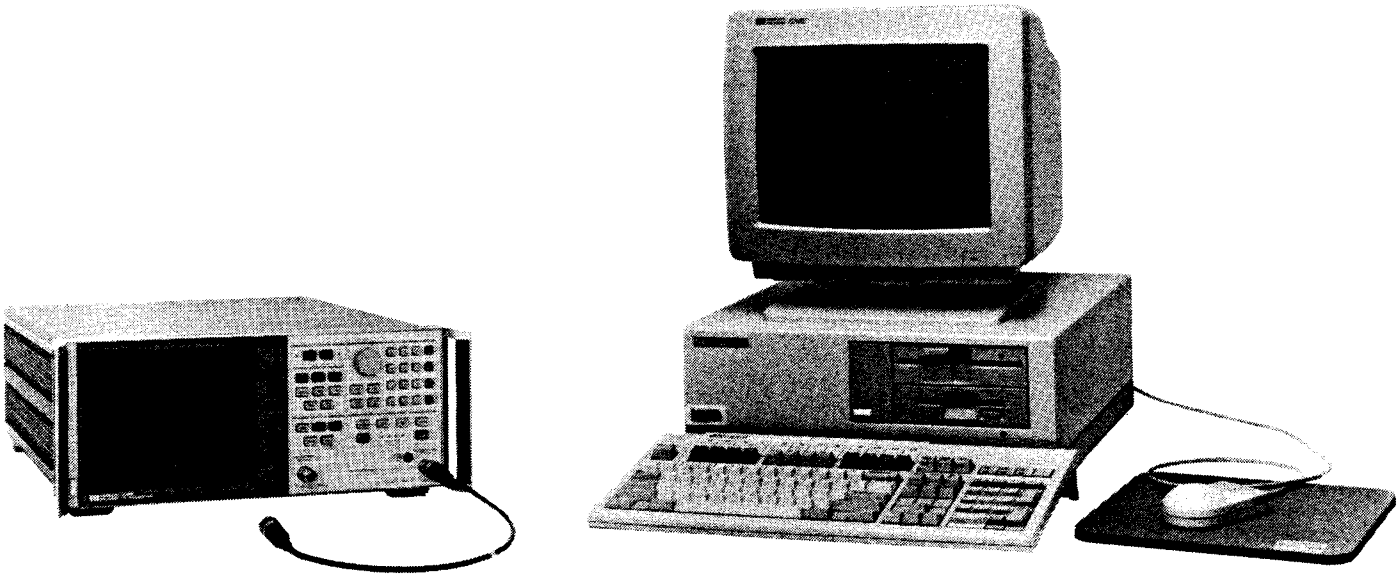
HP-IB Programming Guide

For the HP 8752A and HP 8753C Network Analyzers with the HP Vectra Personal Computer using Microsoft® QuickBASIC 4.5



Introduction

This programming guide is an introduction to remote operation of the HP 8752A and 8753C Network Analyzers with an HP Vectra Personal Computer (or IBM compatible) using the HP 82335A HP-IB Command Library and Microsoft QuickBASIC 4.5. This is a tutorial introduction, using programming examples to demonstrate the control of network analyzers with HP-IB commands. The example programs are on the Example Programs disk (part number 08753-10020) included with the operating manual. This document is closely associated with the *HP-IB Quick Reference* for the HP 8700-series network analyzers, which provides complete programming information in a concise format. Included in the *HP-IB Quick Reference* is an alphabetical list of HP-IB mnemonics and their explanations.

This note assumes that the reader is familiar with the operation of the network analyzer and the HP Vectra Personal Computer (or compatible), particularly HP-IB operation using the HP 82335A Command Library. This document is not intended to teach QuickBASIC programming or to discuss HP-IB theory except at an introductory level. See the section entitled *Reference information* for documents better suited to these tasks.

Reference information

HP 8752A/8753C Network Analyzer literature

User's Guide
Quick Reference
Operating Manual

HP-IB and HP Vectra Personal Computer literature

Tutorial Description of the Hewlett-Packard Interface Bus
Condensed Description of the Hewlett-Packard Interface Bus
HP 82335A HP-IB Command Library Manual

Microsoft QuickBASIC 4.5 literature

Microsoft QuickBASIC: BASIC Language Reference
Microsoft QuickBASIC: Learning and Using Microsoft QuickBASIC
Microsoft QuickBASIC: Programming in BASIC: Selected Topics

Microsoft® is a U.S. registered trademark of Microsoft Corp.

Table of contents

Equipment

To run the examples in this Programming Guide, the following equipment is required:

- HP 8752A or 8753C Network Analyzer.
- HP Vectra Personal Computer (or compatible) with Microsoft QuickBASIC 4.5, HP 82335A HP-IB Interface Card, MS-DOS® 3.2 or higher, and at least 320 Kbytes of memory.
- HP 10833A/B/C/D HP-IB cables to interconnect the computer, the network analyzer, and any peripherals.

The following equipment is optional:

- HP 85032B 50 ohm type-N calibration kit.
- HP 11857D 7 mm test port return cables (HP 8753C only).
- A test device such as a filter to use in the example measurement programs.

Preparation

1. **System.** Connect the network analyzer to the computer with an HP-IB cable. The network analyzer has only one HP-IB interface, but it occupies two addresses: one for the instrument and one for the display. The display address is the instrument address with the least significant bit complemented. The default addresses are 16 for the instrument and 17 for the display. Other devices on the bus cannot occupy the same address as the network analyzer.

2. **Computer.** Turn on the computer and load QuickBASIC by typing `QB /L QBHPIB` at the MS-DOS prompt. Invoking QuickBASIC in this way will load the Quick library `QBHPIB.QLB`, making its contents available for use.

3. **Network analyzer.** