IB-11C001

MOTION CONTROL SYSTEMS, MSC SERIES **JUNE 1995**

MACROPROGRAM DEVELOPMENT SYSTEM

INSTRUCTION MANUAL

INDUSTRIAL INDEXING SYSTEMS, Inc.

Revision - E

Approved By:

TABLE OF CONTENTS

1.0 INTRODUCTION

This document is part of a series of books that support Industrial Indexing Systems MSC family of motion control systems. It provides information about the Macroprogram Development System which serves as a tool to assist the user in development of motion control programs.

1.1 OVERVIEW

The Macroprogram Development System is a software tool designed to provide an effective environment for creating Macroprograms for the MSC family of motion controllers. Program development for the MSC consists of creating and editing text files containing the appropriate program instructions, compiling these files to generate executable programs, and on-line program debugging. In addition to these features, the Macroprogram Development System provides aids for disk file maintenance and configuration.

Highlights of the Macroprogram Development System are:

1. Simple entry and editing of programs.

2. Interaction between editing and compiling to quickly identify program lines containing errors.

3. On-line manual describing the purpose and format of each Macroprogramming Language instruction.

4. File manager to simplify creating and deleting Macroprogram source files.

5. Comprehensive real time test and debug facility, including:
   a. Program tracing
   b .Inspection and modification of data values and input/outputs
   c. Monitoring of Controller status.

The remaining sections of this book assume that you are familiar with the operation of your personal computer. If you are not, please refer to your computer documentation.

1.2 MANUAL CONVENTIONS

Throughout this manual, the following typeface conventions are used:

1. Instruction names appear in **bold** print.

2. Optional parameters appear in *italics*.

3. MSDOS commands typed by the user appear in **BOLD**, all capitals.

2.0 INSTALLATION

2.1 HARDWARE CONFIGURATION

The Macroprogram Development System (Toolkit) is designed to be used with an IBM compatible personal computer.

The Toolkit, like any IIS product, has its' own part number. Software part numbers are assigned using the format **SFO-NNNN RX** where **SFO-NNNN** indicates the software part number and **RX** indicates the revision. Whenever the Toolkit is significantly changed, a new part number is assigned. There are several versions of the Toolkit currently in use.

The minimum recommended hardware configuration necessary for using the Macroprogram Development System, part numbers SFO-3040, SFO-3076, SFO-3082 or SFO-3110 is listed below:

- Computer:IBM PC, XT, or AT computer, or compatible
- Disk Storage:Dual 720K disk drives or single 720K disk drive with hard disk
- Display:Monochrome, CGA, EGA or Hercules Display
- Memory:Minimum 512K bytes user memory
- Comm Ports:Asynchronous communications adapter(RS232C serial interface)
- Op Sys:MS-DOS revision 2.0 or later

The Macroprogram Development System, part number SFO-3136, was released primarily for the support of the MSC-850/32 Controller. Since this controller allows for programs up to 64,000 bytes **and** with additional data storage of 64,000 bytes, the Toolkit memory requirements increased dramatically. The minimum recommended hardware configuration necessary for using this version of the Toolkit is as follows:

- Computer:IBM PC AT computer with 286, 386 OR 486 processor, or compatible
- Disk Storage:Dual 720K disk drives or single 720K disk drive with hard disk
- Display:Monochrome, CGA, EGA or Hercules Display
- Memory:Minimum 640K bytes of base user memory and an additional 2MB of extended and/or expanded memory
- Comm Ports:Asynchronous communications adapter(RS232C serial interface)
- Op Sys:MS-DOS revision 5.0 or later
- Clock rate: 16 Mhz or higher

NOTE

Not all PC systems advertised as being "IBM PC compatible" are 100% compatible. The Toolkit may not work on systems which are not truly 100% IBM PC compatible.

2.2 INSTALLATION

The Toolkit is normally supplied on a 3.5", 720 Kbyte diskette. The diskette will be identified with an SFO (System Functional Operation) number which identifies the Toolkit and the current revision level.

2.2.1 DUAL FLOPPY DISK SYSTEM

To install the Toolkit on a dual floppy system, perform the following steps:

1. Obtain two blank, formatted diskettes. One diskette will be used to make a working copy of the IIS Toolkit diskette. The second diskette will serve to store the programs you will develop.
2. Power up your computer and load the MSDOS operating system.
3. When the **A>** prompt appears on the screen, place one of the blank diskettes in the **B** diskette drive, and place the Toolkit diskette in the **A** drive.
4. Type the following command and press the **Enter** key.

   COPY A:*.* B:/V

   This command instructs MSDOS to copy the Toolkit diskette from drive A to drive B and to verify that all information was copied correctly.
5. Remove the IIS Toolkit diskette from drive A and store it in a safe place. Remove the copy from drive B and label it with the appropriate SFO number.
6. Place the working diskette in drive A and the blank data diskette in drive B.
7. Type **IIS** and press the **Enter** key.
8. After a few seconds, the initial Toolkit screen should appear. Proceed to the Configuration section of this chapter for further instructions.

2.2.2 HARD DISK SYSTEM

To install the Toolkit on a hard disk system, perform the following steps. These instructions assume that your hard disk is designated as drive **C**. If your hard disk is designated by a different letter, use that letter in place of **C** in the instructions below.

1. Power up your computer and load the MSDOS operating system.

2. It is recommended that the Toolkit be installed in a subdirectory rather than in the root directory of your disk. Create a subdirectory named **TOOLKIT** by typing the following command and pressing the **Enter** key.

   MKDIR \TOOLKIT

3. It is suggested that you create a subdirectory to contain the programs you will develop using the Toolkit. For example, to create a subdirectory called **MACROS**, type the following command, followed by the **Enter** key:

   MKDIR \MACROS

4. Place the Toolkit diskette in floppy drive **A**.

5. To copy the Toolkit programs to your hard drive, type the following command, followed by the **Enter** key:

   COPY A:*.* \TOOLKIT /V

This command instructs MSDOS to copy the Toolkit diskette from drive A to the \TOOLKIT subdirectory on your hard drive and to verify that all information was copied correctly.

6. Log on to the TOOLKIT directory by typing the following command and then pressing the **Enter** key.

CD \TOOLKIT

7. Type **IIS** and press the **Enter** key.

8. After a few seconds, the initial Toolkit screen should appear. Proceed to the Configuration section of this chapter for further instructions.

9. For subsequent use of the Toolkit, repeat steps 6 and 7.

2.3 CONFIGURATION

If you are using the Toolkit with part number SFO-3136, a "banner" similar to the one below will be displayed for a few seconds, on Toolkit startup:

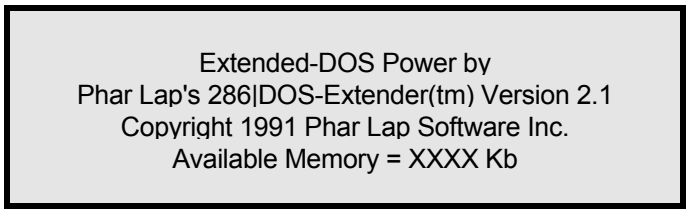


Figure 2.1 - BANNER SCREEN

A software "shell" package is used by the Toolkit in order to access the extended and/or expanded memory in your computer. The "banner" is displayed by this "shell" software package, indicating the manufacturers name, the software "shell" package name, revision and available memory.

If you are using a Toolkit with part number SFO-3110 or SFO-3136, you may be prompted to select the "memory model" to be used for the development of your programs. A screen similar to **Figure 2.2** may be displayed.

The first configuration screen allows you to select the appropriate macroprogram memory size based on your MSC system. See **Figure 2.2**.

STANDARD will accommodate any MSC system as long as your macroprogram is smaller than 16K bytes (PROGRAM + DATA).

MEDIUM would be selected if you are using a MSC-250 and your macroprogram is larger than 16K bytes program + data. The program may not exceed 32K bytes (PROGRAM + DATA).

LARGE is used for the MSC-850/32 system when your macroprogram size exceeds 16K bytes program + data. The program may not exceed either 64K Program and 64K Data.

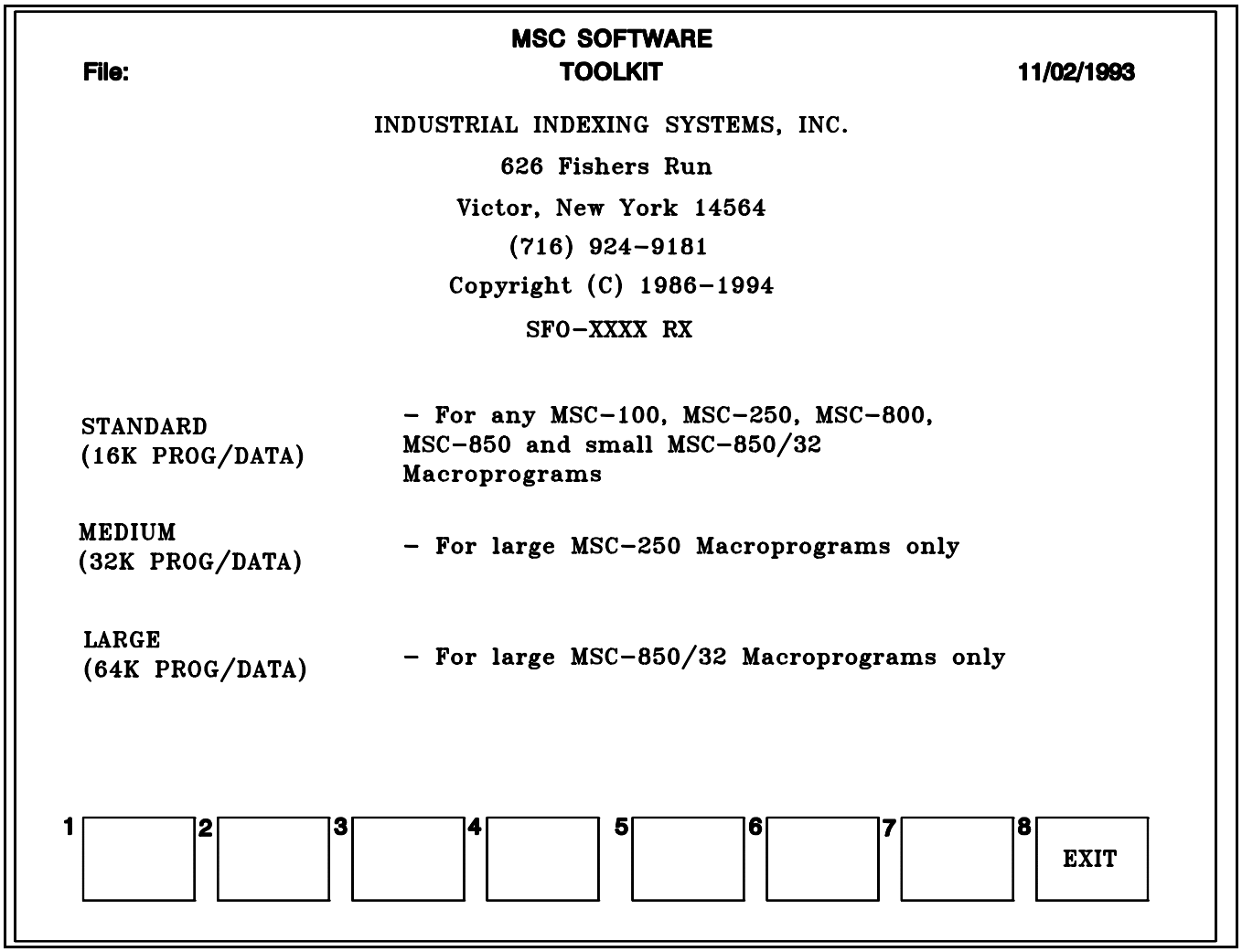


Figure 2.2 - MSC MEMORY SELECTION SCREEN

Generally speaking, if you are creating programs for use with the MSC-850/32 and the program and or data is expected to exceed 16K bytes, you should select the LARGE memory model.

If you are creating programs for use with any other MSC type controller, the STANDARD memory model should be selected, since the sum of the program and data areas cannot exceed 16K bytes anyway.

The Toolkit provides a configuration subsystem that allows you to tailor the Toolkit to your computer. There are two steps to the configuration process. System Configuration covers various options such as specifying where your program files are to be stored, maximum program size in lines, etc. Color Configuration allows you to customize screen colors and characteristics.

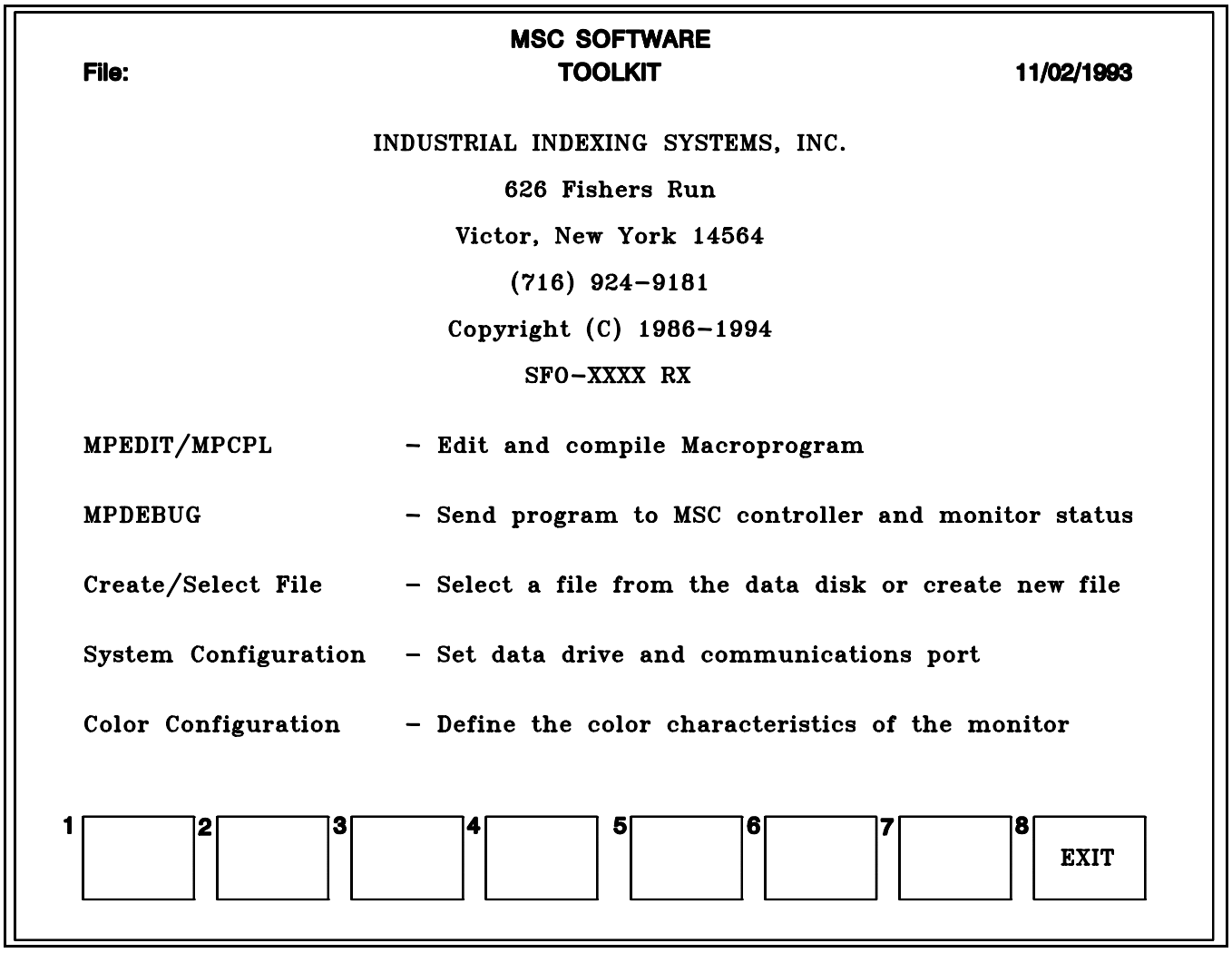


Figure 2.3 - TOOLKIT STARTUP SCREEN

Figure 2.3 shows the Toolkit startup screen. Note that the first selection on the screen, **MPEDIT/MPCPL**, will be highlighted by an inverse video bar. This bar can be moved from one selection to another by using the up and down arrow keys. To activate the desired selection, press the **Enter** key.

2.3.1 SYSTEM CONFIGURATION

The System Configuration option of the Toolkit allows you to set up certain Toolkit default characteristics. These characteristics are usually only set up once, and the configuration will not need to be selected again unless the characteristics change. The System Configuration screen is shown in **Figure 2.4**.

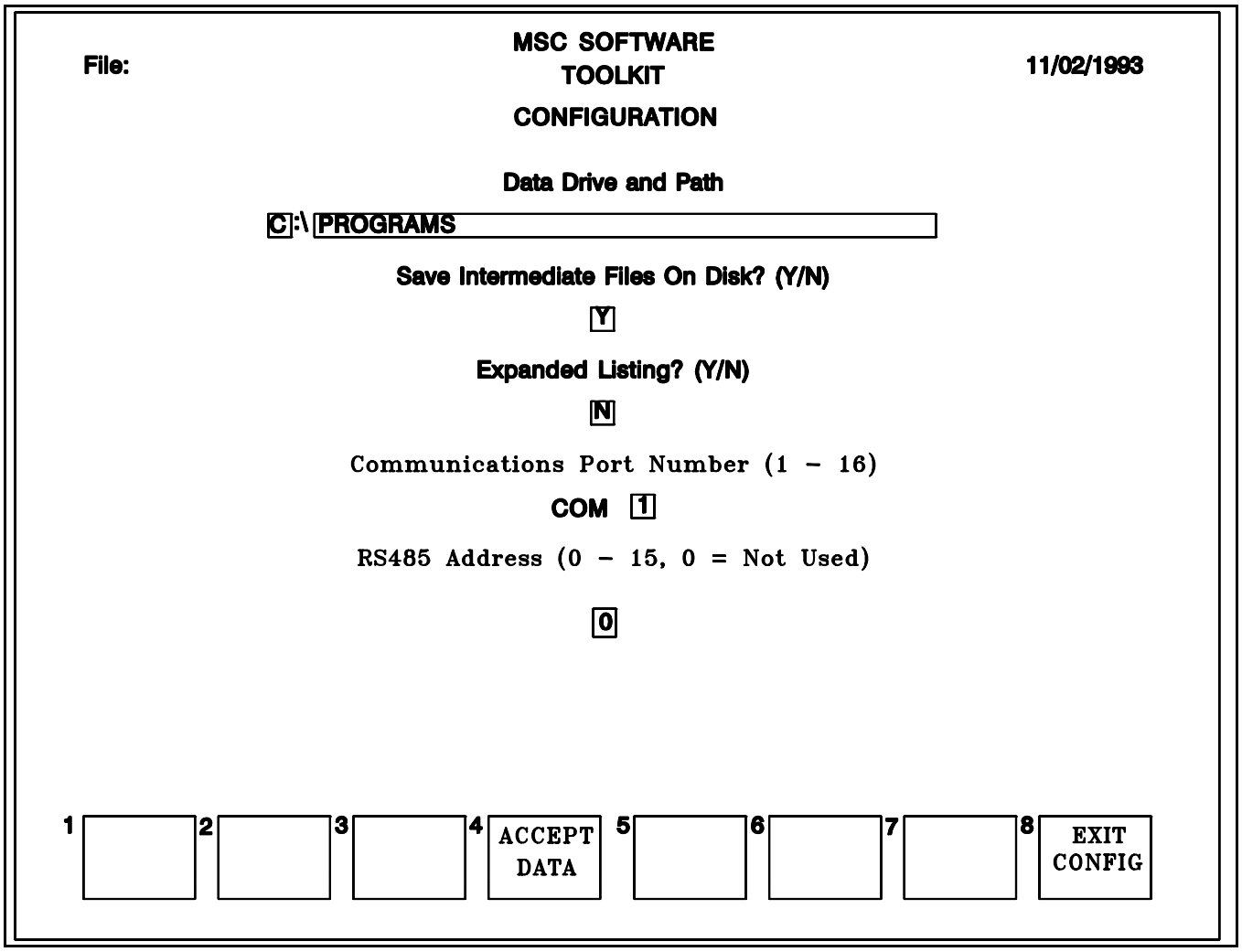


Figure 2.4 - SYSTEM CONFIGURATION SCREEN

The configuration characteristics are as follows:

1. Disk, directory and path selection for program file storage and retrieval.

   The disk selected must be a valid disk on your computer system. Directory and path selection are optional, but if chosen, they too must be valid directories on your computer system. Refer to your computer handbooks or manual for more information on the valid disk and directory names for your computer.

2. Save Intermediate Files on Disk (Y/N)

   During program compilation, the Toolkit creates a number of temporary files which are used during program debugging. If this option is set to **N**, these temporary files are kept only in memory. This speeds up the compilation process. However, if you exit the Toolkit, and at a later time, wish to further test your Macroprogram, the compilation process must be repeated to recreate the temporary files. Setting the option to **Y** causes the temporary files to be written to disk, as well as kept in memory.

3. Expanded Listing (Y/N)

   During program compilation, the development system will, if requested, produce a detailed program listing file. This listing includes tables for the following: all constants, all data variables, all equates, and all instruction labels. The listing also shows all the source program instructions and their compiled "machine" codes. This listing is not required for successful program execution and considerable savings in compilation time and disk storage space can be realized if the listing is omitted.

4. Communications Port Number (1 - 16)

   Many computers have more than one serial communications port available. You may select the appropriate port for your system by entering the appropriate port number. The Toolkit will automatically configure the communications port to the proper settings. Refer to your computer handbook or manual for information on serial communications with your computer. If you are using a port other than 1 or 2, please contact IIS for an application note on non-standard communications ports.

5. RS485 Address (0 - 15)

   If you are using RS485 multi-drop communications, enter the device address that was set up on the MSC controller card. Refer to the instruction book for the type of MSC controller being used.

You can move from field to field on the CONFIGURATION screen by using the **Enter** key and/or the **TAB** and **SHIFT/TAB** keys. Once you have made the appropriate entries, you can press the ACCEPT DATA softkey and these selections will be recorded onto your disk. If you decide not to change these selections, you can press the EXIT CONFIG softkey and the system will return to the Macroprogram Development system selection menu.

2.3.2 COLOR CONFIGURATION

The Color Configuration option of the Toolkit allows the user to customize the colors and/or video enhancements displayed on the computer screen. Selecting this option shows the form displayed in Figure 2.5. The left side of the screen will display eight lines of **A**'s, illustrating the various enhancements and colors that your system is capable of displaying. The right side of the screen represents a miniature Toolkit screen. As you choose various display options, this side of the screen will change accordingly.

