
	MSC-850/32 1 to 8
	MSC-850 1 to 8
counts	The line count of the encoder multiplied by 4.
	Range: 2048, 4096, 8192 or 16384 counts only.

DESCRIPTION:

The ACE-850 power-up routine sets the ACE-850 to a local mode. This means that the encoder is initially sitting at a 0.0 reference location. This instruction is used to change the 0.0 reference location to be at the marker.

Executing this instruction causes the motor shaft to jog in the counter-clockwise direction (at the previously defined accel/decel rate and speed) until the encoder marker is detected.

At the moment the marker is detected:

- 1) the internal turn/position counters will be zeroed
- 2) the motor will do a forced deceleration to zero speed

At the completion of the deceleration:

- 1) the 'bit loss/auto correction' feature will be enabled
- 2) the position counter is now relative to the marker (i.e. a position to 0.0 will move the motor to the marker)

Before using this instruction, the programmer should do the following:

- 1) turn the motor/drive unit on using the **drive_on** instruction
- 2) set the accel/decel rate, using the **set_ac_dc** instruction
- 3) set the motor speed, using the **set_speed** instruction

NOTE:

This instruction is ONLY valid for line counts of 512, 1024, 2048 and 4096.

RETURNS:

None.

USAGE:

MSC-250:	find_mrk_ccw
MSC-850/32:	find_mrk_ccw
MSC-850:	find_mrk_ccw
MSC-800:	N/A

find_mrk_cw**SYNTAX:**

label find_mrk_cw controller#,counts

PARAMETERS:

controller#	controller id#
	Range: MSC-250 1 to 2
	MSC-850/32 1 to 8
	MSC-850 1 to 8
counts	The line count of the encoder multiplied by 4.
	Range: 2048, 4096, 8192 or 16384 counts only.

DESCRIPTION:

The ACE-850 power-up routine sets the ACE-850 to a local mode. This means that the encoder is initially sitting at a 0.0 reference location. This instruction is used to change the 0.0 reference location to be at the marker.

Executing this instruction causes the motor shaft to jog in the clock-wise direction (at the previously defined accel/decel rate and speed) until the encoder marker is detected.

At the moment the marker is detected:

- 1) the internal turn/position counters will be zeroed
- 2) the motor will do a forced deceleration to zero speed

At the completion of the deceleration:

- 1) the 'bit loss/auto correction' feature will be enabled
- 2) the position counter is now relative to the marker
(i.e. a position to 0.0 will move the motor to the marker)

Before using this instruction, the programmer should do the following:

- 1) turn the motor/drive unit on using the **drive_on** instruction
- 2) set the accel/decel rate, using the **set_ac_dc** instruction
- 3) set the motor speed, using the **set_speed** instruction

NOTE:

This instruction is ONLY valid for line counts of 512, 1024, 2048 and 4096.

RETURNS:

None.

USAGE:

MSC-250:	find_mrk_cw
MSC-850/32:	find_mrk_cw
MSC-850:	find_mrk_cw
MSC-800:	N/A

find_tm_ccw**SYNTAX:**

label **find_tm_ccw** controller#,counts

PARAMETERS:

controller#	controller id#
	Range: MSC-250 1 to 2
	MSC-850/32 1 to 8
	MSC-850 1 to 8
counts	The line count of the encoder multiplied by 4.
	Range: 2048, 4096, 8192 or 16384 counts only.

DESCRIPTION:

The ACE-850 power-up routine sets the ACE-850 to a local mode. This means that the encoder is initially sitting at a 0.0 reference location. This instruction is used to change the 0.0 reference location to be at the marker. This instruction is used for markers which stay on for 180 degrees.

Executing this instruction causes the motor shaft to jog in the counter-clockwise direction (at the previously defined accel/decel rate and speed) until the encoder marker is detected.

At the moment the marker is detected:

- 1) the internal turn/position counters will be zeroed
- 2) the motor will do a forced deceleration to zero speed

At the completion of the deceleration:

- 1) the 'bit loss/auto correction' feature will be enabled
- 2) the position counter is now relative to the marker
(i.e. a position to 0.0 will move the motor to the marker)

Before using this instruction, the programmer should do the following:

- 1) turn the motor/drive unit on using the **drive_on** instruction
- 2) set the accel/decel rate, using the **set_ac_dc** instruction
- 3) set the motor speed, using the **set_speed** instruction

NOTE:

This instruction is ONLY valid for line counts of 512, 1024, 2048 and 4096.

RETURNS:

None.

USAGE:

MSC-250:	find_tm_ccw
MSC-850/32:	find_tm_ccw
MSC-850:	find_tm_ccw
MSC-800:	N/A

find_tm_cw**SYNTAX:**

label find_tm_cw controller#,counts

PARAMETERS:

controller#	controller id#
	Range: MSC-250 1 to 2
	MSC-850/32 1 to 8
	MSC-850 1 to 8
counts	The line count of the encoder multiplied by 4.
	Range: 2048, 4096, 8192 or 16384 counts only.

DESCRIPTION:

The ACE-850 power-up routine sets the ACE-850 to a local mode. This means that the encoder is initially sitting at a 0.0 reference location. This instruction is used to change the 0.0 reference location to be at the marker. This instruction is used for markers which stay on for 180 degrees.

Executing this instruction causes the motor shaft to jog in the clock-wise direction (at the previously defined accel/decel rate and speed) until the encoder marker is detected.

At the moment the marker is detected:

- 1) the internal counters will be set to $-(\text{counts}/2)$
(i.e. the actual 0.00 point is 180 degrees clockwise away, this insures that the instructions **find_tm_cw** and **find_tm_ccw** find the same 0.00 reference location)
- 2) the motor will do a forced deceleration to zero speed

At the completion of the deceleration:

- 1) the 'bit loss/auto correction' feature will be enabled
- 2) the position counter is now relative to the marker
(i.e. a position to 0.0 will move the motor to the marker)

Before using this instruction, the programmer should do the following:

- 1) turn the motor/drive unit on using the **drive_on** instruction
- 2) set the accel/decel rate, using the **set_ac_dc** instruction
- 3) set the motor speed, using the **set_speed** instruction

NOTE:

This instruction is ONLY valid for line counts of 512, 1024, 2048 and 4096.

RETURNS:

None.

USAGE:

MSC-250: find_tm_cw

MSC-850/32: find_tm_cw

MSC-850: find_tm_cw

MSC-800: N/A

get_act_spd

SYNTAX:

label get_act_spd controller#,variable

PARAMETERS:

controller#	controller id#
	Range: MSC-250 1 to 3
	MSC-850/32 1 to 8
	MSC-850 1 to 8
variable	Current motion controller speed in RPM. (4096 bits per revolution are assumed)

DESCRIPTION:

Returns the actual speed at the motor shaft of the specified motion controller (in RPM) into 'variable'.

RETURNS:

Actual axis speed.

USAGE:

MSC-250:	get_act_spd
MSC-850/32:	get_act_spd
MSC-850:	get_act_spd
MSC-800:	get_mspd (get master axis speed)

get_angle**SYNTAX:**

label get_angle controller#,variable

PARAMETERS:

controller#	controller ID # Range: MSC-250 1 to 2 MSC-850/32 1 to 8 MSC-850 1 to 8
variable	The current angle of the master axis driving this controller in master/slave lock.

DESCRIPTION:

Returns the current angle of the master axis driving this axis controller in master/slave lock.

RETURNS:

master angle

USAGE:

MSC-250:	get_angle
MSC-850/32:	get_angle
MSC-850:	get_angle
MSC-800:	N/A

get_angle (MSC-850/HPL-850)**SYNTAX:**

label get_angle controller#,variable

PARAMETERS:

controller#	controller ID # Range: MSC-850/32 1 to 8 MSC-850 1 to 8
variable	The value of the HPL-850 accumulator.

DESCRIPTION:

Returns the current HPL-850 accumulator.

RETURNS:

The current HPL-850 accumulator value is placed into 'variable'.

USAGE:

MSC-250:	N/A
MSC-850/32:	get_angle
MSC-850:	get_angle
MSC-800:	N/A

get_cam_cnt

SYNTAX:

label get_cam_cnt controller#,variable

PARAMETERS:

controller#	controller id#
	Range: MSC-250 1 to 2
	MSC-850/32 1 to 8
	MSC-850 1 to 8
variable	The returned cam counter.

DESCRIPTION:

Returns the number of executions of the currently executing cam. The cam counter is incremented when the cam pointer rolls over from last element to first element, and is decremented each time the cam pointer rolls over from first element to last element.

RETURNS:

The number of times this cam has been executed. This value is signed 32-bit number.

USAGE:

MSC-250:	get_cam_cnt
MSC-850/32:	get_cam_cnt
MSC-850:	get_cam_cnt
MSC-800:	N/A

get_cam_end**SYNTAX:**

label get_cam_end controller#,variable

PARAMETERS:

controller#	controller id#
	Range: MSC-250 1 to 2
	MSC-850/32 1 to 8
	MSC-850 1 to 8
variable	The last element in the current cam.

DESCRIPTION:

Returns the ending location of the cam currently being executed, relative to the axis card memory location zero.

RETURNS:

Returns the element number of the last value in the current cam.

USAGE:

MSC-250:	get_cam_end
MSC-850/32:	get_cam_end
MSC-850:	get_cam_end
MSC-800:	N/A

get_cam_ptr

SYNTAX:

label get_cam_ptr controller#,variable

PARAMETERS:

controller#	controller id#
	Range: MSC-250 1 to 2
	MSC-850/32 1 to 8
	MSC-850 1 to 8
	MSC-800 1 to 8
variable	Returned cam array member.

DESCRIPTION:

Stores the cam array element relative to the beginning of the current cam into a variable.

RETURNS:

The number of the current cam element.

USAGE:

MSC-250:	get_cam_ptr
MSC-850/32:	get_cam_ptr
MSC-850:	get_cam_ptr
MSC-800:	store_frm

get_cam_strt**SYNTAX:**

label get_cam_strt controller#,variable

PARAMETERS:

controller#	controller id#
	Range: MSC-250 1 to 2
	MSC-850/32 1 to 8
	MSC-850 1 to 8
variable	The first element in the current cam.

DESCRIPTION:

Retrieves the starting location of the cam currently executed, relative to axis memory location zero.

RETURNS:

The first element of the current cam relative to axis controller memory zero.

USAGE:

MSC-250:	get_cam_strt
MSC-850/32:	get_cam_strt
MSC-850:	get_cam_strt
MSC-800:	N/A

get_cam_sum

SYNTAX:

label get_cam_sum controller#,variable

PARAMETERS:

controller#	controller id#
	Range: MSC-250 1 to 2
	MSC-850/32 1 to 8
	MSC-850 1 to 8
variable	Returned sum of cam elements.

DESCRIPTION:

Returns the result of the last **calc_cam_sum** instruction in 'variable'. This instruction may be executed after the instruction 'calc_cam_sum' has been executed, and the axis status flag CALCULATION IN PROGRESS has gone from ON to OFF.

RETURNS:

The calculated cam sum.

USAGE:

MSC-250:	get_cam_sum
MSC-850/32:	get_cam_sum
MSC-850:	get_cam_sum
MSC-800:	N/A

get_com**SYNTAX:**

label get_com controller#,variable

PARAMETERS:

controller#	controller id#
	Range: MSC-250 1 to 3
	MSC-850/32 1 to 8
	MSC-850 1 to 8
variable	Returned position.

DESCRIPTION:

Gets the absolute commanded motor position of the specified controller and places this position in 'variable'. In the MSC-850/32, MSC-850 and MSC-800 controllers, position is a signed 24 bit number. In the MSC-250, position is a signed 32-bit number.

RETURNS:

Axis controllers commanded position.

USAGE:

MSC-250:	get_com
MSC-850/32:	get_com
MSC-850:	get_com
MSC-800:	store_com

get_fol_err

SYNTAX:

label get_fol_err controller#,variable

PARAMETERS:

controller#	controller id#
	Range: MSC-250 1 to 2
	MSC-850/32 1 to 8
	MSC-850 1 to 8
variable	The current following error angle.

DESCRIPTION:

Returns the current following error angle. This value is the difference between the current commanded position and the current actual position and is represented as a signed value with a normal range of ± 4095 . Altering the digital gain to less than 16 may result in a value which exceeds normal range. CCW will return negative values, and CW motion will return positive values.

RETURNS:

Returns the current difference between the axis commanded and actual positions.

USAGE:

MSC-250:	get_fol_err
MSC-850/32:	get_fol_err
MSC-850:	get_fol_err
MSC-800:	N/A

get_for_ang

SYNTAX:

label get_for_ang controller#,channel#,variable

PARAMETERS:

controller#	controller id#
	Range: MSC-250 Always 3
	MSC-850/32 1 to 8
	MSC-850 1 to 8
channel#	Range: MSC-250 Always 1
	MSC-850/32 1 to 2
	MSC-850 1 to 2
variable	The current fiber optic angle.

DESCRIPTION:

Returns the current fiber optic angle for the specified channel.

RETURNS:

Fiber optic angle. This value is a signed 16-bit number with the range -32768 to +32767.

USAGE:

MSC-250:	get_for_ang
MSC-850/32:	get_for_ang
MSC-850:	get_for_ang
MSC-800:	N/A

get_for_spd

SYNTAX:

label get_for_spd controller#,channel#,variable

PARAMETERS:

controller#	controller id#
	Range: MSC-250 Always 3
	MSC-850/32 1 to 8
channel#	Range: MSC-250 Always 1
	MSC-850/32 1 to 2
variable	The current speed in RPM.

DESCRIPTION:

Returns the current speed in RPM from the device connected to the fiber optic channel. This speed is based on 4096 bits per motor revolution at this device.

RETURNS:

Speed of the device connected to the fiber optic channel. This value is a signed 16-bit number with the range -32768 to +32767.

USAGE:

MSC-250:	get_for_spd
MSC-850/32:	get_for_spd
MSC-850:	N/A
MSC-800:	N/A

get_map

SYNTAX:

label **get_map** variable

PARAMETERS:

variable The currently defined map value.

DESCRIPTION:

Queries the current definition of the master angle bus communication configuration defined by the last **set_map** instruction.

RETURNS:

Returns the current master angle bus configuration. The variable may be interpreted as seen in Figure 3.29 and Figure 3.30.

USAGE:

MSC-250:	get_map
MSC-850/32:	get_map
MSC-850:	get_map
MSC-800:	N/A

get_map_stat

SYNTAX:

label get_map_stat variable

PARAMETERS:

variable Returned 'map' status.

DESCRIPTION:

'map' is an acronym for Master Angle Passing.

Returns the status of the last **set_map** instruction.

Status	Description

0	Valid 'map' value.
1	More than 1 transmitter on Bus 'A'.
2	More than 1 transmitter on Bus 'B'.

RETURNS:

Status of last **set_map** instruction executed.

USAGE:

MSC-250: get_map_stat
MSC-850/32: get_map_stat
MSC-850: get_map_stat
MSC-800: N/A

get_mcf

SYNTAX:

label get_mcf controller#,variable

PARAMETERS:

controller#	controller id#
	Range: MSC-250 Always 3
	MSC-850/32 1 to 8
	MSC-850 1 to 8
variable	The currently defined 'mcf' value.

DESCRIPTION:

'mcf' is an acronym for Multi Function Controller.

Returns the current definition of the specified multi function controller card, as defined by the last valid **set_mcf** instruction.

This 4-byte variable is used as shown in Figure 3.31 and Figure 3.32.

USAGE:

MSC-250:	get_mcf
MSC-850/32:	get_mcf
MSC-850:	get_mcf
MSC-800:	N/A

get_pls_mask

SYNTAX:

label get_pls_mask controller#,variable

PARAMETERS:

controller#	controller id#
	Range: MSC-250 Always 3
	MSC-850/32 1 to 8
	MSC-850 1 to 8
variable	The currently defined 'pls' mask value.

DESCRIPTION:

'pls' is an acronym for Programmable Limit Switch.

Returns the currently defined 'pls' mask value into 'variable', as defined by the last **set_pls_mask** instruction.

Only the low order 3 bytes are used.

RETURNS:

The currently defined pls mask.

USAGE:

MSC-250:	get_pls_mask
MSC-850/32:	get_pls_mask
MSC-850:	get_pls_mask
MSC-800:	N/A

get_pls_out

SYNTAX:

label get_pls_out controller#,variable

PARAMETERS:

controller#	controller id#
	Range: MSC-250 Always 3
	MSC-850/32 1 to 8
	MSC-850 1 to 8
variable	The current state of the 'pls' output modules.

DESCRIPTION:

'pls' is an acronym for Programmable Limit Switch.

Returns the current state of the 'pls' output modules into 'variable'. A bit on (logic 1) indicates that the associated module is 'on'. A bit off (logic 0) indicates that the associated output is 'off'.

Only the low order 3 bytes are used.

Bits 0 to 15 are the hardware 'pls' flags.

Bits 16 to 23 are the internal 'pls' flags.

RETURNS:

Current state of PLS outputs.

USAGE:

MSC-250:	get_pls_out
MSC-850/32:	get_pls_out
MSC-850:	get_pls_out
MSC-800:	N/A

get_pq_space**SYNTAX:**

label get_pq_space variable

PARAMETERS:

variable Return variable.

DESCRIPTION:

Returns the number of bytes available in the output buffer. There is one buffer for all ports. If another 'port_set' instruction is executed, the previous prints still get executed. The size of the buffer is 3060 bytes. Any 'prints' executed which exceed the size of the buffer are lost.

RETURNS:

Number of available bytes in the output buffer.

USAGE:

MSC-250: get_pq_space

MSC-850/32: get_pq_space

MSC-850: N/A

MSC-800: N/A

get_pos**SYNTAX:**

label get_pos controller#,variable

PARAMETERS:

controller#	controller id#
	Range: MSC-250 1 to 3
	MSC-850/32 1 to 8
	MSC-850 1 to 8
	MSC-800 1 to 8
variable	Returned position.

DESCRIPTION:

Gets the absolute actual motor position of the specified controller and places this position in 'variable'. The position is a signed 24-bit value.

RETURNS:

The current actual motor position.

USAGE:

MSC-250:	get_pos
MSC-850/32:	get_pos
MSC-850:	get_pos
MSC-800:	store_pos

get_pstat**SYNTAX:**

label **get_pstat** controller#,status

PARAMETERS:

controller#	controller id#
	Range: MSC-250 1 to 3
	MSC-850/32 1 to 8
	MSC-850 1 to 8
	MSC-800 1 to 9
status	Returned status.

DESCRIPTION:

Returns the calculation or running status of a Piecewise profile.

- A. After a profile calculation has been performed, **get_pstat** is used to check the results of the calculations. If there were no errors, 'status' is zero. If any part of the calculations failed, then **get_pstat** will return a calculation error code and the number of the profile segment that caused the error as follows:

Least significant byte: profile segment number in error
 Next significant byte: error code (see below)

- B. While a profile is executing, **get_pstat** returns the number of the profile segment that is being executed.

RETURNS:

Error Codes:

<u>Code</u>	<u>Meaning</u>
-------------	----------------

- | | |
|---|--|
| 1 | Attempt to change profile direction. |
| 3 | Insufficient distance for specified speed and/or acceleration. |

The following example represents a 'status' where the profile segment number in error is 10 and the error code for that segment is 3:

status: 778 (decimal) status: 0000 030A (hexadecimal)

USAGE:

MSC-250:	get_pstat
MSC-850/32:	get_pstat
MSC-850:	get_pstat
MSC-800:	get_pstat

get_space**SYNTAX:**

label get_space unit,space,status

PARAMETERS:

unit	File identifier. Range: 1 to 8
space	Variable containing the number of remaining bytes.
status	Variable containing the result of the operation.

DESCRIPTION:

This instruction returns to 'space' the number of remaining bytes (characters) in the EPROM.

RETURNS:

The return variable will be zero if the operation is successful. Non-zero status codes are described in Section 15.3.2.

USAGE:

MSC-250:	get_space
MSC-850/32:	get_space
MSC-850:	get_space
MSC-800:	get_space

get_status**SYNTAX:**

label get_status controller#

PARAMETERS:

controller#	controller ID #
	Range: MSC-850/32 1 to 8
	MSC-850 1 to 8
	MSC-800 1 to 8

DESCRIPTION:

Updates the axis status flags for the specified controller. This insures the axis status flags contain the most current information.

In the MSC-850/32, MSC-850, and MSC-800, if this instruction is not used, the axis status flags are automatically updated every 100 milliseconds.

RETURNS:

None.

USAGE:

MSC-250:	N/A
MSC-850/32:	get_status
MSC-850:	get_status
MSC_800:	get_status

get_t_mark

SYNTAX:

label get_t_mark controller#,state

PARAMETERS:

controller#	controller id#
	Range: MSC-250 1 to 2
	MSC-850/32 1 to 8
	MSC-850 1 to 8
state	The current marker state.
	Range: 0 or 1

DESCRIPTION:

The value of 'state' will be either one (1) if the marker is on or zero (0) if the marker is off.

RETURNS:

The current state of the encoder marker.

USAGE:

MSC-250:	get_t_mark
MSC-850/32:	get_t_mark
MSC-850:	get_t_mark
MSC-800:	N/A

get_time**SYNTAX:**

label get_time variable

PARAMETERS:

variable Returned time.

DESCRIPTION:

Return the current system time in 'variable'. The 'variable' is an unsigned 32-bit data value. The resolution of the timer is 5 milliseconds.

The timer will reset to 0 during system power up.

RETURNS:

Current system time from power-up.

USAGE:

MSC-250: get_time
MSC-850/32: get_time
MSC-850: get_time
MSC-800: store_time

get_trap_pos

SYNTAX:

label get_trap_pos controller#,variable

PARAMETERS:

controller#	controller id#	
	Range: MSC-250	1 to 2
	MSC-850/32	1 to 8
	MSC-850	1 to 8
variable	Returned position.	
	Range: MSC-850/32	(-2048*4096) to (+2048*4096)-1 bits
	MSC-850	(-2048*4096) to (+2048*4096)-1 bits
	MSC-250	(-524287*4096) to (+524287*4096)-1 bits

DESCRIPTION:

Returns the actual motor position of the specified controller (saved by executing the last **trap_pos** instruction) into 'variable'.

RETURNS:

Returns last trapped position.

USAGE:

MSC-250:	get_trap_pos
MSC-850/32:	get_trap_pos
MSC-850:	get_trap_pos
MSC-800:	N/A

get_volume

SYNTAX:

label get_volume unit,data_area,status

PARAMETERS:

unit	File identifier. Range: 1 to 8
data_area	Label of a ' text ' instruction.
status	Variable containing the result of the operation.

DESCRIPTION:

This instruction reads the EPROM label area and places this volume name in the 'data_area'. The volume name can be from 1 to 15 characters.

RETURNS:

The return variable will be zero if the operation is successful. Non-zero status codes are described in Section 15.3.2.

USAGE:

MSC-250:	get_volume
MSC-850/32:	get_volume
MSC-850:	get_volume
MSC-800:	get_volume

gosub

SYNTAX:

label gosub subroutine_label

PARAMETERS:

subroutine_label Program label of the subroutine.

DESCRIPTION:

Branches to the specified address. When a **return_sub** instruction is executed the program branches to the instruction following the **gosub** instruction.

The MSC provides 20 levels of subroutine nesting. The **STACK OVERFLOW** status flag will be set if more than 20 levels are used.

RETURNS:

None.

USAGE:

MSC-250: gosub
MSC-850/32: gosub
MSC-850: gosub
MSC-800: gosub

goto**SYNTAX:**

label goto address_label

PARAMETERS:

address_label Label where execution will continue.

DESCRIPTION:

Branches to the specified address label.

RETURNS:

None.

USAGE:

MSC-250: goto
MSC-850/32: goto
MSC-850: goto
MSC-800: goto

if

SYNTAX:

label if compare1 operator compare2, address_label

PARAMETERS:

compare1	Variable or constant to be compared.
operator	Type of comparison to be performed.
compare2	Variable or constant to be compared.
address_label	Branch address if comparison is true.

DESCRIPTION:

Performs arithmetic comparison and causes branching to the specified address if the comparison is true.

If the comparison is false, no branching occurs and execution continues with the next instruction.

Operator symbols	Description
$=$	Equal to
\neq	Not equal to
$>$	Greater than
$<$	Less than
\geq	Greater than or equal to
\leq	Less than or equal to

RETURNS:

None.

USAGE:

MSC-250:	if
MSC-850/32:	if
MSC-850:	if
MSC-800:	if

if_bit_clr

SYNTAX:

label if_bit_clr bit#,variable,address_label

PARAMETERS:

bit#	Bit number of the 4-byte variable to be tested. Range: 0 to 31
variable	The 4-byte variable to be tested.
address_label	Branch address.

DESCRIPTION:

Branch to the specified address if the specified bit is cleared (logic 0).

If the bit tested is on (logic 1), no branching occurs and execution continues with the next instruction.

This instruction will be ignored if 'bit#' is outside the range of 0 to 31.

RETURNS:

None.

USAGE:

MSC-250:	if_bit_clr
MSC-850/32:	if_bit_clr
MSC-850:	if_bit_clr
MSC-800:	N/A

if_bit_set

SYNTAX:

label if_bit_set bit#,variable,address_label

PARAMETERS:

bit#	Bit number of the 4-byte variable to be tested. Range: 0 to 31
variable	The 4-byte variable to be tested.
address_label	Branch address.

DESCRIPTION:

Branch to the specified address if the specified bit is set to on (logic 1).

If the bit tested is off (logic 0), no branching occurs and execution continues with the next instruction.

This instruction will be ignored if 'bit#' is outside the range of 0 to 31.

RETURNS:

None.

USAGE:

MSC-250:	if_bit_set
MSC-850/32:	if_bit_set
MSC-850:	if_bit_set
MSC-800:	N/A

if_char**SYNTAX:**

label if_char port#,address_label

PARAMETERS:

port#	MSC communications port number.
	Range: MSC-250 1,2,3
	MSC-850/32 0I,0R,0,1,2,3
	MSC-850 0,2
	MSC-800 0,2
address_label	Branch address.

DESCRIPTION:

Branches to the specified address if characters are sensed at the specified port.

If no characters are sensed, execution continues with the next instruction.

RETURNS:

None.

USAGE:

MSC-250:	if_char
MSC-850/32:	if_char
MSC-850:	if_char
MSC-800:	if_input_on

if_flag_off**SYNTAX:**

label if_flag_off user_flag#,address_label

PARAMETERS:

user_flag#	Flag number to test. Range: 208 to 255
address_label	Branch address.

DESCRIPTION:

Branch to the specified address if the specified user flag is off.

If the user flag is on, no branching occurs and execution continues with the next instruction.

RETURNS:

None.

USAGE:

MSC-250:	if_flag_off
MSC-850/32:	if_flag_off
MSC-850:	if_flag_off
MSC-800:	if_flag_off

if_flag_on**SYNTAX:**

label if_flag_on user_flag#,address_label

PARAMETERS:

user_flag#	Flag number to test. Range: 208 to 255
address_label	Branch address.

DESCRIPTION:

Branch to the specified address if the specified user flag is on.

If the user flag is not on, no branching occurs and execution continues with the next instruction.

RETURNS:

None.

USAGE:

MSC-250:	if_flag_on
MSC-850/32:	if_flag_on
MSC-850:	if_flag_on
MSC-800:	if_flag_on

if_io_off**SYNTAX:**

label *if_io_off* I/O flag#,address_label

PARAMETERS:

I/O flag#	Input or Output module number.
	Range: MSC-250 0 to 47
	MSC-850/32 0 to 71
	MSC-850 0 to 71
	MSC-800 0 to 71
address_label	Branch address.

DESCRIPTION:

Branches to the specified address if the specified I/O flag is off.

If the I/O flag is on, branching does not occur and execution continues with the next instruction.

RETURNS:

None.

USAGE:

MSC-250:	<i>if_io_off</i>
MSC-850/32:	<i>if_io_off</i>
MSC-850:	<i>if_io_off</i>
MSC-800:	<i>if_io_off</i>

if_io_on**SYNTAX:**

label if_io_on I/O flag#,address_label

PARAMETERS:

I/O flag#	Input or Output module number.
	Range: MSC-250 0 to 47
	MSC-850/32 0 to 71
	MSC-850 0 to 71
	MSC-800 0 to 71
address_label	Branch address.

DESCRIPTION:

Branches to the specified address if the specified I/O flag is on.

If the I/O flag is off, branching does not occur and execution continues with the next instruction.

RETURNS:

None.

