Programming Reference

HP 16500A
Logic Analysis System

Manual Part Number 16500-90908

Printed in U.S.A. September 1988

Product Warranty

This Hewlett-Packard product has a warranty against defects in material and workmanship for a period of three years from date of shipment. During warranty period, Hewlett-Packard Company will, at its option, either repair or replace products that prove to be defective.

For warranty service or repair, this product must be returned to a service facility designated by Hewlett-Packard. However, warranty service for products installed by Hewlett-Packard and certain other products designated by Hewlett-Packard will be performed at the Buyer's facility at no charge within the Hewlett-Packard service travel area. Outside Hewlett-Packard service travel areas, warranty service will be performed at the Buyer's facility only upon Hewlett-Packard's prior agreement and the Buyer shall pay Hewlett-Packard's round trip travel expenses.

For products returned to Hewlett-Packard for warranty service, the Buyer shall prepay shipping charges to Hewlett-Packard and Hewlett-Packard shall pay shipping charges to return the product to the Buyer. However, the Buyer shall pay all shipping charges, duties, and taxes for products returned to Hewlett-Packard from another country.

Hewlett-Packard warrants that its software and firmware designated by Hewlett-Packard for use with an instrument will execute its programming instructions when properly installed on that instrument. Hewlett-Packard does not warrant that the operation of the instrument software, or firmware will be uninterrupted or error free.

Limitation of Warranty

The foregoing warranty shall not apply to defects resulting from improper or inadequate maintenance by the Buyer, Buyer-supplied software or interfacing, unauthorized modification or misuse, operation outside of the environmental specifications for the product, or improper site preparation or maintenance.

NO OTHER WARRANTY IS EXPRESSED OR IMPLIED. HEWLETT-PACKARD SPECIFICALLY DISCLAIMS THE IMPLIED WARRANTIES OR MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

Exclusive Remedies THE REMEDIES PROVIDED HEREIN ARE THE BUYER'S SOLE AND EXCLUSIVE REMEDIES. HEWLETT-PACKARD SHALL NOT BE LIABLE FOR ANY DIRECT, INDIRECT, SPECIAL INCIDENTAL, OR CONSEQUENTIAL DAMAGES, WHETHER BASED ON CONTRACT, TORT, OR ANY OTHER LEGAL THEORY.

Assistance Product maintenance agreements and other customer assistance agreements are available for Hewlett-Packard products.

For any assistance, contact your nearest Hewlett-Packard Sales and Service Office.

Certification Hewlett-Packard Company certifies that this product met its published specifications at the time of shipment from the factory. Hewlett-Packard further certifies that its calibration measurements are traceable to the United States National Bureau of Standards, to the extent allowed by the Bureau's calibration facility, and to the calibration facilities of other International Standards Organization members.

Safety This product has been designed and tested according to International Safety Requirements. To ensure safe operation and to keep the product safe, the information, cautions, and warnings in this manual must be heeded.

Table of Contents

Chapter 2:

Chapter 3:

Chapter 4:

Chapter 5:

Chapter 6:

Mainframe Commands

Chapter 7:

SYSTem Subsystem

Chapter 8:

Chapter 9:

Appendix A:

Message Communication and System Functions

Appendix B:

Status Reporting

Appendix C:

Index

Introduction to Programming an Instrument 1

Introduction

This chapter introduces you to the basic concepts of bus communication and provides information and examples to get you started programming. The exact mnemonics for the commands are listed in chapters 5 through 9 of this manual and in the individual programming manuals for each module. There are three basic operations that can be done with a controller and this instrument via the bus. You can:

1. Set up the instrument and start measurements
2. Retrieve setup information and measurement results
3. Send measurement data to the instrument

Other more complicated tasks are accomplished with a combination of these basic functions.

Chapter 1 deals mainly with how to set up the instrument, how to retrieve setup information and measurement results, and how to pass data to the controller. This chapter is divided into two sections. The first section (page 1-2) concentrates on program syntax, and the second section (page 1-12) discusses programming an instrument.

Note

The programming examples in this manual are written in HP Basic 4.0 using an HP 9000 Series 200/300 Controller over HP-IB and RS-232C.

Programming Syntax

Talking to the Instrument

In general, computers acting as controllers communicate with the instrument by passing messages over a remote interface using the I/O statements provided in the instruction set of the controller's host language. Hence, the HPDIAL messages for programming the HP 16500A, described in this manual, will normally appear as ASCII character strings imbedded inside the I/O statements of your controller's program. For example, the HP 9000 Series 200/300 BASIC and PASCAL language systems use the OUTPUT statement for sending program messages to the HP 16500A, and the ENTER statement for receiving response messages from the HP 16500A.

Messages are placed on the bus using an output command and passing the device address, program message, and terminator. Passing the device address ensures that the program message is sent to the correct interface and instrument.

The following command turns the command headers on:

```
OUTPUT < device address > ;":SYSTEM:HEADER ON" < terminator >
```

< device address > represents the address of the device being programmed.

Note

The actual OUTPUT command you use when programming is dependent on the controller you are using, the programming language you are using, and which interface you are programming over (HP-IB or RS-232C).

Angular brackets "< >," in this manual, enclose words or characters that symbolize a program code parameter or a bus command.

Information that is displayed in quotes represents the actual message that is sent across the bus. The message terminator (NL or EOI) is the only additional information that is also sent across the bus.

For HP 9000 Series 200/300 controllers, it is not necessary to type in the actual <terminator> at the end of the program message. These controllers automatically terminate the program message internally when the return key is pressed.

Addressing the Instrument for HP-IB

Since HP-IB can address multiple devices through the same interface card, the device address passed with the program message must include not only the correct interface code, but also the correct instrument address.

Interface Select Code (Selects Interface). Each interface card has a unique interface select code. This code is used by the controller to direct commands and communications to the proper interface. The default is typically "7" for HP-IB controllers.

Instrument Address (Selects Instrument). Each instrument on an HP-IB bus must have a unique instrument address between decimal 0 and 30. The device address passed with the program message must include not only the correct instrument address, but also the correct interface select code.

DEVICE ADDRESS = (Interface Select Code X 100) + (Instrument Address)

For example, if the instrument address for the HP 16500A is 4 and the interface select code is 7, when the program message is passed, the routine performs its function on the instrument at device address 704.

For the HP 16500A, the instrument address is typically set to "7" at the factory. This address can be changed in the HP-IB pop-up menu of the System Configuration menu.

Addressing the Instrument for RS-232C

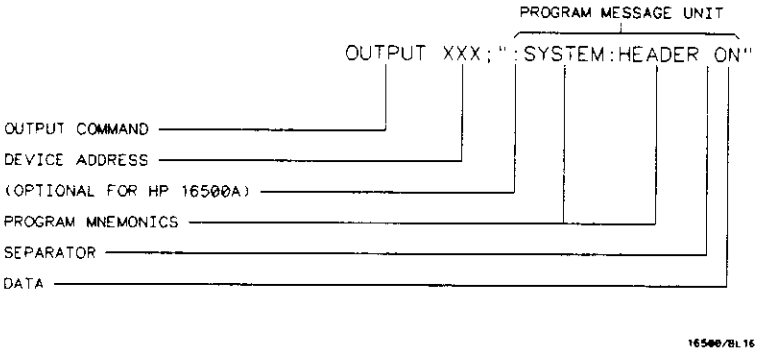
Since RS-232C can only be connected between two devices through the same interface card, only the correct interface code is required for the device address.

Interface Select Code (Selects Interface). Each interface card has its own interface select code. This address is used by the controller to direct commands and communications to the proper interface. Generally, the interface select code can be any decimal value between 0 and 31. This value can be selected through switches on the RS-232C interface card in the controller.

For example, if the interface select code is 20, the device address required to communicate over the bus is 20.

Program Message Syntax

To program the instrument over the bus, you must have an understanding of the command format and structure expected by the instrument. The instrument is remotely programmed with program messages. These are composed of sequences of program message units, with each unit representing a program command or query. A program command or query is composed of a sequence of functional elements that include separators, headers, program data, and terminators. These are sent to the instrument over the system interface as a sequence of ASCII data messages. For example:



Separator

The < separator > shown in the program message refers to a blank space which is required to separate the program mnemonic from the program data.

Command Syntax

A command is composed of a header, any associated data, and a terminator. The header is the mnemonic or mnemonics that represent the operation to be performed by the instrument. The different types of headers are discussed in the following paragraphs.

Simple Command Header. Simple command headers contain a single mnemonic. START and STOP are examples of simple command headers typically used in this instrument. The syntax is:

< program mnemonic > < terminator >

When program data must be included with the simple command header (for example, :SELECT 1), a separator is added. The syntax is:

<program mnemonic> <separator> <program data> <terminator>

Compound Command Header. Compound command headers are a combination of two or more program mnemonics. The first mnemonic selects the subsystem, and the last mnemonic selects the function within that subsystem. Additional mnemonics appear between the subsystem mnemonic and the function mnemonic when there are additional levels within the subsystem that must be transversed. The mnemonics within the compound message are separated by colons. For example:

To execute a single function within a subsystem, use the following:

:<subsystem>:<function> <separator> <program data> <terminator>

(For example :SYSTEM:LONGFORM ON)

To transverse down a level of a subsystem to execute a subsystem within that subsystem:

:<subsystem>:<subsystem>:<function> <separator> <program data> <terminator>

(For example :MMEMORY:LOAD:CONFIG "FILE__")

To execute more than one function within the same subsystem a semi-colon is used to separate the functions:

:<subsystem>:<function> <separator> <data>;<function> <separator> <data> <terminator>

(For example :SYSTEM:LONGFORM ON;HEADER ON)

Identical function mnemonics can be used for more than one subsystem. For example, in the oscilloscope module the function mnemonic RANGE may be used to change the vertical range or to change the horizontal range:

```
:CHANNEL1:RANGE .4
```

- sets the vertical range of channel 1 to 0.4 volts full scale.

```
:TIMEBASE:RANGE 1
```

- sets the horizontal timebase to 1 second full scale.

CHANNEL1 and TIMEBASE are subsystem selectors and determine which range is being modified.

Common Command Header. Common command headers control IEEE 488.2 functions within the instrument (such as clear status, etc.). Their syntax is:

```
*<command header><terminator>
```

No space or separator is allowed between the asterisk and the command header. *CLS is an example of a common command header.

Query Command Command headers immediately followed by a question mark (?) are queries. After receiving a query, the instrument interrogates the requested function and places the answer in its output queue. The output message remains in the queue until it is read or another command is issued. When read, the message is transmitted across the bus to the designated listener (typically a controller). The logic analyzer query :MACHINE1:TWAVEFORM:RANGE? places the current seconds per division full scale range for machine 1 in the output queue. The controller input statement:

```
ENTER < device address > ;Range
```

passes the value across the bus to the controller and places it in the variable Range.

Query commands are used to find out how the instrument is currently configured. They are also used to get results of measurements made by the instrument, with the query actually activating the measurement. For example, the oscilloscope command :MEASURE:RISETIME? instructs the instrument to measure the risetime of your waveform and place the result in the output queue.

Note

The output queue must be read before the next program message is sent. For example, when you send the oscilloscope query :MEASURE:RISETIME? you must follow that query with the program statement ENTER Value_risetime to read the result of the query and place the result in a variable (Value_risetime).

Sending another command before reading the result of the query will cause the output buffer to be cleared and the current response to be lost. This will also generate an error in the error queue.

Program Header Options

Program headers can be sent using any combination of uppercase or lowercase ASCII characters. Instrument responses, however, are always returned in uppercase.

Both program command and query headers may be sent in either longform (complete spelling), shortform (abbreviated spelling), or any combination of longform and shortform. Either of the following examples turn the headers and longform on.

```
:SYSTEM:HEADER ON;LONGFORM ON - longform
```

```
:SYST:HEAD ON;LONG ON - shortform
```

Programs written in longform are easily read and are almost self-documenting. The shortform syntax conserves the amount of controller memory needed for program storage and reduces the amount of I/O activity.

Note

The rules for shortform syntax are shown in the chapter "Programming and Documentation Conventions."

Program Data

Program data is used to convey a variety of types of parameter information related to the command header. At least one space must separate the command header or query header from the program data.

```
<program mnemonic> <separator> <data> <terminator>
```

When a program mnemonic or query has multiple data parameters a comma separates sequential program data.

```
<program mnemonic> <separator> <data>,<data> <terminator>
```

For example, :MENU 0,2 has two data parameters: 0 and 2.

Character Program Data. Character program data is used to convey parameter information as short alpha or alphanumeric strings. For example, the run mode command RMODE can be set to single or repetitive. The character program data in this case may be SINGLE or REPETITIVE. :RMODE SINGLE sets the run mode to single.

Numeric Program Data. Some command headers require program data to be a number. For example, :MENU requires the desired menu selection to be expressed numerically. The instrument recognizes integers, real numbers, and scientific notation. With the proper prefix, the instrument will also recognize binary, octal, and hexadecimal base numbers. If no prefix is added, the default is decimal.

Table 1-1. Numeric Data Prefixes

Base	Prefix	Example
Binary	#B	#B10101010
Octal	#Q	#Q1234567
Hexadecimal	#H	#H2468ABC
Decimal	(none)	1234567

Program Message Terminator

The program codes within a data message are executed after the program message terminator is received. The terminator may be either an NL (New Line) character, an EOI (End-Or-Identify) asserted, or a combination of the two. All three ways are equivalent with the exact encodings for the program terminators listed in the appendix "Message Communication and System Functions." Asserting the EOI sets the EOI control line low on the last byte of the data message. The NL character is an ASCII linefeed (decimal 10).

Note

The NL (New Line) terminator has the same function as an EOS (End Of String) and EOT (End Of Text) terminator.

The EOI terminator only applies to HP-IB.

Selecting Multiple Subsystems

You can send multiple program commands and program queries for different subsystems on the same line by separating each command with a semicolon. The colon following the semicolon enables you to enter a new subsystem. For example:

<program mnemonic> <data>;:<program mnemonic> <data> <terminator>

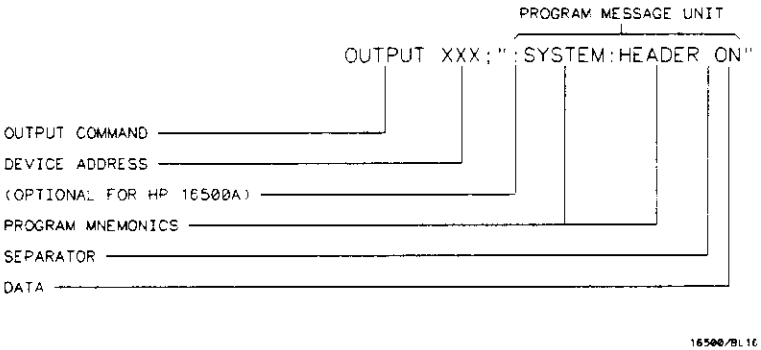
```
:MMEMORY:CATALOG?;:SYSTEM:PRINT ALL
```

Note

Multiple commands may be any combination of compound and simple commands.

Summary

The following illustration summarizes the syntax for programming over the bus.



Programming an Instrument

Initialization

To make sure the bus and all appropriate interfaces are in a known state, begin every program with an initialization statement. For example:

CLEAR XXX ! initializes the interface of the instrument.

Then load a predefined configuration file from the disc to preset the instrument to a known state. For example:

OUTPUT XXX;":MMEMORY:LOAD:CONFIG 'DEFAULT__'"

would load the configuration file "DEFAULT__" into all of the modules and mainframe. Refer to the chapter "Mmemory Subsystem" for more information on the LOAD command.

Note

The three Xs (XXX) after the "CLEAR" and "OUTPUT" statements in the previous examples refer to the device address required for programming over either HP-IB or RS-232C. The commands and syntax for initializing the instrument are discussed in the chapter "Common Commands."

Refer to your controller manual and programming language reference manual for information on initializing the interface.

Selecting an HP 16500A Module

Before you can program an HP 16500A module over the bus, you must first select the appropriate module. To select the module, use the mainframe command :SELECT followed by the numeric reference for the slot location of the master card for that module (1...5 refers to slot A...E respectively). :SELECT 1 selects the module in slot A, :SELECT 2 selects the module in slot B, etc. For example, if the TIMEBASE card for your oscilloscope module is in slot B, then the command:

```
:SELECT 2
```

would select the oscilloscope module. Figure 1-1 shows the command tree for the select command. For more information on the select command, refer to the chapter "Mainframe Commands."

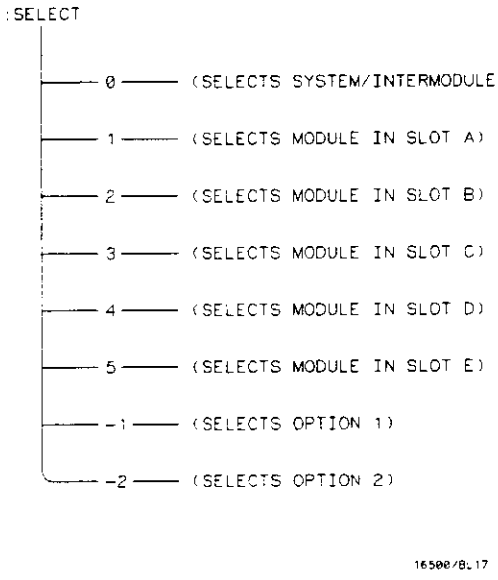


Figure 1-1. Select Command Tree

Note

Commands may be sent over the bus for any module while in any menu as long as the appropriate module has been selected.

Example Program This program demonstrates the basic command structure used to program a module of the HP 16500A.

```
10 CLEAR XXX                        !Initialize instrument interface
20 OUTPUT XXX;":SYSTEM:HEADER ON"      !Turn headers on
30 OUTPUT XXX;":SYSTEM:LONGFORM ON"  !Turn longform on
40 DIM Card$[100]                   !Reserve memory for string variable
50 OUTPUT XXX;":CARDCAGE?"     !Verify which modules are loaded
60 ENTER XXX;Card$                  !Enter result in a string variable
70 PRINT Card$                      !Print result of query
80 OUTPUT XXX;":MMEM:LOAD:CONFIG 'TEST_E',5"    !Load configuration file
                                                    !into module in slot E
90 OUTPUT XXX;":SELECT 5"           !Select module in slot E
100 OUTPUT XXX;":MENU 5,3"          !Select menu for module in slot E
110 OUTPUT XXX;":RMODE SINGLE"      !Select run mode
120 OUTPUT XXX;":START"             !Run the measurement
```

Note

The three Xs (XXX) after the "OUTPUT" and "ENTER" statements in the previous examples refer to the device address required for programming over either HP-IB or RS-232C.

Program Overview

Line 10 initializes the instrument interface to a known state, and lines 20 and 30 turn the headers and longform on.

Line 50 queries the cardcage to confirm which modules are loaded in this mainframe. Then line 70 prints the result of the query on screen. In this example the query returns:

```
:CARDCAGE -1,-1,12,11,31,0,0,4,4,5
```

The first five numbers returned are the card identification numbers. The last five numbers list the module assignment for each card. In this example, the response shows that there is an oscilloscope acquisition card in slot C which is assigned to the oscilloscope timebase card in slot D. Also, there is a logic analyzer card in slot E which is assigned to itself. Refer to the chapter "Mainframe Commands" for more information on CARDCAGE command.

Line 80 loads the configuration file "TEST_E" into the module in slot E and line 90 selects the module in slot E. Then line 100 displays one of menus of the module in slot E.

Lines 110 and 120 tell the analyzer to run the measurement configured by the file "TEST_E" one time.

Receiving Information from the Instrument

After receiving a query (command header followed by a question mark), the instrument interrogates the requested function and places the answer in its output queue. The answer remains in the output queue until it is read or another command is issued. When read, the message is transmitted across the bus to the designated listener (typically a controller). The input statement for receiving a response message from an instrument's output queue typically has two parameters;the device address and a format specification for handling the response message. For example, to read the result of the query command :SYSTEM:LONGFORM? you would execute the statement:

```
ENTER <device address> ;Setting
```

where <device address> represents the address of your device. This would enter the current setting for the longform command in the numeric variable Setting.

Note

All results for queries sent in a program message must be read before another program message is sent. For example, when you send the query :SELECT?, you must follow that query with the program statement ENTER Setting to read the result of the query and place the result in a variable (Setting).

Sending another command before reading the result of the query will cause the output buffer to be cleared and the current response to be lost. This will also cause an error to be placed in the error queue.

The actual ENTER program statement you use when programming is dependent on the programming language you are using.

The format specification for handling the response messages is dependent on both the controller and the programming language.

Response Header Options

The format of the returned ASCII string depends on the current settings of the SYSTEM HEADER and LONGFORM commands. The general format is:

<header> <separator> <data> <terminator>

The header identifies the data that follows and is controlled by issuing a :SYSTEM:HEADER ON/OFF command. If the state of the header command is OFF, only the data is returned by the query. The format of the header is controlled by the :SYSTEM:LONGFORM ON/OFF command. If longform is OFF , the header will be in its shortform and the header will vary in length depending on the particular query. The following would be returned from a :SELECT? command query:

<data> <terminator> (with HEADER OFF)

:SEL <separator> <data> <terminator> (with HEADER ON/LONGFORM OFF)

:SELECT <separator> <data> <terminator> (with HEADER ON/LONGFORM ON)

Note

A command or query may be sent in either longform or shortform, or in any combination of longform and shortform. The HEADER and LONGFORM commands only control the format of the returned data and have no effect on the way commands are sent. Common commands never return a header.

Refer to the chapter "System Subsystem" for information on turning the HEADER and LONGFORM commands on and off.

Response Data Formats

Most data will be returned as exponential or integer numbers. However, query data of instrument setups may be returned as character data. Interrogating the run mode :RMODE? will return one of the following:

:RMODE REPETITIVE<terminator> (with HEADER ON/LONGFORM ON)

:RMOD REP<terminator> (with HEADER ON/LONGFORM OFF)

REPETITIVE<terminator> (with HEADER OFF/LONGFORM ON)

REP<terminator> (with HEADER OFF/LONGFORM OFF)

Note

Refer to the individual commands in this manual and in the individual programming manuals for each module for information on the format (alpha or numeric) of the data returned from each query.

Numeric Base

Most numeric data will be returned in the same base as shown on screen. When the prefix #B precedes the returned data, the value is in the binary base. Likewise, #Q is the octal base and #H is the hexadecimal base. If no prefix precedes the returned numeric data, then the value is in the decimal base.

String Variables

If you want to observe the headers for queries, you must bring the returned data into a string variable. Reading queries into string variables is simple and straightforward, requiring little attention to formatting. For example:

```
ENTER <device address>;Result$
```

places the output of the query in the string variable Result$.

Note

String variables are case sensitive and must be expressed exactly the same each time they are used.

The output of the instrument may be numeric or character data depending on what is queried. Refer to the specific commands for the formats and types of data returned from queries.

Note

For the example programs, the device being programmed is at device address XXX. The actual address will vary according to how you have configured the bus for your own application and whether you are programming over HP-IB or RS-232C.

The following example shows logic analyzer data being returned to a string variable with headers off:

```
10 OUTPUT XXX;":SYSTEM:HEADER OFF"
20 DIM Rang$[30]
30 OUTPUT XXX;":MACHINE1:TWAVEFORM:RANGE?"
40 ENTER XXX;Rang$
50 PRINT Rang$
60 END
```

After running this program, the controller displays:

+1.00000E-05

Numeric Variables

If you do not need to see the headers when a numeric value is returned from the instrument, then you can use a numeric variable. When you are receiving numeric data into a numeric variable, turn the headers off. Otherwise the headers may cause misinterpretation of returned data.

The following example shows logic analyzer data being returned to a numeric variable.

```
10 OUTPUT XXX;":SYSTEM:HEADER OFF"
20 OUTPUT XXX;":MACHINE1:TWAVEFORM:RANGE?"
30 ENTER XXX;Rang
40 PRINT Rang
50 END
```

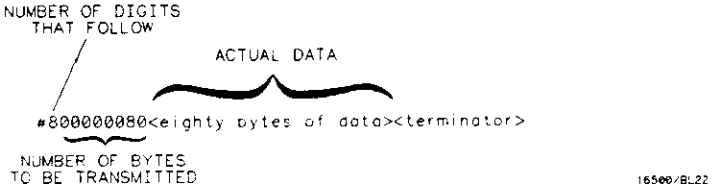
After running this program, the controller displays:

1.E-5

Definite-Length Block Response Data

Definite-length block response data allows any type of device-dependent data to be transmitted over the system interface as a series of 8-bit binary data bytes. This is particularly useful for sending large quantities of data or 8-bit extended ASCII codes. The syntax is a pound sign (#) followed by a non-zero digit representing the number of digits in the decimal integer. After the non-zero digit is the decimal integer that states the number of 8-bit data bytes being sent. This is followed by the actual data.

For example, for transmitting 80 bytes of data, the syntax would be:



The "8" states the number of digits that follow, and "00000080" states the number of bytes to be transmitted.

Note

Indefinite-length block data is not supported on the HP16500A.

Multiple Queries

You can send multiple queries to the instrument within a single program message, but you must also read them back within a single program message. This can be accomplished by either reading them back into a string variable or into multiple numeric variables. For example, you could read the result of the query :SYSTEM:HEADER?;LONGFORM? into the string variable Results$ with the command:

```
ENTER XXX;Results$
```

When you read the result of multiple queries into string variables, each response is separated by a semicolon. For example, the response of the query :SYSTEM:HEADER?:LONGFORM? with HEADER and LONGFORM on would be:

```
:SYSTEM:HEADER 1;:SYSTEM:LONGFORM 1
```

If you do not need to see the headers when the numeric values are returned, then you could use following program message to read the query :SYSTEM:HEADERS?;LONGFORM? into multiple numeric variables:

```
ENTER XXX;Result1,Result2
```

Note

When you are receiving numeric data into numeric variables, the headers should be turned off. Otherwise the headers may cause misinterpretation of returned data.

Instrument Status

Status registers track the current status of the instrument. By checking the instrument status, you can find out whether an operation has been completed, whether the instrument is receiving triggers, and more. The appendix "Status Reporting" explains how to check the status of the instrument.

Programming Over HP-IB 2

Introduction

This section describes the interface functions and some general concepts of the HP-IB. In general, these functions are defined by IEEE 488.1 (HP-IB bus standard). They deal with general bus management issues, as well as messages which can be sent over the bus as bus commands.

Interface Capabilities

The interface capabilities of the HP 16500A, as defined by IEEE 488.1 are SH1, AH1, T5, TE0, L3, LE0, SR1, RL1, PP1, DC1, DT1, C0, and E2. For more information, refer to table 6-1 in the chapter "Mainframe Commands."

Command and Data Concepts

The HP-IB has two modes of operation: command mode and data mode. The bus is in command mode when the ATN line is true. The command mode is used to send talk and listen addresses and various bus commands, such as a group execute trigger (GET). The bus is in the data mode when the ATN line is false. The data mode is used to convey device-dependent messages across the bus. These device-dependent messages include all of the instrument commands and responses found in chapters 5 through 9 of this manual and in the individual programming manuals for each module.

Addressing

By using the front-panel touchscreen, the HP-IB interface can be placed in either talk only mode (Printer) or addressed talk/listen mode (Controller) (see the *HP 16500A Reference Manual*). Talk only mode should be used when you want the instrument to talk directly to a printer without the aid of a controller. Addressed talk/listen mode is used when the instrument will operate in conjunction with a controller. When the instrument is in the addressed talk/listen mode, the following is true:

- Each device on the HP-IB resides at a particular address ranging from 0 to 30.
- The active controller specifies which devices will talk, and which will listen.
- An instrument, therefore, may be talk addressed, listen addressed, or unaddressed by the controller.

If the controller addresses the instrument to talk, it will remain configured to talk until it receives an interface clear message (IFC), another instrument's talk address (OTA), its own listen address (MLA), or a universal untalk command (UNT).

If the controller addresses the instrument to listen, it will remain configured to listen until it receives an interface clear message (IFC) its own talk address (MTA), or a universal unlisten command (UNL).

Communicating Over the HP-IB Bus (HP 9000 Series 200/300 Controller)

Since HP-IB can address multiple devices through the same interface card, the device address passed with the program message must include not only the correct instrument address, but also the correct interface code.

Interface Select Code (Selects Interface). Each interface card has its own interface select code. This code is used by the controller to direct commands and communications to the proper interface. The default is typically "7" for HP-IB controllers.

Instrument Address (Selects Instrument). Each instrument on the HP-IB port must have a unique instrument address between decimal 0 and 30. The device address passed with the program message must include not only the correct instrument address, but also the correct interface select code.

DEVICE ADDRESS = (Interface Select Code X 100) + (Instrument Address)

For example, if the instrument address for the HP 16500A is 4 and the interface select code is 7, when the program message is passed, the routine performs its function on the instrument at device address 704.

Local, Remote, and Local Lockout

The local, remote, and remote with local lockout modes may be used for various degrees of front-panel control while a program is running. The instrument will accept and execute bus commands while in local mode, and the front panel will also be entirely active. If the HP 16500A is in remote mode, the instrument will go from remote to local with any touchscreen or mouse activity. In remote with local lockout mode, all controls (except the power switch) are entirely locked out. Local control can only be restored by the controller.

Note

Cycling the power will also restore local control, but this will also reset certain HP-IB states.

The instrument is placed in remote mode by setting the REN (Remote Enable) bus control line true, and then addressing the instrument to listen. The instrument can be placed in local lockout mode by sending the local lockout command (LLO). The instrument can be returned to local mode by either setting the REN line false, or sending the instrument the go to local command (GTL).

Bus Commands

The following commands are IEEE 488.1 bus commands (ATN true). IEEE 488.2 defines many of the actions which are taken when these commands are received by an instrument.

Device Clear

The device clear (DCL) or selected device clear (SDC) commands clear the input and output buffers, reset the parser, clear any pending commands, and clear the Request-OPC flag.

Group Execute Trigger (GET)

The group execute trigger command will cause the same action as the START command for Group Run: the instrument will acquire data for the active waveform and listing display(s).

Interface Clear (IFC)

This command halts all bus activity. This includes unaddressing all listeners and the talker, disabling serial poll on all devices, and returning control to the system controller.

Programming Over RS-232C 3

Introduction

This section describes the interface functions and some general concepts of the RS-232C. The RS-232C interface on this instrument is Hewlett-Packard's implementation of EIA Recommended Standard RS-232C, "Interface Between Data Terminal Equipment and Data Communications Equipment Employing Serial Binary Data Interchange." With this interface, data is sent one bit at a time and characters are not synchronized with preceeding or subsequent data characters. Each character is sent as a complete entity without relationship to other events.

Note

IEEE 488.2 is designed to work with IEEE 488.1 as the physical interface. When RS-232C is used as the physical interface, as much of IEEE 488.2 is retained as the hardware differences will allow. No IEEE 488.1 messages such as DCL, GET, and END are available.

Interface Operation

The HP 16500A can be programmed with a controller over RS-232C using either a minimum three-wire or extended hardwire interface. The operation and exact connections for these interfaces are described in more detail in the following sections. When you are programming an HP 16500A over RS-232C with a controller, you are normally operating directly between two DTE (Data Terminal Equipment) devices as compared to operating between a DTE device and a DCE (Data Communications Equipment) device. When operating directly between two RS-232C devices, certain considerations must be taken into account. For three-wire operation, XON/XOFF must be used to handle protocol between the devices. For extended hardwire operation, protocol may be handled either with XON/XOFF or by manipulating the CTS and RTS lines of the HP 16500A. For both three-wire and extended hardwire operation, the DCD and DSR inputs to the HP 16500A must remain high for proper operation. With extended hardwire operation, a high on the CTS input allows the HP 16500A to send data and a low on this line disables the HP 16500A data transmission. Likewise, a high on the RTS line allows the controller to send data and a low on this line signals a request for the controller to disable data transmission. Since three-wire operation has no control over the CTS input, internal pull-up resistors in the HP 16500A assure that this line remains high for proper three-wire operation.

Cables

Selecting a cable for the RS-232C interface is dependent on your specific application. The following paragraphs describe which lines of the HP 16500A are used to control the operation of the RS-232C bus relative to the HP 16500A. To locate the proper cable for your application, refer to the reference manual for your controller. This manual should address the exact method your controller uses to operate over the RS-232C bus.

Minimum Three-Wire Interface with Software Protocol

With a three-wire interface, the SOFTWARE (as compared to interface hardware) controls the data flow between the HP 16500A and the controller. This provides a much simpler connection between devices since you can ignore hardware handshake requirements. The HP 16500A uses the following connections on its RS-232C interface for three-wire communication:

- Pin 7 SGND (Signal Ground)
- Pin 2 TD (Transmit Data from HP 16500A)
- Pin 3 RD (Receive Data into HP 16500A)

The TD (Transmit Data) line from the HP 16500A must connect to the RD (Receive Data) line on the controller. Likewise, the RD line from the HP 16500A must connect to the TD line on the controller. Internal pull-up resistors in the HP 16500A assure the DCD, DSR, and CTS lines remain high when you are using a three-wire interface.

Note

The three-wire interface provides no hardware means to control data flow between the controller and the HP 16500A. XON/OFF protocol is the only means to control this data flow.

Extended Interface with Hardware Handshake

With the extended interface , both the software and the hardware can control the data flow between the HP 16500A and the controller. This allows you to have more control of data flow between devices. The HP 16500A uses the following connections on its RS-232C interface for extended interface communication:

- Pin 7 SGND (Signal Ground)
- Pin 2 TD (Transmit Data from HP 16500A)
- Pin 3 RD (Receive Data into HP 16500A)

The additional lines you use depends on your controller's implementation of the extended hardwire interface.

- Pin 4 RTS (Request To Send) is an output from the HP 16500A which can be used to control incoming data flow.
- Pin 5 CTS (Clear To Send) is an input to the HP 16500A which controls data flow from the HP 16500A.
- Pin 6 DSR (Data Set Ready) is an input to the HP 16500A which controls data flow from the HP 16500A within two bytes.
- Pin 8 DCD (Data Carrier Detect) is an input to the HP 16500A which controls data flow from the HP 16500A within two bytes.
- Pin 20 DTR (Data Terminal Ready) is an output from the HP 16500A which is enabled as long as the HP 16500A is turned on.

The TD (Transmit Data) line from the HP 16500A must connect to the RD (Receive Data) line on the controller. Likewise, the RD line from the HP 16500A must connect to the TD line on the controller.

The RTS (Request To Send), is an output from the HP 16500A which can be used to control incoming data flow. A high on the RTS line allows the controller to send data and a low on this line signals a request for the controller to disable data transmission.

The CTS (Clear To Send), DSR (Data Set Ready), and DCD (Data Carrier Detect) lines are inputs to the HP 16500A which control data flow from the HP 16500A (Pin 2). Internal pull-up resistors in the HP 16500A assure the DCD and DSR lines remain high when they are not connected. If DCD or DSR are connected to the controller, the controller must keep these lines and the CTS line high to enable the HP 16500A to send data to the controller. A low on any one of these lines will disable the HP 16500A data transmission. Dropping the CTS line low during data transmission will stop HP 16500A data transmission immediately. Dropping either the DSR or DCD line low during data transmission will stop HP 16500A data transmission, but as many as two additional bytes may be transmitted from the HP 16500A.

Cable Example

Figure 3-1 is an example of how to connect the HP 16500A to the HP 98628 Interface card of an HP 9000 series 200/300 controller. For more information on cabling, refer to the reference manual for your specific controller.

Note

Since this example does not have the correct connections for hardware handshake, XON/XOFF protocol must be used when connecting the HP 16500A as shown in figure 3-1

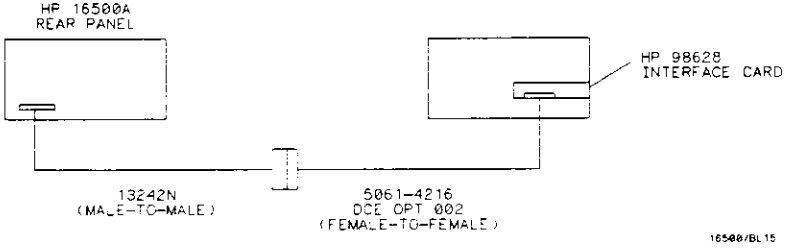


Figure 3-1. Cable Example

Configuring the Interface

By using the front-panel touchscreen, the RS-232C interface can be placed in either the printer mode or the controller mode. The printer mode should be used when you want the instrument to talk directly to a printer over RS-232C without the aid of a controller. The controller mode is used when the instrument will operate in conjunction with a controller over RS-232C.

If you are not familiar with how to configure the RS-232C interface, refer to the HP 16500A Reference Manual.

Interface Capabilities

The baud rate, stop bits, parity, protocol, and data bits must be configured exactly the same for both the controller and the HP 16500A to properly communicate over the RS-232C bus. The HP 16500A RS-232C interface capabilities are listed below:

- Baud Rate: 110, 300, 600, 1200, 2400, 4800, 9600, or 19.2 k
- Stop Bits: 1, 1.5, or 2
- Parity: None, Odd, or Even
- Protocol: None or XON/XOFF
- Data Bits: 7 or 8

Protocol

NONE. With a three-wire interface, selecting NONE for the protocol does not allow the sending or receiving device to control data flow. No control over the data flow increases the possibility of missing data or transferring incomplete data.

With an extended hardwire interface, selecting NONE allows a hardware handshake to occur. With hardware handshake, hardware signals control data flow.

XON/XOFF. XON/XOFF stands for Transmit On/Transmit Off. With this mode the receiver (controller or HP 16500A) controls data flow and can request that the sender (HP 16500A or controller) stop data flow. By sending XOFF (ASCII 17) over its transmit data line, the receiver requests that the sender disables data transmission. A subsequent XON (ASCII 19) allows the sending device to resume data transmission.

A controller sending data to the HP 16500A should send no more than 32 bytes of data after an XOFF.

The HP 16500A will not send any data after an XOFF is received until an XON is received.

Data Bits Data bits are the number of bits sent and received per character that represent the binary code of that character. They consist of either 7 or 8 bits, depending on the application.

8 Bit Mode. Information is usually stored in bytes (8 bits at a time). With 8-bit mode, you can send and receive data just as it is stored, without the need to convert the data.

7 Bit Mode. In 7-bit mode, each byte of data is converted into two separate 7-bit units. The first unit represents the ASCII equivalent of the four most significant bits of the byte and the second unit represents the ASCII equivalent of the four least significant bits of the byte. For example, to send the data

FE, A0, B1

over the bus in 7-bit mode, the instrument would send the ASCII equivalent of:

'F','E','A','0','B','1'

or

46,45,41,30,42,31 (hexadecimal).

Then the receiver would need to convert this 7-bit data back into their 8-bit equivalents.

Note

The controller and the HP 16500A must be in the same bit mode to properly communicate over the RS-232C. This means that both the controller and the HP 16500A must have the capability to send and receive 7 bit data, including the ability to convert and reassemble 7 bit data.

For more information on the RS-232C interface, refer to the HP 16500A Reference Manual. For information on RS-232C voltage levels and connector pinouts, refer to the HP 16500A Service Manual.

Communicating Over the RS-232C Bus (HP 9000 Series 200/300 Controller)

Each RS-232C interface card has its own interface select code. This code is used by the controller to direct commands and communications to the proper interface. Unlike HP-IB, which allows multiple devices to be connected through a single interface card, RS-232C is only connected between two devices at a time through the same interface card. Because of this, only the interface code is required for the device address.

Generally, the interface select code can be any decimal value between 0 and 31, except for those interface codes which are reserved by the controller for internal peripherals and other internal interfaces. This value can be selected through switches on the interface card. For more information, refer to the reference manual for your interface card or controller.

For example, if your RS-232C interface select code is 20, the device address required to communicate over the RS-232C bus is 20.

Lockout Command

To lockout the front panel controls use the system command LOCKOUT. When this function is on, all controls (except the power switch) are entirely locked out. Local control can only be restored by sending the command :LOCKOUT OFF. For more information on this command see the chapter "Mainframe Commands" in this manual.

Note

Cycling the power will also restore local control, but this will also reset certain RS-232C states.

Programming and Documentation Conventions 4

Introduction

This section covers the programming conventions used in programming the instrument, as well as the documentations conventions used in this manual. This chapter also contains a detailed description of the command tree and command tree traversal.

Truncation Rule

The truncation rule for the mnemonics used in headers and alpha arguments is:

- The mnemonic is the first four characters of the keyword unless the fourth character is a vowel, then the mnemonic is the first three characters of the keyword.

This rule will not be used if the length of the keyword is exactly four characters. When the keyword only contains four characters, there is no shortform of the command.

Some examples of how the truncation rule is applied to various commands are shown in table 4-1.

Table 4-1. Mnemonic Truncation

Longform	Shortform
START	STAR
CARDCAGE	CARD
MENU	MENU
SELECT	SEL
PATTERN	PATT

The Command Tree

The command tree (figure 4-1) shows all commands in the HP 16500A mainframe and the relationship of the commands to each other. You should notice that the common commands are not actually included with the command tree. After a <NL> (linefeed - ASCII decimal 10) has been sent to the instrument, the parser will be set to the "root" of the command tree.

Command Types

The commands for this instrument can be placed into three types. The three types are:

Common Commands. Common commands are independent of the tree, and do not affect the position of the parser within the tree.

Example: "*CLS"

Mainframe Commands. The mainframe commands reside at the root of the command tree. These commands are always parsable if they occur at the beginning of a program message, or are preceded by a colon.

Example: ":SELECT 1"

Subsystem Commands. Subsystem commands are grouped together under a common node of the tree, such as the MMEMORY commands.

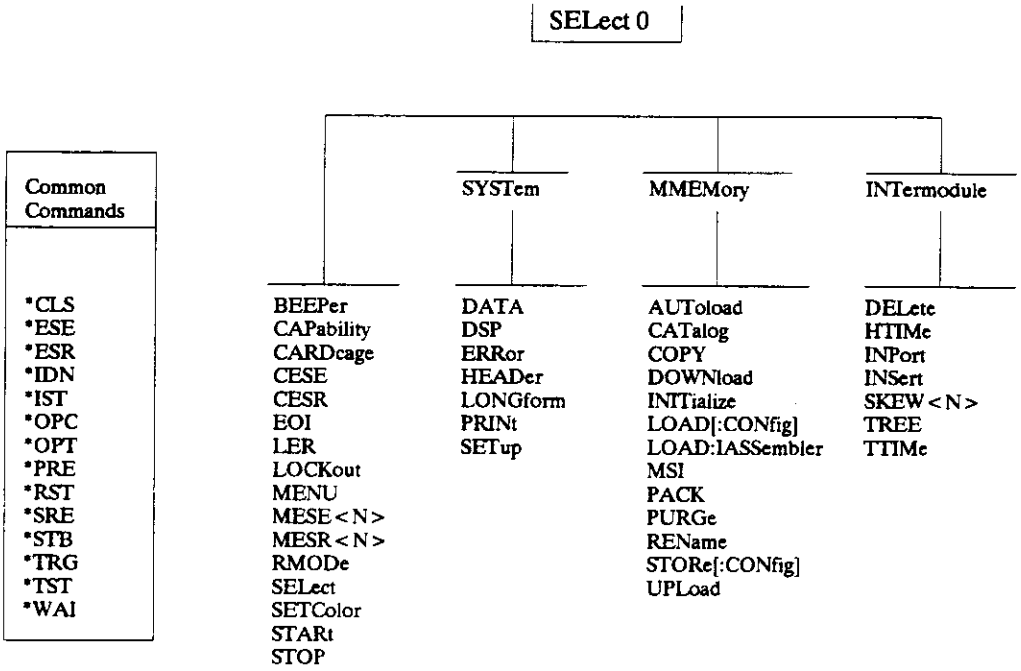


Figure 4-1. The HP 16500A Mainframe Command Tree

Tree Traversal Rules Command headers are created by traversing down the command tree. A legal command header from the command tree in figure 4-1 would be ":MMEM:INITIALIZE." This is referred to as a compound header. A compound header is a header made of two or more mnemonics separated by colons. The mnemonic created contains no spaces. The following rules apply to traversing the tree:

- A leading colon or a < program message terminator > (either a < NL > or EOI true on the last byte) places the parser at the root of the command tree. A leading colon is a colon that is the first character of a program header.

- Executing a subsystem command places you in that subsystem (until a leading colon or a < program message terminator > is found). In the Command Tree, figure 4-1, use the last mnemonic in the compound header as a reference point (for example INITIALIZE). Then find the last colon above that mnemonic (MMEM:), and that is where the parser will be. Any command below that point can be sent within the current program message without sending the mnemonic(s) which appear above them (STORE, etc.).

Examples The following examples are written using HP BASIC 4.0 on a HP 9000 Series 200/300 Controller. The quoted string is placed on the bus, followed by a carriage return and linefeed (CRLF).

The three Xs (XXX) shown in this manual after an ENTER or OUTPUT statement refers to the device address for either HP-IB or RS-232C.

Example 1 `OUTPUT XXX;":SYSTEM:HEADER ON;LONGFORM ON"`

In example 1, the colon between SYSTEM and HEADER is necessary since SYSTEM:HEADER is a compound command. The semicolon between the HEADER command and the LONGFORM command is the required < program message unit separator > . The LONGFORM command does not need SYSTEM preceding it, since the SYSTEM:HEADER command sets the parser to the SYSTEM node in the tree.

Example 2 OUTPUT XXX;":MMEMORY:INITIALIZE;STORE 'FILE__','FILE DESCRIPTION"

or

```
OUTPUT XXX;":MMEMORY:INITIALIZE"
OUTPUT XXX;":MMEMORY:STORE 'FILE__','FILE DESCRIPTION"
```

In the first line of example 2, the "subsystem selector" is implied for the STORE command in the compound command. The STORE command must be in the same program message as the INITIALIZE command, since the < program message terminator > will place the parser back at the root of the command tree.

A second way to send these commands is by placing "MMEMORY:" before the STORE command as shown in the fourth line of example 2.

Example 3 OUTPUT XXX;":MMEM:CATALOG?;:SYSTEM:PRINT ALL"

In example 3, the leading colon before SYSTEM tells the parser to go back to the root of the command tree. The parser can then see the SYSTEM:PRINT command.

Infinity Representation

The representation of infinity is 9.9E + 37 for real numbers and 32767 for integers. This is also the value returned when a measurement cannot be made.

Sequential and Overlapped Commands

IEEE 488.2 makes the distinction between sequential and overlapped commands. Sequential commands finish their task before the execution of the next command starts. Overlapped commands run concurrently, and therefore the command following an overlapped command may be started before the overlapped command is completed. Some examples of overlapped commands on the HP 16500A are:

START
STOP

Response Generation

IEEE 488.2 defines two times at which query responses may be buffered. The first is when the query is parsed by the instrument and the second is when the controller addresses the instrument to talk so that it may read the response. The HP 16500A will buffer responses to a query when it is parsed.

Notation Conventions and Definitions

The following conventions are used in this manual in descriptions of remote (HP-IB and RS-232C) operation:

< > Angular brackets enclose words or characters that are used to symbolize a program code parameter or a bus command.

::= "is defined as." For example, A ::= B indicates that A can be replaced by B in any statement containing A .

| "or": Indicates a choice of one element from a list. For example, A | B indicates A or B, but not both.

... An ellipsis (trailing dots) is used to indicate that the preceding element may be repeated one or more times.

[ ] Square brackets indicate that the enclosed items are optional.

{ } When several items are enclosed by braces, one, and only one of these elements must be selected.

XXX Three Xs after an ENTER or OUTPUT statement refer to the device address for HP-IB or RS-232C.

The following definitions are used:

d ::= A single ASCII numeric character, 0-9.

n ::= A single ASCII non-zero, numeric character, 1-9.

< NL > ::= Linefeed (ASCII decimal 10).

< sp > ::= < white space >

white space ::= 0 through 32 (decimal) except linefeed (decimal 10)

< msus > ::= < mass storage unit specifier > (INTernal1 specifies the front disc drive and INTernal0 specifies the rear disc drive)

Syntax Diagrams

At the beginning of each of the following chapters are syntax diagrams showing the proper syntax for each command. All characters contained in a circle or oblong are literals, and must be entered exactly as shown. Words and phrases contained in rectangles are names of items used with the command and are described in the accompanying text of each command. Each line can only be entered from one direction as indicated by the arrow on the entry line. Any combination of commands and arguments that can be generated by following the lines in the proper direction is syntactically correct. An argument is optional if there is a path around it. When there is a rectangle which contains the word "space," a white space character must be entered. White space is optional in many other places.

Command Structure

The HP 16500A programming commands are divided into three types: common commands, mainframe commands, and subsystem commands. A programming command tree is shown in figure 4-1 and a programming command cross-reference is shown in table 4-2.

Common Commands

The common commands are the commands defined by IEEE 488.2. These commands control some functions that are common to all IEEE 488.2 instruments. Sending the common commands do not take the instrument out of a selected subsystem.

Mainframe Commands

The mainframe commands control many of the basic functions of the instrument.

Subsystem Commands

There are several subsystems in this instrument. Only one subsystem may be selected at any given time. At power on, the command parser is set to the root of the command tree, and therefore, no subsystem is selected.

Note

When a <program message terminator> or a leading colon (:) is sent in a program message, the command parser is returned to the root of the command tree.

The 3 subsystems in the HP 16500A mainframe are:

- System - controls some basic functions of the instrument.
- Mmemory - provides access to both internal disc drives.
- Intermodule - allows intermodule arming between multiple modules.

Program Examples

The program examples given for each command in the following chapters and appendices were written on an HP 9000 Series 200/300 controller using HP BASIC 4.0 language. The programs always assume a generic address of XXX. If a printer is used, it is always assumed to be connected to the non-controller interface and activated by the :SYSTEM:PRINT command.

In these examples, special attention should be paid to the ways in which the command/query can be sent. The way the instrument is set up to respond to a command/query has no bearing on how you send the command/query. That is, the command/query can be sent using the longform or shortform if one exists for that command. You can send the command/query using upper case (capital) letters or lower case (small) letters; both work the same. Also, the data can be sent using almost any form you wish. If you were sending a channel 1 range value to the oscilloscope module of 100 mV, that value could be sent using a decimal (.1), or an exponential (1e-1 or 1.0E-1), or a suffix (100 mV or 100MV).

Note

The contents of a string is case sensitive and must be expressed exactly the same each time it is used.

As an example, set channel 1 range of the oscilloscope module to 100 mV by sending one of the following:

- commands in longform and using the decimal format.

```
OUTPUT XXX;":CHANNEL1:RANGE .1"
```

- commands in shortform and using an exponential format.

```
OUTPUT XXX;":CHAN1:RANG 1E-1"
```

- commands using lower case letters, shortforms, and a suffix.

```
OUTPUT XXX;":chan1:rang 100 mV"
```

Note

In these examples, the colon shown as the first character of the command is optional on the HP 16500A. The space between RANGE and the argument is required.