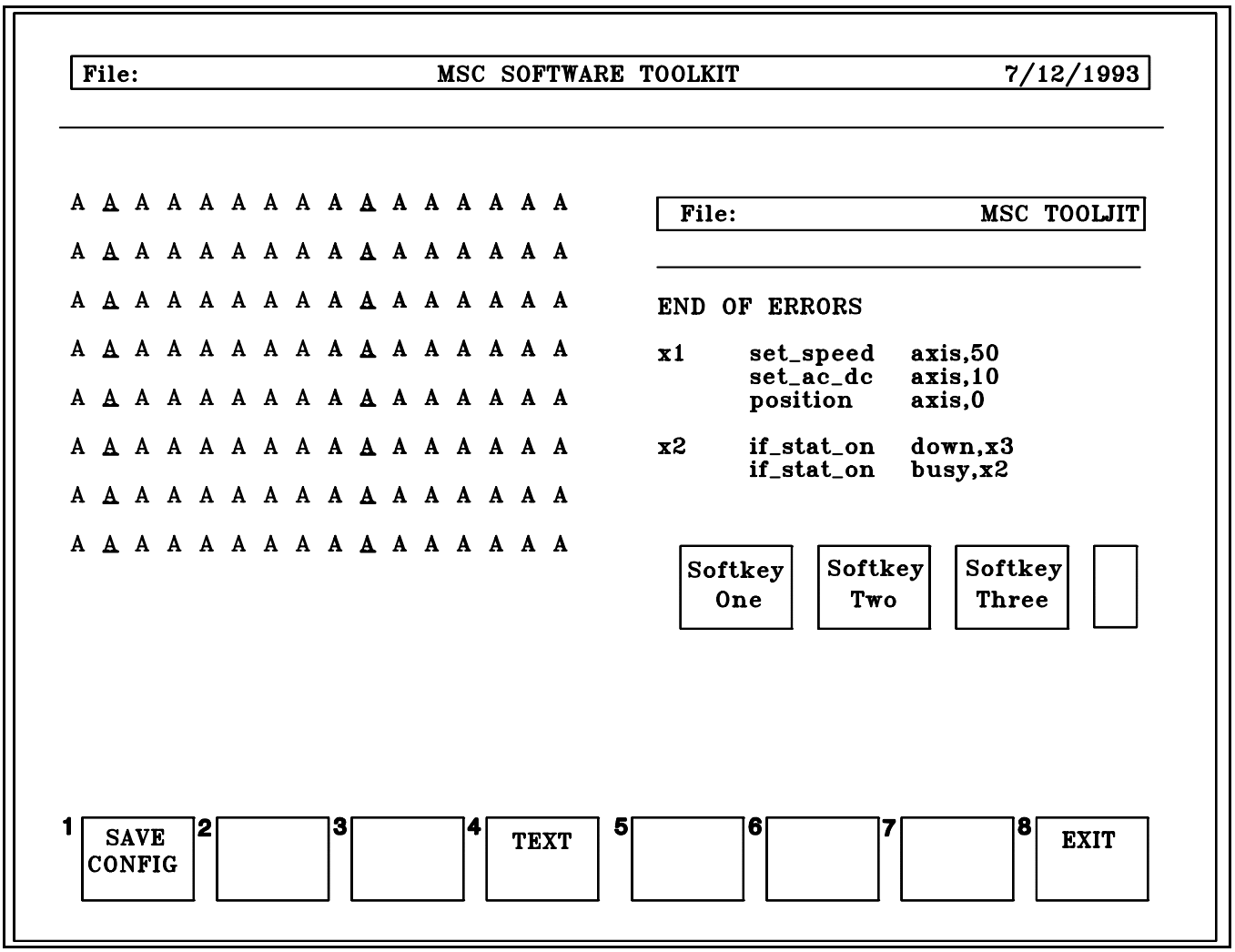


Figure 2.5 - COLOR CONFIGURATION SCREEN

There are six different options which can be set using Color Configuration:

TEXT Controls the enhancement for lines 4 through 21 of the screen.

FUNCTION KEYS Controls the enhancement for the function key labels displayed at the bottom of the screen.

LINE 1 Controls the enhancement for the top line of the screen. This line normally contains the name of the program being developed, the Toolkit banner, and the date.

LINE 2 Controls the enhancement for the second line of the screen. This line is normally used for system messages and user input.

LINE 3 Controls the enhancement for the third line of the screen. This line is normally used for system messages and user input.

CURSOR Controls the enhancement for the higlight bar which appears on certain screens and is moved about using the cursor arrow keys.

To change the enhancement for a particular item, press function key F4 until its label displays the option name. Notice that, on the left side of the screen, a small arrow appears over the **A** which has the current enhancement. Use the cursor arrow keys to move the small arrow to the desired enhancement. Repeat this step for each change you wish to make. When you are satisfied with your choices, press **F1**, **SAVE CONFIG**. Your changes will then be recorded.

WARNING

It is possible to change enhancements so that some or all of the Toolkit screen will not be displayed.

3.0 MPEDIT - MACRO PROGRAM EDITOR

3.1 INTRODUCTION

MPEDIT is a full screen program editor which provides the ability to enter Macroprograms into computer memory and to make modifications simply and effectively.

3.2 FILE MAINTENANCE

To use MPEDIT, it is necessary either to select a currently existing program file, or choose to create a new one. These functions are performed by selecting the **Create/Select File** function from the main menu of the Toolkit. This function will cause a list of any Macroprograms resident on the current data path to be displayed. At this point, you may:

1. Choose an existing Macroprogram file by moving the highlight bar to the desired program and pressing the **Enter** key.
2. Create a new Macroprogram by pressing F4. The Toolkit then asks for the desired file name. Type the file name, followed by the **Enter** key. File names can consists of letters and numbers. Do not use blank spaces in file names. MPEDIT automatically assigns the **.prg** extension to the file name you choose.
3. Delete an existing Macroprogram file by moving the highlight bar to the desired program and pressing F2. The system will prompt you to be sure you wish to delete the file. Answer by pressing the **Y** key or the **N** key accordingly.

After you create or select a file for editing, the Toolkit takes a few seconds to create or read the file and then returns to the main Toolkit menu. At this point, you may choose the MPEDIT option from the menu and press the **Enter** key.

3.3 THE EDITING PROCESS

Entering Macroprograms with MPEDIT is much like using the computer keyboard as a typewriter. Just type the program line as you would like it to appear, and press the **Enter** key when each line is complete. Note that as you type the line, it appears in the highlight bar on your screen. When the return key is pressed, the highlight bar moves down to the next line and the line you typed appears normally. Entering new lines or making corrections always occurs within the highlight bar.

Within this manual, the location of the highlight bar will be referred to as the current line. Within the current line, there will be a blinking underline character, referred to as the cursor.

When typing a line, corrections can be made by backspacing until the cursor is under the error, and retyping the correct information. When making corrections, it is NOT necessary to press the **Enter** key after the correction.

3.4 SPECIAL KEYBOARD KEYS

Table 3.1 - SPECIAL FUNCTION KEY FEATURES

KEY	FUNCTION
Down Arrow	moves current line bar down one line
PgDn	display next page of text
Up Arrow	moves current line bar up one line
PgUp	display previous page of text
Right Arrow	moves cursor one space to the right
Left Arrow	moves cursor one space to the left
SHIFT/DELETE	deletes line at the cursor
SHIFT/INSERT I	nserts a blank line
HOME	moves to the top of the program
END	moves to the bottom of the program
INSERT	turns on insert character mode. Characters subsequently typed will be inserted at the cursor position. Press INSERT again to turn off insert mode
DEL	the character at the current cursor location is deleted
F9	changes from one set of function key definitions to another
TAB	moves the cursor to the next tab stop, tab stops are every eight columns
SHIFT/TAB	moves the cursor to the previous tab stop

Certain keyboard keys aid in the editing process by performing special functions. These keys, along with their functions, are described in Table 3-1.

NOTE

On certain extended function keyboards, a separate set of cursor control and page control function keys are provided. Usually, the SHIFT/INSERT and SHIFT/DELETE functions do NOT work with this second set of keys. This is because the status of the SHIFT key is not transmitted to the computer for these keys.

3.5 FUNCTION KEY FUNCTIONS

The function key functions provided by MPEDIT enable you to perform more sophisticated editing with only a few keystrokes. Each function key function is described in detail in this section.

3.5.1 COPY/EXTRACT

This key provides a means of "snipping" out a piece of text which can be later added to a different location. To use COPY/EXTRACT, move the current line to the first line you wish to copy or extract and press the **COPY/EXTRACT** function key. The function key labels will change so that key 1 says **COPY TO HERE**, key 4 says **EXTRACT TO HERE** and key 8 says **EXIT**. Using the down arrow, move the current line to the end of the area you wish to copy. Notice that all the lines in the selected range are highlighted. When you reach the last line of the copy area, press the key marked **COPY TO HERE** or the key labeled **EXTRACT TO HERE**. The display will return to normal. A copy of the lines within the selected range has now been placed in a holding area so that the copy may be "pasted" into the program in another location.

The copy portion of the **COPY/EXTRACT** function also works with the PREVIEW FILE function, allowing you to copy lines from another file into the one being edited.

3.5.2 PASTE

This function allows a section of text which has been "snipped" out using the COPY/EXTRACT function to be placed at any desired location in the program. Move the current line bar to the desired location and press the PASTE function key. The copied lines will be inserted into the program above the current line bar.

3.5.3 SEARCH

This function allows you to search throughout the text of your program for a particular sequence of characters. Pressing this function key causes MPEDIT to request the character string you wish to search for. Type the characters and press the **Enter** key. Searching begins at the line following the current line and continues until the desired string is found. If the requested string is found, the current line is moved to the line containing the string. If the requested string is not located, the message "Not Found" is displayed.

Subsequent search requests will cause the previous search string to the displayed under the message "Enter Search String". Simply press the **Enter** key to search for the same string again. To search for a new string, type it over the top of the old string. Be sure to blank out any excess characters from the old string.

3.5.4 REPLACE

The replace function works in a manner similar to the search function, except that the replace function allows you to replace the search string with a new sequence of characters. For example, the replace function could be used to change all occurrences of the string **"Axis_1"** to **"Axis_2"**.

Pressing this function key causes MPEDIT to first ask for the search string and then for the replace string. Searching then begins. If the search string is located, four function key options are presented:

(F4) ALL - Changes all occurrences of the search string to the replace string without intervention.

(F5) REPLACE - Changes this occurrence and begins searching for the next one.

(F6) SKIP - leaves this occurrence unchanged and begins searching for the next one.

(F8) RETURN TO EDIT - Returns to the editing mode.

3.5.5 APPEND FILE

This function will add the contents of a specified macroprogram source file to the end of the program currently being edited. Pressing this key will cause MPEDIT to ask for the name of the program file to be appended. Type the name and press the **Enter** key. The BROWSE FILES capability can also be used with APPEND.

3.5.6 FUNCTION DESCRIPTION

Pressing this function key will display the current list of Macroprogram instructions. To view the details of an individual instruction, position the highlight by using the up and down arrow keys to the desired instruction and press the **Enter** key. To exit from these descriptions, press the EXIT DESCRIPT function key.

3.5.7 QUIT NO SAVE

Pressing this function key exits from this session of editing and returns to the Macroprogram Development System selection menu without saving changes made during this edit session. The message

WARNING: FILE WILL NOT BE SAVED. ARE YOU SURE?

will be displayed. If a **Y** is typed, the original program file is reloaded from the disk, and changes are discarded. An **N** response will return control to the edit mode.

3.5.8 EXIT AND SAVE

Pressing this key also exits from this session of editing. However, all changes made during this edit session will be saved onto disk. Before rewriting the edited file to disk, the system prompts for a file name. The current name is shown as a default. To save the new program under the same file name as the original, press the **Enter** key. If you wish to assign a new file name, type the new name over the top of the old name.

When the Toolkit saves the file, it renames the original copy from "filename".PRG to "filename".BAK. In this way you have at least one backup copy.

If any changes were made to the program during the editing session, the Toolkit will automatically invoke MPCPL to compile the program.

3.5.9 GO TO LINE

Moves the highlight bar to a specific line number. This function can be used with the error printout from the MSC Macroprogram Compiler to quickly move to a specific line.

3.5.10 DISP ERRORS

When the MSC Macroprogram Compiler is used to compile a Macroprogram, it produces a list of any errors found. MPEDIT will read this list if it is present, and display the message "Errors Found" on the screen when it starts up. If you wish to review the error list, press the **DISP ERRORS** function key. This function key does not appear if no error list exists for the program being edited.

3.5.11 NEXT ERROR

This function key is used after a Macroprogram has been compiled. It functions like the search key, except that is causes MPEDIT to move to the next line containing an error. Note that lines containing errors will normally be shown as blinking on the screen (some monitors are incapable of displaying blinking characters).

This function key does not appear if no error file exists or the error file is empty for the program being edited.

3.5.12 INDENT

This function will align the instructions in a Macroprogram into formatted columns. Labels in column 1, instructions in column 16, parameters in column 31 and comments in column 61. Column formatting is **NOT** required for successful program compilation and execution.

3.5.13 PREVIEW FILE

This function provides a means of reviewing the contents of one file while editing another. A section of the reviewed file may be copied into the file being edited in a manner similar to the COPY/EXTRACT function. PREVIEW FILE can be useful to inspect other macro programs similar to the one being edited. To use the preview function, press the corresponding function key. MPEDIT will respond by asking for a file name to preview. Type the desired file name and press Return. Note that the BROWSE FILES function can be used to select a file for previewing. It is also possible to use the COPY portion of COPY/EXTRACT to copy lines from a file being previewed into the program currently being edited.

3.5.14 HELP

Pressing this key will display an abbreviated screen listing of the functions of special keyboard keys. By using the **PgDn** and **PgUp** keyboard keys, you can view further information concerning MSC flags, I/O definitions, and status words. Press the **EXIT** function key to return to the editing screen.

3.5.15 OOPS

This function allows the **last** line of text deleted to be added back. This function is only available in the IBM PC version and works only if the line was deleted using the **SHIFT/DEL** keyboard keys.

3.5.16 SAVE POSITION

This function allows you to mark a line in the file to return to should you wish to perform other functions on the remainder of the file. This function is used in conjunction with the BACK TO POSITION function. An example of the use of this feature is: You are presently at the fiftieth line of a given file and you now wish to search through the entire file for a particular character string. Following the search however, you wish return to the fiftieth line. You would press the SAVE POSITION function key, then proceed to do your search. When the search was completed, you would press the BACK TO POSITION function key.

3.5.17 BACK TO POSITION

This function is used in conjunction with the SAVE POSITION and allows you to reposition the current line indicator to the previously marked (SAVED) line. Refer to Section 3.5.16 above for an example of the use of this function key.

3.5.18 CLEAR LINE

This function allows you to erase a portion of a line or even an entire line. Position the cursor to the desired character. Now press this function key and all characters from the cursor to the right end of the line will be erased. This feature is only available in the IBM PC version.

3.5.19 CLEAR DISPLAY

This function key allows you to erase the entire edit work area. Be very careful when using this feature as there is no way to restore the work area other than using the **QUIT NO SAVE** function.

4.0 MPCPL - MACRO PROGRAM COMPILER

4.1 INTRODUCTION

The Macroprogram Compiler (MPCPL) converts MSC Macroprograms from human readable instructions to numeric codes which can be interpreted and acted upon by the MSC. The Compiler is automatically executed on exiting from MPEDIT, or prior to running the debugger if the system cannot find the machine control program file for the program you are editing.

In addition, MPCPL produces information which is used by the Macroprogram Editor (MPEDIT) and Macroprogram Debugger (MPDEBUG) to simplify the programming and testing processes. Optionally, MPCPL may also produce a detailed program listing file.

4.2 USING MPCPL

MPCPL is invoked automatically by the Toolkit whenever compilation is necessary.

4.3 MACROPROGRAM LINE FORMAT

Macroprogram lines can be made of up to four (4) parts, as shown in figure 4.1. These parts are:

Figure 4.1 - MACROPROGRAM INSTRUCTION FORMAT

label instruction *parameters* *comment*

1 .Label - A label can be from one (1) to twelve (12) characters in length, and may consists 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 Macroprogram 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.

4.4 MACRO COMPILER OUTPUT FORMAT

MPCPL produces the following outputs:

1. Listing File (Optional) - The listing 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 macroprogram. 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. Error Summary File - This automatically created file contains a summary of any errors detected in the compilation process. It is used by MPEDIT to assist in locating and correcting errors.

3. Symbol Table File - This optional diskette file contains all symbol, label, and data definitions. It is used by MPDEBUG to assist in the debugging process.

4. Binary Program File - This file contains the compiled program in MSC Machine Instruction Format. It can be transmitted to the MSC by MPDEBUG (or other programs), and the MSC can perform the instructions contained in it.

5. Debugger File - This file contains information used by MPDEBUG to translate trace information into source program lines.

4.5 SPECIAL MPCPL INSTRUCTIONS

MPCPL recognizes certain instructions which are not acted upon by the MSC, but simplify the process of defining data and constants. Chapter Six of this manual contains a description of these statements.

5.0 MPDEBUG - MACRO PROGRAM DEBUGGER

5.1 INTRODUCTION

The Macroprogram Debugger (MPDEBUG) provides a means of transmitting a compiled MSC Macroprogram from a personal computer to the MSC and monitoring its execution. MPDEBUG provides the ability to:

1. Inspect and modify values in the Macroprogram data area.
2. Test and modify the status of MSC flags.
3. Trace program flow and execution.
4. Test the MSC Macroprogram status word.
5 .Inspect the status of each motor axis card.
6. Review program source file, listing file, symbol table, error file.
7. Transfer the resident Macroprogram to an EPROM chip via the PROM POCKET.
8. Place each Axis Controller in the MSC System Unit into Test Mode.

MPDEBUG is able to reference information using the symbols which were used when the Macroprogram being debugged was written. This simplifies the debug process by eliminating the need to converse in decimal or hexadecimal numbers.

This chapter describes the use and features of MPDEBUG.

NOTE

The MSC controller **must** be connected to your computer before selecting MPDEBUG from the Macroprogram Development system selection menu. See IB11C001, the MSC-850 System Unit, for cable information.

5.2 USING MPDEBUG

MPDEBUG is accessed from the main Toolkit menu by highlighting the appropriate option and pressing the **Enter** key. MPDEBUG will verify that an MSC controller is connected to the serial port specified in System Configuration, and if so, will continue to the main MPDEBUG screen.

The top line of the main MPDEBUG screen, shows the name of the program currently residing in the MSC (if any), and the current status of the MSC. Initially, this status is STARTUP. The current status of the MSC and its card configuration will be displayed on the lower portion of the display. (See Section 5.4.1.6 on MACRO STATUS below).

5.3 MPDEBUG CONVENTIONS

Many MPDEBUG commands request data values, flag numbers, etc. These values may be typed in either numerically or as valid labels as defined in the Macroprogram source. For example, if the Macroprogram source defined axis 1 as "rotor", the word "rotor" could be used as a response when requesting the status of axis 1. MPDEBUG Data values may be entered as numbers or as valid arithmetic expressions. For example, to express 3.5 motor turns, where 4096 bits represents one motor turn, the expression

3.5*4096

could be entered. MPDEBUG will perform the computation and use the result.

One function of MPDEBUG is available at all times. This is **STOP PROGRAM**. The **STOP PROGRAM** function commands the MSC to stop executing the current Macroprogram and to send a **f_decel** instruction to all Controller cards.

5.4 MPDEBUG FUNCTIONS

The functions of MPDEBUG are broken into 6 sections. Each section and the functions it provides is described below. To select one of the major function areas, press its associated function key. To return to the main MPDEBUG menu, press the **F9** function key.

5.4.1 READ FUNCTIONS

The READ functions provide a means of retrieving information from the MSC and displaying it on the computer's screen.

5.4.1.1 READ DATA

This function reads data from the MSC data area. Selecting **READ DATA**, will cause MPDEBUG to ask for a data address. Enter the address symbolically or numerically. MPDEBUG will read the data from the appropriate location in the MSC and display its value in decimal and in hexadecimal. You may continue in this function by typing another address, or exit by pressing the **EXIT READ** function key.

5.4.1.2 READ DATA CONTINUOUS

This function performs just like **READ DATA** except that the specified location is read repeatedly and displayed on the screen until the **EXIT READ** function key is pressed. Up to four data locations may be selected to be read continuously.

5.4.1.3 READ FLAG

This function will retrieve the status (ON = 1, OFF = 0) of the desired flag from the MSC. The flag number may be entered as a number or as a label. See Chapter 8 for further discussion of flags.

5.4.1.4 READ FLAG CONTINUOUS

This function will continuously read and display the status of a specified MSC flag. Press the **EXIT READ** function key to stop this function. Up to four flags may be read continuously.

5.4.1.5 AXIS STATUS

This function continuously reads the status bits for the desired axis (Controller) and displays appropriate messages if any of the status bits are set. If no status bits are set, no messages will appear.

5.4.1.6 MACRO STATUS

This command reads and displays the status information from the MSC System Unit. The screen shows the name of the currently loaded Macroprogram and the date and time this Macroprogram was compiled. The lower portion of the screen displays the name and software revision levels for each Controller in the MSC System unit. A description of any bits set in the Macroprogram Status Word also appears. The MACRO STATUS function does NOT continuously update the screen.

5.4.2 WRITE FUNCTIONS

The Write functions provide a means for changing Macroprogram data locations and flags.

5.4.2.1 WRITE DATA

This function writes data into the MSC data area. MPDEBUG will ask first for a data address. Enter the desired address symbolically or numerically. MPDEBUG will then ask for a data value. Enter this value as a single number or an arithmetic expression.

5.4.2.2 WRITE DATA CONTINUOUS

This function allows continued updating of a particular MSC data location. It functions similarly to **WRITE DATA** except that after sending the specified value to the MSC, MPDEBUG will immediately request another value to be written. Exit the **WRITE CONTINUOUS** function by pressing the function key labeled **EXIT WRITE**.

5.4.2.3 WRITE FLAG

This function will set or clear the desired flag. The flag number is entered either as a symbol or a number. To change the state of the specified flag, press the appropriately labeled function key. Alternately, type an **S** to set the flag or a **C** to clear it. If any other character is typed, the status of the flag is not changed.

NOTE

Writing to an I/O flag assigns that flag as an output flag and will no longer respond to an external input.

5.4.2.4 WRITE FLAG CONTINUOUS

This function allows the user to continuously change the state of a selected flag. Exit this function by pressing **EXIT WRITE**.

5.4.3 TRACE FUNCTIONS

The MSC has the ability to "remember" the last 112 instructions that it executed. MPDEBUG can tell the MSC when to start and stop this "memory" and can read and display the values contained in it. These **TRACE** functions are described below.

The trace functions request two address values in order to know how to perform the trace. The first address requested is the focal point for the trace. (The prompt message will depend on the trace mode being used.) The second address requested is the **START TRACE AT** address. The MSC will not activate its trace memory until this address is encountered. Note that simply pressing the **Enter** key in response to the prompt address will cause the MSC to begin the trace immediately. This feature can be used as shown in the following example. Suppose you wish to trace execution of a subroutine labeled **check_status**, but only after the instruction labeled **run_machine** had executed. The proper key sequence would be:

1. Press the **TRACE AFTER** function key.

2. In response to the **TRACE AFTER ADDRESS:** prompt, enter **check_status**.

3. In response to the **START TRACE AT (PRESS ENTER TO START IMMEDIATELY)** prompt, enter **run_machine**.

This sequence of steps tells the MSC to watch for the execution of the instruction at the label **run_machine**, and, after encountering it, to start watching for the instruction labeled **check_status**. When the **check_status** address is encountered, the next 112 instructions will be saved, transmitted to MPDEBUG, and displayed on the screen.

Address values are easily entered by using the appropriate Macroprogram statement label. A list of labels is provided as part of the Macroprogram listing, and can be viewed using the **VIEW SYMBOLS** or the **VIEW SOURCE** function of MPDEBUG. Trace points between statement labels may be referenced by their numeric address. These addresses can be determined using the **VIEW SOURCE** function.

5.4.3.1 TRACE BEFORE

This function allows you to view the Macroprogram instructions that were executed before reaching the specified label.

5.4.3.2 TRACE ABOUT

This function will cause the instructions just before and just after the specified label to be traced.

5.4.3.3 TRACE AFTER

This function will trace the Macroprogram instructions executed after reaching the specified label.

5.4.3.4 TRACE CURRENT

This function commands the MSC to trace the next 112 instructions executed. It can be useful if you are not sure which part of your program is currently being executed.

5.4.3.5 STOP TRACE

Stop Trace commands the MSC to stop the currently active trace. It would most often be used when the specified trace does not finish naturally. For example, if you set up a trace after the label **set_zero**, and the program never got to the specified label, attempts to **READ TRACE** would produce only the message "Trace still running". **STOP TRACE** would then be used to tell the MSC to stop performing the trace. When the **STOP TRACE** function is used, the trace buffer may not contain meaningful information.

5.4.3.6 READ TRACE

This function reads and displays the results of the last trace executed by the MSC. The trace screen may be scrolled up and down by using the cursor keys, the **PgDn** and **PgUp** keys, and the **Home** key. If the trace last specified has not completed, the message "Trace still running" will be displayed.