USAGE:

MSC-250:	if_io_on
MSC-850/32:	if_io_on
MSC-850:	if_io_on
MSC-800:	if_io_on

if_no_char**SYNTAX:**

label if_no_char port#,address_label

PARAMETERS:

port#	MSC communications port number.
	Range: MSC-250 1,2,3
	MSC-850/32 0I,0R,0,1,2,3
	MSC-850 0,2
	MSC-800 0,2
address_label	Branch address.

DESCRIPTION:

Branches to the specified address if no characters are sensed at the specified port.

If characters are sensed, execution continues with the next instruction.

RETURNS:

None.

USAGE:

MSC-250:	if_no_char
MSC-850/32:	if_no_char
MSC-850:	if_no_char
MSC-800:	if_input_off

if_stat_on**SYNTAX:**

label if_stat_on status_flag #,address_label

PARAMETERS:

status_flag#	Controller status flag.
	Range: MSC-250 80 to 143
	MSC-850/32 80 to 207
	MSC-850 80 to 207
	MSC-800 80 to 207
address_label	Branch address.

DESCRIPTION:

Branches to the specified address label if the controller status flag indicated is on.

If the controller status flag is off, branching does not occur and execution continues with the next instruction.

RETURNS:

None.

USAGE:

MSC-250:	if_stat_on
MSC-850/32:	if_stat_on
MSC-850:	if_stat_on
MSC-800:	if_stat_on

if_stat_off

SYNTAX:

label if_stat_off status_flag #,address_label

PARAMETERS:

status_flag#	Controller status flag.
	Range: MSC-250 80 to 143
	MSC-850/32 80 to 207
	MSC-850 80 to 207
	MSC-800 80 to 207
address_label	Branch address.

DESCRIPTION:

Branches to the specified address label if the controller status flag indicated is off.

If the controller status flag is on, branching does not occur and execution continues with the next instruction.

RETURNS:

None.

USAGE:

MSC-250:	if_stat_off
MSC-850/32:	if_stat_off
MSC-850:	if_stat_off
MSC-800:	if_stat_off

if_tmr_off**SYNTAX:**

label if_tmr_off timer_flag#,address_label

PARAMETERS:

timer_flag#	Timer flag number. Range: 72 to 79
address_label	Branch address.

DESCRIPTION:

Branches to the specified address if the timer flag indicated is off.

If the timer flag is on, branching does not occur and execution continues with the next instruction.

RETURNS:

None.

USAGE:

MSC-250:	if_tmr_off
MSC-850/32:	if_tmr_off
MSC-850:	if_tmr_off
MSC-800:	if_tmr_off

if_tmr_on**SYNTAX:**

label if_tmr_on timer_flag#,address_label

PARAMETERS:

timer_flag#	Timer flag number. Range: 72 to 79
address_label	Branch address.

DESCRIPTION:

Branches to the specified address if the timer flag indicated is on.

If the timer flag is off, branching does not occur and execution continues with the next instruction.

RETURNS:

None.

USAGE:

MSC-250:	if_tmr_on
MSC-850/32:	if_tmr_on
MSC-850:	if_tmr_on
MSC-800:	if_tmr_on

incr_com

SYNTAX:

label incr_com controller#,bits,interrupts

PARAMETERS:

controller#	controller id# Range: MSC-250 1 to 2 MSC-850/32 1 to 8 MSC-850 1 to 8
bits	The incremental value, in bits, to be added to the commanded position.
interrupts	The number of motor interrupts to move the distance specified in 'bits'. Motor interrupts occur every 1 ms in the MSC-850/32 & MSC-850 and every 500 ms in the MSC-250.

DESCRIPTION:

This instruction calculates the number of bits to be added to the commanded position, each motor interrupt as a result of the calculations 'bits/motor interrupts'. Resulting accel/decel rates are not limited. 'bits' may be signed for direction.

RETURNS:

None.

USAGE:

MSC-250:	incr_com
MSC-850/32:	incr_com
MSC-850:	incr_com
MSC-800:	N/A

index

SYNTAX:

label index controller#,distance

PARAMETERS:

controller#	controller id#
	Range: MSC-250 1 to 3
	MSC-850/32 1 to 8
	MSC-850 1 to 8
	MSC-800 1 to 9
distance	Incremental number of bits to move.
	Range: MSC-250 (-524287*4096) to (+524287*4096) -1
	MSC-850/32 (-2048 * 4096) to (+2048 * 4096) -1
	MSC-850 (-2048 * 4096) to (+2048 * 4096) -1
	MSC-800 (-2048 * 4096) to (+2048 * 4096) -1

DESCRIPTION:

Commands the specified controller to index the motor the distance given. The speed of the move is determined by the previously set accel/decel rate and speed.

The MSC has a resolution of 4096 bits per turn. To index 1 turn, distance would be 4096.

The axis controller status flags **MOTOR BUSY** and **INDEXING** will be set (ON) while the motor is indexing.

This instruction will be ignored if the controller has not received a **drive_on** instruction or if the controller is busy executing a motion instruction. If the instruction cannot be executed, the axis status flag **COMMAND INVALID IN THIS STATE** will be set. This flag will automatically be cleared by the controller at the next communication between the main processor and the axis controller. In the MSC-850 and MSC-800 this automatically occurs every 100 milliseconds even if no program instruction directs communication to occur.

RETURNS:

Position

USAGE:

MSC-250:	index
MSC-850/32:	index
MSC-850:	index
MSC-800:	index

initialize

SYNTAX:

label initialize unit,data_area,status

PARAMETERS:

unit	File identifier. Range: 1 to 8
data_area	Label of a 'text' instruction containing the volume name.
status	Variable containing the result of the operation.

DESCRIPTION:

This instruction writes a volume name from 'data_area' to the EPROM label area. Only one name may be written to a given EPROM.

RETURNS:

None.

USAGE:

MSC-250:	initialize
MSC-850/32:	initialize
MSC-850:	initialize
MSC-800:	initialize

input

SYNTAX:

label input label,length,decimals,variable,user_flag

PARAMETERS:

label	ASCII string (prompt) to be displayed
length	Maximum length of input value.
decimals	Decimal places in the input value.
variable	The value entered.
user_flag	Flag indicating input is complete.

DESCRIPTION:

Prepares and reads numeric (only) information from an MSC port. The port is selected by using the **port_set** instruction.

Input string 'length' determines the number of characters accepted during input. The number of characters includes the sign and decimal point (if applicable) as well as the number of characters.

'Decimal places' indicates the number of values to the right of the decimal point.

Input 'variable' is the destination address of the numeric input.

The **port_set** instruction must be executed prior to invoking the **input** instruction.

The characters allowed while entering strings with length exceeding 1 character are +,-1234567890.

If the value being entered has an actual character length greater than the specified input string 'length', the excess characters are ignored. Actual string lengths less than the input string length are permitted and are justified accordingly.

RETURNS:

The return 'variable' contains the numeric conversion of the ASCII string entered at the terminal. This number value is value * (10 ^ decimals).

EXAMPLE

If 2.2 is entered
and decimals = 1
then value = $2.2 * (10 ^ 1) = 2.2 * 10 = 22$

USAGE:

MSC-250: input

MSC-850/32: input

MSC-850: input

MSC-800: input

integer

SYNTAX:

label integer

PARAMETERS:

None.

DESCRIPTION:

This statement assigns a 32 bit storage location, or variable, to the name 'label'. The storage location will be initialized to zero when the program is loaded into the MSC Controller.

This instruction requires a 'label'.

RETURNS:

None.

USAGE:

MSC-250: integer

MSC-850/32: integer

MSC-850: integer

MSC-800: integer

jog_ccw

SYNTAX:

label jog_ccw controller#

PARAMETERS:

controller#	controller id#
	Range: MSC-250 1 to 3
	MSC-850/32 1 to 8
	MSC-850 1 to 8
	MSC-800 1 to 9

DESCRIPTION:

Commands the specified controller to turn the motor shaft in a counter-clockwise direction using the last set accel/decel rate and speed. Motion will continue until an **f_decel** instruction is executed.

This instruction sets the controller status flags **BUSY** and **JOGGING**. These flags remain set until an **f_decel** instruction causes the motor to reach zero speed.

This instruction will be ignored if the controller has not received a **drive_on** instruction, or if the controller is busy executing a motion instruction. If the instruction cannot be executed, the axis status flag **COMMAND INVALID IN THIS STATE** will be set. This flag will automatically be cleared by the controller at the next communication between the main processor and the axis controller. This occurs automatically every 100 milliseconds even if no program instruction directs communication to occur.

RETURNS:

None.

USAGE:

MSC-250:	jog_ccw
MSC-850/32:	jog_ccw
MSC-850:	jog_ccw
MSC-800:	jog_ccw

jog_cw

SYNTAX:

label jog_cw controller#

PARAMETERS:

controller#	controller id#
	Range: MSC-250 1 to 3
	MSC-850/32 1 to 8
	MSC-850 1 to 8
	MSC-800 1 to 9

DESCRIPTION:

Commands the specified controller to turn the motor shaft in a clockwise direction using the last set accel/decel rate and speed. Motion will continue until an **f_decel** instruction is executed.

This instruction sets the controller status flags **BUSY** and **JOGGING**. These flags remain set until an **f_decel** instruction causes the motor to reach zero speed.

This instruction will be ignored if the controller has not received a **drive_on** instruction, or if the controller is busy executing a motion instruction. If the instruction cannot be executed, the axis status flag **COMMAND INVALID IN THIS STATE** will be set. This flag will automatically be cleared by the controller at the next communication between the main processor and the axis controller. This occurs automatically every 100 milliseconds even if no program instruction directs communication to occur.

RETURNS:

None.

USAGE:

MSC-250:	jog_cw
MSC-850/32:	jog_cw
MSC-850:	jog_cw
MSC-800:	jog_cw

l_track_spd

SYNTAX:

label l_track_spd controller#,speed

PARAMETERS:

controller#	controller id#
	Range: MSC-250 1 to 3
	MSC-850/32 1 to 8
	MSC-850 1 to 8
	MSC-800 1 to 9
speed	Entry Range: -32768 to +32767
	Resulting Speed: -128 to +127 RPM

DESCRIPTION:

The specified controller tracks (changes to) the speed indicated divided by 256. All speed changes occur at the previously set accel/decel rate divided by 256.

The speed may be changed at any time.

This instruction sets the controller status flags **BUSY** and **JOGGING**. These flags remain set until a **f_decel** instruction causes the motor to reach zero speed.

This instruction will be ignored if the controller has not received a **drive_on** instruction, or if the controller is busy executing a motion instruction. If the instruction cannot be executed, the axis status flag **COMMAND INVALID IN THIS STATE** will be set. This flag will automatically be cleared by the controller at the next communication between the main processor and the axis processor. In the MSC-850, this occurs automatically every 100 milliseconds even if no program instruction directs communication to occur.

RETURNS:

None.

USAGE:

MSC-250:	l_track_spd
MSC-850/32:	l_track_spd
MSC-850:	l_track_spd
MSC-800:	l_track_spd

let

SYNTAX:

<i>label</i>	let	variable=operand1 opcode operand2
<i>label</i>	let	variable=function(operand1)*MSC-850/32, MSC-850, MSC-250 ONLY *

PARAMETERS:

variable	Resulting value.
operand1	Variable or constant.
opcode	Operation to be performed.
operand2	Variable or constant.
function	Allowable function type.
	* MSC-850/32, MSC-850, MSC-250 ONLY *

DESCRIPTION:

Performs the indicated arithmetic operation as follows:

opcode	operation	
-----	-----	
+	addition	
-	subtraction	
*	multiplication	
/	division	
&	bitwise and	
	bitwise or	
^	bitwise exclusive or	* MSC-850/32, MSC-850, MSC-250 ONLY *
[]	array indexing	
>>	shift right	
<<	shift left	
}}	rotate right	
{{	rotate left	
function	operation	
-----	-----	
sqr	square root of operand1	* MSC-850/32, MSC-850, MSC-250 ONLY *
abs	absolute value of operand1	* MSC-850/32, MSC-850, MSC-250 ONLY *
neg	negate operand1	* MSC-850/32, MSC-850, MSC-250 ONLY *

The only arithmetic operation allowed, using arrays, is a simple assign. Operations such as add, subtract, multiply etc. are not allowed.

The **let** instruction can be used to transfer data (in 4-byte blocks) from program data memory to the volatile memory of an axis controller. The following example will dimension a 100 element array

called 'test_array' on controller# 3. The value 1234 will be placed in element 19 (array subscripts are zero based) of this array.

EXAMPLE

```
test_array    dim    3,100
              .
              .
              .
              let    test_array[18]=1234
```

The **let** instruction can be used to transfer data (in 4-byte blocks) from the volatile memory of an axis controller to program data memory. The following example will dimension a 100 element array called 'test_array' on controller# 8. The current value of element 3 (array subscripts are zero based) will be placed into variable 'x'. This type of operation can ONLY be used on MSC-850/32, MSC-850 and MSC-250 systems.

EXAMPLE

```
test_array    dim    8,100
              .
              .
              .
              let    x=test_array[2]
```

RETURNS:

None.

USAGE:

```
MSC-250:      let
MSC-850/32:   let
MSC-850:      let
MSC-800:      let
```

let_byte

SYNTAX:

label let_byte destination=source

PARAMETERS:

destination	Variable or cam table element.
source	Variable or cam table element.

DESCRIPTION:

Used to store and retrieve cam elements, one byte at a time. The only arithmetic operation allowed using the **let_byte** instruction is a simple assign.

Can also be used to pack or split 32 bits to 8 bits or 8 bits to 32 bits.

The **let_byte** instruction can be used to transfer data (in 1-byte blocks) from program data memory to the volatile memory of an axis controller. The following example will dimension a 100 element array called 'test_array' on controller# 3. The value 117 will be placed in byte 19 (array subscripts are zero based) of this array. This type of operation can ONLY be used on MSC-850/32, MSC-850 and MSC-250 systems.

EXAMPLE

```
test_array    dim        3,100
              .
              .
              .
              let_byte    test_array[18]=117
```

The **let_byte** instruction can be used to transfer data (in 1-byte blocks) from the volatile memory of an axis controller to program data memory. The following example will dimension a 100 element (400 byte) array called 'test_array' on controller# 8. The current value of byte 3 (array subscripts are zero based) will be placed into variable 'x'. This type of operation can ONLY be used on MSC-850/32, MSC-850 and MSC-250 systems.

EXAMPLE

```
test_array    dim        8,100
              .
              .
              let_byte    x=test_array[2]
```

RETURNS:

None.

USAGE:

MSC-250: let_byte

MSC-850/32: let_byte

MSC-850: let_byte

MSC-800: let_byte

load

SYNTAX:

label load unit,file_name,status

PARAMETERS:

unit	File identifier. Range: 1 to 8
file_name	Label of the 'text' statement containing the file name.
status	Variable containing the result of the operation.

DESCRIPTION:

This instruction loads the program file specified by 'file_name'.

If the specified program is found, it will replace entirely the program currently residing in the MSC.

When the 'load' is completed, program execution begins automatically. If the file does not contain a program, the current program continues execution at the instruction following the **load** instruction. Data may not be passed to the new program using this instruction.

RETURNS:

The return variable will be non-zero if the operation fails. Non-zero status codes are described in Figure 15.3.2.

USAGE:

MSC-250:	load
MSC-850/32:	load
MSC-850:	load
MSC-800:	load

lock

SYNTAX:

label lock controller#,lock#

PARAMETERS:

controller#	controller id#
	Range: MSC-250 1 to 2
	MSC-850/32 1 to 8
	MSC-850 1 to 8
lock #	Lock method (see explanations below)

NOTE:

Lock modes 0, 5, 8 and 9 (described below) may be used in conjunction with the switch cam feature. See the **switch_cam** instruction for additional information and examples on the use of the combined features.

DESCRIPTIONS:

Locks the specified controller onto the master position vector as defined by the master angle configuration.

<u>Lock #</u>	<u>Lock Method</u>
0	cam lock

The MSC multi-axis controllers provide a mode of operation that effectively emulates mechanical cams. In the electronic cam mode, slave axes follow digital cams based on the master angle from one of the master angle buses. Electronic cams are tables of incremental motor moves. Each incremental value occupies one byte of memory in the MSC and has an allowable range of -127 to +127. The slave controller uses the current master position to determine the proper index into the electronic cam array. As the index moves through the cam table, the incremental distances are added together to form the proper slave axis position. Linear interpolation is performed between elements in the cam array.

When lock method 0 is executed, the cam executes at the first element in the array. When the end of the cam table is reached, the process begins again at the beginning of the table. This process is bi-directional, with the cam moving to increasing positive number elements with clockwise master rotation and decreasing number elements with counterclockwise rotation.

Further information about cam lock can be found by reviewing the following instructions; cam_data, set_cam_ptr, get_cam_ptr, switch_cam, get_cam_strt, get_cam_end, get_cam_sum, calc_unit_cam, begin_cam, end_cam.

<u>Lock #</u>	<u>Lock Method</u>
----------------------	---------------------------

1	simple lock with accel/decel limits
----------	--

Lock method 1 provides the electronic equivalent of a gearbox. The electronic gearbox or ratio is driven by the master angle from one of the master angle buses. The master angle is processed by the previously specified ratio instruction and an offset is added, resulting in an effective master used to drive the slave command position.

When lock method 1 is executed, the axis controller calculates the instantaneous offset between the master angle processed by the ratio and the slave command position. Once the offset is calculated, master slave lock is accomplished. Once per millisecond (MSC-850 and MSC-850/32) or once every 488 microseconds (MSC-250) thereafter, the slave axis position is updated based on the new master position. The slave motion is limited by the previously set accel/decel rate (see **set_ac_dc**) and is limited to a speed of 3600 RPM (MSC-850 and MSC-850/32) or 7200 RPM (MSC-250). The limiting of the slave acceleration can be an advantage in systems that have rough or rapidly changing master speeds. The limited acceleration on the slave dampens the slave motor and smooths out the operation.

The ratio instruction can be executed while in lock method 1. When the **ratio** instruction is executed, the slave controller switches to simply slewing at the **set_ac_dc** rate until the slave reaches the new slave speed. When the speed is matched, a new offset is calculated and lock is resumed. It is important to note that after a **ratio** instruction is executed, a new offset is used. The axis status flag JOGGING is on while the slave motor is changing from one speed to another. This flag goes off when the ratio lock equation starts executing. The acceleration rate may be changed during the slew from one speed to another caused by a ratio change. A **set_ac_dc** executed prior to the **ratio** instruction will cause the slew acceleration rate to change.

The result of executing a **lock** command may vary, depending on the value used in the **ratio** command and the command sequence. This can best be illustrated using the following examples:

EXAMPLE 1

label	ratio	slave,ratio_value
	.	
	.	
	.	
	lock	slave,1

At the moment the **lock** is executed:

- a) the slave axis will calculate an internal master/slave "offset", by comparing the master angle with the slave position
- b) the slave axis will accelerate at the rate set using the last **set_ac_dc** command
- c) the slave axis will "make up" the distance lost during the acceleration time, by running at a faster rate for that same period of time (the slave axis would appear to momentarily "overshoot" the requested ratio)

- d) the slave axis will run at the requested ratio while maintaining the internal master/slave "offset"

Subsequent changes in 'ratio_value' will not result in the slave axis "making up" any lost distance while adjusting to a new ratio.

EXAMPLE 2

In this example, the ratio command is executed twice. The result of locking the slave axis will be different than that of EXAMPLE 1.

```
label      ratio      slave,0
          .
          .
          lock      slave,1
          .
          .
          ratio      slave,ratio_value
```

At the moment the **lock** is executed:

- a) the slave axis will calculate an internal master/slave "offset", but since the ratio is initially set to zero, no motion occurs

At the moment the second **ratio** instruction is executed:

- a) the slave axis will accelerate at the rate set using the last **set_ac_dc** command
 b) the slave axis will run to the requested ratio while maintaining the internal master/slave "offset" (the slave axis will NOT attempt to "make up" any lost distance while adjusting to the new ratio)

Lock # Lock Method

2 **velocity lock**

In velocity lock, the slave axis tracks the master velocity with accel/decel limit.

3 **piecewise profile lock**

In piecewise profile lock, the slave axis will execute the piecewise profile in memory when the master angle (modulo 4096) crosses the specified angle from either direction (CW or CCW). This master angle is specified by the instruction **set_trig_pw**. The instruction **prep_profile** must be executed before the **lock** instruction.

4 **simple lock without accel/decel limits**

Lock method 4 is identical to lock method 1. The only difference is that the slave axis is not limited by the specified slave acceleration rate. This means that during lock, the slave is

commanded directly by the effective master position. Note: Any perturbations or roughness in the master will be passed onto the slave with no accel/decel rate limiting.

The ratio instruction can be executed during lock method 4. When a new ratio is executed, the slave controller breaks lock, slews to new lock speed at the specified acceleration rate and relocks using the above equation. The JOGGING status flag is on during the slew period. Note that after the slew period a new offset is calculated. The acceleration rate may be changed during slew executing a set_ac_dc instruction prior to the ratio instruction.

<u>Lock #</u>	<u>Lock Method</u>
---------------	--------------------

5	cam lock
----------	-----------------

Lock method 5 is identical to lock method 0 except that the user has the option of positioning the cam pointer to a position other than the beginning of the cam. This is accomplished by executing the set_cam_ptr instruction. Execution of the cam will begin at the position in the cam data table at the cam pointer.

6	keyway lock
----------	--------------------

Lock method 6 allows the user to align the absolute position of the slave with the absolute position of the master. The ratio in Lock Method 6 is fixed at 1:1 and may not be changed. The slave controller acceleration rate is limited by the previously executed set_ac_dc instruction. The slave top speed is also limited to 3600 RPM.

When the lock instruction is executed, the slave controller executes the following equation every 1 millisecond:

$$\text{master angle} = \text{slave command position}$$

This equation is executed modulo 1 turn (4096 bits) of the master. This means that when the lock instruction is executed, the slave will move, at the acceleration rate specified, in the shortest direction to bring the master and slave absolute angles into alignment. Note that Lock Method 6 can cause movement when lock is executed even if the master is at rest.

The slave accel and speed are limited. Hunting can occur if the master rate of change is greater than the specified slave acceleration rate. The rule of thumb of having the slave acceleration rate at least 5 times the expected master rate of change is applicable. The limited acceleration rate can be used to smooth a master with fast perturbations (roughness).

7	undefined
----------	------------------

8	lock cam to master angle
----------	---------------------------------

In lock type 8, the slave axis will begin executing its cam when the master axis crosses the specified angle (modulo 4096) from either direction (CW or CCW). This angle is specified using the instruction set_trig_cam. It is possible to start execution of the cam from any point in the cam

by using the instruction set_cam_ptr. The axis status flag SWITCH CAM PENDING will be set until the master crosses the specified lock angle and execution of the cam begins.

9 lock cam to master angle for 1 cycle only

Lock type 9 is identical to lock type 8 except that execution of the cam will automatically terminate at the last element in the cam.

<u>Lock #</u>	<u>Lock Method</u>
----------------------	---------------------------

10	velocity cam lock
-----------	--------------------------

In lock method 10, the output is no longer based on distance but varies with speed changes in the cam table, based on a constant calculated at the time the lock command is issued. Lock method 10 must be executed after lock method 0 or 5. The position output voltage will reflect the velocity of the cam rather than the position of the cam. Following error checking and software digital compensation are disabled.

RETURNS:

None.

USAGE:

MSC-250:	lock	all lock types available
MSC-850/32:	lock	all lock types available
MSC-850:	lock	all lock types available
MSC-800:	lock	lock types 0,1,2,3,4,5

master**SYNTAX:**

label master controller#

PARAMETERS:

controller# controller id #
Range: MSC-800 1 to 8

DESCRIPTION:

Resets all motor fault conditions and puts the specified axis controller into a passive position sensing mode. Turns the servo amplifier off and disables the following error check.

Sets the analog position output to be proportional to the feedback transducer position as follows:

RETURNS:

For an MSC-850/ACE-850 controller, the above table will not be accurate unless an appropriate find marker instruction had been done.

RETURNS:

None.

USAGE:

MSC-250: drive_off
MSC-850/32: drive_off
MSC-850: drive_off
MSC-800: master

msc_type**SYNTAX:**

label msc_type system_type

PARAMETERS:

system_type The type of MSC system type being used.
Range: 800, 850, 250, 850/32

DESCRIPTION:

This statement is a MacroPro Compiler directive.

It is used by the Compiler to identify the instruction set that is allowed for use with the designated system unit. This will help avert the situation where a programmer attempts to use an instruction that is not supported by the system unit.

The Compiler will assume the **msc_type** is 800 if this instruction is not included in the program.

RETURNS:

None.

USAGE:

MSC-250: msc_type
MSC-850/32: msc_type
MSC-850: msc_type
MSC-800: msc_type

no_op**SYNTAX:**

label no_op

PARAMETERS:

None.

DESCRIPTION:

Performs a 'no operation' instruction. Commonly used to provide a short time delay.

RETURNS:

None.

USAGE:

MSC-250: no_op

MSC-850/32: no_op

MSC-850: no_op

MSC-800: no_op

offset_master

SYNTAX:

label offset_master controller id#,offset

PARAMETERS:

controller#	controller id# Range: MSC-250 1 to 3 MSC-850/32 1 to 8
offset	Offset to be added to the master position. This value is a signed 32-bit number.

DESCRIPTION:

This command instructs the operating system to add the specified offset value (in bits) to the actual transducer angle of the specified master controller before it is placed on the master angle bus. The offset is an absolute value added to the master angle and is unaffected. Only the data being placed on the master angle bus is affected.

RETURNS:

None.

USAGE:

MSC-250:	offset_master
MSC-850/32:	offset_master
MSC-850:	N/A
MSC-800:	N/A

open

SYNTAX:

label open unit,file_name,status

PARAMETERS:

unit	File identifier. Range: 1 to 8
file_name	Label of the text statement containing the file name.
status	Variable containing the result of the operation.

DESCRIPTION:

This instruction attempts to open the specified file for reading. The MSC associates the file name with the unit identifier and will set the file record pointer to the beginning of the file. Up to 8 files may be open at one time.

RETURNS:

The return variable will be zero if the operation is successful. Non-zero status codes are described in Section 15.3.2.

USAGE:

MSC-250:	open
MSC-850/32:	open
MSC-850:	open
MSC-800:	open

over_draw**SYNTAX:**

label over_draw controller#,speed,limit,distance

PARAMETERS:

controller#	controller id# Range: MSC-250 1 to 2 MSC-850/32 1 to 8 MSC-850 1 to 8
speed	Desired speed to start overdraw in RPM. Must not exceed speed of associated index or position instruction.
limit	Desired limit in bits. Represents maximum travel allowed if hardware interrupt does NOT occur.
distance	Distance to travel AFTER the hardware interrupt occurs.

DESCRIPTION:

This instruction is typically used in feed to sensor types of applications. Information provided by the **over_draw** instruction modifies an **index** or **position** instruction as shown in Figure 18.1. As the **index** or **position** instruction nears completion, motor speed will begin to decrease according to the set accel/decel rate. When the motor speed reaches the value specified in the **over_draw** instruction, deceleration will cease. The motor will run at constant speed until one of two conditions are met:

1. The distance specified in 'limit' is reached. Should this occur, the motor will decelerate to a stop.
2. The Controller's associated input module turns on. Should this occur, the position of the motor shaft is immediately noted. The motor will stop 'distance' bits from the point the input module turns on.

It should be noted that the motor shaft position is trapped immediately by the controller's hardware. However, it may be as long as 1 millisecond before the trapped position is serviced. For this reason, the 'distance' variable should allow at least enough distance for deceleration **plus** the distance the motor shaft would turn in 1 millisecond at the 'speed' specified in the **over_draw** instruction.

NOTE: This instruction can only be used in conjunction with hardware interrupts.

EXAMPLE:

```

    busy_1      equ      94      busy flag
    hwi_armed_1 equ      91      interrupt armed flag
    .
    .
    set_speed   1,400      400 rpm
    set_ac_dc    1,100      100 rev/sec2
    .
    .
    enable_hwi
    over_draw    1,100,4096,2000

```

! wait for index to finish. If interrupt still armed, sensor not tripped

```

loop
    if_stat_on  1,busy_1,loop
    if_stat_on  1,hwi_armed_1,od_fail
    .
    (normal processing)
    .
od_fail
    (handle case where sensor not seen)

```

RETURNS:

None.

