Programming Reference

HP 1650A/51A
Logic Analyzers

HEWLETT PACKARD

Manual Part Number 01650-90910

Printed in U.S.A. November 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.

Printing History

New editions are complete revisions of the manual. Update packages, which are issued between editions, contain additional and replacement pages to be merged into the manual by the customer. The dates on the title page change only when a new edition or a new update is published. No information is incorporated into a reprinting unless it appears as a prior update; the edition does not change when an update is incorporated.

A software code may be printed before the date; this indicates the version level of the software product at the time of the manual or update was issued. Many product updates and fixes do not require manual changes and, conversely, manual corrections may be done without accompanying product changes. Therefore, do not expect a one to one correspondence between product updates and manual updates.

Edition 1 November 1988 01650-90910

List of Effective Pages

The List of Effective Pages gives the data of the current edition and of any pages changed in updates to that edition. Within the manual, any page changed since the last edition will have the date the changes were made printed on the bottom of the page. If an update is incorporated when a new edition of the manual is printed, the change dates are removed from the bottom of the pages and the new edition date is listed in Printing History and on the title page.

Pages	Effective Date
All	November 1988

Table of Contents

Chapter 2:

Programming Over RS-232C

Chapter 3:

Programming and Documentation Conventions

Chapter 4:

Chapter 5:

Chapter 6:

Chapter 7:

Chapter 11:

STRace Subsystem

Chapter 12:

SLISt Subsystem

Appendix B:

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 16 of this manual. 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 (pages 1-9) concentrates on program syntax, and the second section (pages 10-17) 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.

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 1650A/51A, 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 1650A/51A, and the ENTER statement for receiving response messages from the HP 1650A/51A.

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 and the programming language you are using.

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) 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 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. The following applies to HP 9000 Series 200/300 controllers.

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:

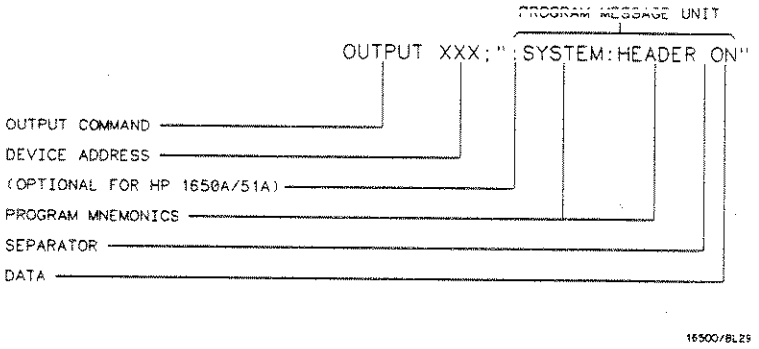


Figure 1-1. Program Message Syntax

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, :MACHINE 1) 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, the function mnemonic MMODE may be used to specify the marker mode in the state listing or the timing waveforms:

```
:SLIST:MMODE PATTERN
```

- sets the marker mode to pattern in the state listing.

SLIST and TWAVEFORM are subsystem selectors and determine which marker mode is being modified.

```
:TWAVEFORM:MMODE TIME
```

- sets the marker mode to time in the timing waveforms.

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. For example, the command :MACHINE1:TWAVEFORM:XOTIME? instructs the instrument to place the X to O time in the output queue.

Note

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

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 a "query UNTERMINATED" 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 turns 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 Chapter 3 "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, :SYSTEM:MENU TRACE, 1 has two data parameters: TRACE and 1.

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, :MACHINE requires the desired analyzer 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. For example:

#B10101010 - Binary Base
#Q1234567 - Octal Base
#H2468ABC - Hexadecimal Base
1234567 - Decimal Base

Program Message Terminator

The program codes within a data message are executed after the program message terminator is received. The terminator is the NL (New Line) character. 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.

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>

```
:MACHINE1:ASSIGN2;:SYSTEM:HEADERS ON
```

Note

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

Summary

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

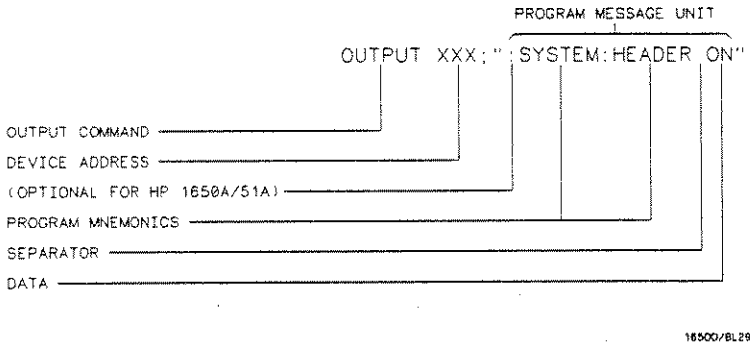


Figure 1-2. Syntax for Programming Over the Bus.

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 the HP 1650A/51A. 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 represent the device address required by your controller. 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.

Example Program

This program demonstrates the basic command structure used to program the HP 1650A/51A.

```
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 OUTPUT XXX;":MMEM:LOAD:CONFIG 'TEST_E'"   !Load configuration file
50 OUTPUT XXX;":MENU FORMAT,1"        !Select Format menu for machine 1
60 OUTPUT XXX;":RMODE SINGLE"         !Select run mode
70 OUTPUT XXX;":START"                !Run the measurement
```

Note

The three Xs (XXX) after the "OUTPUT", "CLEAR" and "ENTER" statements in the previous examples represent the device address required by your controller.

Program Overview

Line 10 initializes the instrument interface to a known state,
Lines 20 and **30** turn the headers and longform on.
Line 40 loads the configuration file "TEST_E" from the disc drive.
Line 50 displays the Format menu for machine 1.
Lines 60 and **70** 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 :MACHINE1:ASSIGN?, you must follow that query with the program statement ENTER Assignment to read the result of the query and place the result in a variable (Assignment).

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 :MACHINE1:SFORMAT:THRESHOLD2? command query:

<data> <terminator> (with HEADER OFF)

:MACH1:SFOR:THR2 <separator> <data> <terminator> (with HEADER ON/LONGFORM OFF)

:MACHINE1:SFORMAT:THRESHOLD2 <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.

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 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 your controller.

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:

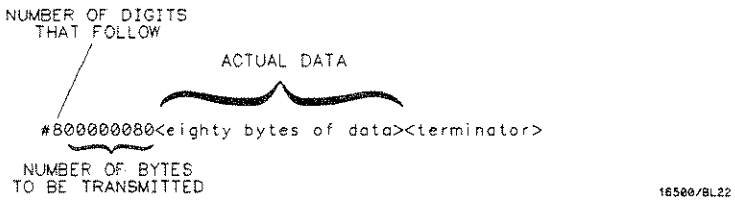


Figure 1-3. Definite-length Block Response Data

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 HP1650A/51A.

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 RS-232C 2

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 preceding or subsequent data characters. Each character is sent as a complete entity without relationship to other events.

Interface Operation

The HP 1650A/51A 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 1650A/51A 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 DTE 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 1650A/51A. For both three-wire and extended hardwire operation, the DCD and DSR inputs to the HP 1650A/51A must remain high for proper operation. With extended hardwire operation, a high on the CTS input allows the HP 1650A/51A to send data and a low on this line disables the HP 1650A/51A 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 1650A/51A 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 1650A/51A are used to control the operation of the RS-232C bus relative to the HP 1650A/51A. 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 1650A/51A and the controller. This provides a much simpler connection between devices since you can ignore hardware handshake requirements. The HP 1650A/51A 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 1650A/51A)
- Pin 3 RD (Receive Data into HP 1650A/51A)

The TD (Transmit Data) line from the HP 1650A/51A must connect to the RD (Receive Data) line on the controller. Likewise, the RD line from the HP 1650A/51A must connect to the TD line on the controller. Internal pull-up resistors in the HP 1650A/51A 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 1650A/51A. 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 1650A/51A and the controller. This allows you to have more control of data flow between devices. The HP 1650A/51A 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 1650A/51A)
- Pin 3 RD (Receive Data into HP 1650A/51A)

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 1650A/51A which can be used to control incoming data flow.
- Pin 5 CTS (Clear To Send) is an input to the HP 1650A/51A which controls data flow from the HP 1650A/51A.
- Pin 6 DSR (Data Set Ready) is an input to the HP 1650A/51A which controls data flow from the HP 1650A/51A within two bytes.
- Pin 8 DCD (Data Carrier Detect) is an input to the HP 1650A/51A which controls data flow from the HP 1650A/51A within two bytes.
- Pin 20 DTR (Data Terminal Ready) is an output from the HP 1650A/51A which is enabled as long as the HP 1650A/51A is turned on.

The TD (Transmit Data) line from the HP 1650A/51A must connect to the RD (Receive Data) line on the controller. Likewise, the RD line from the HP 1650A/51A must connect to the TD line on the controller.

The RTS (Request To Send), is an output from the HP 1650A/51A 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 1650A/51A which control data flow from the HP 1650A/51A (Pin 2). Internal pull-up resistors in the HP 1650A/51A 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 1650A/51A to send data to the controller. A low on any one of these lines will disable the HP 1650A/51A data transmission. Dropping the CTS line low during data transmission will stop HP 1650A/51A data transmission immediately. Dropping either the DSR or DCD line low during data transmission will stop HP 1650A/51A data transmission, but as many as two additional bytes may be transmitted from the HP 1650A/51A.

Cable Example

Figure 2-1 is an example of how to connect the HP 1650A/51A to the HP 98628A 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 1650A/51A as shown in figure 2-1

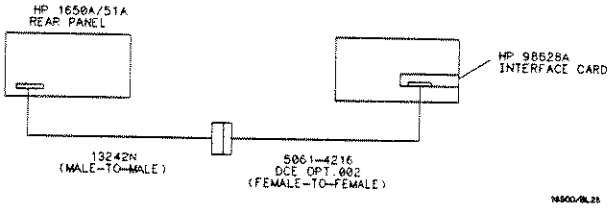


Figure 2-1. Cable Example

Configuring the Interface

The front-panel I/O menu key allows you access to the RS-232C Configuration menu where the RS-232C interface is configured.

If you are not familiar with how to configure the RS-232C interface, refer to the *HP 1650A/51A Front-panel 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 1650A/51A to properly communicate over the RS-232C bus. The HP 1650A/51A 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 1650A/51A) controls data flow and can request that the sender (HP 1650A/51A 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.

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 its 8-bit equivalent.

Note

The controller and the HP 1650A/51A must be in the same bit mode to properly communicate over the RS-232C. This means that both the controller and the HP 1650A/51A 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 1650A/51A Reference Manual. For information on RS-232C voltage levels and connector pinouts, refer to the HP 1650A/51A 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 by specifying the correct interface code 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 "System 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 3

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 3-1.

Table 3-1. Mnemonic Truncation

Longform	Shortform
START	STAR
MASTER	MAST
EDGE	EDGE
DELAY	DEL
PATTERN	PATT

The Command Tree

The command tree (figure 3-1) shows all commands in the HP 1650A/51A logic analyzers 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"

System Commands. The system 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: ":SYSTEM:HEADER ON"

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

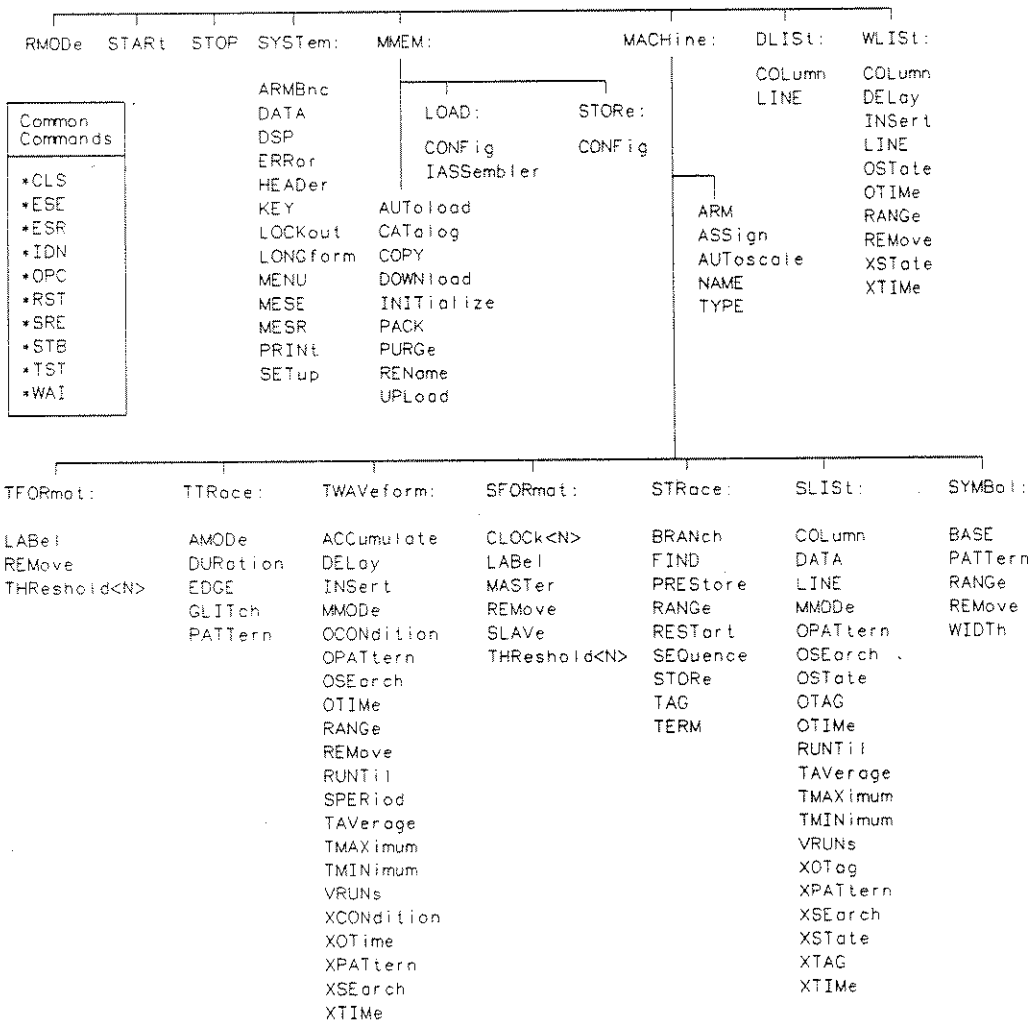


Figure 3-1. The HP 1650A/51A System Command Tree

Tree Traversal Rules

Command headers are created by traversing down the command tree. A legal command header from the command tree in figure 3-1 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> (<NL> 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 3-1, 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 represents the device address required by your controller.

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. The overlapped commands for the HP 1650A/51A are STARt, STOP, and AUTOscale.

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 1650A/51A 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 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 represent the device address required by your controller.

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>

space ::= white space

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 1650A/51A programming commands are divided into three types: common commands, system commands, and subsystem commands. A programming command tree is shown in figure 3-1 and a programming command cross-reference is shown in table 3-2.

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.

System Commands

The system 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 12 subsystems in the HP 1650A/51A are:

- SYSTem - controls some basic functions of the instrument.
- MMEMory - provides access to the internal disc drive.
- MACHine - controls the machine level function and allows access to the instrument configuration subsystems.
- DLISt - allows access to the dual listing function of two state analyzers.
- WLISt - allows access to the mixed (timing/state) functions.
- TFORmat - allows access to the timing format functions.
- TTRace - allows access to the timing trace functions.
- TWAVeform - allows access to the timing waveforms functions.
- SFORmat - allow access to the state format functions.
- STRace - allows access to the state trace functions.
- SLISt - allows access to the state listing functions.
- SYMBols - allows access to the symbol specification functions.

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.

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 timing waveform delay value to the logic analyzer of 100 ms, that value could be sent using a decimal (.1), or an exponential (1e-1 or 1.0E-1), or a suffix (100ms or 100MS).

Note

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

As an example, set Timing Waveform Delay to 100 ms by sending one of the following:

- commands in longform and using the decimal format.

```
OUTPUT XXX;":MACHINE1:TWAVEFORM:DELAY .1"
```

- commands in shortform and using an exponential format.

```
OUTPUT XXX;":MACH1:TWAV:DEL 1E-1"
```

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

```
OUTPUT XXX;":mach1:twav:del 100ms"
```

Note

In these examples, the colon shown as the first character of the command is optional on the HP 1650A/51A. The space between DELAY 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 1650A/51A, 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 1650A/51A logic analyzer is divided into 13 separate groups: common commands, system commands and 11 sets of subsystem commands. Each of the 13 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 AUToload indicates that the longform of the command is AUTOLOAD and the shortform of the command is AUT. Each of the commands contain a description of the command and its arguments, the command syntax, and a programming example.

Table 3-2. Alphabetic Command Cross-Reference

Command	Where used	Command	Where used
ACCumulate	TWAVeform	OTIMe	WLISt
AMODe	TTRace	PACK	MMEMory
ARM	MACHine	PATTern	SYMBol
ARMBnc	SYSTem	PATTern	TTRace
ASSign	MACHine	PREStore	STRace
AUTOload	MMEMory	RANGe	STRace
AUTOscale	MACHine	PRINt	SYSTem
BASE	SYMBol	PURGe	MMEMory
BRANch	STRace	RANGe	SYMBol
CATalog	MMEMory	RANGe	TWAVeform
CLOCk	SFORmat	REMove	SFORmat
COLumn	DLISt	REMove	SYMBol
COLumn	SLISt	REMove	TFORmat
COLumn	WLISt	REMove	TWAVeform
COPY	MMEMory	REName	MMEMory
CPERiod	SFORmat	RMODe	SYSTem
DATA	SLISt	RUNTil	SLISt
DATA	SYSTem	RUNTil	TWAVeform
DOWNload	MMEMory	SEQuence	STRace
DSP	SYSTem	SETup	SYSTem
DELay	TWAVeform	SLAVe	SFORmat
DURation	TTRace	SPERiod	TWAVeform
EDGE	TTRace	STARt	SYSTem
ERRor	SYSTem	STOP	SYSTem
FIND	STRace	STORe	STRace
GLITch	TTRace	STORe:CONfig	MMEMory
HEADer	SYSTem	TAG	STRace
INITialize	MMEMory	TAVerage	SLISt
INSert	TWAVeform	TAVerage	TWAVeform
KEY	SYSTem	TERM	STRace
LABel	SFORmat	THReshold	SFORmat
LABel	TFORmat	THReshold	TFORmat
LINE	DLISt	TMAXimum	SLISt
LINE	SLISt	TMAXimum	TWAVeform
LINE	WLISt	TMINimum	SLISt
LOAD:IASSembler	MMEMory	TMINimum	TWAVeform
LOAD:CONFig	MMEMory	TYPE	MACHine
LOCKout	SYSTem	UPLoad	MMEMory
LONGform	SYSTem	VRUNs	SLISt
MASTer	SFORmat	VRUNs	TWAVeform
MENU	SYSTem	WIDTh	SYMBol
MMODe	SLISt	XCONdition	TWAVeform
MMODe	TWAVeform	XOTag	SLISt
NAME	MACHine	XOTime	TWAVeform
OCONdition	TWAVeform	XPATtern	SLISt
OPATtern	SLISt	XPATtern	TWAVeform
OPATtern	TWAVeform	XSEarch	SLISt
OSEarch	SLISt	XSEarch	TWAVeform
OSEarch	TWAVeform	XSTate	WLISt
OSTate	WLISt	XSTate	SLISt
OSTate	SLISt	XTAG	SLISt
OTAG	SLISt	XTIMe	TWAVeform
OTIMe	TWAVeform	XTIMe	WLISt

Common Commands 4

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 1650A/51A 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 subsystem. 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

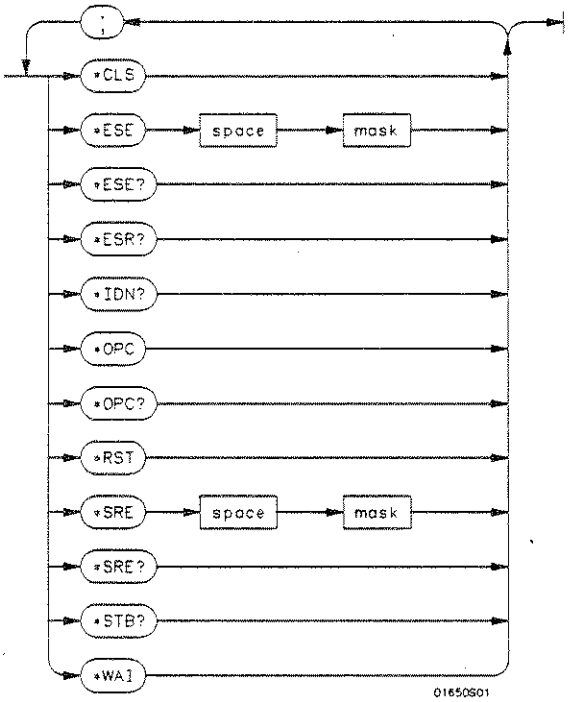
```
":MMEMORY:INITIALIZE;:SYSTEM:HEADERS ON:MMEMORY
:STORE'FILE__','DESCRIPTION'"
```

would initialize the disc, turn headers on, 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 4-1 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.*

Figure 4-1. Common Commands Syntax Diagram

*CLS (Clear Status) command

The *CLS common command clears the status data structures, including the device defined error queue. 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"<NL>

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. A zero will disable the bit. Refer to table 4-1 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.

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 4-1. 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 4-2 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 4-2. 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,1650A,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,1650A,0,REV <revision code>

where:

<revision code> ::= four digit code representing ROM revision

Example:

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

*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. The overlapped commands for the HP 1650A/51A are:

STARt
STOP
AUToscale

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

*RST (Reset) command

The *RST command (488.2) sets the HP 1650A/51A to the power-up default settings as if no autoload file was present.

Command Syntax: *RST

Example: OUTPUT XXX;"*RST"

*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 4-3 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 forces the MSS bit high (see table 4-3).