5.4.4 MSC COMMANDS

This group of commands deals with transmitting Macroprograms to the MSC, starting program execution, and setting the MSC into various modes of operation.

5.4.4.1 STOP PROGRAM

This function causes the currently executing Macroprogram to be stopped. An **f_decel** command is sent to each Controller card. NOTE: The **STOP PROGRAM** function key is available on all MPDEBUG menus.

5.4.4.2 RESET

This function sends a serial RESET command to the MSC. The RESET will clear all flags, reset all Controller cards, and erase any Macroprogram and data resident in the MSC. This function must be preceded by a **STOP PROGRAM** function.

5.4.4.3 SEND PROGRAM

This function will transmit the currently loaded Macroprogram to the MSC. A **RESET** function must be performed prior to using the **SEND PROGRAM** function.

5.4.4.4 START PROGRAM

This function will begin execution of the Macroprogram currently resident in the MSC. This function will not execute if no program has been sent to the MSC.

5.4.4.5 PROM OPTIONS

This function provides access to a sub-menu used to copy the Macroprogram currently resident in the MSC to an EPROM. NOTE: It is possible for the program resident in the MSC to be different from the program resident in the Toolkit.

To copy the resident program to EPROM, perform the following steps:

1. If the resident program is running, press **F1**, **STOP PROGRAM**.

2. The PROGRAM MODE switch on the MSC PROM Pocket must be set to the **Program** position(the MSC-850/32 controller does not have a PROGRAM MODE switch).

3. Insert the EPROM to be programmed in the PROM Pocket. The EPROM must be an INTEL D27256-1 UV erasable EPROM, or equivalent.

4. Press function key **F2**, **BURN PROM**. The message

BURNING PROM.....

will appear at the top of the screen. Depending on the size of the program being copied to EPROM, the message may remain on the display for up to 2 or 3 minutes.

While the EPROM is being programmed, the green LED on the PROM Pocket will be illuminated. When using the MSC-250 or MSC-850/32 controllers, the status display will indicate that the EPROM is being programmed.

On successful completion of the programming operation, the message

PROM Operation Successful

will appear. If an error is detected during programming, refer to Table 15.1 for an explanation of the error code.

5.4.4.6 TEST MODE

This function places each axis in the MSC System Unit into the test mode. Refer to the MSC System Manual Test Procedure section.

5.4.4.7 SET AUTOSTART

This function sets the **AUTOSTART** bit in the MSC status word and, if necessary, begins execution of the Macroprogram residing in MSC non-volatile memory. The MSC operating system firmware tests the **AUTOSTART** bit on power-up. If it is turned on, the Macroprogram currently stored in non-volatile memory will be executed.

5.4.5 VIEW FUNCTIONS

The view functions provide a means of displaying various pertinent files on the screen during the editing process. You may move from place to place in the file being displayed using the **PgDn** and **PgUp** keys.

5.4.5.1 SOURCE

This function displays the source program file (created by MPEDIT) for the program currently being debugged. Program addresses are displayed for each instruction in the program. This is a convenient way to determine an address value to use with trace functions.

5.4.5.2 EQUATE TABLE

This function displays all constants defined using the **equ** compiler directive.

5.4.5.3 LABEL TABLE

This function displays all program labels and their corresponding addresses.

5.4.5.4 CONSTANTS

This function displays the constants defined in the Macroprogram along with their data addresses.

5.4.5.5 DATA TABLE

This function displays all data locations defined using compiler directives such as **begin_data**, **dim**, or **integer**. The format for this display shows the variable name, its decimal address, and its hexadecimal address.

5.4.5.6 LAST TRACE

This function displays the information from the last trace executed by the MSC. If no tracing has been done, an appropriate message is displayed.

5.4.6 BLOCK FUNCTIONS

Block commands are used to read and write data areas. They are helpful when a large amount of continuous data has to be modified or viewed.

When the **BLOCK** function key is pressed, the system responds by showing three additional function keys; **STOP PROGRAM**, **READ DATA**, and **WRITE DATA**.

5.4.6.1 READ DATA

This function key allows you to select the beginning of a data area to be viewed. The data address is entered and the system will display the current contents of each of a series of data locations. By pressing the **PgDn** key, you can view the next group of continuous data locations. To exit BLOCK READ, press the EXIT BLK READ function key.

5.4.6.2 WRITE DATA

This function key allows you to select the beginning of a data area to be viewed and, optionally, modified. The desired data address is entered and the system displays the Macroprogram data area beginning with the selected data address. To modify a data value, use the cursor keys to move the highlight to the desired variable. Type the new value and press the enter key. To change between decimal and hexadecimal format, press the appropriate function key. To exit BLOCK WRITE, press the EXIT BLK WRITE function key.

6.0 INTRODUCTION TO MACROPROGRAMMING LANGUAGE

6.1 BASIC CONCEPTS

A Macroprogram is an organized group of instructions that can be executed by an MSC system unit. Macroprograms reside in a non-volatile memory in the system unit.

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.

6.2 INSTRUCTION FORMAT

Macroprogram instructions are, in many ways, similar to the instructions of the BASIC computer language. Each instruction consists of a statement label, an operation, a list of parameters, and an optional comment. This basic format is illustrated in Figure 6.1.

Figure 6.1 - MACROPROGRAM INSTRUCTION FORMAT

label instruction *parameter list* *comment*

The label portion of the instruction 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.

The parameter list part of an instruction line is also not always required. However, there are usually one or more parameters for an instruction. In the example in Figure 6.2, two parameters are required.

Figure 6.2 - INSTRUCTION WITH TWO PARAMETERS

label set_speed axis_1,300 *comment*

6.3 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 Macroprogram memory space. The liberal use of comments is highly recommended. There are two ways to implement comments within a Macroprogram:

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 Macroprogram shown in Figure 6.3.

6.4 BLANK LINES

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

6.5 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 Macroprogram.

Figure 6.3 - SAMPLE MACROPROGRAM

```
LABEL       OPERATION     PARAMETERS          COMMENT

            msc_type      850
            declare       ON

pos_1       equ           81920               two turns CW
pos_2       equ           -81920              two turns CCW
rack        equ           1                   use axis 1
rack_down   equ           93                  down flag
rack_busy   equ           94                  busy flag
fault_lite  equ           0                   I/O zero

! do initialization functions

            turn_off      fault_lite          show no fault

            drive_on      rack                enable drive
            set_speed     rack,50             500 rpm speed
            set_ac_dc     rack,20             rev/sec/sec

! set a local zero at current position

            set_local     rack

! begin main program loop

restart
            position      rack,pos_1          go 2 turns CW
loop1
            if_stat_on    rack_down,fault     check error
            if_stat_on    rack_busy,loop1     wait for done

            position      rack,pos_2          go 2 turns CCW

loop2
            if_stat_on    rack_down,fault     check error
            if_stat_on    rack_busy,loop2     wait for done

            goto          restart             do it all again

fault
            turn_on       fault_lite          signal error
            sys_return
```

7.0 COMPILER DIRECTIVES

7.1 DESCRIPTION

Compiler directives are a class of instructions which simplify such operations as defining constants, data arrays and cams, or which cause the MPCPL program 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 Table 7.1.

Table 7.1 - COMPILER DIRECTIVES

Instruction	Format			Ref. Page
begin_cam	label	begin_cam		116
begin_data	label	begin_data		117
cam		cam	value,value,etc.	122
data		data	value,value,etc.	134
declare		declare	mode	135
dim	label	dim	controller#,size	138
end_cam		end_cam		146
end_data		end_data		147
equ	label	equ	value	149
integer	label	integer		203
msc_type		msc_type	system_type	217
text	label	text	"ASCII string"	291

8.0 FLAGS

8.1 DESCRIPTION

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 Macroprogram instructions dealing with flags. In fact, taken as a group, flag instructions comprise the largest subset of macroprogram 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.

Tables 8.1 through 8.9 list the flags available in the MSC family of controllers.

NOTE:

The following are special considerations when using the MSC-250:

1) The MSC-250 controller can use only two (2) I/O Expander units and therefore does not use I/O flags 48 - 63.

   Flags 64 - 71 are used as software PLS flags.

2) The motor status flags 128 - 175 are reserved for extended status indicators for axes 1 - 3 respectively on the MSC-250.

3) The motor status flags 176 - 207 are not used in the MSC-250. These **cannot** be used as additional user 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

Table 8.1 - MSC-800, MSC-850, MSC-850/32 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

Table 8.2 - MSC-250 INTERNAL STATUS FLAGS

#1	#2	#3	#4	#5	#6	#7	#8	MEANING
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	AXIS TIME-OUT
88	104	120	136	152	168	184	200	MOTOR JOGGING
89	105	121	137	153	169	185	201	MOTOR INDEXING
90	106	122	138	154	170	186	202	CALCULATING
91	107	123	139	155	171	187	203	HWI ARMED
92	108	124	140	156	172	188	204	FORCE DECEL IN PROGRESS
93	109	125	141	157	173	189	205	AXIS DOWN
94	110	126	142	158	174	190	206	AXIS BUSY
95	111	127	143	159	175	191	207	MASTER/SLAVE LOCK

Table 8.3 - ACR-850 CONTROLLER STATUS FLAGS

#1	#2	#3	#4	#5	#6	#7	#8	MEANING
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	AXIS TIME-OUT
88	104	120	136	152	168	184	200	MOTOR JOGGING
89	105	121	137	153	169	185	201	MOTOR INDEXING
90	106	122	138	154	170	186	202	CALCULATING
91	107	123	139	155	171	187	203	HWI ARMED
92	108	124	140	156	172	188	204	FORCE DECEL IN PROGRESS
93	109	125	141	157	173	189	205	AXIS DOWN
94	110	126	142	158	174	190	206	AXIS BUSY
95	111	127	143	159	175	191	207	MASTER/SLAVE LOCK

Table 8.4 - ACE-850 CONTROLLER STATUS FLAGS

#1	#2	#3	#4	#5	#6	#7	#8	MEANING
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	AXIS TIME-OUT
88	104	120	136	152	168	184	200	MOTOR JOGGING
89	105	121	137	153	169	185	201	MOTOR INDEXING
90	106	122	138	154	170	186	202	CALCULATING
91	107	123	139	155	171	187	203	NOT USED
92	108	124	140	156	172	188	204	FORCE DECEL IN PROGRESS
93	109	125	141	157	173	189	205	AXIS DOWN
94	110	126	142	158	174	190	206	AXIS BUSY
95	111	127	143	159	175	191	207	MASTER/SLAVE LOCK

Table 8.5 - MCF-850 CONTROLLER STATUS FLAGS

#1	#2	#3	#4	#5	#6	#7	#8	MEANING
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	AXIS 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	AXIS DOWN
94	110	126	142	158	174	190	206	NOT USED
95	111	127	143	159	175	191	207	NOT USED

Table 8.6 - HPL-850 CONTROLLER STATUS FLAGS

#1	#2	#3	#4	#5	#6	#7	#8	MEANING
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	AXIS 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	AXIS DOWN
94	110	126	142	158	174	190	206	NOT USED
95	111	127	143	159	175	191	207	NOT USED

Table 8.7 - ACM-850 CONTROLLER STATUS FLAGS

#1	#2	MEANING
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	MOTOR JOGGING
89	105	MOTOR INDEXING
90	106	CALCULATING
91	107	HWI ARMED
92	108	FORCE DECEL IN PROGRESS
93	109	AXIS DOWN
94	110	AXIS BUSY
95	111	MASTER/SLAVE LOCK

Table 8.8 - MSC-250 CONTROLLER STATUS FLAGS AXES 1 & 2

#3	MEANING
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	MOTOR JOGGING
121	MOTOR INDEXING
122	CALCULATING
123	NOT USED
124	FORCE DECEL IN PROGRESS
125	NOT USED
126	AXIS BUSY
127	NOT USED

Table 8.9 - MSC-250 PSEUDO CONTROLLER STATUS FLAGS

DESCRIPTION OF CONTROLLER STATUS FLAGS

FLAG	DESCRIPTION
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 +/- 180*16/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 8.1.

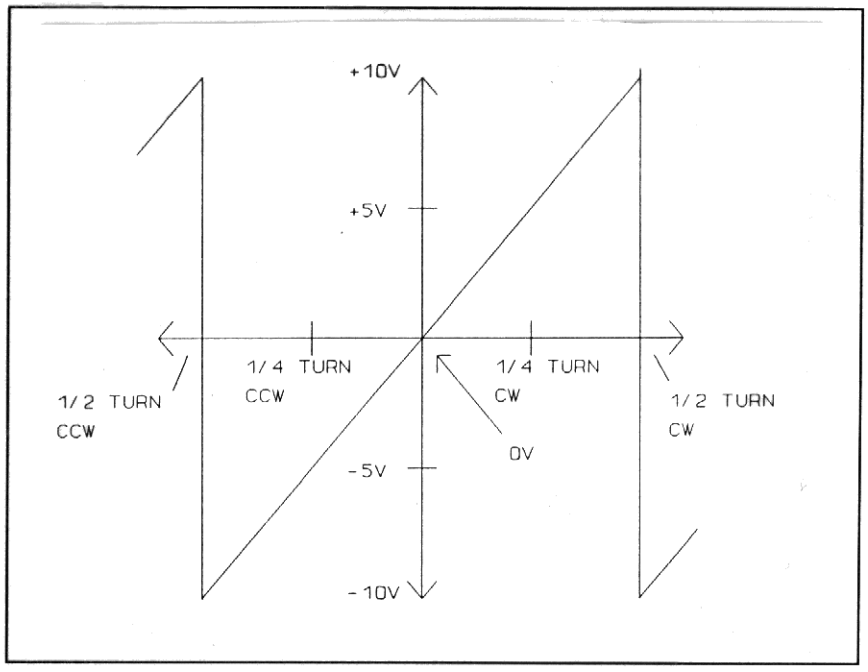


Figure 8.1 - POSITION OUTPUT vs. TRANSDUCER ANGLE

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 convertor has lost one or more of the required signals for conversion or the convertor has lost its ability to track the angle do to speeds in excess of 7200 RPM.
AXIS TIMEOUT	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.
MOTOR 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.
MOTOR INDEXING	This flag will be set when a 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.
HWI ARMED	The 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.
AXIS DOWN	This flag will be set if a following error or axis timeout is encountered (following errors and axis timeouts are described above).

AXIS 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 an **AXIS DOWN** condition while the **AXIS BUSY** flag is set, **AXIS 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 Macroprogram instruction for the specified axis controller. The following Macroprogram 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.**

8.2 TIMERS

The 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.

8.3 FLAG INSTRUCTIONS

Instructions for operating on and testing flags are summarized in Table 8.10.

Table 8.10 - FLAG INSTRUCTIONS

Instruction	Format			Ref. Page
blk_io_in	*label*	blk_io_in	input flag#,variable	118
blk_io_out	*label*	blk_io_out	output flag#,variable	119
clr_flag	*label*	clr_flag	user flag#	129
get_status	*label*	get_status	controller#	179
if_flag_off	*label*	if_flag_off	user_flag#,address_label	190
if_flag_on	*label*	if_flag_on	user_flag#,address_label	191
if_io_off	*label*	if_io_off	I/O flag#,address_label	192
if_io_on	*label*	if_io_on	I/O flag#,address_label	193
if_stat_off	*label*	if_stat_off	status_flag#,address_label	195
if_stat_on	*label*	if_stat_on	status_flag#,address_label	196
if_tmr_off	*label*	if_tmr_off	timer_flag#,address_label	197
if_tmr_on	*label*	if_tmr_on	timer_flag#,address_label	198
set_flag	*label*	set_flag	user_flag	244
set_tmr	*label*	set_tmr	timer_flag#,ticks	278
turn_off	*label*	turn_off	I/O flag#	294
turn_on	*label*	turn_on	I/O flag#	295

9.0 ARITHMETIC INSTRUCTIONS

9.1 OVERVIEW

The Macroprogramming language provides several types of arithmetic functions, 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.

9.2 INTEGER ARITHMETIC

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

```
let  res=var1 op var2
```

where **res** is the result of the operation, **var1** and **var2** are the variables to be operated on, and **op** is the operation to be performed. Note that multiple operations in a single **let** statement are NOT allowed. For example,

```
let  a=b+c
```

is acceptable, but

```
let  a=b+c/d
```

is not. Calculations requiring multiple operations must be performed in multiple **let** statements.

Spaces between variable names and operators should not be used.

9.3 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. See Chapter 18 for an explanation of 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 Macroprogram, 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

```
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

```
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.

9.4 BYTE OPERATIONS

Macroprogramming Language provides a special instruction 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

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

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

```
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. Note: The **let_byte** instruction treats byte information as unsigned.

9.5 BIT ORIENTED OPERATIONS

Macroprogramming Language provides instructions 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 and for programmable limit switches. 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.

9.6 BUILT IN ARITHMETIC FUNCTIONS

Macroprogramming Language supports certain built in arithmetic functions for commonly required calculations. These functions are listed in Table 9.1. The format for usage of these functions is

```
let  ans=fn(x)
```

where fn represents the function name.

Table 9.1 - BUILT IN ARITHMETIC FUNCTIONS

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.

9.7 ARITHMETIC INSTRUCTION SUMMARY

Arithmetic instructions are summarized in table 9.2.

Table 9.2 - ARITHMETIC INSTRUCTIONS

Instruction	Format			Ref. Page
clr_bit	*label*	clr_bit	bit#,variable	128
let_byte	*label*	let_byte	destination=source	209
let	*label*	let	variable=op1 *operation op1*	207, 208
set_bit	*label*	set_bit	bit#,variable	242

10.0 PROGRAM FLOW INSTRUCTIONS

10.1 DESCRIPTION

Normally, Macroprogram 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.

Macroprogramming Language provides 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.

10.2 BRANCHING INSTRUCTIONS

Macroprogramming Language provides 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 can be described verbally as follows:

1. Test the specified condition.
2. If the specified condition is true, transfer program control to the specified address.
3. If the specified 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.

10.3 SUBROUTINE CONTROL

When designing a Macroprogram, 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 Macroprogram. 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 may then be 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.

10.4 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.

A sample **select** group is shown in Figure 10.1.

```
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 10.1 - TYPICAL SELECT STATEMENT GROUP

A Macroprogram may contain up to 30 **select** statement groups. Each group may contain up to 100 **case** instructions.

10.5 PROGRAM FLOW INSTRUCTION SUMMARY

Program flow instructions are summarized in Table 10.1.

Table 10.1 - PROGRAM FLOW INSTRUCTIONS

Instruction	Format			Ref. Page
case	*label*	case	num	125
default	*label*	default		136
end_select	*label*	end_select		148
exit_select	*label*	exit_select		151
gosub	*label*	gosub	subroutine label	184
goto	*label*	goto	address_label	185
if	*label*	if	var1 op var2,address_label	186
if_bit_clr	*label*	if_bit_clr	bit#,variable,address_label	187
if_bit_set	*label*	if_bit_set	bit#,variable,address_label	188
if_char	*label*	if_char	port#,address_label	189
if_flag_off	*label*	if_flag_off	user_flag#,address_label	190
if_flag_on	*label*	if_flag_on	user_flag#,address_label	191
if_io_off	*label*	if_io_off	I/O flag#,address_label	192
if_io_on	*label*	if_io_on	I/O flag#,address_label	193
if_no_char	*label*	if_no_char	port#,address_label	194
if_stat_off	*label*	if_stat_off	status_flag#,address_label	195
if_stat_on	*label*	if_stat_on	status_flag#,address_label	196
if_tmr_off	*label*	if_tmr_off	timer_flag#,address_label	197
if_tmr_on	*label*	if_tmr_on	timer_flag#,address_label	198
restart_at	*label*	restart_at	address_label	236
return_sub	*label*	return_sub		237
select	*label*	select	variable	239

11.0 MOTION INSTRUCTIONS

11.1 OVERVIEW

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

Master/Slave operations are covered separately in Chapter 13. The remaining classes of motion instructions are covered here.

11.2 MSC CONVENTIONS AND MOTION TERMINOLOGY

11.2.1 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.

11.2.2 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.

11.2.3 ACCELERATION DATA

Acceleration (accel/decel) data is expressed in revolutions/second^2. Accelerations may range from 2 to 800 revolutions/second^2 (1600 revolutions/second^2 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.

11.2.4 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.

11.3 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.

At power up, the ACR-850 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.

At power up, ACE-850 controllers and MSC-250 axis controllers also set theirs turns counters to zero. The low order 12 bits of its position are also set to zero. For an ACE-850, absolute position of the feedback transducer is not known until the controller is instructed to find the encoder marker pulse.

Table 11.1 lists motion preparation instructions.

Table 11.1 - MOTION PREPARATION INSTRUCTIONS

Instruction	Format			Ref. Page
clr_local	*label*	clr_local	controller#	131
digi_comp	*label*	digi_comp	controller#,gain,integral,damp	137
drive_off	*label*	drive_off	controller#	141
drive_on	*label*	drive_onc	ontroller#	142
f_decel	label	f_decel	controller#	152
find_mrk_ccw	*label*	find_mrk_ccw	controller#,counts	153
find_mrk_cw	*label*	find_mrk_cw	controller#,counts	154
find_tm_ccw	*label*	find_tm_ccw	controller#,counts	155
find_tm_cw	*label*	find_tm_cw	controller#,counts	156
master	*label*	master	controller#	142, 216
set_ac_dc	*label*	set_ac_dc	controller#,rate	240
set_acy_cnt	*label*	set_acy_cnt	controller#,count	241
set_home	*label*	set_home	controller#,offset	250
set_gl_ccw	*label*	set_gl_ccw	controller#	245, 246
set_gl_cw	*label*	set_gl_cw	controller#	247, 248
set_local	*label*	set_local	controller#	252
set_speed	*label*	set_speed	controller#,speed	251, 276
set_vgain	*label*	set_vgain	controller#,vel_gain	281

11.3.1 DIGITAL COMPENSATION

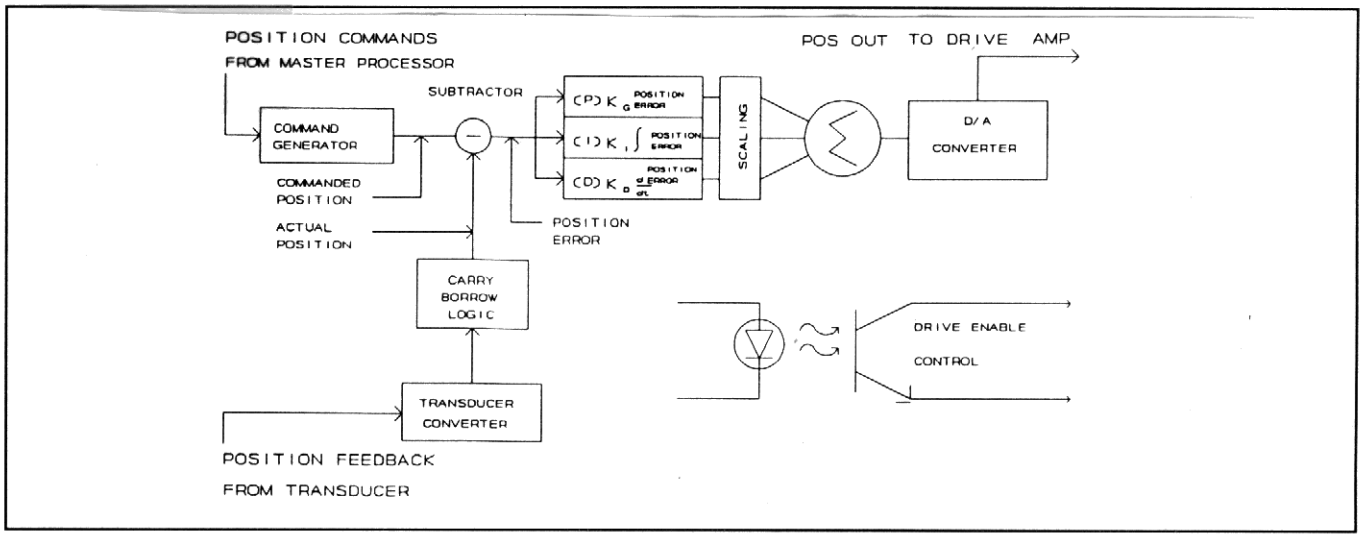


Figure 11.1 - PID Block Diagram

ACR-850, ACE-850 and MSC-250 axis controllers provide a means of digital compensation for their position loop. The Macroprogrammer 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 11.1 illustrates the integration of PID into these controllers.