USAGE:

```

MSC-250:      over_draw
MSC-850/32:   over_draw
MSC-850:      over_draw
MSC-800:      N/A

```


port_set

SYNTAX:

label port_set port#,baud,protocol

PARAMETERS:

port#	Port number on the MSC. Range: MSC-250 2, 3 MSC-850/32 0I,0R,0,1,2,3 MSC-850 0, 2 MSC-800 0, 2
baud	Data transmission rates are listed below.
protocol	Communications characteristics are listed below.

DESCRIPTION:

Open and initialize the specified communications port.

The **port_set** instruction may be executed at any time.

port# - 0 is the active current loop port for the MSC-800, MSC-850 and MSC-850/32.

1 is the executive port for all MSC controllers.

2 is the passive current loop port for all MSC controllers.

3 is an RS-232C port on the MSC-250 and MSC-850/32.

0I is the active/passive current loop port for the MSC-850/32.

0R is an RS-232C port on the MSC-850/32.

baud rate - use any one of the following:
110, 300, 600, 1200, 2400, 4800, 9600 for all MSC controllers.
19200, 38400 with ports 1 or 0R on the MSC-850/32.

protocol - specify the communications characteristics as follows:

protocol	Description

0	1 stop bit, no parity, XON/XOFF disabled
1	2 stop bits, no parity, XON/XOFF disabled
2	1 stop bit, even parity, XON/XOFF disabled
3	2 stop bits, even parity, XON/XOFF disabled
4	1 stop bit, odd parity, XON/XOFF disabled
5	2 stop bits, odd parity, XON/XOFF disabled
6	RESERVED
7	RESERVED
8	1 stop bit, no parity, XON/XOFF enabled
9	2 stop bits, no parity, XON/XOFF enabled
10	1 stop bit, even parity, XON/XOFF enabled

11	2 stop bits, even parity, XON/XOFF enabled
12	1 stop bit, odd parity, XON/XOFF enabled
13	2 stop bits, odd parity, XON/XOFF enabled

If no parity, then data word is 8 bits. If even or odd parity, then data word is 7 bits.

RETURNS:

None.

USAGE:

MSC-250: port_set
MSC-850/32: port_set
MSC-850: port_set
MSC-800: port_set

position

SYNTAX:

label position controller#,abs_position

PARAMETERS:

controller#	controller id#
	Range: MSC-250 1 to 3
	MSC-850/32 1 to 8
	MSC-850 1 to 8
	MSC-800 1 to 9
abs_position	The absolute position relative to position zero.
	Range: MSC-250 (-524287*4096) to (+524287*4096) -1 bits
	MSC-850/32 (-2048*4096) to (+2048*4096) -1 bits
	MSC-850 (-2048*4096) to (+2048*4096) -1 bits
	MSC-800 (-2048*4096) to (+2048*4096) -1 bits

DESCRIPTION:

Move the controller to the specified position, relative to zero. The speed of the move is determined by the previously set accel/decel rate and speed.

The MSC has a resolution of 4096 bits per turn. To position to turn 1, 'abs_position' would be 4096.

RETURNS:

None.

USAGE:

MSC-250:	position
MSC-850/32:	position
MSC-850:	position
MSC-800:	position

prep_profile

SYNTAX:

label *prep_profile* controller#,data_label

PARAMETERS:

controller#	controller id#
	Range: MSC-250 1 to 3
	MSC-850/32 1 to 8
	MSC-850 1 to 8
	MSC-800 1 to 9
data_label	Label of the data for the profile.

DESCRIPTION:

Transmits the desired Piecewise profile data to the specified controller and prepares the data for execution. While the motion controller is performing its' calculations, the **CALCULATING PIECEWISE PROFILE** controller status flag will be on. When profile calculations or execution have been completed, this controller status flag will be turned off. After calculations are completed, the **get_pstat** instruction may be used to determine whether calculations completed successfully.

RETURNS:

None.

USAGE:

MSC-250:	prep_profile
MSC-850/32:	prep_profile
MSC-850:	prep_profile
MSC-800:	profile

preset**SYNTAX:**

label preset controller#,variable

PARAMETERS:

controller#	controller id#	
	Range: MSC-250	Always 3
	MSC-850/32	1 to 8
	MSC-850	1 to 8
variable	offset value	

DESCRIPTION:

Adds the offset value in bits to the master angle used to drive the PLS.

RETURNS:

None.

USAGE:

MSC-250:	preset
MSC-850/32:	preset
MSC-850:	preset
MSC-800:	N/A

preset (MSC-850/HPL-850)**SYNTAX:**

label preset controller#,variable

PARAMETERS:

controller#	controller id# Range: MSC-850/32 1 to 8 MSC-850 1 to 8
variable	The desired HPL accumulator value.

DESCRIPTION:

Sets the HPL-850 accumulator to the specified count value. This instruction allows the user to set the HPL-850 angle to a desired value regardless of the position of the Master Angle Bus driving the HPL-850.

RETURNS:

None.

USAGE:

MSC-250:	N/A
MSC-850/32:	preset
MSC-850:	preset
MSC-800:	N/A

print

SYNTAX:

label **print** text_label

PARAMETERS:

text_label Label of a text string.

DESCRIPTION:

Print the ASCII character string to the port declared in the previous **port_set** instruction.

The execution of a **port_set** instruction must precede the execution of the **print** instruction.

The output to the port is buffered, that is, the speed characteristics of the port do not affect the execution of the MSC.

RETURNS:

None.

USAGE:

MSC-250: **print**
MSC-850/32: **print**
MSC-850: **print**
MSC-800: **print**

print_num

SYNTAX:

label **print_num** length,decimals,value

PARAMETERS:

length	Number of character places of 'value' parameter.
decimals	Number of decimal places of 'value' parameter.
value	Numeric value to be converted to ASCII characters and displayed according to the length and decimal parameters.

DESCRIPTION:

Print the output value to the port declared in the previous **port_set** instruction.

The format of the printed number is defined by 'length' which declares the maximum number of character positions allowed to represent the numeric value and by 'decimals' which declares the number of numeric positions to the right of the decimal point.

If the specified length is less than the actual length of the value to be printed, a string of asterisks (*) will be printed instead. If the specified length is greater than the actual length of the value to be printed, the number will be right justified accordingly.

The execution of a **port_set** instruction must precede the execution of the **print_num** instruction.

The output to the port is buffered, that is the speed characteristics of the port do not affect the execution of the MSC.

RETURNS:

None.

USAGE:

MSC-250:	print_num
MSC-850/32:	print_num
MSC-850:	print_num
MSC-800:	print_num

rand_int

SYNTAX:

label rand_int max_number,variable

PARAMETERS:

max_number	Maximum allowed number returned. Range: 1 to 65535
variable	Variable where result is returned.

DESCRIPTION

Generates a random number in the range from 0 to the maximum number specified.

RETURNS:

The random number is returned in 'variable'.

USAGE:

MSC-250:	rand_int
MSC-850/32:	rand_int
MSC-850:	rand_int
MSC-800:	rand_int

ratio

SYNTAX:

label ratio controller#,ratio

PARAMETERS:

controller#	controller id#
	Range: MSC-250 1 to 2
	MSC-850/32 1 to 8
	MSC-850 1 to 8
	MSC-800 1 to 8
ratio	Desired electronic ratio (in bits).
	Range: -32768 to +32767 (-8.000 to +7.999)

DESCRIPTION:

Scale the master/slave position vector being passed by the 'ratio' value for the specified controller.

Combined with the **lock** instruction (simple lock method), an electronic 'gear box' with a variable ratio may be emulated.

'ratio' is in bits (i.e. 4096 would give a ratio of 1:1).

RETURNS:

None.

USAGE:

MSC-250:	ratio
MSC-850/32:	ratio
MSC-850:	ratio
MSC-800:	ratio

read

SYNTAX:

label read unit,data_area,length,status

PARAMETERS:

unit	File identifier. Range: 1 to 8
data_area	Label of the begin_data , begin_cam , dim or variable name.
length	Number of bytes (characters) to be read.
status	Variable containing the result of the operation.

DESCRIPTION:

This instruction reads data from the given 'data_area' to the specified unit (file). The number of bytes (characters) to be read is given as 'length'.

RETURNS:

The return variable will be zero if the operation is successful. Non-zero status codes are described in Section 15.3.2.

USAGE:

MSC-250:	read
MSC-850/32:	read
MSC-850:	read
MSC-800:	read

read_offset

SYNTAX:

label read_offset controller#,variable

PARAMETERS:

controller#	controller id#
	Range: MSC-250 1 to 2
	MSC-850/32 1 to 8
	MSC-850 1 to 8
	MSC-800 1 to 8
variable	Offset value in bits.

DESCRIPTION:

Returns the current offset as set by the **set_offset** instruction.

RETURNS:

The current offset is returned in the designated 'variable'.

USAGE:

MSC-250:	read_offset
MSC-850/32:	read_offset
MSC-850:	read_offset
MSC-800:	read_offset

restart_at**SYNTAX:**

label restart_at address_label

PARAMETERS:

address_label Branch address.

DESCRIPTION:

Clear all **gosub** return addresses and branch to the specified 'address label'.

RETURNS:

None.

USAGE:

MSC-250: restart_at
MSC-850/32: restart_at
MSC-850: restart_at
MSC-800: restart_at

return_sub

SYNTAX:

label return_sub

PARAMETERS:

None.

DESCRIPTION:

Return from a subroutine invoked by a **gosub** instruction.

The MSC provides 20 levels of subroutine nesting. The program status **STACK OVERFLOW** will occur if more than 20 levels are used.

If a **return_sub** instruction is encountered without a corresponding **gosub**, a program status of **STACK UNDERFLOW** may occur.

RETURNS:

None.

USAGE:

MSC-250: return_sub
MSC-850/32: return_sub
MSC-850: return_sub
MSC-800: return_sub

save**SYNTAX:**

label save unit,file_name,status

PARAMETERS:

unit	File identifier. Range: 1 to 8.
file_name	Label of the text statement containing the file name.
status	Variable containing the result of the operation.

DESCRIPTION:

This instruction saves the program and all data currently residing in the MSC as a file with the name contained in 'file_name'.

RETURNS:

The return variable will be zero if the operation is successful. Non-zero status codes are described in Section 15.3.2.

USAGE:

MSC-250:	save
MSC-850/32:	save
MSC-850:	save
MSC-800:	save

select

SYNTAX:

<i>label</i>	select	variable
--------------	--------	----------

PARAMETERS

variable	Variable to be tested.
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
10	10
11	11
12	12
13	13
14	14
15	15
16	16
17	17
18	18
19	19
20	20
21	21
22	22
23	23
24	24
25	25
26	26
27	27
28	28
29	29
30	30
31	31
32	32
33	33
34	34
35	35
36	36
37	37
38	38
39	39
40	40
41	41
42	42
43	43
44	44
45	45
46	46
47	47
48	48
49	49
50	50
51	51
52	52
53	53
54	54
55	55
56	56
57	57
58	58
59	59
60	60
61	61
62	62
63	63
64	64
65	65
66	66
67	67
68	68
69	69
70	70
71	71
72	72
73	73
74	74
75	75
76	76
77	77
78	78
79	79
80	80
81	81
82	82
83	83
84	84
85	85
86	86
87	87
88	88
89	89
90	90
91	91
92	92
93	93
94	94
95	95
96	96
97	97
98	98
99	99
100	100

DESCRIPTION:

Change program flow based on the value of 'variable'. When executed, a program branch occurs to the **case** statement which matches the specified 'variable'.

If there is no match, program flow resumes at the instruction following the **default** instruction. The **exit_select** instruction causes a branch to the instruction following the **end_select** instruction.

EXAMPLE

```

select          num
      case      1
      .
      .
      exit_select

      case      2
      .
      .
      exit_select

      default
      .
      .
      exit_select
end select

```

RETURNS:

None.

USAGE:

MSC-250:	select
MSC-850/32:	select
MSC-850:	select
MSC-800:	select

set_ac_dc**SYNTAX:**

label set_ac_dc controller#,rate

PARAMETERS:

controller#	controller id#
	Range: MSC-250 1 to 3
	MSC-850/32 1 to 8
	MSC-850 1 to 8
	MSC-800 1 to 9
rate	Desired accel/decel rate in revs/sec/sec.
	Range: MSC-250 2 to 1600
	MSC-850/32 2 to 800
	MSC-850 2 to 800
	MSC-800 2 to 800

DESCRIPTION:

Set the accel/decel rate for the specified controller. All motions for this controller (except cam, ratios and piecewise profiles) will be determined by this rate.

The rate is scaled in revs/sec/sec for all motions (except **vel_cw** and **vel_ccw** instructions) for which the rate scale is (revs/sec/sec) / 256.

The default accel/decel rate is 2 revs/sec/sec. This is also the lowest allowable accel/decel rate. The highest allowable rate is 800 revs/sec/sec.

RETURNS:

None.

USAGE:

MSC-250:	set_ac_dc
MSC-850/32:	set_ac_dc
MSC-850:	set_ac_dc
MSC-800:	set_ac_dc

set_acy_cnt

SYNTAX:

label set_acy_cnt controller#,count

PARAMETERS:

controller#	controller id# Range: MSC-850/32 1 to 8 MSC-850 1 to 8
count	Desired encoder count in bits per revolution. Range: 2048, 4096, 8192 or 16384 counts only.

DESCRIPTION:

Defines the number of bits generated per motor revolution. This number will be the result of multiplying the encoder line count by 4.

NOTE:

This instruction is ONLY valid when used with the ACY-850 axis controller.

RETURNS:

None.

USAGE:

MSC-250:	N/A
MSC-850/32:	set_acy_cnt
MSC-850:	set_acy_cnt
MSC-800:	N/A

set_bit

SYNTAX:

label set_bit bit#,variable

PARAMETERS:

bit#	Number of the bit within the 4-byte variable to be set to on (logic 1). Range: 0 to 31
variable	The 4-byte area in memory affected by this instruction.

DESCRIPTION:

The specified bit will be set to on (logic 1). If that bit has been previously set, it will remain set. If that bit was previously cleared (logic 0), it will now be set to on.

This instruction has no effect on the remaining bits of this 4-byte variable.

This instruction will be ignored if 'bit#' is outside the range of 0 to 31.

RETURNS:

None.

USAGE:

MSC-250:	set_bit
MSC-850/32:	set_bit
MSC-850:	set_bit
MSC-800:	N/A

set_cam_ptr

SYNTAX:

label set_cam_ptr controller#,value

PARAMETERS:

controller#	controller id#
	Range: MSC-250 1 to 2
	MSC-850/32 1 to 8
	MSC-850 1 to 8
	MSC-800 1 to 8
value	The new start position within the cam table.
	Range: 0 to (cam table length - 1)

DESCRIPTION:

Repositions the 'start of cam' pointer in the current cam table array for the selected controller. This instruction must be used in conjunction with **lock** types 5, 8 and 9 in order to execute the cam from within the table, rather than the start of the table.

RETURNS:

None.

USAGE:

MSC-250:	set_cam_ptr
MSC-850/32:	set_cam_ptr
MSC-850:	set_cam_ptr
MSC-800:	cam_pointer

set_com_limit**SYNTAX:**

label set_com_limit controller#,variable

PARAMETERS:

controller#	controller id#	
	Range: MSC-250	1 to 2
variable		

DESCRIPTION:

Limits the error between commanded and actual position by disabling the command generator.

This instruction is canceled when the move is complete or by executing f_decel command.

RETURNS:

None.

USAGE:

MSC-250: set_com_limit

set_flag**SYNTAX:**

label set_flag user_flag#

PARAMETERS:

user_flag#	Number of the user flag. Range: 208 to 255
------------	---

DESCRIPTION:

Sets the specified user flag.

If using a pseudo axis with an MSC-800 controller, flags 208-224 are used as axis status flags and should not be used as user flags.

RETURNS:

None.

USAGE:

MSC-250:	set_flag
MSC-850/32:	set_flag
MSC-850:	set_flag
MSC-800:	set_flag

set_gl_ccw

SYNTAX:

label set_gl_ccw controller#

PARAMETERS:

controller#	controller id#
	Range: MSC-250 1 to 3
	MSC-850/32 1 to 8
	MSC-850 1 to 8
	MSC-800 1 to 9

DESCRIPTION:

Set the absolute 0.0 position to the nearest position transducer zero in the counter-clockwise direction. The current local 0.0 will be cleared. NOTE: For an ACE-850 controller or an MSC-250 controller, a find marker instruction must be done before the **set_gl_ccw** instruction in order to produce meaningful results.

This instruction may be executed while the motor is in motion. If it is performed while the motor is stopped, no motion will occur.

This instruction is often used in conjunction with a zero revolution switch to establish an absolute zero or home position on multi-turn systems.

RETURNS:

None.

USAGE:

MSC-250:	set_gl_ccw
MSC-850/32:	set_gl_ccw
MSC-850:	set_gl_ccw
MSC-800:	set_gl_ccw

set_gl_cw

SYNTAX:

label set_gl_cw controller#

PARAMETERS:

controller#	controller id#
	Range: MSC-250 1 to 3
	MSC-850/32 1 to 8
	MSC-850 1 to 8
	MSC-800 1 to 9

DESCRIPTION:

Set the absolute 0.0 position to the nearest position transducer zero in the clockwise direction. The current local 0.0 will be cleared. NOTE: For an ACE-850 controller, a find marker instruction must be done before the **set_gl_cw** instruction in order to produce meaningful results.

This instruction may be executed while the motor is in motion. If it is performed while the motor is stopped, no motion will occur.

This instruction is often used in conjunction with a zero revolution switch to establish an absolute zero or home position on multi-turn systems.

RETURNS:

None.

USAGE:

MSC-250:	set_gl_cw
MSC-850/32:	set_gl_cw
MSC-850:	set_gl_cw
MSC-800:	set_gl_cw

set_gl_ccw (HPL-850)**SYNTAX:**

label set_gl_ccw controller#

PARAMETERS:

controller#	controller ID#
	Range: MSC-850/32 1 to 8
	MSC-850 1 to 8

DESCRIPTION:

Sets the 0.0 degree reference to the nearest CCW Master Angle Bus 0.0.

RETURNS:

None.

USAGE:

MSC-250:	set_gl_ccw
MSC-850/32:	set_gl_ccw
MSC-850:	set_gl_ccw
MSC-800:	N/A

set_gl_cw (HPL-850)**SYNTAX:**

label set_gl_cw controller#

PARAMETERS:

controller#	controller ID#
	Range: MSC-850/32 1 to 8
	MSC-850 1 to 8

DESCRIPTION:

Sets the 0.0 degree reference to the nearest CW Master Angle Bus 0.0.

RETURNS:

None.

USAGE:

MSC-250:	set_gl_cw
MSC-850/32:	set_gl_cw
MSC-850:	set_gl_cw
MSC-800:	N/A

set_hi_scan**SYNTAX:**

label set_hi_scan

PARAMETERS:

None.

DESCRIPTION:

Sets the I/O expander scan rate to once per 1.2 ms. This instruction may be used in systems with IOE-850 type expanders only. Default scan rates are as follows:

- 1 expander - once every 12msec
- 2 expanders - once every 24msec
- 3 expanders - once every 36msec
- 4 expanders - once every 48msec

RETURNS:

None.

USAGE:

MSC-250: N/A
MSC-850/32: set_hi_scan
MSC-850: N/A
MSC-800: N/A

set_home**SYNTAX:**

label set_home controller#, offset

PARAMETERS:

controller#	controller id#
	Range: MSC-850/32 1 to 8
	MSC-850 1 to 8
offset	Position offset value.
	Range: -4096 to +4096

DESCRIPTION:

Establishes a position offset for the specified controller. This instruction should be used when defining a 'home' or 0.0 reference location.

The following steps illustrate how to clear the encoder 'turns' counter:

- 1) Remove power to the motor/drive system.
- 2) Remove the encoder cable from the drive to be zeroed.
- 3) Position the device to its physical 'home' location.
- 4) Drain the capacitor on the encoder unit for approximately 5 minutes.
- 5) Reconnect the encoder cable.
- 6) Enable power to the motor/drive system.

The following program example illustrates how to establish the 'offset' value and initial use of this value. This procedure will typically be done only one time, until a new 'home' reference location is needed.

EXAMPLE

```
label  drive_on    acy_controller
      get_com      acy_controller,position
      let          offset=0-position
      set_home      acy_controller,offset
```

On subsequent power down/up sequences, it will be necessary to set the offset for the appropriate ACY-850 controller. It is important that the 'offset' value remain the same until a new 'home' reference location is needed. Keep in mind that the value 'offset' is retained in NVRAM.

EXAMPLE

```
label  drive_on    acy_controller
      set_home      acy_controller,offset
```

NOTE:

This instruction is ONLY valid when used with the ACY-850 axis controller.

RETURNS:

None.

USAGE:

MSC-250:	N/A
MSC-850/32:	set_home
MSC-850:	set_home
MSC-800:	N/A

set_local

SYNTAX:

label set_local controller#

PARAMETERS:

controller#	controller id#
	Range: MSC-250 1 to 3
	MSC-850/32 1 to 8
	MSC-850 1 to 8
	MSC-800 1 to 9

DESCRIPTION:

Set the present motor position as the absolute 0.0 position. Will override the **set_gl_ccw** and **set_gl_cw** instructions, however, the global 0.0 position is not changed.

Used to establish a 'floating' home position.

RETURNS:

None.

USAGE:

MSC-250:	set_local
MSC-850/32:	set_local
MSC-850:	set_local
MSC-800:	set_local

set_local (HPL-850)**SYNTAX:**

label set_local controller#

PARAMETERS:

controller# Slot number of the HPL-850 Controller.
Range: 1 to 8

DESCRIPTION:

Causes the HPL-850 Controller accumulator to be set to zero. The current Master Angle Data reading becomes the zero degree position.

RETURNS:

None.

USAGE:

MSC-250: N/A
MSC-850/32: set_local
MSC-850: set_local
MSC-800: N/A

set_map

SYNTAX:

label set_map variable

PARAMETERS:

variable The desired map value.

DESCRIPTION:

'map' is an acronym for Master Angle Passing.

The **set_map** instruction defines the master angle configuration within a system unit. The proper value of 'variable' can be determined from Figure 3.29 and Figure 3.30.

Only 1 transmitter per bus can be defined. Improper bus configurations will cause the **get_map_stat** instruction to return a non-zero value.

RETURNS:

None.

USAGE:

MSC-250: set_map
MSC-850/32: set_map
MSC-850: set_map
MSC-800: N/A

set_mcf (MCF-850)**SYNTAX:**

label set_mcf controller#,variable

PARAMETERS:

controller#	controller id#
	Range: MSC-850/32 1 to 8
	MSC-850 1 to 8

variable Multi-function controller configuration value.

DESCRIPTION:

This instruction is used to configure the MCF-850 controller which is used with the MSC-850 and MSC-850/32 systems. This card controls the Pseudo Axis, Master Angle Bus Network, Fiber Optic Network and Programmable Limit Switch (PLS) functions.

RETURNS:

None.

USAGE:

MSC-250:	set_mcf
MSC-850/32:	set_mcf
MSC-850:	set_mcf
MSC-800:	N/A

set_mcf (ACR-850/ACE-850/ACY-850)**SYNTAX:**

label **set_mcf** controller#,variable

PARAMETERS:

controller#	controller id#
	Range: MSC-850/32 1 to 8
	MSC-850 1 to 8
variable	A value of 1 will enable the "analog mode". A value of 0 will disable the "analog mode".

DESCRIPTION:

The **set_mcf** instruction can be used with the ACR-850, ACE-850 and ACY-850 controller cards in order to implement an open loop mode of operation known as "analog mode".

In this mode of operation, the drive unit will be enabled by an external input source. A **drive_off** instruction followed by a **set_mcf** instruction to the ACE-850, ACR-850 or ACY-850 will put that axis controller into "analog mode". Once in "analog mode", the controller will be in an open loop mode where the analog output to the drive is not driven by the "position loop", but rather is controlled in the Macroprogram using the **analog_out** instruction.

When used with the ACR-850, ACE-850 or ACY-850 cards, the **analog_out** instruction will now function in the same manner as when it is used with the ACM-850 card. A voltage in the range of -10V to +10V, based on an **analog_out** value ranging from -2048 to +2047, will be generated by the ACE-850, ACR-850 or ACY-850 controller cards.

A subsequent **drive_on** instruction will put the controller back into the normal "position loop mode" of operation.

The following program segment shows how a program might select the "analog mode" of operation, then revert back to the normal "position loop mode" of operation.

! ----- This section enables "analog mode" on the WINDER axis controller -----
!

enable_t_modeturn_on	WINDER_ENABLE	! enable the drive with a discreet output
drive_off	WINDER	! disable "position mode"
set_mcf	WINDER,1	! enable "analog mode"

! ----- This section generates an output voltage at the WINDER axis controller -----
!

a_mode	analog_out	WINDER,1,volts	! ranges from -2048 to +2047 (-10V to +10V)
--------	------------	----------------	--

```
if_io_on    ANALOG_MODE,a_mode    ! stay in "analog mode" while
                                           input activated

! ----- This section puts the output voltage at the WINDER back to zero -----
!
analog_out    WINDER,1,0                ! output voltage back to zero
set_tmr72,100                ! short delay to allow the
                               output to get to zero
wait          if_tmr_on    72,wait

! ----- This section enables normal "position mode" -----
!
enable_p_mode    drive_on    WINDER                ! enables "position mode"
set_speed        WINDER,speed                ! set WINDER speed
set_ac_dc        WINDER,acdc                ! set WINDER accel/decel
                                           rate
index          WINDER,distance                ! index the WINDER
```

RETURNS:

None.

USAGE:

MSC-250: set_mcf
MSC-850/32: set_mcf
MSC-850: set_mcf
MSC-800: N/A

set_mcf (MSC-250 controllers 1 & 2)**SYNTAX:**

label **set_mcf** controller#,variable

PARAMETERS:

controller#	controller id# Range: MSC-250 1 or 2
variable	A value of 1 will enable the "analog mode". A value of 0 will disable the "analog mode".

DESCRIPTION:

The **set_mcf** instruction can be used with axis controllers #1 and #2 on the MSC-250 in order to implement an open loop mode of operation known as "analog mode".

In this mode of operation, the drive unit will be enabled by an external input source. A **drive_off** instruction followed by a **set_mcf** instruction to either controller #1 or #2 will put that controller into "analog mode". Once in "analog mode", the controller will be in an open loop mode where the analog output to the drive is not driven by the position loop, but rather is controlled in the Macroprogram using the **analog_out** instruction.

When used with controllers #1 or #2, the **analog_out** instruction will now function in the same manner as when it is used with axis controller #4 of the MSC-250. A voltage in the range of -10V to +10V, based on an analog_out value ranging from -2048 to +2047, will be generated by the axis controller.

A subsequent **set_mcf** instruction, using a value of 0 for the configuration value, will put the controller back into the normal "position loop mode" of operation.

The following program demonstrates how a program might select the "analog mode" of operation as well as reverting back to the normal "position loop mode" of operation.

```

!
! ----- This section enables "analog mode" on the WINDER axis controller -----
!
enable_t_modeturn_on            WINDER_ENABLE            ! enable the drive with a
                                                                    discreet output
                                                                    drive_off        WINDER            ! disable "position mode"
                                                                    set_mcf        WINDER,1            ! enable "analog mode"

!
! ----- This section generates an output voltage at the WINDER axis controller -----
!

```

a_mode	analog_out	WINDER,1,volts	! volts ranges from -2048 to +2047 (-10V to +10V)
	if_io_on	ANALOG_MODE,a_mode	! stay in "analog mode" while input activated

! ----- This section puts the output voltage at the WINDER back to zero -----

!

	analog_out	WINDER,1,0	! output voltage back to zero
	set_tmr	72,100	! short delay to allow the output to get to zero
wait	if_tmr_on	72,wait	

!

! ----- This section enables normal "position mode" -----

!

enable_p_mode	set_mcf	WINDER,0	! disables "analog mode"
	drive_on	WINDER	! enables "position mode"
	set_speed	WINDER,speed	! set WINDER speed
	set_ac_dc	WINDER,acdc	! set WINDER accel/decel rate
	index	WINDER,distance	! index the WINDER

RETURNS:

None.

USAGE:

MSC-250:	set_mcf
MSC-850/32:	set_mcf
MSC-850:	set_mcf
MSC-800:	N/A

set_mcf (MSC-250 controller 3)

SYNTAX:

label set_mcf controller#,variable

PARAMETERS:

controller#	controller id# Range: MSC-250	Always 3
variable	Multi-function controller configuration value.	

DESCRIPTION:

This instruction is used to configure the multi-function controller on the MSC-250, which is controller #3. This controls the Pseudo Axis, Master Angle Bus Network, Fiber Optic Network and Programmable Limit Switch (PLS) functions.