*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 4-3. HP 1650A/51A 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 4-4 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
```

*STB

Table 4-4. 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 = HP 1650A/51A has activity to report 1 = no activity to report

0 = False = Low
1 = True = High

*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. The overlapped commands for the HP 1650A/51A are:

STARt
STOP
AUToscale

Command Syntax: *WAI

Example: OUTPUT XXX;"*WAI"

System Commands 5

Introduction

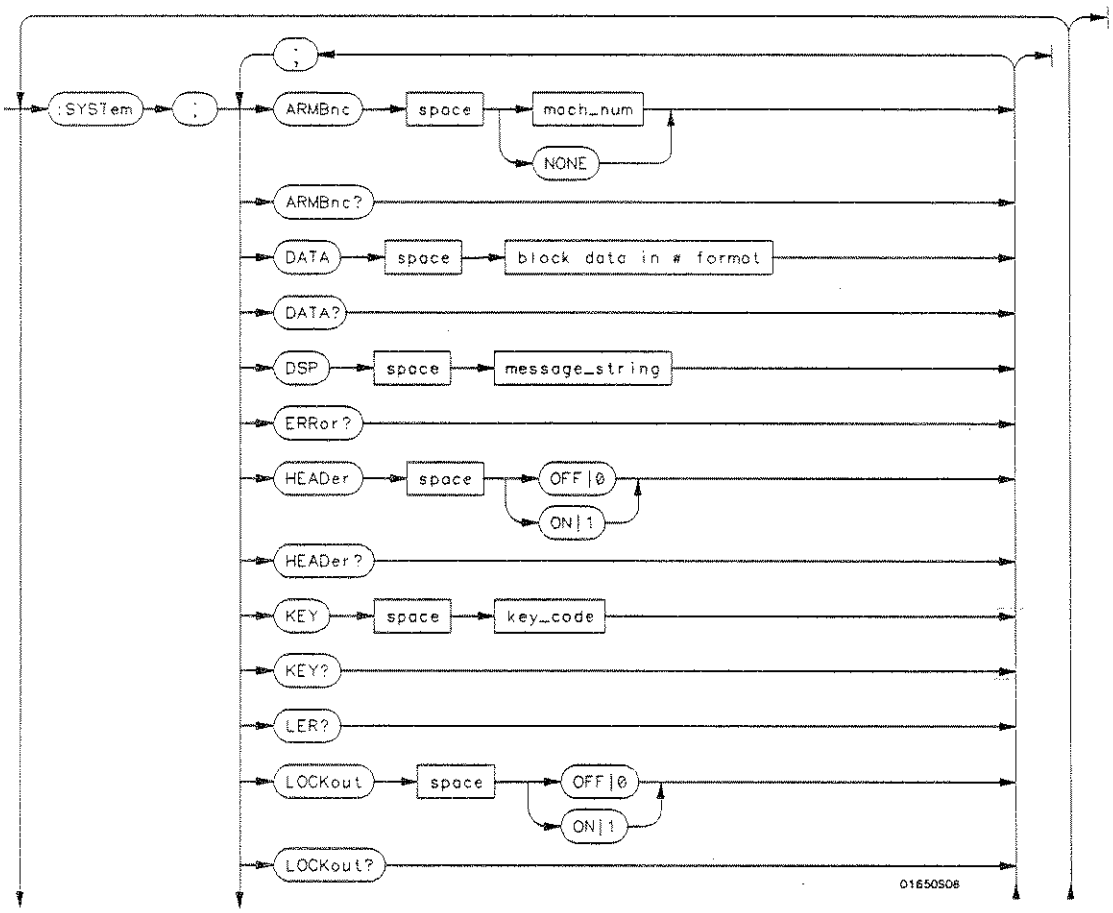
System commands control the basic operation of the instrument including formatting query responses and enabling reading and writing to the advisory line of the instrument's display. They can be called at anytime. The HP 1650A/51A System commands are:

- ARMBnc
- DATA
- DSP (display)
- ERRor
- HEADer
- KEY
- LER (Local Event Register)
- LOCKout
- LONGform
- MENU
- MESE
- MESR
- PRINt
- SETup

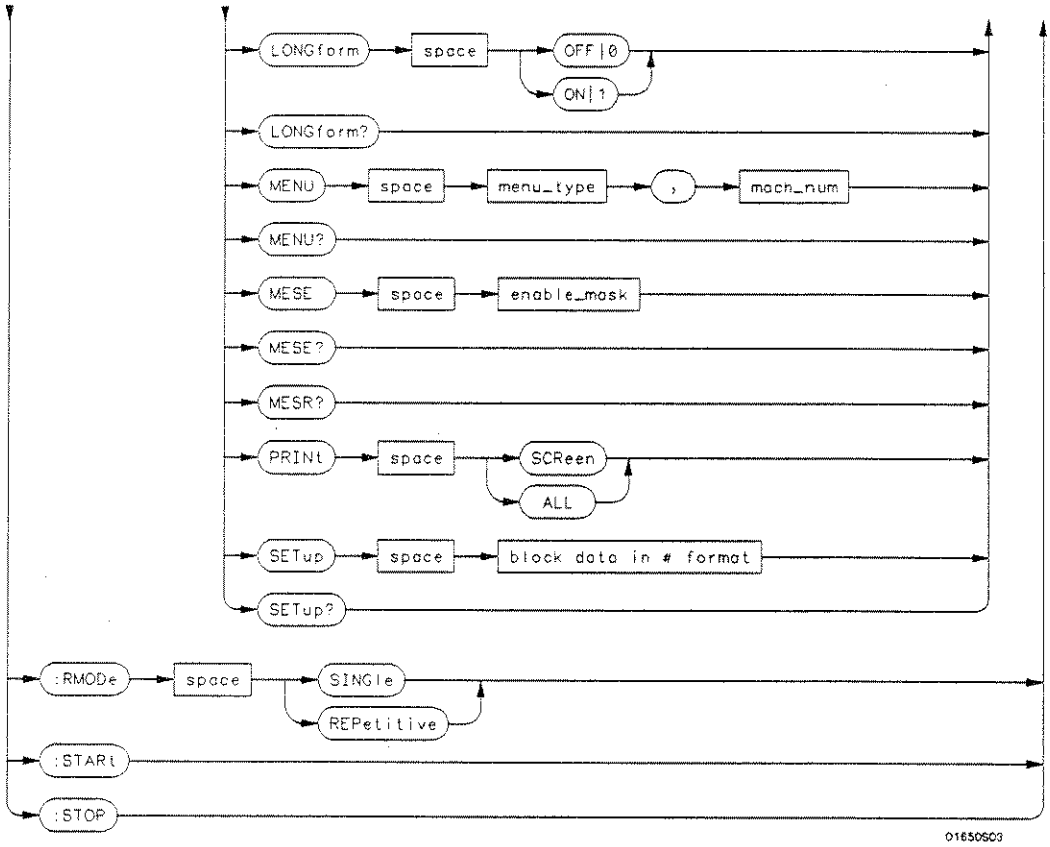
In addition to the system commands, there are three run control commands. These commands are:

- RMODe
- STARt
- STOP

The run control commands can be called at anytime and also control the basic operation of the logic analyzer. These commands are at the same level in the command tree as SYSTem; therefore they are not preceded by the :SYSTem header.



P/O Figure 5-1. System Commands Syntax Diagram



value = *integer, 0 to 255.*
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.*
block_data = *data in IEEE 488.2 # format.*
string = *string of up to 60 alphanumeric characters.*

Figure 5-1. System Commands Syntax Diagram

ARMBnc command/query

The ARMBnc command selects the source that will generate the arm out signal that will appear on the rear panel BNC labelled External Trigger Out.

The ARMBnc query returns the source currently selected.

Command Syntax: :SYSTem:ARMBnc {MACHine{1|2}|NONE}

Example: OUTPUT XXX;":SYSTem:ARMBnc MACHine1"

Query Syntax: :SYSTem:ARMBnc?

Returned Format: [:SYSTem:ARMBnc]MACHine{1|2|NONE}<NL>

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

DATA command/query

The DATA command allows you to send and receive acquired data to and from a controller in block form. This is useful for saving block data for:

- Re-loading to the logic analyzer
- Processing data later
- Processing data in the controller.

The block data of the DATA command and query is broken down into bytes, byte positions and descriptions. This is intended primarily for processing of data in the controller.

Note

Do not change the block data in the controller if you intend to send the block data back into the logic analyzer for later processing. Changes made to the block data in the controller may cause unpredictable results in the logic analyer.

The DATA command transmits the acquisition memory data from the controller to the HP 1650A/51A logic analyzer.

Note

The data sent by the DATA command is dependent on the exact logic analyzer configuration present when the data was sent to the controller in order to be valid. One way to ensure the exact setup is to send in a configuration using the SETup command.

DATA

The block data consists of 14506 bytes containing information captured by the acquisition chips. The information must be in one of four formats, depending on the type of data captured. Since no parameter checking is performed, out-of-range values could cause instrument lockup; therefore, care should be taken when transferring the data string into the HP 1650A/51A.

The DATA query returns the block data. The block data will be in one of four formats which are explained in the following pages.

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

Example: OUTPUT XXX,":SYSTem:DATA" <block data in # format>

where:

<block data in # format> ::= data information

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 one data section.

<block length specifier> <data_section>

The block length specifier is defined as follows:

#8<length>

where:

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

For example, if the total length of the block is 14522 bytes, the block length specifier would be "#800014522" 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 ID code (31 for the HP 1650A/51A logic analyzers)
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, don't forget to add the length of the section headers.

DATA

Example:

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

Data Command Configuration

The DATA command for the HP 1650A/51A logic analyzer consists of 16 bytes of section header and 14506 bytes of data, organized in one of four ways depending on the type of data. The four data types are :

- State (no tagging)
- State (with tagging)
- Glitch timing
- Transitional timing

Data Preamble Description

In general, the block data is set up as 160 bytes of preamble information, followed by 1024 lines of 14 bytes of information each, followed by ten reserved bytes. The preamble gives information for each analyzer describing the amount and type of data captured, where the trace point occurred in the data, which pods are assigned to which analyzer, etc.

Each line consists of two bytes (16 bits) of status for Analyzer 1, two bytes (16 bits) of status for Analyzer 2, then five sets of two bytes (16 bits) of information (either data, count, or glitch) corresponding to the five 16 bit pods of the HP 1650A. In the HP 1651A, the status and format for the sets of bytes are the same, but the data is not valid on pods 3, 4, and 5.

Note

One analyzer's information is independent of the other analyzer's information. In other words, on each line, one analyzer may contain data information for a timing machine, while the other analyzer may contain count information for a state machine with time tags enabled. The preamble for each analyzer describes what the information for each line contains for that analyzer. Therefore, when describing the different formats that data may contain below, keep in mind that this format pertains only to those pods that are assigned to the analyzer of the specified type. The other analyzer's data is TOTALLY independent and conforms to its own format.

Byte Position	
	16 bytes - Section Header
1	10 bytes - section name, "DATA "(six trailing spaces)
11	1 byte - reserved (always 0)
12	1 bytes - compatibility ID (31 for HP 1650A/51A)
13	4 bytes - length (always 14506 for HP 1650A/51A)
	160 bytes - Preamble
17	2 bytes - Instrument ID (always 1650 for both the HP 1650A and HP 1651A)
19	2 bytes - Revision Code
21	78 bytes - Analyzer 1 Data Information

Note

The values stored in the preamble represent the captured data currently stored in this structure and not what the current configuration of the analyzer is. For example, the mode of the data may be STATE with tagging, while the current setup of the analyzer is TIMING.

21 **1 byte** - Machine data mode(0 = OFF, 1 = STATE data with tagging, 2 = STATE data no tagging, 3 = TIMING data Glitch Mode, 4 = TIMING data Transitional Mode)

22 **1 byte** - List of pods in this analyzer

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	bit 0
unused	unused	Pod 1	Pod 2	Pod 3	Pod 4	Pod 5	unused

where:

a "1" in a given location means that this pod is assigned to this analyzer.

23 **1 byte** - Master chip in this analyzer - When several chips are grouped together in a single analyzer, one chip is designated as a master chip. This variable is used to hold this master value. A value of 4 represents POD 1, 3 for POD 2, 2 for POD 3, 1 for POD 4, and 0 for POD 5.

24 **1 byte** - Reserved

25 **10 bytes** - Number of rows of valid data for this analyzer - Indicates the number of rows of valid data for each of the five pods. Two bytes are used to store each pod value, with the first 2 bytes used to hold POD 5 value, the next 2 for POD 4 value, etc.

35 **1 byte** - Trace point seen in this analyzer - Was a trace point seen (value = 1) or forced (value = 0)

36 **1 byte** - Reserved

37 10 bytes - Trace point location for this analyzer - Indicates the row number in which the trace point was found for each of the five pods. Two bytes are used to store the value in a manner similar to the number of valid rows described above.

47 4 bytes - Time from arm to trace point for this analyzer - Time elapsed from the arm of this machine to the trace point of this machine (in 40 ns units). A value of -1 indicates counter overflow.

51 1 byte - Armer of this analyzer - Indicates what armed this analyzer (1 = RUN, 2 = BNC, 3 = other machine)

52 1 byte - Devices armed by this analyzer - Set of devices armed by this machine

-- unused --	BNC OUT	MACHINE B	MACHINE A
Bits 7 through 3	Bit 2	Bit 1	Bit 0

A "1" in a given bit position implies that this analyzer arms that device, while a "0" means the device is not armed by this analyzer.

53 4 bytes - Sample period for this analyzer (timing only) - Sample period at which data was acquired. Value represents the number of nanoseconds per sample.

57 4 bytes - Delay for this analyzer (timing only) - Delay at which data was acquired. Value represents the amount of delay in nanoseconds.

61 1 byte - Time tags on (state with tagging only) - In state tagging mode, was the data captured with time tags (value = 1) or state tags (value = 0).

62 1 byte - Reserved

DATA

63	**5 bytes** - Demultiplexing (state only) - For each of the five pods (first byte is POD 5, fifth byte is POD 1) in a state machine, describes multiplexing of each of the five pods. (0 = NO DEMUX, 1 = TRUE DEMUX, 2 = MIXED CLOCKS).
68	**1 byte** - Reserved
69	**20 bytes** - POD trace point adjustment - for each of five pods (first four bytes are Pod 5, last four bytes are Pod 1) - the number of nanoseconds that are to be subtracted from the trace point described above to get the actual trace point value.
89	**10 bytes** - Reserved
99 - 176	**78 bytes** - Analyzer 2 Data Information (same as analyzer 1 above)

Data Body Description

The Data Body consists of 14336 bytes (14 bytes * 1024 rows). The data contained in the data section will appear in one of four forms depending on the mode in which it was acquired. The four modes are:

- State Data without tags
- State Data with either time or state tags
- Timing Glitch Data
- Transitional Timing Data

The following four sections describe the four data modes that may be encountered.

State Data (no tags)

Status Bytes. In normal state mode, the status bytes provide no information.

Information Bytes. In state acquisition with no tags, data is obtained from the target system with each clock and checked with the sequencer. If the state does not match the sequencer qualifer, it is checked against the prestore qualifier. If it matches the prestore qualifier, then it is placed in the prestore buffer. If the state does not match either the sequencer qualifier or the prestore qualifier, it is discarded. If the state matches a state that should be stored, it is placed into the memory.

Example:

MACH A	MACH B	Pod 5	Pod 4	Pod 3	Pod 2	Pod 1 *
Status	Status	Data	Data	Data	Data	Data
Status	Status	Data	Data	Data	Data	Data
Status	Status	Data	Data	Data	Data	Data
Status	Status	Data	Data	Data	Data	Data

* These labels are not part of the data. They only show how the data is organized.

State Data (with either time or state tags)

Status Bytes. In state tagging mode, the status bytes indicate whether a given row of the data is a data line, a count (tag) line, or a prestore line.

Bit 2 is the Data vs. Count bit. If Bit 2 is set, this row of information contains tags. If Bit 2 is clear, this row of information contains actual acquisition data as obtained from the target system. The counts are relative counts from one state to the one previous and therefore, the count for the first state in the data structure is invalid, and should be ignored. The count is stored in floating point format with 11 bits of mantissa and 5 bits of exponent (EEEEEMMMMMMMMMMM). The actual value of the count is given by the equation :

Count = mantissa * (2 ** exponent)

where: mantissa = MMMMMMMMMMM
exponent = EEEEE

If time tagging is on, the count value represents the number of 40 nanosecond units that have elapsed between the two stored states.

In the case of state tagging, the count represents the number of qualified states that were encountered between the stored states.

Bit 3 is the Prestore vs. Normal store bit. If Bit 3 is set, this data row and its corresponding count row represent prestored information. The count, therefore, should be ignored.

Information Bytes. In the State acquisition mode with tags, data is obtained from the target system with each clock and checked with the sequencer. If the state does not match the sequencer qualifier, it is checked against the prestore qualifier. If it matches the prestore qualifier, then it is placed in the prestore buffer. If the state does not match either the sequencer qualifier or the prestore qualifier, it is discarded.

If a state matches the sequencer qualifiers, the prestore buffer is checked. If there are any states in the prestore buffer at this time, these prestore states are first placed in memory, along with a dummy count row. After this check, the qualified state is placed in memory, followed by the count row which specified how many states (or 40 ns units) have elapsed since the last stored state. If this is the first stored state in memory, then the count information that is stored should be discarded.

Example: MACH A MACHB Pod 5 Pod 4 Pod 3 Pod 2 Pod 1 **

Status	Status	Data	Data	Data	Data	Data
Status	Status	Count	Count	Count	Count	Count
Status	Status	Prestore	Prestore	Prestore	Prestore	Prestore
Status	Status	*	*	*	*	*
Status	Status	Data	Data	Data	Data	Data
Status	Status	Count	Count	Count	Count	Count

* = Invalid data
** These labels are not part of the data. They only show how the data is organized.

The rows alternate between data and count. For example, a count in row (n) provides the relative time between data in rows (n-1) and (n-3).

Timing Glitch Data

Status Bytes. In glitch timing mode, the status bytes indicate whether a given row in the data contains actual acquisition data information or glitch information.

Bit 1 is the Data vs. Glitch bit. If Bit 1 is set, this row of information contains glitch information. If Bit 1 is clear, then this row contains actual acquisition data as obtained from the target system.

Information Bytes. In the Glitch timing mode, the target system is sampled at every sample period. The data is then stored in memory and the glitch detectors are checked. If a glitch has been detected between the previous sample and the current sample, the corresponding glitch bits are set. The glitch information is then stored. If this is the first stored sample in memory, then the glitch information stored should be discarded.

Example:

MACH A	MACH B	Pod 5	Pod 4	Pod 3	Pod 2	Pod 1 *
Status	Status	Data	Data	Data	Data	Data
Status	Status	Glitch	Glitch	Glitch	Glitch	Glitch
Status	Status	Data	Data	Data	Data	Data
Status	Status	Glitch	Glitch	Glitch	Glitch	Glitch

* These labels are not part of the data. They only show how the data is organized.

Transitional Timing Data

Status Bytes. In transitional timing mode, the status bytes indicate whether a given row in the data contains acquisition information or transition count information.

Bits 10 and 9 are the Data vs. Count bits for pod 5. Bits 10 and 9 of the status for transitional timing provide enough information to determine whether the information for pod 5 on a given row is acquisition data, the start of a count block, or the middle of a count block. There are three possible values these two bits may have:

00 - This row contains part of a count, but is not the first row of a count.
01 - This row contains the first word of a count.

10 - This row contains actual acquisition data as obtained from the target system.

Bits 8 and 7 - Data vs. Count Bits for pod 4 (see bits 10 and 9 above)

Bits 6 and 5 - Data vs. Count Bits for pod 3(see bits 10 and 9 above)

Bits 4 and 3 - Data vs. Count Bits for pod 2(see bits 10 and 9 above)

Bits 2 and 1 - Data vs. Count Bits for pod 1(see bits 10 and 9 above)

Information Bytes. In the Transitional timing mode the logic analyzer performs the following steps to obtain the information bytes:

1. Four samples of data are taken at 10 nanosecond intervals. The data is stored and the value of the last sample is retained.

2. Four more samples of data are taken. If any of these four samples differ from the last sample of the step 1, then these four samples are stored and the last value is once again retained.

3. If all four samples of step 2 are the same as the last sample taken in step 1, then no data is stored. Instead, a counter is incremented. This process will continue until a group of four samples is found which differs from the retained sample. At this time, the count will be stored in the memory, the counters reset, the current data stored, and the last sample of the four once again retained for comparison.

Note

The stored count indicates the number of 40 ns intervals that have elapsed between the old data and the new data.

The rows of the acquisition data may, therefore, be either four rows of data followed by four more rows of data, or four rows of data followed by four rows of count. Rows of count will always be followed by four rows of data except for the last row, which may be either data or count.

Note

This process is performed on a pod by pod basis. The individual status bits will indicate what each pod is doing.

Example: MACH A MACH B Pod 5 Pod 4 Pod 3 Pod 2 Pod1 *

Status	Status	Data	Data	Data	Data	Data
Status	Status	Data	Data	Data	Data	Data
Status	Status	Data	Data	Data	Data	Data
Status	Status	Data	Data	Data	Data	Data
Status	Status	Data	Count	Count	Data	Data
Status	Status	Data	Count	Count	Data	Data
Status	Status	Data	Count	Count	Data	Data
Status	Status	Data	Count	Count	Data	Data
Status	Status	Count	Data	Data	Count	Data
Status	Status	Count	Data	Data	Count	Data
Status	Status	Count	Data	Data	Count	Data
Status	Status	Count	Data	Data	Count	Data
Status	Status	Data	Data	Count	Data	Data
Status	Status	Data	Data	Count	Data	Data
Status	Status	Data	Data	Count	Data	Data
Status	Status	Data	Data	Count	Data	Data
Status	Status	Data	Data	Data	Data	Data
Status	Status	Data	Data	Data	Data	Data
Status	Status	Data	Data	Data	Data	Data
Status	Status	Data	Data	Data	Data	Data

* These labels are not part of the data. They only show how the data is organized.

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 60 alphanumeric characters

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

ERRor query

The ERRor query returns the oldest error number from the error queue. A complete list of error numbers for the HP 1650A/51A is shown in appendix C. If no errors are present in the error queue, a zero is returned.

Query Syntax: :SYSTem:ERRor?

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

Example:

```
10 OUTPUT XXX;":SYSTEM:ERROR?"
20 ENTER XXX;Error
30 PRINT Error
40 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.

KEY command/query

The KEY command allows you to simulate pressing a specified front-panel key. Key commands may be sent over the bus in any order that is legal from the front panel. Be sure the instrument is in a desired setup before executing the KEY command. Key codes range from 0 to 36 with 99 representing no key (returned at power-up). See Table 5-1 for key codes.

Note

The external KEY buffer is only two keys deep; therefore, attempting to send KEY commands too rapidly will cause a KEY buffer overflow error to be displayed on the HP 1650A/51A screen.

The KEY query returns the key code for the last front- panel key pressed or the last simulated key press over the bus.

Command Syntax: :SYSTem:KEY <key_code>

where:

<key_code> ::= integer from 0 to 36

Example: OUTPUT XXX;":SYSTEM:KEY 24"

KEY

Query Syntax: :SYSTem:KEY?

Returned Format: [:SYSTem:KEY] <key_code> <NL>

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

Table 5-1. Key codes

Key Value	HP 1650A/51A Key	Key Value	HP 1650A/51A Key
0	RUN	19	D
1	STOP	20	E
2	unused	21	F
3	SELECT	22	unused
4	CHS	23	unused
5	Don't Care	24	Knob left
6	0	25	Knob right
7	1	26	L/R Roll
8	2	27	U/D Roll
9	3	28	unused
10	4	29	unused
11	5	30	unused
12	6	31	"."
13	7	32	Clear Entry
14	8	33	FORMAT
15	9	34	TRACE
16	A	35	DISPLAY
17	B	36	I/O
18	C	99	Power Up

LER (LCL Event Register) query

The LER query allows the LCL (local) 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: :SYSTem:LER?

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

Example:

```
10 DIM Event$[100]
20 OUTPUT XXX;":SYSTem:LER?"
30 ENTER XXX;Event$
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: :SYSTem:LOCKout {{ON|1}|{OFF|0}}

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

Query Syntax: :SYSTem:LOCKout?

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

Example:

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

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 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
```

MENU

command/query

The MENU command puts a menu on the display. The MENU query returns the current menu selection.

Command Syntax: :SYSTem:MENU <menu_type>,<mach_num>

where:

<menu_type> ::= {SCONfig | FORMat | TRACe | DISPlay}
<mach_num> ::= {0 | 1 | 2}
0 ::= mixed mode
1 ::= analyzer 1
2 ::= analyzer 2

Example:
```
OUTPUT XXX;"SYSTem:MENU FORMat,1"
```

Query Syntax: :SYSTem:MENU?

Returned Format: [:SYSTem:MENU] <menu_type>,<mach_num>

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

MESE command/query

The MESE commands sets the Module Event Status Enable Register bits. The MESE register contains a mask value for the bits enabled in the MESR register. A one in the MESE will enable the corresponding bit in the MESR, a zero will disable the bit.

The MESE query returns the current setting.

Refer to table 5-2 for information about the Module Event Status Enable register bits, bit weights, and what each bit masks for the logic analyzer.

Command Syntax: :MESE <enable_mask>

where:

<enable mask> ::= integer from 0 to 255

Example: OUTPUT XXX;":MESE 1"

MESE

Query Syntax: :MESE?

Returned Format: [:MESE] <enable_mask> <NL>

Example:
```
10 OUTPUT XXX;":MESE?"
20 ENTERXXX; Mes
30 PRINT Mes
40 END
```

Table 5-2. Module Event Status Enable Register

Module Event Status Enable Register (A "1" enables the MESR bit)		
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-Run until satisified
0	1	MC-Measurement complete

MESR

query

The MESR query returns the contents of the Module Event Status register.

Note

Reading the register clears the Module Event Status Register.

Table 5-3 shows each bit in Module Event Status Register and their bit weights for the logic analyzer. When you read the MESR, the value returned is the total bit weights of all bits that are set at the time the register is read.

Query Syntax: :MESR?

Returned Format: [MESR] < status > < NL >

where:

< status > ::= 0 to 255

Example:

```
10 OUTPUT XXX;":MESR?"
20 ENTER XXX; Mer
30 PRINT Mer
40 END
```

MESR

Table 5-3. Module Event Status Register

Module Event Status Register		
Bit	**Weight**	**Condition**
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	1 = Run until satisified 0 = Run until not satisified
0	1	1 = Measurement complete

PRINt command

The PRINt command initiates a print of the screen or print all over the RS-232C bus. The PRINt parameters SCReen or ALL specify how the screen data is sent to the controller. PRINt SCReen transfers the data to the controller in a printer specific graphics format. PRINt ALL transfers the data in a raster format for the following menus:

- State and Timing Format menus
- Disc menu
- State and Timing Symbol menus
- State Listing menu
- State Trace

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

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

RMODe

command/query

The RMODe command is a run control command that specifies the run mode for logic analyzer. It is at the same level in the command tree as SYSTem; therefore, it is not preceded by :SYSTem.

The query returns the current setting.

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
```

SETup — command/query

The SETup command allows you to send and receive instrument configuration data to and from a controller in block form. The SETup command configures the logic analyzer as defined by the block data sent by the controller.

The SETup query returns a block of data that contains the current configuration to the controller.

Command syntax: :SYStem:SETup <block data in # format>

Example: OUTPUT XXX;"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 n>

The block length specifier is defined as follows:

#8<length>

where:

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

For example, if the total length of the block (all sections) is 2720 bytes, the block length specifier would be "#800002720" since that length is 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 code (31 for the logic analyzer)
4 bytes for the length of the section data in bytes

The section data format varies for each section and may be of 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 not to forget to add the length of the section headers.

Example:

```
10 DIM Block$[32000]  ! allocate enough memory for block data
20 DIM Specifier$[2]
30 INTEGER Blocklength
40 OUTPUT XXX;":SYSTEM:HEAD OFF"
50 OUTPUT XXX:":SYSTEM:SETUP?"  ! send setup query
60  ENTER XXX USING "#,2A";Specifier$  ! read in #8
70 ENTER XXX USING "#,8D";Blocklength  ! read in block length
80 Image_spec$ = "#,"&VAL$(Blocklength)&"A"
90 ENTER XXX USING Image_spec$;Block$  ! read in data
100 ENTER XXX USING"#,B";Terminator    ! enter <NL>
110 END
```

STARt command

The STARt command is a run control command that starts the logic analyzer running in the specified run mode (see RMODe). The STARt command is on the same level in the command tree as SYSTem; therefore, it is not preceded by :SYSTem.

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.

Command Syntax: :STARt

Example: OUTPUT XXX;":START"

STOP command

The STOP command is a run control command that stops the logic analyzer. The STOP command is on the same level in the command tree as SYSTem; therefore, it is not preceded by :SYSTem.

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.

Command Syntax: :STOP

Example: OUTPUT XXX;":STOP"

MMEMory Subsystem 6

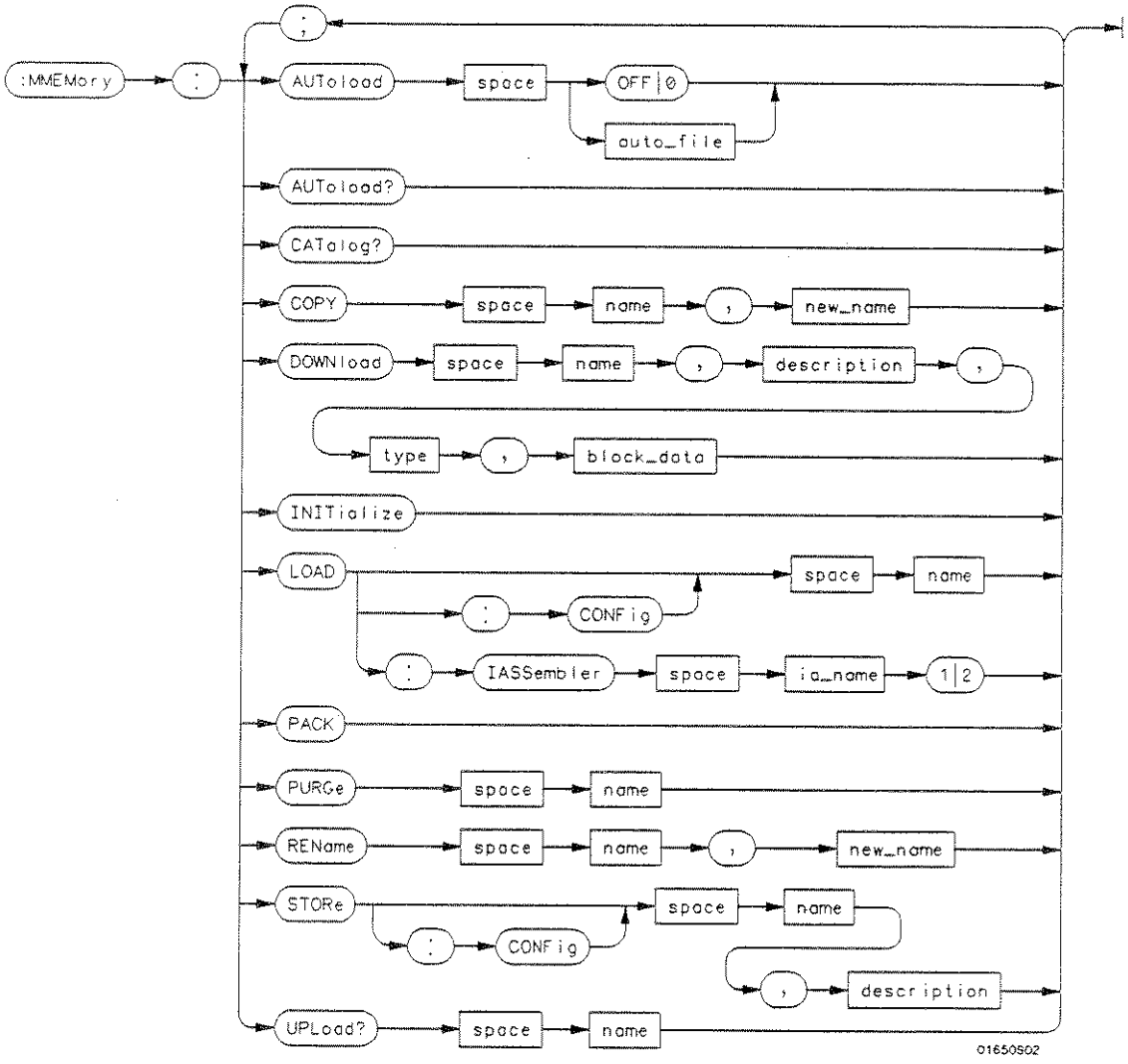
Introduction

MMEMory subsystem commands provide access to the disc drive. The MMEMory Subsystem commands are:

- AUToload
- CATalog
- COPY
- DOWNload
- INITialize
- LOAD
- PACK
- PURGe
- REName
- STORe
- UPLoad

Note

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.



P/O Figure 6-1. MMEMory Subsystem Commands Syntax Diagram

auto_file = *string of up to 10 alphanumeric characters representing a valid file name.*
name = *string of up to 10 alphanumeric characters representing a valid file name.*
description = *string of up to 32 alphanumeric characters.*
type = *integer, refer to table 6-1.*
block_data = *data in IEEE 488.2 # format.*
ia_name = *string of up to 10 alphanumeric characters representing a valid file name.*
new_name = *string of up to 10 alphanumeric characters representing a valid file name.*

Note

Refer to "Disc Operations" in Chapter 5 of the HP 1650A/1651A Logic Analyzers Reference manual for a description of a valid file name.

P/O Figure 6-1. MMEMory Subsystem Commands Syntax Diagram

AUToload command/query

The AUToload command controls the autoload feature which designates a configuration file to be loaded automatically the next time the instrument is turned on. The OFF parameter (or 0) disables the autoload feature. When a string parameter is specified it represents the desired autoload 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.

Command Syntax: :MMEMory:AUToload {{OFF|0}|<auto_file>}

where:

<auto_file> ::= string of up to 10 alphanumeric characters

Examples:
```
OUTPUT XXX;":MMEMORY:AUTOLOAD OFF"
OUTPUT XXX;":MMEMORY:AUTOLOAD 'FILE1'"
OUTPUT XXX;":MMEMORY:AUTOLOAD 'FILE2'"
```

Query Command: :MMEMory:AUToload?

Returned Format: [:MMEMory:AUToload] {0|<auto_file>}<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 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 description.

Query Syntax: :MMEMory:CATalog?

Returned Format: [:MMEMory:CATalog] <block_size>,.<block data>

where:

<block data> ::= <filename> <file type> <file description> ...
<block_data> ::= #8nnnnnnnn (N...N = # of bytes)

Example:

```
10 DIM File$[51]
20 DIM Specifier$[2]
30 OUTPUT XXX;":SYSTEM:HEAD OFF"
40 OUTPUT XXX;":MMEMORY:CATALOG?" !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 the contents of a file to a new file. The two <name> parameters are the filenames. The first parameter specifies the source file. The second specifies the destination file. An error is generated if the source file doesn't exist, if the destination file already exists, or any other disc error is detected.

Command Syntax: :MMEMory:COPY <name>,<name>

where:

<name> :: = string of up to 10 alphanumeric characters representing a valid file name

Examples: To copy the contents of "FILE1" to "FILE2":

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

DOWNload command

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

Table 6-1 lists the file types for the <type> parameter.

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

where:

<name> ::= string of up to 10 alphanumeric characters representing a valid file name
<description> ::= string of up to 32 alphanumeric characters
<type> ::= integer (see Table 6-1)
<block_data> ::= contents of file in block data format

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

Table 6-1. File Types

File	File Type
HP 1650/1 SYSTEM	-16383
1650/1 CONFIG	-16096
AUTOLOAD TYPE	-15615
INVERSE ASSEMBLER	-15614
TEXT TYPE	-15610

INITialize command

The INITialize command formats the disc.

Command Syntax: :MMEMory:INITialize

Example:

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

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 file from the disc into the analyzer. The [:CONfig] specifier is optional and has no effect on the command. The <name> parameter specifies the filename that will be loaded into the logic analyzer.

Note

Any previous setups and data in the instrument are replaced by the contents of the configuration file.

Command Syntax: :MMEMory:LOAD[:CONfig] <name>

where:

<name> ::= string of up to 10 alphanumeric characters representing a valid file name

Examples:
```
OUTPUT XXX;":MMEMORY:LOAD:CONFIG 'FILE__'"
OUTPUT XXX;":MMEMORY:LOAD 'FILE__'"
OUTPUT XXX;":MMEM:LOAD:CONFIG 'FILE_A'"
```

LOAD :IASSembler command

This variation of the LOAD command allows inverse assembler files to be loaded into analyzer 1 or analyzer 2 of the HP 1650A/51A. The <IA_name> parameter specifies the inverse assembler filename. The parameter after the <IA_name> parameter specifies into which machine the inverse assembler is loaded.

Note

Inverse assembler files should only be loaded into the state analyzer. If an inverse assembler file is loaded into the timing analyzer no error will be generated: however, it will not be accessible.

Command Syntax: :MMEMory:LOAD:IASSembler <IA_name>,{1|2}

where:

<IA_name> ::= string of up to 10 alphanumeric characters representing a valid file name

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

PACK command

The PACK command packs the files on a disc in the disc drive.

Command Syntax: :MMEMory:PACK

Example: OUTPUT XXX;":MMEMORY:PACK"

PURGe command

The PURGe command deletes a file from the disc. The <name> parameter specifies the filename to be deleted.

Command Syntax: :MMEMory:PURGe <name>

where:

<name> ::= string of up to 10 alphanumeric characters representing a valid file name

Examples: OUTPUT XXX;":MMEMORY:PURGE 'FILE1'"

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 on the disc. 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>,<new_name>

where:

<name> ::= string of up to 10 alphanumeric characters representing a valid file name
<new_name> ::= string of up to 10 alphanumeric characters representing a valid file name

Examples: OUTPUT XXX;":MMEMORY:RENAME 'OLDFILE','NEWFILE'"

STORe [:CONFig] command

The STORe command stores a configuration onto a disc. The [:CONFig] specifier is optional and has no effect on the command. The <name> parameter specifies the file to be stored to the disc. The <description> parameter specifies the file description.

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

where:

<name> ::= string of up to 10 alphanumeric characters representing a valid file name
<description> ::= string of up to 32 alphanumeric characters

Example: OUTPUT XXX;":MMEM:STORE 'DEFAULTS','DEFAULT SETUPS'"

UPLoad

query

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

Query Syntax: :MMEMory:UPLoad? <name>

where:

<name> ::= string of up to 10 alphanumeric characters representing a valid file name

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

Example:

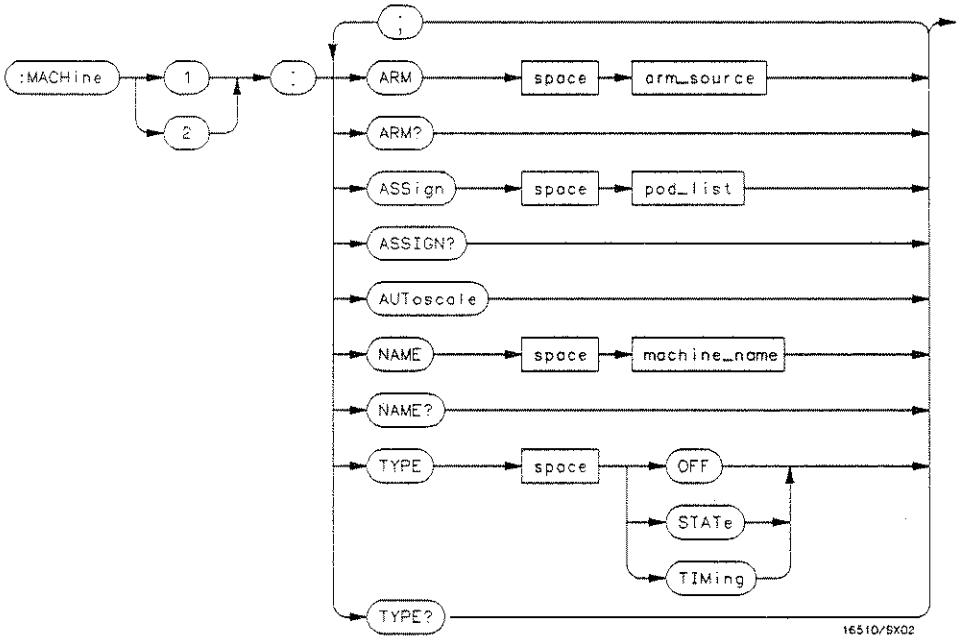
```
10 DIM Block$[32000]  !allocate enough memory for block data
20 DIM Specifier$[2]
30 OUTPUT XXX;":SYSTEM HEAD OFF"
40 OUTPUT XXX;":MMEMORY:UPLOAD? 'FILE1'"  !send upload query
50 ENTER XXX USING "#,2A";Specifier$  !read in #8
60 ENTER XXX USING "#,8D";Length  !read in block length
70 ENTER XXX USING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

MACHine Subsystem 7

Introduction

The MACHine Subsystem contains the commands available for the State/Timing Configuration menu. These commands are:

- ARM
- ASSign
- AUToscale (Timing Analyzer only)
- NAME
- TYPE



P/O Figure 7-1. Machine Subsystem Syntax Diagram

arm_source = {*RUN* | *MACHine* {*1* | *2*}}
pod_list = {*NONE* | <*pod_num*>[, ..., <*pod_num*>]}
pod_num = {*1* | *2* | *3* | *4* | *5*}
machine_name = *string of up to 10 alphanumeric characters*

P/O Figure 7-1. Machine Subsystem Syntax Diagram

MACHine selector

The MACHine <N> selector specifies which of the two analyzers (machines) available in the HP 1650A/51A the commands or queries following will refer to. Since the MACHine <N> command is a root level command, it will normally appear as the first element of a compound header.

Command Syntax: :MACHine <N>

where:

<N> ::= machine 1 or 2

Example: OUTPUT XXX; ":MACHine1:NAMe 'DRAMTEST'"

ARM command/query

The ARM command specifies the arming source of the specified analyzer (machine).

The ARM query returns the source that the current analyzer (machine) will be armed by.

Command Syntax: :MACHine {1|2}:ARM <arm_source>

where:

<arm_source> ::= {RUN|MACHine{1|2}|BNC}

Example:
```
OUTPUT XXX;":MACHine1:ARM MACHine2"
```

Query Syntax: :MACHine {1|2}:ARM?

Returned Format: [:MACHine {1|2}:ARM] <arm_source><NL>

Example:
```
10 DIM String$ [100]
20 OUTPUT XXX; ":MACHINE1:ARM?"
30 ENTER XXX; String$
40 PRINT String$
50 END
```

ASSign — command/query

The ASSign command assigns pods to a particular analyzer (machine). The ASSign query returns which pods are assigned to the current analyzer (machine).

Command Syntax: :MACHine {1|2}:ASSign <pod_list>

where:

<pod_list> ::= {NONE|<pod#>[,<pod #>, ... ,<pod #>]}
<pod#> ::= {1|2|3|4|5}

Example:
```
OUTPUT XXX;":MACHine1:ASSign 5, 2, 1"
```

Query Syntax: :MACHine {1|2}:ASSign?

Returned Format: [:MACHINE {1|2}:ASSign] <pod_list><NL>

Example:
```
10 DIM String$ [100]
20 OUTPUT XXX;":MACHINE1:ASSIGN?"
30 ENTER XXX;String$
40 PRINT String$
50 END
```

AUToscale command

The AUToscale command causes the current analyzer (machine) to autoscale if the current machine is a timing analyzer. If the current machine is not a timing analyzer, the AUToscale command is ignored.

AUToscale is an Overlapped Command. Overlapped Commands allow execution of subsequent commands while the logic analyzer operations initiated by the Overlapped Command are still in progress. Command overlapping can be avoided by using the *OPC and *WAI commands in conjunction with AUToscale (refer to Chapter 4).

Note

When the AUToscale command is issued, existing timing analyzer configurations are erased and the other analyzer is turned off.

Command Syntax: :MACHine {1|2}:AUToscale

Example: OUTPUT XXX;":MACHine1:AUToscale"

NAME

command/query

The NAME command allows you to assign a name of up to 10 characters to a particular analyzer (machine) for easier identification. The NAME query returns the current analyzer name as an ASCII string.

Command Syntax: :MACHine {1|2}:NAME <machine_name>

where:

<machine_name> ::= string of up to 10 alphanumeric characters

Example:
```
OUTPUT XXX;":MACHine1:NAME 'DRAMTEST'"
```

Query Syntax: :MACHine {1|2}:NAME?

Returned Format: [MACHine {1|2}:NAME] <machine name><NL>

Example:
```
10 DIM String$ [100]
20 OUTPUT XXX;":MACHINE1:NAME?"
30 ENTER XXX;String$
40 PRINT String$
50 END
```

TYPE

command/query

The TYPE command specifies what type a specified analyzer (machine) will be. The analyzer types are state or timing. The TYPE command also allows you to turn off a particular machine.

Note

Only one of the two analyzers can be specified as a timing analyzer at one time.

The TYPE query returns the current analyzer type for the specified analyzer.

Command Syntax: :MACHine {1|2}:TYPE <analyzer type>

where:

<analyzer type> ::= {OFF|STATe|TIMing}

Example:
```
OUTPUT XXX;":MACHine1:TYPE STATE"
```

Query Syntax: :MACHine {1|2}:TYPE?

Returned Format: [:MACHine {1|2}:TYPE] <analyzer type><NL>

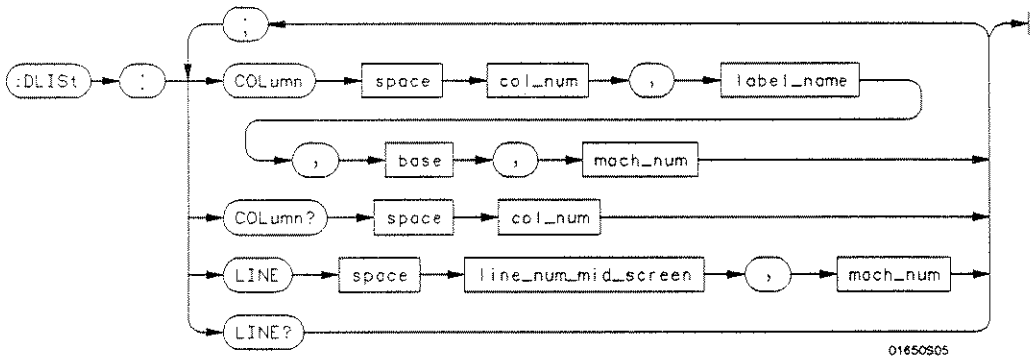
Example:
```
10 DIM String$ [100]
20 OUTPUT XXX;":MACHINE1:TYPE?"
30 ENTER XXX;String$
40 PRINT String$
50 END
```

DLISt Subsystem 8

Introduction

The DLISt (dual list) Subsystem contains the commands in the dual state listing menu. These commands are:

- COLumn
- LINE



col_num = *integer from 1 to 8*
label_name = *a string of up to 6 alphanumeric characters*
base = {*BINary*|*HEXacecimal*|*OCTal*|*DECimal*|*ASCii*|*SYMBol*}
mach_num = {*1*|*2*}
line_num_mid_screen = *a real number from - 511 to + 511*

Figure 8-1. DLISt Subsystem Syntax Diagram

DLISt

selector

The DLIST selector (dual list) is used as part of a compound header to access those settings normally found in the Dual State Listing menu. The dual list displays data when two state analyzers are run simultaneously.

Command Syntax: :DLISt

Example: OUTPUT XXX;":DLIST:LINE 0,1"

COLumn command/query

The COLumn command allows you to configure the state analyzer list display by assigning a label name and base to one of eight vertical columns in the menu. The machine number parameter is required since the same label name can occur in both state machines at once. A column number of 1 refers to the left most column. When a label is assigned to a column it replaces the original label in that column. The label originally in the specified column is placed in the column the specified label is moved from.

When the label name is "TAGS," the TAGS column is assumed and the next parameter must specify RELative or ABSolute. The machine number should be 1.

The COLumn query returns the column number, label name, and base for the specified column.

Command Syntax: :DLISt:COLumn <col_num>,<label_name>,<base>,<mach_num>

where:

<col_num> ::= integer from {1 - 8}
<label_name> ::= a string of up to 6 alphanumeric characters
<base> ::= {BINary|HEXadecimal|OCTal|DECimal|ASCii|SYMBol}
<mach_num> ::= {1|2}

Example:

```
OUTPUT XXX;":DLISt:COLumn 4,'DATA',HEXadecimal,1"
```

COLumn

Query Syntax: :DLISt:COLumn? <col_num>

Returned Format: [:DLISt:COLumn] <col_num>,<label_name>,<base>,<mach_num><NL>

Example:

```
10 DIM CI$[100]
20 OUTPUT XXX;":DLISt:COLumn? 4"
30 ENTER XXX;CI$
40 PRINT CI$
50 END
```

LINE command/query

The LINE command allows you to scroll the state analyzer listing vertically. The command specifies the state line number relative to the trigger that the specified analyzer will highlight at center screen.

The LINE query returns the line number for the state currently in the box at center screen and the machine number to which it belongs.

Command Syntax: :DLISt:LINE <line_num_mid_screen>,<mach_num>

where:

<line_num_mid_screen> ::= a real number from -511 to +511
<mach_num> ::= {1|2}

Example:
```
OUTPUT XXX;":DLISt:LINE 511,1"
```

Query Syntax: :DLISt:LINE?

Returned Format: [DLISt:LINE]<line_num_mid_screen>,<mach_num><NL>

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

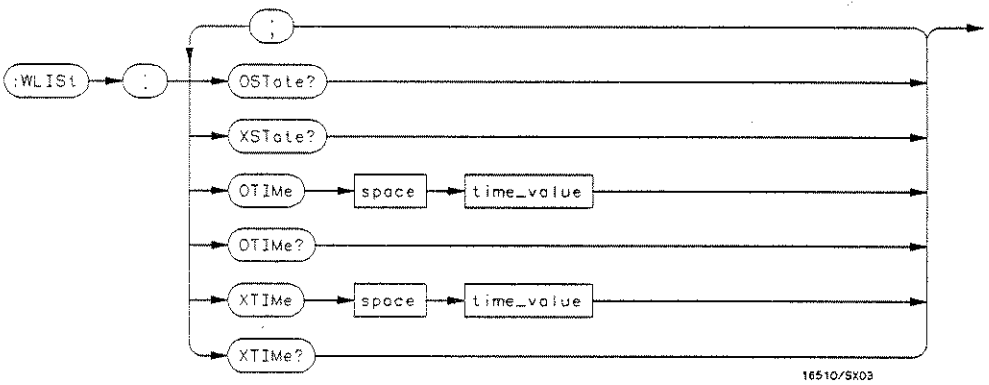
WLISt Subsystem 9

Introduction

Two commands in the WLISt Subsystem control the X and O marker placement on the waveforms portion of the Timing/State mixed mode display. These commands are XTIMe and OTIMe. The XSTate and OSTate queries return what states the X and O markers are on. Since the markers can only be placed on the timing waveforms, the queries return what state (state acquisition memory location) the marked pattern is stored in.

Note

In order to have mixed mode, one machine must be a timing analyzer, the other machine must be a state analyzer with time tagging on (use MACHine<N>:STRace:TAG TIME).



Time_value = *real number*

Figure 9-1. WLISt Subsystem Syntax Diagram

WLISt selector

The WLISt (Waveforms/listing) selector is used as a part of a compound header to access the settings normally found in the Mixed Mode menu. Since the WLISt command is a root level command, it will always appear as the first element of a compound header.

Note

The WLISt Subsystem is only available when one state analyzer (with time tagging on) and one timing analyzer are specified.

Command Syntax: :WLISt

Example: OUTPUT XXX;":WLISt:XTIMe 40.0E-6"

OSTate query

The OSTate query returns the state where the O Marker is positioned. If data is not valid, the query returns 32767.

Query Syntax: :WLISt:OSTate?

Returned Format: [:WLISt:OSTate] <state_num> <NL>

where:

<state_num> ::= integer

Example:

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

XSTate query

The XSTate query returns the state where the X Marker is positioned. If data is not valid, the query returns 32767.

Query Syntax: :WLISt:XSTate?

Example: OUTPUT XXX,":WLISt:XSTATE?

Returned Format: [:WLISt:XSTate] <state_num> <NL>

where:

<state_num> ::= integer

Example:

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

OTIMe command/query

The OTIMe command positions the O Marker on the timing waveforms in the mixed mode display. If the data is not valid, the command performs no action.

The OTIMe query returns the O Marker position in time. If data is not valid, the query returns 9.9E37.

Command Syntax: :WLISt:OTIMe <time_value>

where:

<time_value> ::= real number

Example:
```
OUTPUT XXX,":WLISt:OTIME 40.0E-6"
```

Query Syntax: :WLISt:OTIMe?

Returned Format: [:WLISt:OTIMe] <time_value><NL>

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

XTIMe

command/query

The XTIMe command positions the X Marker on the timing waveforms in the mixed mode display. If the data is not valid, the command performs no action.

The XTIMe query returns the X Marker position in time. If data is not valid, the query returns 9.9E37.

Command Syntax: :WLISt:XTIMe <time_value>

where:

<time_value> ::= real number

Example:
```
OUTPUT XXX,":WLISt:XTIMe 40.0E-6"
```

Query Syntax: :WLISt:XTIMe?

Returned Format: [:WLISt:XTIMe] <time_value><NL>

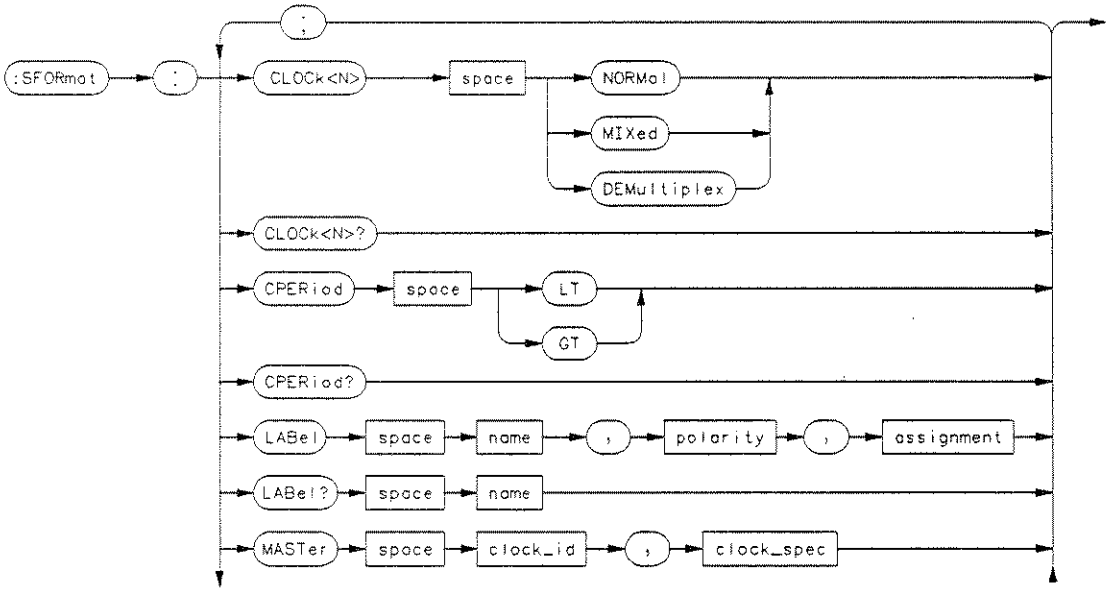
Example:
```
10 DIM Tx$[100]
20 OUTPUT XXX;":WLIST:XTIME?"
30 ENTER XXX;Tx$
40 PRINT Tx$
50 END
```

SFORmat Subsystem 10

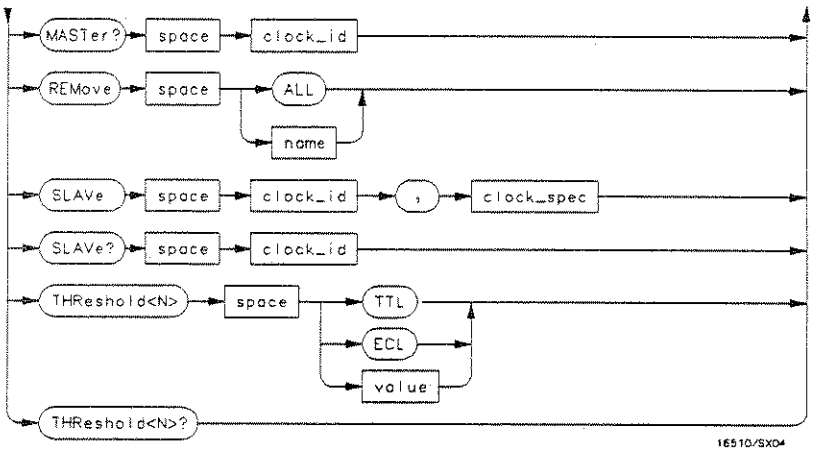
Introduction

The SFORmat Subsystem contains the commands available for the State Format Menu in the HP 1650A/51A logic analyzer. These commands are:

- CLOCk
- CPERiod
- LABel
- MASTer
- REMove
- SLAVe
- THReshold



P/O Figure 5-1. SFORmat Subsystem Syntax Diagram



<N> = {*1* | *2* | *3* | *4* | *5*}
GT = Greater Than 60 ns
LT = Less Than 60 ns
name = *string of up to 6 alphanumeric characters*
polarity = {*POSitive* | *NEGative*}
assignment = {*pod<MS>_spec[,pod<MS-1>_spec,...,pod <LS_spec>]*}
pod<MS|LS> = *pod number, MS being the highest pod number assigned and LS is the lowest pod number assigned*
_spec = *number equal to or greater than 0 (zero) but less or equal to 65535*
clock_id = {*J* | *K* | *L* | *M* | *N*}
clock_spec = {*OFF* | *RISing* | *FALLing* | *BOTH* | *LOW* | *HIGH*}
value = *voltage (real number) -9.9 to +9.9*

P/O Figure 5-1. SFORmat Subsystem Syntax Diagram

SFORmat selector

The SFORmat (State Format) selector is used as a part of a compound header to access the settings in the State Format Menu. It always follows the MACHine selector because it selects a branch directly below the MACHine level in the command tree.

Command Syntax: :MACHine{1|2}:SFORmat

Example: OUTPUT XXX;":MACHine2:SFORmat:MASTer J, RISing"

CLOCk command/query

The clock command selects the clocking mode for a given pod when the pod is assigned to the state analyzer. When the NORMal option is specified, the pod will sample all 16 channels on the master clock. When the MIXed option is specified, the upper 8 bits will be sampled by the master clock and the lower 8 bits will be sampled by the slave clock. When the DEMultiplex option is specified, the lower 8 bits will be sampled on the slave clock and then sampled again on the master clock. The master clock always follows the slave clock when both are used.

The CLOCk query returns the current clocking mode for a given pod.

Command Syntax: :MACHine{1|2}:SFORmat:CLOCk<N> <clock_mode>

where:

<N> ::= {1|2|3|4|5}
<clock_mode> ::= {NORMal|MIXed|DEMultiplex}

Example:
```
OUTPUT XXX;":MACHine1:SFORmat:CLOCk2 NORMal"
```

Query Syntax: :MACHine{1|2}:SFORmat:CLOCk<N>?

Returned Format: [:MACHine{1|2}:SFORmat:CLOCK<N>] <clock_mode><NL>

Example:
```
10 DIM String$ [100]
20 OUTPUT XXX; ":MACHINE1:SFORMAT:CLOCK2?"
30 ENTER XXX; String$
40 PRINT String$
50 END
```

CPERiod

command/query

The CPERiod (clock period) command allows you to set the state analyzer for input clock period of greater than or less than 60 ns. If the state input clock period is less than 60 ns, this setting should be set to Less Than (60 ns). If the state input clock period is greater than 60 ns, this setting should be Greater Than (60 ns). If CPERiod is set to Less Than 60 ns, tagging will be set to OFF in the STRace Subsystem. If the count is set to time or states using the TAG command, this setting will automatically be set to Greater Than because the minimum clock period when counting is 60 ns.

The CPERiod returns the current setting of clock period.

Command Syntax: :MACHine{1|2}:SFORmat:CPERiod {LT|GT}

where:

GT ::= greater than 60 ns
LT ::= less than 60 ns

Example:
```
OUTPUT XXX;":MACHine2:SFORmat:CPERiod GT"
```

Query Syntax: :MACHine{1|2}:SFORmat:CPERiod?

Returned Format: [:MACHine{1|2}:SFORmat:CPERiod] {GT|LT}

Example:
```
10 DIM String$[100]
20 OUTPUT XXX;":MACHINE2:SFORMAT:CPERIOD?
30 ENTER XXX; String$
40 PRINT String$
50 END
```

LABel

command/query

The LABel command allows you to specify polarity and assign channels to new or existing labels. If the specified label name does not match an existing label name, a new label will be created.

The order of the <assignment> parameters (from left to right as seen on screen) is assumed to be the highest pod number first (left-most) followed by any other assigned pods in decreasing pod number order. When viewing the assignment value in binary (base 2), the binary value represents the bit values in the label. A "1" in a bit position means the associated channel in that pod is assigned to that pod and bit. A "0" in a bit position means the associated channel in that pod is excluded from the label. Since pods contain 16 channels (2^{16}-1) 65535 decimal is the maximum value for the pod specification and 0 is the minimum value.

The LABel query returns the current specification for the selected (by name) label. If the label does not exist, nothing is returned.

Command Syntax: :MACHine{1|2}:SFORmat:LABel <name>,<polarity>,<assignment>

where:

<name>	::= string of up to 6 alphanumeric characters
<polarity>	::= {POSitive|NEGative}
<assignment>	::= {pod<MS>_spec[,pod<MS-1>_spec...,pod<LS_spec>]}
pod<MS|LS>	::= pod number, MS being the highest pod number assigned and LS is the lowest pod number assigned
_spec	::= number equal to or greater than 0 (zero) but less than or equal to 65535

Example: OUTPUT XXX;":MACHine2:SFORmat:LABel 'A', POSitive, 65535,127,0"

Query Syntax: :MACHine{1|2}:SFORmat:LABel? <name>

Returned Format: [:MACHine{1|2}:SFORmat:LABel] <name>,<polarity>,<assignment> <NL>

Example:

```
10 DIM String$[100]
20 OUTPUT XXX;":MACHINE2:SFORMAT:LABEL? 'DATA'"
30 ENTER XXX String$
40 PRINT String$
50 END
```

MASTer command/query

The MASTer clock command allows you to specify a master clock for a given machine. The master clock is used in all clocking modes (Normal, Mixed, and Demultiplexed). Each command deals with only one clock (J,K,L,M,N); therefore, a complete clock specification requires five commands, one for each clock. Edges are ORed with edges, levels are ORed with levels. ORed edges are ANDed with ORed levels. Levels specified with MASTer will be the same for SLAVe.

The MASTer query returns the clock specification for the specified clock.

Note

At least one clock edge must be specified.

Command Syntax: :MACHine{1|2}:SFORmat:MASTer <clock_id>,<clock_spec>

where:

<clock_id> ::= {J|K|L|M|N}
<clock_spec> ::= {OFF|RISing|FALLing|BOTH|LOW|HIGH}

Example: OUTPUT XXX;":MACHine2:SFORmat:MASTer J, RISing"

Query Syntax: :MACHine{1|2}:SFORmat:MASTer? <clock_id>

Returned Format: [:MACHine{1|2}:SFORmat:MASTer] <clock_id>,<clock_spec><NL>

Example:

```
10 DIM String$[100]
20 OUTPUT XXX;":MACHINE2:SFORMAT:MASTER? <clock_id>"
30 ENTER XXX String$
40 PRINT String$
50 END
```

REMove command

The REMove command allows you to delete all labels or any one label for a given machine.

Command Syntax: :MACHine{1|2}:SFORmat:REMove {<name>|ALL}

where:

<name> ::= string of up to 6 alphanumeric characters

Examples:
```
OUTPUT XXX;":MACHINE2:SFORMAT:REMOVE 'A'"
OUTPUT XXX;":MACHINE2:SFORMAT:REMOVE ALL"
```

SLAVe

command/query

The SLAVe clock command allows you to specify a slave clock for a given machine. The slave clock is only used in the Mixed and Demultiplexed clocking modes. Each command deals with only one clock (J,K,L,M,N); therefore, a complete clock specification requires five commands, one for each clock. Edges are ORed with edges. Levels are ORed with levels. ORed edges are ANDed with ORed levels. Levels specified with SLAVe will be the same in MASTer.

The SLAVe query returns the clock specification for the specified clock.

Note

When slave clock is being used at least one edge must be specified.

Command Syntax: :MACHine{1|2}:SFORmat:SLAVe <clock_id>,<clock_spec>

where:

<clock_id> ::= {J|K|L|M|N}
<clock_spec> ::= {OFF|RISing|FALLing|BOTH|LOW|HIGH}

Example:
```
OUTPUT XXX;":MACHine2:SFORmat:SLAVe J, RISing"
```

Query Syntax: :MACHine{1|2}:SFORmat:SLAVe?<clock_id>

Returned Format: [:MACHine{1|2}:SFORmat:SLAVe]<clock_id>,<clock_spec><NL>

Example:
```
10 DIM String$[100]
20 OUTPUT XXX;":MACHINE2:SFORMAT:SLAVE? <clock_id>"
30 ENTER XXX String$
40 PRINT String$
50 END
```

THReshold command/query

The THReshold command allows you to set the voltage threshold for a given pod to ECL, TTL, or a specific voltage from -9.9V to +9.9V in 0.1 volt increments.

Note

The pod thresholds of pods 1, 2, and 3 can be set independently. The pod thresholds of pods 4 and 5 are slaved together; therefore when you set the threshold on either pod 4 or 5, both thresholds will be changed to the specified value.

The THReshold query returns the current threshold for a given pod.

Command Syntax: :MACHine{1|2}:SFORmat:THReshold<N> {TTL|ECL|<value>}

where:

<N> ::= pod number {1|2|3|4|5}
<value> ::= voltage (real number) -9.9 to +9.9
TTL ::= default value of +1.6V
ECL ::= default value of -1.3V

Example: OUTPUT XXX;":MACHine1:SFORmat:THReshold1 4.0"

Query Syntax: :MACHine{1|2}:SFORmat:THReshold<N>?

Returned Format: [:MACHine{1|2}:SFORmat:THReshold<N>] <value><NL>

Example:

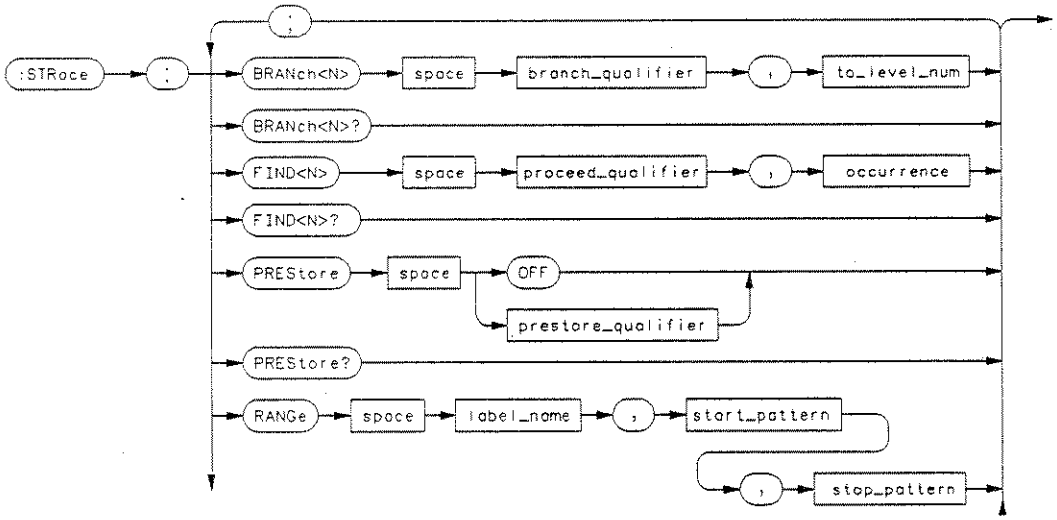
```
10 DIM Value$ [100]
20 OUTPUT XXX;":MACHINE1:SFORMAT:THRESHOLD4?"
30 ENTER XXX;Value$
40 PRINT Value$
50 END
```

STRace Subsystem 11

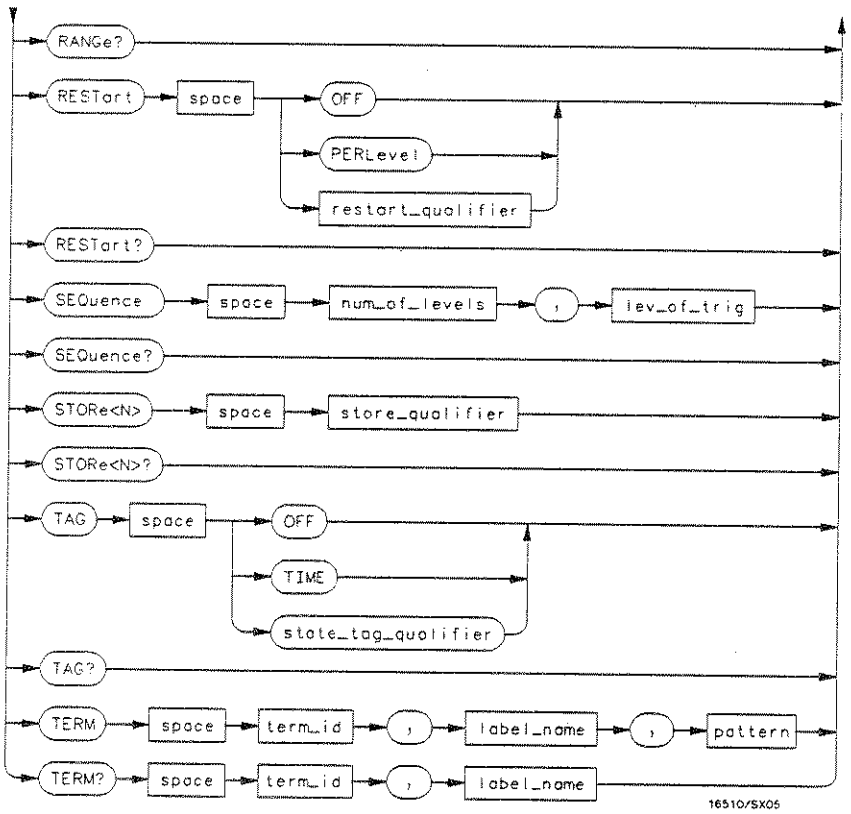
Introduction

The STRace Subsystem contains the commands available for the State Trace Menu in the HP 1650A/51A logic analyzer. The STRace Subsystem commands are:

- BRANch
- FIND
- PREStore
- RANGe
- RESTart
- SEQuence
- STORe
- TAG
- TERM



P/O Figure 11-1. STRace Subsystem Syntax Diagram



branch_qualifier = { <*simple_qual*> | <*complex_qual*> }
to_lev_num = *{1 to <lev_of_trig> -1} before or on the trigger level or {(lev_of_trig) + 1 to end} after the trigger level*
proceed_qualifier = { <*simple_qual*> | <*complex_qual*> }
occurrence = *number from 1 to 65535*
prestore_qual = { <*simple_qual*> | <*complex_qual*> }
label_name = *string of up to 6 alphanumeric characters*

P/O Figure 11-1. STRace Subsystem Syntax Diagram

start_pattern = *string in one of the following forms:*
"#B01..." for binary
"#Q01234567..." for octal
"#H0123456789ABCDEF..." for hex
"0123456789..." for decimal
stop_pattern = *string in one of the following forms:*
"#B01..." for binary
"#Q01234567..." for octal
"#H0123456789ABCDEF..." for hex
"0123456789..." for decimal
restart_qualifier = *{ <simple_qual> | <complex_qual> }*
num_of_levels = *{2 to 8} when ARM is RUN or {2 to 7} otherwise*
lev_of_trig = *1 to (<number_of_levels> -1)*
store_qualifier = *{ <simple_qual> | <complex_qual> }*
state_tag_qualifier = *{ <simple_qual> | <complex_qual> }*
term_id = *{A|B|C|D|E|F|G|H}*
pattern = *string in one of the following forms:*
"#B01X..." for binary
"#Q01234567X..." for octal
"#H0123456789ABCDEFX..." for hex
"0123456789..." for decimal
simple_qual = *{ANYState|NOSTate| <range_pick> | <term> |NOT <term> }*
range_pick = *{INRange|OUTRange}*
term = *{A|B|C|D|E|F|G|H}*
complex_qual = *(<logical_operand> [<logical_operator> <logical_operand>])*
logical_operator = *{AND|OR}*
logical_operand = *{ <simple_qualifier> | <and_expression> | <or_expression> }*
and_expression = *(<and_term> AND <and_term> [...AND <and_term>])*
or_expression = *(<or_term> OR <or_term> [...OR <or_term>])*
or_term = *{A|B|C|D| <range_pick> } in an ORed expression or {E|F|G|H} in a different ORed expression*
and_term = *{NOTA|NOTB|NOTC|NOTD| <range_pick> } in an ANDed expression or {NOTE|NOTF|NOTG|NOTH} in a different ANDed expression*

P/O Figure 11-1. STRace Subsystem Syntax Diagram

STRace

selector

The STRace (State Trace) selector is used as a part of a compound header to access the settings found in the State Trace menu. It always follows the MACHine selector because it selects a branch directly below the MACHine level in the command tree.

Command Syntax: :MACHine{1|2}:STRace

Example: OUTPUT XXX,":MACHine1:STRace:TAG TIME"

BRANch command/query

The BRANch command defines the branch qualifier for a given sequence level. When this branch qualifier is matched, it will cause the sequencer to jump to the specified sequence level.

Note

Branching above the trigger term or below the trigger term is valid. Branching from above to below or from below to above the trigger term is not allowed.

The branch qualifier consists of pattern or range recognizer terms explained in the TERM and RANGe commands. It can be a single term in the simple case, or a logical combination of two groups of logically combined terms in the most complex case. An example of a complex qualifier is shown in Figure 11-2.

Note

The RESTart PERLevel must be in effect for this command to impact sequencer transitions (see RESTart command in this chapter).

The BRANch query returns the current branch qualifier specification for a given sequence level.

Command Syntax: :MACHine{1|2}:STRace:BRANch<N><branch_qualifier>,<to_level_num>

where:

<N>	::= an integer from 1 to <number_of_levels>
<to_lev_num>	::= {1 to <lev_of_trig> -1} before or on the trigger level
	or
	::= {(1 + trigger level) to <num_of_levels>} after the trigger level
<num_of_levels>	::= {2 to 8}
<branch_qualifier>	::= {<simple_qual> | <complex_qual>}
<simple_qual>	::= {ANYState|NOSTate|<range_pick>|<term>|NOT<term>}
<range_pick>	::= {INRange|OUTRange}

BRANch

```
<term>              ::= {A|B|C|D|E|F|G|H}
<complex_qual>      ::= (<logical_operand>[<logical_operator><logical_operand>])
<logical_operator>  ::= {AND|OR}
<logical_operand>   ::= {<simple_qualifier>|<and_expression>|<or_expression>}
<and_expression>    ::= (<and_term> AND <and_term>[...AND <and_term>])
<or_expression>     ::= (<or_term> OR <or_term>[...OR <or_term>])
<or_term>           ::= {A|B|C|D|<range_pick>} in an ORed expression
                    or
                    ::= {E|F|G|H} in a different ORed expression
<and_term>          ::= {NOTA|NOTB|NOTC|NOTD|<range_pick>} in an ANDed expression
                    or
                    ::= {NOTE|NOTF|NOTG|NOTH} in a different ANDed expression
```

Examples

```
OUTPUT XXX,":MACHine1:STRace:BRANch1 ANYState,3"
OUTPUT XXX,":MACHine1:STRace:BRANch2 A,7"
OUTPUT XXX,":MACHine1:STRace:BRANch3 ((A OR B) OR NOTG),1"
```

Query Syntax

```
:MACHine{1|2}:STRace:BRANch<N>?
```

Returned Format:

```
[:MACHine{1|2}:STRace:BRANch<N>]
<branch_qualifier>,<to_level_num><NL>
```

Example:

```
10 DIM String$[100]
20 OUTPUT XXX;":MACHINE1:STRACE:BRANCH3?"
30 ENTER XXX;String$
40 PRINT String$
50 END
```

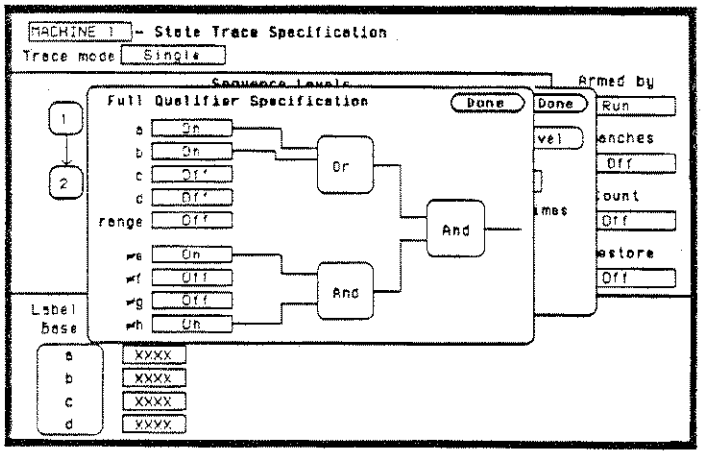


Figure 11-2. Complex qualifier

Figure 11-2 (above) is a front panel representation of the complex qualifier **(A or B) and (≠E and ≠H)**. The following example would be used to specify this complex qualifier.

```
OUTPUT XXX;":MACHINE1:STRACE:BRANCH2 ((A OR B) AND (NOTE AND NOTH)),4"
```

Note

Qualifiers **a** *through* **d** *and* **range** *must be grouped together and qualifiers* **e** *through* **h** *must be grouped together. A qualifier from one group cannot be specified in the other qualifier group. For example, the boolean equation (a + b + range)*(e + c + h) is not allowed because qualifier* **c** *cannot be specified in the* **e-h** *group. The <logical_operator> AND can only be used to AND the not terms within a group (i.e., ≠e - ≠h in figure 11-2). The <logical _operator> AND can be used to AND combined terms (or not terms) of one group with the terms (or not terms) of the other group. For example, the combined term (a OR b) is ANDed with the combined terms (≠e AND ≠h) as shown in figure 11-2.*

FIND command/query

The FIND command defines the proceed qualifier, for a given sequence level, that tells the state analyzer when to proceed to the next sequence level. When this proceed qualifier is matched the specified number of times, the sequencer will proceed to the next sequence level. The state that causes the sequencer to switch levels is automatically stored in memory whether it matches the associated store qualifier or not. In the sequence level where the trigger is specified, the FIND specifies the trigger qualifier.

The proceed qualifier consists of pattern or range recognizer terms explained in the TERM and RANGE commands. It can be a single term in the simple case, or a logical combination of two groups of logically combined terms in the most complex case. An example of a complex qualifier is shown in Figure 11-2.

The FIND query returns the current proceed qualifier specification for a given sequence level.

Command Syntax: :MACHine{1|2}:STRace:FIND<N> <proceed_qualifier>,<occurrence>

where:

<N> ::= an integer from 1 to <number_of_levels>
<occurrence> ::= number from 1 to 65535
<proceed_qualifier> ::= {<simple_qual>|<complex_qual>}
<simple_qual> ::= {ANYState|NOSTate|<range_pick>|<term>|NOT<term>}
<range_pick> ::= {INRange|OUTRange}
<term> ::= {A|B|C|D|E|F|G|H}
<complex_qual> ::= (<logical_operand>[<logical_operator><logical_operand>])
<logical_operator> ::= {AND|OR}
<logical_operand> ::= {<simple_qualifier>|<and_expression>|<or_expression>}
<and_expression> ::= (s<and_term> AND <and_term>[...AND <and_term>])
<or_expression> ::= (<or_term> OR <or_term>[...OR <or_term>])
<or_term> ::= {A|B|C|D|<range_pick>} in an ORed expression
or
::= {E|F|G|H} in a different ORed expression

<and_term> ::= {NOTA|NOTB|NOTC|NOTD|<range_pick>} in an ANDed expression
or
::= {NOTE|NOTF|NOTG|NOTH} in a different ANDed expression

Examples:

```
OUTPUT XXX,":MACHine1:STRace:FIND1 ANYState,1"
OUTPUT XXX,":MACHine1:STRace:FIND2 A,512"
OUTPUT XXX,":MACHine1:STRace:FIND3 ((A OR B) OR NOTG),1"
```

Query Syntax: :MACHine{1|2}:STRace:FIND4?

Returned Format: [:MACHine{1|2}:STRace:FIND<N>]<proceed_qualifier>,<occurrence><NL>

Example:

```
10 DIM String$[100]
20 OUTPUT XXX;":MACHINE1:STRACE:FIND<N>?"
30 ENTER XXX;String$
40 PRINT String$
50 END
```

PREStore

command/query

The PREStore command turns the prestore feature on and off. It also defines the qualifier required to prestore only selected states. The prestore qualifier consists of pattern or range recognizer terms explained in the TERM and RANGe commands. PREStore can be a single term in the simple case, or a logical combination of two groups of combined terms in the most complex case. An example of a complex qualifier is shown in Figure 11-2.

The PREStore query returns the current prestore specification.

Command Syntax: :MACHine{1|2}:STRace:PREStore {OFF|<prestore_qualifier>}

where:

<prestore_qualifier> ::= {<simple_qual>|<complex_qual>}
<simple_qual> ::= {ANYState|NOSTate|<range_pick>|<term>|NOT<term>}
<range_pick> ::= {INRange|OUTRange}
<term> ::= {A|B|C|D|E|F|G|H}
<complex_qual> ::= (<logical_operand>[<logical_operator><logical_operand>])
<logical_operator> ::= {AND|OR}
<logical_operand> ::= {<simple_qualifier>|<and_expression>|<or_expression>}
<and_expression> ::= (<and_term> AND <and_term>[...AND <and_term>])
<or_expression> ::= (<or_term> OR <or_term>[...OR <or_term>])
<or_term> ::= {A|B|C|D|<range_pick>} in an ORed expression
or
::= {E|F|G|H} in a different ORed expression
<and_term> ::= {NOTA|NOTB|NOTC|NOTD|<range_pick>} in an ANDed expression
or
::= {NOTE|NOTF|NOTG|NOTH} in a different ANDed expression

Examples:
```
OUTPUT XXX,":MACHine1:STRace:PREStore OFF"
OUTPUT XXX,":MACHine1:STRace:PREStore ANYState"
OUTPUT XXX,":MACHine1:STRace:PREStore (A)"
OUTPUT XXX,":MACHine1:STRace:PREStore((A OR B) AND NOTG)"
```

Query Syntax: :MACHine{1|2}:STRace:PREStore?

Returned Format: [:MACHine{1|2}:STRace:PREStore]{OFF|<prestore_qualifier>}<NL>

Example:
```
10 DIM String$[100]
20 OUTPUT XXX;":MACHINE1:STRACE:PRESTORE?"
30 ENTER XXX;String$
40 PRINT String$
50 END
```

RANGe command/query

The RANGe command allows you to specify a range recognizer term in the specified machine. Since a range can only be defined across one label and, since a label must contain 32 or less bits, the value of the start pattern or stop pattern will be between (2^{32})-1 and 0.

Note

A label can be defined across all pods (maximum of 32 channels) but the range term can only be specified on a label defined across a maximum of two pods.

When viewing the values in binary, the binary value represents the bit values for the label at one of the range recognizers' end points. Don't cares are not allowed in the end point pattern specifications. Since only one range recognizer exists, it is always used by the first state machine defined.

The RANGe query returns the range recognizer end point specifications for the range.

Note

When two state analyzers are on, the RANGe term is not available in the second state analyzer assigned and there are only 4 pattern recognizers per analyzer.

Command Syntax: :MACHine{1|2}:STRace:RANGE <label_name>,<start_pattern>,<stop_pattern>

where:

<label_name> ::= string of up to 6 alphanumeric characters

<start_pattern> ::= string in one of the following forms:

"#B01..." for binary
"#Q01234567..." for octal
"#H0123456789ABCDEF..." for hex
"0123456789..." for decimal

<stop_pattern> ::= string in one of the following forms:

"#B01..." for binary
"#Q01234567..." for octal
"#H0123456789ABCDEF..." for hex
"0123456789..." for decimal

Examples:

```
OUTPUT XXX,":MACHine1:STRace:RANGe 'DATA','127','255' "
OUTPUT XXX,":MACHine1:STRace:RANGe 'ABC','#B00001111','#HCF' "
```

Query Syntax:

```
:MACHine{1|2}:STRace:RANGe?
```

Returned Format:

```
[:MACHine{1|2}:STRAce:RANGe] <label_name>,<start_pattern>,<stop_pattern>
<NL>
```

Example:

```
10 DIM String$[100]
20 OUTPUT XXX;":MACHINE1:STRACE:RANGE?"
30 ENTER XXX;String$
40 PRINT String$
50 END
```

RESTart

command/query

The RESTart command selects the type of restart to be enabled during the trace sequence. It also defines the global restart qualifier that restarts the sequence in global restart mode. The restart qualifier consists of pattern or range recognizer terms explained in the TERM and RANGe commands. RESTart can be a single term in the simple case, or a logical combination of two groups of combined terms in the most complex case. An example of a complex qualifier is shown in Figure 11-2.

The RESTart query returns the current restart specification.

Command Syntax: :MACHine{1|2}:STRace:RESTart {OFF|PERLevel|<restart_qualifier>}

where:

<restart_qualifier> ::= {<simple_qual>|<complex_qual>}
<simple_qual> ::= {ANYState|NOSTate|<range_pick>|<term>|NOT<term>}
<range_pick> ::= {INRange|OUTRange}
<term> ::= {A|B|C|D|E|F|G|H}
<complex_qual> ::= (<logical_operand>[<logical_operator><logical_operand>])
<logical_operator> ::= {AND|OR}
<logical_operand> ::= {<simple_qualifier>|<and_expression>|<or_expression>}
<and_expression> ::= (<and_term> AND <and_term>[...AND <and_term>])
<or_expression> ::= (<or_term> OR <or_term>[...OR <or_term>])
<or_term> ::= {A|B|C|D|<range_pick>}in an ORed expression
or
::={E|F|G|H} in a different ORed expression
<and_term> ::= {NOTA|NOTB|NOTC|NOTD|<range_pick>} in an ANDed expression
or
::={NOTE|NOTF|NOTG|NOTH} in a different ANDed expression

Examples:
```
OUTPUT XXX,":MACHine1:STRace:RESTart OFF"
OUTPUT XXX,":MACHine1:STRace:RESTart PERLevel"
OUTPUT XXX,":MACHine1:STRace:RESTart (A)"
OUTPUT XXX,":MACHine1:STRace:RESTart ((A OR B) AND NOTG)"
```

Query Syntax: :MACHine{1|2}:STRace:RESTart?

Returned Format: [:MACHine{1|2}:STRace:RESTart]{OFF|PERLevel|<restart_qualifier>} <NL>

Example:
```
10 DIM String$[100]
20 OUTPUT XXX;":MACHINE1:STRACE:RESTART?"
30 ENTER XXX;String$
40 PRINT String$
50 END
```

SEQuence command/query

The SEQuence command redefines the state analyzer trace sequence. First, it deletes the current trace sequence. Then it inserts the number of levels specified, with default settings, and assigns the trigger to be at a specified sequence level. The number of levels can be between 2 and 8 when the analyzer is armed by the RUN key. When armed by the BNC or the other machine, a level is used by the arm in; therefore, only seven levels are available in the sequence.

The SEQuence query returns the current sequence specification.

Command Syntax: :MACHine{1|2}:STRace:Sequence <num_of_levels>,<lev_of_trig>

where:

<num_of_levels> ::= {2-8} when ARM is RUN or {2-7} otherwise
<lev_of_trig> ::= 1 to (<number_of_levels>-1)

Example:
```
OUTPUT XXX,":MACHine1:STRace:Sequence 4,3"
```

Query Syntax: :MACHine{1|2}:STRace:Sequence?

Returned Format: [:MACHine{1|2}:STRace:Sequence] <num_of_levels>,<lev_of_trig><NL>

Example:
```
10 DIM String$[100]
20 OUTPUT XXX;":MACHINE1:STRACE:SEQUENCE?"
30 ENTER XXX;String$
40 PRINT String$
50 END
```

STORe command/query

The STORe command defines the store qualifier for a given sequence level. Any data matching the STORe qualifier will actually be stored in memory as part of the current trace data. The store qualifier consists of pattern or range recognizer terms explained in the TERM and RANGE commands. It can be a single term in the simple case, or a logical combination of two groups of logically combined terms in the most complex case. An example of a complex qualifier is shown in Figure 11-2.

The STORe query returns the current store qualifier specification for a given sequence level <N>.

Command Syntax: :MACHine{1|2}:STRace:STORe <N> <store_qualifier>

where:

<N>	::= an integer from 1 to <number_of_levels>
<store_qualifier>	::= {<simple_qual> | <complex_qual>}
<simple_qual>	::= {ANYState | NOSTate | <range_pick> | <term> | NOT <term>}
<range_pick>	::= {INRange | OUTRange}
<term>	::= {A|B|C|D|E|F|G|H}
<complex_qual>	::= (<logical_operand> [<logical_operator> <logical_operand>])
<logical_operator>	::= {AND|OR}
<logical_operand>	::= {<simple_qualifier> | <and_expression> | <or_expression>}
<and_expression>	::= (<and_term> AND <and_term> [...AND <and_term>])
<or_expression>	::= (<or_term> OR <or_term> [...OR <or_term>])
<or_term>	::= {A|B|C|D| <range_pick>} in an ORed expression or ::= {E|F|G|H} in a different ORed expression
<and_term>	::= {NOTA|NOTB|NOTC|NOTD| <range_pick>} in an ANDed expression or ::= {NOTE|NOTF|NOTG|NOTH} in a different ANDed expression

STORe

Examples:
```
OUTPUT XXX,":MACHine1:STRace:STORe1 ANYState"
OUTPUT XXX,":MACHine1:STRace:STORe2 A"
OUTPUT XXX,":MACHine1:STRace:STORe3 ((A OR B) OR NOTG)"
```

Query Syntax:
```
:MACHine{1|2}:STRace:STORe<N>?
```

Returned Format:
```
[:MACHine{1|2}:STRace:STORe<N>] <store_qualifier><NL>
```

Example:
```
10 DIM String$[100]
20 OUTPUT XXX;":MACHINE1:STRACE:STORE4?"
30 ENTER XXX;String$
40 PRINT String$
50 END
```

TAG

command/query

The TAG command selects the type of count tagging (state or time) to be performed during data acquisition. It also defines the state tag qualifier that will be counted in the qualified state mode. The state tag qualifier basically consists of the pattern or range recognizer terms explained in the TERM and RANGe commands. The tag can be a single term in the simple case, or a logical combination of two groups of combined terms in the most complex case. An example of a complex qualifier is shown in Figure 11-2. If CPERiod (SFORmat Subsystem) is set to Less Than 60 ns and TAG is set to time or states, CPERiod will automatically be set to Greater Than 60 ns.

The TAG query returns the current count tag specification.

Command Syntax: :MACHine{1|2}:STRace:TAG {OFF|TIME|<state_tag_qualifier>}

where:

<state_tag_qualifier> ::= {<simple_qual>|<complex_qual>}
<simple_qual> ::= {ANYState|NOSTate|<range_pick>|<term>|<NOT<term>}
<range_pick> ::= {INRange|OUTRange}
<term> ::= {A|B|C|D|E|F|G|H}
<complex_qual> ::= (<logical_operand>[<logical_operator><logical_operand>])
<logic_operator> ::= {AND|OR}
<logical_operand> ::= {<simple_qualifier>|<and_expression>|<or_expression>}
<and_expression> ::= (<and_term> AND <and_term[...AND <and_term>])
<or_expression> ::= (<or_term> OR <or_term>{... OR <or_term>})
<or_term> ::= {A|B|C|D|<range_pick>} in an OR expression
or
::= {E|F|G|H} in a different OR expression
<and_term> ::= {NOTA|NOTB|NOTG|NOTD|<range_pick>} in an AND expression
or
::= {NOTE|NOTF|NOTG|NOTH} in a different AND expression

TAG

Examples:
```
OUTPUT XXX,":MACHine1:STRace:TAG OFF"
OUTPUT XXX,":MACHine1:STRace:TAG TIME"
OUTPUT XXX,":MACHine1:STRace:TAG (A)"
OUTPUT XXX,":MACHine1:STRace:TAG ((A OR B) AND NOTG)"
```

Query Syntax: :MACHine{1|2} :STRace:TAG?

Returned Format: [:MACHine{1|2}:STRace:TAG] {OFF|TIME|<state_tag_qualifier>}<NL>

Example:
```
10 DIM String$[100]
20 OUTPUT XXX;":MACHINE1:STRACE:TAG?"
30 ENTER XXX;String$
40 PRINT String$
50 END
```

TERM command/query

The TERM command allows you to a specify a pattern recognizer term in the specified machine. Each command deals with only one label in the given term; therefore, a complete specification could require several commands. Since a label can contain 32 or less bits, the range of the pattern value will be between (2^{32})-1 and 0. When viewing the value of a pattern in binary, the binary value represents the bit values for the label inside the pattern recognizer term. Since the pattern parameter may contain don't cares and be represented in several bases, it is handled as a string of characters rather than a number.

When a single state machine is on, all eight terms (A through H) are available in the single machine. When two state machines are on, terms A through D are used by the first state machine defined. The terms E through H are used by the second state machine defined.

The TERM query returns the specification of the term specified by term identification and label name.

Command Syntax: :MACHine{1|2}:STRace:TERM <term_id>,<label_name>,<pattern>

where:

<term_id> ::= {A|B|C|D|E|F|G|H}
<label_name> ::= string of up to 6 alphanumeric characters
<pattern> ::= string in one of the following forms:

"#B01X..." for binary
"#Q01234567X..." for octal
"#H0123456789ABCDEFX..." for hex
"0123456789..." for decimal

Example:
```
OUTPUT XXX,":MACHine1:STRace:TERM A,'DATA','255' "
OUTPUT XXX,":MACHine1:STRace:TERM B,'ABC','#BXXXX1101' "
```

TERM

Query Syntax: :MACHine{1|2}:STRace:TERM? <term_id>,<label_name>

Returned Format: [:MACHine{1|2}:STRAce:TERM] <term_id>,<label_name>,<pattern><NL>

Example:

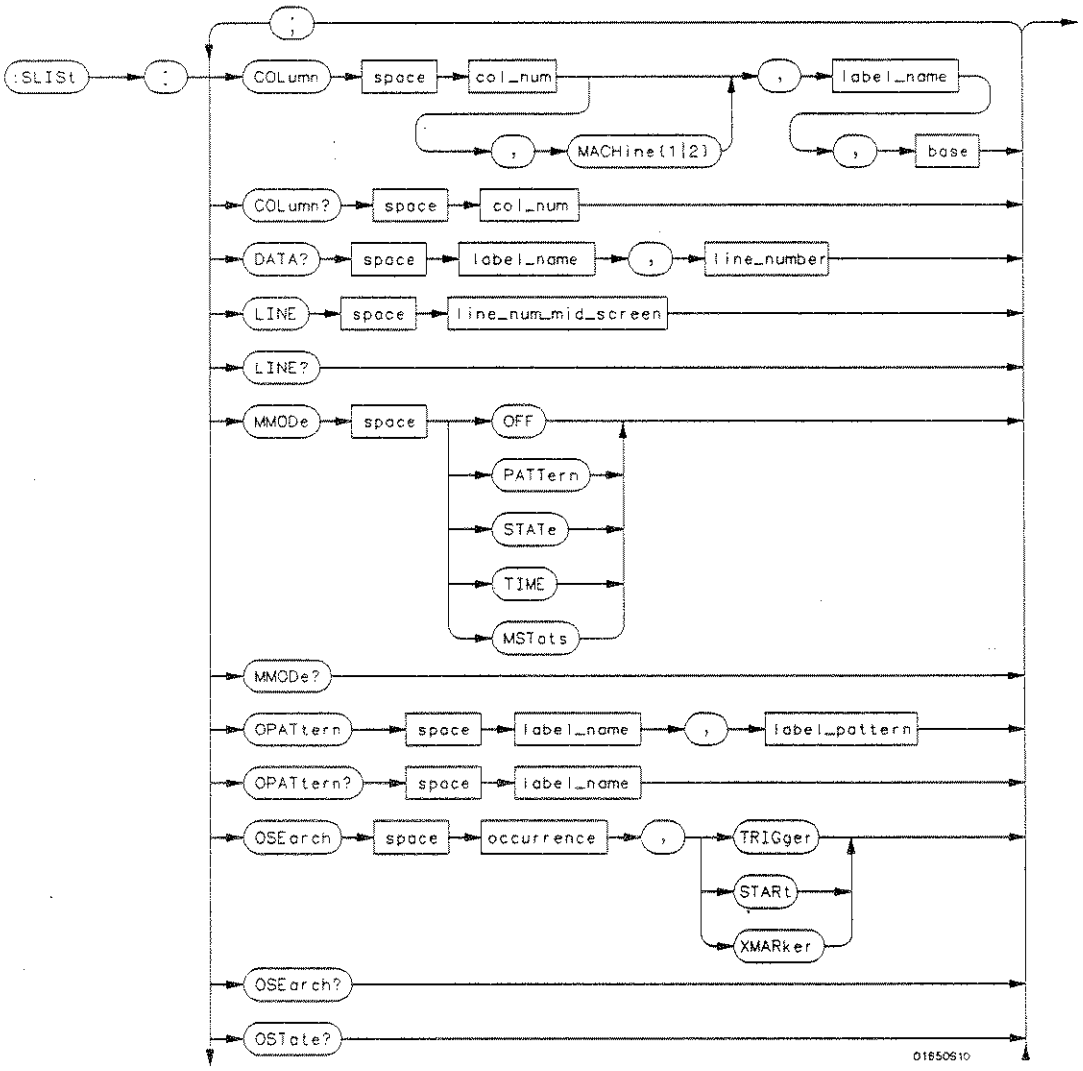
```
10 DIM String$[100]
20 OUTPUT XXX;":MACHINE1:STRACE:TERM? B,'DATA' "
30 ENTER XXX;String$
40 PRINT String$
50 END
```

SLISt Subsystem 12

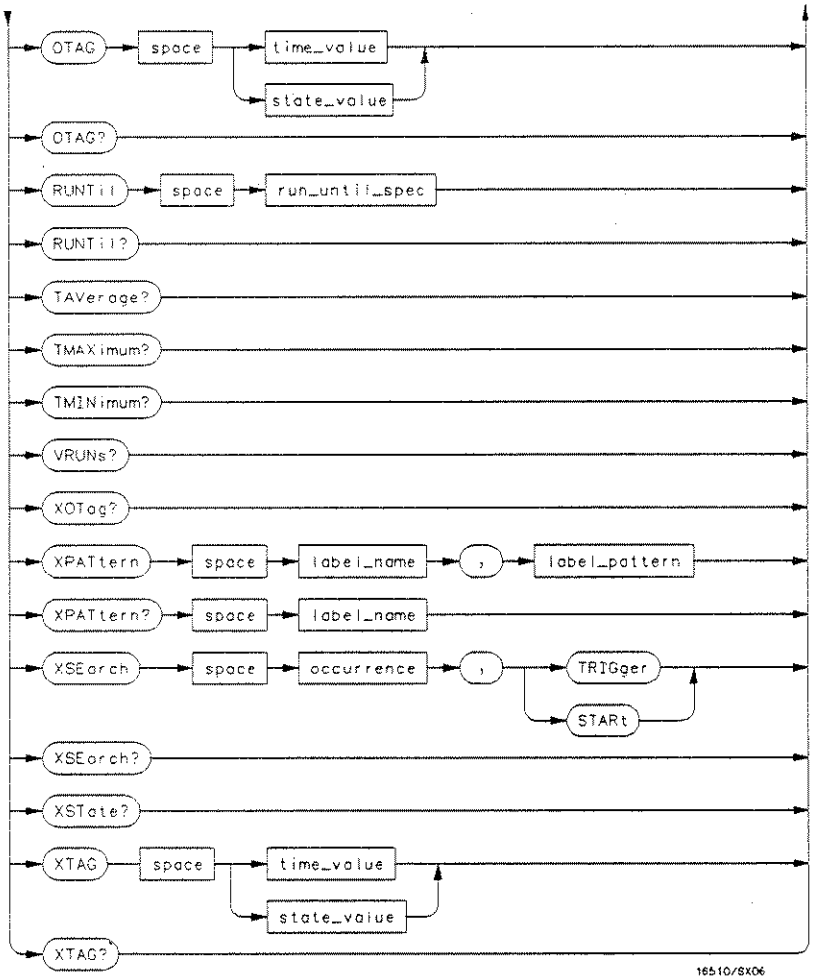
Introduction

The SLISt Subsystem contains the commands available for the State Listing Menu in the HP 1650A/51A logic analyzer. These commands are:

- COLumn
- DATA
- LINE
- MMODe
- OPATtern
- OSEarch
- OSTate
- OTAG
- RUNTil
- TAVerage
- TMAXimum
- TMINimum
- VRUNs
- XOTag
- XPATtern
- XSEarch
- XSTate
- XTAG



P/O Figure 12-1. SLISt Subsystem Syntax Diagram



mach_num = {*1*|*2*}
col_num = *integer from* {*1-8*}
line_number = *integer from -1023 to + 1024*
label_name = *a string of up to 6 alphanumeric characters*

P/O Figure 12-1. SLISt Subsystem Syntax Diagram

base = {*BINary*|*HEXadecimal*|*OCTal*|*DECimal*|*ASCii*|*SYMBol*|*IASSembler*}
for labels or {*ABSolute*|*RELative*} *for tags*
line_num_mid_screen = *integer from -1023 to + 1024*
label_pattern = *string in one of the following forms:*
"#B01X..." for binary
"#Q01234567X..." for octal
"#H0123456789ABCDEFX..." for hex
"0123456789..." for decimal
occurrence = *integer*
time_value = *real number*
state_value = *real number*
run_until_spec =
{*OFF*|*LT*, <*value*>|*GT*, <*value*>|*INRange*, <*value*>, <*value*>|*OUTRange*, <*value*>, <*value*>}
value = *real number*

P/O Figure 12-1. SLISt Subsystem Syntax Diagram

SLISt selector

The SLISt selector is used as part of a compound header to access those settings normally found in the State Listing menu. It always follows the MACHine selector because it selects a branch directly below the MACHine level in the command tree.

Command Syntax: :MACHine{1|2}:SLISt

Example: OUTPUT XXX,":MACHine1:SLISt:LINE 256"

COLumn

command/query

The COLumn command allows you to configure the state analyzer list display by assigning a label name and base to one of the eight vertical columns in the menu. A column number of 1 refers to the left most column. When a label is assigned to a column it replaces the original label in that column. The label originally in the specified column is placed in the column the specified label is moved from.

When the label name is "TAGS," the TAGS column is assumed and the next parameter must specify RELative or ABSolute.

The COLumn query returns the column number, label name, and base for the specified column.

Command Syntax: :MACHine{1|2}:SLISt:COLumn <col_num>,<label_name>,<base>

where:

<col_num> ::= integer from {1-8}

<label_name> ::= a string of up to 6 alphanumeric characters

<base> ::= {BINary|HEXadecimal|OCTal|DECimal|ASCii|SYMBol|IASSembler} for labels

or

::= {ABSolute|RELative} for tags

Note

A label for tags must be assigned in order to use ABSolute or RELative state tagging.

Examples:
```
OUTPUT XXX,":MACHine1:SLISt:COLumn 4,'A',HEX"
OUTPUT XXX,":MACHine1:SLISt:COLumn 1,'TAGS', ABSolute"
```

Query Syntax: :MACHine{1|2}:SLISt:COLumn? <col_num>

Returned Format: [:MACHine{1|2}:SLISt:COLumn] <col_num>,<label_name>,<base><NL>

Example:
```
10 DIM CI$[100]
20 OUTPUT XXX;":MACHINE1:SLIST:COLUMN? 4"
30 ENTER XXX;CI$
40 PRINT CI$
50 END
```

DATA query

The DATA query returns the value at a specified line number for a given label.

Query Syntax: :MACHine{1|2}:SLISt:DATA? <label_name>,<line_number>

Returned Format: [:MACHine{1|2}:SLISt:DATA]<label_name>,<line_number>,
<pattern_string><NL>

where:

<line_number> ::= integer from -1023 to +1024
<label_name> ::= string of up to 6 alphanumeric characters
<pattern_string> ::= string in one of the following forms:

"#B01..." for binary
"#Q01234567..." for octal
"#H0123456789ABCDEF..." for hex
"0123456789..." for decimal

Example:

```
10 DIM Sd$[100]
20 OUTPUT XXX;":MACHINE1:SLIST:DATA? 512, 'RAS'"
30 ENTER XXX;Sd$
40 PRINT Sd$
50 END
```

LINE command/query

The LINE command allows you to scroll the state analyzer listing vertically. The command specifies the state line number, relative to the trigger, that the analyzer will highlight at center screen.

The LINE query returns the line number for the state currently in the box at center screen.

Command Syntax: :MACHine{1|2}:SLISt:LINE <line_num_mid_screen>

where:

<line_num_mid_screen> ::= a real number from -1023 to +1024

Example:
```
OUTPUT XXX;":MACHine1:SLISt:LINE 0"
```

Query Syntax: :MACHine{1|2}:SLISt:LINE?

Returned Format: [:MACHine{1|2}:SLISt:LINE] <line_num_mid_screen><NL>

Example:
```
10 DIM Ln$[100]
20 OUTPUT XXX;":MACHINE1:SLIST:LINE?"
30 ENTER XXX;Ln$
40 PRINT Ln$
50 END
```

MMODe command/query

The MMODe command (Marker Mode) selects the mode controlling the marker movement and the display of marker readouts. When PATTern is selected, the markers will be placed on patterns. When STATe is selected and state tagging is on, the markers move on qualified states counted between normally stored states. When TIME is selected and time tagging is enabled, the markers move on time between stored states. When MSTats is selected and time tagging is on, the markers are placed on patterns, but the readouts will be time statistics.

The MMODe query returns the current marker mode selected.

Command Syntax: :MACHine{1|2}:SLISt:MMODe <marker_mode>

where:

<marker_mode> ::= {OFF|PATTern|STATe|TIME|MSTats}

Example:
```
OUTPUT XXX,":MACHine1:SLISt:MMODe TIME"
```

Query Syntax: :MACHine{1|2}:SLISt:MMODe?

Returned Format: [:MACHine{1|2}:SLISt:MMODe] <marker_mode><NL>

Example:
```
10 DIM Mn$[100]
20 OUTPUT XXX;":MACHINE1:SLIST:MMODE?"
30 ENTER XXX;Mn$
40 PRINT Mn$
50 END
```

OPATtern

command/query

The OPATtern command allows you to construct a pattern recognizer term for the O Marker which is then used with the OSEarch criteria when moving the marker on patterns. Since each command deals with only one label in the pattern recognizer, a complete specification could require several commands. Since a label can contain up to 32 bits, the range of the pattern value will be between 0 and (2^{32})-1.

When the value of a pattern is expressed in binary, it represents the bit values for the label inside the pattern recognizer term. Since a pattern recognizer may contain don't cares and be represented in several bases, the <label_pattern> parameter is handled as a string of characters rather than a number.

The OPATtern query returns the pattern specification for a given label name.

Command Syntax: :MACHine{1|2}:SLISt:OPATtern <label_name>,<label_pattern>

where:

<label_name> ::= string of up to 6 alphanumeric characters

<label_pattern> ::= string in one of the following forms:

"#B01X..." for binary
"#Q01234567X..." for octal
"#H0123456789ABCDEFX..." for hex
"0123456789..." for decimal

Examples:

```
OUTPUT XXX,":MACHine1:SLISt:OPATtern 'DATA','255' "
OUTPUT XXX,":MACHine1:SLISt:OPATtern 'ABC','#BXXXX1101' "
```

OPATtern

Query Syntax: :MACHine{1|2}:SLISt:OPATtern? <label_name>

Returned Format: [:MACHine{1|2}:SLISt:OPATtern]<label_name>,<label_pattern><NL>

Example:

```
10 DIM Op$[100]
20 OUTPUT XXX;":MACHINE1:SLIST:OPATTERN? 'A'"
30 ENTER XXX;Op$
40 PRINT Op$
50 END
```

OSEarch

command/query

The OSEarch command defines the search criteria for the O marker, which is then used with the associated OPATtern recognizer specification when moving the markers on patterns. The origin parameter tells the marker to begin a search with the trigger, the start of data, or with the X marker. The actual occurrence the marker searches for in the OPATtern recognizer specification is determined by the occurrence parameter, relative to the origin. An occurrence of 0 places the marker on the selected origin. With a negative occurrence, the marker searches before the origin. With a positive occurrence, the marker searches after the origin.

The OSEarch query returns the search criteria for the O marker.

Command Syntax: :MACHine{1|2}:SLISt:OSEarch <occurrence>,<origin>

where:

<occurrence> ::= integer
<origin> ::= {TRIGger|STARt|XMARker}

Example:
```
OUTPUT XXX,":MACHine1:SLISt:OSEarch +10,TRIGGER"
```

Query Syntax: :MACHine{1|2}:SLISt:OSEarch?

Returned Format: [:MACHine{1|2}:SLISt:OSEarch]<occurrence>,<origin><NL>

Example:
```
10 DIM Os$[100]
20 OUTPUT XXX;":MACHINE1:SLIST:OSEARCH?"
30 ENTER XXX;Os$
40 PRINT Os$
50 END
```

OSTate

query

The OSTate query returns the line number in the listing where the O marker resides (-1023 to +1024). If data is not valid , the query returns 32767.

Query Syntax: :MACHine{1|2}:SLISt:OSTate?

Returned Format: [:MACHine{1|2}:SLISt:OSTate] <state_num><NL>

where:

<state_num> ::= a integer from -1023 to +1024 or 32767

Example:

```
10 DIM Os$[100]
20 OUTPUT XXX;":MACHINE1:SLIST:OSTATE?"
30 ENTER XXX;Os$
40 PRINT Os$
50 END
```

OTAG

command/query

The OTAG command specifies the tag value on which the O Marker should be placed. The tag value is time when time tagging is on or states when state tagging is on. If the data is not valid tagged data, no action is performed.

The OTAG query returns the O Marker position in time when time tagging is on or in states when state tagging is on, regardless of whether the marker was positioned in time, states, or through a pattern search. If data is not valid, the query returns 9.9E37.

Command Syntax: :MACHine{1|2}:SLISt:OTAG { <time_value> | <state_value> }

where:

<time_value> ::= real number
<state_value> ::= integer

Example:
```
:OUTPUT XXX,":MACHine1:SLISt:OTAG 40.0E-6"
```

Query Syntax: :MACHine{1|2}:SLISt:OTAG?

Returned Format: [:MACHine{1|2}:SLISt:OTAG] { <time_value> | <state_value> } <NL>

Example:
```
10 DIM Ot$[100]
20 OUTPUT XXX;":MACHINE1:SLIST:OTAG?"
30 ENTER XXX;Ot$
40 PRINT Ot$
50 END
```

RUNTil command/query

The RUNTil (run until) command defines stop criteria based on the time between the X and O markers, when the trace mode is repetitive. When OFF is selected, the analyzer will make runs until either the STOP key is pressed or the STOP command is issued. The choices for Run Until the time between the X and O Markers are:

- OFF
- Less Than (LT) some time value
- Greater Than (GT) some time value
- In the range (INRange) between two time values
- Out of the range (OUTRange) between two time values

End points for the INRange and OUTRange should be at least 10 ns apart.

The RUNTil query returns the current stop criteria.

Command Syntax: :MACHine{1|2}:SLISt:RUNTil <run_until_spec>

where:

<run_until_spec> ::= {OFF|LT,<value>|GT,<value>|INRange,<value>,<value>|OUTRange,<value>,<value>}

<value> ::= real number

Example: OUTPUT XXX,":MACHine1:SLISt:RUNTil GT,800.0E-6"

RUNTil

Query Syntax: :MACHine{1|2}:SLISt:RUNTil?

Returned Format: [:MACHine{1|2}:SLISt:RUNTil] <run_until_spec><NL>

Example:

```
10 DIM Ru$[100]
20 OUTPUT XXX;":MACHINE1:SLIST:RUNTIL?"
30 ENTER XXX;Ru$
40 PRINT Ru$
50 END
```

TAVerage

TAVerage query

The TAVerage query returns the value of the average time between the X and O Markers. If the number of valid runs is zero, the query returns 9.9E37. Valid runs are those where the pattern search for both the X and O markers was successful resulting in valid delta-time measurements.

Query Syntax: :MACHine{1|2}:SLISt:TAVerage?

Returned Format: [:MACHine{1|2}:SLISt:TAVerage] <time_value> <NL>

where:

<time_value> ::= real number

Example:

```
10 DIM Tv$[100]
20 OUTPUT XXX;":MACHINE1:SLIST:TAVERAGE?"
30 ENTER XXX;Tv$
40 PRINT Tv$
50 END
```

TMAXimum query

The TMAXimum query returns the value of the maximum time between the X and O Markers. If data is not valid, the query returns 9.9E37.

Query Syntax: :MACHine{1|2}:SLISt:TMAXimum?

Returned Format: [:MACHine{1|2}:SLISt:TMAXimum] <time_value> <NL>

where:

<time_value> ::= real number

Example:

```
10 DIM Tx$[100]
20 OUTPUT XXX;":MACHINE1:SLIST:TMAXIMUM?"
30 ENTER XXX;Tx$
40 PRINT Tx$
50 END
```

TMINimum

query

The TMINimum query returns the value of the minimum time between the X and O Markers. If data is not valid, the query returns 9.9E37.

Query Syntax: :MACHine{1|2}:SLISt:TMINimum?

Returned Format: [:MACHine{1|2}:SLISt:TMINimum]<time_value><NL>

where:

<time_value> ::= real number

Example:

```
10 DIM Tm$[100]
20 OUTPUT XXX;":MACHINE1:SLIST:TMINIMUM?"
30 ENTER XXX;Tm$
40 PRINT Tm$
50 END
```

VRUNs query

The VRUNs query returns the number of valid runs and total number of runs made. Valid runs are those where the pattern search for both the X and O markers was successful resulting in valid delta time measurements.

Query Syntax: :MACHine{1|2}:SLISt:VRUNs?

Returned Format: [:MACHine{1|2}:SLISt:VRUNs]<valid_runs>,<total_runs><NL>

where:

<valid_runs> ::= zero or positive integer
<total_runs> ::= zero or positive integer

Example:

```
10 DIM Vr$[100]
20 OUTPUT XXX;":MACHINE1:SLIST:VRUNS?"
30 ENTER XXX;Vr$
40 PRINT Vr$
50 END
```

XOTag

Query

The XOTag query returns the time from the X to O markers when the marker mode is time or number of states from the X to O markers when the marker mode is state. If there is no data in the time mode the query returns 9.9E37.

Query Syntax: :MACHine{1|2}:SLISt:XOTag?

Returned Format: [:MACHine{1|2}:SLISt:XOTag]{<XO_time>|<XO_states>}<NL>

where:

<XO_time> ::= real number
<XO_states> ::= integer

Example:

```
10 DIM Xot$[100]
20 OUTPUT XXX;":MACHINE1:SLIST:XOTAG?"
30 ENTER XXX;Xot$
40 PRINT Xot$
50 END
```

XPATtern command/query

The XPATtern command allows you to construct a pattern recognizer term for the X Marker which is then used with the XSEarch criteria when moving the marker on patterns. Since each command deals with only one label in the pattern recognizer, a complete specification could require several commands. Since a label can contain up to 32 bits, the range of the pattern value will be between 0 and (2^{32})-1.

When the value of a pattern is expressed in binary, it represents the bit values for the label inside the pattern recognizer term. Since a pattern recognizer can contain don't cares and be represented in several bases, the <label_pattern> parameter is handled as a string of characters rather than a number.

The XPATtern query returns the pattern specification for a given label name.

Command Syntax: :MACHine{1|2}:SLISt:XPATtern <label_name>,<label_pattern>

where:

<label_name> ::= string of up to 6 alphanumeric characters
<label_pattern> ::= string in one of the following forms:

"#B01X..." for binary
"#Q01234567X..." for octal
"#H0123456789ABCDEFX..." for hex
"0123456789..." for decimal

Examples:
```
OUTPUT XXX,":MACHine1:SLISt:XPATtern 'DATA','255' "
OUTPUT XXX,":MACHine1:SLISt:XPATtern 'ABC','#BXXXX1101' "
```

XPATtern

Query Syntax: :MACHine{1|2}:SLISt:XPATtern? <label_name>

Returned Format: [:MACHine{1|2}:SLISt:XPATtern]<label_name>,<label_pattern><NL>

Example:

```
10 DIM Xp$[100]
20 OUTPUT XXX;":MACHINE1:SLIST:XPATTERN? 'A'"
30 ENTER XXX;Xp$
40 PRINT Xp$
50 END
```

XSEarch command/query

The XSEarch command defines the search criteria for the X Marker, which is then used with the associated XPATtern recognizer specification when moving the markers on patterns. The origin parameter tells the Marker to begin a search with the trigger or with the start of data. The occurrence parameter determines which occurrence of the XPATtern recognizer specification, relative to the origin, the marker actually searches for. An occurrence of 0 places a marker on the selected origin.

The XSEarch query returns the search criteria for the X marker.

Command Syntax: :MACHine{1|2}:SLISt:XSEarch <occurrence>,<origin>

where:

<occurrence> ::= integer
<origin> ::= {TRIGger|STARt}

Example:
```
OUTPUT XXX,":MACHine1:SLISt:XSEarch +10,TRIGger"
```

Query Syntax: :MACHine{1|2}:SLISt:XSEarch?

Returned Format: [:MACHine{1|2}:SLISt:XSEarch]<occurrence>,<origin><NL>

Example:
```
10 DIM Xs$[100]
20 OUTPUT XXX;":MACHINE1:SLIST:XSEARCH?"
30 ENTER XXX;Xs$
40 PRINT Xs$
50 END
```

XSTate

query

The XSTate query returns the line number in the listing where the X marker resides (-1023 to +1024). If data is not valid, the query returns 32767.

Query Syntax: :MACHine{1|2}:SLISt:XSTate?

Returned Format: [:MACHine{1|2}:SLISt:XSTate] <state_num><NL>

where:

<state_num> ::= an integer from -1023 to +1024 or 32767

Example:

```
10 DIM Xs$[100]
20 OUTPUT XXX;":MACHINE1:SLIST:XSTATE?"
30 ENTER XXX;Xs$
40 PRINT Xs$
50 END
```

XTAG

command/query

The XTAG command specifies the tag value on which the X Marker should be placed. The tag value is time when time tagging is on or states when state tagging is on. If the data is not valid tagged data, no action is performed.

The XTAG query returns the X Marker position in time when time tagging is on or in states when state tagging is on, regardless of whether the marker was positioned in time, states, or through a pattern search. If data is not valid tagged data, the query returns 9.9E37.

Command Syntax: :MACHine{1|2}:SLISt:XTAG { <time_value> | <state_value> }

where:

<time_value> ::= real number
<state_value> ::= integer

Example:
```
:OUTPUT XXX,":MACHine1:SLISt:XTAG 40.0E-6"
```

Query Syntax: :MACHine{1|2}:SLISt:XTAG?

Returned Format: [:MACHine{1|2}:SLISt:XTAG] { <time_value> | <state_value> } <NL>

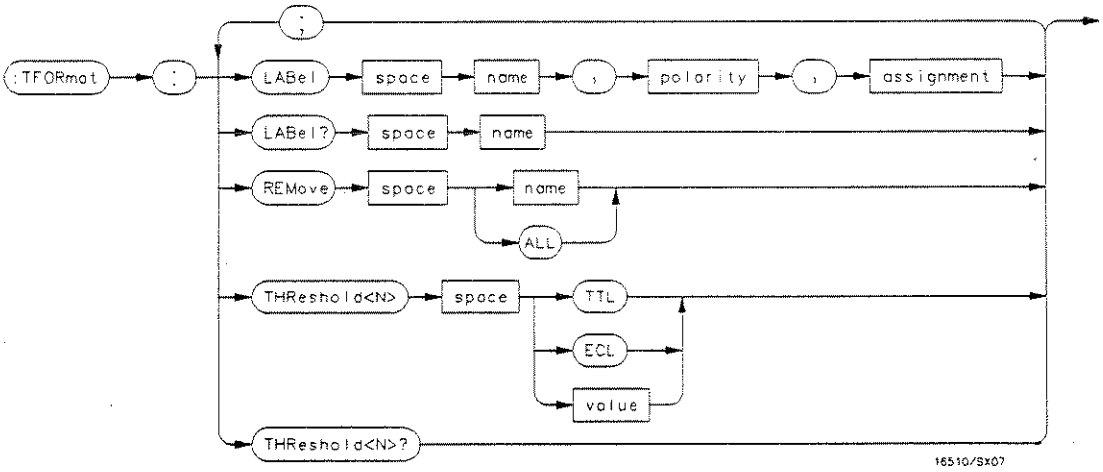
Example:
```
10 DIM Xt$[100]
20 OUTPUT XXX;":MACHINE1:SLIST:XTAG?"
30 ENTER XXX;Xt$
40 PRINT Xt$
50 END
```

TFORmat Subsystem 13

Introduction

The TFORmat Subsystem contains the commands available for the Timing Format menu in the HP 1650A/51A logic analyzer. These commands are:

- LABel
- REMove
- THReshold



<N> = *{1 | 2 | 3 | 4 | 5}*
name = *string of up to 6 alphanumeric characters*
polarity = *{POSitive | NEGative}*
assignment = {pod<*MS*>_spec[,pod<*MS-1*>_spec,...,pod <*LS_spec*>]}
pod<MS|LS> = *pod number, MS being the highest pod number assigned and LS is the lowest pod number assigned*
_spec = *number equal to or greater than 0 (zero) but less or equal to 65535*
value = *voltage (real number) -9.9 to +9.9*

Figure 13-1. TFORmat Subsystem Syntax Diagram

TFORmat

TFORmat selector

The TFORmat selector is used as part of a compound header to access those settings normally found in the Timing Format menu. It always follows the MACHine selector because it selects a branch directly below the MACHine level in the language tree.

Command Syntax: :MACHine{1|2}:TFORmat

Example: OUTPUT XXX;":MACHine1:TFORmat:LABel?"

LABel command/query

The LABel command allows you to specify polarity and assign channels to new or existing labels. If the specified label name does not match an existing label name, a new label will be created.

The pod number order and the assignment (number) is significant. The order is assumed to be the highest pod number first (left-most) followed by any other assigned pods in decreasing pod number order. When viewing the assignment value in binary (base 2), the binary value represents the bit values in the label. A "1" in a bit position means the associated channel in the pod is included in the label. A "0" in a bit position means the associated channel in the pod is excluded from the label. Since pods contain 16 channels, (2^{16}) -1 is the maximum value for the pod specification and 0 is the minimum value.

The LABel query returns the current definition for the selected (by name) label. If the label does not exist, nothing is returned.

Command Syntax: :MACHine{1|2}:TFORmat:LABel <name>,<polarity>,<assignment>

where:

<name> ::= string of up to 6 alphanumeric characters
<polarity> ::= {POSitive|NEGative}
<assignment> ::= {pod<MS>_spec[,pod<MS-1>_spec,...,pod<LS-spec>]}
pod<MS|LS> ::= pod number, MS being the highest pod number assigned and LS being the lowest pod number assigned
_spec ::= number equal to or greater than 0 (zero) but less than or equal to 65535

Example: OUTPUT XXX; ":MACHine1:TFORmat:LABel 'A',POSITIVE,66535,127,0"

LABel

Query Syntax: :MACHine{1|2}:TFORmat:LABel? <name>

Returned Format: [:MACHine{1|2}:TFORmat:LABel] <name>,<polarity>,<assignment><NL>

Example:

```
10 DIM Lb$ [100]
20 OUTPUT XXX;":MACHINE1:TFORMAT:LABEL?'DATA'"
30 ENTER XXX;Lb$
40 PRINT Lb$
50 END
```

REMove command

The REMove command allows you to delete all labels or any one label specified by name for a given machine.

Command Syntax: :MACHine{1|2}:TFORmat:REMove {<name>|ALL}

where:

<name> ::= string of up to 6 alphanumeric characters

Examples:
```
OUTPUT XXX;":MACHine1:TFORmat:REMove 'A'"
or
OUTPUT XXX;":MACHine1:TFORmat:REMove ALL"
```

THReshold command/query

The THReshold command allows you to set the voltage threshold for a given pod to ECL, TTL or a specific voltage from -9.9V to +9.9V in 0.1 volt increments.

Note

The pod thresholds of pods 1, 2, and 3 can be set independently. The pod thresholds of pods 4 and 5 are slaved together; therefore when you set the threshold on pod 4 or 5, both thresholds will be changed to the specified value.

The THReshold query returns the current threshold for a given pod.

Command Syntax: :MACHine{1|2}:TFORmat:THReshold <N> {TTL|ECL|<value>}

where:

<N> ::= pod number {1|2|3|4|5}
<value> ::= voltage (real number) -9.9 to +9.9
TTL ::= default value of +1.6V
ECL ::= default value of -1.3V

Example:
```
OUTPUT XXX;":MACHine1:TFORmat:THReshold1 4.0"
```

Query Syntax: :MACHine{1|2} :TFORmat:THReshold <N>?"

Returned Format: [:MACHine{1|2}:TFORmat:THReshold <N>] <value> <NL>

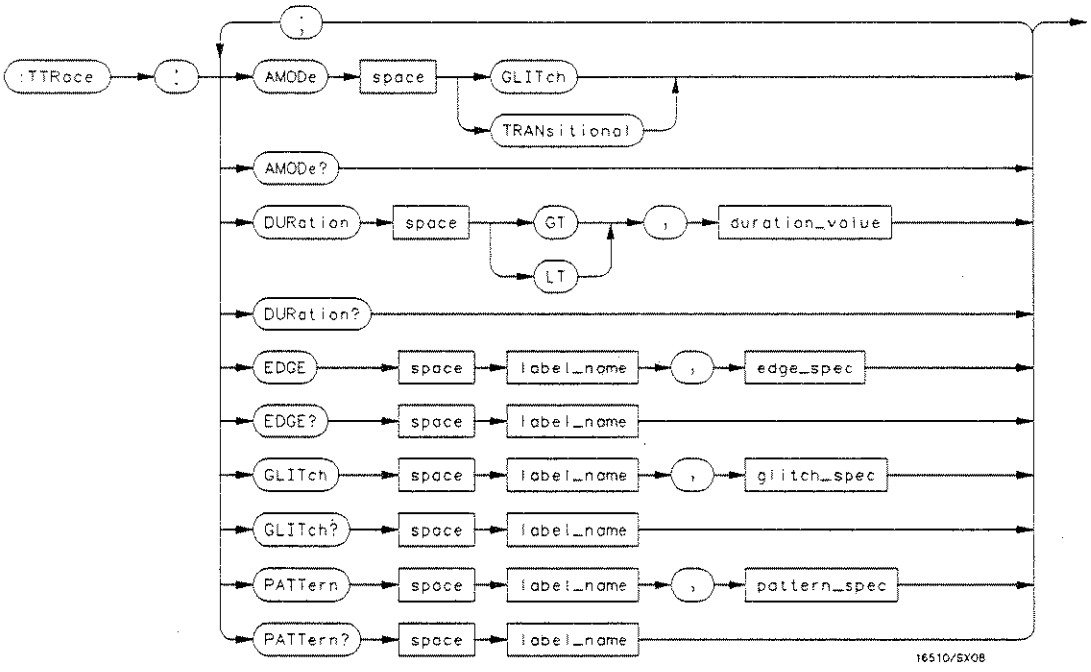
Example:
```
10 DIM Value$ [100]
20 OUTPUT XXX;":MACHINE1:TFORMAT:THRESHOLD2?"
30 ENTER XXX;Value$
40 PRINT Value$
50 END
```

TTRace Subsystem 14

Introduction

The TTRace Subsystem contains the commands available for the Timing Trace Menu in the HP 1650A/51A logic analyzer. These commands are:

- AMODe
- DURation
- EDGE
- GLITch
- PATTern



P/O Figure 14-1. TTRace Subsystem Syntax Diagram

GT = *greater than*
LT = *less than*
duration_value = *real number*
label_name = *string of up to 6 alphanumeric characters*
edge_spec = *string of characters* {*R*|*F*|*T*|*X*}
R = *rising edge*
F = *falling edge*
T = *toggling or either edge*
X = *don't care or ignore this channel*
glitch_spec = *string of characters* {*|.}
***** = *search for a glitch on this channel*
. = *ignore this channel*
pattern_spec = *string in one of the following forms:*
"#B01X..." for binary
"#Q01234567X..." for octal
"#H0123456789ABCDEFX..." for hex
"0123456789..." for decimal

P/O Figure 14-1. TTRace Subsystem Syntax Diagram

TTRace Selector

The TTRace selector is used as part of a compound header to access the settings found in the Timing Trace menu. It always follows the MACHine selector because it selects a branch directly below the MACHine level in the language tree.

Command Syntax: :MACHine{1|2}:TTRace

Example: OUTPUT XXX;":MACHine1:TTRace:GLITCH 'ABC','....****"

AMODe

command/query

The AMODe command allows you to select the acquisition mode used for a particular timing trace. The acquisition modes available are TRANsitional and GLITch.

The AMODe query returns the current acquisition mode.

Command Syntax: :MACHine{1|2}:TTRace:AMODe <acquisition_mode>

where:

<acquisition_mode> ::= {GLITch|TRANsitional}

Example:
```
OUTPUT XXX; ":MACHine1:TTRace:AMODe GLITch"
```

Query Syntax: :MACHine1:TTRace:AMODe?

Returned Format: [:MACHine1:TTRace:AMODe] {GLITCH|TRANSITIONAL}

Example:
```
10 DIM M$[100]
20 OUTPUT XXX; ":MACHINE1:TTRACE:AMODE?"
30 ENTER XXX;M$
40 PRINT M$
50 END
```

DURation command/query

The DURation command allows you to specify the duration qualifier to be used with the pattern recognizer term in generating the timing trigger. The duration value can be specified in 10 ns increments within the following ranges:

- Greater than (GT) qualification - 30 ns to 10 ms.
- Less than (LT) qualification - 40 ns to 10 ms.

The DURation query returns the current pattern duration qualifier specification.

Command Syntax: :MACHine{1|2}:TTRace:DURation {GT|LT},<duration_value>

where:

GT ::= greater than
LT ::= less than
<duration_value> ::= real number

Example:
```
OUTPUT XXX; ":MACHine1:TTRace:DURation GT, 40.0E-9"
```

Query Syntax: :MACHine{1|2}:TTRace:DURation?

Returned Format: [:MACHine{1|2}:TTRace:DURation] {GT|LT},<duration_value><NL>

Example:
```
10 DIM D$[100]
20 OUTPUT XXX; ":MACHINE1:TTRACE:DURATION?"
30 ENTER XXX;D$
40 PRINT D$
50 END
```

EDGE

command/query

The EDGE command allows you to specify the edge recognizer term for the timing analyzer trigger on a per label basis. Each command deals with only one label in the given edge specification; therefore, a complete specification could require several commands. The edge specification uses the characters R, F, T, X to indicate the edges or don't cares as follows:

R = rising edge
F = falling edge
T = toggling or either edge
X = don't care or ignore the channel

The position of these characters in the string corresponds with the position of the channels within the label. All channels without "X" are ORed together to form the edge trigger specification.

The EDGE query returns the edge specification for the specified label.

Command Syntax: :MACHine{1|2}:TTRace:EDGE <label_name>,<edge_spec>

where:

<label_name> ::= string or up to 6 alphanumeric characters
<edge_spec> ::= string of characters {R|F|T|X}

Example: OUTPUT XXX; ":MACHine1:TTRace:EDGE 'POD1','XXXXXXXR'"

EDGE

Query Syntax: :MACHine{1|2}:TTRace:EDGE? <label_name>

Returned Format: [:MACHine{1|2}:TTRace:] <label_name>,<edge_spec><NL>

Example:

```
10 DIM E$[100]
20 OUTPUT XXX; ":MACHINE1:TTRACE:EDGE? 'POD1'"
30 ENTER XXX;E$
40 PRINT E$
50 END
```

GLITch

GLITch **command/query**

The GLITch command allows you to specify the glitch recognizer term for the timing analyzer trigger on a per label basis. Each command deals with only one label in a given glitch specification; therefore, a complete specification could require several commands. The glitch specification uses the characters "*" (asterisk) and "." (period).

where:

"*" (asterisk) ::= search for a glitch on this channel
"." (period) ::= ignore this channel

The position of these characters in the string corresponds with the position of the channels within the label. All channels with the "*" are ORed together to form the glitch trigger specification.

The GLITch query returns the glitch specification for the specified label.

Command Syntax: :MACHine{1|2}:TTRace:GLITch <label_name>,<glitch_spec>

where:

<label_name> ::= string of up to 6 alphanumeric characters
<glitch_spec> ::= string of characters {*|.}

Example: OUTPUT XXX; ":MACHine1:TTRace:GLITch 'POD1','**.....*'"

Query Syntax: :MACHine1:TTRace:GLITch? <label_name>

Returned Format: [:MACHine1:TTRace:GLITch] <label_name>,<glitch_spec><NL>

Example:

```
10 DIM G$[100]
20 OUTPUT XXX; ":MACHINE1:TTRACE:GLITCH? 'POD1'"
30 ENTER XXX;G$
40 PRINT G$
50 END
```

PATTern

command/query

The PATTern command allows you to construct a pattern recognizer term for the timing analyzer trigger on a per label basis. Each command deals with only one label in the given pattern; therefore, a complete timing trace specification could require several commands. Since a label can contain up to 32 bits, the range of the pattern value will be between 0 and (2^{32})-1. When viewing the value of a pattern in binary, the binary value represents the bit values for the label inside the pattern recognizer term. Since a pattern value can contain don't cares and be represented in several bases, the pattern specification parameter is handled as a string of characters instead of a number.

The PATTern query returns the pattern specification for the specified label in the base previously defined for the label.

Command Syntax: :MACHine{1|2}:TTRace:PATTern <label_name>,<pattern_spec>

where:

<label_name> ::= string of up to 6 alphanumeric characters

<pattern_spec> ::= string in one of the following forms:

"#B01X..." for binary
"#Q01234567X..." for octal
"#H0123456789ABCDEFX..." for hex
"0123456789..." for decimal

Example: OUTPUT XXX; ":MACHine1:TTRace:PATTern 'DATA', '255'"

PATTern

Query Syntax: :MACHine{1|2}:TTRace:PATTern? <label_name>

Returned Format: [:MACHine{1|2}:TTRace:PATTern] <label_name>,<pattern_spec><NL>

Example:

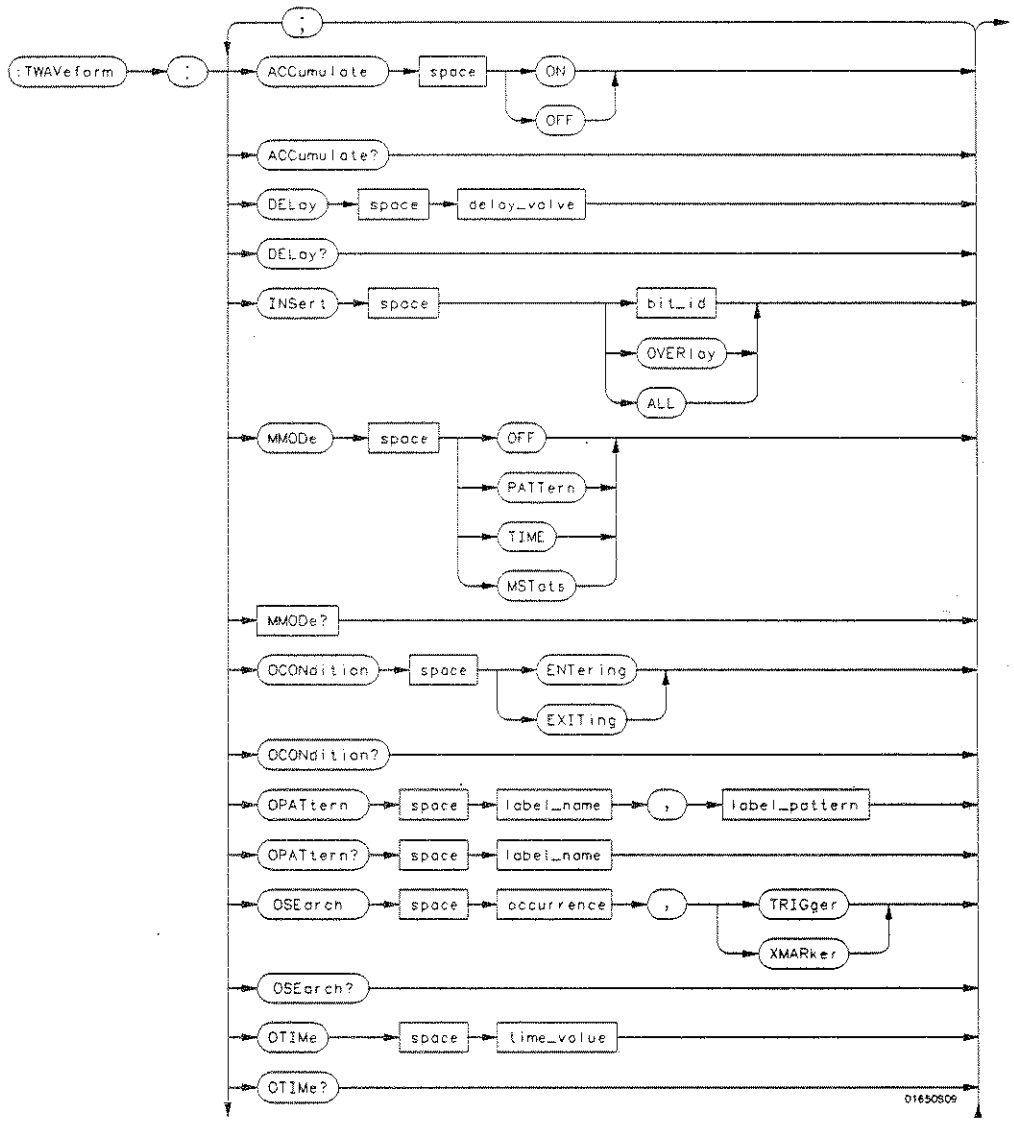
```
10 DIM P$[100]
20 OUTPUT XXX; ":MACHINE2:TTRACE:PATTERN? 'DATA'"
30 ENTER XXX;P$
40 PRINT P$
50 END
```

TWAVeform Subsystem 15

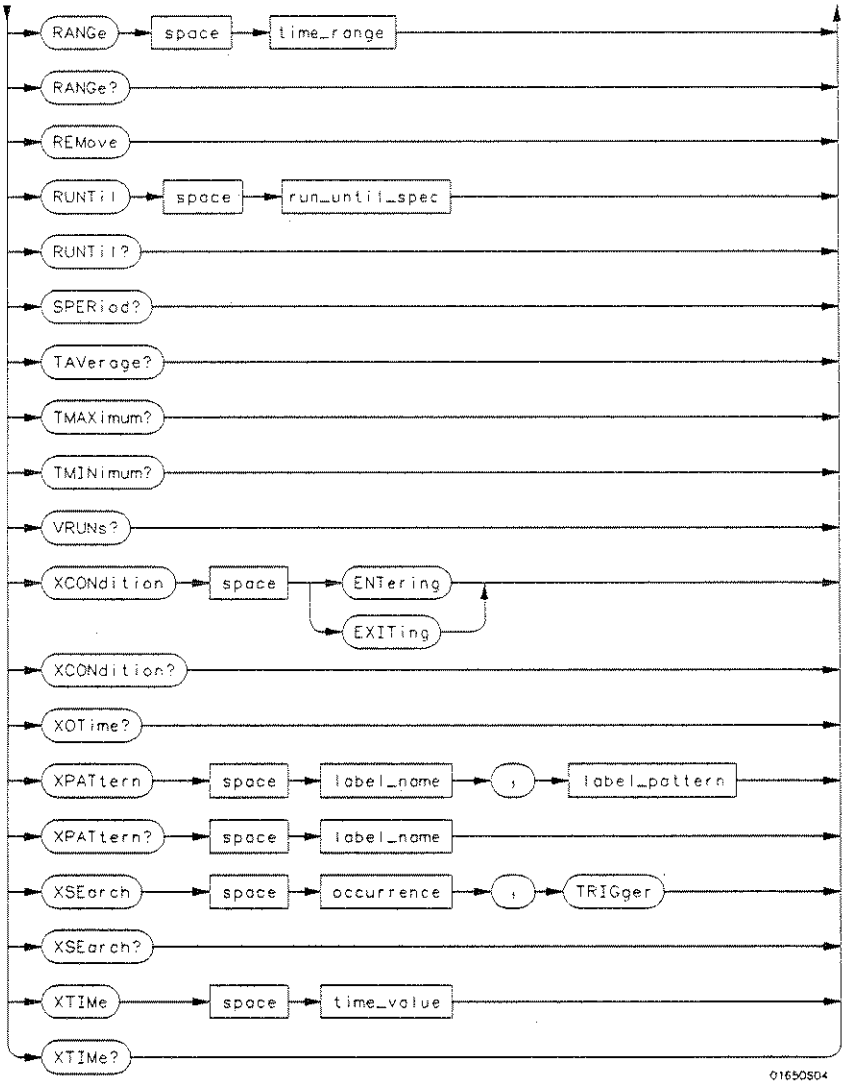
Introduction

The TWAVeform Subsystem contains the commands available for the Timing Waveforms Menu in the HP 1650A/51A logic analyzer. These commands are:

- ACCumulate
- DELay
- INSert
- MMODe
- OCONdition
- OPATtern
- OSEarch
- OTIMe
- RANGe
- REMove
- RUNTil
- SPERiod
- TAVerage
- TMAXimum
- TMINimum
- VRUNs
- XCONdition
- XOTime
- XPATtern
- XSEarch
- XTIMe



P/O Figure 15-1. TWAVeform Subsystem Syntax Diagram



P/O Figure 15-1. TWAVeform Subsystem Syntax Diagram

delay_value = *real number between -2500 s and + 2500 s*
bit_id = *integer from 0 to 31*
label_name = *string of up to 6 alphanumeric characters*
label_pattern = *string in one of the following forms:*
"#B01X..." for binary
"#Q01234567X..." for octal
"#H0123456789ABCDEFX..." for hex
"0123456789..."for decimal
occurrence = *integer*
time_value = *real number*
label_id = *string of one alpha and one numeric character*
time_range = *real number between 100 ns and 10 ks*
run_until_spec =
{OFF|LT, <value> |GT, <value> |INRange <value>, <value> |OUTRange <value>, <value> }
GT = *greater than*
LT = *less than*
value = *real number*

P/O Figure 15-1. TWAVeform Subsystem Syntax Diagram

TWAVeform Selector

The TWAVeform selector is used as part of a compound header to access the settings found in the Timing Waveforms Menu. It always follows the MACHine selector because it selects a branch below the MACHine level in the command tree.

Command Syntax: :MACHine{1|2}:TWAVeform

Example: OUTPUT XXX;":MACHine1:TWAVeform:DELay 100E-9"

ACCumulate command/query

The ACCumulate command controls whether the waveform display gets erased between individual runs (accumulate off) or whether it accumulates (accumulate on).

The ACCumulate query returns the current setting.

Command Syntax: :MACHine{1|2}:TWAVeform:ACCumulate <setting>

where:

<setting> ::= {0|OFF} or {1|ON}

Example:
```
OUTPUT XXX;":MACHine1:TWAVeform:ACCumulate ON"
```

Query Syntax: :MACHine{1|2}:TWAVeform:ACCumulate?

Returned Format: [:MACHine{1|2}:TWAVeform:ACCumulate]{0|1}<NL>

Example:
```
10 DIM P$ [100]
20 OUTPUT XXX;":MACHINE1:TWAVEFORM:ACCUMULATE?"
30 ENTER XXX P$
40 PRINT P$
50 END
```

DELay command/query

The DELay command specifies the amount of time between the timing trigger and the horizontal center of the timing waveform display. The allowable values for delay are -2500 s to +2500 s. In glitch acquisition mode, as delay becomes large in an absolute sense, the sample rate is adjusted so that data will be acquired in the time window of interest. In transitional acquisition mode, data may not fall in the time window since the sample period is fixed at 10 ns and the amount of time covered in memory is dependent on how frequent the input signal transitions occur.

The DELay query returns the current time offset (delay) value from the trigger.

Command Syntax: :MACHine{1|2}:TWAVeform:DELay <delay_value>

where:

<delay_value> ::= real number between -2500 s and +2500 s

Example:
```
OUTPUT XXX ":MACHine1:TWAVeform:DELay 100E-6"
```

Query Syntax: :MACHine{1|2}:TWAVeform:DELay?

Returned Format: [:MACHine{1|2}:TWAVeform:DELay]<time_value><NL>

Example:
```
10 DIM DI$ [100]
20 OUTPUT XXX;":MACHINE1:TWAVEFORM:DELAY?"
30 ENTER XXX DI$
40 PRINT DI$
50 END
```

INSert command

The INSert command inserts waveforms in the timing waveform display. The waveforms are added from top to bottom. When 24 waveforms are present, inserting additional waveforms replaces the last waveform .

The first parameter specifies the label name that will be inserted. The second parameter specifies the label bit number or overlay.

If OVERLAY is specified, all the bits of the label are displayed as a composite overlaid waveform.

Command Syntax: :MACHine{1|2}:TWAVeform:INSert <label_name>,{<bit_id>|OVERlay}

where:

<label_name> ::= string of up to 6 alphanumeric characters
<bit_id> ::= integer from 0 to 31

Example: OUTPUT XXX;":MACHine1:TWAVeform:INSert, 'WAVE',10"

MMODe command/query

The MMODe (Marker Mode) command selects the mode controlling marker movement and the display of the marker readouts. When PATTern is selected, the markers will be placed on patterns. When TIME is selected, the markers move on time. In MSTats, the markers are placed on patterns, but the readouts will be time statistics.

The MMODe query returns the current marker mode.

Command Syntax: :MACHine{1|2}:TWAVeform:MMODe {OFF|PATTern|TIME|MSTats}

Example: OUTPUT XXX; ":MACHine1:TWAVeform:MMODe TIME"

Query Syntax: :MACHine{1|2}:TWAVeform:MMODe?

Returned Format: [:MACHine{1|2}:TWAVeform:MMODe]<marker_mode><NL>

where

<marker_mode> ::= {OFF|PATTern|TIME|MSTats}

Example:

```
10 DIM M$ [100]
20 OUTPUT XXX;":MACHINE1:TWAVEFORM:MMODE?"
30 ENTER XXX M$
40 PRINT M$
50 END
```

OCONdition command/query

The OCONdition command specifies where the O marker is placed. The O marker can be placed on the entry or exit point of the OPATtern when in the PATTern marker mode.

The OCONdition query returns the current setting.

Command Syntax: :MACHine{1|2}:TWAVeform:OCONdition {ENTering|EXITing}

Example: OUTPUT XXX; ":MACHine1:TWAVeform:OCONdition ENTERing"

Query Syntax: :MACHine{1|2}:TWAVeform:OCONdition?

Returned Format: [:MACHine{1|2}:TWAVeform:OCONdition] {ENTering|EXITing}<NL>

Example:
```
10 DIM Oc$ [100]
20 OUTPUT XXX;":MACHINE1:TWAVEFORM:OCONDITION?"
30 ENTER XXX Oc$
40 PRINT Oc$
50 END
```

OPATtern command/query

The OPATtern command allows you to construct a pattern recognizer term for the O marker which is then used with the OSEarch criteria and OCONdition when moving the marker on patterns. Since each command deals with only one label in the pattern recognizer, a complete specification could require several commands. Since a label can contain up to 32 bits, the range of the pattern value will be from 0 to (2^{32})-1. When viewing the value of a pattern in binary, the binary value represents the bit values for the label inside the pattern recognizer term.

The OPATtern query, in pattern marker mode, returns the pattern specification for a given label name. In the time marker mode, the query returns the pattern under the O marker for a given label. If the O marker is not placed on valid data, dashes (- - - -) are returned.

Command Syntax: :MACHine{1|2}:TWAVeform:OPATtern <label_name>,<label_pattern>

where:

<label_name> ::= string of up to 6 alphanumeric characters

<label_pattern> ::= string in one of the following forms:

"#B01X..." for binary
"#Q01234567X..." for octal
"#H0123456789ABCDEFX..." for hex
"0123456789..."for decimal

Example: OUTPUT XXX; ":MACHine1:TWAVeform:OPATtern 'A',511"

OPATtern

Query Syntax: :MACHine{1|2}:TWAVeform:OPATtern? <label_name>

Returned Format: [:MACHine{1|2}:TWAVeform:OPATtern]<label_name>,
<label_pattern><NL>

Example:
```
10 DIM Op$ [100]
20 OUTPUT XXX;":MACHINE1:TWAVEFORM:OPATTERN? 'A'"
30 ENTER XXX Op$
40 PRINT Op$
50 END
```

OSEarch

command/query

The OSEarch command defines the search criteria for the O marker which is then used with the associated OPATtern recognizer specification and the OCONdition when moving markers on patterns. The origin parameter tells the marker to begin a search with the trigger or with the X marker. The actual occurrence the marker searches for in the OPATtern recognizer specification is determined by the occurrence parameter, relative to the origin. An occurrence of 0 places a marker on the selected origin. With a negative occurrence, the marker searches before the origin. With a positive occurrence, the marker searches after the origin.

The OSEarch query returns the search criteria for the O marker.

Command Syntax: :MACHine{1|2}:TWAVeform:OSEarch <occurrence>,<origin>

where:

<origin> ::= {TRIGger|XMARker}
<occurrence> ::= integer

Example:
```
OUTPUT XXX; ":MACHine1:TWAVeform:OSEarch +10,TRIGger"
```

Query Syntax: :MACHine{1|2}:TWAVeform:OSEarch?

Returned Format: [:MACHine{1|2}:TWAVeform:OSEarch] <occurrence>,<origin><NL>

Example:
```
10 DIM Os$ [100]
20 OUTPUT XXX;":MACHINE1:TWAVEFORM:OSEARCH?"
30 ENTER XXX Os$
40 PRINT Os$
50 END
```

OTIMe

command/query

The OTIMe command positions the O marker in time when the marker mode is TIME. If data is not valid, the command performs no action.

The OTIMe query returns the O marker position in time. If data is not valid, the query returns 9.9E37.

Command Syntax: :MACHine{1|2}:TWAVeform:OTIMe <time_value>

where:

<time_value> :: = real number

Example:
```
OUTPUT XXX; ":MACHine1:TWAVeform:OTIMe 30.0E-6"
```

Query Syntax: :MACHine{1|2}:TWAVeform:OTIMe?

Returned Format: [:MACHine{1|2}:TWAVeform:OTIMe]<time_value><NL>

Example:
```
10 DIM Ot$ [100]
20 OUTPUT XXX;":MACHINE1:TWAVEFORM:OTIME?"
30 ENTER XXX Ot$
40 PRINT Ot$
50 END
```

RANGe command/query

The RANGe command specifies the full-screen time in the timing waveform menu. It is equivalent to ten times the seconds-per-division setting on the display. The allowable values for RANGe are from 100 ns to 10 ks.

The RANGe query returns the current full-screen time.

Command Syntax: :MACHine{1|2}:TWAVeform:RANGe <time_value>

where:

<time_range> ::= real number between 100 ns and 10 ks

Example:
```
OUTPUT XXX;":MACHine1:TWAVeform:RANGe 100E-9"
```

Query Syntax: :MACHine{1|2}:TWAVeform:RANGe?

Returned Format: [:MACHine{1|2}:TWAVeform:RANGe]<time_value><NL>

Example:
```
10 DIM Rg$ [100]
20 OUTPUT XXX;":MACHINE1:TWAVEFORM:RANGE?"
30 ENTER XXX Rg$
40 PRINT Rg$
50 END
```

REMove command

The REMove command deletes all waveforms from the display.

Command Syntax: :MACHine{1|2}:TWAVeform:REMove

Example: OUTPUT XXX;":MACHine1:TWAVeform:REMove"

RUNTil

command/query

The RUNTil (run until) command defines stop criteria based on the time between the X and O markers when the trace mode is in repetitive. When OFF is selected, the analyzer will run until either the STOP key is pressed or the STOP command is sent. Run until the time between X and O marker options are:

- OFF
- Less Than (LT) a specified time value
- Greater Than (GT) a specified time value
- In the range (INRange) between two time values
- Out of the range (OUTRange) between two time values

End points for the INRange and OUTRange should be at least 10 ns apart since this is the minimum time at which data is sampled.

The RUNTil query returns the current stop criteria.

Command Syntax: :MACHine{1|2}:TWAVeform:RUNTil <run_until_spec>

where:

<run_until_spec> ::= {OFF|LT,<value>|GT,<value>|INRange<value>,<value>|OUTRange<value>,<value>}

<value> ::= real number

Example: OUTPUT XXX; ":MACHine1:TWAVeform:RUNTil GT,800.0E-6"

RUNTil

Query Syntax: :MACHine{1|2}:TWAVeform:RUNTil?

Returned Format: [:MACHine{1|2}:TWAVeform:RUNTil]<run_until_spec><NL>

Example:

```
10 DIM Ru$ [100]
20 OUTPUT XXX;":MACHINE1:TWAVEFORM:RUNTIL?"
30 ENTER XXX Ru$
40 PRINT Ru$
50 END
```

SPERiod query

The SPERiod query returns the sample period of the last run.

Query Syntax: :MACHine{1|2}:TWAVeform:SPERiod?

Returned Format: [:MACHine{1|2}:TWAVeform:SPERiod] <time_value><NL>

where:

<time_value> ::= real number

Example:

```
10 DIM Sp$ [100]
20 OUTPUT XXX;":MACHINE1:TWAVEFORM:SPERIOD?"
30 ENTER XXX Sp$
40 PRINT Sp$
50 END
```

TAVerage

TAVerage query

The TAVerage query returns the value of the average time between the X and O markers. If there is no valid data, the query returns 9.9E37.

Query Syntax: :MACHine{1|2}:TWAVeform:TAVerage?

Returned Format: [:MACHine{1|2}:TWAVeform:TAVerage]<time_value><NL>

where:

<time_value> ::= real number

Example:

```
10 DIM Tv$ [100]
20 OUTPUT XXX;":MACHINE1:TWAVEFORM:TAVERAGE?"
30 ENTER XXX Tv$
40 PRINT Tv$
50 END
```

TMAXimum query

The TMAXimum query returns the value of the maximum time between the X and O markers. If there is no valid data, the query returns 9.9E37.

Query Syntax: :MACHine{1|2}:TWAVeform:TMAXimum?

Returned Format: [:MACHine{1|2}:TWAVeform:TMAXimum]<time_value><NL>

where

<time_value> ::= real number

Example:

```
10 DIM Tx$ [100]
20 OUTPUT XXX;":MACHINE1:TWAVEFORM:TMAXIMUM?"
30 ENTER XXX Tx$
40 PRINT Tx$
50 END
```

TMINimum

query

The TMINimum query returns the value of the minimum time between the X and O markers. If there is no valid data, the query returns 9.9E37.

Query Syntax: :MACHine{1|2}:TWAVeform:TMINimum?

Returned Format: [:MACHine{1|2}:TWAVeform:TMINimum]<time_value><NL>

where:

<time_value> ::= real number

Example:

```
10 DIM Tm$ [100]
20 OUTPUT XXX;":MACHINE1:TWAVEFORM:TMINIMUM?"
30 ENTER XXX Tm$
40 PRINT Tm$
50 END
```

VRUNs — query

The VRUNs query returns the number of valid runs and total number of runs made. Valid runs are those where the pattern search for both the X and O markers was successful resulting in valid delta time measurements.

Query Syntax: :MACHine{1|2}:TWAVeform:VRUNs?

Returned Format: [:MACHine{1|2}:TWAVeform:VRUNs]<valid_runs>,<total_runs><NL>

where:

<valid_runs> ::= zero or positive integer
<total_runs> ::= zero or positive integer

Example:

```
10 DIM Vr$ [100]
20 OUTPUT XXX;":MACHINE1:TWAVEFORM:VRUNS?"
30 ENTER XXX Vr$
40 PRINT Vr$
50 END
```

XCONdition

command/query

The XCONdition command specifies where the X marker is placed. The X marker can be placed on the entry or exit point of the XPATtern when in the PATTern marker mode.

The XCONdition query returns the current setting.

Command Syntax: :MACHine{1|2}:TWAVeform:XCONdition {ENTering|EXITing}

Example: OUTPUT XXX; ":MACHine1:TWAVeform:XCONdition ENTering"

Query Syntax: :MACHine{1|2}:TWAVeform:XCONdition?

Returned Format: [:MACHine{1|2}:TWAVeform:XCONdition] {ENTering|EXITing}<NL>

Example:

```
10 DIM Xc$ [100]
20 OUTPUT XXX;":MACHINE1:TWAVEFORM:XCONDITION?"
30 ENTER XXX Xc$
40 PRINT Xc$
50 END
```

XOTime query

The XOTime query returns the time from the X marker to the O marker. If data is not valid, the query returns 9.9E37.

Query Syntax: :MACHine{1|2}:TWAVeform:XOTime?

Returned Format: [:MACHine{1|2}:TWAVeform:XOTime]<time_value><NL>

where:

<time_value> ::= real number

Example:

```
10 DIM Xot$ [100]
20 OUTPUT XXX;":MACHINE1:TWAVEFORM:XOTIME?"
30 ENTER XXX Xot$
40 PRINT Xot$
50 END
```

XPATtern

command/query

The XPATtern command allows you to construct a pattern recognizer term for the X marker which is then used with the XSEarch criteria and XCONdition when moving the marker on patterns. Since each command deals with only one label in the pattern recognizer, a complete specification could require several commands. Since a label can contain up to 32 bits, the range of the pattern value will be from 0 to (2^{32})-1. When viewing the value of a pattern binary, the binary value represents the bit values for the label inside the pattern recognizer term.

The XPATtern query, in pattern marker mode, returns the pattern specification for a given label name. In the time marker mode, the query returns the pattern under the X marker for a given label. If the X marker is not placed on valid data, dashes (- - - -) are returned.

Command Syntax: :MACHine{1|2}:TWAVeform:XPATtern <label_name>,<label_pattern>

where:

<label_name> ::= string of up to 6 alphanumeric characters

<label_pattern> ::= string in one of the following forms:

"#B01X..." for binary
"#Q01234567X..." for octal
"#H0123456789ABCDEFX..." for hex
"0123456789..."for decimal

Example: OUTPUT XXX; ":MACHine1:TWAVeform:XPATtern 'A',511"

Query Syntax: :MACHine{1|2}:TWAVeform:XPATtern? <label_name>

Returned Format: [:MACHine{1|2}:TWAVeform:XPATtern] <label_name>
<label_pattern><NL>

Example:
```
10 DIM Xp$ [100]
20 OUTPUT XXX;":MACHINE1:TWAVEFORM:XPATTERN? 'A'"
30 ENTER XXX Xp$
40 PRINT Xp$
50 END
```

XSEarch

command/query

The XSEarch command defines the search criteria for the X marker which is then used with the associated XPATtern recognizer specification and the XCONdition when moving markers on patterns. The origin parameter tells the marker to begin a search with the trigger. The occurrence parameter determines which occurrence of the XPATtern recognizer specification, relative to the origin, the marker actually searches for. An occurrence of 0 (zero) places a marker on the origin.

The XSEarch query returns the search criteria for the X marker.

Command Syntax: :MACHine{1|2}:TWAVeform:XSEarch <occurrence>,<origin>

where:

<origin> ::= TRIGger
<occurrence> ::= integer

Example:
```
OUTPUT XXX; ":MACHine1:TWAVeform:XSEarch+10,TRIGger"
```

Query Syntax: :MACHine{1|2}:TWAVeform:XSEarch? <occurrence>,<origin>

Returned Format: [:MACHine{1|2}:TWAVeform:XSEarch] <occurrence>,<origin><NL>

Example:
```
10 DIM Xs$ [100]
20 OUTPUT XXX;":MACHINE1:TWAVEFORM:XSEARCH?"
30 ENTER XXX Xs$
40 PRINT Xs$
50 END
```

XTIMe command/query

The XTIMe command positions the X marker in time when the marker mode is TIME. If data is not valid, the command performs no action.

The XTIMe query returns the X marker position in time. If data is not valid, the query returns 9.9E37.

Command Syntax: :MACHine{1|2}:TWAVeform:XTIMe <time_value>

where:

<time_value> ::= real number

Example:
```
OUTPUT XXX; ":MACHine1:TWAVeform:XTIMe 40.0E-6"
```

Query Syntax: :MACHine{1|2}:TWAVeform:XTIMe?

Returned Format: [:MACHine{1|2}:TWAVeform:XTIMe]<time_value><NL>

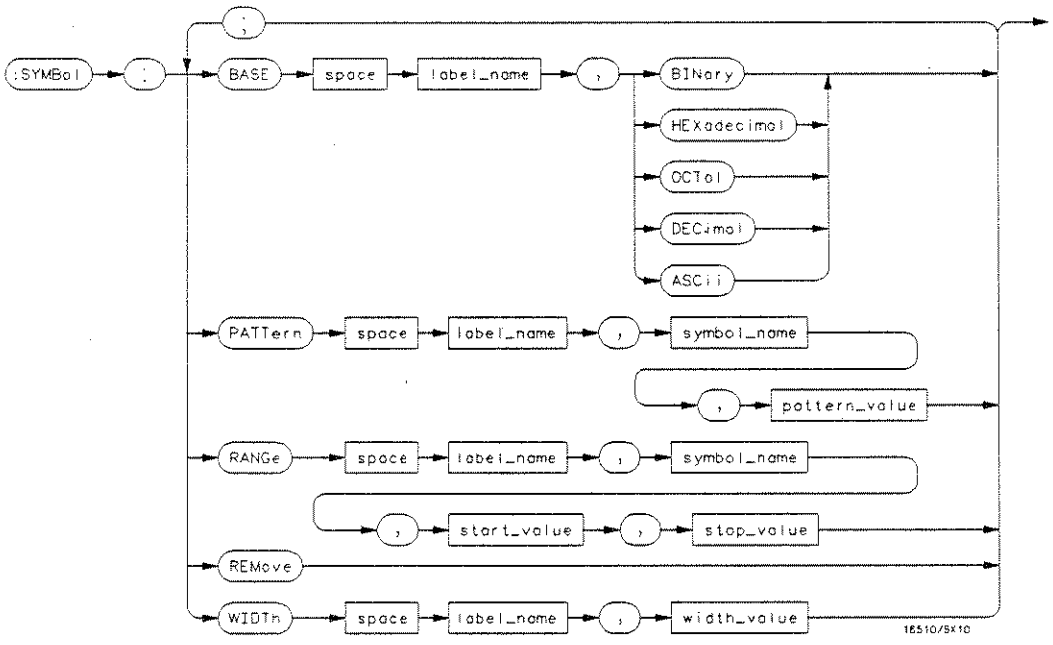
Example:
```
10 DIM Xt$ [100]
20 OUTPUT XXX;":MACHINE1:TWAVEFORM:XTIME?"
30 ENTER XXX Xt$
40 PRINT Xt$
50 END
```

SYMBol Subsystem 16

Introduction

The SYMBol subsystem contains the commands that allow you to define symbols on the controller and download them to the HP 1650A/51A Logic Analyzer. The commands in this subsystem are:

- BASE
- PATTern
- RANGe
- REMove
- WIDTh



P/O Figure 16-1. SYMBol Subsystem Syntax Diagram

<label_name> = *string of up to 6 alphanumeric characters*
<symbol_name> = *string of up to 16 alphanumeric characters*
<pattern_value> = *string of one of the following forms:*
"#B01X..." for binary
"#Q01234567X.." for octal
"#H0123456789ABCDEFX..." for hexadecimal
"0123456789..." for decimal
<start_value> = *string of one of the following forms:*
"#B01..." for binary
"#Q01234567.." for octal
"#H0123456789ABCDEF..." for hexadecimal
"0123456789..." for decimal
<stop_value> = *string of one of the following forms:*
"#B01..." for binary
"#Q01234567.." for octal
"#H0123456789ABCDEF..." for hexadecimal
"0123456789..." for decimal
<width_value> = *integer from 1 to 16*

P/O Figure 16-1. SYMBol Subsystem Syntax Diagram

SYMBol selector

The SYMBol selector is used as a part of a compound header to access the commands used to create symbols. It always follows the MACHine selector because it selects a branch directly below the MACHine level in the command tree.

Command Syntax: :MACHine{1|2}:SYMBol

Example: OUTPUT XXX;":MACHine1:SYMBol:BASE 'DATA',BINary"

BASE command

The BASE command sets the base in which symbols for the specified label will be displayed in the symbol menu. It also specifies the base in which the symbol offsets are displayed when symbols are used.

Note

BINary is not available for labels with more than 20 bits assigned. In this case the base will default to HEXadecimal.

Command Syntax: :MACHine{1|2}:SYMBol:BASE <label_name>,<base_value>

where:

<label_name> ::= string of up to 6 alphanumeric characters
<base_value> ::= {BINary | HEXadecimal | OCTal | DECimal | ASCii}

Example: OUTPUT XXX;":MACHine1:SYMBol:BASE 'DATA',HEXadecimal"

PATTern command

The PATTern command allows you to create a pattern symbol for the specified label. The pattern may contain "don't cares" in the form of XX...X's.

Command Syntax: :MACHine{1|2}:SYMBol:PATTern <label_name>,<symbol_name>,<pattern_value>

where:

<label_name> ::= string of up to 6 alphanumeric characters
<symbol_name> ::= string of up to 16 alphanumeric characters
<pattern_value> ::= string of one of the following forms:

"#B01X..." for binary
"#Q01234567X.." for octal
"#H0123456789ABCDEFX..." for hexadecimal
"0123456789..." for decimal

Example: OUTPUT XXX;":MACHine1:SYMBol:PATTern 'STAT', 'MEM_RD','#H01XX'"

RANGe command

The RANGe command allows you to create a range symbol containing a start value and a stop value for the specified label.

Note

Don't cares are not allowed in range symbols.

Command Syntax: :MACHine{1|2}:SYMBol:RANGe <label_name>,<symbol_name>, <start_value>,<stop_value>

where:

<label_name> ::= string of up to 6 alphanumeric characters
<symbol_name> ::= string of up to 16 alphanumeric characters
<start_value> ::= string of one of the following forms:

"#B01..." for binary
"#Q01234567.." for octal
"#H0123456789ABCDEF..." for hexadecimal
"0123456789..." for decimal

<stop_value> ::= string of one of the following forms:

"#B01..." for binary
"#Q01234567.." for octal
"#H0123456789ABCDEF..." for hexadecimal
"0123456789..." for decimal

Example: OUTPUT XXX;":MACHine1:SYMBol:RANGe 'STAT', 'IO_ACCESS','#H0000','#H000F'"

REMove command

The REMove command deletes all symbols from a specified machine.

Command Syntax: :MACHine{1|2}:SYMBol:REMove

Example: OUTPUT XXX;":MACHine1:SYMBol:REMove"

WIDTh command

The WIDTh command specifies the width (number of characters) in which the symbol names will be displayed when symbols are used.

Note

The WIDTh command does not affect the displayed length of the symbol offset value.

Command Syntax: :MACHine{1|2}:SYMBol:WIDTh <label_name>,<width_value>

where:

<label_name> ::= string of up to 6 alphanumeric characters
<width_value> ::= integer from 1 to 16

Example: OUTPUT XXX;":MACHine1:SYMBol:WIDTh 'DATA',9 "

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. Although the HP 1650A and HP 1651A logic analyzers are RS-232C instruments, they were designed to be compatible with other Hewlett-Packard IEEE 488.2 compatible instruments.

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 1650A/51A are structured very similar 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.

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 recognizes 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, the input buffer and output queue are cleared, and the parser is reset to the root level of the command tree.

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.

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.

Command Error. A command error will be reported if the instrument detects a syntax error or an unrecognized command header.

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.

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). 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 MACHINE1:ASSIGN 2,3 is an example of a compound header. It places the parser in the machine subsystem until the <NL> is encountered.

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

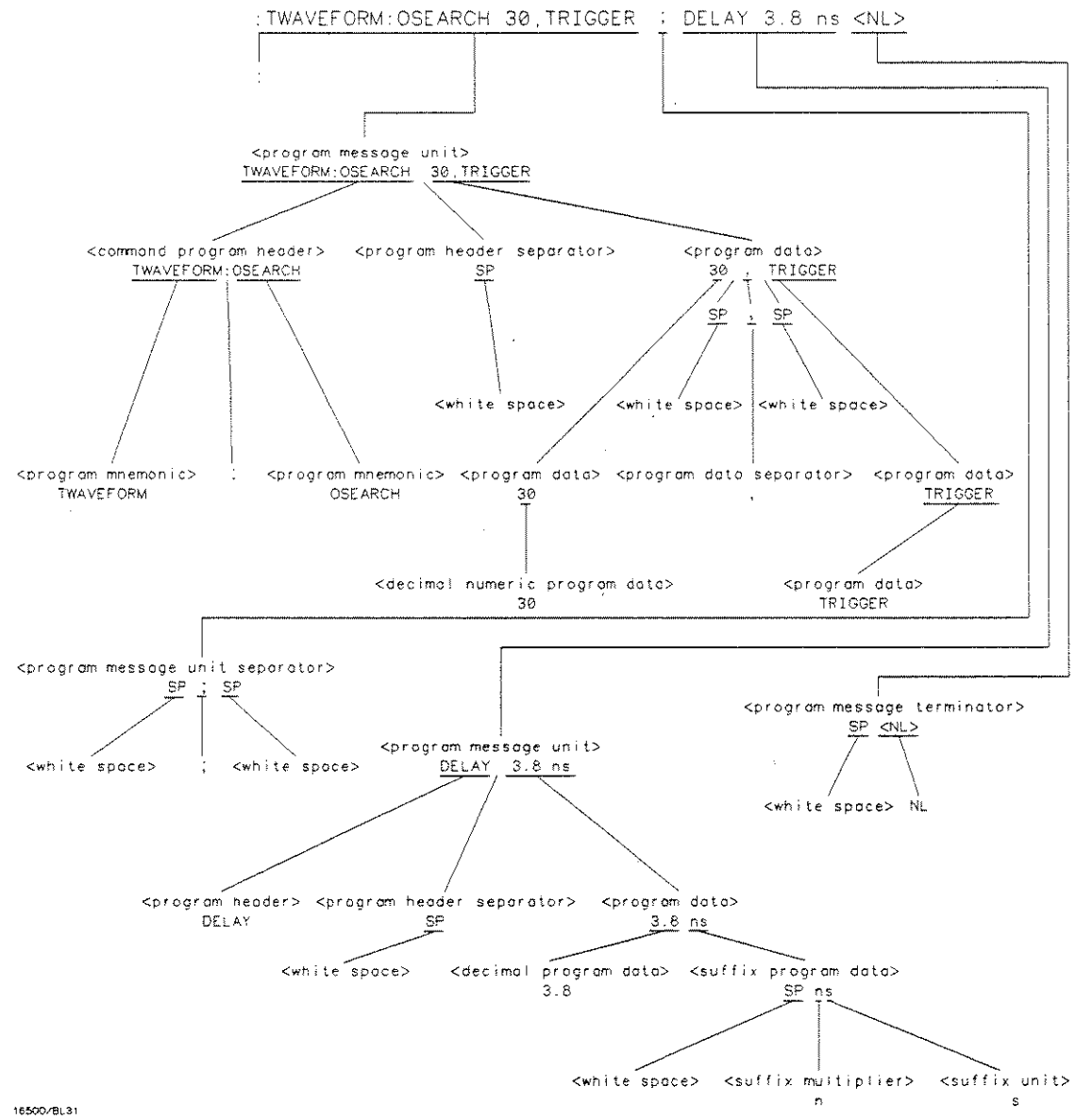


Figure A-1. <program message> Parse Tree

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.

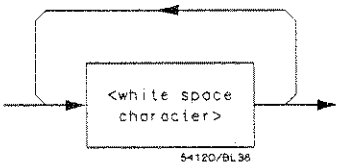


Figure A-2. < white space >

< 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.

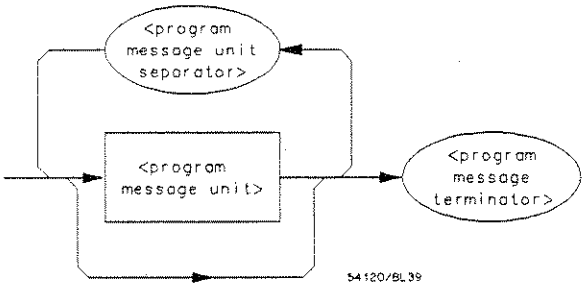


Figure A-3. < program message >

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

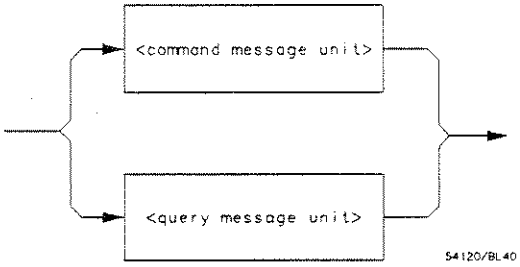


Figure A-4. < program message unit >

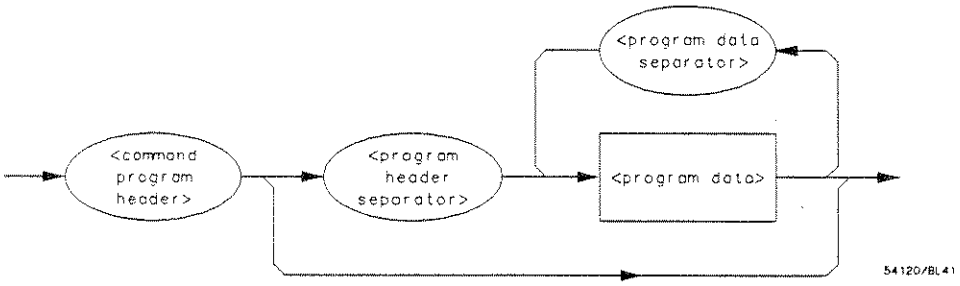


Figure A-5. <command message unit>

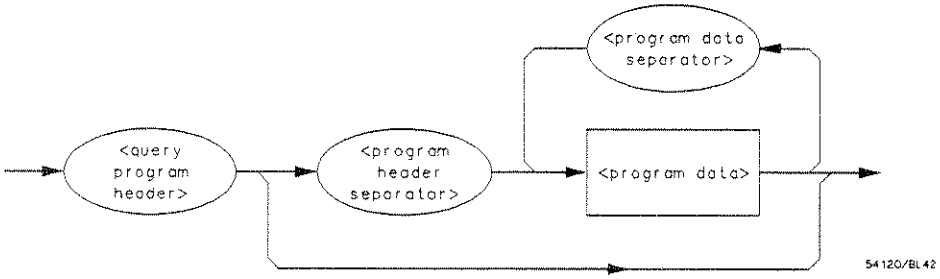


Figure A-6. <query message unit>

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

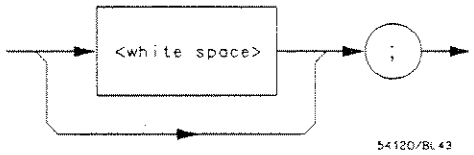


Figure A-7. < program message unit separator >

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

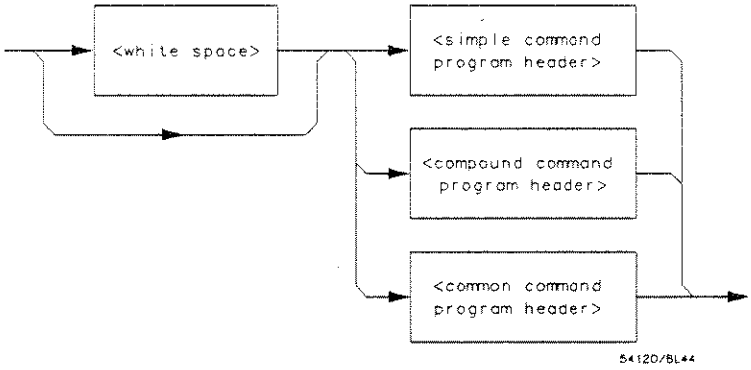
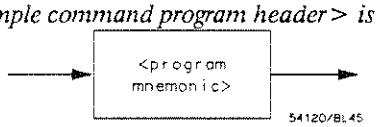
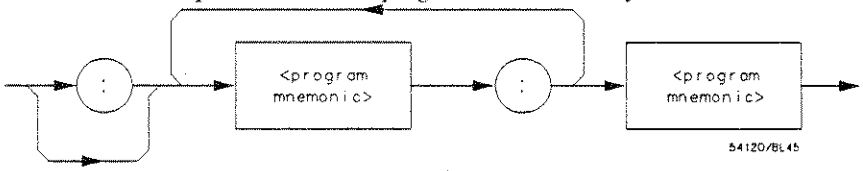


Figure A-8. < command program header >

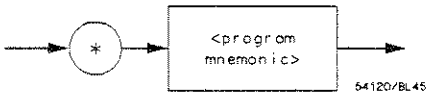
Where <simple command program header> is defined as



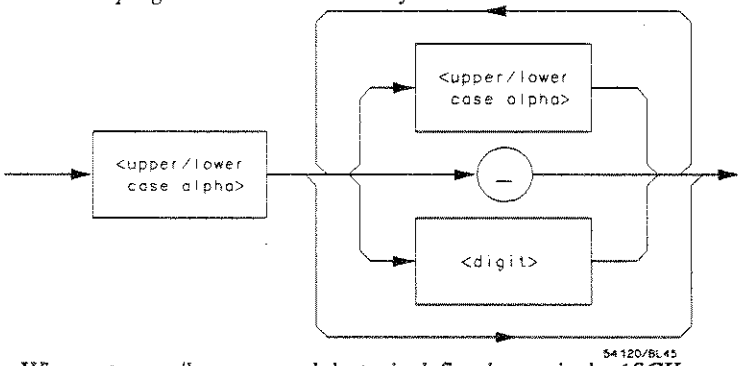
Where <compound command program header> is defined as



Where <common command program header> is defined as



Where <program mnemonic> is defined as

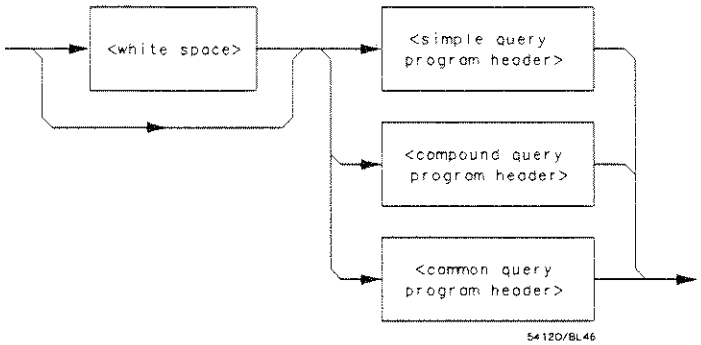


Where <upper/lower case alpha> is defined as a single ASCII encoded byte in the range 41 - 5A, 61 - 7A (65 - 90, 97 - 122 decimal).

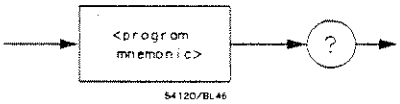
Where <digit> is defined as a single ASCII encoded byte in the range 30 - 39 (48 - 57 decimal).

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

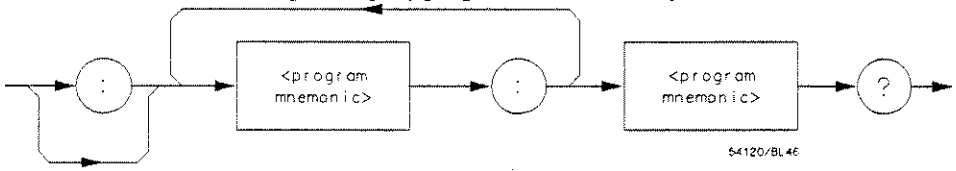
Figure A-8. <command program header> (continued)



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 1650A/51A will accept the following data types: < character program data >, < decimal numeric program data >, < suffix program data >, < string program data >, and < arbitrary block program data >.

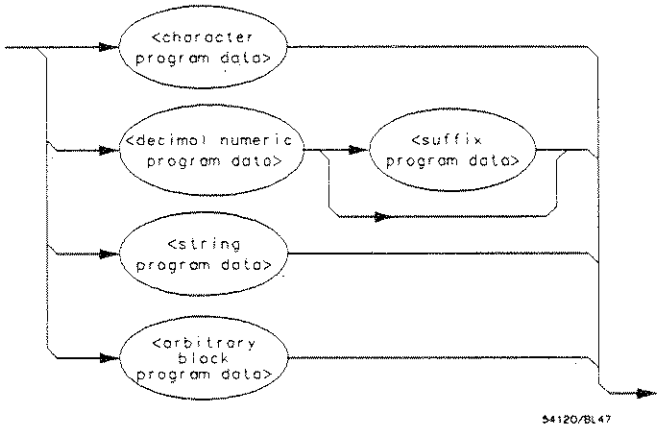


Figure A-10. < program data >

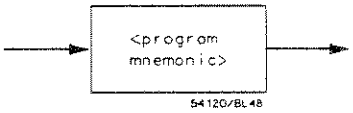
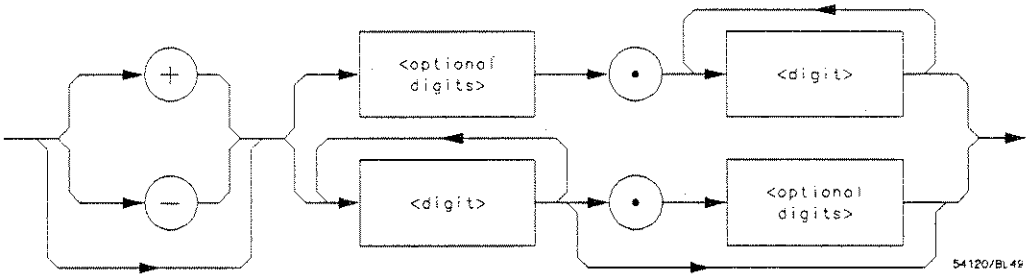
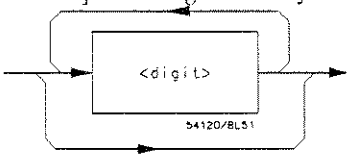


Figure A-11. < character program data >

Where <mantissa> is defined as



Where <optional digits> is defined as



Where <exponent> is defined as

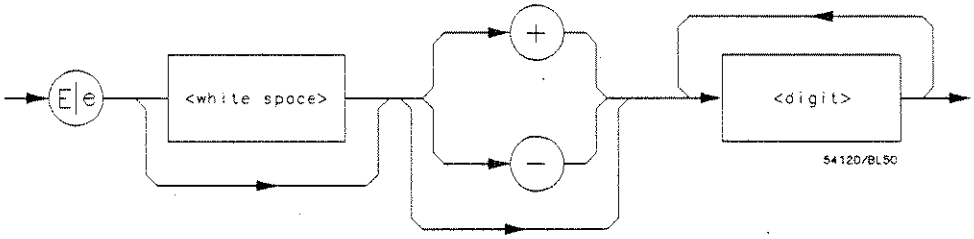


Figure A-12. <decimal numeric program data>

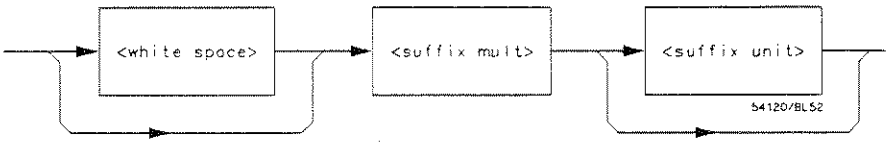


Figure A-13. <suffix program data>

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

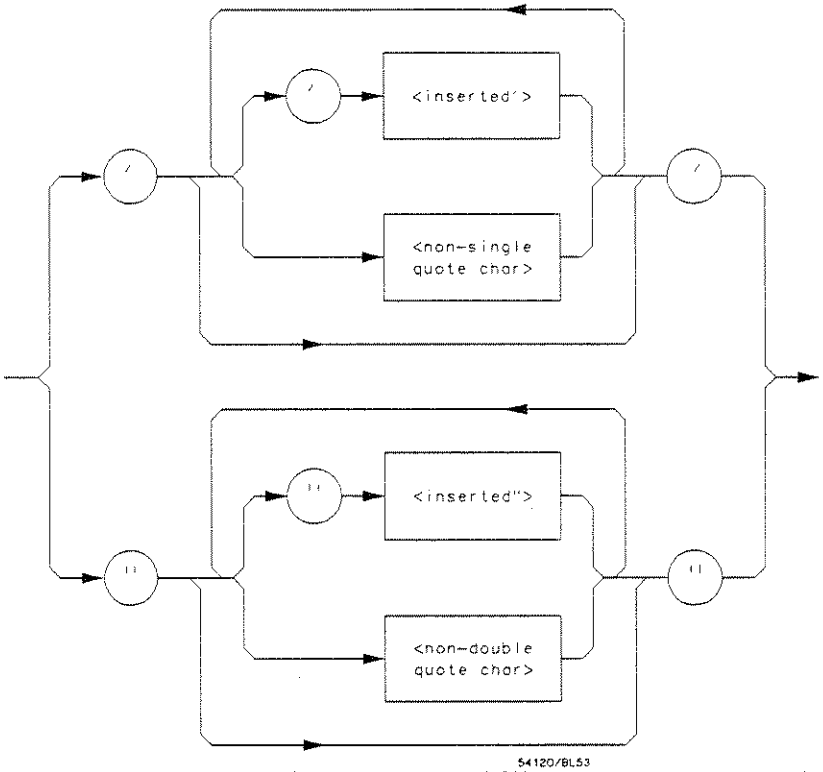
Table A-1. <suffix mult>

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



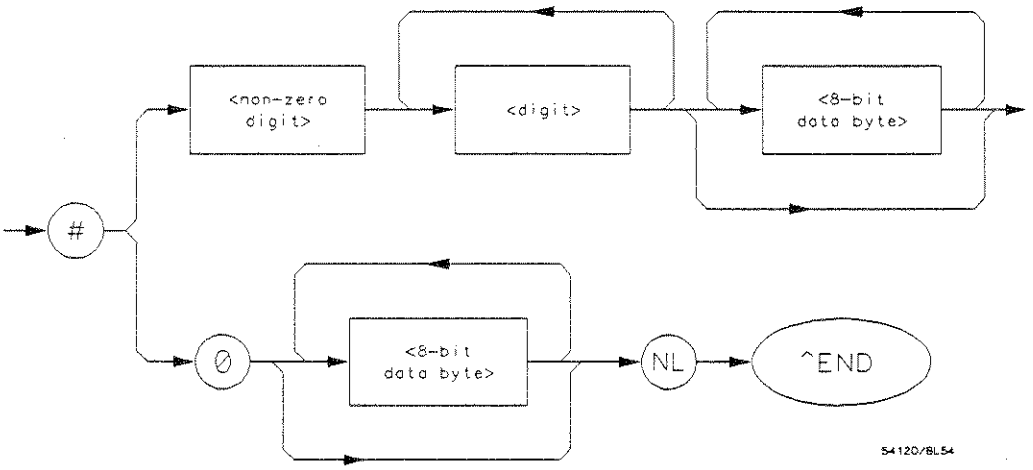
Where <inserted '> is defined as a single ASCII character with the value 27 (39 decimal).

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.

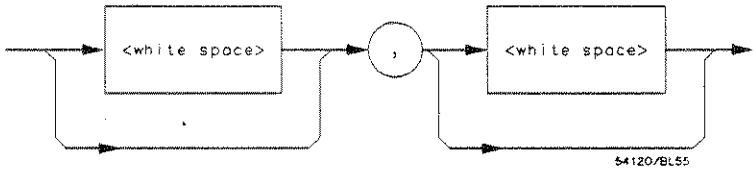


Figure A-16. <program data separator>

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

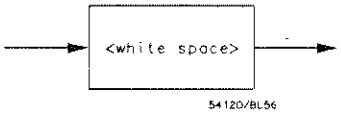
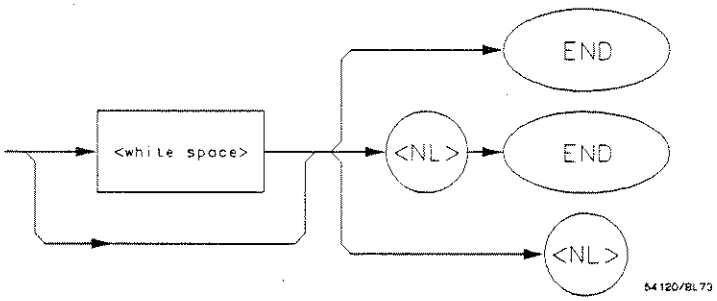


Figure A-17. < program header separator >

< 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 >

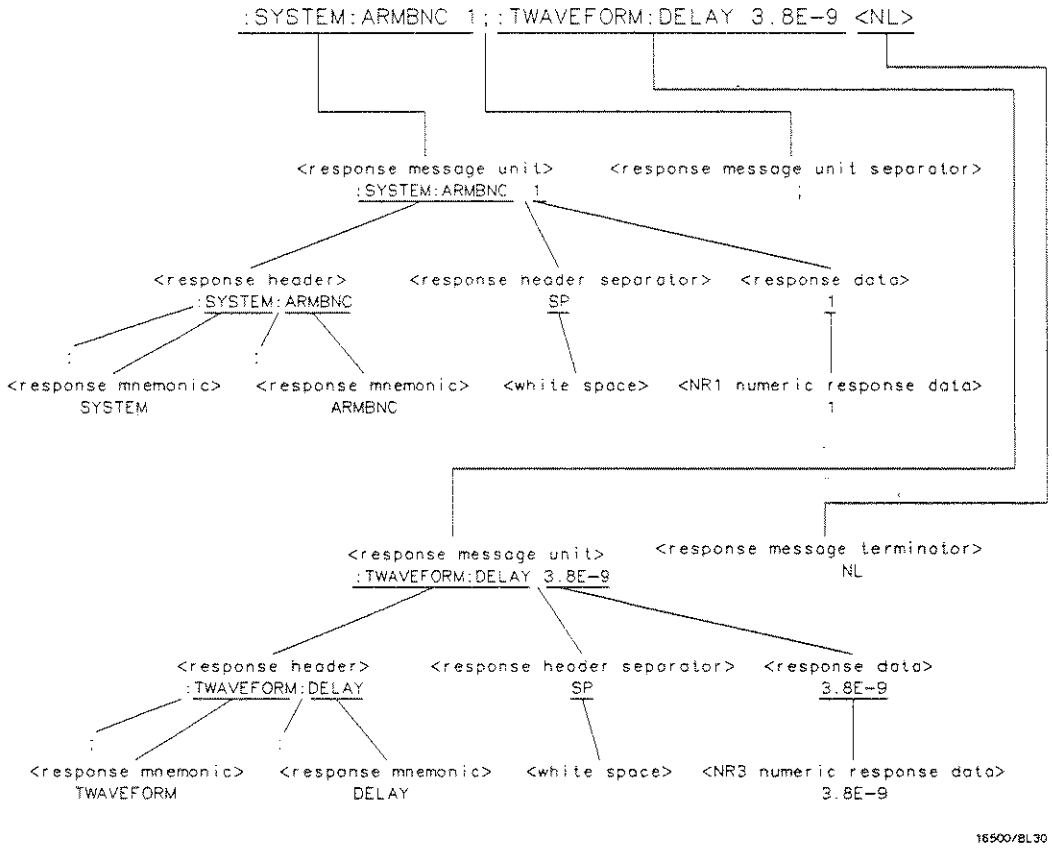


Figure A-19. <response message> Tree

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.

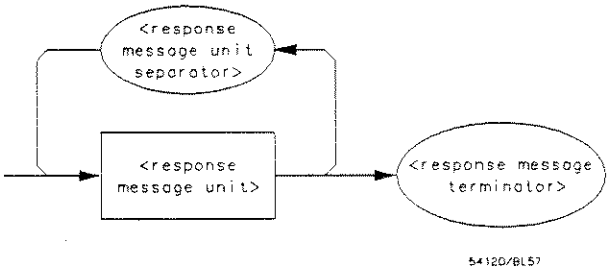
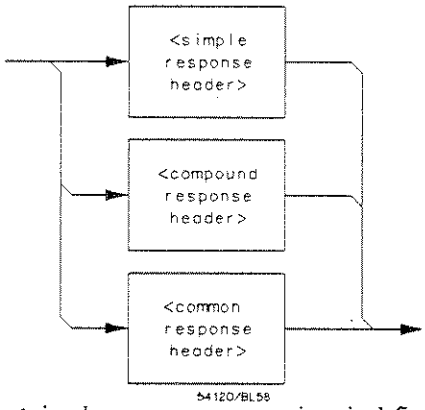


Figure A-20. < response message >

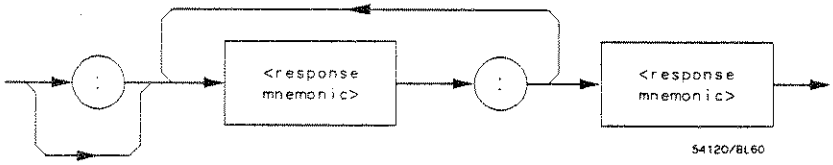
< response message unit >. This element serves as the container of individual pieces of a response. Typically a < query message unit > will generate one < response message unit >, although a < query message unit > may generate multiple < response message unit > s.

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



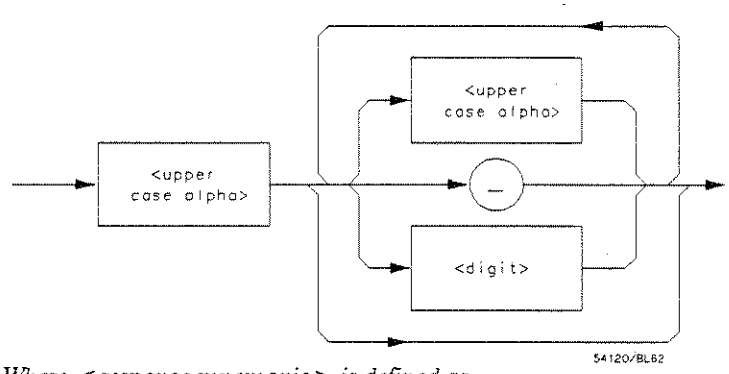
Where <simple response mnemonic> is defined as

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>.

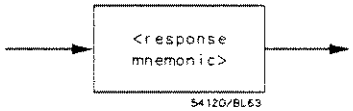


Figure A-22. <character response data>

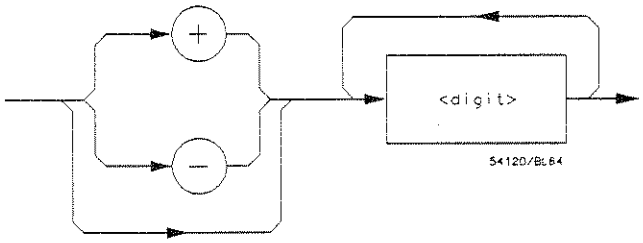


Figure A-23. <nr1 numeric response data>

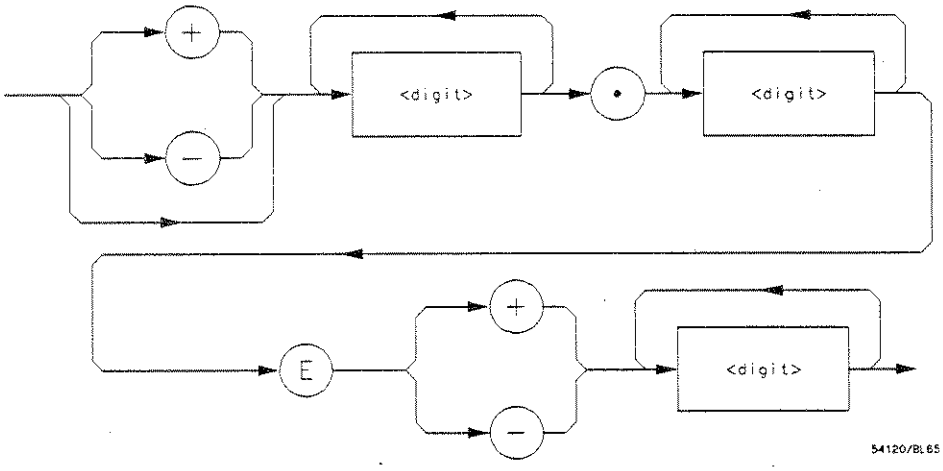


Figure A-24. <nr3 numeric response data>

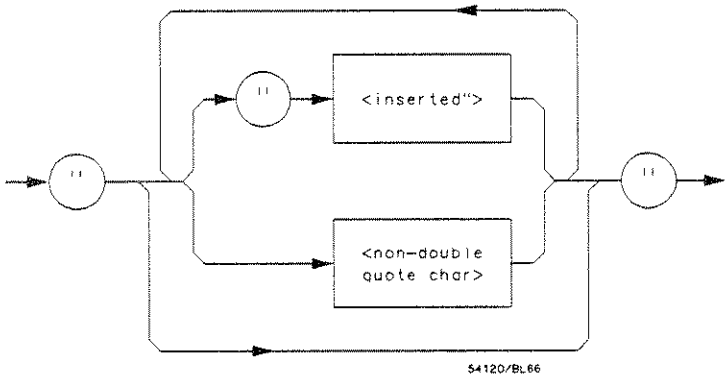


Figure A-25. <string response data>

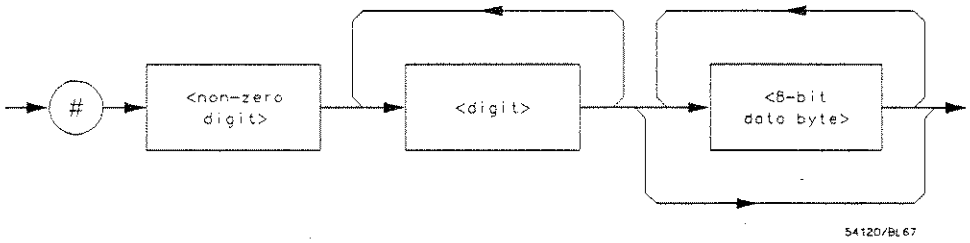
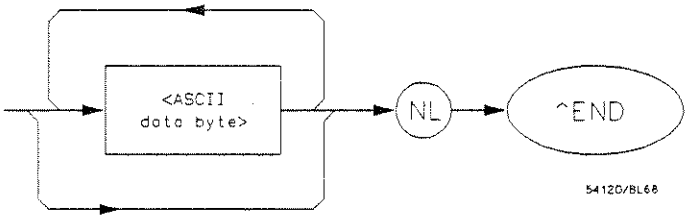


Figure A-26. <definite length arbitrary block response data>



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 1650A/51A.

Figure A-27. <arbitrary ASCII response data>

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

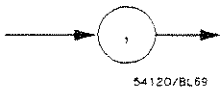


Figure A-28. < response data separator >

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

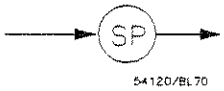


Figure A-29. < response header separator >

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

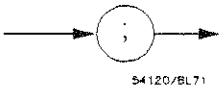


Figure A-30. < response message unit separator >

< 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.

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 1650A/51A'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
*OPC	Operation Complete Command
*OPC?	Operation Complete Query
*RST	Reset (not implemented on HP 1650A/51A)
*SRE	Service Request Enable Command
*SRE?	Service Request Enable Query
*STB?	Read Status Byte Query
*WAI	Wait-to-Continue Command

Status Reporting B

Introduction

The status reporting feature available over the bus is the serial poll. IEEE 488.2 defines data structures, commands, and common bit definitions. 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.

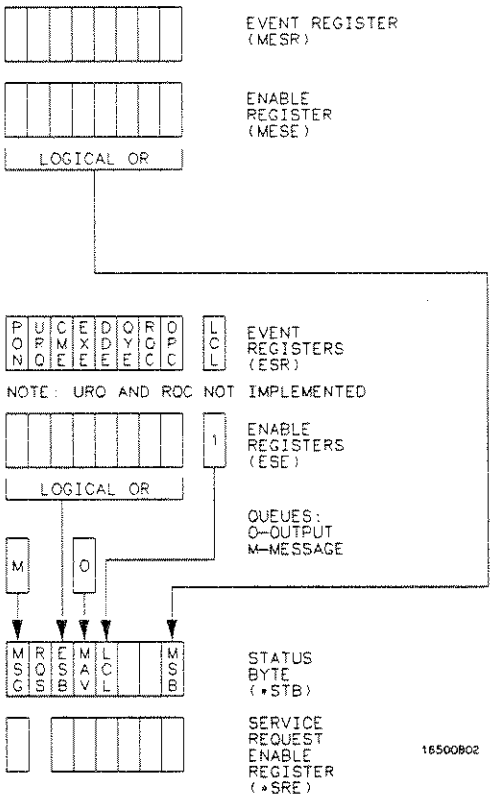


Figure B-1. Status Byte Structures and Concepts

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?).

MSG - message. Indicates whether there is a message in the message queue.

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

URQ - user request. Always 0 on the HP 1650A/51A.

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 1650A/51A.

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.

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

Error Messages C

This section covers the error messages that relate to the HP 1650A/51A Logic Analyzers.

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

E

F

G

H

I

L

M

N

O

P

Q

R

S

T

U

V

W

X