Some terms used in the following paragraphs are:

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.

11.3.1.1 THE P TERM (PROPORTIONAL GAIN)

At power up the 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.

11.3.1.2 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 dependant; 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.

11.3.1.3 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 dependant; 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.

11.3.2 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.

11.4 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 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.

Velocity control instructions are summarized in Table 11.2.

Table 11.2 - VELOCITY CONTROL INSTRUCTIONS

Instruction	Format			Ref. Page
f_decel	*labe*	/f_decel	controller#	152
jog_ccw	*label*	jog_ccw	controller#	204
jog_cw	*label*	jog_cw	controller#	205
l_track_spd	*label*	l_track_spd	controller#,speed	206
track_spd	*label*	track_spd	controller#,speed	292
vel_ccw	*label*	vel_ccw	controller#	297
vel_cw	*label*	vel_cw	controller#	298

11.5 POSITIONING INSTRUCTIONS

MSC Controllers support two types of positioning instructions - absolute and incremental. Absolute motions are made relative to the zero or home position for the controller. Incremental motions are made relative to the current controller position. An absolute move is referred to as a **position**. An incremental move is called an **index**.

Positioning instructions are listed in Table 11.3.

Table 11.3 - POSITIONING INSTRUCTIONS

InstructionFormat				Ref. Page
Index	*label*	index	controller#,distance	200
Position	*label*	position	controller#,abs_position	226

11.6 PIECEWISE PROFILES

11.6.1 DESCRIPTION

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

MSC controllers perform piecewise motion 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.

11.6.2 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/second^2) 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.

Profiles may be conveniently entered using the data definition capability of Macroprogramming Language. The following example defines a simple piecewise profile.

```
prodatabegin_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/sec^2, continue until the motor has turned 6.2 revolutions (including acceleration distance).
2. Accelerate to 1000 RPM at an acceleration rate of 200 revs/sec^2. 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/sec^2. 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.

A sample program segment to execute this sample profile is shown in Figure 11.2.

Figure 11.2 - PIECEWISE PROFILES EXAMPLE PROGRAM SEQUENCE

```
pw_calc         equ             90                      calcs busy flag

prodata         begin_data
                data            500,100,6.2*4096
                data            1000,200,8.6*4096
                data            0,150,15.2*4096
                end_data
                .
                .
                prep_profile    1,prodata               send profile
ck_calc         if_stat_on      pw_busy,ck_calc         wait for calcs
                get_pstat       1,status
                if              status<>0,calc_err      error routine
                .
                exec_profile    1                       do the profile
wt_busy         if_stat_on      busy,wt_busy            wait til done
                .
                (instructions performed when execution is ok)
                .

!     Error handler

calc_err
                if              status=0,profile_ok             all zero = OK
                let_byte piece=status[3]                get piece
                let_byte error=status[2]                get error code
                .
                (process errors here)
                .
                .
```

Piecewise motion profiles may also be calculated by a macroprogram and stored in an array in the appropriate format. This format is always

speed
accel
distance
speed
accel
distance
speed
accel
distance
.
.

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.

11.6.3 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 Section 13.7, page **Error! Bookmark not defined.**, for a discussion of this technique.

Table 11.4 lists Piecewise profile instructions.

Table 11.4 - PIECEWISE PROFILE INSTRUCTIONS

Instruction	Format			Ref. Page
exec_profile	*label*	exec_profile	controller#	150
get_pstat	*label*	get_pstat	controller#,status	177
prep_profile	*label*	prep_profile	controller#,array_label	227
set_trig_pw	*label*	set_trig_pw	controller#,master_angle	280

11.7 READING CONTROLLER POSITION

The MSC-850/32 and MSC-850 provides a means for reading the current transducer position for the ACR-850 and ACE-850 Controllers, as well as for the MCF card when it is being used as a "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 that for the "**pseudo axis**", commanded position is always equal to actual position.

Table 11.5 summarizes instructions for reading controller positions.

Table 11.5 - INSTRUCTIONS FOR READING CONTROLLER POSITION

Instruction	Format			Ref. Page
get_com	*label*	get_com	controller#,variable	165
get_fol_err	*label*	get_fol_err	controller#,variable	166
get_pos	*label*	get_pos	controller#,variable	176

12.0 INTERRUPTS

12.1 DESCRIPTION

It is often necessary for an MSC controller, and therefore the Macroprogram, to respond rapidly to an external event, such as a switch closure or an operator input. Macroprogramming Language provides two methods for responding to this type of event.

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

12.2 SOFTWARE INTERRUPTS

The Software Interrupts feature of the MSC provides for automatic Macroprogram 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 Macroprogram 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 Macroprogram 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 macroprogram which should not be "interrupted".

Software Interrupt processing is edge triggered -- the specified gosub will occur only on the leading or trailing edge of the specified transition, depending on the type of interrupt specified. For example, if a software interrupt is specified to occur whenever input module number 1 turns 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.

A description of the macro instructions associated with software interrupts is at the end of this chapter.

12.3 HARDWARE INTERRUPTS

In some cases, it is desirable for a particular Controller Module 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 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 **HWI ARMED** flags. After the controller detects the interrupt and completes the task, its **BUSY** and **HWI 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 **HWI 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.

12.4 INTERRUPT INSTRUCTIONS

Table 12.1 lists software and hardware interrupt instructions.

Table 12.1 - SOFTWARE AND HARDWARE INTERRUPT INSTRUCTIONS

Instruction	**Format**			**Ref. Page**
clr_all_swi	*label*	clr_all_swi		127
clr_swi	*label*	clr_swi	interrupt#	132
disable_hwi	*label*	disable_hwi	controller#	139
disable_swi	*label*	disable_swi		140
enable_hwi	*label*	enable_hwi		143, 144
enable_swi	*label*	enable_swi		145
f_decel	*label*	f_decel	controller#	152
get_trap_pos	*label*	get_trap_pos	controller#,variable	182
over_draw	*label*	over_draw	controller#,speed,limit,distance	221
set_swi_mask	*label*	swt_swi_mask	variable	277
swi_if_off	*label*	swi_if_off	interrupt#,flag,sub_label	283
swi_if_on	*label*	swi_if_on	interrupt#,flag,sub_label	284
trap_pos	*label*	trap_pos	controller#	293

13.0 MASTER SLAVE CONCEPTS

13.1 DESCRIPTION

MSC-850/32, MSC-850 and MSC-250 Controllers provide a mode of operation whereby the motions of one axis, called a master axis, can be used to control the motions 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 System Units using the Fiber Optic Network feature (See Section 13.9).

Slave Controllers use the Master Controller 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 Controller transmits its current position transducer reading onto the designated data bus every millisecond. In the MSC-250, this occurs every 488 microseconds. Each slave Controller 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.

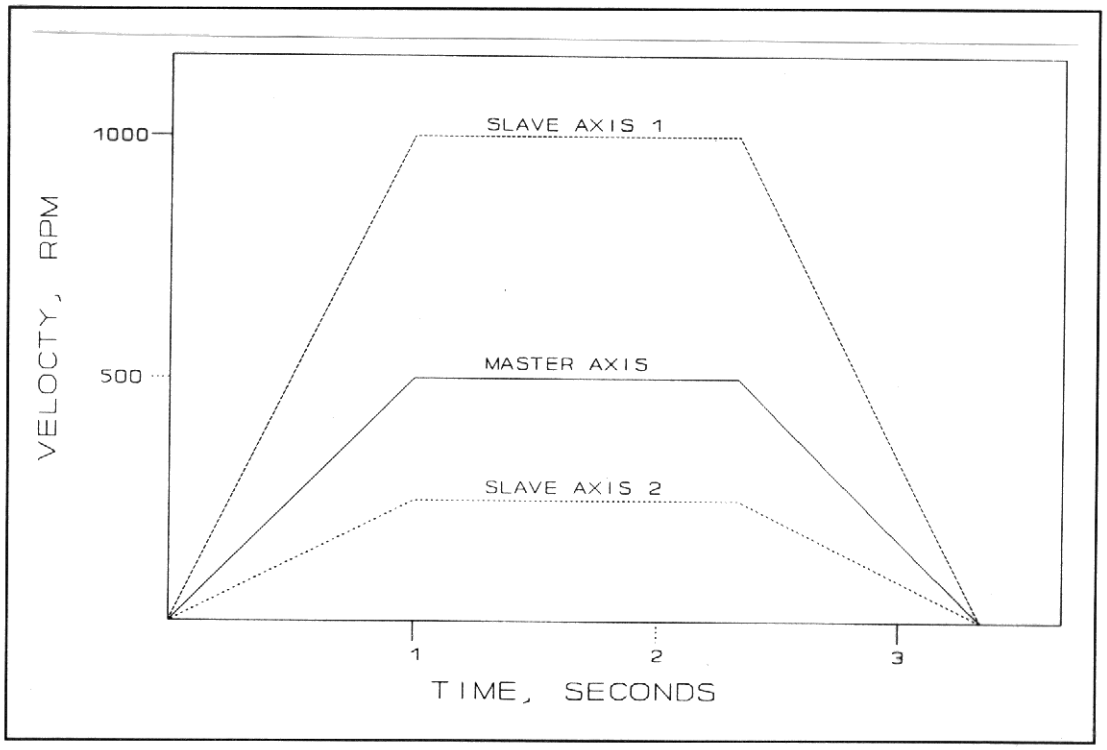


Figure 13.1 - SIMPLE LOCK ILLUSTRATION

13.2 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 13.1 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 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.

13.2.1 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. At the start of a Macroprogram, the ratio value for each axis is set 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 (see Section 13.4).

   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 **AXIS 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.

13.3 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 Table 13.1.

Table 13.1 - LOCK METHODS FOR SIMPLE LOCK

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.

Details for each of these lock methods are presented below.

13.3.1 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 **MOTOR 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 13.2.

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

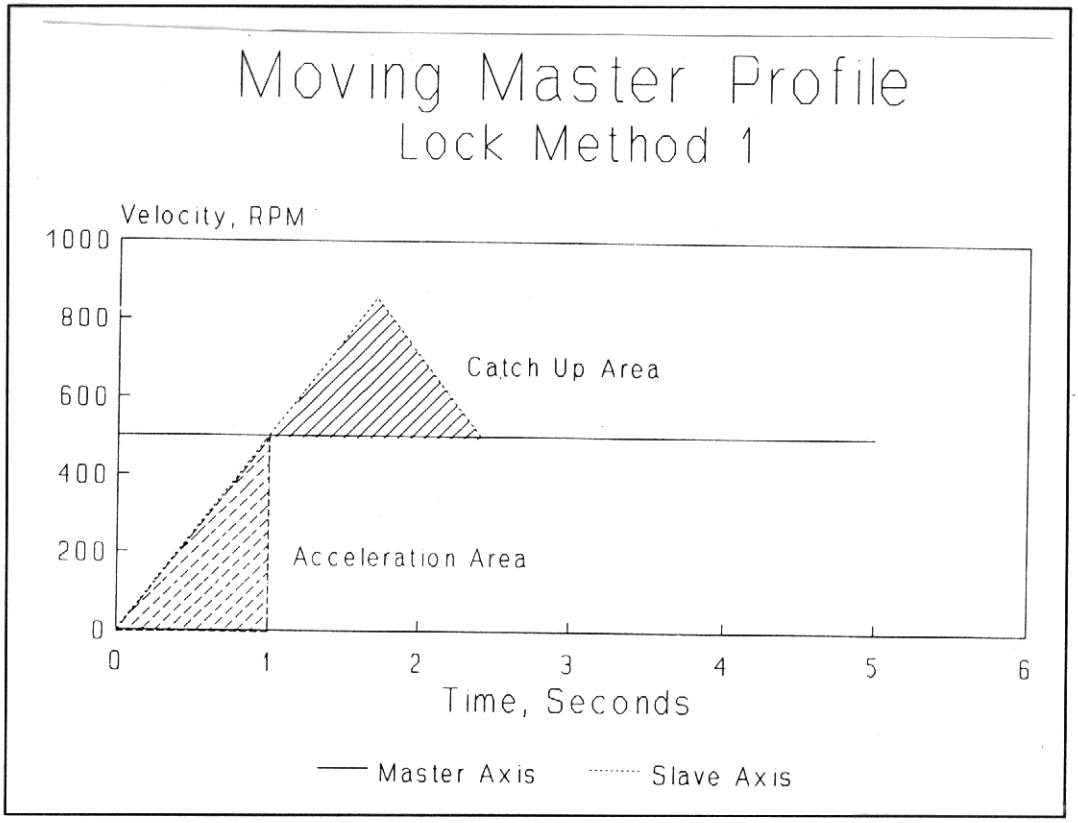


Figure 13.2

13.3.2 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 that 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 **MOTOR 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.

13.3.3 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.

Master Angle = 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).

13.4 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 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.

13.4.1 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.

Table 13.2 summarizes the effect of master scaling.

Table 13.2 - 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

13.4.2 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.

13.4.3 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.

13.4.4 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 Macroprogram, 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 reponsible 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 transitted 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 AXIS 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:

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.

EXAMPLE

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:

```
                msc_type        850
                declare off

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 (SCALED by 2^15) -----
!
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
.
.
.
.
```

13.5 ELECTRONIC CAM LOCK METHODS

Table 13.3 - 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.

A detailed explanation of each of these lock methods follows.

13.5.1 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.

13.5.2 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.

13.5.3 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**).

13.5.4 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.

13.5.5 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 be 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.

13.6 SAMPLE ELECTRONIC CAM APPLICATION

An application requires a slave axis to follow a motion described by the mechanical cam shown in Figure 13.3. 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.

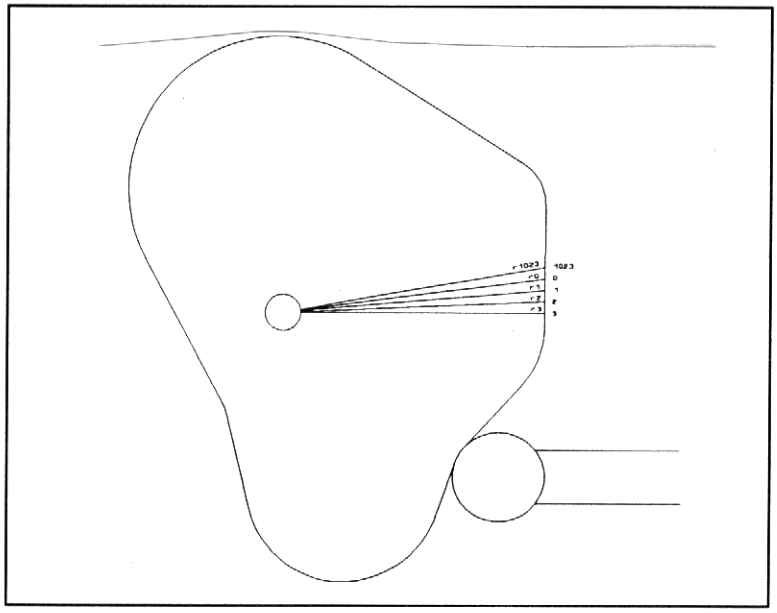


Figure 13.3 - MECHANICAL CAM

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 an MSC Macroprogram to produce the desired motion. Figure 13.4 shows a graphic representation of the resulting cam motion.

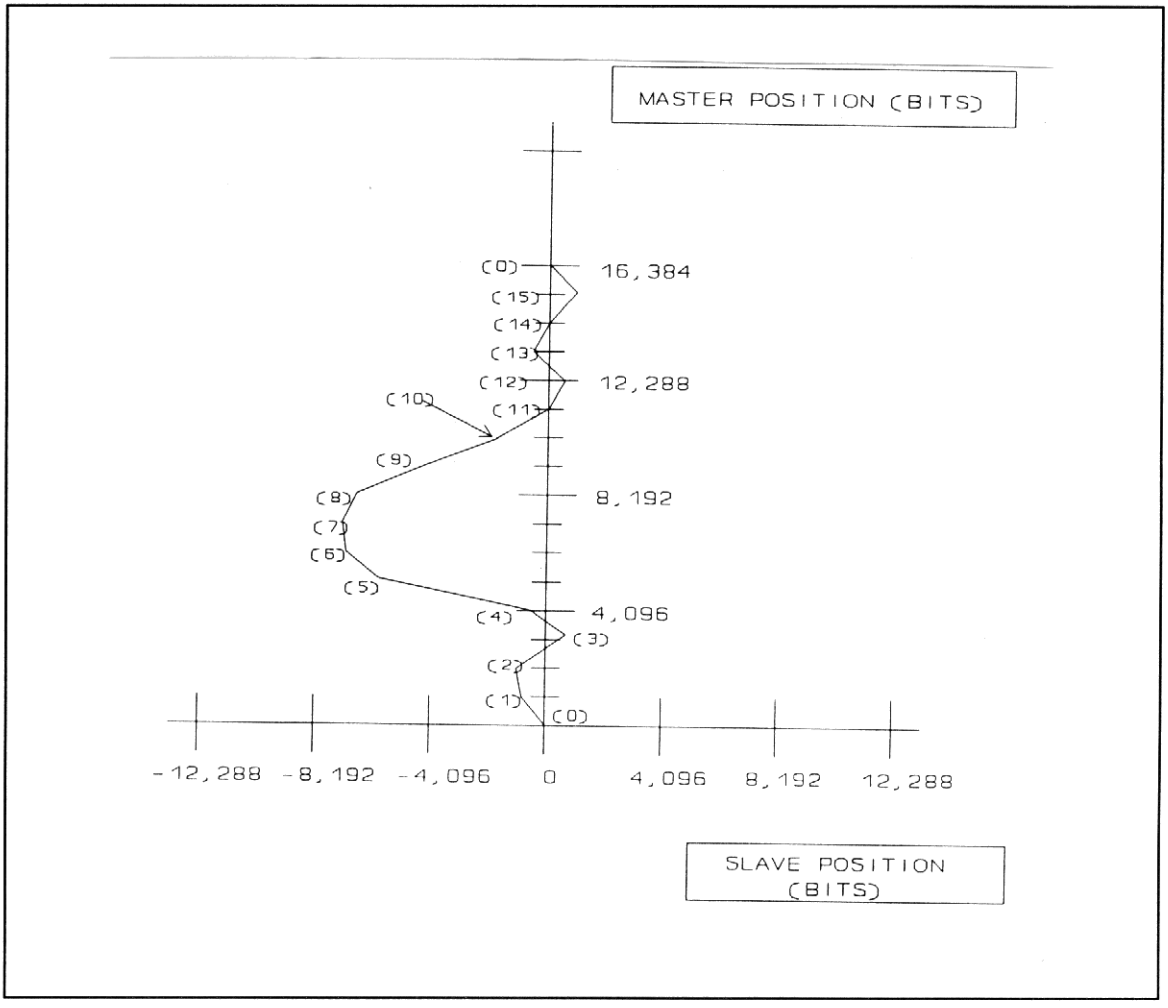


Figure 13.4 - RESULTING MASTER/SLAVE MOTION

13.7 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.

13.8 MASTER ANGLE BUS

The master angle bus is a high speed data highway which links the controllers in an MSC-850/32 or MSC-850 System Unit. Each System Unit has two independent buses, designated as Master Bus A and Master Bus B. The **set_map** Macroprogram 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 13.5 diagrams the configuration of the **set_map** variable for the MSC-850/32 and MSC-850. Figure 13.6 shows the configuration of the **set_map** variable for the MSC-250.

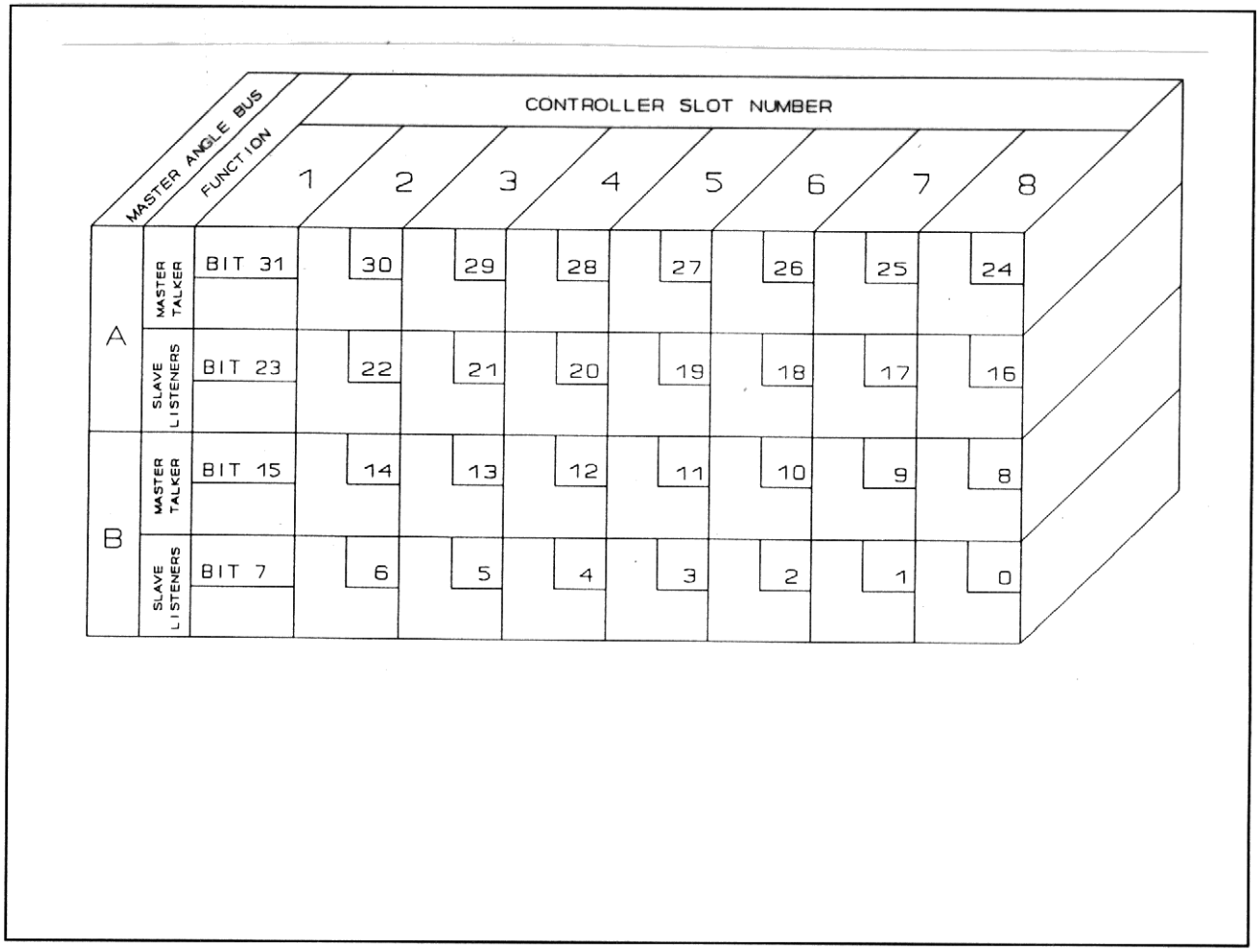


Figure 13.5 - MSC-850, MSC-850/32 MASTER ANGLE BUS CONFIGURATION CHART

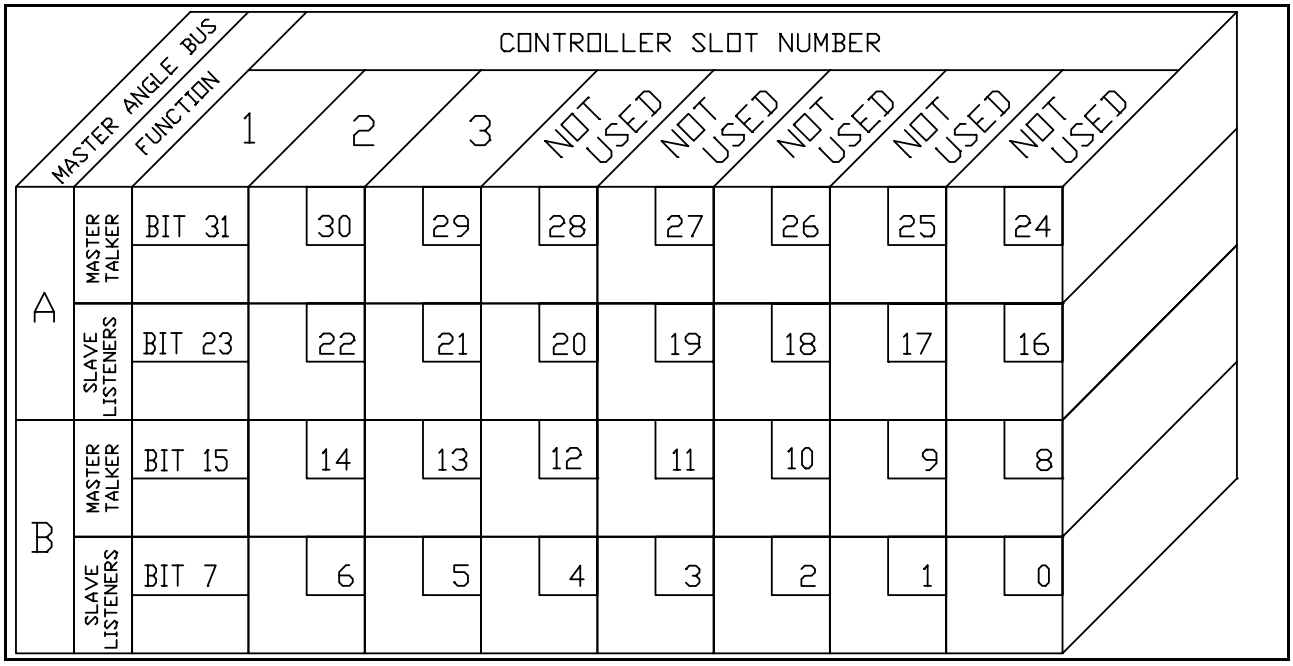


Figure 13.6 - MSC-250 MASTER ANGLE BUS CONFIGUARTION CHART

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 bus A will be used for the data path. The following Macroprogram instructions could be used to configure the master angle bus:

```
talk_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
            .
            .
            let         mapvar=0        clear all bits
            set_map     mapvar          stop all map activity
            set_bit     talk_3,mapvar   set appropriate bits
            set_bit     slave_1,mapvar
            set_bit     slave_4,mapvar
            set_bit     slave_5,mapvar
            set_map     mapvar          get communications going
(instructions to set up desired master slave modes)
```

13.8.1 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**. Macroprogram 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.

13.9 FIBER OPTIC NETWORK

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

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

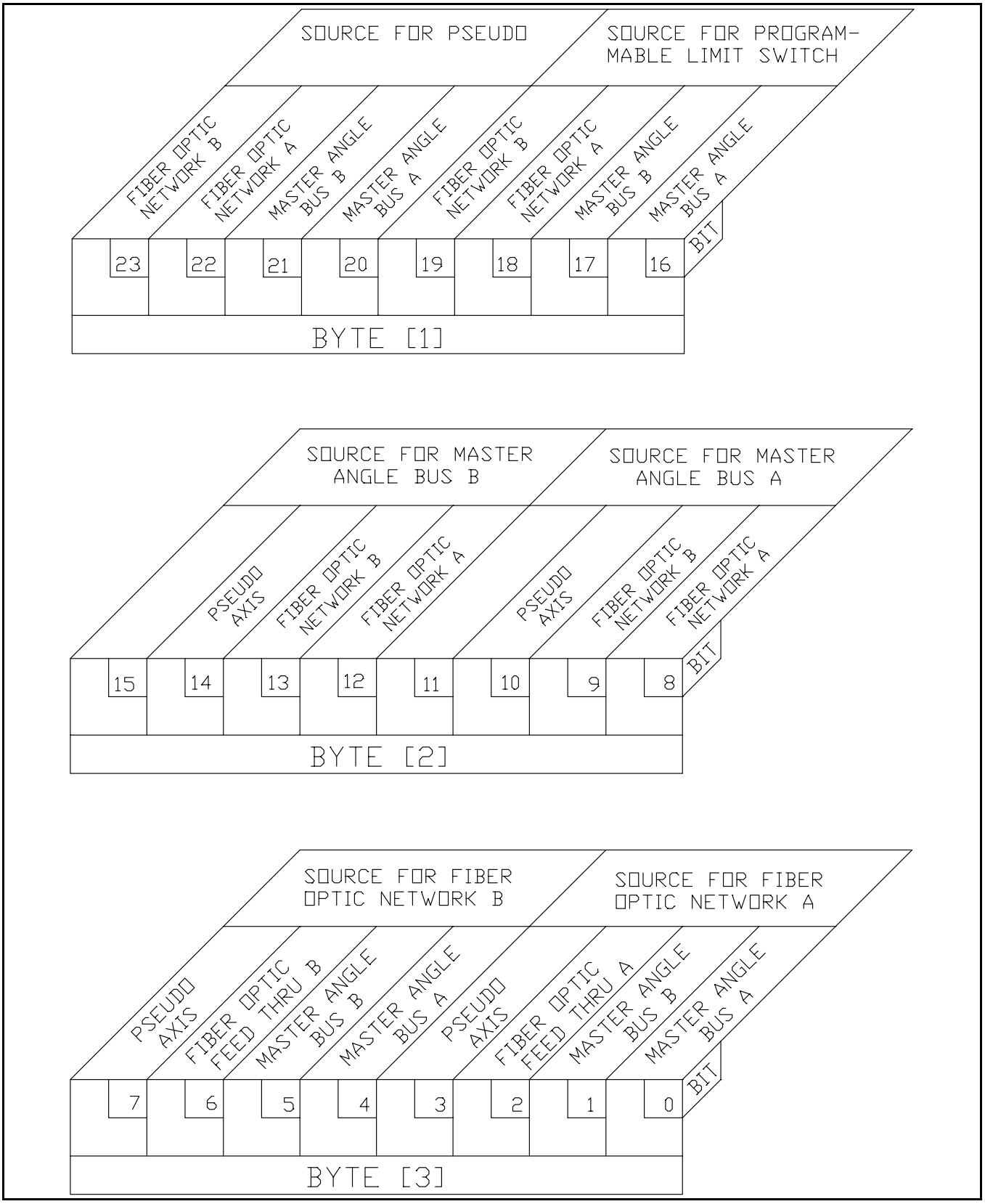


Figure 13.7 - MSC-850/MCF-850 CONFIGURATION CHART

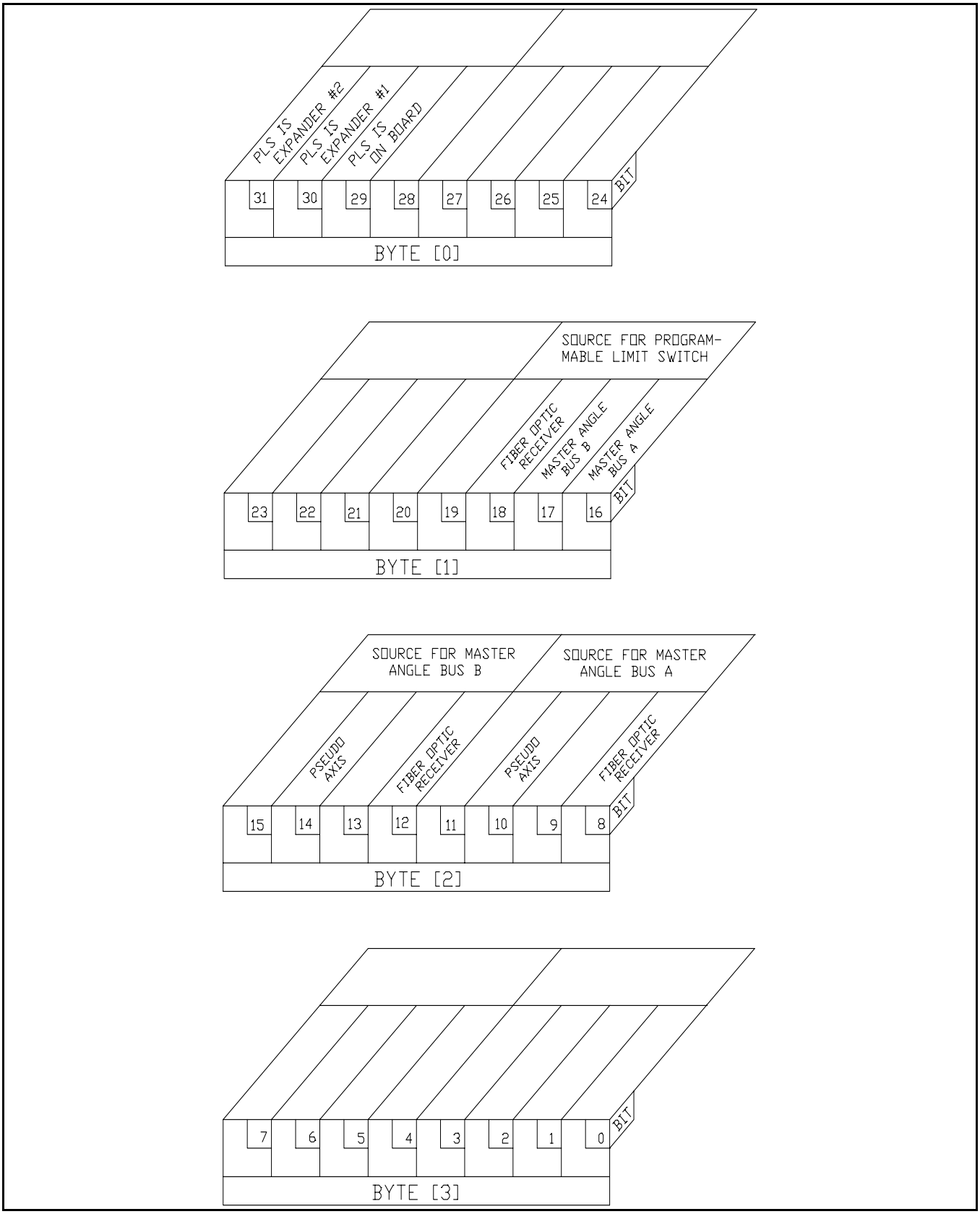


Figure 13.8 - MSC-250 MULTI-FUNCTION CONFIGURATION CHART

Each MCF-850 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 13.7 shows the bit assignments used to define the network data paths.

Each MSC-250 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 13.8 shows the bit assignments used to define the network data paths.

Consider the example diagrammed in Figure 13.9. In this example, three MSC-850 System Units are linked in a daisy chain fashion. The location of master axes and communication interrelationships are shown in the figure. The following Macroprogram segments could be used to establish the desired network configuration.

Program for System Unit 1

```
talk_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
MABAA       equ     0               Bus A is source for Fiber Optic A
mcf_slot    equ     6               MCF is in Slot 6
              .
              .
            let     mcf_var=0
            set_bit MABAA,mcf_var   Master bus A is Fiber Optic A source
            set_mcf mcf_slot,mcf_var

            let     mapvar=0        clear all bits
            set_map mapvar          stop all map activity
            set_bit talk_3,mapvar   set appropriate bits
            set_bit slave_1,mapvar
            set_bit slave_4,mapvar
            set_bit slave_5,mapvar
            set_map mapvar          get communications going
              .
(instructions to set up desired master slave modes)
               .
```

Program for System Unit 2

```
talk_1      equ     29              Slot 1 talker on Bus B
slave_2     equ     23              Slot 1 listen on Bus B
slave_3     equ     20              Slot 4 listen on Bus B
MABAA       equ     0               Bus A is source for Fiber Optic A
MABBB       equ     13              Bus B is source for Fiber Optic B
mcf_slot    equ     4               MCF is in Slot 4
              .
              .
            let     mcf_var=0
            set_bit MABAA,mcf_var   Master bus A is Fiber Optic A source
            set_bit MABBB,mcf_var   Master bus B is Fiber Optic B source
            set_mcf mcf_slot,mcf_var

            let     mapvar=0        clear all bits
            set_map mapvar          stop all map activity
            set_bit talk_1,mapvar   set appropriate bits
            set_bit slave_2,mapvar
            set_bit slave_3,mapvar
            set_map mapvar          get communications going
              .
(instructions to set up desired master slave modes)
        .
```

Program for System Unit 3

```
slave_1A    equ     23              Slot 1 listen on Bus A
slave_2B    equ     20              Slot 2 listen on Bus B
slave_3B    equ     19              Slot 3 listen on Bus B
MABAA       equ     0               Bus A is source for Fiber Optic A
MABBB       equ     13              Bus B is source for Fiber Optic B
mcf_slot    equ     4               MCF is in Slot 4
              .
              .
            let     mcf_var=0
            set_bit MABAA,mcf_var   Master bus A is Fiber Optic A source
            set_bit MABBB,mcf_var   Master bus B is Fiber Optic B source
            set_mcf mcf_slot,mcf_var

            let     mapvar=0        clear all bits
            set_map mapvar          stop all map activity
            set_bit slave_1A,mapvar
            set_bit slave_2B,mapvar
            set_bit slave_3B,mapvar
            set_map mapvar          get communications going
              .
(instructions to set up desired master slave modes)
              .
```

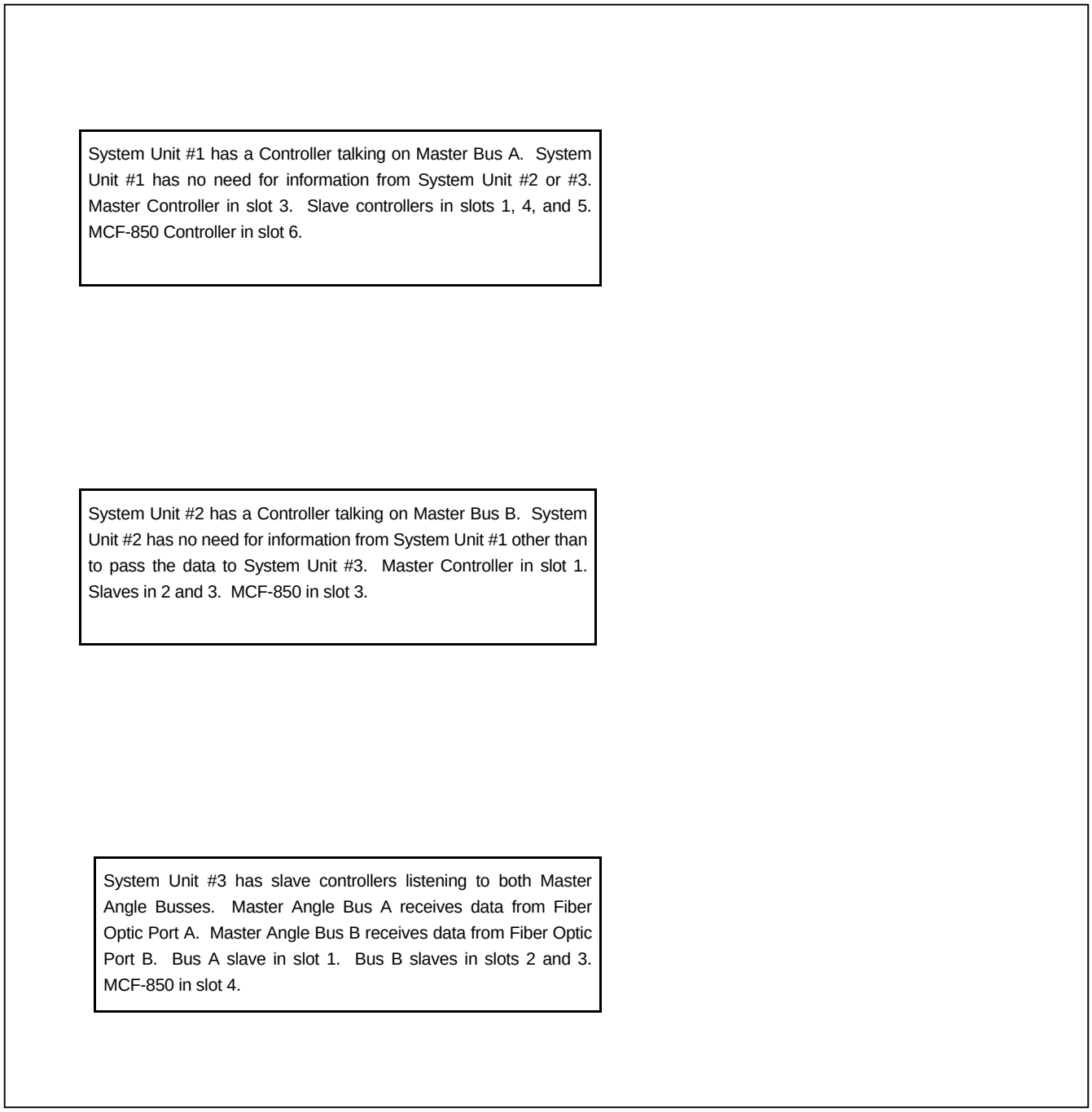


Figure 13.9 - DAISY CHAIN FIBER OPTIC NETWORK

13.10 MASTER SLAVE INSTRUCTIONS

Table 13.4 summarizes master slave instructions.

Table 13.4 - MASTER SLAVE INSTRUCTION SUMMARY

Instruction	Format			Ref. Page
calc_cam_sum	*label*	calc_cam_sum	controller#,start_element,ending_element	120
calc_unit_cam	*label*	calc_unit_cam	controller#,distance,# elements,start_element	121
cam_data	*label*	cam_data	controller#,label,ms,ds	123, 124
get_cam_cnt	*label*	get_cam_cnt	controller#,variable	160
get_cam_end	*label*	get_cam_end	controller#,variable	161
get_cam_ptr	*label*	get_cam_ptr	controller#,variable	162
get_cam_strt	*label*	get_cam_strt	controller#,variable	163
get_map	*label*	get_map	variable	168, 169
get_map_stat	*label*	get_map_stat	variable	170
get_mcf	*label*	get_mcf	controller#,variable	171, 172
incr_com	*label*	incr_com	controller#,bits,interrupts	199
lock	*label*	lock	controller#,lock#	211
ratio	*label*	ratio	controller#,ratio	233
set_cam_ptr	*label*	set_cam_ptr	controller#,value	243
set_map	*label*	set_map	variable	254
set_mcf	*label*	set_mcf	controller#,variable	258
set_trig_cam	*label*	set_trig_cam	controller#,master_angle	279
switch_cam	*label*	switch_cam	controller#,start_element	285
unlock	*label*	unlock	controller#,mode#	296

14.0 PROGRAMMABLE LIMIT SWITCHES

14.1 DESCRIPTION

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.

14.2 MSC-850/MCF-850 and MSC-250 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. In the MSC-250, they may be 16 modules located at the on-board I/O, I/O expander #1 or I/O expander #2. 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 user's Macroprogram 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.

14.2.1 PROGRAMMING

Before attempting to write a Macroprogram 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.

14.2.2 PROCESSING

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 CALCULATING flag of the axis controller will be activated. The Macroprogram 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 CALCULATING flag is activated.

14.2.3 EXECUTION

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.

NOTES

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 macroprogram.

14.3 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 Macroprogram. 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 Macroprogram.

The application Macroprogram 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.

14.3.1 THEORY OF OPERATION

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 Macroprogram.

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).

14.3.2 PROGRAMMING CONSIDERATIONS

Before attempting to write a Macroprogram 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.

14.4 HPL-850 PROGRAMMING EXAMPLE #1

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^{o} and 200^{o}. This output fires an air cylinder which has a 50 millisecond time delay, requiring the output signal to be advanced by 50 milliseconds.

The following Macroprogram excerpt could be used for this application.

```
hpl         equ         3               HPL-850 card is in slot 3
hpl_calc    equ         122             calculating data flag, slot 3
pls_cnt     equ         4096            4096 counts per machine cycle

! constants to define on angle and off angle

on_angle    equ         (4096*160)/360
off_angle   equ         (4096*200)/360

air_cyl     equ         0               module 0 is air cylinder output
time_adv    equ         5               advance 5 timer ticks = 50 ms.

            .
            .
            set_hpl  hpl,1              data source is Master Bus A
            set_pls_cnt hpl,pls_cnt     set counts per cycle

! set up the on, off angles and the output module number

            set_pls_ang hpl,on_angle,off_angle,air_cyl
w1          if_stat_on  hpl,hpl_calc,w1

! set up the time advance

            set_pls_time hpl,time_adv,air_cyl
w2          if_stat_on  hpl,hpl_calc,w2

! set the HPL-850 accumulator to zero (assume master is already at zero)

            set_local   hpl
            .
            (continue program execution)
```

14.5 HPL-850 PROGRAMMING EXAMPLE #2

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 Macroprogram excerpt could be used for this application.

```
hpl             equ             3               HPL-850 card is in slot 3
hpl_calc        equ             122             calculating data flag, slot 3
pls_count       equ             26*4096         counts per machine cycle

! constants to define on angle and off angle

on_angle        equ             14*4096
off_angle       equ             21*4096

oil_mist        equ             0               module 0 is oil mist output

                .
                .
                set_hpl         hpl,2           data source is Master Bus B
                set_pls_cnt     hpl,pls_count   set counts per cycle

! set up the on, off angles and the output module number

                set_pls_ang     hpl,on_angle,off_angle,oil_mist
w1              if_stat_on      hpl,hpl_calc,w1
                 .
                 .
                (instructions to initialize lead screw and position to zero)
                 .
                 .

! set the HPL-850 accumulator to zero (master is already at zero)

                set_gl_cw       hpl_slot
                 .
                 .
                (continue program execution)
                 .
                 .
```

14.6 PROGRAMMABLE LIMIT SWITCH INSTRUCTIONS

Table 14.1 lists the Macroprogram instructions associated with Programmable Limit Switch Functions.

Table 14.1 - PROGRAMMABLE LIMIT SWITCH INSTRUCTIONS

Instruction	Format			Ref. Page
get_angle	*label*	get_angle	controller#,variable	158, 159
get_for_ang	*label*	get_for_ang	controller,channel#,variable	167
get_pls_mask	*label*	get_pls_mask	controller#,variable	173
get_pls_out	*labe*	/get_pls_out	controller#,variable	174
preset	*labe*	/preset	controller#,variable	228, 229
set_gl_ccw	*labe*	/set_gl_ccw	controller#	245, 246
set_gl_cw	*label*	set_gl_cw	controller	247, 248
set_local	*label*	set_local	controller#	252
set_mcf	*labe*	/set_mcf	controller,variable	256
set_pls_ang	*label*	set_pls_ang	controller#,on,off,module#	267
set_pls_cnt	*label*	set_pls_cnt	controller#,count	272
set_pls_mask	*label*	set_pls_mask	controller#,variable	274
set_pls_time	*label*	set_pls_time	controller#,time,module#	275

15.0 EXTENDED MEMORY OPERATIONS

15.1 DESCRIPTION

The MSC-850/32 and MSC-850 provides 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 in to each MSC System Unit.

15.2 EXTENDED RAM MEMORY

Using the MSC-850/32 or the MSC-850, the ACE-850, ACR-850 and ACY-850 Controllers have 28K bytes of volatile data memory available for use by a Macroprogram. In the MSC-250, axis 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 and for other operations which need large amounts of temporary storage.

The MCF-850 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.

15.2.1 EXTENDED RAM MEMORY 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 exactly the same way as for any other 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.

15.2.2 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.

15.3 EPROM MEMORY

EPROM Memory on the MSC is controlled through a series of Macroprogram 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, 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 Macroprogram 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.

15.3.1 AUTOMATIC PROGRAM LOAD FROM EPROM

The MSC power up sequence allows the automatic loading of a macroprogram from the PROM POCKET EPROM. If an EPROM is in the PROM POCKET and the EPROM contains a Macroprogram file as the first file, that Macroprogram will be loaded, execution will begin and the MSC AUTOSTART bit will be set. This sequence will take place even if there is a valid Macroprogram already in the MSC and even if that Macroprogram has been set to AUTOSTART.

15.3.2 EPROM STATUS CODES

Each EPROM Manager instruction returns a status code that indicates the result of the instruction. The status return codes are shown in Table 15.1.

Table 15.1 - EPROM STATUS CODES

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 macroprogram 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

15.4 EPROM MANAGER INSTRUCTIONS

Instructions for use with the EPROM Manager are summarized in Table 15.2.

Table 15.2 - EPROM MANAGER INSTRUCTIONS

Instruction	Format			Ref. Page
Close	*label*	close	unit,status	126
create	*label*	create	unit,f_n,status	133
get_space	*label*	get_space	unit,space,status	178
get_volume	*label*	get_volume	unit,d_a,status	183
load	*label*	load	unit,f_n,status	210
initialize	*label*	initialize	unit,d_a,status	201
open	*label*	open	unit,f_n,status	220
read	*label*	read	unit,d_a,l,status	234
save	*label*	save	unit,f_n,status	238
write	*label*	write	unit,d_a,l,status	299

16.0 ANALOG INPUT/OUTPUT

16.1 DESCRIPTION

MSC-850/32 and MSC-850

The ACM-850 Analog Control Module provides the MSC-850 system 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.

16.2 CAPABILITIES

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.

16.3 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.

16.4 POWER ON STATES

On power up, or after a **RESET** command from **MPDEBUG**, 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

16.5 ACM-850 INSTRUCTIONS

Table 16.1 - ACM-850 INSTRUCTIONS

Instruction	Format			Ref. Page
analog_in	*label*	analog_in	controller#,ch#,value	109
analog_out	*label*	analog_out	controller#,ch#,output	110
analog_rt	*label*	analog_rt	controller#,ch#,value	114
analog_zo	*label*	analog_zo	controller#,ch#,value	115

17.0 USER SERIAL PORTS

17.1 DESCRIPTION

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:

MSC TYPE	PORT NUMBER	DESCRIPTION
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 IIS Packet Protocol method of communication. This port is typically used by the Macroprogram Development System for the loading and testing of IIS Macroprograms.

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

17.2 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_setport,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 Table 17.1.

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.

Table 17.1 - COMMUNICATION PROTOCOL SELECTION

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

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.

17.3 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 Macroprogram 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 instruction sequence

   ```
   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 instruction 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.

17.4 SERIAL INSTRUCTIONS

Table 17.2 summarizes instructions used with MSC serial ports.

Table 17.2 - SERIAL PORT INSTRUCTION SUMMARY

Instruction	Format			Ref. Page
get_pq_space	*label*	get_pq_space	space	175
if_char	*label*	if_char	port#,address_label	189
if_no_char	*label*	if_no_char	port#,address_label	194
input	*label*	input	t_l,len,dec,var,u_f	202
port_set	*label*	port_set	port#,baud,protocol	224
print	*label*	print	text_label	230
print_num	*label*	print_num	len,dec,var	231
stop_input	*label*	stop_input		282

18.0 INSTRUCTION REFERENCE

INSTRUCTION	DESCRIPTION
analog_in	controller#,channel#,variable
analog_out	controller#,channel#,value
analog_rt	controller#,channel#,value
analog_zo	controller#,channel#,value
begin_cam	
begin_data	
blk_io_in	input_flag#,variable
blk_io_out	output_flag#,variable
calc_cam_sum	controller#,starting element,ending element
calc_unit_cam	controller#,distance,# of elements,starting element
cam	value,value,etc.
cam_data	controller#,data_label,master_scale,data_scale
case	num
close	unit,status
clr_all_swi	
clr_bit	bit#,variable
clr_flag	user_flag#
clr_hi_scan	
clr_local	controller#
clr_swi	interrupt#
create	unit,file_name,status
data	value,value,etc.
declare	mode
default	
digi_comp	controller#,gain,integral,damp
dim	size
dim	controller#,size
disable_hwi	controller#
disable_swi	
drive_off	controller#
drive_on	controller#
enable_hwi	
enable_swi	
end_cam	
end_data	
end_select	
equ	constant_expression
exec_profile	controller#
exit_select	
f_decel	controller#
find_mrk_ccw	controller#,counts
find_mrk_cw	controller#,counts
find_tm_ccw	controller#,counts
find_tm_cw	controller#,counts
get_act_spd	controller#,variable
get_angle	controller#,variable
get_cam_cnt	controller#,variable
get_cam_end	controller#,variable
get_cam_ptr	controller#,variable

get_cam_strt	controller#,variable
get_cam_sum	controller#,variable
get_com	controller#,variable
get_fol_err	controller#,variable
get_for_ang	controller#,channel#,variable
get_map	variable
get_map_stat	variable
get_mcf	controller#,variable
get_pls_mask	controller#,variable
get_pls_out	controller#,variable
get_pq_space	variable
get_pos	controller#,variable
get_pstat	controller#,status
get_space	unit,space,status
get_status	controller#
get_t_mark	controller#,state
get_time	variable
get_trap_pos	controller#,variable
get_volume	unit,data_area,status
gosub	subroutine_label
goto	address_label
if	compare1 operator compare2,address_label
if_bit_clr	bit#,variable,address_label
if_bit_set	bit#,variable,address_label
if_char	port#,address_label
if_flag_off	user_flag#,address_label
if_flag_on	user_flag#,address_label
if_io_off	I/O flag#,address_label
if_io_on	I/O flag#,address_label
if_no_char	port#,address_label
if_stat_off	status_flag#,address_label
if_stat_on	status_flag#,address_label
if_tmr_off	timer_flag#,address_label
if_tmr_on	timer_flag#,address_label
incr_com	controller#,bits,interrupts
index	controller#,distance
initialize	unit,data_area,status
input	label,length,decimals,variable,user_flag
integer	
jog_ccw	controller#
jog_cw	controller#
l_track_spd	controller#,speed
let	variable=operand1 opcode operand2
let_byte	destination=source
load	unit,file_name,status
lock	controller#,lock#
master	controller#
msc_type	system_type
no_op	
offset_master	controller id#,offset
open	unit,file_name,status
over_draw	controller#,speed,limit,distance
port_set	port#,baud,protocol

position	controller#,abs_position
prep_profile	controller#,data_label
preset	controller#,variable
print	text_label
print_num	length,decimals,value
rand_int	max_number,variable
ratio	controller#,ratio
read	unit,data_area,length,status
read_offset	controller#,variable
restart_at	address_label
return_sub	
save	unit,file_name,status
select	variable
set_ac_dc	controller#,rate
set_acy_cnt	controller#,count
set_bit	bit#,variable
set_cam_ptr	controller#,value
set_flag	user_flag#
set_gl_ccw	controller#
set_gl_cw	controller#
set_hi_scan	
set_home	controller#,offset
set_local	controller#
set_map	variable
set_mcf	controller#,variable
set_offset	controller#,value
set_ovd_mode	controller#,mode
set_pls_ang	controller#,on_angle,off_angle,module#
set_pls_cnt	controller#,count
set_pls_mask	controller#,variable
set_pls_time	controller#,time,module#
set_speed	controller#,speed
set_swi_mask	variable
set_tmr	timer_flag#,time
set_trig_cam	controller#,master_angle
set_trig_pw	controller#,master_angle
set_vgain	controller#,vel_gain
stop_input	
swi_if_off	interrupt#,flag,subroutine_label
swi_if_on	interrupt#,flag,subroutine_label
switch_cam	controller#,start element, # of elements
sys_fault	
sys_return	
test_mode	controller#
text	"ASCII string"
track_spd	controller#,speed
trap_pos	controller#
turn_off	I/O flag#
turn_on	I/O flag#
unlock	controller#,mode#
vel_ccw	controller#
vel_cw	controller#
write	unit,data_area,length,status

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.

SEE:

analog_out
analog_rt
analog_zo

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-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	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.

SEE:

analog_in
analog_zo
analog_rt

USAGE:

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

analog_out (ACR-850 or ACE-850 or 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.

SEE:

set_mcf (ACR-850 or ACE-850 or ACY-850 cards)

USAGE:

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

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.

SEE:

analog_in
analog_zo
analog_rt

USAGE:

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

analog_out (MSC-250 controller 1 and 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.

SEE:

set_mcf (MSC-250 controller 1 and 2)

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:

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	1 (for input channel) & 9 (for output channel)
		MSC-850/32	1 to 12
		MSC-850	1 to 12
		MSC-800	1 to 12
value	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.

SEE:

analog_in
analog_out
analog_zo

USAGE:

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

analog_zo

SYNTAX:

label analog_zo controller#,channel#,value

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	1 (for input channel) & 9 (for output channel)
		MSC-850/32	1 to 12
		MSC-850	1 to 12
		MSC-800	1 to 12
value	Offset value.		
	Range:	-2048 to +2047	

DESCRIPTION:

Sets the specified channel offset value in 4.88 millivolts per bit. Offset value is cleared on power-up. Outputs will slew to 0.0 volts when program is stopped by the MSC Toolkit. Zero offset values are cleared on power-up.

RETURNS:

None.

SEE:

analog_in
analog_out
analog_rt

USAGE:

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

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.

SEE:

cam
end_cam

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.

SEE:

end_data
data

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)

SEE:

blk_io_out

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.

SEE:

blk_io_in

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 'calculating' will be ON until calculations are complete.
The calculated sum can be retrieved using the **get_cam_sum** instruction.

RETURNS:

None.

SEE:

get_cam_sum
calc_unit_cam

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.

SEE:

calc_cam_sum
get_cam_sum

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.

SEE:

begin_cam
end_cam

USAGE:

MSC-250: am
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 macroprogram 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.

cam_data (continued)

SEE:

begin_cam
cam
end_cam
lock
set_map
let
let_byte

USAGE:

MSC-250:c	am_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.

SEE:

select
default
exit_select
end_select

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.

SEE:

write
open
read
create
initialize
get_space
get_volume
save
load

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.

SEE:

swi_if_on
swi_if_off
clr_swi
enable_swi
disable_swi
set_swi_mask

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.

SEE:

set_bit

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.

SEE:

set_flag

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.

SEE:

set_hi_scan

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.

SEE:

set_local

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.

SEE:

clr_all_swi
enable_swi
disable_swi
swi_if_on
swi_if_off
set_swi_mask

USAGE:

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

create

SYNTAX:

label create 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 creates a new file on the EPROM and prepares to write data in the file. The MSC will associate the file name with the unit identifier and will set the file record pointer to the beginning of the file.

Only 1 file may be written to 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.

SEE:

open
close
write
initialize
get_space
get_volume
load
save

USAGE:

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

data

SYNTAX:

label data value,value,etc.

PARAMETERS:

value 32-bit constant
Range :-2,147,483,648 to +2,147,483,647

DESCRIPTION:

Specifies 32-bit values. More than 1 value may be contained in a statement.

EXAMPLE

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

RETURNS:

None.

SEE:

begin_data
end_data

USAGE:

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

declare

SYNTAX:

label declare mode

PARAMETERS:

mode This entry can be only ON or OFF.

DESCRIPTION:

This is a directive to the MSC Toolkit 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 macroprogram.

RETURNS:

None.

SEE:

No related instructions.

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.

SEE:

select
end_select
exit_select
case

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.

SEE:

set_vgain

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.

SEE:

No related instructions.

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 **HWI ARMED** status in the controller.

RETURNS:

None.

SEE:

enable_hwi

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.

SEE:

enable_swi
swi_if_on
swi_if_off
clr_swi
clr_all_swi
set_swi_mask

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:

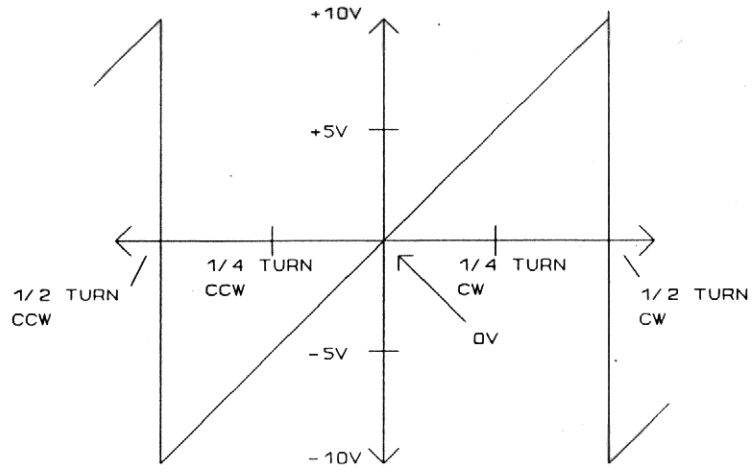
controller# controller ID #

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 follows:



NOTE:

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

RETURNS:

None.

SEE:

drive_on

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.

SEE:

drive_off, set_mcf
set_mcf (ACR-850 or ACE-850 or ACY-850 cards)
set_mcf (MSC-250 controller 1 and 2)

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 MSC Toolkit 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 **HWI ARMED** status in the controller will be on after this instruction has been executed.

RETURNS:

None.

enable_hwi (continued)

SEE:

disable_hwi
over_draw
trap_pos
ratio
position
index
lock
exec_profile
f_decel

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.

SEE:

disable_swi
swi_if_on
swi_if_off
clr_swi
clr_all_swi
set_swi_mask

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.

SEE:

begin_cam
cam

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.

SEE:

begin_data
data

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.

SEE:

select
case
exit_select
default

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 MSC Toolkit 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.

SEE:

text
integer

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 **AXIS BUSY** and **MOTOR 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 Macroprogram instruction directs communication to occur.

RETURNS:

None.

SEE:

prep_profile
get_pstat

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.

SEE:

select
case
end_select
default

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 **FORCE 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.

SEE:

unlock

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.

SEE:

find_mrk_cw

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.

SEE:

find_mrk_ccw

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-2501 to 2
MSC-850/321 to 8
MSC-8501 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.

SEE:

find_tm_cw

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 -(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.

SEE:

find_tm_ccw

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.

SEE:

No related instructions.

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

SEE:

get_for_ang

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'.

SEE:

preset

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.

SEE:

No related instructions.

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.

SEE:

switch_cam
get_cam_strt
get_cam_count

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.

SEE:

set_cam_ptr

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.

SEE:

get_cam_end
get_cam_count

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 'CALCULATING' has gone from ON to OFF.

RETURNS:

The calculated cam sum.

SEE:

calc_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.

SEE:

get_pos

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.

SEE:

No related instructions.

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.

SEE:

No related instructions.

USAGE:

MSC-250:	get_for_ang
MSC-850/32:	get_for_ang
MSC-850:g	et_for_ang
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.

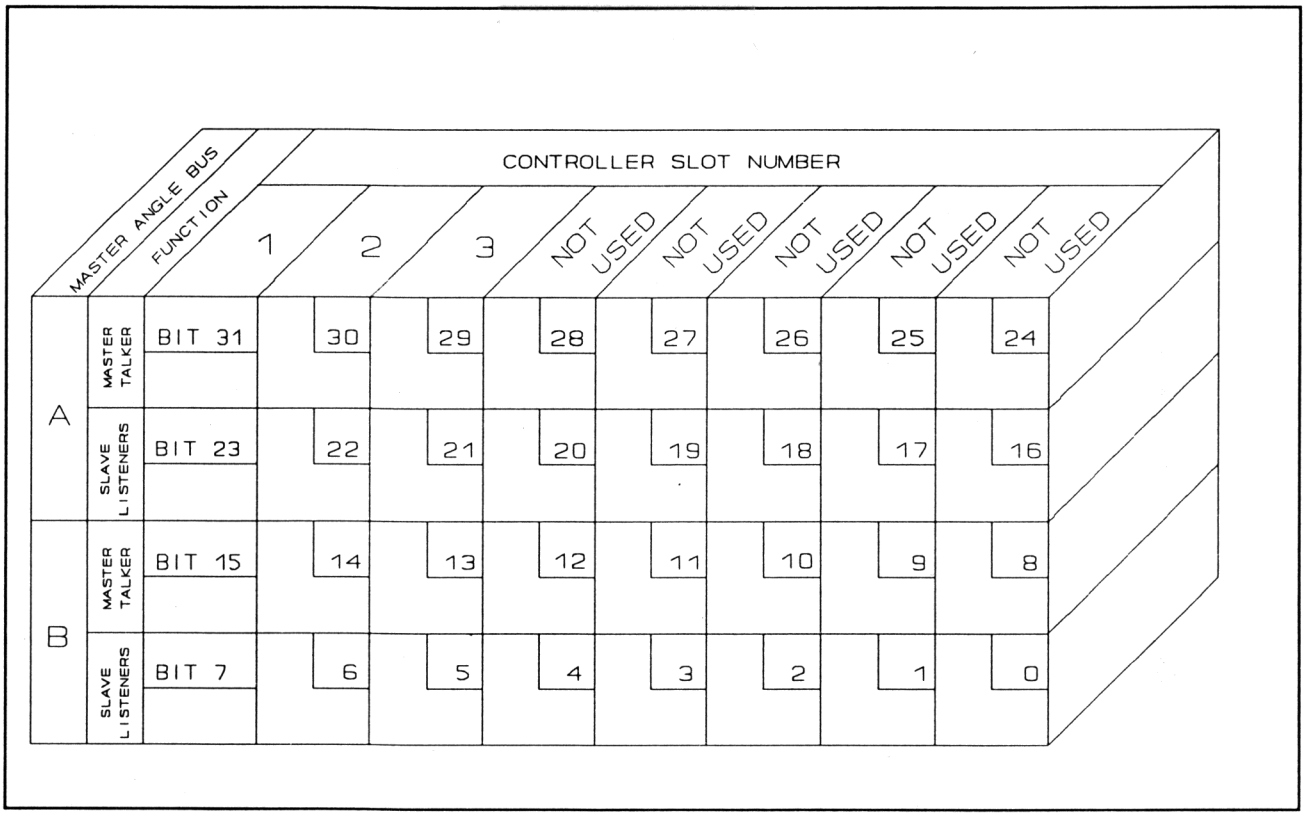
RETURNS:

Returns the current master angle bus configuration. The variable may be interpreted as follows:

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

MSC-850 MASTER ANGLE BUS CONFIGURATION CHART

get_map (continued)



MSC-250 MASTER ANGLE BUS CONFIGURATION CHART

SEE:

set_map
get_map_stat

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.

SEE:

set_map

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:

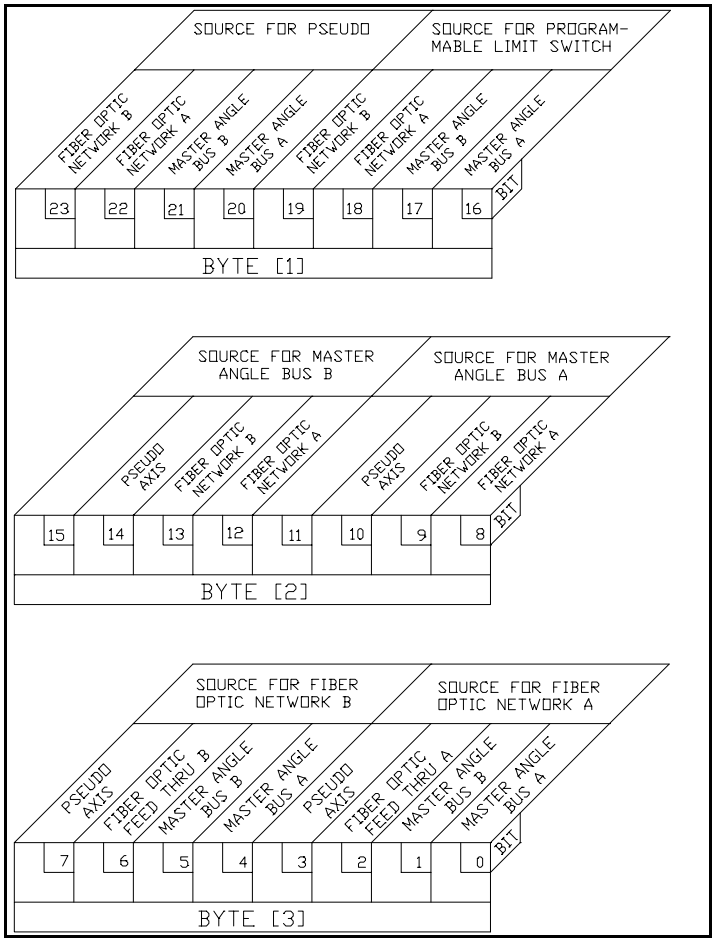
controller#	controller id#	
	Range: MSC-250	Always 3
	MSC-850/32	1 to 8
	MSC-850	1 to 8
variable	The currently defined 'mcf' value.	

DESCRIPTION:

'mcf' is an acronym for Multi Function Controller.

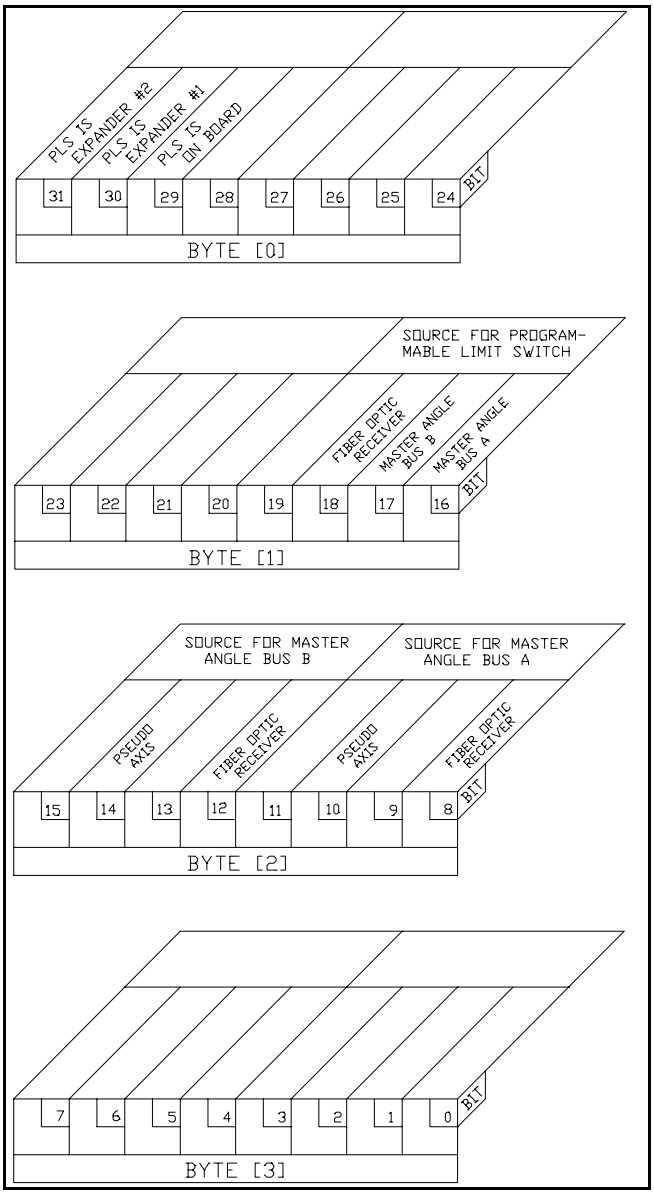
Returns the current definition of the specified multi function controller card, as defined by the last valid **set_mcf** instruction.

This 4-byte variable is used as shown in the following diagrams:



MCF-850 MULTI-FUNCTION CONFIGURATION

get_mcf (continued)



MSC-250 MULTI-FUNCTION CONFIGURATION

SEE:

set_mcf

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.