RETURNS:

None.

USAGE:

MSC-250:	set_mcf
MSC-850/32:	set_mcf
MSC-850:	set_mcf
MSC-800:	N/A

set_mcf (HPL-850)**SYNTAX:**

label set_mcf controller#, variable

PARAMETERS:

controller#	controller id# Range: MSC-850/32 1 to 8 MSC-850 1 to 8
variable	This value defines the source of the master angle which will drive the Programmable Limit Switches.

DESCRIPTION:

The value of variable above can be only one of the following:

- 1) If variable = 0, there is no source for the PLS. This disables the PLS function.
- 2) If variable = 1, the source of the master angle is Master Angle Bus A.
- 3) If variable = 2, the source of the master angle is Master Angle Bus B.

RETURNS:

None.

USAGE:

MSC-250:	set_mcf
MSC-850/32:	set_mcf
MSC-850:	set_mcf
MSC-800:	N/A

set_nar_ang**SYNTAX:**

label set_nar_ang controller#,variable

PARAMETERS:

controller#	controller id#	
	Range: MSC-250	1 to 2
variable	Desired offset in bits	
	Range: 64 to 4096	

DESCRIPTION:

While the axis is in the idle state, this instruction allows the following error window to be changed from a default of 200 bits to the above range.

RETURNS:

None.

USAGE:

MSC-250: set_nar_ang

set_offset

SYNTAX:

label set_offset controller#,value

PARAMETERS:

controller#	controller id#
	Range: MSC-250 1 to 2
	MSC-850/32 1 to 8
	MSC-850 1 to 8
value	Desired offset in bits
	Range: -32768 to +32767

DESCRIPTION:

Offsets all positions by the specified value in bits.

This instruction will cause the motor to move the distance specified by 'value' within the next motor interrupt.

RETURNS:

None.

USAGE:

MSC-250:	set_offset
MSC-850/32:	set_offset
MSC-850:	set_offset
MSC-800:	set_offset

set_ovd_mode

SYNTAX:

label set_ovd_mode controller#,mode

PARAMETERS:

controller#	controller id# Range: MSC-250 1 to 2 MSC-850/32 1 to 8 MSC-850 1 to 8
mode	Selects the section of the motion profile where the overdraw sensor is active. Range: 0 to 1

DESCRIPTION:

Selects the section of the motion profile where the overdraw sensor is active.

Mode 0 - the sensor is active through the entire profile.

Mode 1 - the sensor is only active during the overdraw section of the profile.

In mode 0, when the sensor is activated during the index part of his profile, the index will be treated as a regular index with no search speed section executed.

RETURNS:

None.

USAGE:

MSC-250:	set_ovd_mode
MSC-850/32:	set_ovd_mode
MSC-850:	set_ovd_mode
MSC-800:	N/A

set_pls_ang (MCF-850)

SYNTAX:

label set_pls_ang controller#,on_angle,off_angle,module#

PARAMETERS:

controller#	controller id# Range: MSC-850/32 1 to 8 MSC-850 1 to 8
on_angle	Turn ON module# at this angle in bits. Range: 0 to 4095 bits (representing 360 degrees)
off_angle	Turn OFF module# at this angle in bits. Range: 0 to 4095 bits (representing 360 degrees)
module#	Desired output module on the PLS-850 output rack. Range: 0 to 23

DESCRIPTION:

'pls' is an acronym for Programmable Limit Switch.

This instruction defines the action that will occur on the designated 'pls' output unit.

'Module#' 0 to 15 are mapped to the PLS outputs. 'Module#' 16 to 23 are mapped internally to flags.

Each time a **set_pls_ang** instruction is issued, the axis controller executes a sorting algorithm and will activate its' CALCULATION IN PROGRESS flag. The program must verify that this flag is no longer active before executing subsequent **set_pls_ang** instructions.

The 'rollover point' is the master cycle length in bits. The default 'rollover point' is 4096. That is, the cycle is defined as a value between 0 and 4095. For an MCF-850 controller, the 'rollover point' can be modified to be a value (in bits) of 4096, 8192, 16384, 32768 or 65536. The 'rollover point' can be changed from the default value of 4096 by using the set_pls_cnt instruction. The programmed PLS ON/OFF angles can still only range from 0 to 4095. These angles will be automatically scaled by the controllers operating system to function with the new 'rollover point'.

RETURNS:

None.

USAGE:

MSC-250:	set_pls_ang
MSC-850/32:	set_pls_ang
MSC-850:	set_pls_ang
MSC-800:	N/A

set_pls_ang (HPL-850)**SYNTAX:**

label **set_pls_ang** controller#,on_angle,off_angle,module#

PARAMETERS:

controller#	controller id# Range: MSC-850/32 1 to 8 MSC-850 1 to 8
on_angle	Turn ON module# at this angle in bits. Range: 0 to 4095 bits (representing 360 degrees)
off_angle	Turn OFF module# at this angle in bits. Range: 0 to 4095 bits (representing 360 degrees)
module#	Desired output module on the PLS-850 output rack. Range: 0 to 23

DESCRIPTION:

'pls' is an acronym for Programmable Limit Switch.

This instruction defines the action that will occur on the designated 'pls' output unit.

'Module#' 0 to 15 are mapped to the PLS outputs. 'Module#' 16 to 23 are mapped internally to flags.

Each time a **set_pls_ang** instruction is issued, the axis controller executes a sorting algorithm and will activate its' CALCULATION IN PROGRESS flag. The program must verify that this flag is no longer active before executing subsequent **set_pls_ang** instructions.

The 'rollover point' is the master cycle length in bits. The default 'rollover point' is 4096. That is, the cycle is defined as a value between 0 and 4095. For an MCF-850 controller, the 'rollover point' can be modified to be a value (in bits) of 4096, 8192, 16384, 32768 or 65536. The 'rollover point' can be changed from the default value of 4096 by using the set_pls_cnt instruction. The programmed PLS ON/OFF angles can still only range from 0 to 4095. These angles will be automatically scaled by the controllers operating system to function with the new 'rollover point'.

RETURNS:

None.

USAGE:

MSC-250:	set_pls_ang
MSC-850/32:	set_pls_ang
MSC-850:	set_pls_ang
MSC-800:	N/A

set_pls_ang (MSC-250)

SYNTAX:

label set_pls_ang controller#,on_angle,off_angle,module#

PARAMETERS:

controller#	controller id# Range: MSC-250 Always 3
on_angle	Turn ON module# at this angle in bits. Range: 0 to 4095 bits (representing 360 degrees)
off_angle	Turn OFF module# at this angle in bits. Range: 0 to 4095 bits (representing 360 degrees)
module#	Desired output module on the PLS-850 output rack. Range: 0 to 23

DESCRIPTION:

'pls' is an acronym for Programmable Limit Switch.

This instruction defines the action that will occur on the designated 'pls' output unit.

'Module#' 0 to 15 are mapped to the PLS outputs. 'Module#' 16 to 23 are mapped internally to flags.

Each time a **set_pls_ang** instruction is issued, the axis controller executes a sorting algorithm and will activate its' CALCULATION IN PROGRESS flag. The program must verify that this flag is no longer active before executing subsequent **set_pls_ang** instructions.

RETURNS:

None.

USAGE:

MSC-250:	set_pls_ang
MSC-850/32:	set_pls_ang
MSC-850:	set_pls_ang
MSC-800:	N/A

set_pls_cnt (MCF-850)**SYNTAX:**

label set_pls_cnt controller#,count

PARAMETERS:

controller#	controller id# of the HPL-850 controller. Range: MSC-850/32 1 to 8 MSC-850 1 to 8
count	Sets the cycle length for the HPPLS in bits/cycle. Range: 0 to 4

DESCRIPTION:

This instruction sets the cycle length for the PLS function in the MCF-850 card. The default PLS cycle length is 4096 master bits per cycle. The following table shows the relationship between the 'count' value and the number of master bits per PLS cycle.

<u>count</u>	<u>master bits per PLS cycle</u>
0	4096
1	8192
2	16384
3	32768
4	65536

Changing the cycle length, however, does not change the allowable range of PLS ON/OFF angles. This will still be a value from 0 to 4095, scaled by the controller to work within the new range of bits per cycle.

RETURNS:

None.

USAGE:

MSC-250:	N/A
MSC-850/32:	set_pls_cnt
MSC-850:	set_pls_cnt
MSC-800:	N/A

set_pls_cnt (HPL-850)**SYNTAX:**

label set_pls_cnt controller#,count

PARAMETERS:

controller#	controller id# of the HPL-850 controller. Range: MSC-850/32 1 to 8 MSC-850 1 to 8
count	Sets the cycle length for the HPPLS in bits/cycle. Range: 256 to 8,388,607

DESCRIPTION:

'HPPLS' is an acronym for High Performance Programmable Limit Switch.

This instruction sets the cycle length for the HPPLS (360 degree reference). The 'count' is entered in terms of bits/cycle on the Master Angle Bus.

Changing the cycle length changes the allowable range of ON/OFF angles that may be entered, up to the specified cycle length as indicated by this instruction.

RETURNS:

None.

USAGE:

MSC-250:	N/A
MSC-850/32:	set_pls_cnt
MSC-850:	set_pls_cnt
MSC-800:	N/A

set_pls_mask

SYNTAX:

label set_pls_mask controller#,variable

PARAMETERS:

controller#	controller id#
	Range: MSC-250 Always 3
	MSC-850/32 1 to 8
	MSC-850 1 to 8
variable	The currently defined 'pls' mask value.

DESCRIPTION:

'pls' is an acronym for Programmable Limit Switch.

Defines the 'pls' mask value used to enable/disable pls functioning of the pls outputs. The 3 low order bytes of 'variable' represent the 24 associated output modules/flags.

Those bits set in 'variable' will be masked with the currently defined 'pls' state to determine which 'pls' modules will continue to be affected.

The 'pls mask' is used to control the activity of the output modules without clearing and/or redefining the output modules using the **set_pls_ang** instruction.

RETURNS:

None.

USAGE:

MSC-250:	set_pls_mask
MSC-850/32:	set_pls_mask
MSC-850:	set_pls_mask
MSC-800:	N/A

set_pls_time

SYNTAX:

label set_pls_time controller#,time,module#

PARAMETERS:

controller#	controller id# of the HPL-850 controller. Range: MSC-850/32 1 to 8 MSC-850 1 to 8
time	Turn on/off 'module#' in advance of crossing its specified angle. Enter 'time' as milliseconds*10. Range: 0 to 255
module#	Desired output module on the PLS-850 output rack. Range: 0 to 15

DESCRIPTION:

'HPPLS' is an acronym for High Performance Programmable Limit Switch.

This instruction assigns a time advance to the specified 'module#'.

This will cause 'module#' to trigger 'time/10 milliseconds' in advance of crossing the specified on/off angle.

NOTE:

This instruction is ONLY valid when used with the HPL-850.

RETURNS:

None.

USAGE:

MSC-250:	N/A
MSC-850/32:	set_pls_time
MSC-850:	set_pls_time
MSC-800:	N/A

set_speed

SYNTAX:

label set_speed controller#,speed

PARAMETERS:

controller#	controller id#
	Range: MSC-250 1 to 3
	MSC-850/32 1 to 3
	MSC-850 1 to 8
	MSC-800 1 to 9
speed	Desired speed in RPM.
	Range: MSC-250 5 to 7200 RPM
	MSC-850/32 5 to 3600 RPM
	MSC-850 5 to 3600 RPM
	MSC-800 5 to 3600 RPM

DESCRIPTION:

Set the speed of the specified controller to 'speed'.

RETURNS:

None.

USAGE:

MSC-250:	set_speed
MSC-850/32:	set_speed
MSC-850:	set_speed
MSC-800:	set_speed

set_swi_mask**SYNTAX:**

label set_swi_mask variable

PARAMETERS:

variable The software interrupt mask variable.

DESCRIPTION:

Determines which of the 32 software interrupts are active. A value of 1 in the corresponding bit of the mask 'variable' enables the software interrupt.

RETURNS:

None.

USAGE:

MSC-250: set_swi_mask
MSC-850/32: set_swi_mask
MSC-850: N/A
MSC-800: N/A

set_tmr**SYNTAX:**

label set_tmrtimer_flag#,time

PARAMETERS:

timer_flag#	Timer flag number. Range: 72 to 79
time	Time to set.

DESCRIPTION:

Activates or enables a timer for a time = 'time'. Time is given in .01 second intervals.

The specified flag remains set until 'time' has expired.

RETURNS:

None.

USAGE:

MSC-250:	set_tmr
MSC-850/32:	set_tmr
MSC-850:	set_tmr
MSC-800:	set_tmr

set_trig_cam

SYNTAX:

label set_trig_cam controller#,master_angle

PARAMETERS:

controller#	controller id#
	Range: MSC-250 1 to 2
	MSC-850/32 1 to 8
	MSC-850 1 to 8
master_angle	Angle of the master controller.
	Range: 0 to +4095

DESCRIPTION:

Sets the trigger point for lock methods 8 & 9 (see **lock**). When the master controller angle crosses the trigger angle from either direction, the slave axis begins executing the cam at the specified cam pointer. (The cam pointer will be set to zero if no **set_cam_ptr** instruction has been executed prior to the lock).

RETURNS:

None.

USAGE:

MSC-250:	set_trig_cam
MSC-850/32:	set_trig_cam
MSC-850:	set_trig_cam
MSC-800:	N/A

set_trig_pw

SYNTAX:

label set_trig_pw controller#,master_angle

PARAMETERS:

controller#	controller id#
	Range: MSC-250 1 to 2
	MSC-850/32 1 to 8
	MSC-850 1 to 8
	MSC-800 1 to 8
master_angle	Angle of the master controller.
	Range: 0 to +4095

DESCRIPTION:

Sets the trigger point for lock method 3 (see **lock**). When the master controller angle equals the angle position, a Piecewise profile is executed.

RETURNS:

None.

USAGE:

MSC-250:	set_trig_pw
MSC-850/32:	set_trig_pw
MSC-850:	set_trig_pw
MSC-800:	set_trig_pw

set_vgain**SYNTAX:**

label set_vgain controller#,vel_gain

PARAMETERS:

controller#	controller id#
	Range: MSC-250 1 to 2
	MSC-850/32 1 to 8
vel_gain	Feed forward gain value.

DESCRIPTION:

This command sets the velocity feed forward gain for the specified 'controller id#'. During calculation of the position output value for the specified controller, the velocity feed forward gain term is multiplied by the current commanded velocity. The resultant value is added to the other digital compensation terms for that controller.

A starting value for the velocity gain term may be calculated as follows:

$$Vg = \frac{K * 402,650}{BPR}$$

where 'K' is the velocity scale factor for the motor/drive system, in volts per 1000 RPM, and 'BPR' is the number of transducer bits per revolution, after quadrature, of the motor shaft. For example, 'BPR' would be 4096 for a 1024 line encoder.

This instruction is used to compensate for following error caused by an extend velocity mode motor and driver.

RETURNS:

None.

USAGE:

MSC-250:	set_vgain
MSC-850/32:	set_vgain
MSC-850:	N/A
MSC-800:	N/A

set_wide_ang

SYNTAX:

label set_wide_ang controller#,variable

PARAMETERS:

controller#	controller id#
	Range: MSC-250 1 to 2
variable	Desired value in bits
	Range: 1024 to 32768 bits

DESCRIPTION:

Allows the following error window to be changed from a default of 2048 bits/gain to the above range while the axis is in motion.

RETURNS:

None.

USAGE:

MSC-250: set_wide_ang

stop_input

SYNTAX

label stop_input

PARAMETERS:

None.

DESCRIPTION:

Terminates all active **input** instructions and clears the input buffer area of all characters.

RETURNS:

None.

USAGE:

MSC-250: stop_input
MSC-850/32: stop_input
MSC-850: stop_input
MSC-800: stop_input

swi_if_off**SYNTAX:**

label swi_if_off interrupt#,flag,subroutine_label

PARAMETERS:

interrupt#	Interrupt number. Range: 0 to 31
flag	Flag to trigger the interrupt. Range: 0 to 255
subroutine_label	Subroutine branch address.

DESCRIPTION:

Sets up an event which will trigger when 'flag' changes from on to off.

RETURNS:

None.

USAGE:

MSC-250:	swi_if_off
MSC-850/32:	swi_if_off
MSC-850:	swi_if_off
MSC-800:	off_swi

swi_if_on**SYNTAX:**

label swi_if_on interrupt#,flag,subroutine_label

PARAMETERS:

interrupt#	Interrupt number. Range: 0 to 31
flag	Flag to trigger the interrupt. Range: 0 to 255
subroutine_label	Subroutine branch address.

DESCRIPTION:

Sets up an event which will trigger when 'flag' changes from off to on.

RETURNS:

None.

USAGE:

MSC-250:	swi_if_on
MSC-850/32:	swi_if_on
MSC-850:	swi_if_on
MSC-800:	on_swi

switch_cam

SYNTAX:

label switch_cam controller#,start element,# of elements

PARAMETERS:

controller#	controller id#
	Range: MSC-250 1 to 2
	MSC-850/32 1 to 8
	MSC-850 1 to 8
start element	Element relative to axis card memory array element zero
# of elements	Number of cam elements

DESCRIPTION:

The SWITCH CAM feature allows the programmer to switch from executing one cam to another. The programmer specifies the starting element relative to element 0 in the axis controller's 28K array and the number of elements to be executed.

The manner in which the SWITCH CAM feature works, is dependent on a number of parameters:

- 1) the **lock** type being used
- 2) the order in which the **switch_cam** and **lock** instructions are executed
- 3) the current state of the axis controller, as defined by its axis status flags

When a cam is being executed and a **switch_cam** instruction is issued, the LOCK PENDING status flag is set until the cam "rolls" from the last to the first or the first to the last cam element. When the "roll" occurs, the LOCK PENDING flag is cleared, the MASTER/SLAVE lock is set and the new cam begins execution.

Execution of an **unlock** command will clear the LOCK PENDING axis status flag, if currently set.

EXAMPLE 1:

When using the **switch_cam** instruction with lock mode 0 or 5 (assume the axis is not busy and a **cam_data** instruction has been executed).

MSC COMMAND	RESULT
-----	-----
switch_cam	a) sets the start/end of cam pointers b) no affect on axis status flags
lock	a) turn on MASTER/SLAVE LOCK and BUSY flags

In this example, the switch takes place immediately (affecting the cam pointers) and the actual lock will occur at the moment the **lock** command is issued.

EXAMPLE 2:

When using the **switch_cam** instruction with lock mode 0 or 5 (assume the axis is not busy and a **cam_data** instruction has been executed).

MSC COMMAND	RESULT
-----	-----
lock	a) turn on MASTER/SLAVE LOCK and BUSY flags
switch_cam	a) sets the start/end of cam pointers for the next cam to be executed b) turns on the LOCK PENDING flag

In this example, a lock takes place immediately. The **switch_cam** instruction will set up the cam pointers to be used for the next cam, that is, the cam to be executed upon completion of the current cam.

EXAMPLE 3:

When using the **switch_cam** instruction with lock mode 8 or 9 (assume the axis is not busy and a **cam_data** instruction has been executed).

MSC COMMAND	RESULT
-----	-----
set_trig_cam	a) sets the lock trigger angle
switch_cam	a) sets the start/end of cam pointers b) no affect on axis status flags
lock	a) turn on LOCK PENDING and BUSY flags

In this example, the switch takes place immediately (affecting the cam pointers) and the actual lock will occur at the moment the master angle has crossed the defined trigger angle. Once the lock takes place, the LOCK PENDING flag is turned off and the MASTER/SLAVE LOCK flag is turned on.

EXAMPLE 4:

When using the **switch_cam** instruction with lock mode 8 or 9 (assume the axis is not busy and a **cam_data** instruction has been executed).

MSC COMMAND	RESULT
-----	-----
set_trig_cam	a) sets the lock trigger angle
lock	a) turn on LOCK PENDING and BUSY flags
switch_cam	if the LOCK PENDING flag is on, then

- a) sets the start/end of cam pointers
 - b) no affect on axis status flags
 - c) the actual lock occurs at the trigger angle
- if the MASTER/SLAVE LOCK flag is on, then
- a) turns on the LOCK PENDING flag
 - b) the switch occurs upon completion of the current cam

In this example, if the LOCK PENDING flag is on at the moment the switch is executed, the result of the switch is that the cam start/end pointers will be realigned and the lock will still occur when the master angle crosses the trigger angle.

In this example, if the LOCK PENDING flag is off at the moment the switch is executed (the master angle has already crossed the trigger angle), the current cam will be completed and then the switch will occur.

NOTES:

- 1) The **lock** instruction will only be executed when the BUSY flag is off, otherwise, it will be ignored.
- 2) A **switch_cam** instruction will override a previous **switch_cam** instruction that has not been already implemented.
- 3) A **set_trig_cam** instruction will override a previous **set_trig_cam** instruction that has not been already implemented.

RETURNS:

None.

USAGE:

MSC-250: switch_cam
MSC-850/32: switch_cam
MSC-850: switch_cam
MSC-800: N/A

sys_fault

SYNTAX:

label sys_fault

PARAMETERS:

None.

DESCRIPTION:

Stop execution of the program and set the system fault bit in the MSC status word.

An **f_decel** instruction is executed for all controllers.

RETURNS:

None.

USAGE:

MSC-250:	sys_fault
MSC-850/32:	sys_fault
MSC-850:	sys_fault
MSC-800:	sys_fault

sys_return**SYNTAX:**

label sys_return

PARAMETERS:

None.

DESCRIPTION:

Stop execution of the program and set the system return bit in the MSC status word.

An **f_decel** instruction is executed for all controllers.

RETURNS:

None.

USAGE:

MSC-250:	sys_return
MSC-850/32:	sys_return
MSC-850:	sys_return
MSC-800:	sys_return

test_mode**SYNTAX:**

label test_mode controller#

PARAMETERS:

controller#	controller id#	
	Range: MSC-250	1 to 2
	MSC-850/32	1 to 8
	MSC-850	1 to 8

DESCRIPTION:

The designated controller will be put into **test** mode. See the hardware manual for the particular controller to determine the function of test mode.

RETURNS:

None.

USAGE:

MSC-250:	test_mode
MSC-850/32:	test_mode
MSC-850:	test_mode
MSC-800:	N/A

text**SYNTAX:**

label text "ASCII string"

PARAMETERS:

string ASCII string (enclosed in quotes).

DESCRIPTION:

Defines a string of characters for use with the **print** and **input** instructions.

Control and special keyboard characters which cannot be typed may be used by entering the ASCII decimal equivalent enclosed in '<' and '>'.

This instruction requires a 'label'.

RETURNS:

None.

USAGE:

MSC-250: text
MSC-850/32: text
MSC-850: text
MSC-800: text

track_spd

SYNTAX:

label track_spd controller#,speed

PARAMETERS:

controller#	controller id#
	Range: MSC-250 1 to 3
	MSC-850/32 1 to 8
	MSC-850 1 to 8
	MSC-800 1 to 9
speed	Desired speed.
	Range: -3600 to +3600 RPM

DESCRIPTION:

The specified controller tracks (changes to) the speed indicated. All speed changes occur at the previously set accel/decel rate. The speed may be changed at any time.

This instruction sets the controller status flags **BUSY** and **JOGGING**. These flags remain set until a **f_decel** instruction causes the motor to reach zero speed.

This instruction will be ignored if the controller has not received a **drive_on** instruction, or if the controller is busy executing a motion instruction. If the instruction cannot be executed, the axis status flag **COMMAND INVALID IN THIS STATE** will be set. This flag will automatically be cleared by the controller at the next communication between the main processor and the axis controller. This occurs automatically every 100 milliseconds even if no program instruction directs communication to occur.

RETURNS:

None.

USAGE:

MSC-250:	track_spd
MSC-850/32:	track_spd
MSC-850:	track_spd
MSC-800:	track_spd

trap_pos**SYNTAX:**

label trap_pos controller#

PARAMETERS:

controller#	controller id#	
	Range: MSC-250	1 to 2
	MSC-850/32	1 to 8
	MSC-850	1 to 8

DESCRIPTION:

When the hardware interrupt signal on the designated motion controller is detected, the current position will be saved immediately.

This position can be retrieved later using the **get_trap_pos** instruction.

EXAMPLE

```
hwi_armed_1 equ 91    interrupt armed flag
.
.
enable_hwi
trap_pos      1

! wait for indication that position has been trapped

loop

if_stat_on    1,hwi_armed_1,loop
get_trap_pos  1,pos
.
```

RETURNS:

None.

USAGE:

MSC-250:	trap_pos
MSC-850/32:	trap_pos
MSC-850:	trap_pos
MSC-800:	N/A

turn_off

SYNTAX:

label turn_off I/O flag#

PARAMETERS:

I/O flag#	Output module number.
Range: MSC-250	0 to 47
MSC-850/32	0 to 71
MSC-850	0 to 71
MSC-800	0 to 71

DESCRIPTION:

Causes the specified I/O flag to be turned off. If the corresponding I/O module is an output module, the output module will turn off.

The **turn_off** instruction should not be used for I/O flag positions equipped with input modules. This will cause the MSC to designate that I/O flag as an output. The system will no longer respond correctly to the corresponding input module.

RETURNS:

None.

USAGE:

MSC-250:	turn_off
MSC-850/32:	turn_off
MSC-850:	turn_off
MSC-800:	turn_off

turn_on**SYNTAX:**

label turn_on I/O flag#

PARAMETERS:

I/O flag#	Output module number.
Range: MSC-250	0 to 47
MSC-850/32	0 to 71
MSC-850	0 to 71
MSC-800	0 to 71

DESCRIPTION:

Causes the specified I/O flag to be turned on. If the corresponding I/O module is an output module, the output module will turn on.

The **turn_on** instruction should not be used for I/O flag positions equipped with input modules. This will cause the MSC to designate that I/O flag as an output. The system will no longer respond correctly to the corresponding input module.

RETURNS:

None.

USAGE:

MSC-250:	turn_on
MSC-850/32:	turn_on
MSC-850:	turn_on
MSC-800:	turn_on

unlock

SYNTAX:

label unlock controller#,mode#

PARAMETERS:

controller#	controller id#
	Range: MSC-250 1 to 2
	MSC-850/32 1 to 8
	MSC-850 1 to 8
	MSC-800 1 to 8
mode#	unlock method
	Range: 0 or 1

DESCRIPTION:

Terminate the master/slave or CAM lock.

Using 'mode' 0, the controller will decelerate to zero speed at the previously set accel/decel rate.

Using 'mode' 1, the controller will unlock from the master, but will continue to run at the last commanded speed. It will not decelerate until it is commanded to do so using the **f_decel** instruction.

RETURNS:

None

USAGE:

MSC-250:	unlock
MSC-850/32:	unlock
MSC-850:	unlock
MSC-800:	unlock

vel_ccw**SYNTAX:**

label vel_ccw controller#

PARAMETERS:

controller#	controller id#	
	Range: MSC-250	1 to 3
	MSC-850/32	1 to 8
	MSC-850	1 to 8
	MSC-800	1 to 9

DESCRIPTION:

The specified controller accelerates and runs in a counter-clockwise direction at the previously set accel/decel rate and speed. The current accel/decel rate is divided by 256 before acceleration begins.

Accel/decel rate and speed may be modified while executing the **vel_ccw** instruction.

This instruction sets the controller status flags **BUSY** and **JOGGING**. These flags remain set until a **f_decel** instruction causes the motor to reach zero speed.

This instruction will be ignored if the controller has not received a **drive_on** instruction, or if the controller is busy executing a motion instruction. If the instruction cannot be executed, the axis status flag **COMMAND INVALID IN THIS STATE** will be set. This flag will automatically be cleared by the controller at the next communication between the main processor and the axis controller. This occurs automatically every 100 milliseconds even if no program instruction directs communication to occur.

RETURNS:

None.

USAGE:

MSC-250:	vel_ccw
MSC-850/32:	vel_ccw
MSC-850:	vel_ccw
MSC-800:	vel_ccw

vel_cw

SYNTAX:

label vel_cw controller#

PARAMETERS:

controller#	controller id#
	Range: MSC-250 1 to 3
	MSC-850/32 1 to 8
	MSC-850 1 to 8
	MSC-800 1 to 9

DESCRIPTION:

The specified controller accelerates and runs in a clockwise direction at the previously set accel/decel rate and speed. The current accel/decel rate is divided by 256 before acceleration begins.

Accel/decel rate and speed may be modified while executing the **vel_cw** instruction.

This instruction sets the controller status flags **BUSY** and **JOGGING**. These flags remain set until a **f_decel** instruction causes the motor to reach zero speed.