To observe the headers for queries, you must bring the returned data into a string variable. Generally, you should also dimension all string variables before reading the data.

If you do not need to see the headers and a numeric value is returned from the HP 16500A, then you should use a numeric variable. In this case the headers should be turned off.

Note

The contents of strings " " are case sensitive (label names, etc.).

Command Set Organization

The command set for the HP 16500A mainframe is divided into 5 separate groups: Common commands, mainframe commands and 3 sets of subsystem commands. Each of the 5 groups of commands is described in the following chapters. Each of the chapters contain a brief description of the subsystem, a set of syntax diagrams for those commands, and finally, the commands for that subsystem in alphabetical order. The commands are shown in the longform and shortform using upper and lowercase letters. As an example SELect indicates that the longform of the command is SELECT and the shortform of the command is SEL. Each of the commands contain a description of the command and its arguments, the command syntax, and a programming example.

Note

Each module within the HP 16500A will include additional sets of subsystem commands. For a list of these subsystem commands, refer to the individual programming manuals for each module.

Table 4-2. Alphabetic Command Cross-Reference

Command	Where Used	Command	Where Used
AUToload	MMEMory Subsystem	:MMEMory	Subsystem Selector
BEEPer	Mainframe Command	MSI	MMEMory Subsystem
CAPability?	Mainframe Command	*OPC	Common Command
CARDcage?	Mainframe Command	*OPT?	Common Command
CATalog?	MMEMory Subsystem	PACK	MMEMory Subsystem
*CLS	Common Command	*PRE	Common Command
COPY	MMEMory Subsystem	PRINt	SYSTem Subsystem
DEBug	Mainframe Command	PURGe	MMEMory Subsystem
DELete	INTermodule Subsystem	REName	MMEMory Subsystem
DOWNload	MMEMory Subsystem	RMODe	Mainframe Command
DSP	SYSTem Subsystem	*RST	Common Command
EOI	Mainframe Command	SELect	Mainframe Command
ERRor?	SYSTem Subsystem	SETColor	Mainframe Command
*ESE	Common Command	SETup	SYSTem Subsystem
*ESR?	Common Command	SKEW	INTermodule Subsystem
HEADer	SYSTem Subsystem	SOUNd	Mainframe Command
HTIMe?	INTermodule Subsystem	*SRE	Common Command
*IDN?	Common Command	STARt	Mainframe Command
INITialize	MMEMory Subsystem	*STB?	Common Command
INPort	INTermodule Subsystem	STOP	Mainframe Command
INSert	INTermodule Subsystem	STORe	MMEMory Subsystem
:INTermodule	Subsystem Selector	:SYSTem	Subsystem Selector
*IST?	Common Command	TREE	INTermodule Subsystem
LER?	Mainframe Command	*TRG	Common Command
LOAD	MMEMory Subsystem	*TST?	Common Command
LOCKout	Mainframe Command	TTIMe?	INTermodule Subsystem
LONGform	SYSTem Subsystem	UPLoad?	MMEMory Subsystem
MENU	Mainframe Command	*WAI	Common Command

Common Commands 5

Introduction

The common commands are defined by the IEEE 488.2 standard. These commands will be common to all instruments that comply with this standard.

The common commands control some of the basic instrument functions, such as instrument identification and reset, how status is read and cleared, and how commands and queries are received and processed by the instrument.

Common commands can be received and processed by the HP 16500A whether they are sent over the bus as separate program messages or within other program messages. If an instrument subsystem has been selected and a common command is received by the instrument, the instrument will remain in the selected sybsystem. For example if the program message

":MMEMORY:INITIALIZE;*CLS; STORE 'FILE__','DESCRIPTION'"

is received by the instrument, the instrument will initialize the disc and store the file; and clear the status information. This would not be the case if some other type of command were received within the program message. For example, the program message

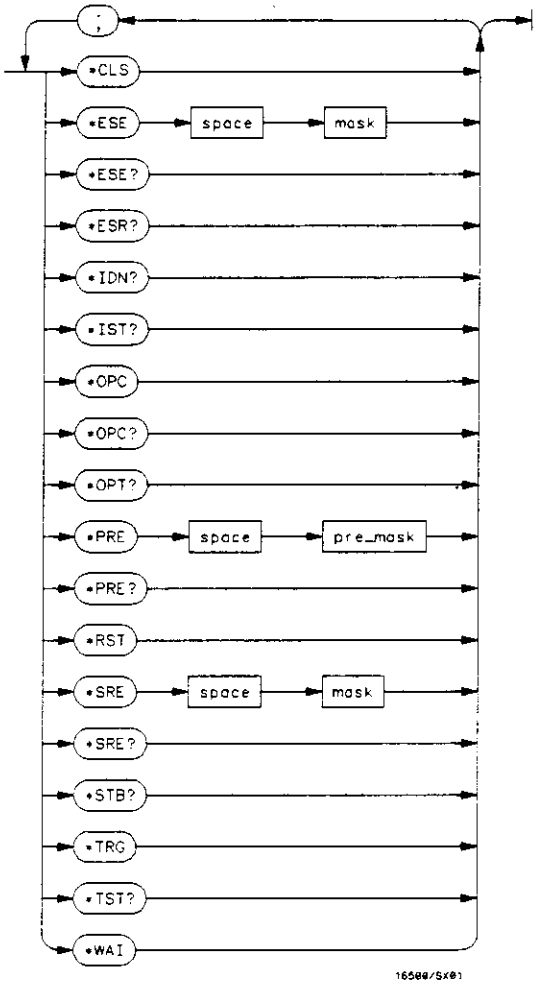
":MMEMORY:INITIALIZE;:SELECT 1;:MMEMORY:STORE 'FILE__','DESCRIPTION'"

would initialize the disc, select the module in slot A, then store the file. In this example :MMEMORY must be sent again in order to reenter the mmemory subsystem and store the file.

Each status register has an associated status enable (mask) register. By setting the bits in the mask value you can select the status information you wish to use. Any status bits that have not been masked (enabled in the enable register) will not be used to report status summary information to bits in other status registers.

Refer to appendix B for a complete discussion of how to read the status registers and how to use the status information available from this instrument.

Refer to figure 5-1 for the common commands syntax diagram.



mask = *An integer, 0 through 255. This number is the sum of all the bits in the mask corresponding to conditions that are enabled. Refer to the *ESE and *SRE commands for bit definitions in the enable registers.*

pre_mask = *An integer, 0 through 65535. This number is the sum of all bits in the mask corresponding to conditions that are enabled. Refer to the *PRE command for bit definitions in the enable register.*

Figure 5-1. Common Commands Syntax Diagram

*CLS (Clear Status) command

The *CLS common command clears all event status registers, queues, and data structures, including the device defined error queue and status byte. If the *CLS command immediately follows a < program message terminator >, the output queue and the MAV (Message Available) bit will be cleared.

Command Syntax: *CLS

Example: OUTPUT XXX;"*CLS"

Note

Refer to Appendix B for a complete discussion of status.

*ESE (Event Status Enable) command/query

The *ESE command sets the Standard Event Status Enable Register bits. The Standard Event Status Enable Register contains a mask value for the bits to be enabled in the Standard Event Status Register. A one in the Standard Event Status Enable Register will enable the corresponding bit in the Standard Event Status Register which sets the ESB bit in the status byte. A zero will disable the bit. Refer to table 5-1 for information about the Standard Event Status Enable Register bits, bit weights, and what each bit masks.

The *ESE query returns the current contents of the enable register.

Note

Refer to Appendix B for a complete discussion of status.

Command Syntax: *ESE <mask>

where:

<mask> ::= 0 to 255 (integer)

Example: OUTPUT XXX;"*ESE 32"

In this example, the *ESE 32 command will enable CME (Command Error), bit 5 of the Standard Event Status Enable Register. Therefore, when a command error occurs, the event summary bit (ESB) in the Status Byte Register will also be set.

*ESE

Query Syntax: *ESE?

Returned Format: <mask><NL>

Example:
```
10 DIM Event$[100]
20 OUTPUT XXX;"*ESE?"
30 ENTER XXX;Event$
40 PRINT Event$
50 END
```

Table 5-1. Standard Event Status Enable Register

Bit	Weight	Enables
7	128	PON - Power On
6	64	URQ - User Request
5	32	CME - Command Error
4	16	EXE - Execution Error
3	8	DDE - Device Dependent Error
2	4	QYE - Query Error
1	2	RQC - Request Control
0	1	OPC - Operation Complete

High - enables the ESR bit

*ESR (Event Status Register) query

The *ESR query returns the contents of the Standard Event Status Register. Reading the register clears the Standard Event Status Register.

Query Syntax: *ESR?

Returned Format: <status><NL>

where:

<status> ::= 0 to 255 (integer)

Example:

```
10 DIM Esr_event$[100]
20 OUTPUT XXX;"*ESR?"
30 ENTER XXX;Esr_event$
40 PRINT Esr_event$
50 END
```

With the example, if a command error has occurred the variable "Esr_event" will have bit 5 (the CME bit) set.

Table 5-2 shows the Standard Event Status Register. The table shows each bit in the Standard Event Status Register, and the bit weight. When you read Standard Event Status Register, the value returned is the total bit weights of all bits that are high at the time you read the byte.

Table 5-2. The Standard Event Status Register.

BIT	BIT WEIGHT	BIT NAME	CONDITION
7	128	PON	0 = Register read - not in power up mode 1 = Power up
6	64	URQ	0 = user request - not used - always zero
5	32	CME	0 = no command errors 1 = a command error has been detected
4	16	EXE	0 = no execution errors 1 = an execution error has been detected
3	8	DDE	0 = no device dependent errors 1 = a device dependent error has been detected
2	4	QYE	0 = no query errors 1 = a query error has been detected
1	2	RQC	0 = request control - NOT used - always 0
0	1	OPC	0 = operation is not complete 1 = operation is complete

0 = False = Low
1 = True = High

*IDN (Identification Number) query

The *IDN? query allows the instrument to identify itself. It returns the string:

"HEWLETT-PACKARD,16500A,0,REV <revision code>"

An *IDN? query must be the last query in a message. Any queries after the *IDN? in the program message will be ignored.

Query Syntax: *IDN?

Returned Format: HEWLETT-PACKARD,16500A,0,REV <revision code>

where:

<revision code> ::= four digit code in the format XX.XX representing the current ROM revision

Example:

```
10 DIM Id$[100]
20 OUTPUT XXX;"*IDN?"
30 ENTER XXX;Id$
40 PRINT Id$
50 END
```

*IST (Individual Status) query

The *IST query allows the instrument to identify itself during parallel poll by allowing the controller to read the current state of the IEEE 488.1 defined "ist" local message in the instrument. The response to this query is dependent upon the current status of the instrument.

Figure 5-2 shows the *IST data structure.

Query Syntax: *IST?

Returned Format: <id><NL>

where:

<id> ::= 0 or 1
1 ::= indicates the "ist" local message is false
0 ::= indicates the "ist" local message is true

Example:

```
10 DIM Event$[100]
20 OUTPUT XXX;"*IST?"
30 ENTER XXX;Event$
40 PRINT Event$
50 End
```

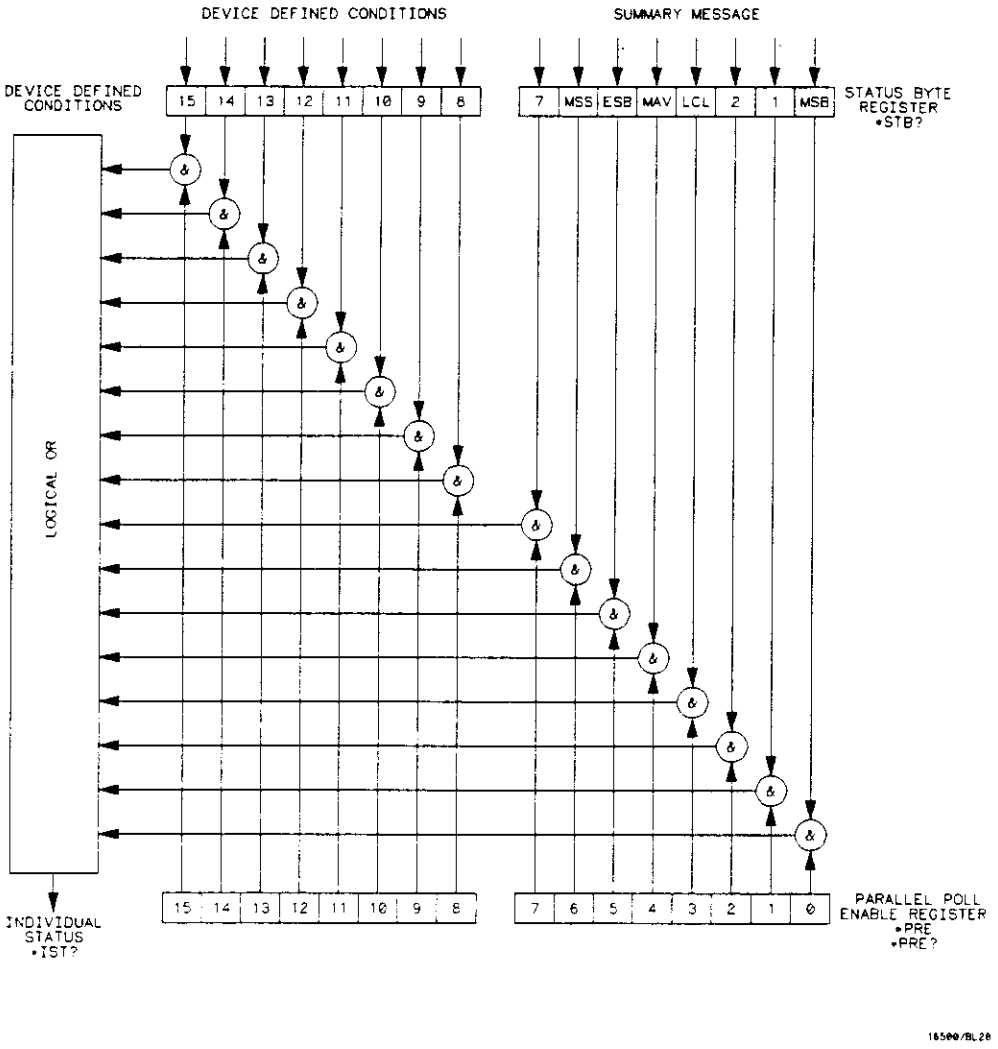


*Figure 5-2. *IST Data Structure*

*OPC (Operation Complete) command/query

The *OPC command will cause the instrument to set the operation complete bit in the Standard Event Status Register when all pending device operations have finished. The commands which affect this bit are the Overlapped Commands. An Overlapped Command is a command that allows execution of subsequent commands while the device operations initiated by the Overlapped Command are still in progress. Some examples of overlapped commands for the HP 16500A are:

STARt
STOP

Additional overlapped commands are defined in the individual programming manuals for each module.

The *OPC query places an ASCII "1" in the output queue when all pending device operations have been completed.

Command Syntax: *OPC

Example:
```
OUTPUT XXX;"*OPC"
```

Query Syntax: *OPC?

Returned Format: 1<NL>

Example:
```
DIM Status$[100]
20 OUTPUT XXX;"*OPC?"
30 ENTER XXX;Status$
40 PRINT Status$
50 END
```

*OPT (Option Identification) query

The *OPT query identifies the software installed in the HP 16500A. This query returns nine parameters. The first parameter indicates whether you are in the System. The next two parameters indicate any software options installed, and the next parameter indicates whether intermodule is available for the System. The last five parameters list the installed software for the modules in slot A through E of the mainframe. A zero in any of the last eight parameters indicates that the corresponding software is not currently installed.

Query Syntax: *OPT?

Returned Format: {SYSTEM},{ <option> |0},{ <option> |0},
{INTERMODULE|0},{ <module> |0},{ <module> |0},
{ <module> |0},{ <module> |0},{ <module> |0}<NL>

where:

<option> ::= name of software option
<module> ::= name of module software

Note

The name returned for software options and module software is the same name that appears in the field in the upper-left corner of the menu for each option or module.

Example:

```
10 DIM Option$[200]
20 OUTPUT XXX;"*OPT?"
30 ENTER XXX;Option$
40 PRINT Option$
50 END
```

*PRE (Parallel Poll Enable Register Enable) command/query

The *PRE command sets the parallel poll register enable bits. The Parallel Poll Enable Register contains a mask value which is ANDed with the bits in the Status Bit Register to enable an "ist" during a parallel poll. The query returns the current value of the register.

Refer to table 5-3 for the bits in the Parallel Poll Enable Register and what they mask.

Command Syntax: *PRE <mask>

where:

<pre_mask> ::= 0 to 65535 (integer)

Example:
```
Output XXX;"*PRE 16"
```

This example will allow the HP 16500A to generate an "ist" when a message is available in the output queue. When a message is available, the MAV (Message Available) bit in the Status Byte Register will be high.

Query Syntax: "*PRE?"

Returned format: <mask> <NL>

where:

<mask> ::= sum of all bits that are set - 0 through 65535

Example:
```
10 DIM Pre_value$[100]
20 OUTPUT XXX;"*PRE?"
30 ENTER XXX;Pre_value$
40 PRINT Pre_value$
50 END
```

Table 5-3. HP 16500A Parallel Poll Enable Register

Bit	Weight	Enables
15-8		not used
7	128	not used
6	64	MSS - Master Summary Status
5	32	ESB - Event Status
4	16	MAV - Message Available
3	8	LCL - Local
2	4	not used
1	2	not used
0	1	MSB - Module Summary

*RST (Reset) command

The *RST command (488.2) is not implemented on the HP 16500A. The HP 16500A will accept this command, but the command has no affect on the instrument.

Note

*The *RST command is generally used to place the instrument in a predefined state. Since the HP 16500A allows you to store predefined configuration files for individual modules or the entire system, resetting the instrument can be accomplished by simply loading the appropriate configuration file. For more information, refer to the chapter "Mmemory Subsystem" in this manual.*

*SRE (Service Request Enable) command/query

The *SRE command sets the Service Request Enable Register bits. The Service Request Enable Register contains a mask value for the bits to be enabled in the Status Byte Register. A one in the Service Request Enable Register will enable the corresponding bit in the Status Byte Register. A zero will disable the bit. Refer to table 5-4 for the bits in the Service Request Enable Register and what they mask.

The *SRE query returns the current value.

Note

Refer to Appendix B for a complete discussion of status.

Command Syntax: *SRE <mask>

where:

<mask> ::= 0 to 255 (integer)

Example: OUTPUT XXX;"*SRE 16"

This example enables a service request to be generated when a message is available in the output queue. When a message is available, the MAV (Message Available) bit will be high.

*SRE

Query Syntax: *SRE?

Returned Format: <mask><NL>

where:

<mask> ::= sum of all bits that are set - 0 through 255

Example:

```
10 DIM Sre_value$[100]
20 OUTPUT XXX;"*SRE?"
30 ENTER XXX;Sre_value$
40 PRINT Sre_value$
50 END
```

Table 5-4. HP 16500A Service Request Enable Register

Bit	Weight	Enables
15-8		not used
7	128	not used
6	64	MSS - Master Summary Status
5	32	ESB - Event Status
4	16	MAV - Message Available
3	8	LCL - Local
2	4	not used
1	2	not used
0	1	MSB - Module Summary

*STB (Status Byte) query

The *STB query returns the current value of the instrument's status byte. The MSS (Master Summary Status) bit and not RQS (Request Service) bit is reported on bit 6. The MSS indicates whether or not the device has at least one reason for requesting service. Refer to table 5-5 for the meaning of the bits in the status byte.

Note

Refer to Appendix B for a complete discussion of status.

Query Syntax: *STB?

Returned Format: <value><NL>

where:

<value> ::= 0 through 255 (integer)

Example:

```
10 DIM Stb_value$[100]
20 OUTPUT XXX;"*STB?"
30 ENTER XXX;Stb_value$
40 PRINT Stb_value$
50 END
```

Table 5-5. The Status Byte Register.

BIT	BIT WEIGHT	BIT NAME	CONDITION
7	128	---	0 = not used
6	64	MSS	0 = instrument has no reason for service 1 = instrument is requesting service
5	32	ESB	0 = no event status conditions have occurred 1 = an enabled event status condition has occurred
4	16	MAV	0 = no output messages are ready 1 = an output message is ready
3	8	LCL	0 = a remote-to-local transition has not occurred 1 = a remote-to-local transition has occurred
2	4	---	not used
1	2	---	not used
0	1	MSB	0 = a module or the system has activity to report 1 = no activity to report

0 = False = Low
1 = True = High

*TRG (Trigger) command

The *TRG command has the same effect as a Group Execute Trigger (GET). That effect is as if the START command had been sent for intermodule group run. If no modules are configured in the Intermodule menu, this command has no effect.

Command Syntax: *TRG

Example: OUTPUT XXX;"*TRG"

*TST (Test) query

The *TST query returns the results of the power-up self-test. The result of the test is a 9-bit mapped value which is placed in the output queue. A one in the corresponding bit means that the test failed and a zero in the corresponding bit means that the test passed. Refer to table 5-6 for the meaning of the bits returned by a TST? query.

Query Syntax: *TST?

Returned Format: <result><NL>

where:

<result> ::= 0 through 511 (integer)

Example:
```
10 OUTPUT XXX;"*TST?"
20 ENTER XXX;Tst_value
30 PRINT Tst_value
40 END
```

Table 5-6. Bits Returned by TST? Query (Power-Up Test Results).

BIT	BIT WEIGHT	TEST
8	256	Front Disc Test
7	128	Rear Disc Test
6	64	Touchscreen Test
5	32	(not used - always zero)
4	16	(not used - always zero)
3	8	Display Test
2	4	Interrupt Test
1	2	RAM Test
0	1	ROM Test

*WAI (Wait) command

The *WAI command causes the device to wait until the completion of all overlapped commands before executing any further commands or queries. An overlapped command is a command that allows execution of subsequent commands while the device operations initiated by the overlapped command are still in progress. Some examples of overlapped commands for the HP 16500A are:

STARt
STOP

Additional overlapped commands are defined in the individual programming manuals for each module.

Command Syntax: *WAI

Example: OUTPUT XXX;"*WAI"

Mainframe Commands 6

Introduction

Mainframe commands control the basic operation of the instrument. They can be called at anytime, and from any module. Refer to figure 6-1 for the Mainframe commands syntax diagram.

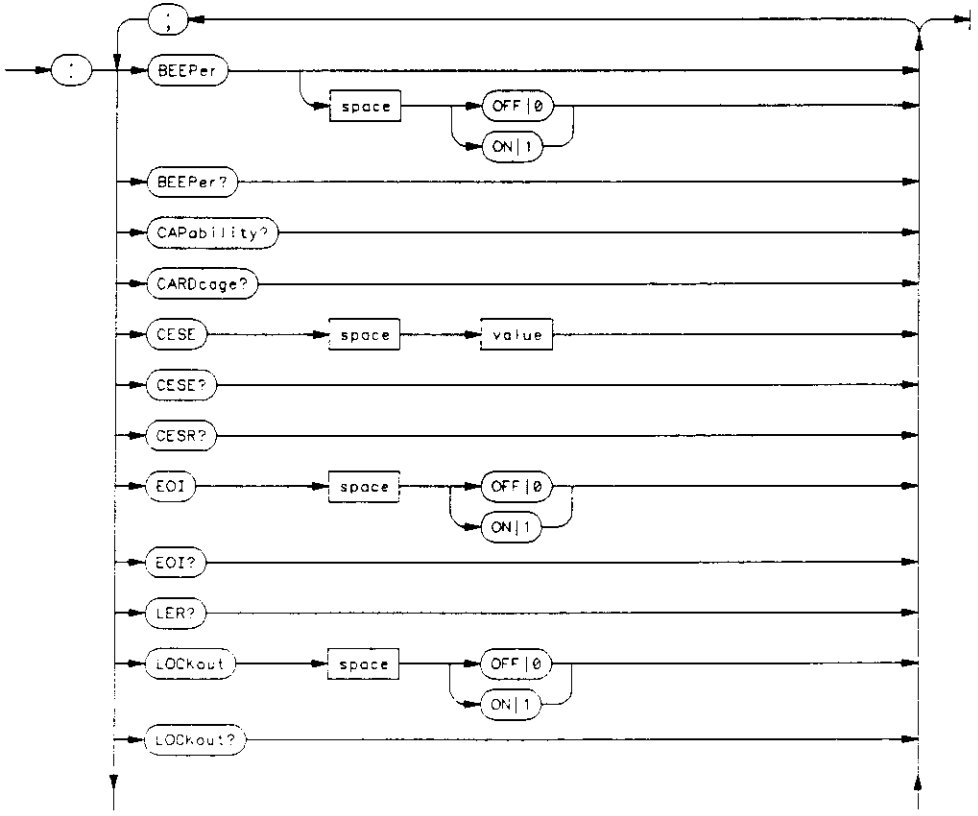
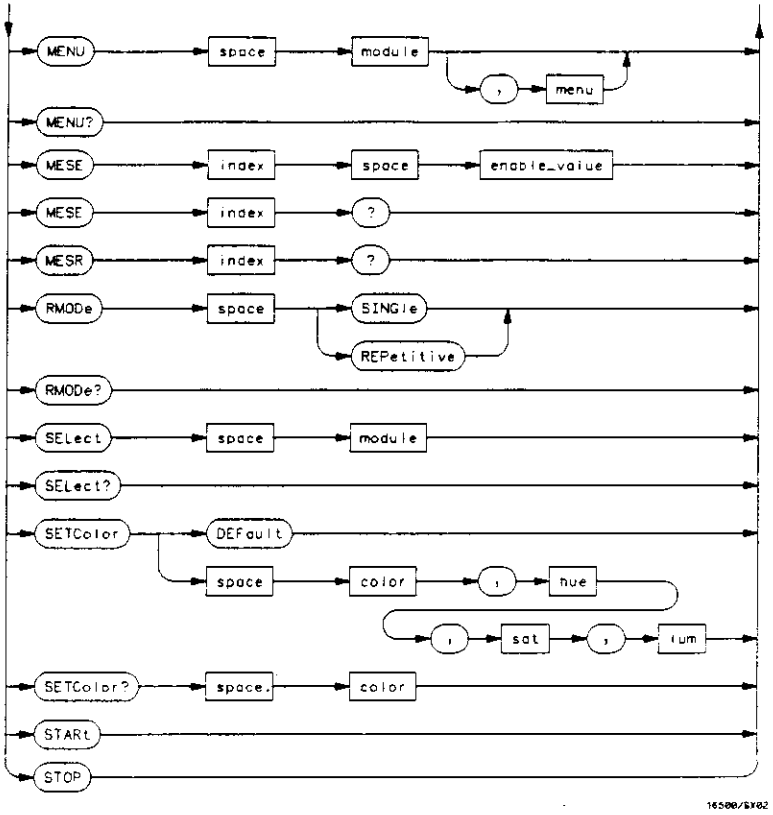


Figure 6-1. Mainframe Commands Syntax Diagram



value = *integer, 0 to 255.*
module = *integer, -2 to 5.*
menu = *integer. Refer to the individual programming manuals for each module and the system for specific menu number definitions.*
enable_value = *integer, 0 to 255.*
index = *integer, 0 to 5.*
color = *integer, 0 to 7. Color number 0 cannot be changed.*
hue = *integer, 0 to 100.*
sat = *integer, 0 to 100.*
lum = *integer, 0 to 100.*

Figure 6-1. Mainframe Commands Syntax Diagram (Continued)

BEEPer command/query

The BEEPer command sets the beeper mode, which turns the beeper sound of the instrument on and off. When BEEPer is sent with no argument, the beeper will be sounded without affecting the current mode. The query returns the mode currently selected.

Command Syntax: :BEEPer [{ON|1}|{OFF|0}]

Examples:
```
OUTPUT XXX;":BEEPER"
OUTPUT XXX;":BEEP ON"
```

Query Syntax: :BEEPer?

Returned Format: [:BEEPer] {1|0}<NL>

Example:
```
10 DIM Mode$[100]
20 OUTPUT XXX;":BEEPER?"
30 ENTER XXX;Mode$
40 PRINT Mode$
50 END
```

CAPability query

The CAPability query returns the HP-SL and lower level capability sets implemented in the device.

Table 6-1 lists the capability sets implemented in the HP 16500A

Query Syntax: :CAPability?

Returned Format: [:CAPability] IEEE488,1987,SH1,AH1,T5,L4,SR1,RL1,PP1,DC1,DT1,C0,E2<NL>

Example:
```
10 DIM Response$[100]
20 OUTPUT XXX;":CAPABILITY?"
30 ENTER XXX;Response$
40 PRINT Response$
50 END
```

Table 6-1. HP 16500A Capability Sets

Mnemonic	Capability Name	Implementation
SH	Source Handshake	SH1
AH	Acceptor Handshake	AH1
T	Talker (or TE - Extended Talker)	T5
L	Listener (or LE - Extended Listener)	L4
SR	Service Request	SR1
RL	Remote Local	RL1
PP	Parallel Poll	PP1
DC	Device Clear	DC1
DT	Device Trigger	DT1
C	Any Controller	C0
E	Electrical Characteristic	E2

CARDcage query

The CARDcage query returns a series of integers which identifies the modules that are installed in the mainframe. The first five numbers returned are the card identification numbers (a -1 means no card is in the slot). The remaining five numbers returned indicate the module assignment for each card. The possible values for the module assignment are 0, 1, 2, 3, 4, and 5 where 0 indicates an empty slot or the module software is not recognized or not loaded. 1...5 indicates the number of the slot in which the master card for this card is located.

Table 6-2 lists the card identification numbers for the first five parameters and their associated cards.

Query Syntax: :CARDcage?

Returned Format: [:CARDcage] <ID>,<ID>,<ID>,<ID>,<ID>,
<assign>,<assign>,<assign>,<assign>,<assign><NL>

where:

<ID> ::= card identification number (integer)
<assign> ::= module assignment (integer)

Example:

```
10 DIM Card$[100]
20 OUTPUT XXX;":CARDCAGE?"
30 ENTER XXX;Card$
40 PRINT Card$
50 END
```

Table 6-2. Card Identification Numbers

ID Number	Card
31	HP 16510A Logic Analyzer Card
1	HP 16515A 1 GHz Timing Master Card
2	HP 16516A I GHz Timing Expansion Card
21	HP 16520A Pattern Generator Master Card
22	HP 16521A Pattern Generator Expansion Card
11	HP 16530A Oscilloscope Timebase Card
12	HP 16531A Oscilloscope Acquisition Card

Note

Refer to the individual programming manuals for each module for cards not listed in table 6-2.

CESE (Combined Event Status Enable) command/query

The CESE command sets the Combined Event Status Enable register. This register is the enable register for the CESR register and contains the combined status of all of the MESE (Module Event Status Enable) registers of the HP 16500A. The query returns the current setting.

Table 6-3 lists the bit values for the CESE register.

Command Syntax: :CESE <value>

where:

<value> ::= 0 to 255 (integer)

Example:
```
OUTPUT XXX;":CESE 32"
```

Query Syntax: :CESE?

Returned Format: [:CESE] <value> <NL>

Example:
```
10 DIM Setting$[100]
20 OUTPUT XXX;":CESE?"
30 ENTER XXX;Setting$
40 PRINT Setting$
50 END
```

Table 6-3. HP 16500A Combined Event Status Enable Register

Bit	Weight	Enables
7	128	not used
6	64	not used
5	32	Module in slot E
4	16	Module in slot D
3	8	Module in slot C
2	4	Module in slot B
1	2	Module in slot A
0	1	Intermodule

CESR (Combined Event Status Register) query

The CESR query returns the contents of the Combined Event Status register. This register contains the combined status of all of the MESRs (Module Event Status Registers) of the HP 16500A. Table 6-4 lists the bit values for the CESR register.

Query Syntax: :CESR?

Returned Format: [:CESR] <value> <NL>

where:

<value> ::= 0 to 255 (integer)

Example:
```
10 DIM Event$[100]
20 OUTPUT XXX;":CESR?"
30 ENTER XXX;Event$
40 PRINT Event$
50 END
```

Table 6-4. HP 16500A Combined Event Status Register

Bit	Bit Weight	Bit Name	Condition
7	128		0 = not used
6	64		0 = not used
5	32	Module E	0 = No New Status 1 = Status to Report
4	16	Module D	0 = No New Status 1 = Status to Report
3	8	Module C	0 = No New Status 1 = Status to Report
2	4	Module B	0 = No New Status 1 = Status to Report
1	2	Module A	0 = No New Status 1 = Status to Report
0	1	Intermodule	0 = No New Status 1 = Status to Report

EOI (End Or Identify) command/query

The EOI command specifies whether or not the last byte of a reply from the instrument is to be sent with the EOI bus control line set true or not. If EOI is turned off, the box will no longer be sending 488.2 compliant responses. The query returns the current status of EOI.

Command Syntax: :EOI {{ON|1}|{OFF|0}}

Example: OUTPUT XXX;":EOI ON"

Query Syntax: :EOI?

Returned Format: [:EOI] {1|0}<NL>

Example:
```
10 DIM Mode$[100]
20 OUTPUT XXX;":EOI?"
30 ENTER XXX;Mode$
40 PRINT Mode$
50 END
```

LER (LCL Event Register) query

The LER query allows the LCL Event Register to be read. After the LCL Event Register is read, it is cleared. A one indicates a remote-to-local transition has taken place. A zero indicates a remote-to-local transition has not taken place.

Query Syntax: :LER?

Returned Format: [:LER] {0|1}<NL>

Example:

```
10 DIM Event$[100]
20 OUTPUT XXX;":LER?"
30 ENTER XXX;Event$
40 PRINT Event$
50 END
```

LOCKout command/query

The LOCKout command locks out or restores front panel operation. When this function is on, all controls (except the power switch) are entirely locked out. The LOCKout query returns the current status of the LOCKout command.

Command Syntax: :LOCKout {{ON|1}|{OFF|0}}

Example:
```
OUTPUT XXX;":LOCKOUT ON"
```

Query Syntax: :LOCKout?

Returned Format: [:LOCKout] {0|1}<NL>

Example:
```
10 DIM Status$[100]
20 OUTPUT XXX;":LOCKOUT?"
30 ENTER XXX;Status$
40 PRINT Status$
50 END
```

MENU command/query

The MENU command puts a menu on the display. The first parameter specifies the desired module. The optional second parameter specifies the desired menu in the module (defaults to 0).

Command Syntax: :MENU <module>[,<menu>]

where:

<module> ::= selects module or system (-2 to 5) (integer)
<menu> ::= selects menu (integer)

Example: OUTPUT XXX;":MENU 0,1"

For the first parameter:

0 - System/Intermodule
1 - Module in slot A
2 - Module in slot B
3 - Module in slot C
4 - Module in slot D
5 - Module in slot E
-1 - Software option 1
-2 - Software option 2

For the System:

MENU 0,0 - System Configuration menu
MENU 0,1 - Rear disc menu
MENU 0,2 - Front disc menu
MENU 0,3 - Utilities menu
MENU 0,4 - Test menu
MENU 0,5 - Intermodule menu

Note

Refer to the individual programming manuals for each module for specific menu number definitions for each module.

The MENU query returns the current menu selection.

Query Syntax: :MENU?

Returned Format: [:MENU] <module>,<menu><NL>

Example:
```
10 DIM Response$[100]
20 OUTPUT XXX;":MENU?"
30 ENTER XXX;Response$
40 PRINT Response$
50 END
```

MESE<N> (Module Event Status Enable) command/query

The MESE command sets the Module Event Status Enable register. This register is the enable register for the MESR register. The <N> index specifies the module, and the parameter specifies the enable value. 1...5 refers to module in slot A...E and 0 refers to intermodule. The query returns the current setting.

Refer to table 6-5 and the individual programming manuals for each module for information about the Module Event Status Enable register bits, bit weights, and what each bit masks.

Command Syntax: :MESE<N> <enable_value>

where:

<N> ::= 0 through 5 (integer)
<enable_value> ::= 0 through 255 (integer)

Example:
```
OUTPUT XXX;":MESE1 3"
```

Query Syntax: :MESE<N>?

Returned Format: [:MESE<N>] <enable_value> <NL>

Example:
```
10 DIM Event$[100]
20 OUTPUT XXX;":MESE1?"
30 ENTER 707;Event$
40 PRINT Event$
50 END
```

Table 6-5. HP 16500A Mainframe (Intermodule) Module Event Status Enable Register

Bit	Weight	Enables
7	128	not used
6	64	not used
5	32	not used
4	16	not used
3	8	not used
2	4	not used
1	2	RNT - Intermodule Run Until Satisfied
0	1	MC - Intermodule Measurement Complete

MESR<N> (Module Event Status Register) query

The MESR query returns the contents of the Module Event Status register. The <N> index specifies the module. 1...5 refers to module in slot A...E and 0 refers to intermodule.

Refer to table 6-6 and the individual programming manuals for each module for information about the Module Event Status Register bits and their bit weights.

Query Syntax: :MESR<N>?

Returned Format: [:MESR<N>] <enable_value><NL>

where:

<N> ::= 0 through 5 (integer)
<enable_value> ::= 0 through 255 (integer)

Example:

```
10 DIM Event$[100]
20 OUTPUT XXX;":MESR1?"
30 ENTER XXX;Event$
40 PRINT Event$
50 END
```

Table 6-6. HP 16500A Mainframe Module Event Status Register

Bit	Bit Weight	Bit Name	Condition
7	128		0 = not used
6	64		0 = not used
5	32		0 = not used
4	16		0 = not used
3	8		0 = not used
2	4		0 = not used
1	2	RNT	0 = Intermodule Run until not satisfied 1 = Intermodule Run until satified
0	1	MC	0 = Intermodule Measurement not complete 1 = Intermodule Measurement complete

RMODe command/query

The RMODe command specifies the run mode for the selected module (or Intermodule). The query returns the current setting. If the selected module is in the intermodule configuration, then the "intermodule" run mode will be set by this command.

Note

After specifying the run mode, use the STARt command to start the acquisition.

Command Syntax: :RMODe {SINGle|REPetitive}

Example:
```
OUTPUT XXX;":RMODE SINGLE"
```

Query Syntax: :RMODe?

Returned Format: [:RMODe] {SINGle|REPetitive}<NL>

Example:
```
10 DIM Mode$[100]
20 OUTPUT XXX;":RMODE?"
30 ENTER XXX;Mode$
40 PRINT Mode$
50 END
```

SELect command/query

The SELect command selects which module (or System) will have parser control. The appropriate module (or System) must be selected before any module (or system) specific commands can be sent. SELECT 0 selects System, SELECT 1 through 5 selects modules A through E. -1 and -2 selects software options 1 and 2 respectively. The query returns the current module selection.

Figure 6-2 shows the command tree for the SELect command.

Note

SELect defaults to System (0) at power up.

Command Syntax: :SELect <module>

where:

<module> ::= -2 through 5 (integer)

Example: OUTPUT XXX;":SELECT 0"

Query Syntax: :SELect?

Returned Format: [:SELect] <module><NL>

Example:
```
10 DIM Module$[100]
20 OUTPUT XXX;":SELECT?"
30 ENTER XXX;Module$
40 PRINT Module$
50 END
```

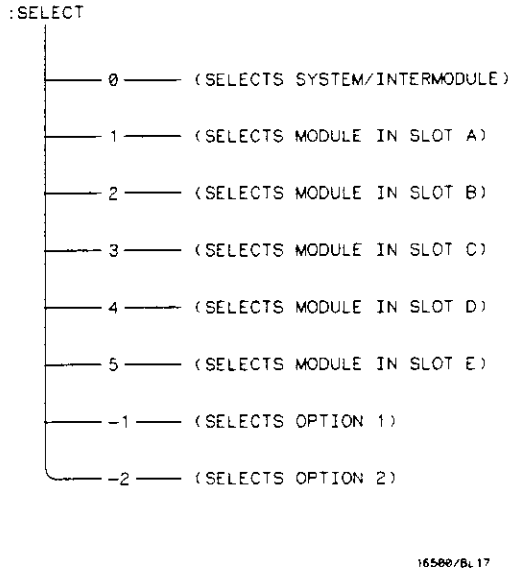


Figure 6-2. Select Command Tree

Note

When a module is selected, the parser recognizes the module's commands and the System/Intermodule commands. When SELECT 0 is used, only the System/Intermodule commands are recognized by the parser.

SETColor

command/query

The SETColor command is used to change one of the color selections on the CRT, or to return to the default screen colors. Four parameters are sent with the command to change a color:

- Color Number (first parameter),
- Hue (second parameter),
- Saturation (third parameter), and
- Luminosity (last parameter).

Command Syntax: :SETColor {<color>,<hue>,<sat>,<lum>|DEFault}

where:

<color> ::= 0 to 7 (integer)
<hue> ::= 0 to 100 (integer)
<sat> ::= 0 to 100 (integer)
<lum> ::= 0 to 100 (integer)

Note

Color Number 0 cannot be changed.

Example:
```
OUTPUT XXX;":SETCOLOR 3,60,100,60"
OUTPUT XXX;":SETC DEFAULT"
```

SETColor

The SETColor query returns the hue, saturation, and luminosity values for a specified color.

Query Syntax: :SETColor? <color>

Returned Format: [:SETColor] <color>,<hue>,<sat>,<lum><NL>

Example:

```
10 DIM Color$[100]
20 OUTPUT XXX;":SETCOLOR? 3"
30 ENTER XXX;Color$
40 PRINT Color$
50 END
```

STARt command

The STARt command starts the selected module (or Intermodule) running in the specified run mode (see RMODe). If the specified module is in the Intermodule configuration, then the "Intermodule" run will be started.

Note

*The STARt command is an Overlapped Command. An Overlapped Command is a command that allows execution of subsequent commands while the device operations initiated by the Overlapped Command are still in progress. For more information, refer to the *OPC and *WAI commands in the chapter "Common Commands."*

Command Syntax: :STARt

Example: OUTPUT XXX;":START"

STOP command

The STOP command stops the selected module (or Intermodule). If the specified module is in the Intermodule configuration, then the "Intermodule" run will be stopped.

Note

*The STOP command is an Overlapped Command. An Overlapped Command is a command that allows execution of subsequent commands while the device operations initiated by the Overlapped Command are still in progress. For more information, refer to the *OPC and *WAI commands in the chapter "Common Commands."*

Command Syntax: :STOP

Example: OUTPUT XXX;":STOP"

SYSTem Subsystem 7

Introduction

SYSTem subsystem commands control functions that are common to all modules, including formatting query responses and enabling reading and writing to the advisory line of the instrument. Refer to figure 7-1 for the System Subsystem commands syntax diagram.

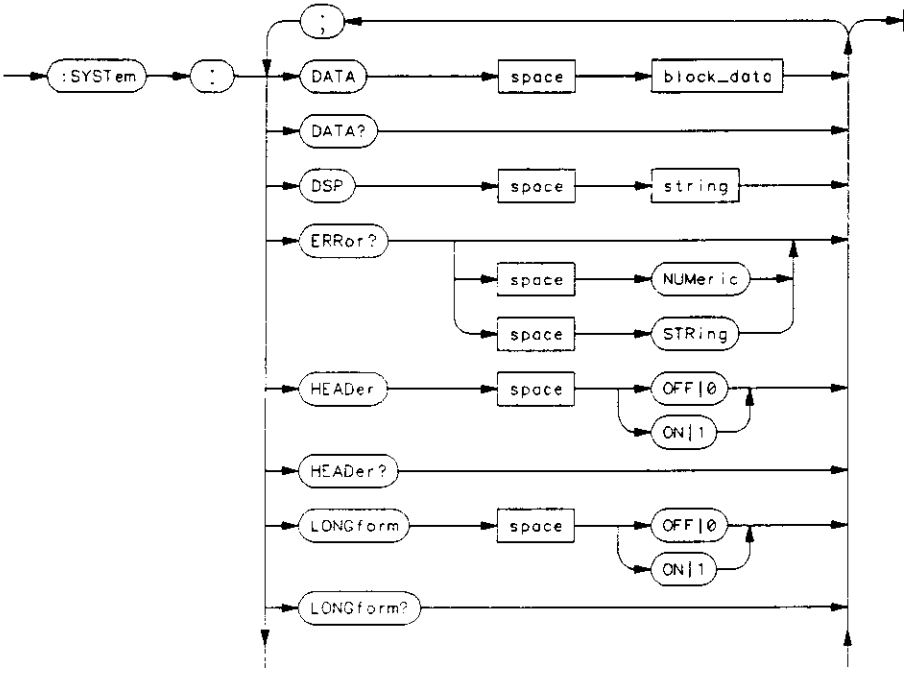
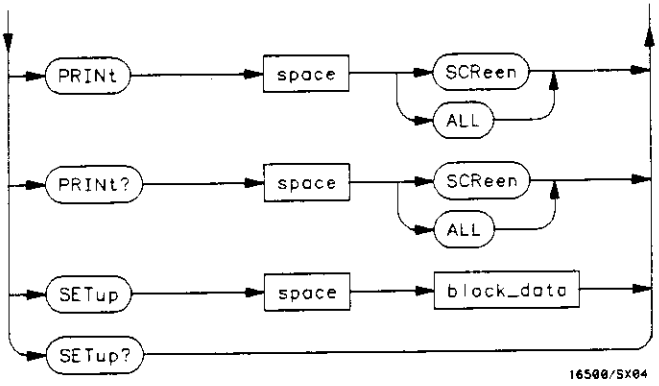


Figure 7-1. System Subsystem Commands Syntax Diagram



block_data = *data in IEEE 488.2 # format.*
string = *string of up to 68 alphanumeric characters.*

Figure 7-1. System Subsystem Commands Syntax Diagram (Continued)

DATA command/query

The DATA command transmits the data part of the setup configuration of the selected module in block data format. The DATA query returns the current contents of the acquisition buffer from the selected module to the controller.

Note

The DATA command is only used in conjunction with modules that are loaded in the mainframe. The System does not contain any acquired data.

Command Syntax: :SYSTem:DATA <block data in # format>

Query Syntax: :SYSTem:DATA?

Returned Format: [:SYSTem:DATA] <block data in # format><NL>

Definition of Block Data Block data in the # format is made up of a block length specifier and a variable number of sections.

<block length specifier><section 1><section 2>

The block length specifier is defined as follows:

#8<length>

where:

<length> ::= the total length of all sections in byte format (must be represented with 8 digits)

For example, if the total length of the block (all the sections) is 144 bytes, the block length specifier would be "#800000144" since the length must be represented with 8 digits.

Sections consist of a section header followed by the section data as follows:

<section header> <section data>

where:

<section header> ::= 10 bytes for the section name
1 byte reserved (always 0)
1 byte for the module ID number (see table 7-1)
4 bytes for the length of the section data in bytes

The section data format varies for each section and may be any length.

Note

The total length of a section is 16 (for the section header) plus the length of the section data. Thus, when calculating the length of a block of configuration data, care should be taken to not forget to add the length of the section headers.

HP-IB Example:

```
10 DIM Block$[32000]  !allocate enough memory for block data
20 DIM Specifier$[2]
30 OUTPUT XXX;":EOI ON"
40 OUTPUT XXX;":SYSTEM:HEAD OFF"
50 OUTPUT XXX;":SELECT 4"     !select module
60 OUTPUT XXX;":SYSTEM:DATA?"  !send data query
70 ENTER XXX USING "#,2A";Specifier$  !read in #8
80 ENTER XXX USING "#,8D";Blocklength  !read in block length
90 ENTER XXX USING "-K";Block$  !read in data
100 END
```

Table 7-1. Card Identification Numbers

ID Number	Card
31	HP 16510A Logic Analyzer Card
1	HP 16515A 1 GHz Timing Master Card
2	HP 16516A I GHz Timing Expansion Card
21	HP 16520A Pattern Generator Master Card
22	HP 16521A Pattern Generator Expansion Card
11	HP 16530A Oscilloscope Timebase Card
12	HP 16531A Oscilloscope Acquisition Card

DSP (Display) command

The DSP command writes the specified quoted string to a device dependent portion of the instrument display.

Command Syntax: :SYSTem:DSP <string>

where:

<string> ::= string of up to 68 alphanumeric characters

Example: OUTPUT XXX;":SYSTEM:DSP 'The message goes here'"

ERRor query

The ERRor query returns the oldest error from the error queue. The optional parameter determines whether the error string should be returned along with the error number. If no parameter is received, or if the parameter is NUM, then only the error number is returned. If the value of the parameter is STRing, then the error should be returned in the following form:

<error number>,<error message (string)>

A complete list of error messages for the HP 16500A mainframe is shown in appendix C. If no errors are present in the error queue, a zero (No Error) is returned.

Query Syntax: :SYSTem:ERRor? [NUMeric|STRing]

Returned Formats: **Numeric:**

[:SYSTem:ERRor] <error number><NL>

String:

[:SYSTem:ERRor] <error number>,<string><NL>

where:

<error number> ::= integer

Examples: **Numeric:**

```
10 OUTPUT XXX;":SYSTEM:ERROR?"
20 ENTER XXX;Numeric
30 PRINT Numeric
40 END
```

String:

```
10 DIM String$[100]
20 OUTPUT XXX;":SYST:ERR? STRING"
30 ENTER XXX;String$
40 PRINT String$
50 END
```

HEADer

command/query

The HEADer command tells the instrument whether or not to output a header for query responses. When HEADer is set to ON, query responses will include the command header.

The HEADer query returns the current state of the HEADer command.

Command Syntax: :SYSTem:HEADer {{ON|1}|{OFF|0}}

Example:
```
OUTPUT XXX;":SYSTEM:HEADER ON"
```

Query Command: :SYSTem:HEADer?

Returned Format: [:SYSTem:HEADer] {1|0}<NL>

Example:
```
10 DIM Mode$[100]
20 OUTPUT XXX;":SYSTEM:HEADER?"
30 ENTER XXX;Mode$
40 PRINT Mode$
50 END
```

Note

Headers should be turned off when returning values to numeric variables.

LONGform command/query

The LONGform command sets the longform variable which tells the instrument how to format query responses. If the LONGform command is set to OFF, command headers and alpha arguments are sent from the instrument in the abbreviated form. If the the LONGform command is set to ON, the whole word will be output. This command has no affect on the input data messages to the instrument. Headers and arguments may be input in either the longform or shortform regardless of how the LONGform command is set. The query returns the status of the LONGform command.

Command Syntax: :SYSTem:LONGform {{ON|1}|{OFF|0}}

Example:
```
OUTPUT XXX;":SYSTEM:LONGFORM ON"
```

Query Syntax: :SYSTem:LONGform?

Returned Format: [:SYSTem:LONGform] {1|0}<NL>

Example:
```
10 DIM Mode$[100]
20 OUTPUT XXX;":SYSTEM:LONGFORM?"
30 ENTER XXX;Mode$
40 PRINT Mode$
50 END
```

PRINt command/query

The PRINt command initiates a print of the screen or listing buffer over the current PRINTER communication interface. The query sends the screen or listing buffer data over the current CONTROLLER communication interface.

Note

The print query should NOT be sent in conjunction with any other command or query on the same command line.

The print query never returns a header. Also, since response data from a print query may be sent directly to a printer without modification, the data is not returned in block mode.

Command Syntax: :SYSTem:PRINt {SCReen|ALL}

Example: OUTPUT XXX;":SYSTEM:PRINT SCREEN"

PRINt

Query Syntax: SYSTem:PRINt? {SCReen | ALL}

Note

PRINT? ALL is only available in menus that have the "Print All" option available on the front panel. For more information, refer to the individual front-panel manuals for each module.

HP-IB Example:

```
10 OUTPUT 707;":SYSTEM:PRINT? SCREEN"
20 SEND 7;UNT UNL
30 SEND 7;LISTEN 1
40 SEND 7;TALK 7
50 SEND 7;DATA    !drop ATN line
60 PRINT "WAITING FOR PRINT"
70 END
```

SETup command/query

The SETup command configures the selected module (or System) as defined by the block of data sent by the controller. The query returns a block of data that contains the current configuration for the selected module (or System) to the controller.

Command Syntax: :SYSTem:SETup <block data in # format>

Query Syntax: :SYSTem:SETup?

Returned Format: [:SYSTem:SETup] <block data in # format> <NL>

Definition of Block Data Block data in the # format is made up of a block length specifier and a variable number of sections.

<block length specifier> <section 1> <section 2>

The block length specifier is defined as follows:

#8<length>

where:

<length> ::= the total length of all sections in byte format (must be represented with 8 digits)

For example, if the total length of the block (all the sections) is 144 bytes, the block length specifier would be "#800000144" since the length must be represented with 8 digits.

Sections consist of a section header followed by the section data as follows:

<section header> <section data>

where:

<section header> ::= 10 bytes for the section name
1 byte reserved (always 0)
1 byte for the module ID number (see table 7-2)
4 bytes for the length of the section data in bytes

The section data format varies for each section and may be any length.

Note

The total length of a section is 16 (for the section header) plus the length of the section data. Thus, when calculating the length of a block of configuration setup data, care should be taken to not forget to add the length of the section headers.

HP-IB Example:

```
10 DIM Block$[32000] !allocate enough memory for block data
20 DIM Specifier$[2]
30 OUTPUT XXX;":EOI ON"
40 OUTPUT XXX;":SYSTEM:HEAD OFF"
50 OUTPUT XXX;":SELECT 0"     !select System
60 OUTPUT XXX;":SYSTEM:SETUP?" !send setup query
70 ENTER XXX USING "#,2A";Specifier$ !read in #8
80 ENTER XXX USING "#,8D";Blocklength !read in block length
90 ENTER XXX USING "-K";Block$ !read in data
100 END
```

Table 7-2. Card Identification Numbers

ID Number	Card
31	HP 16510A Logic Analyzer Card
1	HP 16515A 1 GHz Timing Master Card
2	HP 16516A I GHz Timing Expansion Card
21	HP 16520A Pattern Generator Master Card
22	HP 16521A Pattern Generator Expansion Card
11	HP 16530A Oscilloscope Timebase Card
12	HP 16531A Oscilloscope Acquisition Card

MMEMory Subsystem 8

Introduction

MMEMory (mass memory) subsystem commands provide access to both internal disc drives. Refer to figure 8-1 for the MMEMory Subsystem commands syntax diagram.

Note

<msus> refers to the mass storage unit specifier. INTernal1 specifies the front disc drive and INTernal0 specifies the rear disc drive.

If you are not going to store information to the configuration disc, or if the disc you are using contains information you need, it is advisable to write protect your disc. This will protect the contents of the disc from accidental damage due to incorrect commands, etc.

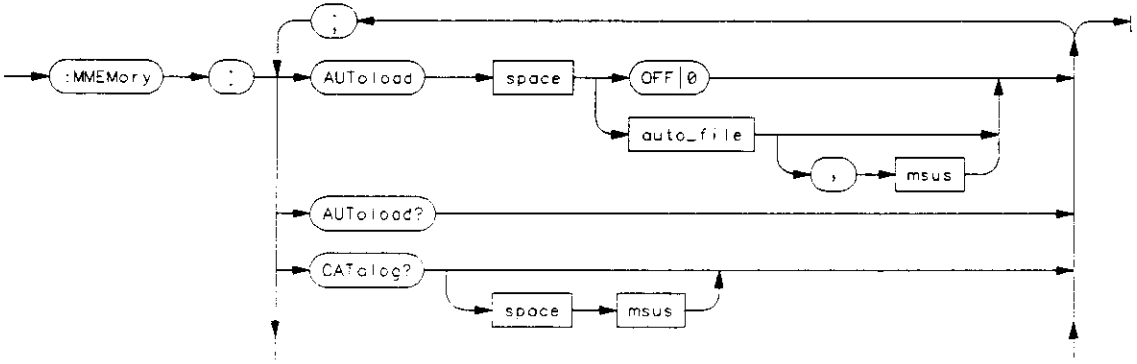


Figure 8-1. Mmemory Subsystem Commands Syntax Diagram

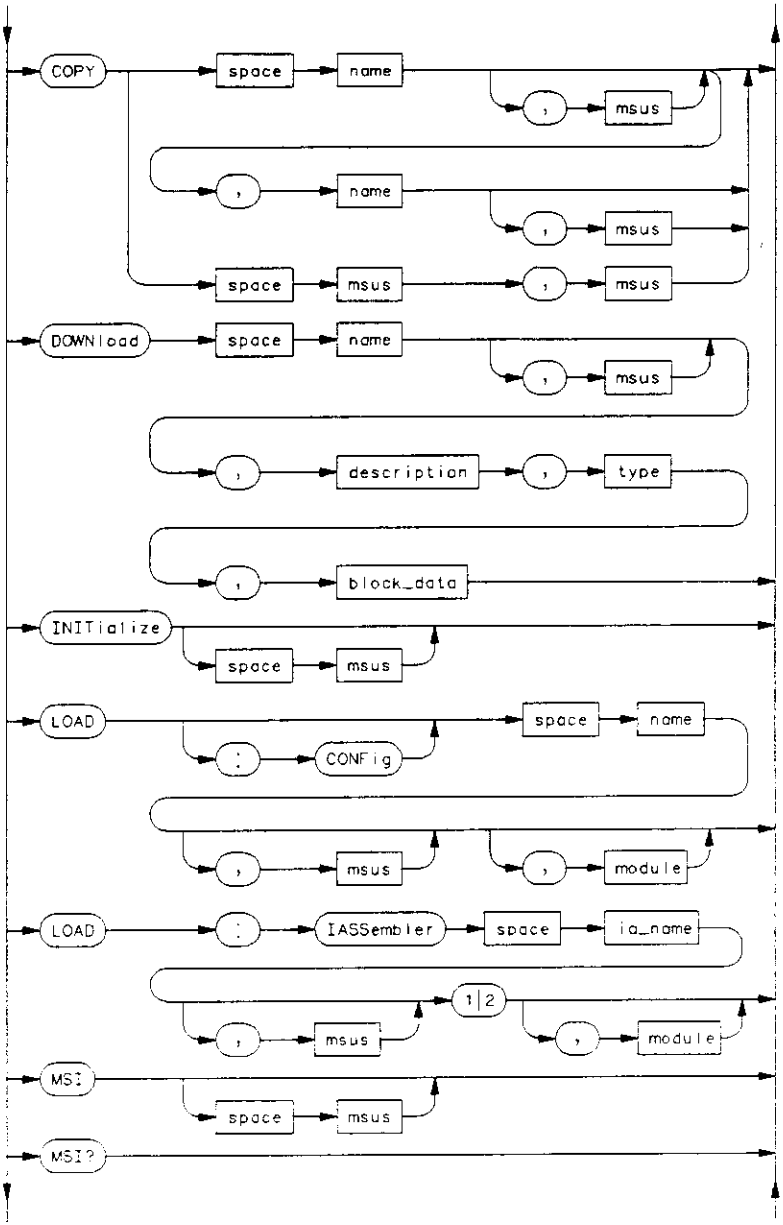
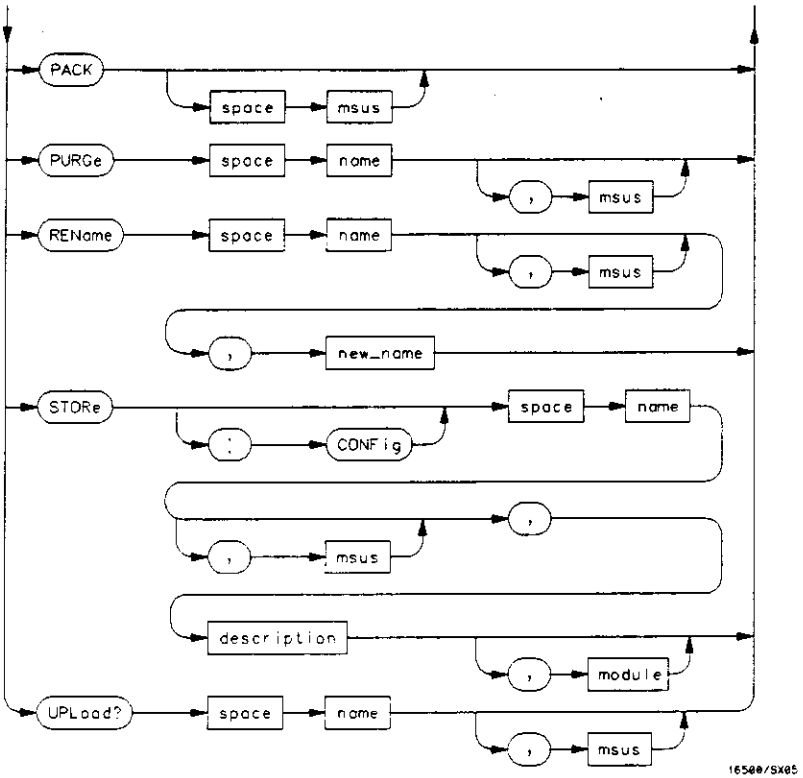


Figure 8-1. Mmemory Subsystem Commands Syntax Diagram (Continued)



auto_file = *string of up to 10 alphanun* ...*cters.*
msus = *INTernal1|INTernal0*
name = *string of up to 10 alphanumeric characters.*
description = *string of up to 32 alphanumeric characters.*
type = *integer, refer to table 8-1.*
block_data = *data in IEEE 488.2 # format.*
ia_name = *string of up to 10 alphanumeric characters.*
new_name = *string of up to 10 alphanumeric characters.*
module = *integer, 1 to 5.*

Figure 8-1. Mmemory Subsystem Commands Syntax Diagram (Continued)

AUToload command/query

The AUToload command controls the autoload feature which designates a set of configuration files to be loaded automatically the next time the instrument is turned on. The OFF parameter (or 0) disables the autoload feature. A string parameter may be specified instead to represent the desired autoload file. If the file is on the current <msus>, the autoload feature is enabled to the specified file.

The AUToload query returns 0 if the autoload feature is disabled. If the autoload feature is enabled, the query returns a string parameter that specifies the current autoload file. The appropriate slot designator "_X" is included in the filename. "X" refers to the slot designator A...E for the corresponding module. If the slot designator is "_?" the file is for all modules.

Command Syntax: :MMEMory:AUToload {{OFF|0}|{<auto_file>}}[,<msus>]

where:

<auto_file> ::= string of up to 10 alphanumeric characters
<msus> ::= {INTernal1|INTernal0}

Examples:
```
OUTPUT XXX;":MMEMORY:AUTOLOAD OFF"
OUTPUT XXX;":MMEMORY:AUTOLOAD 'FILE1_A'"
OUTPUT XXX;":MMEMORY:AUTOLOAD 'FILE2_ _',INTERNAL1"
```

Query Command: :MMEMory:AUToload?

Returned Format: [:MMEMory:AUToload] {0|<auto_file>},<msus><NL>

Example:
```
10 DIM Auto_status$[100]
20 OUTPUT XXX;":MMEMORY:AUTOLOAD?"
30 ENTER XXX;Auto_status$
40 PRINT Auto_status$
50 END
```

CATalog query

The CATalog query returns the directory of the currently selected disc in block data format. The directory consists of a 51 character string for each file on the disc. Each file entry is formatted as follows:

"NNNNNNNNNN TTTTTTT DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD"

where N is the filename, T is the file type (a number), and D is the file descriptor. If the <msus> is not specified, the last disc drive specified by the MSI command will be used.

For more information on block data format, refer to the section "Definite-Length Block Response Data" in the chapter 1.

Query Syntax: :MMEMory:CATalog? [<msus>]

where:

<msus> ::= {INTernal1|INTernal0}

Returned Format: [:MMEMory:CATalog] <block data>

where:

<block data> ::= <filename> <file type> <file description>...

Example:

```
10 DIM File$[51]
20 DIM Specifier$[2]
30 OUTPUT XXX;":SYSTEM:HEAD OFF"
40 OUTPUT XXX;":MMEMORY:CATALOG? INTERNAL1"  !send catalog query
50 ENTER XXX USING "#,2A";Specifier$  !read in #8
60 ENTER XXX USING "#,8D";Length  !read in length
70 FOR I = 1 TO Length STEP 51  !read and print each file in the directory
80   ENTER XXX USING "#,51A";File$
90   PRINT File$
100 NEXT I
110 ENTER XXX USING "A";Specifier$  !read in final line feed
120 END
```

COPY command

The COPY command copies one file to a new file or an entire disc's contents to another disc. The two < name > parameters are the filenames. The first pair of parameters specifies the source file. The second pair specifies the destination file. An error is generated if the source file doesn't exist, or if the destination file already exists.

If the filename is not specified for both the source and destination, then the entire contents of the source are transferred to the destination (same as the Duplicate Disc front panel operation). The previous contents of the destination media are destroyed.

If the < msus > is not specified, the last disc drive specified by the MSI command will be used.

Command Syntax: :MMEMory:COPY [< name >][, < msus >],[< name >][, < msus >]

where:

< name > :: = string of up to 10 alphanumeric characters
< msus > :: = {INTernal1 | INTernal0}

Examples: To copy the contents of "FILE1" to "FILE2" when both files are on a disc on the last disc drive specified by the MSI command:

```
OUTPUT XXX;":MMEMORY:COPY 'FILE1','FILE2'"
```

To copy the contents of "FILE1" on the rear disc drive to "FILE2" on the front disc drive:

```
OUTPUT XXX;":MMEM:COPY 'FILE1',INTERNAL0,'FILE2',INTERNAL1"
```

To copy the contents of "FILE1" on the last disc drive specified by the MSI command to "FILE2" on the rear disc drive:

```
OUTPUT XXX;":MMEMORY:COPY 'FILE1','FILE2',INTERNAL0"
```

To duplicate the contents of the disc in the rear disc drive to a disc in the front disc drive:

```
OUTPUT XXX;":MMEM:COPY INTERNAL0,INTERNAL1"
```

DOWNload command

The DOWNload command downloads a file to the specified mass storage device. The <name> parameter specifies the filename, the <description> parameter specifies the file descriptor, and the <block_data> contains the contents of the file to be downloaded.

If the <msus> is not specified, the last disc drive specified by the MSI command will be used.

Table 8-1 lists the file types for the <type> parameter.

Command Syntax: :MMEMory:DOWNload <name>[,<msus>],<description>,<type>,<block_data>

where:

<name> ::= string of up to 10 alphanumeric characters
<msus> ::= {INTernal0|INTernal1}
<description> ::= string of up to 32 alphanumeric characters
<type> ::= integer (see table 8-1)
<block_data> ::= contents of file in block data format

Example:
```
OUTPUT XXX;":MMEMORY:DOWNLOAD 'SETUP__',INTERNAL0,'FILE CREATED
FROM SETUP QUERY',-16127,#800000643..."
```

Table 8-1. File Types

File	File Type
HP 16500A System Software	-16128
HP 16500A Mainframe (System) Configuration	-16127
HP 16510A Logic Analyzer Configuration	-16096
HP 16515A 1 GHz Timing Configuration	-16126
HP 16520A Pattern Generator Configuration	-16106
HP 16530A Oscilloscope Configuration	-16116
Autoload File	-15615
Inverse Assembler	-15614
Option Software	-15613
Calibration Factors	-15611
Text (Generic) Type	-15610

INITialize command

The INITialize command formats the specified disc. If no disc drive is specified, then the initialize command will format the disc in the currently selected disc drive.

Command Syntax: :MMEMory:INITialize [<msus>]

where:

<msus> ::= {INTernal0|INTernal1}

Examples:
```
OUTPUT XXX;":MMEMORY:INITIALIZE"
OUTPUT XXX;":MMEMORY:INITIALIZE INTERNAL0"
```

Note

Once executed, the initialize command formats the specified disc, permanently erasing all existing information from the disc. After that, there is no way to retrieve the original information.

LOAD [:CONFig] command

The LOAD command loads a configuration file from the disc into modules, software options, or the system. The <name> parameter specifies the filename from the specified mass storage device. The optional <module> parameter specifies which module(s) to load the file into. The accepted values are 0 for system, 1...5 for the module in slot A...E respectively, and -1...-2 for software options 1 and 2 respectively. Not specifying the <module> parameter is equivalent to performing a 'LOAD ALL' from the front panel which loads the appropriate file for every installed module, software option, and for the system.

Command Syntax: :MMEMory:LOAD[:CONfig] <name>[,<msus>][,<module>]

where:

<name> ::= string of up to 10 alphanumeric characters
<msus> ::= {INTernal0|INTernal1}
<module> ::= -2 through 5 (integer)

Examples:
```
OUTPUT XXX;":MMEMORY:LOAD:CONFIG 'FILE__'"
OUTPUT XXX;":MMEMORY:LOAD 'FILE__',0"
OUTPUT XXX;":MMEM:LOAD:CONFIG 'FILE_A',INTERNAL1,1"
```

LOAD :IASSembler command

This variation of the LOAD command allows inverse assembler files to be loaded into a module that performs state analysis. The <IA_name> parameter specifies the inverse assembler filename from the desired <msus>. The parameter after the optional <msus> specifies which machine to load the inverse assembler into.

The optional <module> parameter is used to specify which slot the state analysis module is in. 1...5 refers to the module in slot A...E. If this parameter is not specified, the state analysis module closest to slot A is selected.

Command Syntax: :MMEMory:LOAD:IASSembler <IA_name>[,<msus>],{1|2}[,<module>]

where:

<IA_name> ::= string of up to 10 alphanumeric characters
<msus> ::= {INTernal0|INTernal1}
<module> ::= 1 through 5 (integer)

Examples:
```
OUTPUT XXX;":MMEMORY:LOAD:IASSEMBLER 'I68020_IP',1"
OUTPUT XXX;":MMEM:LOAD:IASS 'I68020_IP',INTERNAL1,1,2"
```

MSI (Mass Storage Is) command/query

The MSI command selects a default mass storage device. If the parameter is omitted, the default mass storage device (front disc drive) is selected. The query returns the current MSI setting.

Command Syntax: :MMEMory:MSI [<msus>]

where:

<msus> ::= {INTernal0|INTernal1}

Examples:
```
OUTPUT XXX;":MMEMORY:MSI"
OUTPUT XXX;":MMEM:MSI INTERNAL1"
```

Query Syntax: :MMEMory:MSI?

Returned Format: [:MMEMory:MSI] <msus><NL>

Example:
```
10 DIM Device$[100]
20 OUTPUT XXX;":MMEMORY:MSI?"
30 ENTER XXX;Device$
40 PRINT Device$
50 END
```

PACK command

The PACK command packs the files on a disc on the specified mass storage device. If no disc drive is specified, this command will pack the disc in the last disc drive specified by the MSI command.

Command Syntax: :MMEMory:PACK [<msus>]

where:

<msus> ::= {INTernal0|INTernal1}

Examples:
```
OUTPUT XXX;":MMEMORY:PACK"
OUTPUT XXX;":MMEM:PACK INTERNAL1"
```

PURGe command

The PURGe command deletes a file from the specified mass storage device. The <name> parameter specifies the filename to be deleted.

Command Syntax: :MMEMory:PURGe <name>[,<msus>]

where:

<name> ::= string of up to 10 alphanumeric characters
<msus> ::= {INTernal0|INTernal1}

Examples:
```
OUTPUT XXX;":MMEMORY:PURGE 'FILE1'"
OUTPUT XXX;":MMEM:PURG 'FILE1',INTERNAL0"
```

Note

Once executed, the purge command permanently erases all the existing information from the specified file. After that, there is no way to retrieve the original information.

RENname command

The RENname command renames a file from the specified mass storage device. The <name> parameter specifies the filename to be changed and the <new_name> parameter specifies the new filename.

Note

You cannot rename a file to an already existing filename.

Command Syntax: :MMEMory:RENname <name>[,<msus>],<new_name>

where:

<name> ::= string of up to 10 alphanumeric characters
<msus> ::= {INTernal0|INTernal1}
<new_name> ::= string of up to 10 alphanumeric characters

Examples:
```
OUTPUT XXX;":MMEMORY:RENAME 'OLDFILE','NEWFILE'"
OUTPUT XXX;":MMEM:REN 'OLDFILE',INTERNAL1,'NEWFILE'"
```

STORe [:CONFig] command

The STORe command stores module or system configurations onto a disc. The [:CONFig] specifier is optional and has no effect on the command. The <name> parameter specifies the file on the specified mass storage device. The <description> parameter describes the contents of the file. The optional <module> parameter allows you to store the configuration for a specific module or modules. 1...5 refers to the module in slot A...E and 0 refers to the system. -1 and -2 are for software options.

If the optional <module> parameter is not specified, the configurations for all of the modules are stored.

Command Syntax: :MMEMory:STORe [:CONfig]<name>[,<msus>],<description>[,<module>]

where:

<name> ::= string of up to 10 alphanumeric characters
<msus> ::= {INTernal0|INTernal1}
<description> ::= string of up to 32 alphanumeric characters
<module> ::= -2 through 5 (integer)

Examples:
```
OUTPUT XXX;":MMEM:STOR 'DEFAULTS','SETUPS FOR ALL MODULES"
OUTPUT XXX;":MMEMORY:STORE:CONFIG 'SCOPE',INTERNAL1,'SLOT B SCOPE
CONFIG',2"
```

Note

The appropriate slot designator "_X" is added to all files when they are stored. "X" refers to the letter A...E of the corresponding slot for each module.

UPLoad

query

The UPLoad query uploads a file. The <name> parameter specifies the file to be uploaded from the specified mass storage device. The contents of the file are sent out of the instrument in block data form.

Note

This command should only be used for HP 165XX or HP 165X files.

Query Syntax: :MMEMory:UPLoad? <name>[,<msus>]

where:

<name> ::= string of up to 10 alphanumeric characters
<msus> ::= {INTernal0|INTernal1}

Returned Format: [:MMEMory:UPLoad] <block_data><NL>

HP-IB Example:

```
10 DIM Block$[32000]  !allocate enough memory for block data
20 DIM Specifier$[2]
30 OUTPUT XXX;":EOI ON"
40 OUTPUT XXX;":SYSTEM HEAD OFF"
50 OUTPUT XXX;":MMEMORY:UPLOAD? 'FILE1',INTERNAL1"  !send upload query
60 ENTER XXX USING "#,2A";Specifier$  !read in #8
70 ENTER XXX USING "#,8D";Length  !read in block length
80 ENTER XXX USING "-K";Block$  !read in file
90 END
```

INTermodule Subsystem 9

Introduction

INTermodule subsystem commands specify intermodule arming between multiple modules. Refer to figure 9-1 for the INTermodule Subsystem commands syntax diagram.