SEE:

set_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.

SEE:

set_pls_ang
set_pls_mask

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.

SEE:

port_set

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.

SEE:

get_com

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:

Code	Meaning
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)

SEE:

prep_profile
exec_profile

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.

SEE:

open
close
write
create
initialize
read
get_volume
load
save

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.

SEE:

No related instructions.

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.

SEE:

find_tm_cw
find_tm_ccw
find_mrk_cw
find_mrk_ccw

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.

SEE:

set_tmr

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.

SEE:

trap_pos
enable_hwi

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.

SEE:

None.

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.

SEE:

goto

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.

SEE:

gosub

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
<>	Not equal to
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to

RETURNS:

None.

SEE:

No related instructions.

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.

SEE:

if_bit_set

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.

SEE:

if_bit_clr

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.

SEE:

if_no_char

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.

SEE:

if_flag_on

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.

SEE:

if_flag_off

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.

SEE:

if_io_on

USAGE:

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

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.

SEE:

if_io_off

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-850/320	I,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.

SEE:

if_char

USAGE:

MSC-250:	if_no_char
MSC-850/32:	if_no_char
MSC-850:	if_no_char
MSC-800:	if_input_off

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.

SEE:

if_stat_on

USAGE:

MSC-250:	if_stat_off
MSC-850/32:	if_stat_off
MSC-850:	if_stat_off
MSC-800:	if_stat_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.

SEE:

if_stat_off

USAGE:

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

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.

SEE:

set_tmr
if_tmr_on

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.

SEE:

set_tmr
if_tmr_off

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 µs 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.

SEE:

No related instructions.

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 **MOTOR 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 Macroprogram instruction directs communication to occur.

RETURNS:

Position

SEE:

No related instructions.

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.

SEE:

create
write
read
close
get_space
get_volume
load
save
text
get_volume

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

SEE:

text
port_set
print
print_num

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 Macroprogram is loaded into the MSC Controller.

This instruction requires a 'label'.

RETURNS:

None.

SEE:

dim
equ
text

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 **AXIS BUSY** and **AXIS 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 Macroprogram instruction directs communication to occur.

RETURNS:

None.

SEE:

jog_cw
set_speed
set_ac_dc
f_decel

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 **AXIS BUSY** and **AXIS 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 Macroprogram instruction directs communication to occur.

RETURNS:

None.

SEE:

jog_ccw
set_speed
set_ac_dc
f_decel

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 **AXIS BUSY** and **AXIS 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 Macroprogram instruction directs communication to occur.

RETURNS:

None.

SEE:

track_spd

USAGE:

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

let

SYNTAX:

label	let	variable=operand1 opcode operand2	
label	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 Macroprogram 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.

let (continued)

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 Macroprogram 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.

SEE:

let_byte

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 Macroprogram 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 Macroprogram 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.

SEE:

let

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 macroprogram file specified by 'file_name'.

If the specified macroprogram is found, it will replace entirely the macroprogram currently residing in the MSC.

When the 'load' is completed, program execution begins automatically. If the file does not contain a macroprogram, 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 Table 15.3.2.

SEE:

open
close
create
read
write
initialize
get_space
get_volume
load

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.

Lock #	Lock Method
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.

lock (continued)

Lock #	Lock Method
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 MOTOR 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.

lock (continued)

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 sped at the specified acceleration rate and relocks using the above equation. The MOTOR 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.

lock (continued)

Lock #	Lock Method

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:

master angle = 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.

lock (continued)

Lock #	Lock Method
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 ot 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.

SEE:

set_map
ratio
set_trig_pw
set_trig_cam
set_cam_ptr

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.

SEE:

drive_on
drive_off
enable

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 MSC Toolkit 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 macroprogram.

RETURNS:

None.

SEE:

No related instructions.

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.

SEE:

No related instructions.

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.

SEE:

No related instructions.

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.

SEE:

create
initialize
write
read
close
get_space
get_volume
load
save

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.

over_draw (continued)

NOTE: This instruction can only be used in conjunction with hardware interrupts.

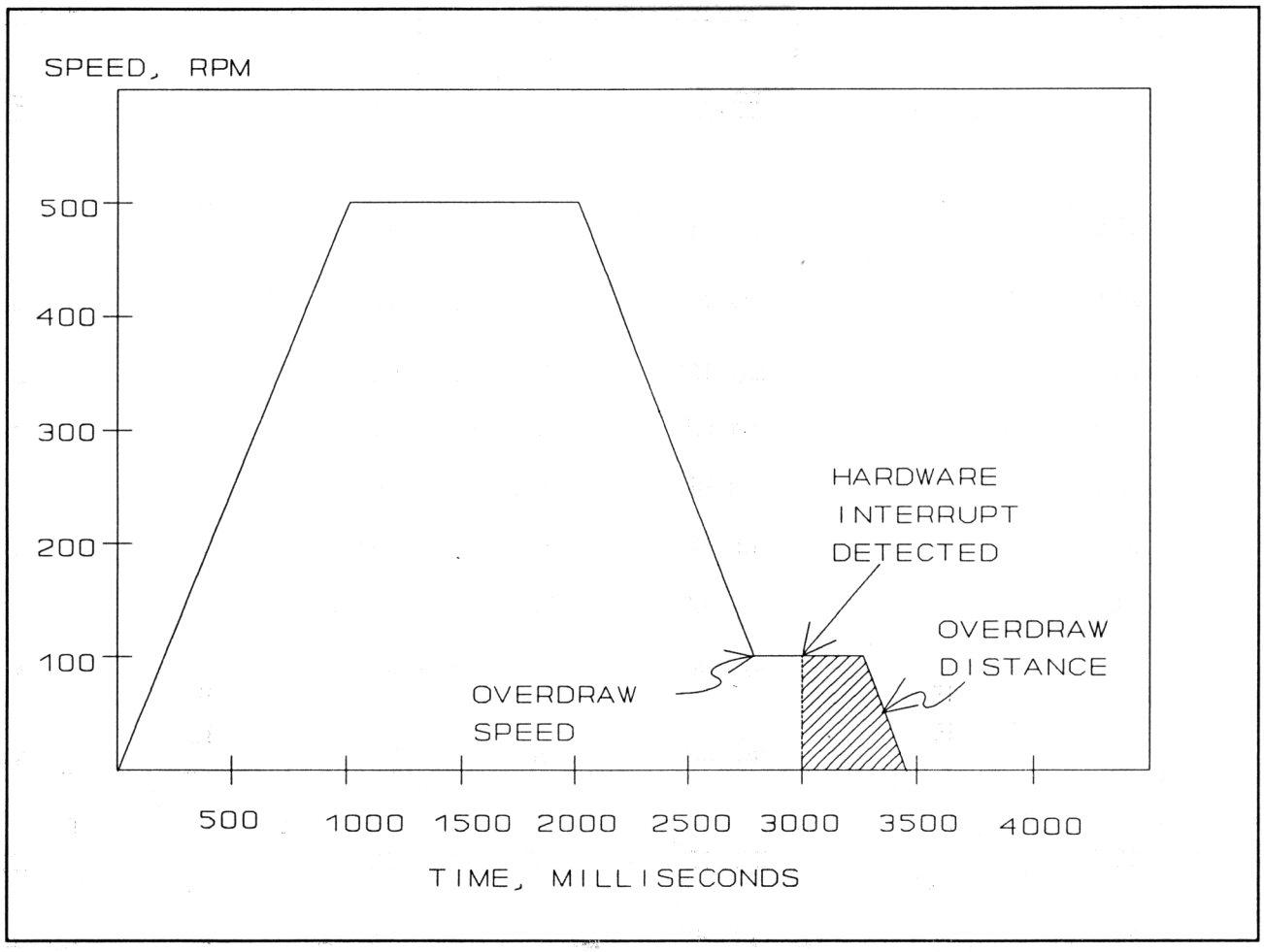


Figure 18.1 - TYPICAL OVERDRAW MOTION PROFILE

over_draw (continued)

EXAMPLE:

```
busy_1      equ         94          axis busy flag
hwi_armed_1 equ         91          interrupt armed flag
             .
             .
            set_speed   1,400       400 rpm
            set_ac_dc   1,100       100 rev/sec^2
             .
             .
            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.

SEE:

enable_hwi
disable_hwi

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.

port_set (continue)

RETURNS:

None.

SEE:

print_num
print
input
stop_input
get_pq_space
if_char
if_no_char

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.

SEE:

index

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.

SEE:

exec_profile
get_pstat

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.

SEE:

No related instructions.

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.

SEE:

set_local

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.

SEE:

input
text

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.

SEE:

print
text

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'.

SEE:

No related instructions.

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.

SEE:

set_map
unlock
lock

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.

SEE:

open
close
write
create
initialize
get_space
get_volume
load
save

USAGE:

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

read_offset

SYNTAX:

label read_offset controller#,variable

PARAMETERS:

controller#c	ontroller 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'.

SEE:

set_offset

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.

SEE:

goto
gosub

return_sub

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 Macroprogram status **STACK OVERFLOW** will occur if more than 20 levels are used.

If a **return_sub** instruction is encountered without a corresponding **gosub**, a Macroprogram status of **STACK UNDERFLOW** may occur.

RETURNS:

None.

SEE:

gosub

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 macroprogram 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.

SEE:

open
close
create
read
write
initialize
get_space
get_volume
load

USAGE:

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

select

SYNTAX:

label select variable

PARAMETERS

variable Variable to be tested.

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.

SEE:

case
exit_select
default
end_select

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.

SEE:

set_speed

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 (ACY-850)

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.

SEE:

No related instructions.

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.

SEE:

clr_bit
if_bit_set
if_bit_clr

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.

SEE:

get_cam_ptr

USAGE:

MSC-250:	set_cam_ptr
MSC-850/32:	set_cam_ptr
MSC-850:	set_cam_ptr
MSC-800:	cam_pointer

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.

SEE:

clr_flag

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.

SEE:

set_gl_cw

USAGE:

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

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.

SEE:

set_local
set_gl_cw

USAGE:

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

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.

SEE:

set_gl_ccw

USAGE:

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

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.

SEE:

set_local
set_gl_ccw

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.

SEE:

clr_hi_scan

USAGE:

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

set_home (ACY-850)

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 Macroprogram 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.

SEE:

No related instructions.

set_home (ACY-850) continued

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.

SEE:

clr_local
set_gl_cw
set_gl_ccw

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.

SEE:

set_gl_cw
set_gl_ccw

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

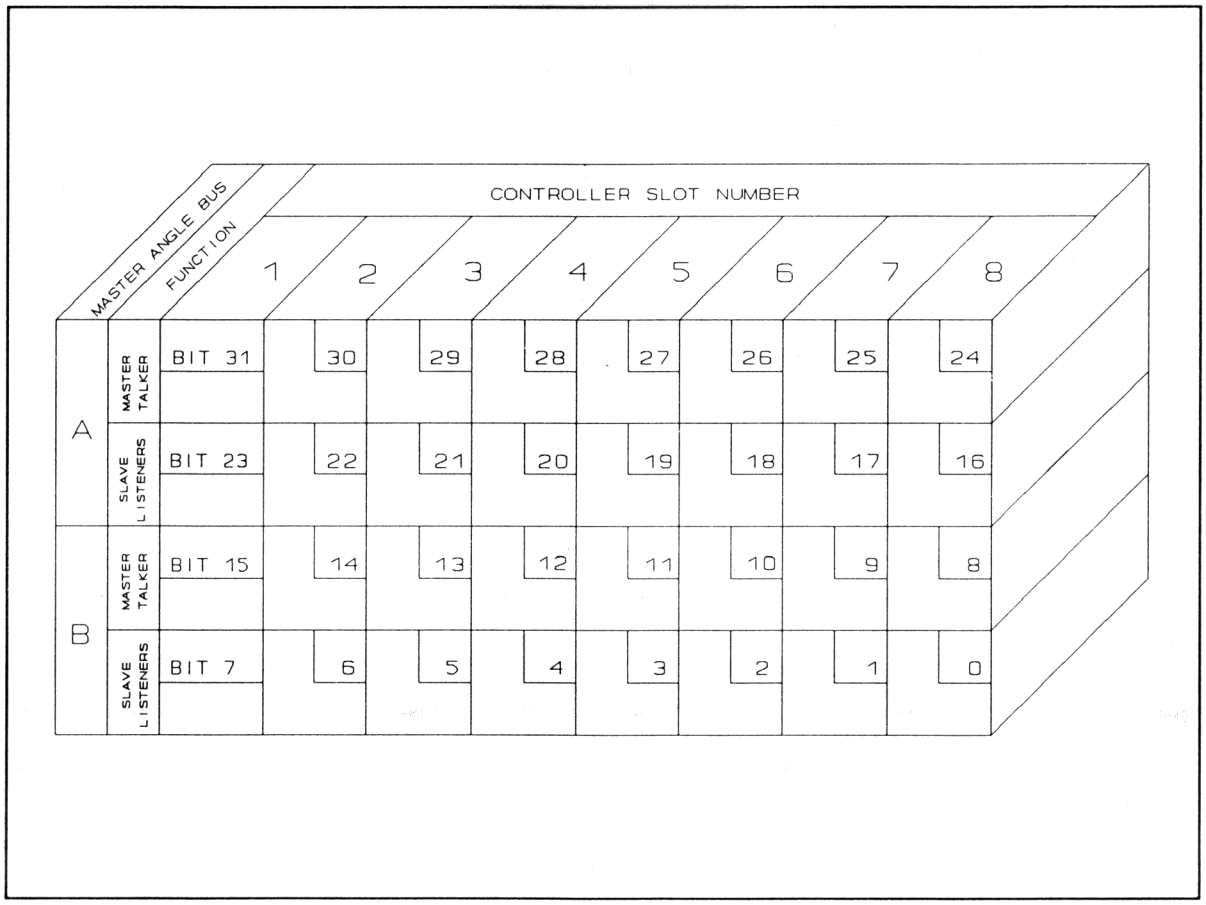
PARAMETERS:

variable The desired map value.

DESCRIPTION:

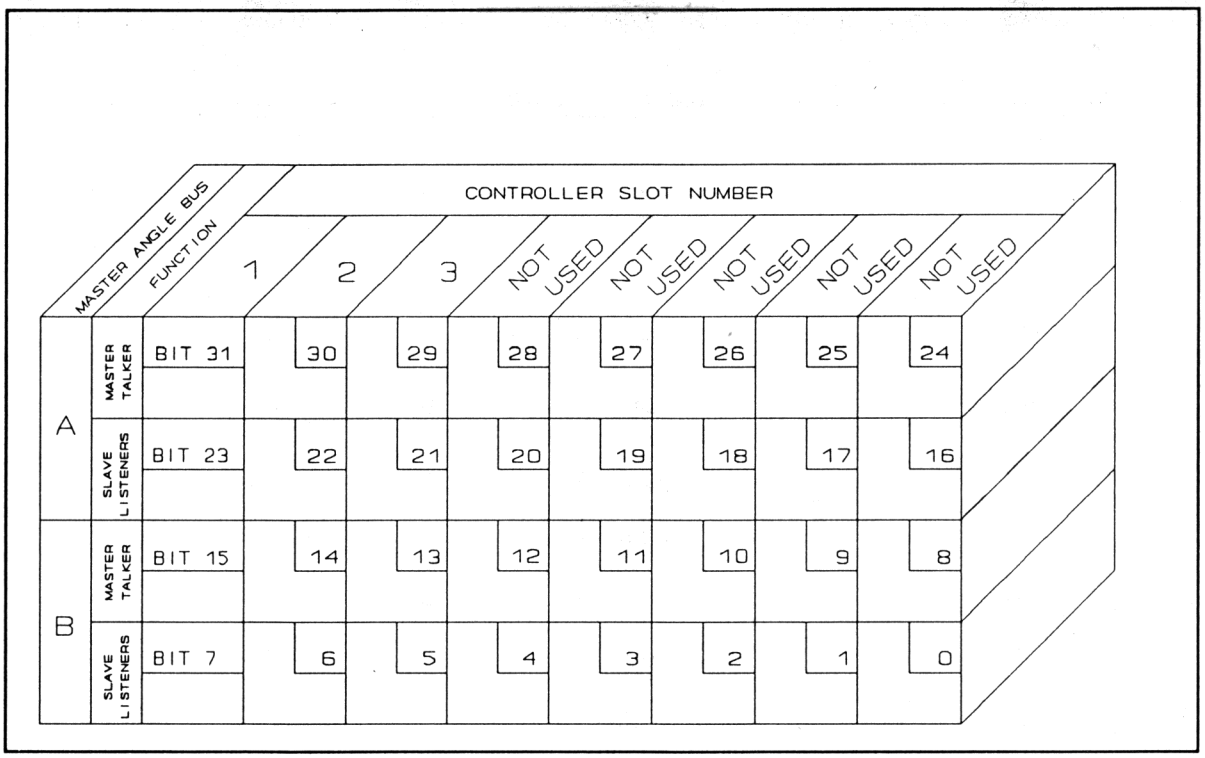
'map' is an acronym for Master Angle Passing.

The **set_map** instruction defines the master angle configuration within a system unit. The proper value of 'variable' can be determined from the following table:



MSC-850 MASTER ANGLE BUS CONFIGURATION

set_map (continued)



MSC-250 MASTER ANGLE BUS CONFIGURATION CHART

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.

SEE:

get_map
get_map_stat

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

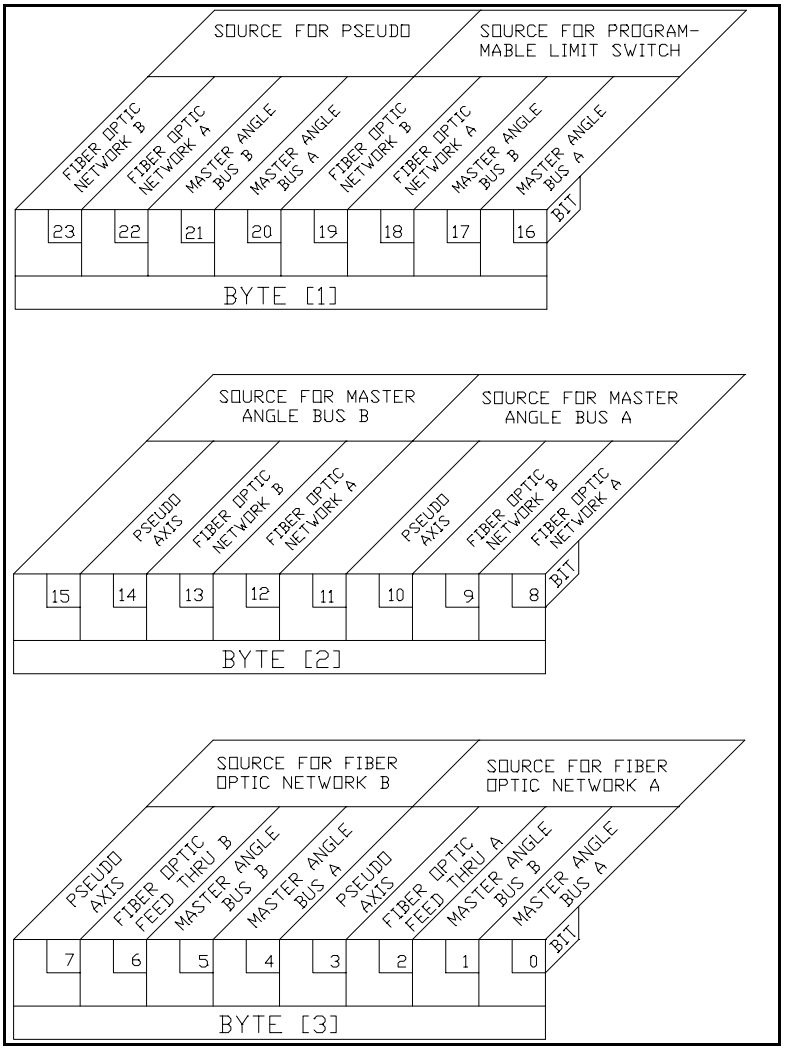
PARAMETERS:

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.



MSC-850/MCF-850 MULTI-FUNCTION CONFIGURATION

set_mcf (continued)

SEE:

get_mcf
set_map

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.

SEE:

set_pls_ang (HPL-850 card)
set_pls_cnt (HPL-850 card)

USAGE:

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

set_mcf (ACR-850 or ACE-850 or 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_mode turn_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_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" -----
!
```

set_mcf (continued)

```
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.

SEE:

analog_out (ACR-850 or ACE-850 or ACY-850 cards)
drive_on

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

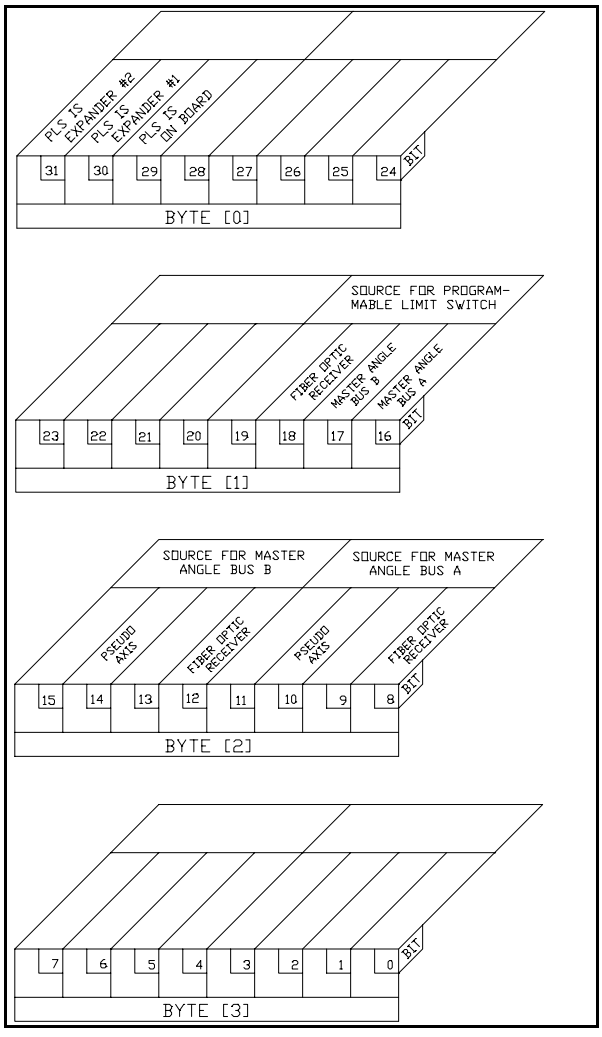
PARAMETERS:

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.



MSC-250 MULTI-FUNCTION
CONFIGURATION

set_mcf (continued)

RETURNS:

None.

SEE:

get_mcf
set_map

USAGE:

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

set_mcf (MSC-250 controller 1 and 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_mode turn_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" -----
!
```

set_mcf (continued)

```
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.

SEE:

analog_out (MSC-250 controller 1 and 2)

USAGE:

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

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.

SEE:

No related instructions.

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.

SEE:

over_draw

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 (representing 1 full master cycle).
off_angle	Turn OFF module# at this angle in bits.
	Range: 0 to 4095 (representing 1 full master cycle).
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 module. 'module#' 0 to 15 are mapped to the PLS outputs. 'module#' 16 to 23 are mapped internally to the controller status flags.

Each time a **set_pls_ang** instruction is issued, the axis controller executes a sorting algorithm and will activate its' CALCULATING flag. The Macroprogram 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". The following two examples show the effect of using the default "rollover point" versus a longer "rollover point", using the **set_pls_cnt** instruction.

EXAMPLE 1:

ROLLOVER POINT = 4096
PLS ON ANGLE = 1024
PLS OFF ANGLE = 2047

```
|----------|----------|----------|----------|  MASTER BITS
0       1023     2047     3071     4095

|---off---|----on----|---off---|---off---|  PLS STATUS
```

set_pls_ang (continued)

EXAMPLE 2:

ROLLOVER POINT = 8192
PLS ON ANGLE = 1024
PLS OFF ANGLE = 2047