This instruction will be ignored if the controller has not received a **drive_on** instruction, or if the controller is busy executing a motion instruction. If the instruction cannot be executed, the axis status flag **COMMAND INVALID IN THIS STATE** will be set. This flag will automatically be cleared by the controller at the next communication between the main processor and the axis controller. This occurs automatically every 100 milliseconds even if no program instruction directs communication to occur.

RETURNS:

None.

USAGE:

MSC-250:	vel_cw
MSC-850/32:	vel_cw
MSC-850:	vel_cw
MSC-800:	vel_cw
MSC-100:	vel_cw

write

SYNTAX:

label write unit,data_area,length,status

PARAMETERS:

unit	File identifier. Range: 1 to 8
data_area	Data source address. May be defined by begin_data , begin_cam , dim , or variable name.
length	Number of bytes (characters) to be written.
status	Variable containing the result of the operation.

DESCRIPTION:

This instruction writes data from the given 'data_area' to the specified unit (file). The number of bytes (characters) to be written is given as 'length'.

RETURNS:

The return variable will be zero if the operation is successful. Non-zero status codes are described in Section 15.3.2.

USAGE:

MSC-250:	write
MSC-850/32:	write
MSC-850:	write
MSC-800:	write

Analog Functions

```

!-----
! TITLE:      Analog Functions
!
! DESCRIPTION: This program outputs an analog signal which changes in
!              amplitude from -10 Vdc to +10 Vdc and does so cyclically.
!              The output is verified by hardwiring it to an input and
!              monitoring that input.
!
!              Hardwire is wired in the following manner:
!              chl = ch 9
!
! EQUIPMENT:   MSC250
!
! COMMANDS: analog_in      analog_out      analog_rt
!              analog_zo
!
! The above commands are commonly used in many programs.  This
! illustration provides the context and syntax required to successfully
! integrate these commands into a complete program.  While the use of
! these commands varies from program to program, the following example
! will give the programmer an indication of a typical usage.
!-----

!-----
!
! Declare constants and variables
!-----

!
! Axis related constants
!

ANALOG      equ      4      ! Analog controller card

CHANNEL_IN  equ      1      ! Analog input channel.
CHANNEL_OUT equ      9      ! Analog output channel.

!
! I/O and Flags
!

SYS_OK      equ      7      ! System okay (output)

TIMER      equ      72      ! Timer to create time
!                          delays.

!
! Variable definitions
!

rate        integer      ! The slew rate used for the analog
!                          channels.
change      integer      ! The incremental change for the analog

```

```

!                                     output.
alog_out      integer      ! Defines the value in bits of the analog
!                                     output.
alog_in       integer      ! Defines the value in bits of the analog
!                                     input.
diff          integer      ! Stores the difference between two
!                                     variables.
time          integer      ! The time used to create a delay.

!-----
!                                     -
! Initialize variables                                     -
!                                     -
!-----

Defaults      let          rate=2048
               let          change=256
               let          alog_out=0
               let          time=20

!-----
!                                     -
! Program Setup                                           -
! Define the analog zero and slew rates for the channels. -
!                                     -
!-----

Setup          analog_zo    ANALOG,CHANNEL_IN,0
               gosub        Delay
               analog_rt    ANALOG,CHANNEL_IN,rate
               analog_rt    ANALOG,CHANNEL_OUT,rate
               gosub        Delay

!-----
!                                     -
! Main body of the program.                               -
! Adjust the analog output by an incremental amount.      -
! Wait for the output to settle.                          -
! Measure the analog input and compare the values.        -
! If out of range then stop system, otherwise continue in Loop. -
!                                     -
!-----

Main           turn_on      SYS_OK

Loop           let          alog_out=alog_out+change
               if           alog_out<2049,Adjust_Out
               let          alog_out=-2047

Adjust_Out     analog_out    ANALOG,CHANNEL_OUT,alog_out
               gosub        Delay

               analog_in    ANALOG,CHANNEL_IN,alog_in
               let          diff=alog_out-alog_in
               let          diff=abs(diff)
               if           diff<50,Loop

```

```

        turn_off      SYS_OK
        sys_fault

!-----
!
! Subroutine used to create a time delay.
!  Wait until the timer times out.
!
!-----
Delay      set_tmr      TIMER,time
Wait_Time  if_tmr_on    TIMER,Wait_Time
           return_sub
```

CAMS with switch_cam

```

!-----
! TITLE:          CAMS with switch_cam
!
! DESCRIPTION: This program uses two distinct CAMs to illustrate
!              the use of a 'switch_cam' and other associated cam
!              functions. Software interrupts are used to demonstrate
!              the ability to change cams on the fly, as well as
!              the ability to reposition and lock a cam at a
!              different point.
!
! EQUIPMENT:     MSC850/32
!                ACR850C in slot one
!                MCF850C in slot three
!
! COMMANDS: begin_cam          cam          cam_data
!             clr_flag          end_cam      f_decel
!             get_cam_cnt       get_cam_end  get_cam_ptr
!             get_cam_strt      get_map      get_map_stat
!             get_mcf           get_status   if_flag_off
!             if_flag_on        if_io_off    if_stat_on
!             lock              restart_at   set_cam_ptr
!             set_flag          set_map      set_mcf
!             switch_cam        sys_fault    sys_return
!             turn_off          turn_on      unlock
!
! The above commands are commonly used in many programs. This
! illustration provides the context and syntax required to successfully
! integrate these commands into a complete program. While the use of
! these commands varies from program to program, the following example
! will give the programmer an indication of a typical usage.
!-----
!
! msc_type          850/32
! declare           on
!
!-----
!
! Declare constants and variables
!-----
!
! Axis related constants
!
PEN          equ          1
PSEUDO       equ          3
!
BUSY          equ          78          ! busy flag offset
DOWN          equ          77          ! down flag offset
PEN_BUSY      equ          (PEN*16)+BUSY ! busy flag - PEN
PSEUDO_BUSY   equ          (PSEUDO*16)+BUSY ! busy flag - PSEUDO
PEN_DOWN      equ          (PEN*16)+DOWN ! down flag - PEN

```

```

PEN_SWITCH      equ          81

UNLK_MODE_0     equ          0      ! unlock mode 0
LK_MODE_0       equ          0      ! lock mode 0
LK_MODE_5       equ          5      ! lock mode 5
MS              equ          6      ! 1/64
DS              equ          0      ! no multiply
PSEUDO_MABA     equ         10      ! PSEUDO sends data to
!                                     MABA
MABA_TALKER     equ         29      ! PSEUDO is talker on MABA
MABA_LISTEN     equ         23      ! PEN is listener on MABA
END_CAM         equ        128      ! Value for end of cam

!
!   I/O and Flags
!

SWITCH_RQST     equ          0      ! Switch request (input)
REPEAT         equ          1      ! Repeat section of cam
!                                     (input)
RESET           equ          6      ! Reset a PEN fault (input)
SYS_OK          equ          7      ! System okay (output)

TIMER           equ         72      ! Timer for fixed delay.

IN_CAM_FWD      equ        254      ! Flag to indicate current
!                                     cam.
SWITCH          equ        255      ! Flag to indicate a switch
!                                     request.

!
!   Variable definitions
!

speed           integer       ! Defines the speed of the PSEUDO.
length          integer       ! Determines the length of the cam.
start           integer       ! Starting location of cam.
cam_start       integer       ! Return variable for cam starting
!                                     location.
cam_end         integer       ! Return variable for cam ending
!                                     location.
number_el       integer       ! Number of cam elements.
count          integer       ! Return variable for number of cam
!                                     cycles.
pointer         integer       ! Return variable for the cam pointer.
new_pointer     integer       ! The variable used to reposition the
!                                     cam pointer.
last_el        integer       ! Defines the location of the last
!                                     cam element.
temp           integer       ! Used as a temporary storage variable.
ctr            integer       ! Used as a loop counter.
ctr2           integer       ! Used as a loop counter.
time           integer       ! Determines the delay time for a
!                                     timer.
time2          integer       ! Determines the delay time for a
!                                     timer.

```

[illegible]


```

cam      -32,-32,-32,-32,-32,-32,-32,-32
cam      -32,-32,-32,-32,-32,-32,-32,-32
cam      -32,-32,-32,-32,-32,-32,-32,-32
cam      -32,-32,-32,-32,-32,-32,-32,-32
cam      -32,-32,-32,-32,-32,-32,-32,-32
cam      -32,-32,-32,-32,-32,-32,-32,-32
cam      -32,-32,-32,-32,-32,-32,-32,-32
cam      -32,-32,-32,-32,-32,-32,-32,-32
cam      64,64,64,64,64,64,64,64
cam      64,64,64,64,64,64,64,64
cam      64,64,64,64,64,64,64,64
cam      64,64,64,64,64,64,64,64
cam      64,64,64,64,64,64,64,64
cam      64,64,64,64,64,64,64,64
cam      64,64,64,64,64,64,64,64
cam      64,64,64,64,64,64,64,64
cam      0,0,0,0,0,0,0,0
cam      0,0,0,0,0,0,0,0
cam      0,0,0,0,0,0,0,0
cam      0,0,0,0,0,0,0,0
end_cam

!-----
!
! Initialize variables
!-----

Defaults      let      speed=100
               let      time=1
               let      time2=50

!-----
!
! Program Setup
! Define axis motion parameters.
! Configure map.
! Load cam tables into pen_cams array.
! Define software interrupts.
!-----

Setup      set_ac_dc      PSEUDO,100
           set_ac_dc      PEN,400

           gosub      Set_Config
           gosub      Load_Cams

           swi_if_on      0,PEN_DOWN,Fault
           swi_if_on      1,SWITCH_RQST,Make_Switch
           swi_if_on      2,REPEAT,Reposition_Cam
           enable_swi

!-----
!
! Main body of program
!-----

```

```

! Enable PEN. -
! Start motion. -
! Check conditons for a 'switch_cam'. -
! If switch is not correct then stop system. -
! Get information about current cam. -
! Remain in Loop, waiting for software interrupts. -
! -
!-----
Main          turn_on      SYS_OK
              drive_on    PEN

              lock        PEN,LK_MODE_0
              track_spd   PSEUDO,speed

Loop          get_status   PEN
              if_flag_off SWITCH,Continue
              if_stat_on  PEN_SWITCH,Continue
              get_cam_strt PEN,cam_start
              get_cam_end  PEN,cam_end
              let         number_el=cam_end-cam_start
              let         number_el=number_el+1
              if         number_el<>length,System_Halt
              clr_flag    SWITCH

Continue     get_cam_ptr   PEN,pointer
              get_cam_cnt  PEN,count
              set_tmr     TIMER,time

Delay        if_tmr_on    TIMER,Delay
              goto       Loop

!-----
!
! Software Interrupt Routine indicating a request to switch cams. -
! Test for current cam and switch to alternate cam. -
! Set the flag that indicates a cam switch has occurred. -
! Enable the software interrupts and return. -
! -
!-----

Make_Switch  if_flag_on   IN_CAM_FWD,Switch_Reverse
              set_flag    IN_CAM_FWD
              let         start=0
              let         length=208
              switch_cam   PEN,start,length
              goto       Exit_Switch

Switch_Reverse no_op
              clr_flag    IN_CAM_FWD
              let         start=208
              let         length=208
              switch_cam   PEN,start,length

Exit_Switch  set_flag     SWITCH
              enable_swi
              return_sub

```

```

!-----
!
! Software Interrupt Routine used to repeat a section of the current cam.-
! Stop the PEN. -
! Wait for a short period of time. -
! Reposition the cam pointer then start again. -
! Enable the software interrupts and return. -
! -
!-----

Reposition_Cam no_op
                unlock          PEN,UNLK_MODE_0
Wait_Busy       if_stat_on      PEN_BUSY,Wait_Busy

                set_tmr         TIMER,time2
Wait_Time       if_tmr_on       TIMER,Wait_Time

                let             new_pointer=128
                if             pointer<128,Cont_Repos
                let             new_pointer=0

Cont_Repos      set_cam_ptr     PEN,new_pointer
                lock           PEN,LK_MODE_5
                enable_swi
                return_sub

!-----
!
! Software Interrupt Routine used for a PEN fault condition. -
! Turn off the System Ok output. -
! Check to see that all motion has stopped. -
! Wait for reset input, then restart at Main. -
! -
!-----

Fault           no_op
                turn_off       SYS_OK
                f_decel        PEN
                f_decel        PSEUDO
Wait_Busy1      if_stat_on     PEN_BUSY,Wait_Busy1
                if_stat_on     PSEUDO_BUSY,Wait_Busy1

Start_Over      if_io_off      RESET,Start_Over
                enable_swi
                restart_at     Main

!-----
!
! Subroutine used when an incorrect switch occurs. -
! Turn off the System Ok output. -
! Check to see that all motion has stopped. -
! Disable PEN drive, and indicate a system fault on the System Status. -
! -
!-----

```

```

System_Halt      no_op
                  turn_off          SYS_OK
                  f_decel            PEN
                  f_decel            PSEUDO
Wait_Busy2       if_stat_on          PEN_BUSY,Wait_Busy2
                  if_stat_on          PSEUDO_BUSY,Wait_Busy2

                  drive_off          PEN
                  sys_fault

!-----
!
! Subroutine to configure the MAB map and the MCF map.
! Set the appropriate bits, and check the results.
!-----

Set_Config       let          map=0
                  set_map      map
                  set_bit      MABA_TALKER,map
                  set_bit      MABA_LISTEN,map
                  set_map      map
                  get_map      map_back
                  get_map_stat map_stat

                  if          map_stat<>0,Config_Fail
                  if          map<>map_back,Config_Fail

Config_Pseudo    let          mcf=0
                  set_mcf      PSEUDO,mcf
                  set_bit      PSEUDO_MABA,mcf
                  set_mcf      PSEUDO,mcf

Delay1           set_tmr      TIMER,time
                  if_tmr_on    TIMER,Delay1
                  get_mcf      PSEUDO,mcf_back

                  if          mcf<>mcf_back,Config_Fail

Exit_Config      return_sub

Config_Fail      sys_return

!-----
!
! Subroutine to load cams into pen_cams array.
! Load each cam consecutively into the array.
! The last element in the array needs to indicate the end of the cam.
! Execute a cam_data command to load the axis with the cam information.
! Immediately switch to a subset of the array which will be the
! equivalent of the forward_cam table.
!-----

Load_Cams        no_op
                  let          ctr=0
                  let          length=208
                  let          last_el=length

```

Load_Forward	let_byte	temp=forward_cam[ctr]
	let_byte	pen_cams[ctr]=temp
	let	ctr=ctr+1
	if	ctr<last_el,Load_Forward
Load_Reverse	let	ctr2=0
	let_byte	temp=reverse_cam[ctr2]
	let_byte	pen_cams[ctr]=temp
	let	ctr=ctr+1
	let	ctr2=ctr2+1
	if	ctr2<last_el,Load_Reverse
	let_byte	pen_cams[ctr]=END_CAM
	cam_data	PEN,pen_cams,MS,DS
	let	start=0
	let	length=208
	switch_cam	PEN,start,length
	set_flag	IN_CAM_FWD
	return_sub	

Clock Simulation

```

!-----
! TITLE:          Clock Simulation
!
! DESCRIPTION: This program functions as a fictitious time clock
!              operating on either military or civilian time. The
!              program is used to illustrate many of the arithmetic
!              functions of the controller.
!
! EQUIPMENT:     MSC250
!
! COMMANDS:  clr_bit      declare      equ
!            gosub        goto         if
!            if_bit_clr   if_bit_set   if_tmr_on
!            integer      let          msc_type
!            no_op        rand_int     return_sub
!            set_bit      set_tmr
!
! The above commands are commonly used in many programs. This
! illustration provides the context and syntax required to successfully
! integrate these commands into a complete program. While the use of
! these commands varies from program to program, the following example
! will give the programmer an indication of a typical usage.
!-----
!
! msc_type      250
! declare      ON
!
!-----
!
!-----
!
! Declare constants and variables
!-----
!
TYPE      equ      0      ! Bit used to indicate
!                      clock type
! TYPE=0 for Military; TYPE=1 for Civilian

TIMER      equ      72    ! Timer used for delays

clock      integer
!                      ! The variable used for
!                      clock type
second     integer      ! Counter indicates # of
!                      seconds
minute     integer      ! Counter indicates # of
!                      minutes
hour       integer      ! Counter indicates # of
!                      hours
one_sec    integer      ! # of 10 msec clock ticks

!-----
!
! Initialize variables
!-----

```

```

Defaults      let      second=0
              let      minute=0
              rand_int  23,hour

! Using a 20 msec clock for demonstration purposes.

              let      one_sec=2

!-----
!
! Program Setup
! Check the type of clock previously used and toggle the bit.
! The clock type alternates each time the program is restarted.
! The program starts on a Military type clock whenever a reset command
! is issued from the Analyzer.
!-----

Setup         no_op
Set_Clk_Type  if_bit_set      TYPE,clock,Set_Civilian

Set_Military  set_bit        TYPE,clock
              goto          Main

Set_Civilian  clr_bit        TYPE,clock

!-----
!
! Main body of program.
! The clock counters are incremented appropriately.
! The hour is incremented based on the bit 'TYPE'.
!-----

Main          no_op

Start_Clk     no_op
Time_Seconds  gosub          One_Second
              let      second=second+1
              if      second<60,Time_Seconds
              let      second=0

Time_Minutes  let      minute=minute+1
              if      minute<60,Time_Seconds
              let      minute=0

Time_Hours    no_op
              if_bit_clr  TYPE,clock,Civilian_Type
Military_Type let      hour=hour+1
              if      hour<24,Time_Seconds
              let      hour=0
              goto     Time_Seconds

```

```
Civilian_Type  let          hour=hour+1
                if          hour<13,Time_Seconds
                let          hour=1
                goto         Time_Seconds
```

```
!-----
!  
! Subroutine used to delay a specified amount of time.      -  
!   For demonstration purposes timer will be 20 msec.      -  
!  
!-----
```

```
One_Second     set_tmr      TIMER,one_sec  
Wait_Sec       if_tmr_on    TIMER,Wait_Sec  
  
               return_sub
```


Data Storage Techniques onto EPROM with Operator Interface

```

!-----
! TITLE:           Data Storage Techniques onto EPROM with Operator Interface
!
! DESCRIPTION:     This program will prompt the operator to enter speed,
!                  accel/decel rate and an absolute position
!                  (as motor turns xx.xx) and then execute a move to that
!                  position with the parameters specified.
!
!                  As each entry is made, the program will verify that it is
!                  within the proper range and will alert the operator if
!                  not.
!
!                  The program also allows for the storage and
!                  retrieval of 'setup' data on EPROM. This data
!                  consists of speed, accel/decel and position information.
!
! EQUIPMENT:       MSC850/32
!                  ACR850C in slot one
!                  OPI connected to port two
!
! COMMANDS: case   close           create
!                  default         drive_off       drive_on
!                  end_select      exit_select     get_pq_space
!                  get_space       get_volume      if_char
!                  if_no_char      initialize      input
!                  load            master          open
!                  port_set        print           print_num
!                  read            save            select
!                  stop_input      text            write
!
! The above commands are commonly used in many programs. This
! illustration provides the context and syntax required to successfully
! integrate these commands into a complete program. While the use of
! these commands varies from program to program, the following example
! will give the programmer an indication of a typical usage.
!-----
!
!                  msc_type        850/32
!                  declare         on
!
!-----
!
! Declare constants and variables
!-----
!
! Axis related constants
!
FEED      equ      1
BUSY      equ      78          ! busy flag offset
DOWN      equ      77          ! down flag offset
FEED_BUSY equ      (FEED*16)+BUSY ! busy flag - FEED

```

```

FEED_DOWN      equ          (FEED*16)+DOWN      ! down flag - FEED

!
! I/O and Flags
!

RESET          equ          6                   ! Reset Fault (input)
SYS_OK         equ          7                   ! System okay (output)

TIMER          equ          72                  ! Timer used for delays

!
! Constants related to motion parameters
!

BITS_TURN      equ          4096                ! Bits per turn

DEF_SPEED      equ          10                  ! Default speed of 10 RPM
MIN_SPEED      equ          1                   ! Minimum speed of 1 RPM
MAX_SPEED      equ          2000               ! Maximum speed of 2000 RPM

DEF_ACDC       equ          100                 ! Default acdc of
!                                     100 Rev/Sec^2
MIN_ACDC       equ          5                  ! Minimum acdc of 5
!                                     Rev/Sec^2
MAX_ACDC       equ          800                ! Maximum acdc of
!                                     800 Rev/Sec^2

DEF_TURNS      equ          0                   ! Default turns of 0
MIN_TURNS      equ          -9999              ! Minimum of -99.99 turns
MAX_TURNS      equ          9999               ! Maximum of +99.99 turns

MIN_PART       equ          1                   ! Minimum part number of 1
MAX_PART       equ          9999              ! Maximum part number of
!                                     9999

!
! Constants related to the OPI
!

PORT           equ          2                   ! Defines the port number

MINUS_1        equ          -1
ZERO           equ          48                 ! ASCII equivalent of 0
F1             equ          65
F2             equ          66
F3             equ          67
F4             equ          68
NEXT           equ          78
PREV           equ          80
ENTER          equ          255

!
! Text strings used for display
!

```

MENU_1_1	text	"F1 SETUP<13><10>"
MENU_1_2	text	"F2 RUN<13><10>"
MENU_1_3	text	"F3 PROM OPTIONS<13><10>"
MENU_1_4	text	"F4 STOP PROG<13><10>"
MENU_2_1	text	"F1 LOAD<13><10>"
MENU_2_2	text	"F2 SAVE<13><10>"
MENU_2_3	text	"<13><10>"
MENU_2_4	text	"F4 MAIN MENU<13><10>"
MENU_3_1	text	"F1 ENTER PART<13><10>"
MENU_3_2	text	"<13><10>"
MENU_3_3	text	"<13><10>"
MENU_3_4	text	"F4 EXIT<13><10>"
MENU_4_1	text	"F1 LOAD PROGRAM<13><10>"
MENU_4_2	text	"F2 LOAD DATA<13><10>"
MENU_4_3	text	"<13><10>"
MENU_4_4	text	"F4 MAIN MENU<13><10>"
MENU_5_1	text	"F1 SAVE PROGRAM<13><10>"
MENU_5_2	text	"F2 SAVE DATA<13><10>"
MENU_5_3	text	"<13><10>"
MENU_5_4	text	"F4 MAIN MENU<13><10>"
SPEED_IN	text	"SPEED: "
ACDC_IN	text	"AC/DC: "
TURNS_IN	text	"TURN: "
PART_TXT	text	"PART: "
POSITIONING	text	"POSITIONING ..."
CONTINUE_1	text	"PRESS ANY KEY TO<13><10>"
CONTINUE_2	text	" CONTINUE "
STORED_TXT	text	"<27>ESTORED PART:"
LOADED_TXT	text	"<27>ELOADED PART:"
E2_TXT	text	"INVALID ENTRY"
E3_TXT	text	"ERR-NO FILE"
E4_TXT	text	"ERR-DUPLICATE"
E5_TXT	text	"ERR-WRITE FAIL"
E6_TXT	text	"ERR-EPROM FULL"
E7_TXT	text	"ERR-NOT OPEN"
E9_TXT	text	"ERR-PART NUM"
E10_TXT	text	"ERR-PROM READ"
E11_TXT	text	"ERR-PROM WRITE"
NO_MATCH	text	"ERR-NO MATCH"
ERR_1	text	"ENTRY:<13><10>"
ERR_2	text	" OUT OF RANGE<13><10>"
ERR_3	text	" TO<13><10>"
ERR_4	text	" PRESS ANY KEY "
FILE	text	"PARTxxxx"
PROG_FILE	text	"EPROM EXAMPLE"
VOL_NAME	text	"PROM"

CRLF	text	"<13><10>"
NULL	text	" "
CURSOR_POS	text	"<27>I "
KEYBOARD	text	"<27>Mu147.2580369-<08>NP<13>ABCD"
KEYSET	text	"<27>ND"
BACKLIGHT	text	"<27>V"
C_ON	text	"<27>wC"
C_OFF	text	"<27>wA"
C_HOME	text	"<27>H"
C_DOWN	text	"<27>B"
C_RIGHT	text	"<27>C"
CLR_SCREEN	text	"<27>E<27>H"
CLR_TO_END	text	"<27>J"
!		
! Variable definitions		
!		
temp	integer	! Used as a temporary storage location.
speed	integer	! Defines the speed to be used for motion.
acdc	integer	! Defines the acdc to be used for motion.
turns	integer	! Defines the distance to move in turns.
key	integer	! Stores the value of the key from the OPI
error	integer	! Stores the error value for prom
!		operations.
row	integer	! Defines the row of the cursor position.
col	integer	! Defines the column of the cursor
!		position.
whole_turn	integer	! Used to store the number of whole turns.
turns	integer	! The original value for number of turns.
turn_bits	integer	! The equivalent turn value in bits.
frac_turn	integer	! Used to store the fractional part of
!		turns.
r_value	integer	! Used to keep track of the roundoff value.
result	integer	! Used as a temporary storage location.
new_result	integer	! Used as a temporary storage location.
diff	integer	! Used to store the difference between
!		variables.
time	integer	! Determines the delay time for a timer.
time_1	integer	! Determines the delay time for a timer
pq_space	integer	! The amount of print queue space
!		available.
ctr	integer	! Used as a loop counter.
part	integer	! Defines the part number.
new_part	integer	! Defines the new part number.
i	integer	! Used as a temporary storage location.
f_ptr	integer	! Used as a pointer into an array.
dig	integer	! Used in the conversion of the part
!		number.
dig_pos	integer	! Used in the conversion of the part
!		number.
space	integer	! The amount of space left on the eprom.
part_save	integer	! Used in the conversion of the part
!		number.
fstat	integer	! Stores the file status for some prom
!		operations.

```

str          integer      ! Stores the value of a text string

temp_str     dim          20    ! Array to store a text string

!-----
!
! Initialize variables
!
!-----

Defaults     let          speed=DEF_SPEED
              let          acdc=DEF_ACDC
              let          turns=DEF_TURNS

!-----
!
! Program Setup
!   Setup the OPI.
!
!-----

Setup        gosub        Opi_Setup

!-----
!
! Main body of the program.
!   Enable FEED.
!   Display Main Menu.
!   Wait for a key to be pressed and act on the key.
!
!-----

Main         turn_on      SYS_OK
              drive_on    FEED
              set_local   FEED

Main_Menu    print       CLR_SCREEN
              print       MENU_1_1
              print       MENU_1_2
              print       MENU_1_3
              print       MENU_1_4

Main_Key     gosub        Ret_Fkey
              if          key=F1,Set_Speed
              if          key=F2,Run
              if          key=F3,Prom_Options
              if          key=F4,Exit
              goto        Main_Key

!-----
!
! Routine to input the speed.
!   Display the current speed.
!   Enter new speed, check if within range.
!   If out of range display error and allowable range.
!
!-----

```

```

!   Save the new value.
!
!-----
Set_Speed      print      CLR_SCREEN
                print      SPEED_IN
                print_num   4,0,speed
                print      CRLF
                let         temp=speed

Speed_Wt       input      NULL,7,0,temp,ENTER
                if_flag_off ENTER,Speed_Wt
                if         temp>MAX_SPEED,Speed_Err
                if         temp<MIN_SPEED,Speed_Err
                let         speed=temp
                goto        Set_Acdc

Speed_Err      gosub      Err_Screen
                let         row=0
                let         col=9
                gosub      Cursor
                print_num   7,0,temp

                let         row=2
                let         col=0
                gosub      Cursor
                print_num   4,0,MIN_SPEED

                let         row=2
                let         col=12
                gosub      Cursor
                print_num   4,0,MAX_SPEED
                gosub      Press_Key
                goto        Set_Speed

!-----
!
! Routine to input the acdc.
! Display the current acdc.
! Enter new acdc, check if within range.
!   If out of range display error and allowable range.
!   Save the new value.
!
!-----

Set_Acdc       print      CLR_SCREEN
                print      ACDC_IN
                print_num   4,0,acdc
                print      CRLF
                let         temp=acdc

Acdc_Wt        input      NULL,7,0,temp,ENTER
                if_flag_off ENTER,Acdc_Wt
                if         temp>MAX_ACDC,Acdc_Err
                if         temp<MIN_ACDC,Acdc_Err