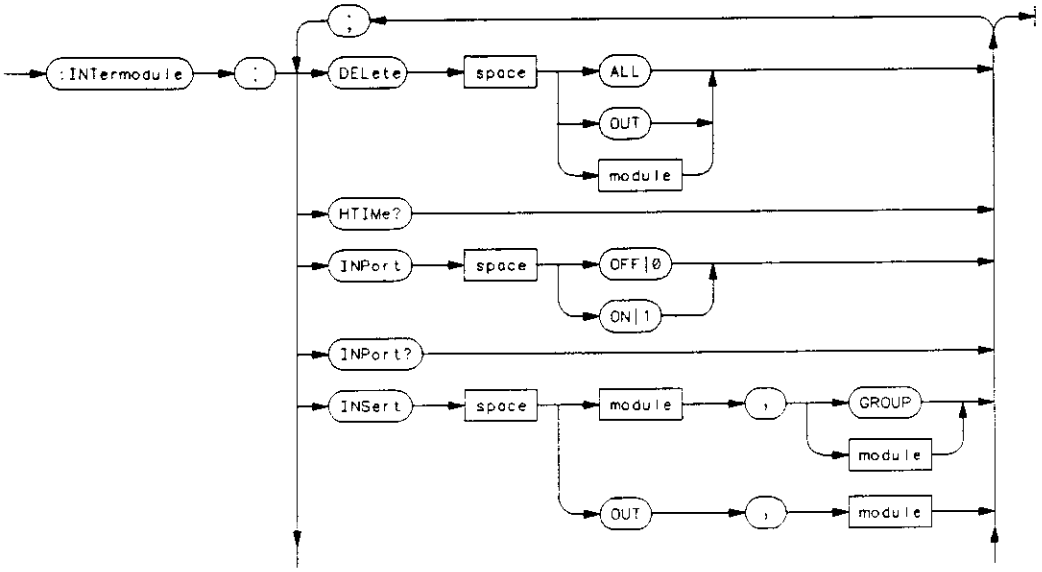
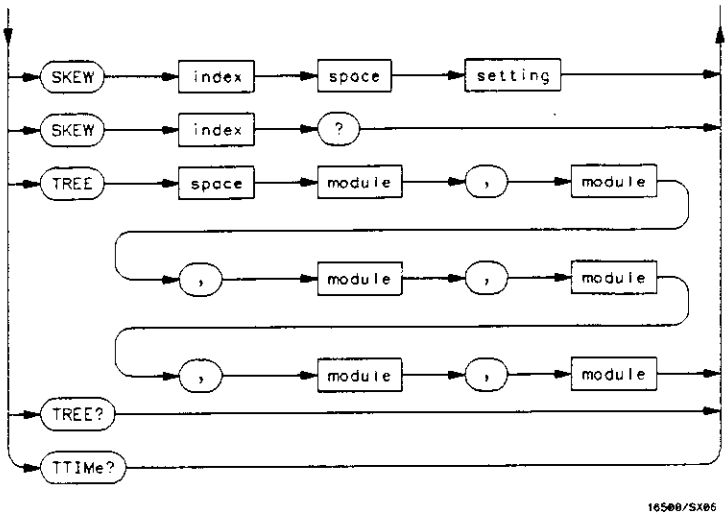


Figure 9-1. Intermodule Subsystem Commands Syntax Diagram



module = *integer, 1 to 5.*
index = *integer, 1 to 5.*
setting = *numeric, -1.0 to 1.0 in seconds.*

Figure 9-1. Intermodule Subsystem Commands Syntax Diagram (Continued)

DELete command

The DELete command is used to delete a module, group of modules, PORT OUT, or an entire intermodule tree. The <module> parameter sent with the delete command refers to the slot location of the module (1...5 corresponds to the module in slot A...E).

Command Syntax: :INTermodule:DELete {ALL|OUT|<module>}

where:

<module> ::= 1 through 5 (integer)

Example:
```
OUTPUT XXX;":INTERMODULE:DELETE ALL"
OUTPUT XXX;":INT:DEL 2"
```

HTIMe query

The HTIMe query returns five values representing the internal hardware skew for all of the modules in the Intermodule configuration. The first value is the skew for the module in slot A, the second value is for the module in slot B, the third value is for slot C, etc. If there is no internal skew, 9.9E37 is returned.

Note

The internal hardware skew is only a display adjustment for time correlated waveforms. The values returned are the average propagation delays of the trigger lines through the intermodule bus circuitry for each module. These values are for reference only because the values returned by TTIMe include the internal hardware skew represented by HTIMe.

Query Syntax: :INTermodule:HTIMe?

Returned Format: [:INTermodule:HTIMe]
<value_1>,<value_2>,<value_3>,<value_4>,<value_5><NL>

where:

<value_1> ::= skew for module in slot A (real number)
<value_2> ::= skew for module in slot B (real number)
<value_3> ::= skew for module in slot C (real number)
<value_4> ::= skew for module in slot D (real number)
<value_5> ::= skew for module in slot E (real number)

Examples:

```
10 DIM Setting$[100]
20 OUTPUT XXX;":INTERMODULE:HTIME?"
30 ENTER XXX;Setting$
40 PRINT Setting$
50 END
```

INPort

command/query

The INPort command causes Intermodule runs to be armed from the Input port. The INPort query returns the current setting.

Command Syntax: :INTermodule:INPort {{ON|1}|{OFF|0}}

Example: OUTPUT XXX;":INTERMODULE:INPORT ON"

Query Syntax: :INTermodule:INPort?

Returned Format: [:INTermodule:INPort] {1|0}<NL>

Examples:

```
10 DIM Setting$[100]
20 OUTPUT XXX;":INTERMODULE:INPORT?"
30 ENTER XXX;Setting$
40 PRINT Setting$
50 END
```

INSert command

The INSert command adds a module or PORT OUT to the Intermodule configuration. The first parameter selects the module or PORT OUT to be added to the intermodule configuration, and the second parameter tells the instrument where the module or PORT OUT will be located. 1...5 corresponds to the slot location of the module A...E.

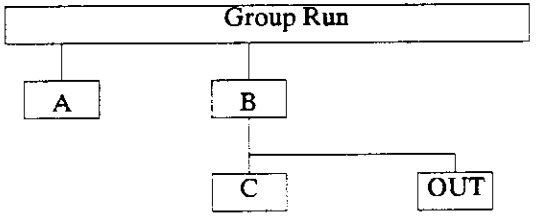
Command Syntax: :INTermodule:INSert {<module>|OUT},{GROUP|<module>}

where:

<module> ::= 1 through 5 (integer)

Examples:
```
OUTPUT XXX;":INTERMODULE:INSERT 1,GROUP"
OUTPUT XXX;":INT:INSERT 2,GROUP"
OUTPUT XXX;":INTERMODULE:INS 3,2;INS OUT,2"
```

The following figure shows the result of the example output commands:



SKEW < N > command/query

The SKEW command sets the skew value for a module. The query returns the user defined skew setting. The < N > index value is the module number (1...5 corresponds to modules A...E respectively) and the < setting > parameter is the skew setting (-1.0 to 1.0) in seconds.

Command Syntax: :INTermodule:SKEW< N > < setting >

where:

< N > ::= 1 through 5 (integer)
< setting > ::= -1.0 to 1.0 seconds (real number)

Example:
```
OUTPUT XXX;":INTERMODULE:SKEW2 3.0E-9"
```

Query Syntax: :INTermodule:SKEW< N >?

Returned Format: [INTermodule:SKEW< N >] < setting > < NL >

Example:
```
10 DIM Setting$[100]
20 OUTPUT XXX;":INTERMODULE:SKEW2?"
30 ENTER XXX;Setting$
40 PRINT Setting$
50 END
```

TREE command/query

The TREE command allows an intermodule setup to be specified in one command. The query returns a string that represents the intermodule tree. A -1 means the module is not in the intermodule tree, a 0 value means the module is armed from the Intermodule run button (Group run), and a positive value indicates the module is being armed by another module with that slot location (1...5 corresponds to the module in slot A...E, respectively).

The first five parameters are the intermodule arm values for modules A through E respectively. The last parameter corresponds to the intermodule arm value for PORT OUT.

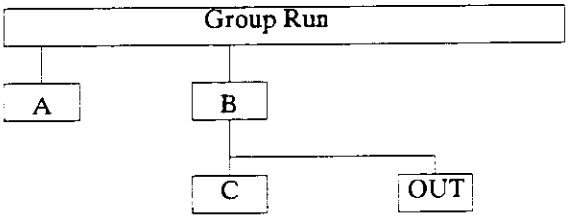
Command Syntax: :INTermodule:TREE
<module>,<module>,<module>,<module>,<module>,<module>

where:

<module> ::= -1 through 5 (integer)

Example: OUTPUT XXX;":INTERMODULE:TREE 0,0,2,-1,-1,2"

The following figure shows the result of the example output commands:



Query Syntax: :INTermodule:TREE?

Returned Format: [INTermodule:TREE]
<module>,<module>,<module>,<module>,<module>,<module><NL>

Example:
```
10 DIM Config$[100]
20 OUTPUT XXX;":INTERMODULE:TREE?"
30 ENTER XXX;Config$
40 PRINT Config$
50 END
```

TTIMe query

The TTIMe query returns five values representing the absolute intermodule trigger times for all of the modules in the Intermodule configuration. The first value is the trigger time for the module in slot A, the second value is for the module in slot B, the third value is for slot C, etc.

The value 9.9E37 is returned when:

- No module is installed in the corresponding slot;
- The module in the corresponding slot is not time correlated; or
- A time correlatable module did not trigger.

Note

The trigger times returned by this command have already been offset by the INTermodule:SKEW values and internal hardware skews (INTermodule:HTIMe).

Query Syntax: :INTermodule:TTIMe?

Returned Format: [:INTermodule:TTIMe]
<value_1>,<value_2>,<value_3>,<value_4>,<value_5><NL>

where:

<value_1> ::= trigger time for module in slot A (real number)
<value_2> ::= trigger time for module in slot B (real number)
<value_3> ::= trigger time for module in slot C (real number)
<value_4> ::= trigger time for module in slot D (real number)
<value_5> ::= trigger time for module in slot E (real number)

Examples:

```
10 DIM Setting$[100]
20 OUTPUT XXX;":INTERMODULE:TTIME?"
30 ENTER XXX;Setting$
40 PRINT Setting$
50 END
```

Message Communication and System Functions

A

Introduction

This appendix describes the operation of instruments that operate in compliance with the IEEE 488.2 (syntax) standard. The IEEE 488.2 standard is a new standard. Instruments that are compatible with IEEE 488.2 must also be compatible with IEEE 488.1 (HP-IB bus standard); however, IEEE 488.1 compatible instruments may or may not conform to the IEEE 488.2 standard. The IEEE 488.2 standard defines the message exchange protocols by which the instrument and the controller will communicate. It also defines some common capabilities, which are found in all IEEE 488.2 instruments. This appendix also contains a few items which are not specifically defined by IEEE 488.2, but deal with message communication or system functions.

Note

The syntax and protocol for RS-232C program messages and response messages for the HP 16500A are structured very similiar to those described by 488.2. In most cases, the same structure shown in this appendix for 488.2 will also work for RS-232C. Because of this, no additional information has been included for RS-232C.

Also, the common commands listed in the chapter "Common Commands" may be sent over both HP-IB and RS-232C.

Protocols

The protocols of IEEE 488.2 define the overall scheme used by the controller and the instrument to communicate. This includes defining when it is appropriate for devices to talk or listen, and what happens when the protocol is not followed.

Functional Elements

Before proceeding with the description of the protocol, a few system components should be understood.

Input Buffer. The input buffer of the instrument is the memory area where commands and queries are stored prior to being parsed and executed. It allows a controller to send a string of commands to the instrument which could take some time to execute, and then proceed to talk to another instrument while the first instrument is parsing and executing commands.

Output Queue. The output queue of the instrument is the memory area where all output data (< response messages >) are stored until read by the controller.

Parser. The instrument's parser is the component that interprets the commands sent to the instrument and decides what actions should be taken. "Parsing" refers to the action taken by the parser to achieve this goal. Parsing and executing of commands begins when either the instrument sees a < program message terminator > (defined later in this appendix) or the input buffer becomes full. If you wish to send a long sequence of commands to be executed and then talk to another instrument while they are executing, you should send all the commands before sending the < program message terminator > .

Protocol Overview

The instrument and controller communicate using < program message > s and < response message > s. These messages serve as the containers into which sets of program commands or instrument responses are placed. < program message > s are sent by the controller to the instrument, and < response message > s are sent from the instrument to the controller in response to a query message. A < query message > is defined as being a < program message > which contains one or more queries. The instrument will only talk when it has received a valid query message, and therefore has something to say. The controller should only attempt to read a response after sending a complete query message, but before sending another < program message >. The basic rule to remember is that the instrument will only talk when prompted to, and it then expects to talk before being told to do something else.

Protocol Operation

When the instrument is turned on or when it receives a device clear command, the input buffer and output queue are cleared, and the parser is reset to the root level of the command tree.

Note

When the instrument receives a device clear command, the module (or system) selected prior to the command remains selected.

The instrument and the controller communicate by exchanging complete < program message > s and < response message > s. This means that the controller should always terminate a < program message > before attempting to read a response. The instrument will terminate < response message > s except during a hardcopy output.

If a query message is sent, the next message passing over the bus should be the < response message >. The controller should always read the complete < response message > associated with a query message before sending another < program message > to the same instrument.

The instrument allows the controller to send multiple queries in one query message. This is referred to as sending a "compound query." As will be noted later in this appendix, multiple queries in a query message are separated by semicolons. The responses to each of the queries in a compound query will also be separated by semicolons.

Commands are executed in the order they are received. This also applies to the reception of the HP-IB group execute trigger (GET) bus command. The group execute trigger command should not be sent in the middle of a < program message >.

Protocol Exceptions

If an error occurs during the information exchange, the exchange may not be completed in a normal manner. Some of the protocol exceptions are shown below.

Addressed to talk with nothing to say. If the instrument is addressed to talk before it receives a query, it will indicate a query error and will not send any bytes over the bus. If the instrument has nothing to say because queries requested were unable to be executed because of some error, the device will not indicate a query error, but will simply wait to receive the next message from the controller.

Addressed to talk with no listeners on the bus. If the instrument is addressed to talk and there are no listeners on the bus, the instrument will wait for a listener to listen, or for the controller to take control.

Command Error. A command error will be reported if the instrument detects a syntax error or an unrecognized command header. An HP-IB group execute trigger (GET) sent in the middle of a < program message > will also cause a command error.

Execution Error. An execution error will be reported if a parameter is found to be out of range, or if the current settings do not allow execution of a requested command or query.

Device-specific Error. A device-specific error will be reported if the instrument is unable to execute a command for a strictly device dependent reason.

Query Error. A query error will be reported if the proper protocol for reading a query is not followed. This includes the interrupted and unterminated conditions described in the following paragraphs.

Unterminated Condition. If the controller attempts to read a < response message > before terminating the < program message >, a query error will be generated. The parser will reset itself, and the response will be cleared from the output queue of the instrument without being sent over the bus.

Interrupted Condition. If the controller does not read the entire < response message > generated by a query message and then attempts to send another < program message >, the device will generate a query error. The unread portion of the response will then be discarded by the instrument. The interrupting < program message > will not be affected.

Buffer Deadlock. The instrument may become deadlocked if the input buffer and output queue both become full. This condition can occur if a very long < program message > is sent containing queries that generate a great deal of response data. The instrument cannot accept any more bytes, and the controller cannot read any of the response data until it has completed sending the entire < program message >. Under this condition the instrument will break the deadlock by clearing the output queue, and continuing to discard responses until it comes to the end of the current < program message >. The query error bit will also be set.

Syntax Diagrams

The syntax diagrams in this appendix are similar to the syntax diagrams in the IEEE 488.2 specification. Commands and queries are sent to the instrument as a sequence of data bytes. The allowable byte sequence for each functional element is defined by the syntax diagram that is shown with the element description.

The allowable byte sequence can be determined by following a path in the syntax diagram. The proper path through the syntax diagram is any path that follows the direction of the arrows. If there is a path around an element, that element is optional. If there is a path from right to left around one or more elements, that element or those elements may be repeated as many times as desired.

Syntax Overview

This overview is intended to give a quick glance at the syntax defined by IEEE 488.2. It should allow you to understand many of the things about the syntax you need to know. This appendix also contains the details of the IEEE 488.2 defined syntax.

IEEE 488.2 defines the blocks used to build messages which are sent to the instrument. A whole string of commands can therefore be broken up into individual components.

Figure A-1 shows a breakdown of an example <program message>. There are a few key items to notice:

1. A semicolon separates commands from one another. Each <program message unit> serves as a container for one command. The <program message unit>s are separated by a semicolon.

2. A <program message> is terminated by a <NL> (new line), a <NL> with EOI asserted, or EOI being asserted on the last byte of the message. The recognition of the <program message terminator>, or <PMT>, by the parser serves as a signal for the parser to begin execution of commands. The <PMT> also affects command tree traversal (see the Programming and Documentation Conventions chapter).

3. Multiple data parameters are separated by a comma.

4. The first data parameter is separated from the header with one or more spaces.

5. The header INTERMODULE:INSERT is an example of a compound header. It places the parser in the intermodule subsystem until the <NL> is encountered.

6. A colon preceding the command header returns you to the top of the command tree for the selected module.

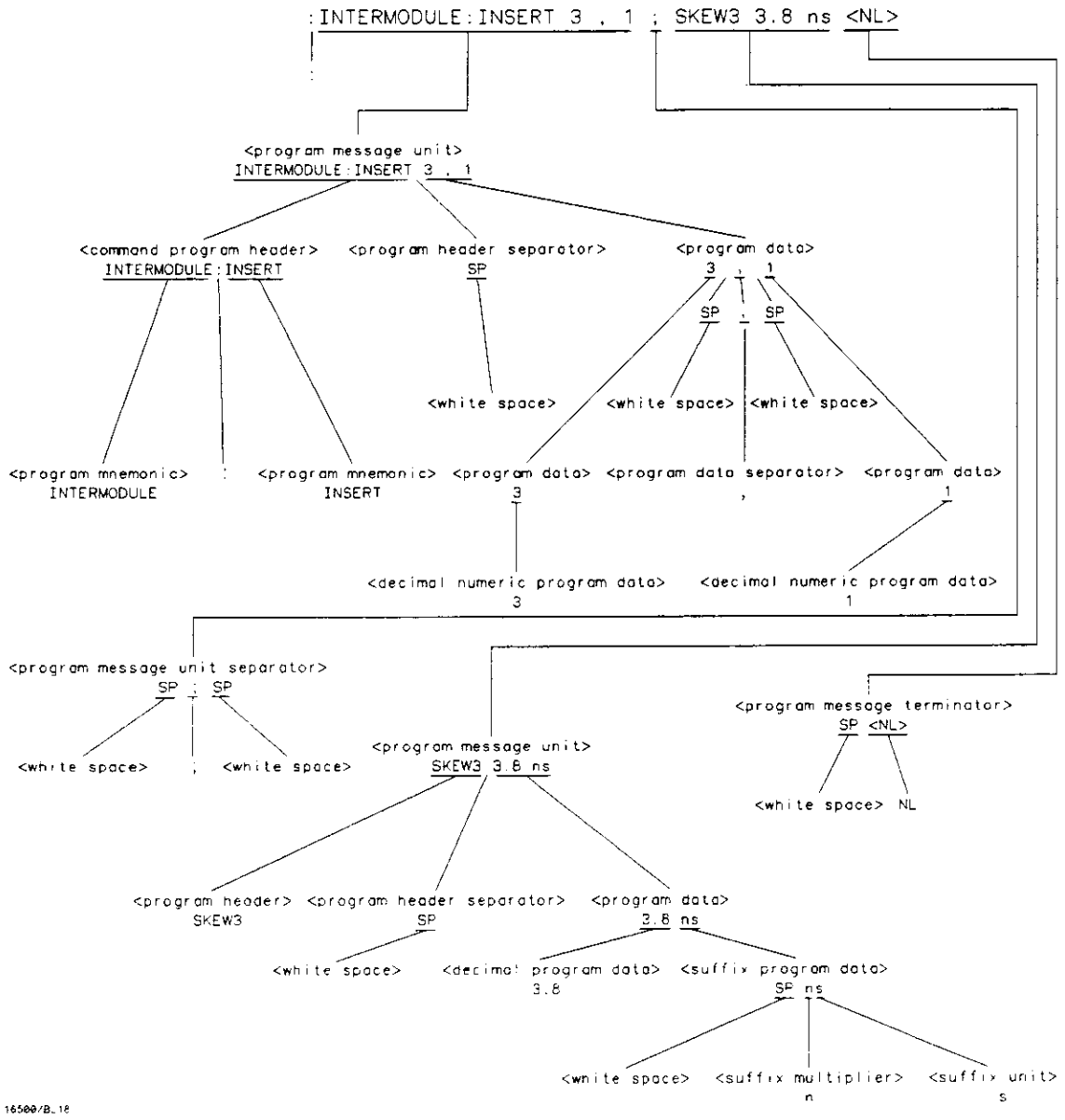


Figure A-1. <program message> Parse Tree

Device Listening Syntax

The listening syntax of IEEE 488.2 is designed to be more forgiving than the talking syntax. This allows greater flexibility in writing programs, as well as allowing them to be easier to read.

Upper/Lower Case Equivalence. Upper and lower case letters are equivalent. The mnemonic SINGLE has the same semantic meaning as the mnemonic single.

< white space >. < white space > is defined to be one or more characters from the ASCII set of 0 - 32 decimal, excluding 10 decimal (NL). < white space > is used by several instrument listening components of the syntax. It is usually optional, and can be used to increase the readability of a program.

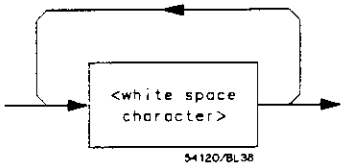


Figure A-2. < white space >

< program message >. The < program message > is a complete message to be sent to the instrument. The instrument will begin executing commands once it has a complete < program message >, or when the input buffer becomes full. The parser is also repositioned to the root of the command tree after executing a complete < program message >. Refer to the Tree Traversal Rules in the Programming and Documentation Conventions chapter for more details.

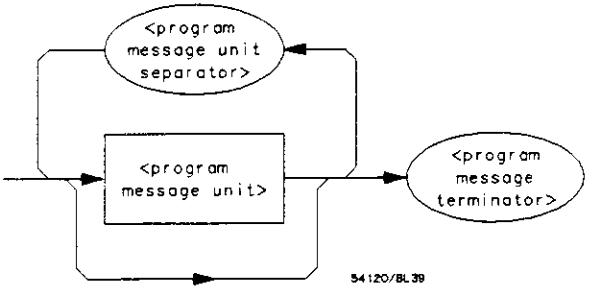


Figure A-3. < program message >

< program message unit >. The < program message unit > is the container for individual commands within a < program message >.

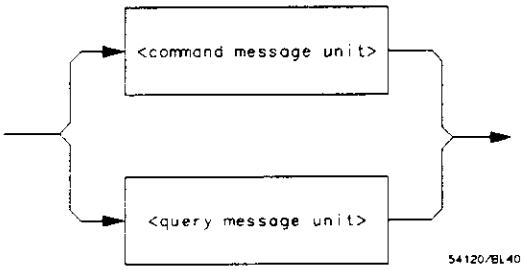


Figure A-4. < program message unit >

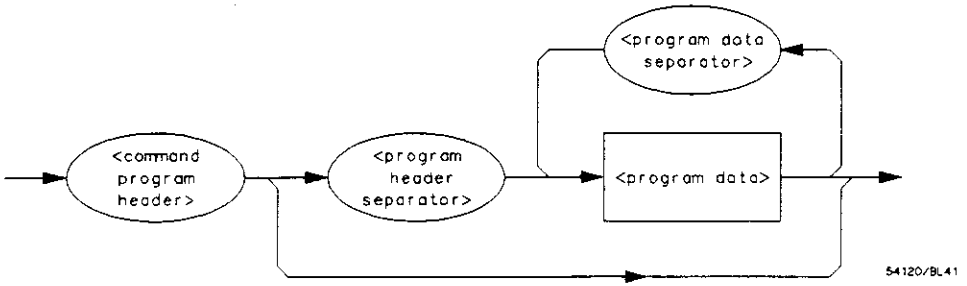


Figure A-5. <command message unit>

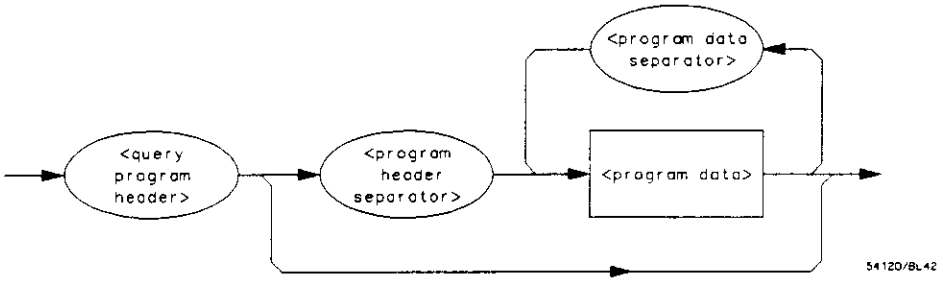


Figure A-6. <query message unit>

< program message unit separator >. A semicolon separates < program message unit > s, or individual commands.

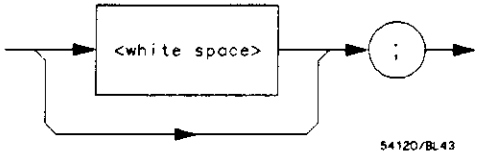


Figure A-7. < program message unit separator >

< command program header > / < query program header >. These elements serve as the headers of commands or queries. They represent the action to be taken.

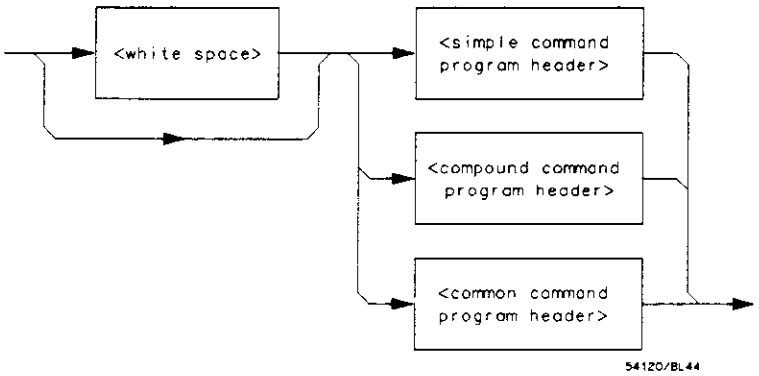
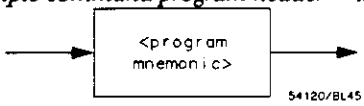
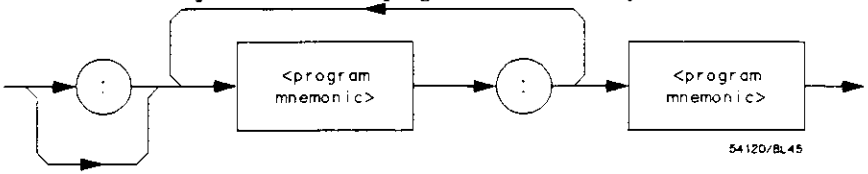


Figure A-8. < command program header >

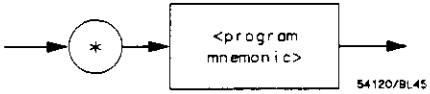
Where <simple command program header> is defined as



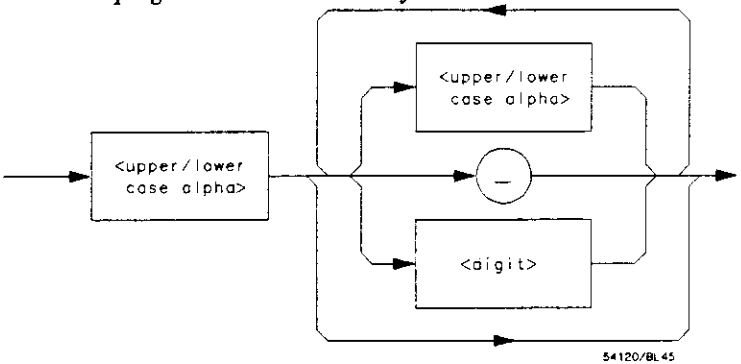
Where <compound command program header> is defined as



Where <common command program header> is defined as



Where <program mnemonic> is defined as

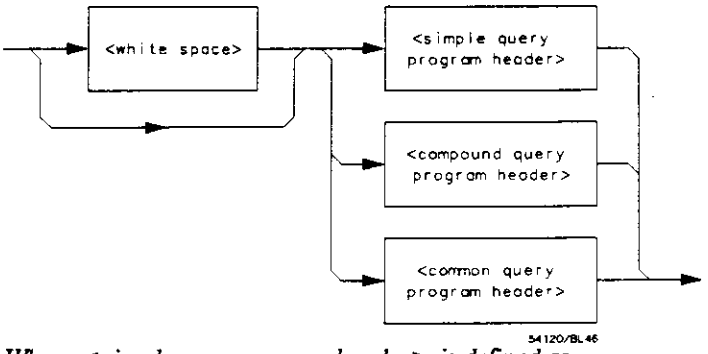


Where <upper/lower case alpha> is defined as a single ASCII encoded byte in the range 41 - 5A, 61 - 7A (65 - 90, 97 - 122 decimal).

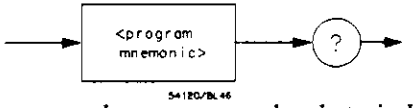
Where <digit> is defined as a single ASCII encoded byte in the range 30 - 39 (48 - 57 decimal).

Where (_) represents an "underscore", a single ASCII-encoded byte with the value 5F (95 decimal).

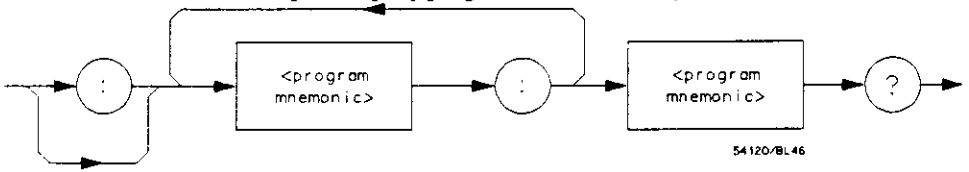
Figure A-8. <command program header> (continued)



Where <simple query program header> is defined as



Where <compound query program header> is defined as



Where <common query program header> is defined as

Figure A-9. <query program header>

< program data >. The < program data > element represents the possible types of data which may be sent to the instrument. The HP 16500A will accept the following data types: < character program data >, < decimal numeric program data >, < suffix program data >, < string program data >, and < arbitrary block program data >.

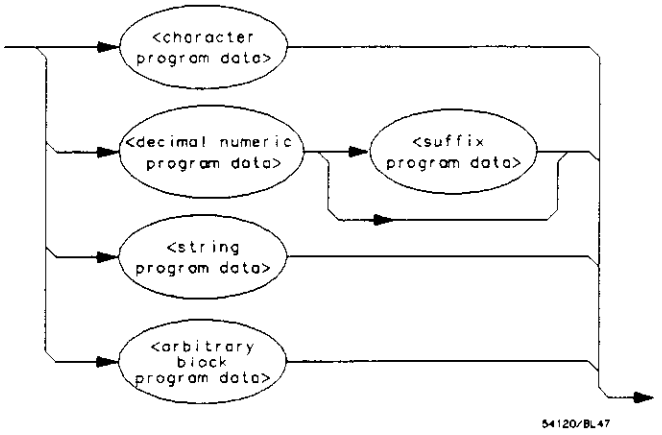


Figure A-10. < program data >

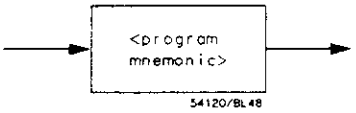
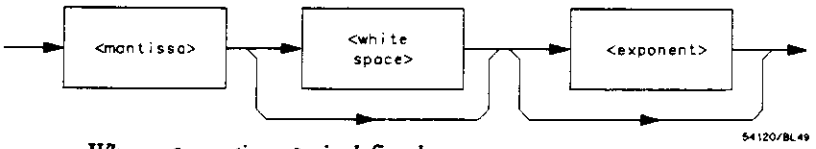
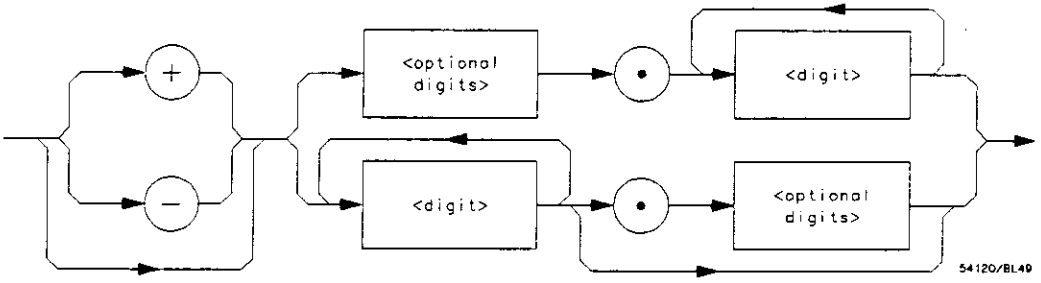


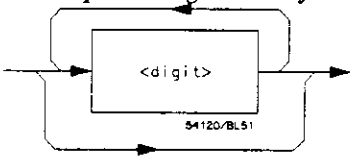
Figure A-11. < character program data >



Where <mantisssa> is defined as



Where <optional digits> is defined as



Where <exponent> is defined as

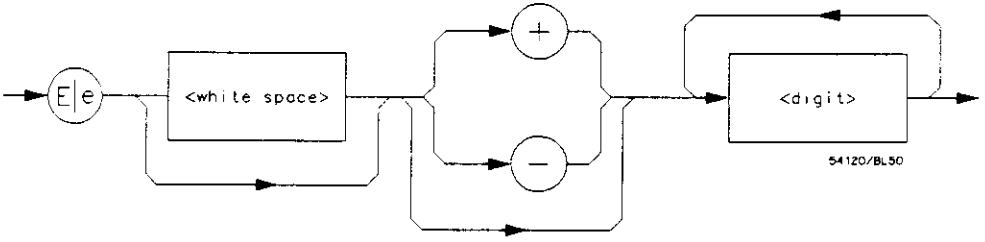


Figure A-12. <decimal numeric program data>

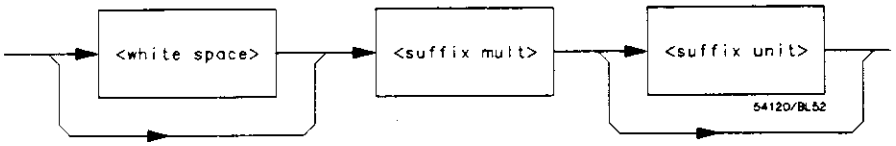


Figure A-13. <suffix program data>

Suffix Multiplier. The suffix multipliers that the instrument will accept are shown in table A-1.

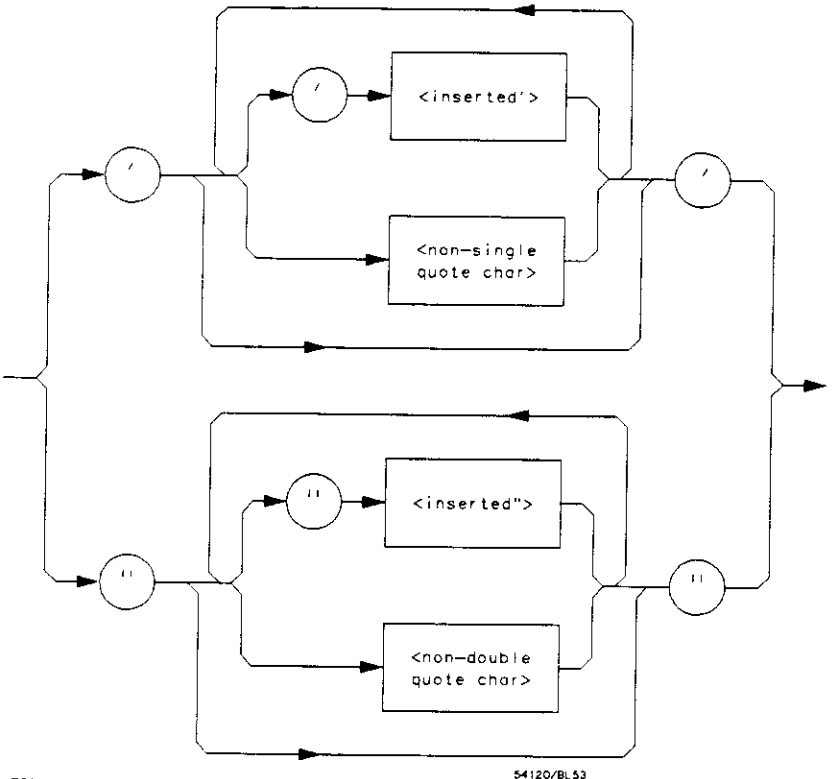
Table A-1. <suffix mult>

Value	Mnemonic
1E18	EX
1E15	PE
1E12	T
1E9	G
1E6	MA
1E3	K
1E-3	M
1E-6	U
1E-9	N
1E-12	P
1E-15	F
1E-18	A

Suffix Unit. The suffix units that the instrument will accept are shown in table A-2.

Table A-2. <suffix unit>

Suffix	Referenced Unit
V	Volt
S	Second



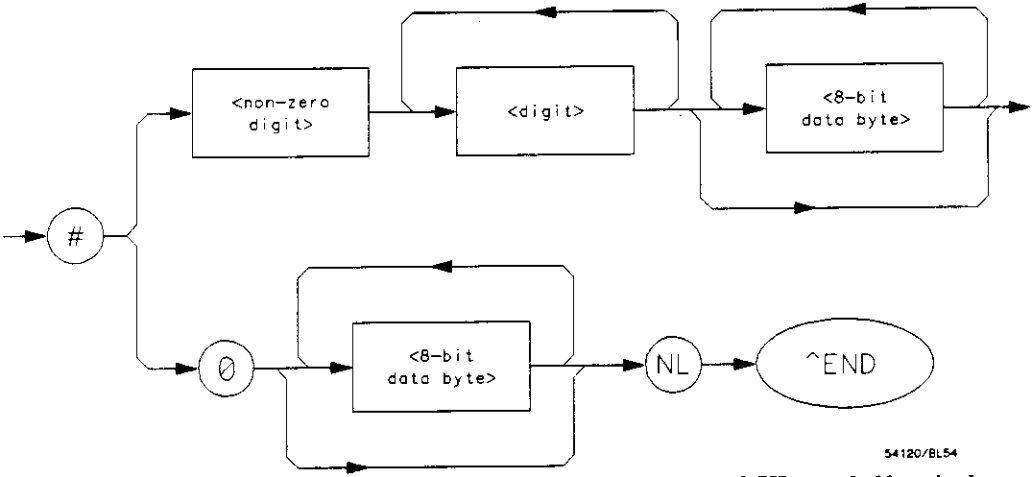
Where <inserted '> is defined as a single ASCII character with the value 27 (39 decimal).

Where <non-single quote char> is defined as a single ASCII character of any value except 27 (39 decimal).

Where <inserted "> is defined as a single ASCII character with the value 22 (34 decimal).

Where <non-double quote char> is defined as a single ASCII character of any value except 22 (34 decimal)

Figure A-14. <string program data>



Where <non-zero digit> is defined as a single ASCII encoded byte in the range 31 - 39 (49 - 57 decimal).

Where <8-bit byte> is defined as an 8-bit byte in the range 00 - FF (0 - 255 decimal).

Figure A-15. <arbitrary block program data>

<program data separator>. A comma separates multiple data parameters of a command from one another.

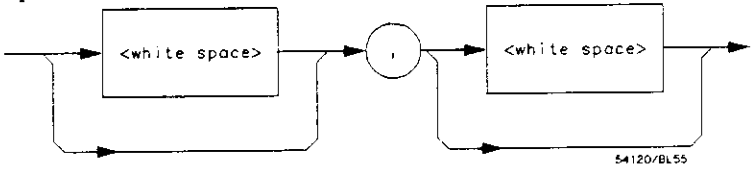


Figure A-16. <program data separator>

< program header separator >. A space separates the header from the first or only parameter of the command.

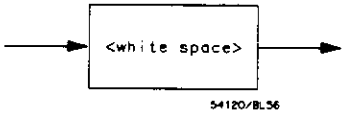
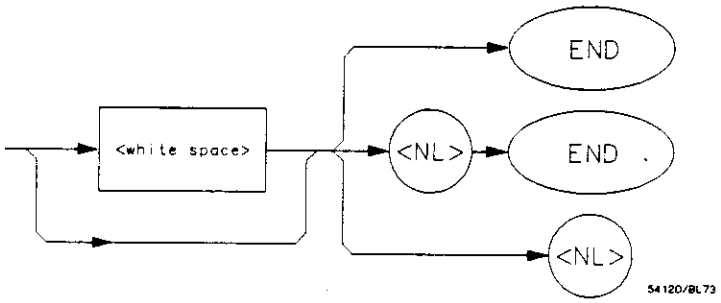


Figure A-17. < program header separator >

< program message terminator >. The < program message terminator > or < PMT > serves as the terminator to a complete < program message >. When the parser sees a complete < program message > it will begin execution of the commands within that message. The < PMT > also resets the parser to the root of the command tree.



Where < NL > is defined as a single ASCII-encoded byte 0A (10 decimal).

Figure A-18. < program message terminator >

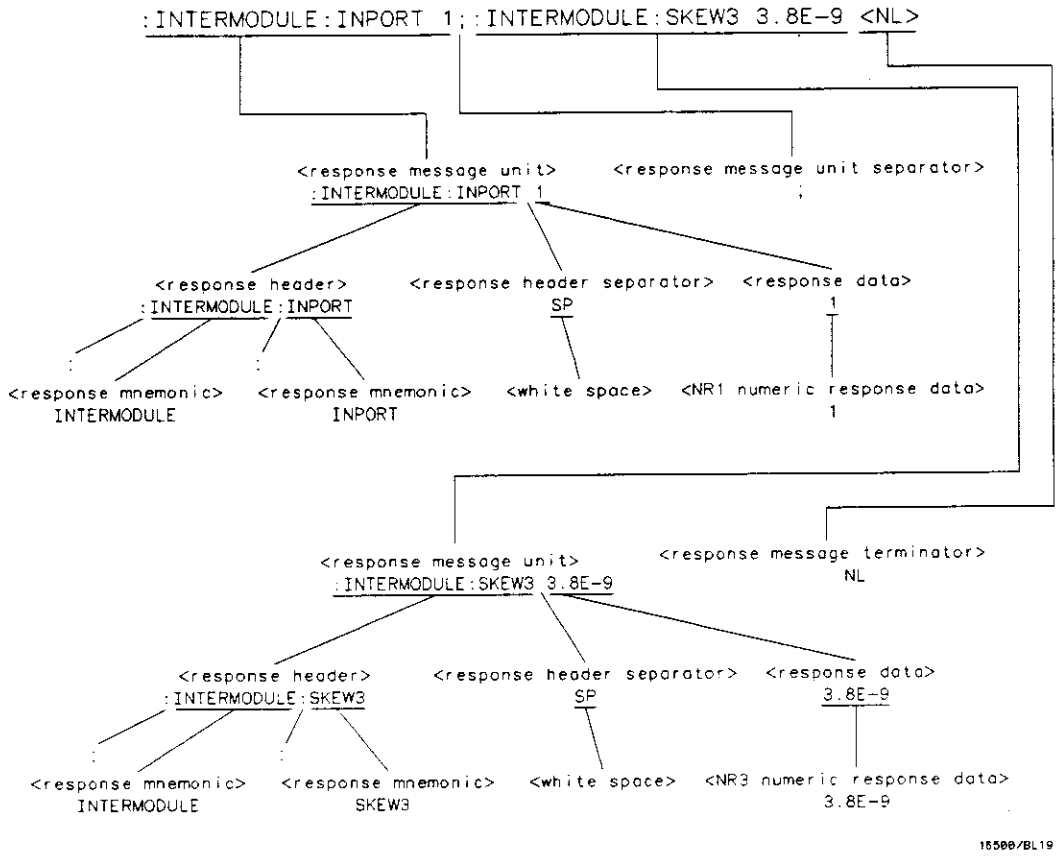


Figure A-19. <response message> Tree

Device Talking Syntax

The talking syntax of IEEE 488.2 is designed to be more precise than the listening syntax. This allows the programmer to write routines which can easily interpret and use the data the instrument is sending. One of the implications of this is the absence of < white space > in the talking formats. The instrument will not pad messages which are being sent to the controller with spaces.

< response message >. This element serves as a complete response from the instrument. It is the result of the instrument executing and buffering the results from a complete < program message >. The complete < response message > should be read before sending another < program message > to the instrument.

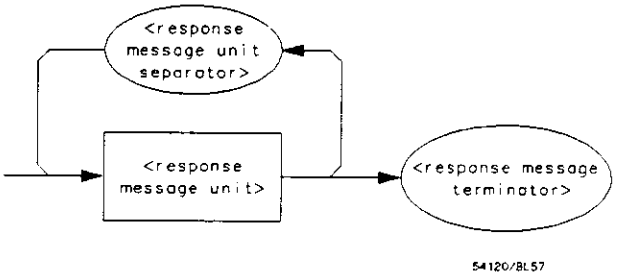
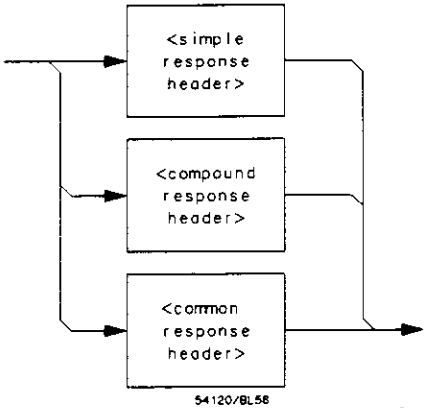


Figure A-20. < response message >

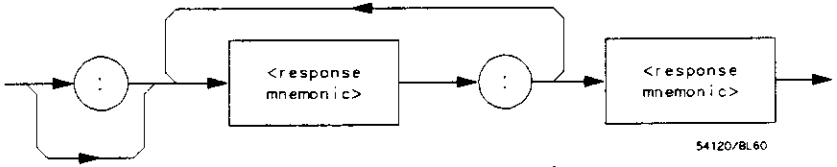
< response message unit >. This element serves as the container of individual pieces of a response. Typically a < query message unit > will generate one < response message unit >, although a < query message unit > may generate multiple < response message unit > s.

< response header >. The < response header >, when returned, indicates what the response data represents.



Where <simple response mnenomic> is defined as

Where <compound response header> is defined as



Where <common response header> is defined as

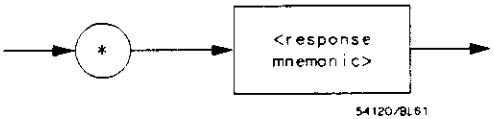
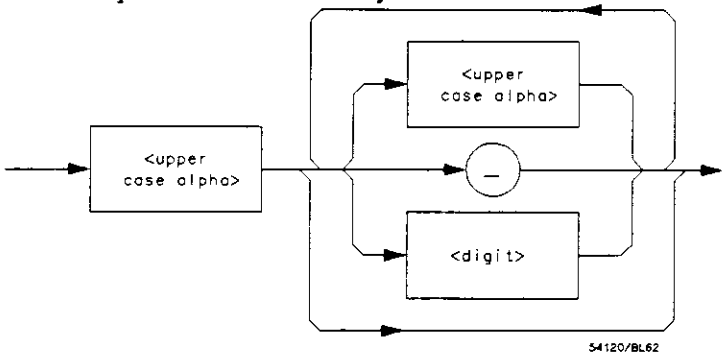


Figure A-21. <response message unit>

Where <response mnemonic> is defined as



Where <uppercase alpha> is defined as a single ASCII encoded byte in the range 41 - 5A (65 - 90 decimal).

Where (_) represents an "underscore", a single ASCII-encoded byte with the value 5F (95 decimal).

Figure A-21. <response message unit> (Continued)

<response data>. The <response data> element represents the various types of data which the instrument may return. These types include: <character response data>, <nr1 numeric response data>, <nr3 numeric response data>, <string response data>, <definite length arbitrary block response data>, and <arbitrary ASCII response data>.

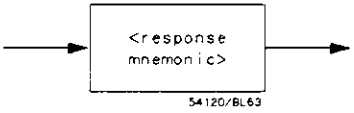


Figure A-22. <character response data>

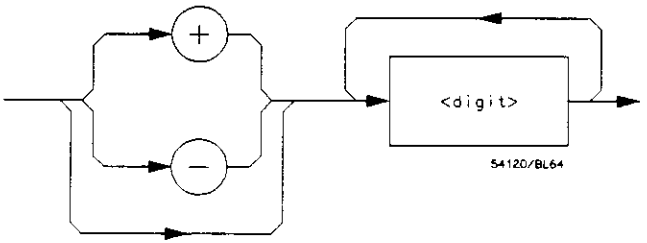


Figure A-23. <nr1 numeric response data>

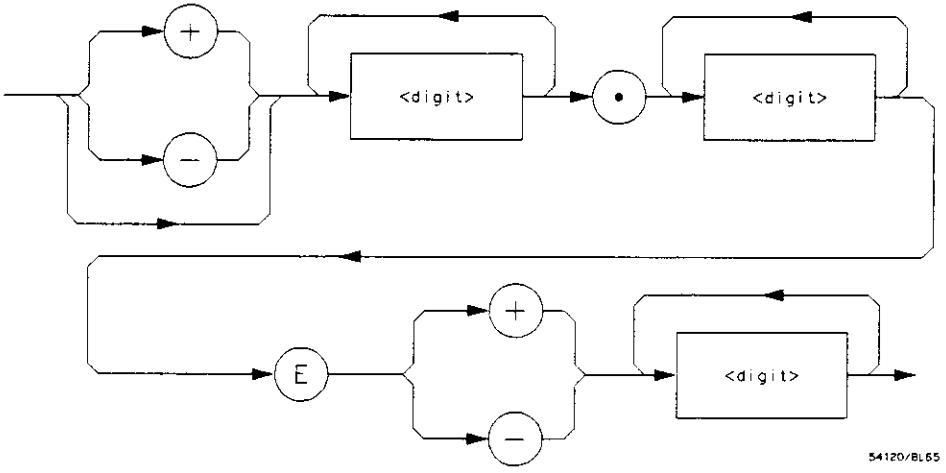


Figure A-24. <nr3 numeric response data>

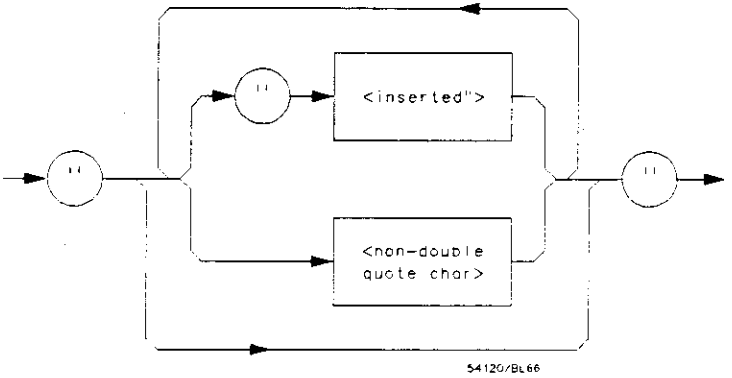


Figure A-25. <string response data>

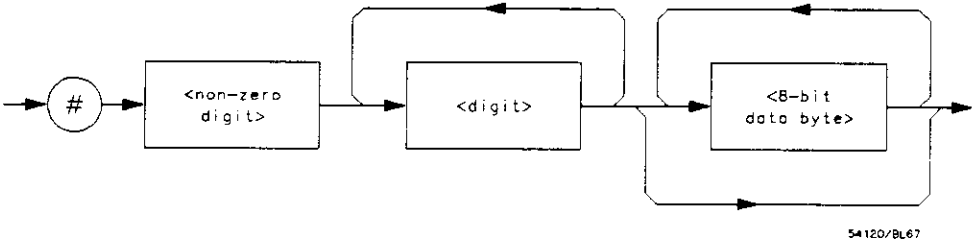
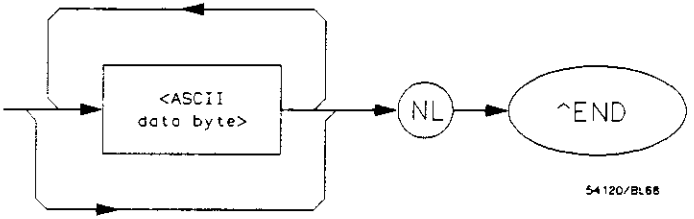


Figure A-26. <definite length arbitrary block response data>



Where <ASCII data byte> represents any ASCII-encoded data byte except <NL> (0A, 10 decimal).