```
|-------------------|-------------------|-------------------|-------------------|  MASTER BITS
0                2047               4095               6143               8191

|--------off--------|--------on---------|---------off--------|--------off---------|  PLS STATUS
```

SEE:

get_pls_out
set_pls_mask
set_pls_cnt (MCF-850 card)

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 the "rollover point" in bits (representing 1 full master cycle).
off_angle	Turn OFF module# at this angle in bits.
	Range: 0 to the "rollover point" in bits (representing 1 full master cycle).
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 module. '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' CALCULATING flag. The Macroprogram 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 HPL-850 controller, the "rollover point" can be modified to be a value (in bits) between 256 and 8,388,607. 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 range up to the "rollover point" set using the **set_pls_cnt** instruction. The following two examples show the effect of having a longer master cycle using the **set_pls_cnt** instruction.

EXAMPLE 1:

ROLLOVER POINT = 4096
PLS ON ANGLE = 1024
PLS OFF ANGLE = 2047

```
|----------|----------|----------|----------|  MASTER BITS
0       1023     2047     3071     4095

|---off---|----on----|---off---|---off---|  PLS STATUS
```

set_pls_ang (continued)

EXAMPLE 2:

ROLLOVER POINT = 12000
PLS ON ANGLE = 3000
PLS OFF ANGLE = 5999

```
|-------------------|-------------------|-------------------|-------------------|  MASTER BITS
0             2999            5999            8999           11999

|--------off--------|--------on---------|---------off--------|--------off---------|  PLS STATUS
```

RETURNS:

None.

SEE:

get_pls_out
set_pls_mask
set_pls_cnt (HPL-850 card)

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 (representing 1 full master cycle).
off_angle	Turn OFF module# at this angle in bits. Range: 0 to 4095 (representing 1 full master cycle).
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 module. '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' CALCULATING flag. The Macroprogram 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. This value is always 4096 for the MSC-250.

EXAMPLE 1:

ROLLOVER POINT = 4096
PLS ON ANGLE = 1024
PLS OFF ANGLE = 2047

```
|----------|----------|----------|----------|  MASTER BITS
0      1023     2047     3071     4095

|---off---|----on----|---off---|---off---|  PLS STATUS
```

RETURNS:

None.

SEE:

get_pls_out
set_pls_mask

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 PLS function 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.

count	master bits per PLS cycle
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.

SEE:

set_pls_ang (MCF-850 card)

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.

SEE:

set_mcf (HPL-850 card)
set_pls_ang (HPL-850 card)

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.

SEE:

get_pls_mask

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.

SEE:

set_pls_ang

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.

SEE:

set_ac_dc

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.

SEE:

swi_if_on
swi_if_off
enable_swi
disable_swi
clr_swi
clr_all_swi

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_tmr timer_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.

SEE:

if_tmr_on
if_tmr_off

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.

SEE:

lock

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.

SEE:

prep_profile
lock

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.

SEE:

digi_comp

USAGE:

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

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.

SEE:

input
if_char
if_no_char

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.

SEE:

swi_if_on
enable_swi
disable_swi
clr_swi
clr_all_swi
set_swi_mask

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.

SEE:

enable_swi
disable_swi
swi_if_off
clr_swi
clr_all_swi
set_swi_mask

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, tyhe 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 AXIS 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.

switch_cam (continued)

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 AXIS 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 AXIS 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 AXIS 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

switch_cam (continued)

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 AXIS 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.

SEE:

lock

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 macroprogram and set the system fault bit in the MSC status word.

An **f_decel** instruction is executed for all controllers.

RETURNS:

None.

SEE:

sys_return

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 macroprogram and set the system return bit in the MSC status word.

An **f_decel** instruction is executed for all controllers.

RETURNS:

None.

SEE:

sys_fault

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.

SEE:

No related instructions.

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.

SEE:

print
input

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 **AXIS BUSY** and **AXIS 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 Macroprogram instruction directs communication to occur.

RETURNS:

None.

SEE:

l_track_spd

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.

SEE:

get_trap_pos

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.

SEE:

turn_on
if_io_on
if_io_off

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.

SEE:

turn_off
if_io_on
if_io_off

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

SEE:

lock

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 **AXIS BUSY** and **AXIS 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 Macroprogram instruction directs communication to occur.

RETURNS:

None.

SEE:

vel_cw

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 **AXIS BUSY** and **AXIS 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 Macroprogram instruction directs communication to occur.

RETURNS:

None.

SEE:

vel_ccw

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.

SEE:

open
close
read
create
initialize
get_space
get_volume
load
save

USAGE:

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

APPENDIX A Macroprogram Instruction Listing

INSTRUCTION	COMMENTS	MSC-100	MSC-800	MSC-850	MSC-250	MSC-850/32
analog_in			✔	✔	✔	✔
analog_out			✔	✔	✔	✔
analog_rt			✔	✔	✔	✔
analog_zo			✔	✔	✔	✔
begin_cam			✔	✔	✔	✔
begin_data		✔	✔	✔	✔	✔
blk_io_in		✔		✔	✔	✔
blk_io_out		✔	✔	✔	✔	✔
calc_cam_sum				✔	✔	✔
calc_unit_cam				✔	✔	✔
cam			✔	✔	✔	✔
cam_data			✔	✔	✔	✔
cam_pointer	same as 'set_cam_ptr'	✔				
case		✔	✔	✔	✔	✔
close			✔	✔	✔	✔
clr_all_swi			✔	✔	✔	✔
clr_bit				✔	✔	✔
clr_event	obsolete - same as 'clr_swi'	✔	✔			
clr_flag		✔	✔	✔	✔	✔
clr_hi_scan						✔
clr_local		✔	✔	✔	✔	✔
clr_swi		✔	✔	✔	✔	✔
create			✔	✔	✔	✔
data		✔	✔	✔	✔	✔
declare		✔	✔	✔	✔	✔
default		✔	✔	✔	✔	✔
digi_comp		✔	✔	✔	✔	✔
dim		✔	✔	✔	✔	✔
disable_hwi				✔	✔	✔
disable_swi	same as 'swi_off'			✔	✔	✔
drive_off	same as 'master'			✔	✔	✔
drive_on	same as 'enable'			✔	✔	✔
enable	same as 'drive_on'	✔	✔			
enable_hwi				✔	✔	✔
enable_swi	same as 'swi_on'			✔	✔	✔
end_cam			✔	✔	✔	✔
end_data		✔	✔	✔	✔	✔
end_select		✔	✔	✔	✔	✔
equ		✔	✔	✔	✔	✔
events_off	same as 'disable_swi'	✔				
events_on	same as 'enable_swi'	✔				
exec_profile		✔	✔	✔	✔	✔
exit_select		✔	✔	✔	✔	✔
f_decel	also used with HWI	✔	✔	✔	✔	✔
find_mrk_ccw				✔	✔	✔
find_mrk_cw				✔	✔	✔

INSTRUCTION	COMMENTS	MSC-100	MSC-800	MSC-850	MSC-250	MSC-850/32
find_tm_ccw				✔	✔	✔
find_tm_cw				✔	✔	✔
get_act_spd				✔	✔	✔
get_angle				✔	✔	✔
get_cam_cnt				✔	✔	✔
get_cam_end				✔	✔	✔
get_cam_ptr			✔	✔	✔	✔
get_cam_strt				✔	✔	✔
get_cam_sum				✔	✔	✔
get_com	same as 'store_com'			✔	✔	✔
get_fol_err				✔	✔	✔
get_for_ang				✔	✔	✔
get_frame	obsolete		✔			
get_mcf				✔	✔	✔
get_map				✔	✔	✔
get_map_stat				✔	✔	✔
get_mpos			✔			
get_mspd			✔			
get_pls_mask				✔	✔	✔
get_pls_out				✔	✔	✔
get_pos	same as 'store_pos'			✔	✔	✔
get_pq_space					✔	✔
get_pstat		✔	✔	✔	✔	✔
get_space			✔	✔	✔	✔
get_status		✔	✔	✔		✔
get_t_mrk				✔	✔	✔
get_time	same as 'store_time'			✔	✔	✔
get_trap_pos				✔	✔	✔
get_volume			✔	✔	✔	✔
gosub		✔	✔	✔	✔	✔
goto		✔	✔	✔	✔	✔
if		✔	✔	✔	✔	✔
if_bit_clr				✔	✔	✔
if_bit_set				✔	✔	✔
if_char	same as 'if_input_on'			✔	✔	✔
if_flag_off		✔	✔	✔	✔	✔
if_flag_on		✔	✔	✔	✔	✔
if_input_off	same as 'if_no_char'	✔	✔			
if_input_on	same as 'if_char'	✔	✔			
if_io_off		✔	✔	✔	✔	✔
if_io_on		✔	✔	✔	✔	✔
if_no_char	same as 'if_input_off'			✔	✔	✔
if_stat_off		✔	✔	✔	✔	✔
if_stat_on		✔	✔	✔	✔	✔
if_tmr_off		✔	✔	✔	✔	✔
if_tmr_on		✔	✔	✔	✔	✔
incr_com				✔	✔	✔
index	also used with HWI	✔	✔	✔	✔	✔

INSTRUCTION	COMMENTS	MSC-100	MSC-800	MSC-850	MSC-250	MSC-850/32
initialize			✔	✔	✔	✔
input		✔	✔	✔	✔	✔
integer		✔	✔	✔	✔	✔
jog_ccw		✔	✔	✔	✔	✔
jog_cw		✔	✔	✔	✔	✔
l_track_spd		✔	✔	✔	✔	✔
let		✔	✔	✔	✔	✔
let_byte		✔	✔	✔	✔	✔
load			✔	✔	✔	✔
lock	also used with HWI		✔	✔	✔	✔
master	same as 'drive_off'	✔	✔			
msc_type		✔	✔	✔	✔	✔
no_op		✔	✔	✔	✔	✔
off_event	same as 'swi_if_off'	✔	✔			
off_swi	same as 'swi_if_off'	✔				
on_event	same as 'swi_if_on'	✔	✔			
on_swi	same as 'swi_if_on'	✔				
open			✔	✔	✔	✔
over_draw	used with HWI only			✔	✔	✔
p_vector	obsolete		✔			
port_set		✔	✔	✔	✔	✔
pos_rewind	obsolete		✔			
position	also used with HWI	✔	✔	✔	✔	✔
prep_profile	same as 'profile'			✔	✔	✔
preset				✔	✔	✔
print		✔	✔	✔	✔	✔
print_num		✔	✔	✔	✔	✔
profile	same as 'prep_profile'	✔	✔			
rand_int		✔	✔	✔	✔	✔
ratio	also used with HWI		✔	✔	✔	✔
read			✔	✔	✔	✔
read_offset		✔	✔	✔	✔	✔
reset	obsolete - same as 'drive_on'	✔	✔			
restart_at		✔	✔	✔	✔	✔
restart_cp	obsolete		✔			
return_sub		✔	✔	✔	✔	✔
rewind_cp	obsolete		✔			
save			✔	✔	✔	✔
select		✔	✔	✔	✔	✔
set_ac_dc		✔	✔	✔	✔	✔
set_acy_cnt					✔	✔
set_bit				✔	✔	✔
set_cam_ptr	same as 'cam_pointer'			✔	✔	✔
set_flag		✔	✔	✔	✔	✔
set_gl_ccw		✔	✔	✔	✔	✔
set_gl_cw		✔	✔	✔	✔	✔
set_hi_scan						✔
set_home				✔	✔	✔

INSTRUCTION	COMMENTS	MSC-100	MSC-800	MSC-850	MSC-250	MSC-850/32
set_local		✔	✔	✔	✔	✔
set_map				✔	✔	✔
set_mcf				✔	✔	✔
set_offset		✔	✔	✔	✔	✔
set_ovd_mode				✔	✔	✔
set_pls_ang				✔	✔	✔
set_pls_cnt				✔	✔	✔
set_pls_mask				✔	✔	✔
set_pls_time				✔	✔	✔
set_speed		✔	✔	✔	✔	✔
set_swi_mask					✔	✔
set_tmr		✔	✔	✔	✔	✔
set_trig_cam				✔	✔	✔
set_trig_pw		✔	✔	✔	✔	✔
set_vgain				✔	✔	✔
start_cp	obsolete		✔			
stop_input		✔	✔	✔	✔	✔
store_com	same as 'get_com'	✔	✔			
store_frm	obsolete		✔			
store_pos	same as 'get_pos'	✔	✔			
store_time	same as 'get_time'	✔	✔			
swi_if_off	same as 'off_swi'			✔	✔	✔
swi_if_on	same as 'on_swi'			✔	✔	✔
swi_off	same as 'disable_swi'	✔	✔			
swi_on	same as 'enable_swi'	✔	✔			
switch_cam				✔	✔	✔
sys_fault		✔	✔	✔	✔	✔
sys_return		✔	✔	✔	✔	✔
test_mode				✔	✔	✔
text		✔	✔	✔	✔	✔
track_spd		✔	✔	✔	✔	✔
trap_pos	used with HWI only			✔	✔	✔
turn_off		✔	✔	✔	✔	✔
turn_on		✔	✔	✔	✔	✔
unlock			✔	✔	✔	✔
vel_ccw		✔	✔	✔	✔	✔
vel_cw		✔	✔	✔	✔	✔
write			✔	✔	✔	✔

APPENDIX B CUSTOM SERIAL PORT CONFIGURATION FOR THE MSC SOFTWARE TOOLKIT

The MSC Software Toolkit, beginning with SFO3110R0, provides a feature to allow the use of non-standard serial communications configurations. This feature is needed when:

1. Using devices with port addresses other than those for the standard COM1 and COM2 ports. Typical examples include COM3 and COM4, and RS485 cards.

2. Using devices with IRQ levels other than the standards defined for COM1 and COM2.

3. Using various multiport serial communications cards like the Digiboard PC/X series.

Configuration of the Toolkit for non-standard communication ports is accomplished using a data file which contains port address and interrupt information for each communication channel in use. A typical file is shown in Figure 1.

Notice that the file consists of a series of three entries for each COM port defined, separated from the next group by a blank line. The word END indicates the end of the definition file.

For each COM port defined, you will need to know the base address of the UART registers and the interrupt request (IRQ) setting. In our example file, the standard settings for COM1 and COM2 are shown. The remaining six entries are typical of factory default settings for the first six ports of a Digiboard PC/8 multiport serial card. Up to 16 COM ports may be defined in this manner.

Table I Configuration File Format

COM1	COM5
0x3f8	0x110
IRQ4	IRQ5
COM2	COM6
0x2f8	0x118
IRQ3	IRQ5
COM3	COM7
0x100	0x120
IRQ5	IRQ5
COM4	COM8
0x108	0x128
IRQ5	IRQ5
→	END

GLOSSARY OF TERMS

ACC-800 A card used with an MSC-800 to control a resolver based motor/drive unit

ACCEL/DECEL The rate of change to achieve a desired speed. Unit of measure is revs/sec^2.

ACE-850 A card used with an MSC-850 to control an encoder based motor/drive axis

ACM-800 A controller card used with an MSC-800 to send and receive analog signals

ACM-850 A card used with an MSC-850 to send/receive analog signals

ACTUATOR A device which creates mechanical motion by converting various forms of energy to mechanical energy.

AXIS An independent motor/drive unit which is controlled by an IIS type controller

AXIS CONTROLLER A card which is used with either an MSC-800 or MSC-850 to control an independent motor/drive unit

CLOSED LOOP A regulating device in which the actuator position is sensed, and a signal proportional to this position (feed-back position) is compared with a signal proportional to the desired actuator position (command position). The difference between these signals is the error signal. The error signal causes a change in the actuator so as to force this signal to be zero.

COMMAND SIGNAL GENERATOR A device that supplies a command position signal to an automatic control system. This signal represents the desired motion of the actuator that is required to accomplish a task such as making a part. This signal is usually in the form of an electrical signal.

COMMUNICATIONS The transmission of information from one device to another. The information can take many forms such as command signals, device status and fault conditions.

COMMUTATOR The part of the rotating armature of a motor that causes the electrical current to be switched to various armature windings. The proper sequenced switching of the windings creates the motor torque. The commutator also provides the means to transmit the electrical current from the stationary body of the motor to moving rotor.

COMPARATOR A device where the feedback signal is subtracted from the command signal. The difference output of the comparator is called the error signal.

DC DRIVE An electronic control unit for running DC motors. The DC drive converts AC line current to a variable DC current to control a DC motor. The DC drive has a signal input that controls the torque and speed of the motor.

ENCODER A type of feedback device which converts mechanical motion into electrical signals to indicate actuator position. Typical encoders are designed with a printed disc and a light source. As the disc turns with the actuator shaft, the light source shines through the printed pattern onto a sensor. The light transmission is interrupted by the patterns on the disk. These interruptions are sensed and converted to electrical pulses. By counting these pulses, actuator shaft position is determined.

FLAG A bit in memory used by the programer to evaulate
action to be taken. A program branch may be executed depending on the true or false result of a bit test. There are four types of flags supported by the Macroprogram language; axis status flags, I/O status flags, timer status flags and user flags.

HOME A known point of reference frequently referred to as "global zero". Used as a point of reference for absolute positioning.

HOST A computer system whose function is to monitor and coordinate the processes of other devices. A host computer will typically coordinate motion control functions as well as their interaction with other machine processes.

HPL-850 A card used with an MSC-850 multi-axis controller which provides high performance programmable limit switches.

I/O The reception and transmission of information (Input/Output) between control devices. In modern control systems, I/O has two distinct forms. In one, it refers to switches, relays, etc. In the other form, I/O refers to analog signals that are continuous in nature such as speed, temperature, flow, etc.

INDEX An **index** instruction will move the motor shaft an absolute distance relative to the current position.

INERTIA The measure of an object's resistance to a change in its current velocity.

INITIALIZE To execute a series of macro program instructions in order to teach an MSC axis controller an absolute zero reference.

INSTRUCTION A Macroprogram command to the MSC. All instructions have the following format:
have the following format:

label instruction parameters comment

INSTABILITY Undesirable motion of an actuator that is different from the commanded motion. Instability can take the form of irregular speed or hunting of the final rest position.

IOE-800 A rack containing up to 16 interchangeable I/O modules. The modules may be AC or DC and any combination of input and output.

oan MSC-100 will support up to 2 IOE-800 racks
oan MSC-800 will support up to 4 IOE-800 racks
oan MSC-850 will support up to 4 IOE-800 racks

This device is also commonly referred to as an I/O Expander.

JOG A **jog** instruction will move the motor shaft in the specified direction using the predefined speed and accel/decel rates. Once at speed, the motor will continue to jog until commanded to stop.

LINE SHAFT A shaft rotated by the primary motor drive. The line shaft transmits power from the motor to a load or series of loads. In the multiple load case, the motions of the loads are synchronized to one another because they are driven from a common shaft.

MSC TOOLKIT The personal computer based software package used to edit, compile and debug Macroprograms developed for MSC controllers.

MACROPROGRAM A program written in IIS' basic-like language for the MSC family of motion controllers.

MICROPROCESSOR A miniaturized computer system that executes instructions in a sequential manner. The sequential instructions form a control strategy for devices that may be connected to the system.

The sequential instructions are loaded into the microprocessor and can be easily changed or modified. Modern microprocessors are small electronic devices that execute a wide range of instructions at speeds as high as 1,000,000 instructions per second.

MOTOR DIRECTION The direction that a motor shaft is turning is determined by the facing the front of the motor shaft. From this perspective, direction is determined to be clockwise or counter-clockwise.

MAC-800 The main processor within the MSC-800 System Unit, which communicates with the axis controllers as well as external devices such as Personal Computers, I/O Expanders and operator interface devices.

MCF-850 A card used with an MSC-850 controller which provides optional high level functions such as 32K of non-volatile data storage, pseudo axis capability, and programmable limit switches.

MSC-100 A single axis, programmable servo motion controller

MSC-800 A multi-axis, programmable servo motion controller capable of synchronously controlling from 1 to 8 axes.

MSC-850 A multi-axis, programmable servo motion controller capable of synchronously controlling from 1 to 8 axes. This controller provides more advanced capability than the MSC-800.

MULTI-AXIS CONTROLLER A system designed to control more than one actuator. This type of controller allows the actuators to work independently or as a coordinated group to perform more complex tasks.

ON-BOARD I/O The I/O module slots that are provided with an MSC System Unit. Each MSC-800 or MSC-850 may have up to 8 onboard input and/or output modules.

OPEN-LOOP SYSTEM A system where a command signal results in actuator movement which is not sensed and therefore not corrected for error. Open loop means no feedback.

OPERATOR INTERFACE A device that allows the operator to communicate with a machine. This device typically has a keyboard or thumbwheel to enter instructions into the machine. It may also have a display device that allows the machine to display messages. An example of an operator interface is the OPI-1.

OSCILLATION Undesirable motion of an actuator that is different from the command motion. See INSTABILITY.

PARAMETER A value required for correct execution of an instruction. Each instruction has a parameter list which is used by the controller to execute the instruction. An example would be the **index** instruction. The controller needs to know which axis to index and the distance to index.

PLS-850 A rack containing up to 15 output modules for use with either an MCF-850 programmable limit switch feature, or an HPL-850 high performance programmable limit switch feature.

PERIPHERAL Various kinds of devices that operate in conjunction with an MSC controller. Examples of these are PLC's, joysticks, computers, and operator interfaces.

POSITION ERROR The difference between the present actuator position (feedback) and the desired position (command).

POSITION FEEDBACK Present actuator position as measured by a position transducer.

PROGRAMMABLE LOGIC CONTROLLER (PLC) (PLC) An electronic device that scans discrete (on/off) type inputs and controls discrete (on/off) type outputs. The relationship between the inputs and outputs are logical statements that are programmable by the user.

RESOLVER A type of feedback device which converts mechanical position into an electrical signal. A resolver is a variable transformer that divides the impressed AC signal into two outputs, referred to as sine and cosine output signals. The comparison of these two signals is used to determine the absolute position of the resolver shaft.

ROTOR The rotating part of a magnetic structure. In a motor, the rotor is connected to the motor shaft.

SERVO An automatic control device for controlling large amounts of power by means of very small amounts of power and automatically correcting the performance of the mechanism. Servo systems are closed loop systems.

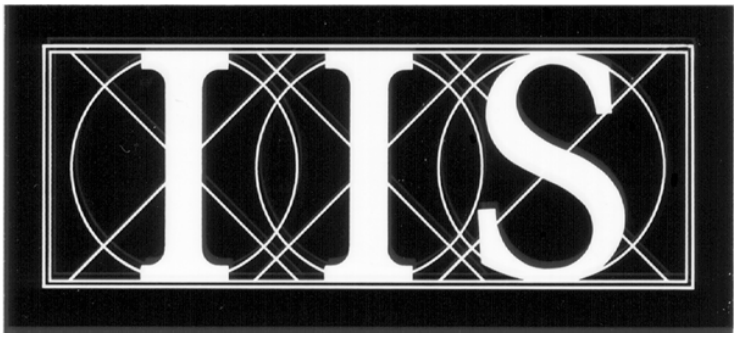
SERVO AMPLIFIER An electronic device which produces the winding current for a servo motor. The amplifier converts a low level control signal into high voltage and current levels to produce torque in the motor.

SERVO MOTOR An actuator which converts electrical energy (winding current) to mechanical energy (torque). Servo motor construction is optimized to provide maximum torque with minimum rotor inertia.

SYSTEM UNIT A system unit consists of an ENC850 Unit Enclosure and MAC850 Main Processor. The Unit Enclosure is a metal cabinet with a motherboard mounted on the back plate. The Main Processor plugs into a connector on the motherboard and provides the processing power to run the operating system and customer application software. The motherboard serves as a multipurpose backplane in which the command bus, on-board I/O bus, two angle busses, and power distribution conductors reside.

INDEX

IB-11C001



INDUSTRIAL INDEXING SYSTEMS INC.

626 FISHERS RUN
VICTOR, NEW YORK 14564

(585) 924-9181
FAX: (585) 924-2169

PRINTED IN USA