```

```

        let          acdc=temp
        goto         Set_Turns

Acdc_Err      gosub      Err_Screen
               let       row=0
               let       col=9
               gosub     Cursor
               print_num 7,0,temp

               let       row=2
               let       col=0
               gosub     Cursor
               print_num 3,0,MIN_ACDC

               let       row=2
               let       col=13
               gosub     Cursor
               print_num 3,0,MAX_ACDC
               gosub     Press_Key
               goto      Set_Acdc

!-----
!
! Routine to input the number of turns.
! Display the current number of turns.
! Enter new number of turns, check if within range.
! If out of range display error and allowable range.
! Save the new value.
!-----

Set_Turns     print      CLR_SCREEN
               print     TURNS_IN
               print_num 8,2,turns
               print     CRLF
               let       temp=turns

Turns_Wt      input     NULL,8,2,temp,ENTER
               if_flag_off ENTER,Turns_Wt
               if       temp>MAX_TURNS,Turns_Err
               if       temp<MIN_TURNS,Turns_Err
               let       turns=temp
               goto      Main_Menu

Turns_Err     gosub      Err_Screen
               let       row=0
               let       col=9
               gosub     Cursor
               print_num 7,2,temp

               let       row=2
               let       col=0
               gosub     Cursor
               print_num 6,2,MIN_TURNS

               let       row=2

```

```

        let          col=10
        gosub        Cursor
        print_num    6,2,MAX_TURNS
        gosub        Press_Key
        goto         Set_Turns

!-----
!
! Routine to position the FEED to the current turns value.
! Define axis motion parameters.
! Convert the turns value to bits.
! Position the FEED axis.
! Wait for motion to finish.
! Return to Main Menu.
!-----

Run      print      CLR_SCREEN
        print      POSITIONING
        set_speed   FEED,speed
        set_ac_dc   FEED,acdc
        gosub      Turns_To_Bits

Run_Busy position    FEED,turn_bits
        if_stat_on  FEED_DOWN,Fault
        if_stat_on  FEED_BUSY,Run_Busy
        goto        Main_Menu

!-----
!
! Subroutine to convert the turns value to bits.
! Mathematical conversion and roundoff.
!-----

Turns_To_Bits let      turn_bits=0
              if      turns=0,Ttb_End
              let      whole_turn=turns/100
              let      turn_bits=whole_turn*BITS_TURN
              let      whole_turn=whole_turn*100
              let      frac_turn=turns-whole_turn
              let      r_value=frac_turn*BITS_TURN
              gosub    Round      ! intentionally 'round' twice
              gosub    Round      ! due to turns to hundredths
              let      turn_bits=turn_bits+r_value
Ttb_End      return_sub

!-----
!
! Subroutine to roundoff.
!-----

Round      let      result=r_value/10
          let      new_result=result*10
          let      diff=r_value-new_result

```



```

        let          r_value=result
        if          diff<5,End_Round
        let          r_value=result+1
End_Round    return_sub

```

```

!-----
!
! Routine used to stop program execution.
!   Stop all motion.
!   Disable FEED.
!-----

```

```

Exit          f_decel      FEED
              let          time=200
              gosub        Pause
              turn_off     SYS_OK
              drive_off    FEED
              master       FEED
!
!
! This command is used on the MSC800 in place of 'drive_off'.
              print        CLR_SCREEN
              sys_return

```

```

!-----
!
! Routine used for a FEED fault condition.
!   Turn off the System Ok output.
!   Check to see that all motion has stopped.
!   Wait for reset input, then restart at Main.
!-----

```

```

Fault          no_op
              turn_off     SYS_OK
              f_decel      FEED
Wait_Busy1     if_stat_on  FEED_BUSY,Wait_Busy1

Start_Over     if_io_off   RESET,Start_Over
              restart_at   Main

```

```

!-----
!
! Subroutine to initialize the OPI.
!   Setup the communications port.
!   Transmit setup parameters.
!   Check print queue space, and return.
!-----

```

```

Opi_Setup      port_set    2,9600,8
              let          time_1=200
              gosub        Pause
              print         KEYBOARD
              print         KEYSET

```

```

        print          BACKLIGHT
        get_pq_space   pq_space
        if             pq_space>1000,Exit_Opi_Setup
        let           time_1=10
        gosub          Pause

Exit_Opi_Setup return_sub

!-----
!
! Subroutine to return the selected key.
! Clear out the input buffer.
! Wait for a key to be pressed.
! Check if key is valid, and repeat if not valid or return.
!
!-----

Ret_Fkey      if_no_char   PORT,Cont_Fkey
Clr_Key       input        NULL,0,0,key,ENTER
Wait_Clr      if_flag_off  ENTER,Wait_Clr
              gosub        Pause
              if_char       PORT,Clr_Key

Cont_Fkey     print        C_OFF

              input        NULL,0,0,key,ENTER
Rfk_Wait      if_flag_off  ENTER,Rfk_Wait

              if           key=F1,Rfk_Ok
              if           key=F2,Rfk_Ok
              if           key=F3,Rfk_Ok
              if           key=F4,Rfk_Ok
              if           key=NEXT,Rfk_Ok
              if           key=PREV,Rfk_Ok
              if           key=ENTER,Rfk_Ok
              goto         Ret_Fkey

Rfk_Ok        print        C_ON
              return_sub

!-----
!
! Subroutine to wait for any key.
! Stop on active inputs.
! Wait for a key to be pressed, and return.
!
!-----

Press_Key     stop_input

              input        NULL,0,0,key,ENTER
Pk_Wait       if_flag_off  ENTER,Pk_Wait
              return_sub

```

```

!-----
!
! Subroutine to position cursor.
!   Move cursor to HOME position.
!   Position by column, then by row and return.
!-----

Cursor          print          C_HOME

C_Next_Col      let           ctr=0
                  if           ctr=col,C_Row
                  print        C_RIGHT
                  let          ctr=ctr+1
                  goto         C_Next_Col

C_Row           let           ctr=0
C_Next_Row      if           ctr=row,C_Done
                  print        C_DOWN
                  let          ctr=ctr+1
                  goto         C_Next_Row
C_Done          return_sub

!-----
!
! Subroutine to display error screen.
!-----

Err_Screen      print          CLR_SCREEN
                  print        ERR_1
                  print        ERR_2
                  print        ERR_3
                  print        ERR_4
                  return_sub

!-----
!
! Subroutine to create a time delay.
!   Wait for timer to time out, and return.
!-----

Pause           set_tmr        TIMER,time_1
Pausing         if_tmr_on      TIMER,Pausing
                  return_sub

!-----
!
! Routine to select prom options.
!   Display prom option menu.
!   Wait for a key to be pressed and act on the key.
!-----

Prom_Options    print          CLR_SCREEN
                  print        MENU_2_1

```

```

        print          MENU_2_2
        print          MENU_2_3
        print          MENU_2_4

Po_Key      gosub      Ret_Fkey
            if          key=F1,Load_Prom
            if          key=F2,Save_Prom
            if          key=F4,Main_Menu
            goto        Po_Key

!-----
!
! Routine to load data or program from eprom.
! Check volume name. If there is an error, display it then return
! to prom options menu.
! Display load prom menu.
! Wait for a key to be pressed and act on the key.
!
!-----

Load_Prom   get_volume  1,temp_str,error
            if          error=0,Load_Start
            gosub      Display_Err
            goto        Prom_Options

Load_Start  print      CLR_SCREEN
            print      MENU_4_1
            print      MENU_4_2
            print      MENU_4_3
            print      MENU_4_4

Lp_Key      gosub      Ret_Fkey
            if          key=F1,Load_Program
            if          key=F2,Ld_Data
            if          key=F4,Prom_Options
            goto        Lp_Key

!-----
!
! Routine to load data from eprom.
! Display load data menu.
! Wait for a key to be pressed and act on the key.
!
!-----

Ld_Data     print      CLR_SCREEN
            print      MENU_3_1
            print      MENU_3_2
            print      MENU_3_3
            print      MENU_3_4

Ld_Key      gosub      Ret_Fkey
            if          key=F1,Ld_Enter
            if          key=F4,Prom_Options
            goto        Ld_Key

```

```

!-----
!
! Routine to load data based on part entry.
!   Enter the part to be loaded, check if within range.
!   If out of range display error and allowable range.
!   Read eprom for a part match.
!   If read returns a nonzero error display it and return to prom options-
!   Otherwise display data loaded and return to Main Menu.
!-----

```

```

Ld_Enter      print      CLR_SCREEN
              print      PART_TXT
              print_num   4,0,part
              print      CRLF

              let         new_part=part
Ep_Wt_Part    input      NULL,5,0,new_part,ENTER
              if_flag_off ENTER,Ep_Wt_Part

              if          new_part>MAX_PART,Ep_Error
              if          new_part<MIN_PART,Ep_Error
              goto        Ep_No_Error

Ep_Error      let         error=9
              gosub       Display_Err
              goto        Prom_Options

Ep_No_Error   let         part=new_part
              gosub       Read_Prom
              if          error=0,Ep_No_Promerr
              gosub       Display_Err
              goto        Prom_Options

Ep_No_Promerr print      LOADED_TXT
              print_num   4,0,part
              print      CRLF
              print      CRLF
              print      CONTINUE_1
              print      CONTINUE_2
              gosub       Press_Key
              goto        Main_Menu

```

```

!-----
!
! Routine to load program from eprom.
!   Load the program into memory.
!   If load returns a nonzero error display it and return to prom options-
!   Otherwise program will restart from beginning of new program.
!-----

```

```

Load_Program print      CLR_SCREEN
              load        2,PROG_FILE,error
              if          error=0,Ep_No_Promerr
              gosub       Display_Err

```

```

                                goto          Prom_Options

!-----
!
! Routine to save data or program to eprom.
!   Check volume name. If there is an error, display it then return
!       to prom options menu.
!   Or if a volume has not been defined, define it now.
!   Display save prom menu.
!   Wait for a key to be pressed and act on the key.
!-----

Save_Prom      get_volume      1,temp_str,error
                if              error=0,Check_Init
                gosub           Display_Err
                goto            Prom_Options

Check_Init     let              str=VOL_NAME
                if              temp_str=str,Save_Prom2
Init_Prom      initialize      1,VOL_NAME,error
                if              error=0,Save_Prom2
                gosub           Display_Err
                goto            Prom_Options

Save_Prom2     print           CLR_SCREEN
                print           MENU_5_1
                print           MENU_5_2
                print           MENU_5_3
                print           MENU_5_4

Save_Prom_Key  gosub           Ret_Fkey
                if              key=F1,Save_Program
                if              key=F2,Save_Data
                if              key=F4,Prom_Options
                goto            Save_Prom_Key

!-----
!
! Routine to save data to eprom.
!   Display save data menu.
!   Wait for a key to be pressed and act on the key.
!-----

Save_Data      print           CLR_SCREEN
                print           MENU_3_1
                print           MENU_3_2
                print           MENU_3_3
                print           MENU_3_4

Sp_Key         gosub           Ret_Fkey
                if              key=F1,Sp_Enter
                if              key=F4,Prom_Options
                goto            Sp_Key

```

```

!-----
!
! Routine to save data to eprom based on part entry.
!   Enter the part to be saved, check if within range.
!   If out of range display error and return to save prom menu.
!   Check for duplicate part number.
!   If duplicate display error and return to save prom menu.
!   Write part data to the eprom and check for errors.
!   If write is successful return to prom options menu.
!   Otherwise display error before returning to prom options menu.
!-----

```

```

Sp_Enter      print      CLR_SCREEN
              print      PART_TXT
              let        part=0
              print_num   4,0,part
              print      CRLF

              let        new_part=part
Sp_Wt_Part    input      NULL,5,0,new_part,ENTER
              if_flag_off ENTER,Sp_Wt_Part

              if         new_part>MAX_PART,Sp_Error
              if         new_part<MIN_PART,Sp_Error

Sp_No_Error_1 let        part=new_part
              gosub      Dupl_Prom
              if         error=0,Sp_No_Error_2
              gosub      Display_Err
              goto        Save_Prom2

Sp_No_Error_2 gosub      Write_Prom
              if         error=0,Prom_Options
              gosub      Display_Err
              goto        Prom_Options

Sp_Error      let        error=9
              gosub      Display_Err
              goto        Save_Prom2

```

```

!-----
!
! Routine to save program to eprom.
!   Save the program to eprom.
!   If save is successful return to prom options.
!   Otherwise display error before returning to prom options menu.
!-----

```

```

Save_Program  save      2,PROG_FILE,error
              if        error=0,Prom_Options
              gosub      Display_Err
              goto        Prom_Options

```

```

!-----
!
! Subroutine to check for duplicate part name.
! Create name based on part number entry.
! Open file on eprom and check for duplicate filename.
! Save results to error and return.
!-----

Dupl_Prom      let          error=0
               gosub       Make_Name
               open        3,FILE,fstat
               if          fstat=4,Dp_Dupl      ! duplicate
               if          fstat=3,Dp_Done      ! empty

Dp_No_Prom     let          error=10           ! prom read error
               goto        Dp_Done

Dp_Dupl        let          error=4
               goto        Dp_Done

Dp_Done        close       3,fstat
               return_sub

!-----
!
! Subroutine to make part name.
! Imbed the part number into the text string 'FILE'.
!-----

Make_Name      let          part_save=part
               let          f_ptr=4
               let          dig_pos=1000

Mn_Loop        let_byte    FILE[f_ptr]=ZERO
               if          part_save<dig_pos,Mn_Dec
               let          dig=part_save/dig_pos
               let          i=dig*dig_pos
               let          part_save=part_save-i
               let          dig=dig+ZERO
               let_byte    FILE[f_ptr]=dig
Mn_Dec         let          f_ptr=f_ptr+1
               let          dig_pos=dig_pos/10
               if          f_ptr<7,Mn_Loop

               let          dig=part_save+ZERO
               let_byte    FILE[f_ptr]=dig
               return_sub

!-----
!
! Subroutine to read part from eprom.
! Create name based on part number entry.
! Open file on eprom and read information.
! If any errors occur close file and return.
!-----

```



```

Read_Prom      let      error=0
               gosub    Make_Name

               open     3,FILE,fstat
               if       fstat<>0,Rp_Err_1

               read     3,speed,4,fstat
               if       fstat<>0,Rp_Err_1
               read     3,acdc,4,fstat
               if       fstat<>0,Rp_Err_1
               read     3,turns,4,fstat
               if       fstat<>0,Rp_Err_1

               goto     Rp_Close

Rp_Err_1       let      error=10
               goto     Rp_Close

Rp_Close       close    3,fstat

Rp_Done        return_sub

```

```

!-----
!
! Subroutine to write part to eprom.
! Check the amount of empty space left on eprom.
!   If not enough display error and return.
! Create name based on part number entry.
! Open file on eprom and write information.
! If any errors occur close file and return.
!   Otherwise display part saved then wait for key before returning.
!-----

```

```

Write_Prom     let      error=0

               get_space 3,space,error
               if       error=0,Check_Space
               gosub    Display_Err
               goto     Wp_Done

Check_Space    if       space>12,Write_Cont
               let      error=6
               gosub    Display_Err
               goto     Wp_Done

Write_Cont     gosub    Make_Name
               open     3,FILE,fstat
               if       fstat=3,Wp_Empty
               close    3,fstat
               goto     Wp_Err

Wp_Empty       close    3,fstat
               create   3,FILE,fstat

```

```

        if                fstat<>0,Wp_Err

        write             3,speed,4,fstat
        if                fstat<>0,Wp_Err
        write             3,acdc,4,fstat
        if                fstat<>0,Wp_Err
        write             3,turns,4,fstat
        if                fstat<>0,Wp_Err

Wp_Close      close       3,fstat
              if          fstat<>0,Wp_Err

              print       CLR_SCREEN
              print       STORED_TXT
              print_num    4,0,part
              print       CRLF
              print       CRLF
              print       CONTINUE_1
              print       CONTINUE_2
              gosub       Press_Key
              goto        Wp_Done

Wp_Err        let         error=11
              gosub       Display_Err

Wp_Done       return_sub

```

```

!-----
!
! Subroutine to display error.
! Use a select structure to determine the error to be displayed.
! Print the error message and return.
!
!-----

```

```

Display_Err   print       CLR_SCREEN
              select      error
                case      2
                  print    E2_TXT
                  exit_select
                case      3
                  print    E3_TXT
                  exit_select
                case      4
                  print    E4_TXT
                  exit_select
                case      5
                  print    E5_TXT
                  exit_select
                case      6
                  print    E6_TXT
                  exit_select
                case      7
                  print    E7_TXT
                  exit_select

```

```
        case          9
            print      E9_TXT
            exit_select
        case          10
            print      E10_TXT
            exit_select
        default
            print      NO_MATCH
            exit_select
    end_select
    let              time_1=200
    gosub            Pause
    return_sub
```

Home Sequence

```

! -----
! TITLE:          Home Sequence
!
! DESCRIPTION: This program illustrates a homing sequence for an encoder
!              based system. The program takes into account two
!              different marker types (single pulse or 180 degrees)
!              which are commonly used. After the homing sequence has
!              been initiated, the operator can execute either an index
!              or a positional move.
!
! EQUIPMENT:      MSC250
!
! COMMANDS: disable_swi      find_mrk_ccw      find_mrk_cw
!              find_tm_ccw    find_tm_cw        get_t_mark
!              index          jog_ccw          jog_cw
!              position       set_ac_dc         set_gl_ccw
!              set_gl_cw      set_speed         set_swi_mask
!              set_com_limit  set_nar_ang       set_wide_ang
!
! The above commands are commonly used in many programs. This
! illustration provides the context and syntax required to successfully
! integrate these commands into a complete program. While the use of
! these commands varies from program to program, the following example
! will give the programmer an indication of a typical usage.
! -----
!
!              msc_type      250
!              declare       on
!
! -----
!
! Declare constants and variables
! -----
!
! Axis related constants
!
XFR          equ          1          ! Transfer axis
BUSY         equ          78         ! busy flag offset
DOWN         equ          77         ! down flag offset
XFR_BUSY     equ          (XFR*16)+BUSY ! busy flag - Transfer
XFR_DOWN     equ          (XFR*16)+DOWN ! down flag - Transfer
!
! I/O and Flags
!
INIT         equ          0          ! Initialize axis (input)
INDEX        equ          1          ! Start index (input)
POSITION     equ          2          ! Position to zero (input)
DIR          equ          3          ! Direction:ON=CCW;OFF=CW

```

```

!
MARK_TYPE      equ      4      ! (input)
!              ! Type of marker:ON=180;
!              ! OFF=pulse (input)
ZREV           equ      5      ! Zero rev switch (input)
RESET          equ      6      ! Reset Fault (input)
SYS_OK         equ      7      ! System okay (output)

TIMER          equ      72     ! Timer used for delays

INIT_CMPT      equ      255    ! Flag for initialization
!              ! complete

!
! Motion related constants
!

INIT_SPD       equ      200    ! Initialize jog speed
INIT_ACDC      equ      200    ! Initialize accel/decel
!              ! rate
IDX_SPD        equ      800    ! Index speed
IDX_ACDC       equ      300    ! Index accel/decel rate
POS_SPD        equ      300    ! Position speed
POS_ACDC       equ      200    ! Position accel/decel rate
IDX_DIST       equ      40960  ! Index distance = 10 turns
COM_LIMIT      equ      3000   ! To limit error b/n actual and

NAR_ANG        equ      400    ! Following error window = 400
WIDE_ANG       equ      1024   ! Following error window = 1024
HOME           equ      0      ! Absolute home

!
! Variable definitions
!

mrk_state      integer      ! Return variable for 'get_t_mark'.
mrk_state2     integer      ! Another return variable for
!              ! 'get_t_mark'.
time           integer      ! Determines the delay time for a timer.
swi_mask       integer      ! Determines the active software
!              ! interrupts.

!-----
!
! Initialize variables
!
!-----

Defaults      let      time=100
              let      swi_mask=8

!-----
!
! Program Setup
!-----

```

```

! Define software interrupts. -
! -
!-----
Setup      swi_if_on      0,INIT,Home_Xfr
           swi_if_on      1,INDEX,Index_Xfr
           swi_if_on      2,POSITION,Position_Xfr
           swi_if_on      3,XFR_DOWN,Fault
           enable_swi

!-----
!
! Main body of program. -
! Enable XFR. -
! Remain in Loop, waiting for software interrupts. -
! -
!-----

Main      clr_flag      INIT_CMPT
           turn_on      SYS_OK
           drive_on     XFR

Loop      goto          Loop

!-----
!
! Software Interrupt Routine to Home XFR axis. -
! Setup software interrupt mask to only allow XFR_DOWN to be active. -
! Define XFR motion parameters. -
! Test to determine direction of home sequence. -
! Test to determine type of marker to find. -
! Find appropriate marker in the desired direction. -
! Test ZREV switch for current state. -
! If on the switch, must jog off switch to start. -
! Jog toward the ZREV switch and monitor the state of the IO. -
! When ZREV turns on, stop and set a global zero according to the -
! HOME direction. -
! If MARK_TYPE=1 then test the marker by getting the state 180 -
! degrees apart. If the marker does not change state then stop system. -
! Positon the XFR to HOME. -
! Set the flag INIT_CMPT. -
! Enable software interrupts and return. -
! -
!-----

Home_Xfr   let          swi_mask=8
           set_swi_mask swi_mask
           enable_swi

           set_speed     XFR,INIT_SPD
           set_ac_dc     XFR,INIT_ACDC
           if_io_on      DIR,Home_Xfr_Ccw

Home_Xfr_Cw if_io_off    MARK_TYPE,Find_Mark_Cw
           find_tm_cw    XFR,4096
           gosub         Wait_Stop

```

	goto	Find_Home_Cw
Find_Mark_Cw	find_mrk_cw gosub	XFR,4096 Wait_Stop
Find_Home_Cw	if_io_off jog_ccw	ZREV,Off_Switch_Cw XFR
On_Switch_Cw	if_stat_on if_io_on f_decel gosub	XFR_DOWN,Fault ZREV,On_Switch_Cw XFR Wait_Stop
Off_Switch_Cw	jog_cw if_stat_on if_io_off set_gl_cw f_decel gosub goto	XFR XFR_DOWN,Fault ZREV,Off_Switch_Cw XFR XFR Wait_Stop Exit_Home
Home_Xfr_Ccw	if_io_off find_tm_ccw gosub goto	MARK_TYPE,Find_Mark_Ccw XFR,4096 Wait_Stop Find_Home_Ccw
Find_Mark_Ccw	find_mrk_ccw gosub	XFR,4096 Wait_Stop
Find_Home_Ccw	if_io_off jog_cw	ZREV,Off_Switch_Ccw XFR
On_Switch_Ccw	if_stat_on if_io_on f_decel gosub	XFR_DOWN,Fault ZREV,On_Switch_Ccw XFR Wait_Stop
Off_Switch_Ccw	jog_ccw if_stat_on if_io_off set_gl_ccw f_decel gosub	XFR XFR_DOWN,Fault ZREV,Off_Switch_Ccw XFR XFR Wait_Stop
Exit_Home	if_io_off get_t_mark index gosub get_t_mark if sys_fault	MARK_TYPE,Xfr_At_Home XFR,mrk_state XFR,2048 Wait_Stop XFR,mrk_state2 mrk_state<>mrk_state2,Xfr_At_Home
Xfr_At_Home	position gosub	XFR,HOME Wait_Stop

```

Exit_Home_Xfr  set_flag      INIT_CMPT
                disable_swi
                let          swi_mask=15
                set_swi_mask swi_mask
                enable_swi
                return_sub

!-----
!
! Software Interrupt Routine to Index the XFR 10 turns.
! Setup software interrupt mask to only allow XFR_DOWN to be active.
! Check to see that INIT_CMPT has been set.
!   If not skip index.
! Define axis motion parameters.
! Index XFR 10 turns.
! Wait for index to complete.
! Enable software interrupts
!
! NOTE: Unless torque limit is exceeded, no change will be seen in the
!       index, using set_com_limit, set_nar_ang or set_wide_ang
!       instructions.
!-----

Index_Xfr      let          swi_mask=8
                set_swi_mask swi_mask
                enable_swi

                if_flag_off  INIT_CMPT,Exit_Index
                set_com_limit XFR,COM_LIMIT
                set_nar_ang   XFR,NAR_ANG
                set_wide_ang  XFR,WIDE_ANG
                set_speed     XFR,IDX_SPD
                set_ac_dc     XFR,IDX_ACDC
                index         XFR,IDX_DIST
                gosub         Wait_Stop

Exit_Index     disable_swi
                let          swi_mask=15
                set_swi_mask swi_mask
                enable_swi
                return_sub

!-----
!
! Software Interrupt Routine to Position XFR to HOME.
! Setup software interrupt mask to only allow XFR_DOWN to be active.
! Check to see that INIT_CMPT has been set.
!   If not skip position.
! Define axis motion parameters.
! Position XFR to HOME.
! Wait for index to complete.
! Enable software interrupts
!
!-----

```



```

Position_Xfr    let          swi_mask=8
                set_swi_mask swi_mask
                enable_swi

                if_flag_off  INIT_CMPT,Exit_Position
                set_speed    XFR,POS_SPD
                set_ac_dc    XFR,POS_ACDC
                position     XFR,HOMEB
                gosub        Wait_Stop

Exit_Position   disable_swi
                let          swi_mask=15
                set_swi_mask swi_mask
                enable_swi
                return_sub

!-----
!
! Subroutine to wait for motion to be completed.
!
!-----

Wait_Stop      if_stat_on    XFR_BUSY,Wait_Stop  ! check for axis busy
                return_sub

!-----
!
! Software Interrupt Routine used for a XFR fault condition.
! Turn off the System Ok output.
! Check to see that all motion has stopped.
! Wait for reset input, then restart at Main.
!
!-----

Fault          no_op
                turn_off    SYS_OK
                f_decel     XFR
Wait_Busy1     if_stat_on    XFR_BUSY,Wait_Busy1

Start_Over     if_io_off    RESET,Start_Over
                let          swi_mask=15
                set_swi_mask swi_mask
                enable_swi
                restart_at   Main

```

Index with over_draw

```

! -----
! TITLE:      Index with over_draw
!
! DESCRIPTION: This program illustrates the two overdraw modes and the
!              differences between them.  The operator can observe
!              these differences by manually selecting the overdraw
!              mode and then triggering the sensor input during the
!              execution of the index.
!
! EQUIPMENT:   MSC250
!
! COMMANDS:  clr_local      disable_hwi      enable_hwi
!             get_com        get_pos          get_trap_pos
!             if_io_on       if_stat_off      over_draw
!             set_local      set_ovd_mode
!
! The above commands are commonly used in many programs.  This
! illustration provides the context and syntax required to successfully
! integrate these commands into a complete program.  While the use of
! these commands varies from program to program, the following example
! will give the programmer an indication of a typical usage.
! -----
!
! msc_type      250
! declare       ON
!
! -----
!
! Declare constants and variables
! -----
!
! Axis related constants
!
FEED          equ          1
!
BUSY          equ          78      ! busy flag offset
DOWN         equ          77      ! down flag offset
ARMED        equ          75      ! HWI armed flag
!
INDEXING      equ          73      ! indexing flag offset
FEED_BUSY     equ          (FEED*16)+BUSY  ! busy flag - FEED
FEED_DOWN     equ          (FEED*16)+DOWN  ! down flag - FEED
FEED_ARMED    equ          (FEED*16)+ARMED  ! HWI armed flag -
!                                     FEED
FEED_INDEXING equ          (FEED*16)+INDEXING ! indexing flag -
!                                     FEED
!
OD_MODE_0     equ          0      ! Overdraw mode - sensor always
!                                     active
OD_MODE_1     equ          1      ! Overdraw mode - sensor only
!                                     active during porch

```

```

!
! I/O and Flags
!

SENSOR      equ      0      ! Photo sensor (input)
INDEX       equ      1      ! Start index (input)
OVD_MODE    equ      2      ! Select overdraw mode
!                      (input)
RESET       equ      6      ! Reset Fault (input)
SYS_OK      equ      7      ! System okay (output)

TIMER       equ      72     ! Timer used for delays.

!
! Variable definitions
!

com          integer      ! Stores the current 'get_com' value.
pos          integer      ! Stores the current 'get_pos' value.
tpos         integer      ! Stores the current 'get_trap_pos' value.
diff         integer      ! Variable used to measure differences.
end_com      integer      ! Used to store the ending com value.
end_pos      integer      ! Used to store the ending pos value.
temp         integer      ! Used as a temporary storage location.
idx_dist     integer      ! The index distance.
ovd_speed    integer      ! The speed at which the search occurs.
srch_dist    integer      ! The distance to search for sensor.
post_dist    integer      ! The distance travelled after the sensor
!                      trips.

!-----
!
! Initialize variables
!
!-----

Defaults      let      ovd_speed=25
               let      srch_dist=8192
               let      post_dist=1024
               let      idx_dist=40960

!-----
!
! Program Setup
! Define FEED motion parameters.
!
!-----

Setup          set_ac_dc      FEED,50
               set_speed     FEED,200

!-----
!
! Main body of program
! Enable DRAW.
! Remain in Loop, waiting for transition on INDEX.

```

```

!
!-----
Main          drive_on      FEED

Loop          if_io_on      INDEX,Loop
Loop1         if_io_off     INDEX,Loop1
              if_io_on      OVD_MODE,Mode_1

!-----
!
! Routine for overdraw mode 0 (HWI is always active).
!   Clear any local zero.
!   Check status of HWI armed flag.
!   If flag on disarm HWI, otherwise continue.
!   Setup the HWI for an overdraw of mode 0.
!   Set a local zero in order to have an accurate start point.
!   Make sure the FEED HWI is armed.
!   Index the FEED and wait until motion is complete.
!   Based on the status of the FEED_ARMED flag, determine if the
!   distances are correct.
!-----

Mode_0        clr_local      FEED
              if_stat_off    FEED_ARMED,No_Disable
              disable_hwi    FEED

No_Disable    set_ovd_mode   FEED,OD_MODE_0
              set_local      FEED

              enable_hwi
              over_draw      FEED,ovd_speed,srch_dist,post_dist

Not_Armed     if_stat_off    FEED_ARMED,Not_Armed

              index          FEED,idx_dist
              gosub           Wait_Stop

              if_stat_on     FEED_ARMED,Check_Dist
              get_com         FEED,end_com
              if              end_com>=idx_dist,Loop
              turn_off        SYS_OK
              sys_fault

Check_Dist    get_com        FEED,end_com
              let             temp=idx_dist+srch_dist
              if              end_com=temp,Loop
              turn_off        SYS_OK
              sys_fault

!-----
!
! Routine for overdraw mode 1 (HWI is only active in overdraw porch).
!   Clear any local zero.
!   Check status of HWI armed flag.

```

```

!   If flag on disarm HWI, otherwise continue.           -
!   Setup the HWI for an overdraw of mode 1.             -
!   Set a local zero in order to have an accurate start point. -
!   Make sure the FEED HWI is armed.                     -
!   Index the FEED and wait until motion is complete.    -
!   Based on the status of the FEED_ARMED flag, determine if the -
!   distances are correct.                                -
!                                                         -
!-----
Mode_1      clr_local      FEED
            if_stat_off    FEED_ARMED,No_Disable1
            disable_hwi    FEED

No_Disable1 set_ovd_mode    FEED,OD_MODE_1
            set_local      FEED
            get_pos        FEED,pos

            enable_hwi
            over_draw      FEED,25,8192,1024

Not_Armed1  if_stat_off    FEED_ARMED,Not_Armed1

            index          FEED,40960
            gosub          Wait_Stop

            if_stat_on     FEED_ARMED,Check_Dist1
            get_com        FEED,end_com
            get_trap_pos   FEED,tpos
            let            diff=end_com-tpos
            if             diff=1024,Loop
            turn_off      SYS_OK
            sys_return

Check_Dist1 get_pos        FEED,end_pos
            let            end_pos=end_pos-pos
            let            temp=idx_dist+srch_dist
            if             end_pos<>temp,Check_Dist1
            if             end_pos=temp,Loop
            turn_off      SYS_OK
            sys_return

!-----
!
! Routine used for a FEED fault condition.
!   Turn off the System Ok output.
!   Check to see that all motion has stopped.
!   Wait for reset input, then restart at Main.
!
!-----

Fault      no_op
            turn_off      SYS_OK
            f_decel       FEED

Wait_Busy1 if_stat_on     FEED_BUSY,Wait_Busy1

```

```
Start_Over      if_io_off      RESET,Start_Over
                  restart_at    Main
```

```
!-----
!  
! Routine used to wait for motion to stop on FEED.      -  
!   Check the DOWN and INDEXING flags, and when index is complete return.-  
!  
!-----
```

```
Wait_Stop      no_op  
                if_stat_on      FEED_DOWN,Fault  
                if_stat_on      FEED_INDEXING,Wait_Stop  
                return_sub
```

Programmable Limit Switch Functions

```

!-----
! TITLE:          Programmable Limit Switch Functions
!
! DESCRIPTION:    This program demonstrates the available Programmable
!                Limit Switch (PLS) functions.  These functions include
!                time angle advance, creating an angular offset, and
!                masking desired outputs.  Each of these functions can be
!                visually observed during program execution.
!
! EQUIPMENT:     MSC850
!                MCF850C in slot three.
!                HPL850 in slot four with a PLS850
!
! COMMANDS:      get_angle          get_pls_mask          get_pls_out
!                l_track_spd        preset                set_pls_ang
!                set_pls_cnt        set_pls_mask          set_pls_time
!                track_spd
!
! The above commands are commonly used in many programs.  This
! illustration provides the context and syntax required to successfully
! integrate these commands into a complete program.  While the use of
! these commands varies from program to program, the following example
! will give the programmer an indication of a typical usage.
!-----

                msc_type          850
                declare          on

!-----
!
! Declare constants and variables
!-----

!
! Axis related constants
!

PSEUDO          equ          3
PLS              equ          4

CALC_PLS        equ          138          ! Calculating flag on PLS

MABA_TALKER      equ          29          ! PSEUDO is talker on MABA
MABA_LISTEN      equ          20          ! PLS is listener on MABA
PSEUDO_MABA      equ          10          ! PSEUDO sends data to
!                                     MABA
PLS_MABA         equ          0           ! PLS receives data from
!                                     MABA
MAX_PLS          equ          16          ! 16 PLS outputs

!
! I/O and Flags
!

```

```

DO_MASK      equ      0      ! Define output mask
!              (input)
ADVANCE      equ      1      ! Define time advance
!              (input)
OFFSET      equ      2      ! Create an offset (input)
SYS_OK       equ      7      ! System okay (output)

TIMER        equ      72     ! Timer used for delays

FIRST_TIME   equ      251    ! Flag indicating first
!              time

```

```

!
!  Variable definitions
!

```

```

map          integer      ! Used to configure the MASTER/SLAVE on
!                          the MAB.
map_back     integer      ! Return variable for the current 'map'
!                          setup.
map_stat     integer      ! Return variable for the current 'map'
!                          status.
mcf          integer      ! Used to configure the PSEUDO.
mcf_back     integer      ! Return variable for the current 'mcf'
!                          setup.
hpl          integer      ! Used to configure the PLS.
on_angle     integer      ! The angle to turn on a module.
off_angle    integer      ! The angle to turn off a module.
mod_lo       integer      ! Used to define modules 0-7.
mod_hi       integer      ! Used to define modules 8-15.
out_stat     integer      ! Return variable for 'get_pls_out'.
angle        integer      ! Return variable for 'get_angle'.
time         integer      ! Determines the delay time for a timer.
mask         integer      ! Used to mask off specific pls modules.
mask_back    integer      ! Return variable for 'get_pls_mask'.
pls_time     integer      ! Determines the delay time for a pls
!                          output.
offset       integer      ! The amount of offset added to the MAB.

```

```

!-----
!
!  Initialize variables
!
!-----

```

```

Defaults     let      on_angle=0
              let      off_angle=4096
              let      mod_lo=0
              let      mod_hi=1
              let      pls_time=20

```

```

!-----
!
!  Program Setup
!  Define the PLS parameters.
!-----

```



```

! Define axis motion parameters. -
! Define software interrupts. -
! -
!-----
Setup      set_pls_cnt    PLS,32768
           gosub        Wait_Calc

           gosub        Set_Config

Define_Pls  set_pls_ang   PLS,on_angle,off_angle,mod_lo
           gosub        Wait_Calc

           set_pls_ang   PLS,on_angle,off_angle,mod_hi
           gosub        Wait_Calc

           let          on_angle=on_angle+4096
           let          off_angle=off_angle+4096
           let          mod_lo=mod_lo+2
           let          mod_hi=mod_hi+2
           if           mod_lo<MAX_PLS,Define_Pls

           set_ac_dc     PSEUDO,20

           swi_if_on     0,DO_MASK,Define_Mask
           swi_if_on     1,ADVANCE,Time_Advance
           swi_if_on     2,OFFSET,Create_Offset
           enable_swi

!-----
!
! Main body of program -
! Start motion. -
! Get current angle and PLS output information. -
! Remain in Loop, waiting for software interrupts. -
! -
!-----

Main      no_op
           turn_on       SYS_OK
           track_spd     PSEUDO,100
           l_track_spd   PSEUDO,100
!
!
! This command could be used in place of a 'track_spd' when slow
! or fractional speeds are required.

Loop      get_pls_out    PLS,out_stat
           get_angle     PLS,angle
           gosub        Delay
           goto         Loop

!-----
!
! Software Interrupt Routine used to define a PLS output mask. -

```

```

! Test to see if this is the first time executing this routine.      -
! Setup the PLS mask.                                              -
! Check for correct mask.                                          -
!   If mask is incorrect then stop system.                         -
! Enable the software interrupts and return.                       -
!                                                                    -
!-----

Define_Mask    no_op
               if_flag_on    FIRST_TIME,Skip_First
               let           mask=2147483646
               set_flag      FIRST_TIME

Skip_First     let           mask=mask{2
               set_pls_mask   PLS,mask
               gosub          Delay

               get_pls_mask   PLS,mask_back
               if             mask=mask_back,Exit_Mask
               turn_off       SYS_OK
               sys_return

Exit_Mask      enable_swi
               return_sub

!-----
!
! Software Interrupt Routine to define time angle advance.         -
! Define the amount of advance.                                    -
! Enable software interrupts and return.                           -
!                                                                    -
!-----

Time_Advance   no_op
               let           mod_lo=0
               set_pls_time   PLS,pls_time,mod_lo
               gosub          Wait_Calc

               enable_swi
               return_sub

!-----
!
! Software Interrupt Routine to create an offset on the MAB.       -
! Define the amount of offset.                                     -
! Enable software interrupts and return.                           -
!                                                                    -
!-----

Create_Offset   no_op
               preset        PLS,offset
               gosub          Wait_Calc

               enable_swi
               return_sub

```

```

!-----
!
! Subroutine to configure the MAB map and the MCF map.
! Set the appropriate bits, and check the results.
!-----

Set_Config      let      map=0
                 set_map  map
                 set_bit  MABA_TALKER,map
                 set_bit  MABA_LISTEN,map
                 set_map  map
                 get_map  map_back
                 get_map_stat map_stat

                 if      map_stat<>0,Config_Fail
                 if      map<>map_back,Config_Fail

Config_Pseudo   let      mcf=0
                 set_mcf  PSEUDO,mcf
                 set_bit  PSEUDO_MABA,mcf
                 set_mcf  PSEUDO,mcf

                 gosub    Delay
                 get_mcf  PSEUDO,mcf_back

                 if      mcf<>mcf_back,Config_Fail

Config_Pls      let      hpl=0
                 set_mcf  PLS,hpl
                 set_bit  PLS_MABA,hpl
                 set_mcf  PLS,hpl
                 gosub    Delay

Exit_Config     return_sub

Config_Fail     turn_off  SYS_OK
                 sys_return

!-----
!
! Subroutine to monitor the CALCULATING flag on the HPL card.
! Wait until the card is finished calculating.
!-----

Wait_Calc       no_op
                 if_stat_on  CALC_PLS,Wait_Calc
                 return_sub

!-----
!
! Subroutine used to create a time delay.
! Wait until the timer times out.
!-----

```

Delay	set_tmr	TIMER,time
Wait_Time	if_tmr_on	TIMER,Wait_Time
	return_sub	

Piecewise Profile

```

!-----
! TITLE:      Piecewise Profile
!
! DESCRIPTION: This program illustrates the use of a piecewise profile,
!              and the methods by which to execute a profile.  The
!              time to execute the selected profile is compared to the
!              calculated time as a measure of performance.
!
! EQUIPMENT:   MSC850
!              ACR850C or ACE850 in slot one.
!              ACR850C or ACE850 in slot eight.
!
! COMMANDS:   exec_profile      get_pstat      get_time
!              prep_profile     set_trig_pw
!
! The above commands are commonly used in many programs.  This
! illustration provides the context and syntax required to successfully
! integrate these commands into a complete program.  While the use of
! these commands varies from program to program, the following example
! will give the programmer an indication of a typical usage.
!-----

                msc_type      850
                declare      on

!-----
!
! Declare constants and variables
!-----

!
! Axis related constants
!

FEED      equ      1
MASTER    equ      8

BUSY      equ      78      ! busy flag offset
DOWN      equ      77      ! down flag offset
CALC      equ      74      ! calculating flag offset
LOCK      equ      79      ! lock flag offset
FEED_BUSY equ      (FEED*16)+BUSY ! busy flag - FEED
FEED_DOWN equ      (FEED*16)+DOWN ! down flag - FEED
FEED_CALC equ      (FEED*16)+CALC ! calculating flag - FEED
FEED_LOCK equ      (FEED*16)+LOCK ! lock flag - FEED

LK_MODE_3 equ      3      ! lock mode 9
MABA_TALKER equ      24    ! MASTER is talker on MABA
MABA_LISTEN equ      23    ! FEED is listener on MABA

!
! I/O and Flags
!

```

```

EXEC_PROF      equ      0      ! Start profile execution
!              (input)
EXEC_TYPE      equ      1      ! Execution type (input)
!              ON=Lock;OFF=Normal
RESET          equ      6      ! Reset Fault (input)
SYS_OK         equ      7      ! System okay (output)

TIMER          equ      72     ! Timer used for delays.

!
!  Variable Definitions
!

strt_angle     integer      ! The start angle for the Lock to begin.
temp           integer      ! Used as a temporary storage variable.
prof_stat      integer      ! Stores the result of a get_pstat.
diff          integer      ! Stores the difference between two
!                          variables.
error          integer      ! The error between measured and
!                          calculated time.
exec_time      integer      ! The calculated execution time.
time          integer      ! Determines the delay time for a timer.
strt_time      integer      ! Stores the time value at start of
!                          profile.
end_time       integer      ! Stores the time value at end of profile.
map            integer      ! Used to configure the MASTER/SLAVE on
!                          the MAB.
map_back       integer      ! Return variable for the current 'map'
!                          setup.
map_stat       integer      ! Return variable for the current 'map'
!                          status.

!
!  Data array for piecewise profile
!

prodata        begin_data
                data        500,250,8192
                data        1000,250,8192
                data        250,100,8192
                data        0,100,8192
                end_data

!-----
!
! Initialize variables
!
!-----

Defaults      let      strt_angle=1024
               let      exec_time=210

!-----
!
! Program Setup
!
!-----

```

```

!   Configure map.                                     -
!   Prepare piecewise profile.                         -
!   Define and enable software interrupts.             -
!                                                     -
!-----
Setup          no_op

               gosub          Set_Config
               gosub          Calc_Profile

               swi_if_on       0,FEED_DOWN,Fault
               enable_swi

!-----
!
! Main body of program                                -
!   Enable FEED.                                       -
!   Remain in Loop, waiting for transition on EXEC_PROF to execute -
!   profile.                                           -
!                                                     -
!-----

Main           turn_on         SYS_OK
               drive_on        FEED

Loop           if_io_on        EXEC_PROF,Loop
Loop1          if_io_off       EXEC_PROF,Loop1
               if_io_on        EXEC_TYPE,With_Lock

!-----
!
! Routine to execute a piecewise profile.             -
!   Execute the piecewise profile.                   -
!   Measure the time to complete the profile, and compare to calculated -
!   time. If not correct stop system.                 -
!   Otherwise return to Loop.                         -
!                                                     -
!-----

Profile        get_time        strt_time
               exec_profile     FEED
               gosub           Wait_Stop
               get_time         end_time
               let              diff=end_time-strt_time
               let              error=exec_time-diff
               let              temp=abs(error)
               if               temp<=2,Loop
               turn_off         SYS_OK
               sys_return

!-----
!
! Routine to execute piecewise profile using lock mode 3. -
!   Define the trigger angle and lock the FEED.       -
!   Wait for motion to stop, and return to Loop.      -
!                                                     -
!-----

```

```

With_Lock      set_trig_pw    FEED,strt_angle
                lock          FEED,LK_MODE_3
                gosub         Wait_Stop
                goto          Loop

```

```

!-----
!
! Software Interrupt Routine used for a FEED fault condition.
! Turn off the System Ok output.
! Check to see that all motion has stopped.
! Wait for reset input, then restart at Main.
!-----

```

```

Fault          no_op
                turn_off      SYS_OK
                f_decel        FEED
Wait_Busy1      if_stat_on    FEED_BUSY,Wait_Busy1

Start_Over      if_io_off      RESET,Start_Over
                enable_swi
                restart_at     Main

```

```

!-----
!
! Subroutine to configure the MAB map.
! Set the appropriate bits, and check the results.
!-----

```

```

Set_Config      let          map=0
                set_map      map
                set_bit       MABA_TALKER,map
                set_bit       MABA_LISTEN,map
                set_map      map
                get_map      map_back
                get_map_stat  map_stat

                if            map_stat<>0,Config_Fail
                if            map<>map_back,Config_Fail

```

```

Exit_Config      return_sub

```

```

Config_Fail      sys_return

```

```

!-----
!
! Subroutine used to prepare the piecewise profile.
! Prepare the piecewise profile.
! Check the profile status. If correct return, otherwise stop program.
!-----

```

```

Calc_Profile     prep_profile    FEED,prodata

```



```

Calc_Busy      if_stat_on      FEED_CALC,Calc_Busy
               get_pstat      FEED,prof_stat
               if              prof_stat=0,Exit_Calc
               sys_fault
Exit_Calc      return_sub

```

```

!-----
!
! Subroutine used to create a time delay.
!   Wait for timer to time out and return.
!
!-----

```

```

Delay          set_tmr          TIMER,time
Wait_Time      if_tmr_on        TIMER,Wait_Time
               return_sub

```

```

!-----
!
! Subroutine to wait for motion to be completed.
!
!-----

```

```

Wait_Stop      get_status      FEED
               if_stat_on      FEED_BUSY,Wait_Stop  ! check for axis busy
               return_sub

```

Ratio Lock

```

!-----
! TITLE:          Ratio Lock
!
! DESCRIPTION: This program uses the fiber optic input channel as the
!               master data. The lock is a simple ration lock, meaning
!               that the slave will track the master proportional to the
!               currently set ratio. With the use of the Analyzer, the
!               values of the ratio can be changed on the fly and when
!               the CHG_RATIO input is triggered, the new ratio will take
!               effect.
!
! EQUIPMENT:      MSC250
!
! COMMANDS: digi_comp      get_act_spd      get_for_ang
!               get_fol_err      get_for_spd      ratio
!               set_vgain
!
! The above commands are commonly used in many programs. This
! illustration provides the context and syntax required to successfully
! integrate these commands into a complete program. While the use of
! these commands varies from program to program, the following example
! will give the programmer an indication of a typical usage.
!-----

                msc_type      250
                declare      on

!-----
!
! Define constants and variables
!-----

!
! Axis related constants
!

SHEER      equ      1
FIBER      equ      3

BUSY      equ      78
DOWN      equ      77
SHEER_BUSY      equ      (SHEER*16)+BUSY

SHEER_DOWN      equ      (SHEER*16)+DOWN

LK_MODE_1      equ      1
FIBER_MABA      equ      8      ! FIBER sends data to MABA
MABA_TALKER      equ      29     ! FIBER is talker on MABA
MABA_LISTEN      equ      23     ! SHEER is listener on MABA

!
! I/O and Flags
!

```

```

CHG_RATIO      equ      0      ! Input to change ratio.
RESET          equ      6      ! Input to reset a SHEER
!                               fault.
SYS_OK         equ      7      ! Output indicating system
!                               ok.

TIMER          equ      72     ! Timer used to create
!                               delays.
VG_CHK         equ      255    ! Flag indicating vgain
!                               set.

```

```

!
!  Variable definitions
!

```

```

time           integer      ! Determines the delay time for a timer.
map            integer      ! Used to configure the MASTER/SLAVE on
!                               the MAB.
map_back       integer      ! Return variable for the current 'map'
!                               setup.
map_stat       integer      ! Return variable for the current 'map'
!                               status.
mcf            integer      ! Used to configure the FIBER.
mcf_back       integer      ! Return variable for the current 'mcf'
!                               setup.
ratio          integer      ! The current master/slave ratio.
gain           integer      ! The gain of the system.
integral       integer      ! The amount of integral in the system.
damp           integer      ! The amount of damp in the system.
vgain          integer      ! The velocity gain of the system.
angle          integer      ! Stores the current fiber optic angle.
angle2         integer      ! Stores the old fiber optic angle.
error          integer      ! Stores the amount of following error.
temp           integer      ! Used as a temporary storage location.
scale          integer      ! The scale factor for the ratio.
sheer_spd      integer      ! The current SHEER speed.
fiber_spd      integer      ! The current fiber optic speed.
new_f_spd      integer      ! The fiber optic speed scaled by the
!                               ratio.

```

```

!-----
!
!  Initialize variables
!
!-----

```

```

Defaults      let      ratio=4096
              let      time=1
              let      vgain=0
              let      gain=16
              let      integral=0
              let      damp=0

```

```

!-----
!
! Program Setup
! Define axis motion parameters.
! Configure map.
! Wait for motion on fiber optic channel.
! Define software interrupts.
!-----

Setup          set_ac_dc      SHEER,400

               gosub          Set_Config

               get_for_ang     FIBER,1,angle

Wait_Angle      let           angle2=angle
               get_for_ang     FIBER,1,angle
               if              angle2=angle,Wait_Angle

               swi_if_on       0,SHEER_DOWN,Fault
               swi_if_on       1,CHG_RATIO,New_Ratio
               enable_swi

!-----
!
! Main body of program
! Enable SHEER.
! Get information about current speeds.
! Wait until speeds are the same, then check for vgain.
! If vgain has been set continue in Loop, otherwise calculate vgain.
!-----

Main           turn_on        SYS_OK
               drive_on       SHEER

               ratio           SHEER,ratio
               lock            SHEER,LK_MODE_1
               let             scale=ratio*100
               let             scale=scale/4096

Loop           get_status      SHEER
               get_act_spd     SHEER,sheer_spd
               get_for_spd     FIBER,1,fiber_spd
               let             new_f_spd=fiber_spd*scale
               let             new_f_spd=new_f_spd/100
               set_tmr         TIMER,time
Delay          if_tmr_on       TIMER,Delay
               if              sheer_spd<>new_f_spd,Loop
               if_flag_on      VG_CHK,Loop
               gosub          Calc_Vgain
               digi_comp       SHEER,gain,integral,damp
               goto            Loop

```

```

!-----
!
! Software Interrupt Routine used for a SHEER fault condition.
! Turn off the System Ok output.
! Check to see that all motion has stopped.
! Wait for reset input, then restart at Main.
!-----

```

Fault	no_op	
	turn_off	SYS_OK
	f_decel	SHEER
Wait_Busy1	if_stat_on	SHEER_BUSY,Wait_Busy1
Start_Over	if_io_off	RESET,Start_Over
	enable_swi	
	restart_at	Main

```

!-----
!
! Subroutine used to calculate vgain.
! Get the current following error.
! Check to see if within allowable window, if not adjust vgain.
! Otherwise set the VG_CHK flag and return.
!-----

```

Calc_Vgain	no_op	
	get_fol_err	SHEER,error
	let	temp=abs(error)
	if	temp<5,Exit_Vgain
	if	error<5,Less_Vgain
More_Vgain	let	vgain=vgain+10
	set_vgain	SHEER,vgain
	goto	Calc_Vgain
Less_Vgain	let	vgain=vgain-3
	set_vgain	SHEER,vgain
	goto	Calc_Vgain
Exit_Vgain	set_flag	VG_CHK
	return_sub	

```

!-----
!
! Subroutine to configure the MAB map and the MCF map.
! Set the appropriate bits, and check the results.
!-----

```

Set_Config	let	map=0
	set_map	map
	set_bit	MABA_TALKER,map
	set_bit	MABA_LISTEN,map
	set_map	map
	get_map	map_back
	get_map_stat	map_stat
	if	map_stat<>0,Config_Fail
	if	map<>map_back,Config_Fail

```

Config_Pseudo  let          mcf=0
                set_mcf      FIBER,mcf
                set_bit       FIBER_MABA,mcf
                set_mcf       FIBER,mcf

                set_tmr       TIMER,time
Delay1          if_tmr_on     TIMER,Delay1
                get_mcf       FIBER,mcf_back

                if            mcf<>mcf_back,Config_Fail

Exit_Config     return_sub

Config_Fail     sys_return

```

```

!-----
!
! Software Interrupt Routine used to change curretn ratio.
!   Implement new ratio and scaling.
!   Enable interrupts and return.
!-----

```

```

New_Ratio      ratio          SHEER,ratio
                let           scale=ratio*100
                let           scale=scale/4096
                enable_swi
                return_sub

```

CAMS with calc_unit_cam

```

!-----
! TITLE:          CAMS with calc_unit_cam
!
! DESCRIPTION: This program implements the 'calc_unit_cam' feature to
!              calculate a cam of specified length. The lock method
!              demonstrated is mode 9, which causes the slave to lock
!              at a specific angle of the master. This particular lock
!              method unlocks after completing one cam cycle. In this
!              program the lock will be reinitiated each time the cam
!              cycle is completed.
!
! EQUIPMENT:      MSC850/32
!                  ACR850C or ACE850 in slot one.
!                  ACR850C or ACE850 in slot eight.
!
! COMMANDS: begin_data      calc_cam_sum      calc_unit_cam
!              data          dim              dim(card)_cam
!              enable_swi    end_data          get_cam_sum
!              let_byte       set_trig_cam      swi_if_off
!              swi_if_on
!
! The above commands are commonly used in many programs. This
! illustration provides the context and syntax required to successfully
! integrate these commands into a complete program. While the use of
! these commands varies from program to program, the following example
! will give the programmer an indication of a typical usage.
!-----

          msc_type      850
          declare        off

!-----
!
! Declare constants and variables
!-----

!
! Axis related constants
!

DRAW      equ          1
MASTER    equ          8

BUSY      equ          78      ! busy flag offset
DOWN      equ          77      ! down flag offset
CALC      equ          74      ! calculating flag offset
LOCK      equ          79      ! lock flag offset
DRAW_BUSY equ          (DRAW*16)+BUSY ! busy flag - DRAW
DRAW_DOWN equ          (DRAW*16)+DOWN ! down flag - DRAW
DRAW_CALC equ          (DRAW*16)+CALC ! calculating flag - DRAW
DRAW_LOCK equ          (DRAW*16)+LOCK ! lock flag - DRAW

MS        equ          6      ! 1/64

```

```

DS                equ            0            ! no multiply
LK_MODE_9         equ            9            ! lock mode 9
MABA_TALKER       equ            24           ! MASTER is talker on MABA
MABA_LISTEN       equ            23           ! DRAW is listener on MABA

!
!   I/O and Flags
!

RESET             equ            6            ! Reset Fault (input)
SYS_OK            equ            7            ! System okay (output)

TIMER             equ            72           ! Timer used for delays.

!
!   Variable Definitions
!

distance          integer        ! Defines the distance of the cam.
start_cam         integer        ! The position in memory to start the cam.
end_cam           integer        ! The ending position of the cam in memory.
num_elements      integer        ! The number of cam elements.
sum               integer        ! The sum of all the cam elements.
strt_angle        integer        ! The start angle for the Lock to begin.
tmp               integer        ! Used as a temporary storage variable.
ctr               integer        ! Used as a loop counter.
ctr2              integer        ! Used as a loop counter.
time              integer        ! Determines the delay time for a timer.
time2             integer        ! Determines the delay time for a timer.
map               integer        ! Used to configure the MASTER/SLAVE on
!                               ! the MAB.
map_back          integer        ! Return variable for the current 'map'
!                               ! setup.
map_stat          integer        ! Return variable for the current 'map'
!                               ! status.

!
!   Data arrays and unit cam table
!

cam               dim            DRAW,6750
table             dim            DRAW,130
copy              dim            300

trap             begin_data
data              0,14,87,205,357
data              568,744,976,1227,1496
data              1789,2097,2422,2761,3117
data              3487,3671,4267,4677,5100
data              5535,5882,6440,6909,7390
data              7661,8380,8890,9410,9940
data              10480,11030,11588,12153,12728
data              13310,13902,14501,15109,15725
data              16349,16961,17620,18266,18919
data              19579,20246,20919,21596,22281
data              22968,23659,24355,25052,25753