Notes:

1. The END message provides an unambiguous termination to an element that contains arbitrary ASCII characters.

2. The IEEE 488.1 END message serves the dual function of terminating this element as well as terminating the <RESPONSE MESSAGE>. It is only sent once with the last byte of the indefinite block data. The NL is present for consistency with the <RESPONSE MESSAGE TERMINATOR>. Indefinite block data format is not supported in the HP 16500A.

Figure A-27. <arbitrary ASCII response data>

< response data separator >. A comma separates multiple pieces of response data within a single < response message unit >.

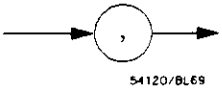


Figure A-28. < response data separator >

< response header separator >. A space (ASCII decimal 32) delimits the response header, if returned, from the first or only piece of data.

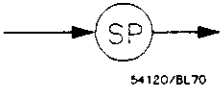


Figure A-29. < response header separator >

< response message unit separator >. A semicolon delimits the < response message unit > s if multiple responses are returned.

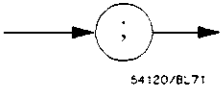


Figure A-30. < response message unit separator >

< response message terminator >. A < response message terminator > (NL) terminates a complete < response message >. It should be read from the instrument along with the response itself.

Note

If you do not read the < response message terminator > the HP 16500A will produce an interrupted error.

Common Commands

IEEE 488.2 defines a set of common commands. These commands perform functions which are common to any type of instrument. They can therefore be implemented in a standard way across a wide variety of instrumentation. All the common commands of IEEE 488.2 begin with an asterisk. There is one key difference between the IEEE 488.2 common commands and the rest of the commands found in this instrument. The IEEE 488.2 common commands do not affect the parser's position within the command tree. More information about the command tree and tree traversal can be found in the Programming and Documentation Conventions chapter.

Table A-3. HP 16500A's Common Commands

Command	Command Name
*CLS	Clear Status Command
*ESE	Event Status Enable Command
*ESE?	Event Status Enable Query
*ESR?	Event Status Register Query
*IDN?	Identification Query
*IST?	Individual Status Query
*OPC	Operation Complete Command
*OPC?	Operation Complete Query
*OPT?	Option Identification Query
*PRE	Parallel Poll Enable Register Enable Command
*PRE?	Parallel Poll Enable Register Enable Query
*RST	Reset (not implemented on HP 16500A)
*SRE	Service Request Enable Command
*SRE?	Service Request Enable Query
*STB?	Read Status Byte Query
*TRG	Trigger Command
*TST?	Self-Test Query
*WAI	Wait-to-Continue Command

Status Reporting B

Introduction

The status reporting features which are available over the bus include the serial and parallel polls. IEEE 488.2 defines data structures, commands, and common bit definitions for each. There are also instrument defined structures and bits.

The bits in the status byte act as summary bits for the data structures residing behind them. In the case of queues, the summary bit is set if the queue is not empty. For registers, the summary bit is set if any enabled bit in the event register is set. The events are enabled via the corresponding event enable register. Events captured by an event register remain set until the register is read or cleared. Registers are read with their associated commands. The "*CLS" command clears all event registers and all queues except the output queue. If "*CLS" is sent immediately following a < program message terminator > , the output queue will also be cleared.

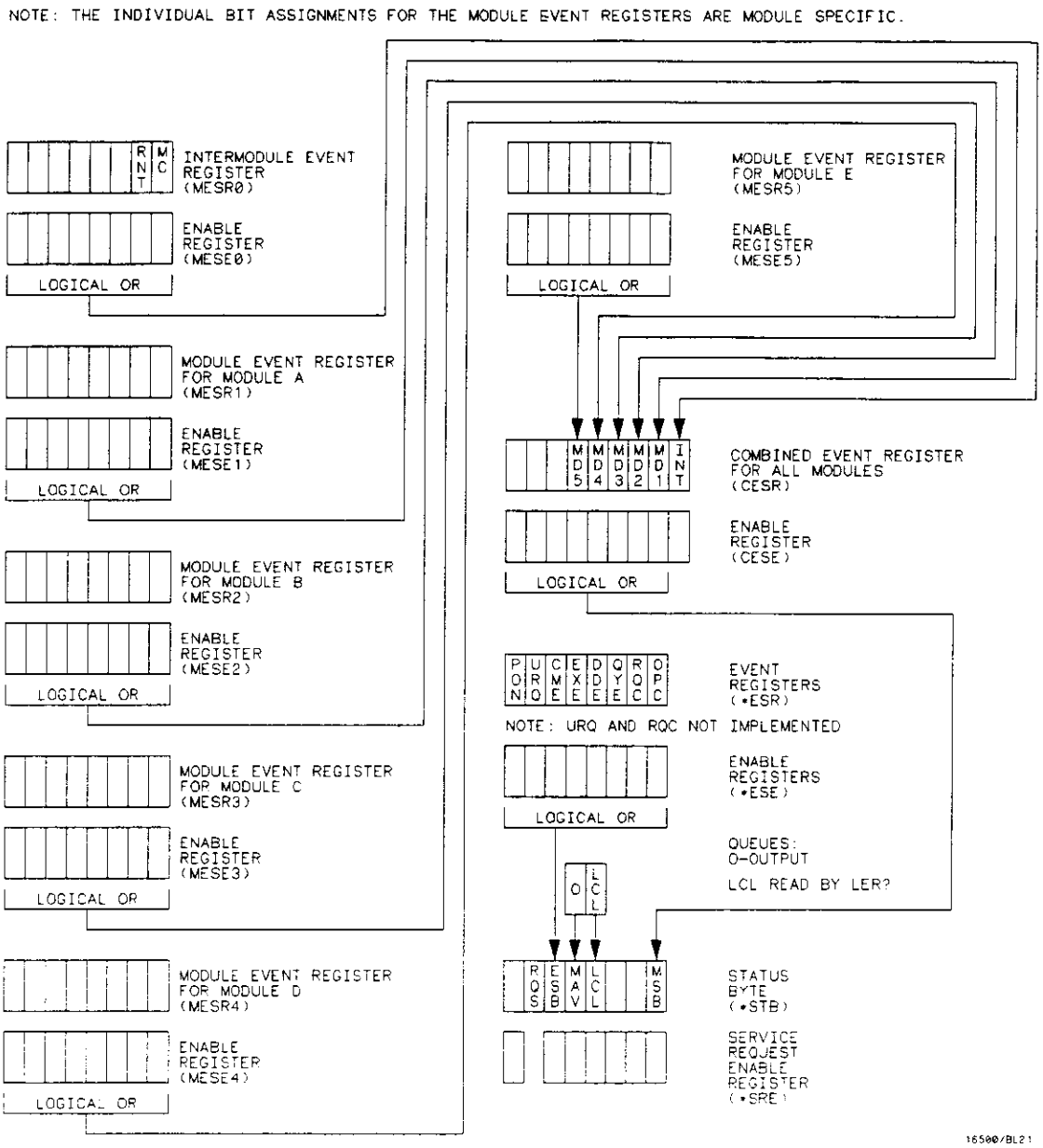


Figure B-1. Status Byte Structures and Concepts

Event Status Register

The Event Status Register is a 488.2 defined register. The bits in this register are "latched." That is, once an event happens which sets a bit, that bit will only be cleared if the register is read.

Service Request Enable Register

The Service Request Enable Register is an 8-bit register. Each bit enables the corresponding bit in the status byte to cause a service request. The sixth bit does not logically exist and is always returned as a zero. To read and write to this register use the *SRE? and *SRE commands.

Bit Definitions

MAV - message available. Indicates whether there is a response in the output queue.

ESB - event status bit. Indicates if any of the conditions in the Standard Event Status Register are set and enabled.

MSS - master summary status. Indicates whether the device has a reason for requesting service. This bit is returned for the *STB? query.

RQS - request service. Indicates if the device is requesting service. This bit is returned during a serial poll. RQS will be set to 0 after being read via a serial poll (MSS is not reset by *STB?).

PON - power on. Indicates power has been turned on.

URQ - user request. Always 0 on the HP 16500A.

CME - command error. Indicates whether the parser detected an error.

Note

The error numbers and/or strings for CME, EXE, DDE, and QYE can be read from a device defined queue (which is not part of 488.2) with the query :SYSTEM:ERROR?.

EXE - execution error. Indicates whether a parameter was out of range, or inconsistent with current settings.

DDE - device specific error. Indicates whether the device was unable to complete an operation for device dependent reasons.

QYE - query error. Indicates whether the protocol for queries has been violated.

RQC - request control. Always 0 on the HP 16500A.

OPC - operation complete. Indicates whether the device has completed all pending operations. OPC is controlled by the *OPC common command. Because this command can appear after any other command, it serves as a general purpose operation complete message generator.

LCL - remote to local. Indicates whether a remote to local transition has occurred.

MSB - module summary bit. Indicates that an enable event in one of the modules Status registers has occurred.

Key Features

A few of the most important features of Status Reporting are listed in the following paragraphs.

Operation Complete. The IEEE 488.2 structure provides one technique which can be used to find out if any operation is finished. The *OPC command, when sent to the instrument after the operation of interest, will set the OPC bit in the Standard Event Status Register. If the OPC bit and the RQS bit have been enabled a service request will be generated. The commands which affect the OPC bit are the overlapped commands.

```
OUTPUT XXX;"*SRE 32 ; *ESE 1"  !enables an OPC service request
```

Status Byte. The Status Byte contains the basic status information which is sent over the bus in a serial poll. If the device is requesting service (RQS set), and the controller serial polls the device, the RQS bit is cleared. The MSS (Master Summary Status) bit (read with *STB?) and other bits of the Status Byte are not be cleared by reading them. Only the RQS bit is cleared when read. To see how the RQS and MSS bits are set and used, see figure B-2.

The Status Byte is cleared with the *CLS common-command.

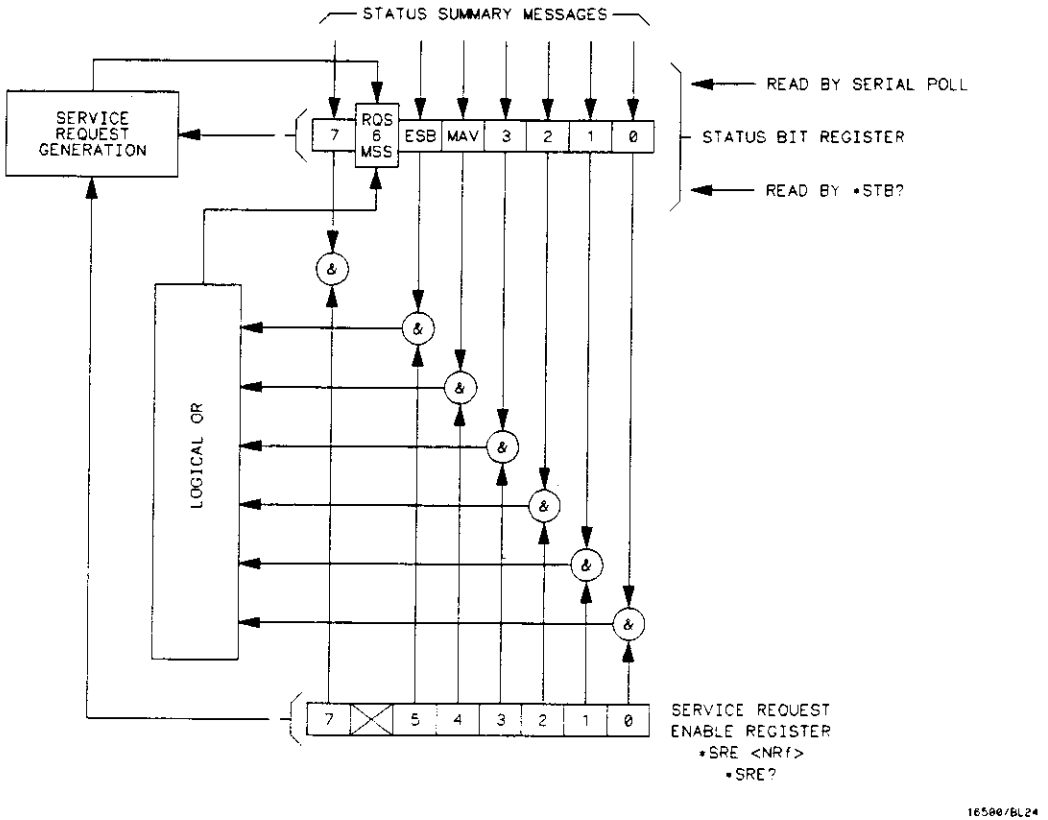


Figure B-2. Service Request Enabling

Serial Poll

The HP 16500A supports the IEEE 488.1 serial poll feature. When a serial poll of the instrument is requested, the RQS bit is returned on bit 6 of the status byte.

Using Serial Poll (HP-IB)

This example will show how to use the service request by conducting a serial poll of all instruments on the HP-IB bus. In this example, assume that there are two instruments on the bus; a Logic Analysis System at address 7 and a printer at address 1.

The program command for serial poll using HP BASIC 4.0 is Stat = SPOLL(707). The address 707 is the address of the Logic Analysis System in the this example. The command for checking the printer is Stat = SPOLL(701) because the address of that instrument is 01 on bus address 7. This command reads the contents of the HP-IB Status Byte Register into the variable called Stat. At that time bit 6 (RQS bit) of the variable Stat can be tested to see if it is set (bit 6 = 1).

The serial poll operation can be conducted in the following manner:

1. Enable interrupts on the bus. This allows the controller to "see" the SRQ line.

2. Disable interrupts on the bus.

3. If the SRQ line is high (some instrument is requesting service) then check the instrument at address 1 to see if bit 6 of its status register is high.

4. To check whether bit 6 of an instruments status register is high, use the following Basic statement:

```
IF BIT (Stat, 6) THEN
```

5. If bit 6 of the instrument at address 1 is not high, then check the instrument at address 7 to see if bit 6 of its status register is high.

6. As soon as the instrument with status bit 6 high is found check the rest of the status bits to determine what is required.

The SPOLL(707) command causes much more to happen on the bus than simply reading the register. This command clears the bus automatically, addresses the talker and listener, sends SPE (serial poll enable) and SPD (serial poll disable) bus commands, and reads the data. For more information about serial poll, refer to your controller manual, and programming language reference manuals.

After the serial poll is completed, the RQS bit in the HP 16500A Status Byte Register will be reset if it was set. Once a bit in the Status Byte Register is set, it will remain set until the status is cleared with a *CLS command, or the instrument is reset.

Parallel Poll

Parallel poll is a controller initiated operation which is used to obtain information from several devices simultaneously. When a controller initiates a Parallel Poll, each device returns a Status Bit via one of the DIO data lines. Device DIO assignments are made by the controller using the PPC (Parallel Poll Configure) sequence. Devices respond either individually, each on a separate DIO line; collectively on a single DIO line; or any combination of these two ways. When responding collectively, the result is a logical AND (True High) or logical OR (True Low) of the groups of status bits.

Figure B-3 shows the Parallel Poll Data Structure. The summary bit is sent in response to a Parallel Poll. This summary bit is the "ist" (individual status) local message.

The Parallel Poll Enable Register determines which events are summarized in the ist. The *PRE command is used to write to the enable register and the *PRE? query is used to read the register. The *IST? query can be used to read the "ist" without doing a parallel poll.

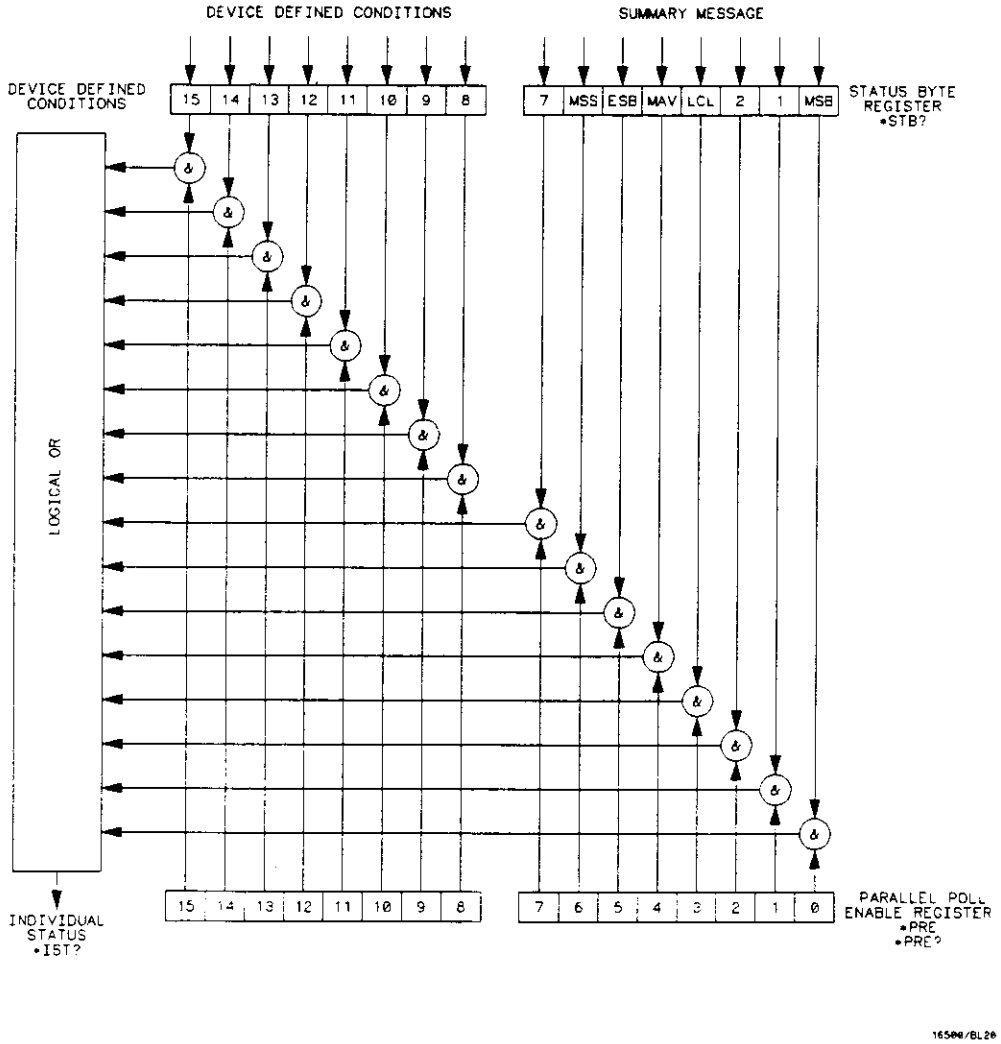


Figure B-3. Parallel Poll Data Structure

Polling HP-IB Devices

Parallel Poll is the fastest means of gathering device status when several devices are connected to the bus. Each device (with this capability) can be programmed to respond with one bit of status when parallel polled. This makes it possible to obtain the status of several devices in one operation. If a device responds affirmatively to a parallel poll, more information about its specific status can be obtained by conducting a serial poll of the device.

Configuring Parallel Poll Responses

Certain devices, including the HP 16500A, can be remotely programmed by a controller to respond to a parallel poll. A device which is currently configured for a parallel poll responds to the poll by placing its current status on one of the bus data lines. The response and the data-bit number can then be programmed by the PPC (parallel Poll Configure) statement. No multiple listeners can be specified in this statement. If more than one device is to respond on a single bit, each device must be configured with a separate PPC statement.

Example:

```
ASSIGN @Device TO 707
PPOLL CONFIGURE @Device;Mask
```

The value of Mask (any numeric expression can be specified) is first rounded and then used to configure the device's parallel response. The least significant 3 bits (bits 0 through 2) of the expression are used to determine which data line the device is to respond on (place its status on). Bit 3 specifies the "true" state of the parallel poll response bit of the device. A value of 0 implies that the device's response is 0 when its status bit message is true.

Example: The following statement configures the device at address 07 on the interface select code 7 to respond by placing a 0 on bit 4 when its status response is "true."

```
PPOLL CONFIGURE 707;4
```

Conducting a Parallel Poll

The PPOLL (Parallel Poll) function returns a single byte containing up to 8 status bit messages for all devices on the bus capable of responding to the poll. Each bit returned by the function corresponds to the status bit of the device(s) configured to respond to the parallel poll (one or more devices can respond on a single line). The PPOLL function can only be executed by the controller. It is initiated by the simultaneous assertion of ATN and EOI.

Example:

```
Response = PPOLL(7)
```

Disabling Parallel Poll Responses

The PPU (Parallel Poll Unconfigure) statement gives the controller the capability of disabling the parallel poll responses of one or more devices on the bus.

Examples: The following statement disables device 5 only:

```
PPOLL UNCONFIGURE 705
```

This statement disables all devices on interface select code 8 from responding to a parallel poll:

```
PPOLL UNCONFIGURE 8
```

If no primary address is specified, all bus devices are disabled from responding to a parallel poll. If a primary address is specified, only the specified devices (which have the parallel poll configure capability) are disabled.

HP-IB Commands

The following paragraphs describe actual HP-IB commands which can be used to perform the functions of the Basic commands shown in the previous examples.

Parallel Poll Unconfigure Command. The parallel poll unconfigure command (PPU) resets all parallel poll devices to the idle state (unable to respond to a parallel poll).

Parallel Poll Configure Command. The parallel poll configure command (PPC) causes the addressed listener to be configured according to the parallel poll enable secondary command PPE.

Parallel Poll Enable Command. The parallel poll enable secondary command (PPE) configures the devices which have received the PPC command to respond to a parallel poll on a particular HP-IB DIO line with a particular level.

Parallel Poll Disable Command. The parallel poll disable secondary command (PPD) disables the devices which have received the PPC command from responding to the parallel poll.

Table B-1. Parallel Poll Commands

Command	Mnemonic	Decimal Code	ASCII/ISO Character
Parallel Poll Unconfigure (Multiline Command)	PPU	21	NAK
Parallel Poll Configure (Addressed Command)	PPC	05	ENQ
Parallel Poll Enable (Secondary Command)	PPE	96-111	I-O
Parallel Poll Disable (Secondary Command)	PPD	112	P

Error Messages C

This section covers the error messages that relate to the HP 16500A mainframe and modules.

Device Dependent Errors

200 Label not found

201 Pattern string invalid

202 Qualifier invalid

203 Data not available

300 RS-232C error

Command Errors

-100 Command error (unknown command)(generic error)

-101 Invalid character received

-110 Command header error

-111 Header delimiter error

-120 Numeric argument error

-121 Wrong data type (numeric expected)

-123 Numeric overflow

-129 Missing numeric argument

-130 Non numeric argument error (character,string, or block)

-131 Wrong data type (character expected)

-132 Wrong data type (string expected)

-133 Wrong data type (block type #D required)

-134 Data overflow (string or block too long)

-139 Missing non numeric argument

-142 Too many arguments

-143 Argument delimiter error

-144 Invalid message unit delimiter

Execution Errors

-200 No Can Do (generic execution error)

-201 Not executable in Local Mode

-202 Settings lost due to return-to-local or power on

-203 Trigger ignored

-211 Legal command, but settings conflict

-212 Argument out of range

-221 Busy doing something else

-222 Insufficient capability or configuration

-232 Output buffer full or overflow

-240 Mass Memory error (generic)

-241 Mass storage device not present

-242 No media

-243 Bad media

-244 Media full

-245 Directory full

-246 File name not found

-247 Duplicate file name

-248 Media protected

Internal Errors

-300 Device Failure (generic hardware error)

-301 Interrupt fault

-302 System Error

-303 Time out

-310 RAM error

-311 RAM failure (hardware error)

-312 RAM data loss (software error)

-313 Calibration data loss

-320 ROM error

-321 ROM checksum

-322 Hardware and Firmware incompatible

-330 Power on test failed

-340 Self Test failed

-350 Too Many Errors (Error queue overflow)

Query Errors

-400 Query Error (generic)

-410 Query INTERRUPTED

-420 Query UNTERMINATED

-421 Query received. Indefinite block response in progress

-422 Addressed to Talk, Nothing to Say

-430 Query DEADLOCKED

Index

D

M

N

O

P

Q

R

S

T

U

W

X