```



```

data      26454,27155,27856,28557,29258
data      29959,30660,31361,32062,32763
data      33464,34165,34866,35567,36288
data      36969,37670,38371,39072,39773
data      40474,41171,41867,42560,43247
data      43930,44609,45282,45949,46609
data      47262,47908,48547,49179,49803
data      50419,51027,51626,52218,52800
data      53375,53940,54498,55048,55586
data      56118,56638,57148,57647,58138
data      58619,59088,59546,59993,60428
data      60851,61261,61657,62041,62441
data      62767,63106,63431,63739,64030
data      64301,64552,64784,64990,65171
data      65323,65441,65514,65536,65536
end_data

!-----
!
! Initialize variables
!-----

Defaults      let      distance=16384
               let      num_elements=300
               let      start_cam=0
               let      end_cam=num_elements-1
               let      strt_angle=1024

!-----
!
! Program Setup
! Setup the unit cam table.
! Configure map.
! Define and enable software interrupts.
!-----

Setup          no_op

               gosub      Setup_Unit_Cam
               gosub      Set_Config

               swi_if_on   0,DRAW_DOWN,Fault
               swi_if_off  1,DRAW_LOCK,Lock_Ang
               enable_swi

!-----
!
! Main body of program
! Enable DRAW.
! Setup DRAW for lock in mode 9.
! Get status for DRAW.
! Remain in Loop, waiting for software interrupts.
!-----

```

```

Main          turn_on      SYS_OK
              drive_on     DRAW̄

              cam_data     DRAW,cam,MS,DS

              set_trig_cam  DRAW,strt_angle
              lock         DRAW,LK_MODE_9

Loop          get_status   DRAW
              gosub        Delay
              goto         Loop

!-----
!
! Software Interrupt Routine indicating that the cam has completed
!   and the DRAW has been unlocked.
!   Lock the DRAW axis in mode 9.
!   Enable the software interrupts and return.
!-----

Lock_Ang      no_op
              set_trig_cam  DRAW,strt_angle
              lock         DRAW,LK_MODE_9
              enable_swi
              return_sub

!-----
!
! Software Interrupt Routine used for a DRAW fault condition.
!   Turn off the System Ok output.
!   Check to see that all motion has stopped.
!   Wait for reset input, then restart at Main.
!-----

Fault         no_op
              turn_off     SYS_OK
              f_decel      DRAW̄

Wait_Busy1    if_stat_on   DRAW_BUSY,Wait_Busy1

Start_Over    if_io_off    RESET,Start_Over
              enable_swi
              restart_at    Main

!-----
!
! Subroutine to configure the MAB map.
!   Set the appropriate bits, and check the results.
!-----

Set_Config    let          map=0
              set_map      map
              set_bit      MABA_TALKER,map
              set_bit      MABA_LISTEN,map

```

```

        set_map      map
        get_map      map_back
        get_map_stat map_stat

        if          map_stat<>0,Config_Fail
        if          map<>map_back,Config_Fail

Exit_Config  return_sub

Config_Fail  sys_return

!-----
!
! Subroutine to prepare and create the unit cam.
! Clear the contents of the cam array.
! Load the unit cam array into memory on DRAW.
! Calculate the cam based on the distance, starting location, and
!   number of elements.
! Calculate the cam sum and compare it to the desired distance.
! If the sum does not equal the distance, then stop system.
! To observe the cam elements they are loaded into memory on the MSC.
!
!-----

Setup_Unit_Cam no_op
let          ctr=0
Clr_Cam      let      cam[ctr]=0
            let      ctr=ctr+1
            if      ctr<6750,Clr_Cam

Load_Table   let      ctr=0
Load_Loop    let      tmp=trap[ctr]
            let      table[ctr]=tmp
            let      ctr=ctr+1
            if      ctr<130,Load_Loop

Calc_Busy    calc_unit_cam DRAW,distance,num_elements,start_cam
            if_stat_on DRAW_CALC,Calc_Busy

Sum_Busy     calc_cam_sum DRAW,start_cam,end_cam
            if_stat_on DRAW_CALC,Sum_Busy

            get_cam_sum DRAW,sum
            if      sum=distance,Copy

Calc_Err     sys_fault

Copy         let      ctr=0
Load_Copy    let_byte tmp=cam[ctr]
            let_byte copy[ctr]=tmp
            let      ctr=ctr+1
            if      ctr<300,Load_Copy

```

```
!-----  
!  
! Subroutine used to create a time delay. -  
!   Wait for timer to time out and return. -  
!  
!-----
```

```
Delay          set_tmr          TIMER,time  
Wait_Time      if_tmr_on        TIMER,Wait_Time  
               return_sub
```

I/O Functions with IOE

```

!-----
! TITLE:          I/O functions with IOE
!
! DESCRIPTION: This program was written to use IO's as a binary code
!              bus to select a particular type of motion.
!
! EQUIPMENT:  Hardware is wired in the following manor;
!              Outputs  Inputs          WIRING
!              MSC-850/32 IOE-850  MSC-850/32 IOE-850  MSC-850/32
!              IO#0 -> IO#8          (5) -> (5)      JUMP ALL (6) -> GND
!              IO#1 -> IO#9          (5) -> (5)      IOE-850
!              IO#2 -> IO#10         (5) -> (5)      JUMP ALL (6) -> +15V
!
! COMMANDS: blk_io_in          blk_io_out          clr_hi_scan
!            if_tmr_off        set_hi_scan         test_mode
!            vel_ccw           vel_cw
!
! The above commands are commonly used in many programs.  This
! illustration provides the context and syntax required to successfully
! integrate these commands into a complete program.  While the use of
! these commands varies from program to program, the following example
! will give the programmer an indication of a typical usage.
!-----

                msc_type          850/32
                declare           on

!-----
!
!   Declare constants and variable
!-----

!
!   Axis related constants
!

ROLL_1          equ              1
ROLL_2          equ              2

BUSY            equ              78
DOWN            equ              77
ROLL_1_BUSY     equ              (ROLL_1*16)+BUSY
ROLL_2_BUSY     equ              (ROLL_2*16)+BUSY

ROLL_1_DOWN     equ              (ROLL_1*16)+DOWN
ROLL_2_DOWN     equ              (ROLL_2*16)+DOWN

!
!   I/O and Flags
!

ON_BOARD        equ              0              ! start of output IO block
!                                           (8 IO's)

```

```

IOE1          equ          8          ! start of input IO block
!              (8 IO's)
RESET         equ          22         ! Reset Fault (input)
SYS_OK        equ          23         ! System Ok (output)
TIMER         equ          72         ! Timer Flag

!
!   Variable Definition
!

code           integer          ! output variable to IO
!              block
copy           integer          ! input variable to IO
!              block
ac_dc          integer          ! acceleration, deceleration
!              ratio
speed          integer          ! speed
time_delay     integer          ! variable carrying
!              required delay
ctr            integer          ! array indexer (counter)

!
!   Array Definition
!

c_array        begin_data          ! array with codes (binary)
                data                1,2,3,4
                end_data

!-----
!
!   Initialize variables
!
!-----

Defaults      no_op
                let                ac_dc=89
                let                speed=200
                let                ctr=0

!-----
!
!   Program Setup
!   Define ROLL_1 and ROLL_2 motion parameters.
!   Define and enable software interrupts.
!
!-----
Setup         no_op
                set_ac_dc          ROLL_1,ac_dc  ! set accel. decel. for
!              roll 1
                set_ac_dc          ROLL_2,ac_dc  ! set accel. decel. for
!              roll 2
                set_speed          ROLL_1,speed  ! set speed for roll 1
                set_speed          ROLL_2,speed  ! set speed for roll 2
                set_hi_scan        ! enable fast IOE scan

                swi_if_on          0,ROLL_1_DOWN,Fault

```

```

        swi_if_on      1,ROLL_1_DOWN,Fault
        enable_swi

!-----
!
!      Main body of the program.
!      Send a binary code from array through on-board IO's.
!      Read code into 'copy' through first 8 IO's at IOE.
!      Compare and direct to motion routine.
!      Increment array counter, get next value and loop again.
!-----
Main      no_op
          turn_on      SYS_OK      ! System Ready
          drive_on     ROLL_1
          drive_on     ROLL_2

Start     no_op
          let          code=c_array[ctr]
          blk_io_out   ON_BOARD,code

          let          time_delay=0
          gosub        Delay

Block_Read no_op
          blk_io_in    IOE1,copy
          if           copy=1,Run_Roll1
          if           copy=2,Run_Roll2
          if           copy=3,Run_Roll1ccw
          if           copy=4,Run_Roll2ccw
          goto         Block_Read

Run_Roll1 no_op
          vel_cw       ROLL_1
          let          time_delay=2000
          gosub        Delay
          f_decel      ROLL_1

Wait_Stop1 no_op
          if_stat_on   ROLL_1_BUSY,Wait_Stop1
          goto         Do_Next

Run_Roll2 no_op
          vel_cw       ROLL_2
          let          time_delay=2000
          gosub        Delay
          f_decel      ROLL_2

Wait_Stop2 no_op
          if_stat_on   ROLL_2_BUSY,Wait_Stop2
          goto         Do_Next

Run_Roll1ccw no_op
          vel_ccw      ROLL_1
          let          time_delay=2000
          gosub        Delay
          f_decel      ROLL_1

```

```

Wait_Stop1ccw  no_op
                if_stat_on    ROLL_1_BUSY,Wait_Stop1ccw
                goto          Do_Next

Run_Roll2ccw   no_op
                vel_ccw       ROLL_2
                let           time_delay=2000
                gosub         Delay
                f_decel        ROLL_2

Wait_Stop2ccw  no_op
                if_stat_on    ROLL_2_BUSY,Wait_Stop2ccw
                goto          Do_Next

Do_Next        no_op
                let           ctr=ctr+1
                if            ctr>3,Program_End
                goto          Start

Program_End    no_op
                let           ctr=0
                goto          Start

```

```

!-----
!
! Subroutine used to create a time delay.
! Wait until the timer times out.
!-----

```

```

Delay          no_op
                set_tmr       TIMER,time_delay
Wait_Tmr       if_tmr_off    TIMER,Exit_Delay
                goto          Wait_Tmr

Exit_Delay     no_op
                return_sub

```

```

!-----
!
! Software Interrupt Routine used for a ROLL_x fault condition.
! Turn off the System Ok output.
! Check to see that all motion has stopped.
! Reset 'ctr' and place controller in test mode.
! Wait for reset input, then restart at Main.
!-----

```

```

Fault          no_op
                f_decel       ROLL_1
                f_decel       ROLL_2
                clr_hi_scan
                turn_off      SYS_OK

Wait_F_Stop    if_stat_on    ROLL_1_BUSY,Wait_F_Stop
                if_stat_on    ROLL_2_BUSY,Wait_F_Stop

```


	drive_off	ROLL_1
	drive_off	ROLL_2
	let	ctr=0
	test_mode	ROLL_1
	test_mode	ROLL_2
Start_Over	if_io_off	RESET,Start_Over
	enable_swi	
	restart_at	Main

Offset Methods

```

!-----
! TITLE:          Offset methods
!
! DESCRIPTION: This program uses MASTER-SLAVE ratios to demonstrate
!              the offset and speed control of a master axis.
!
! EQUIPMENT:     MSC-850/32
!                ACR-850 or ACE-850 in slot 1
!                ACR-850 or ACE-850 in slot 2
!                MCF-850 in slot 3
!
! COMMANDS: incr_com          offset_master      read_offset
!            set_offset
!
! The above commands are commonly used in many programs. This
! illustration provides the context and syntax required to successfully
! integrate these commands into a complete program. While the use of
! these commands varies from program to program, the following example
! will give the programmer an indication of a typical usage.
!-----

                msc_type      850/32
                declare      on

!-----
!
! Declare constants and variables
!-----

!
! Axis related constants
!

CONVEYOR      equ      1
COVER        equ      2
PSEUDO       equ      3

BUSY         equ      78
DOWN        equ      77
CONVEYOR_BUSY equ      (CONVEYOR*16)+BUSY
COVER_BUSY   equ      (COVER*16)+BUSY
PSEUDO_BUSY  equ      (PSEUDO*16)+BUSY

CONVEYOR_DOWN equ      (CONVEYOR*16)+DOWN
COVER_DOWN   equ      (COVER*16)+DOWN
PSEUDO_DOWN  equ      (PSEUDO*16)+DOWN

MABA_TALKER   equ      29
!
MABA_LISTEN1  equ      23
!
MABA_LISTEN2  equ      22
!
! PSEUDO is talker
! on MABA
! CONVEYOR listen
! to MABA
! CONVEYOR listen
! to MABA

```

```

PSEUDO_MABA      equ          10          ! PSEUDO talks to
!                                     MABA

!
!   I/O and Flags
!

INCREMENT        equ          0          ! Activate Inc. Conv.Pos.
!                                     (input)
ADVANCE           equ          1          ! Activate offset master
!                                     (input)
POSITIONING       equ          2          ! Activate positioning
!                                     (input)
RESET             equ          6          ! Reset Fault (input)
SYS_OK            equ          7          ! System okay (output)
TIMER             equ          72         ! Timer used for delays.
WATCHDOG          equ          73         ! Timer used for watchdog

!
!   Data tables
!

POS_ARRAY         begin_data
                  data          10240,-40960,20480,0,40960,0
                  end_data

!
!   Misc. Constants
!
WATCHDOG_TIME     equ          50          !      mSec

!
!   Variable definitions
!

map               integer          ! Var for configuring
!                                     Hardware
map_copy          integer          ! Var used to verify config.
mcf               integer          ! Var used to setup MCF
mcf_copy          integer          ! Var used to verify MCF
!                                     setup
time_delay        integer          ! Var with delay requirement
ac_dc             integer          ! Accel decel
speed             integer          ! Speed
offset            integer          ! Var used to offset master
offset_copy       integer          ! Angle read back from
!                                     offset
conv_offset       integer          ! Var used to inc. pos
!                                     for Conv.
advance           integer          ! Var used to move master
!                                     offset
advance_copy      integer          ! Used to verify move by
!                                     advance
ctr               integer          ! counter, index of pos
!                                     array
actual_pos        integer          ! Var containing value
!                                     from array

```

```

time_mark      integer      ! Test variable used for
!                                     time test
delta_time     integer      ! Var used to calc.
!                                     interval
start_pos      integer      ! Var used to get starting
!                                     pos
conv_pos       integer      ! Var used to get commanded
!                                     pos
delta_pos      integer      ! Var used to measure
!                                     displacement
no_of_times    integer      ! Software trap for one
!                                     time usage
millisec       integer      ! Motor interrupts in mSec.

```

```

!-----
!                                     -
!   Initialize variables                                     -
!                                     -
!-----

```

```

Defaults      no_op
               let      ac_dc=300
               let      speed=1000
               let      offset=1024
               let      conv_offset=1024
               let      advance=2048
               let      millisec=200

               let      ctr=0
               let      no_of_times=0
               let      time_mark=0
               let      conv_pos=0
               let      start_pos=0

```

```

!-----
!                                     -
!   Program setup                                           -
!   Define motion parameters.                               -
!   Configure map.                                          -
!   Define software interrupts.                             -
!                                     -
!-----

```

```

Setup         no_op
               set_ac_dc      PSEUDO,ac_dc
               set_speed      PSEUDO,speed

               gosub          Set_Config

               swi_if_on      0,CONVEYOR_DOWN,Fault
               swi_if_on      1,COVER_DOWN,Fault
               swi_if_on      2,INCREMENT,Inc_Conveyor
               swi_if_on      3,ADVANCE,Set_Advance
               swi_if_on      4,POSITIONING,Positioning

               enable_swi

```

```

!-----
!
! Main body of the program.
! Start timed motion.
! Start Position of Master.
!-----
Main          no_op
              turn_on          SYS_OK
              drive_on         CONVEYOR
              drive_on         COVER
Stay          goto            Stay

!-----
!
! Increment Offset Conveyor
! Test for proper displacement.
!-----
Inc_Conveyor  no_op

              let              conv_pos=0
              let              start_pos=0

              get_pos          CONVEYOR,start_pos
              incr_com          CONVEYOR,conv_offset,millisecond

Get_Inc_Pos   set_tmr          WATCHDOG,WATCHDOG_TIME
              get_pos          CONVEYOR,conv_pos
              let              delta_pos = conv_pos - start_pos
              if_tmr_off       WATCHDOG,Fail_Inc
              if               delta_pos<>conv_offset,Get_Inc_Pos

              enable_swi
              return_sub

Fail_Inc      no_op
              gosub            Invalid_Ofst

!-----
!
! Set Advance (Correction) for Master
! Only to be used once per operation.
!-----
Set_Advance   no_op
              if               no_of_times<>0,Exit_Advance
              let              no_of_times=1
              lock              CONVEYOR,1
              lock              COVER,1

              let              time_delay=20
              gosub            Delay

              offset_master    PSEUDO,advance

```

```

        let          time_delay=20
        gosub        Delay

        get_angle    CONVEYOR,advance_copy
        if           advance_copy<>advance,Wrong_Ofst

!
!  Move to Offset Location
!

Wait_Ready          set_offset    PSEUDO,offset
                    if_stat_on    PSEUDO_BUSY,Wait_Ready
                    read_offset   PSEUDO,offset_copy
                    if           offset<>offset_copy,Wrong_Ofst

Exit_Advance        unlock        CONVEYOR,0
                    unlock        COVER,0
                    enable_swi
                    return_sub

!
!  Offset Failure Gateway to Service Routine
!

Wrong_Ofst          no_op
                    gosub          Invalid_Ofst

!-----
!
!  Positioning of master (and Slaves) with corrections (offsets)
!
!-----
Positioning          no_op
                    lock          CONVEYOR,1
                    lock          COVER,1
                    let          actual_pos=POS_ARRAY[ctr]
                    position      PSEUDO,actual_pos
Wait_Position        if_stat_on    PSEUDO_BUSY,Wait_Position
                    let          ctr=ctr+1
                    if           ctr>5,Clr_Ctr
                    goto          Return_Main

!
!  End of routine
!

Clr_Ctr              no_op
                    let          ctr=0

Return_Main          unlock        CONVEYOR,0
                    unlock        COVER,0
                    enable_swi
                    return_sub

```

```

!-----
!
! Subroutine used to configure the MAB map and the MCF map.
! Set the appropriate bits, and check the results.
!-----

Set_Config      let          map=0
                set_map      map
                set_bit      MABA_TALKER,map
                set_bit      MABA_LISTEN1,map
                set_bit      MABA_LISTEN2,map
                set_map      map
                let          time_delay=30
                gosub         Delay
                get_map      map_copy

                if          map <> map_copy,Config_Fail

Config_Pseudo   no_op
                let          mcf=0
                set_mcf      PSEUDO,mcf
                set_bit      PSEUDO_MABA,mcf
                set_mcf      PSEUDO,mcf

                let          time_delay=30
                gosub         Delay

                get_mcf      PSEUDO,mcf_copy
                if          mcf<>mcf_copy,Config_Fail

                enable_swi
                return_sub

Config_Fail     no_op
                sys_return

!-----
!
! Subroutine used to create a time delay.
! Wait until the timer times out.
!-----

Delay           no_op
                set_tmr      TIMER,time_delay
Wait_Time       if_tmr_on    TIMER,Wait_Time
                return_sub

!-----
!
! Subroutine used to halt operation on erroneous offset
!-----

Invalid_Ofst    no_op
                f_decel      PSEUDO

```

```

Wait_Stop      if_stat_on    PSEUDO_BUSY,Wait_Stop
                unlock        CONVEYOR,0
                unlock        COVER,0
Wait_Loop      no_op
                if_stat_on    CONVEYOR_BUSY,Wait_Loop
                if_stat_on    COVER_BUSY,Wait_Loop

                sys_return

```

```

!-----
!
! Software Interrupt Routine used to handle fault conditions .
! Turn off the System Ok output.
! Check to see that all motion has stopped.
! Wait for reset input, then restart at Main.
!-----
!

```

```

Fault          no_op
                turn_off      SYS_OK
                f_decel       CONVEYOR
                f_decel       COVER
Wait_Busy1     if_stat_on    CONVEYOR_BUSY,Wait_Busy1
                if_stat_on    COVER_BUSY,Wait_Busy1

Start_Over     if_io_off     RESET,Start_Over
                enable_swi
                restart_at    Main

```


Clear Software Interrupts

```

!-----
! TITLE:          Clear Software Interrupts
!
! DESCRIPTION: This program was written to demonstrate the usage of
!              interrupt managing and predefined-function hardware
!              interrupt usage (trap_pos).
!
! EQUIPMENT:  MSC-850/32
!              ACR-850 or ACE-850 in slot 1
!              ACR-850 or ACE-850 in slot 2
!              Input IO or Switch in location 1
!              Input IO or Switch in location 8
!
! COMMANDS: clr_all_swi          clr_swi          trap_pos
!
! The above commands are commonly used in many programs.  This
! illustration provides the context and syntax required to successfully
! integrate these commands into a complete program.  While the use of
! these commands varies from program to program, the following example
! will give the programmer an indication of a typical usage.
!-----

                msc_type          850/32
                declare           on

!-----
!
!  Declare constants and variables
!-----

!
!  Axis related constants
!

WHEEL          equ          1
BUSY           equ          78
DOWN           equ          77
WHEEL_BUSY     equ          (WHEEL*16)+BUSY
WHEEL_DOWN     equ          (WHEEL*16)+DOWN

!
!  I/O and Flags
!

EMERGENCY      equ          6          ! Input
DIRECTION      equ          7          ! Input
TIMER          equ          72         ! Timer flag
HWI_ARMED_1    equ          91         ! Got position

!
!  Variable definitions
!

```

```

speed          integer
ac_dc          integer
time_delay     integer
next_bottle    integer
pos_trapped    integer

```

```

!-----
!
!   Initialize variables
!
!-----

```

```

Defaults      no_op
               let          ac_dc=50
               let          speed=1000
               let          next_bottle=40960

```

```

!-----
!
!   Program setup
!
!-----

```

```

Setup         no_op
               set_ac_dc    WHEEL,ac_dc
               set_speed    WHEEL,speed
               drive_on     WHEEL

               swi_if_on    0,EMERGENCY,Emergency
               swi_if_on    1,DIRECTION,Do_Forward
               swi_if_on    2,HWI_ARMED_1,Grab_Pos_1
               enable_swi

```

```

!-----
!
!   Main body of the program.
!   This routine has only a loop to allow the interrupts to work
!
!-----

```

```

Main          no_op
               goto          Main

```

```

!-----
!
!   This routine was created to do a forward motion based on the level
!   of the IO (7) as s. interrupt signal.
!
!-----

```

```

Do_Forward    no_op
               clr_swi      1
               swi_if_on    1,DIRECTION,Do_Reverse
               enable_swi

Idx_Loop      no_op
               index        WHEEL,next_bottle
Wait_Busy     if_stat_on    WHEEL_BUSY,Wait_Busy

               goto         Idx_Loop

               return_sub

```

```

!-----
!
!   This routine was created to do a reverse motion based on the level
!   of the IO (7) as s. interrupt signal.
!-----
Do_Reverse      no_op
                 clr_swi          1
                 swi_if_on        1,DIRECTION,Do_Forward
                 enable_swi
Idx_Loop_R      no_op
                 let              next_bottle=0-next_bottle
                 index            WHEEL,next_bottle
Wait_Busy_R     if_stat_on        WHEEL_BUSY,Wait_Busy_R
                 goto             Idx_Loop_R
                 return_sub

!-----
!
!   Subroutine used to create a time delay.
!   Wait until the timer times out.
!-----
Delay           no_op
                 set_tmr          TIMER,time_delay
Wait_Time       if_tmr_on         TIMER,Wait_Time
                 return_sub

!-----
!
!   Routine created to capture the prescent position of the motor when a
!   hardware interrupt comes in.
!-----
Grab_Pos_1      no_op
                 enable_hwi
                 trap_pos         WHEEL
                 get_trap_pos     WHEEL,pos_trapped

!-----
!
!   Subroutine used to service emergency stop.
!-----
Emergency       no_op
                 clr_all_swi
Wt_Emrg_Stop    if_stat_on        WHEEL_BUSY,Wt_Emrg_Stop
                 drive_off        WHEEL
                 sys_return

```

Enable Sequence for Absolute Encoder

```

!-----
! TITLE:          Enable Sequence for Absolute Encoder
!
! DESCRIPTION: This program was written to demonstrate the sequence for
!               initializing, enabling and starting motion in an absolute
!               encoder based system.
!
! EQUIPMENT:
!               MSC-850/32
!               ACY-850   in slot 1
!
! COMMANDS: set_acy_cnt          set_home
!
! The above commands are commonly used in many programs.  This
! illustration provides the context and syntax required to successfully
! integrate these commands into a complete program.  While the use of
! these commands varies from program to program, the following example
! will give the programmer an indication of a typical usage.
!-----

                msc_type          850/32
                declare            on

!-----
!
! Declare Constants and Variables
!-----

!
! Axis Related Constants
!

LIFT            equ                1                ! ACY-850 controller card

!
! I/O and Flags
!

INITIALIZE      equ                0                ! Initialize Motion (input)
HOME            equ                1                ! Go to 'home' (input)
POSITION        equ                2                ! Position motor (input)
TIME_OUT        equ                5                ! Time out (output)
FAULT          equ                6                ! Fault (output)
SYS_OK          equ                7                ! System Operative (output)
LIFT_DOWN      equ                82               ! Lift DOWN flag
LIFT_BUSY       equ                94               ! Lift BUSY flag
TIMER          equ                72               ! Timer flag
DRIVE_ON       equ                254              ! Drive on user flag

!
! Miscellaneous Equates
!

```

```

TEN_TURNS      equ      40960      ! 4096*10
ZERO_POS       equ      0          ! Position 0
ACY_CNTS       equ      4096      ! Encoder counts per
!                               revolutions

```

```

!
! Variable Definition
!

```

```

speed          integer
ac_dc          integer
time           integer
com            integer
pos            integer
zcom           integer
zoff           integer

```

```

!-----
!
! Initialize Variables
!
!-----

```

```

Defaults      no_op
               let      speed=100
               let      ac_dc=20

```

```

!-----
!
! Program Setup
!
!-----

```

```

Setup         no_op
               set_acy_cnt  LIFT,ACY_CNTS

               drive_off    LIFT
               turn_on      SYS_OK

               let          time=100
               gosub        Wait

               drive_on     LIFT
               gosub        Wait

               set_home     LIFT,zoff

               swi_if_on    0,LIFT_DOWN,Fault
               enable_swi

```

```

!-----
!
! Main body of the Program
! Read an input, select type of operation.
!
!-----

```

```

Welcome      no_op
              if_io_on      INITIALIZE, Initialize
              if_io_on      HOME, Home
              if_io_on      POSITION, Position
              goto          Welcome

!-----
!
! Initialize Motor Position
!   Read offset from home.
!   Set home with offset.
!
!-----
Initialize   no_op
              drive_on      LIFT
              let            time=100
              gosub          Wait

              get_com        LIFT, zcom
              let            zoff=0-zcom
              set_home        LIFT, zoff

              goto          Welcome

!-----
!
! Subroutine used to direct the motor to 'home'
!   Position motor to 'HOME'
!
!-----
Home          no_op

              set_speed      LIFT, speed
              set_ac_dc      LIFT, ac_dc
              position        LIFT, ZERO_POS

Homing        if_stat_on     LIFT_DOWN, Fault
              if_stat_on     LIFT_BUSY, Homing

              goto          Welcome

!-----
!
! Subroutine used to position to the next location on table
!   Position Motor to required location.
!
!-----
Position      no_op

              set_speed      LIFT, speed
              set_ac_dc      LIFT, ac_dc
              position        LIFT, TEN_TURNS

Positioning    if_stat_on     LIFT_DOWN, Fault
              if_stat_on     LIFT_BUSY, Positioning
              goto          Welcome

```

```
!-----
!
! Subroutine created to handle motor fault.
!
!-----
Fault          no_op
               turn_off      SYS_OK
               turn_on       FAULT
               drive_on      LIFT
               let           time=100
               gosub         Wait
               if_stat_on    LIFT_DOWN,Fault
               turn_on      SYS_OK
               turn_off     FAULT
               restart_at   Welcome

!-----
!
! Subroutine used to crate a time delay.
! Wait until the timer times out.
!
!-----
Wait           no_op
               set_tmr       TIMER,time
Wait_Tmr       if_tmr_on     TIMER,Wait_Tmr
               return_sub
```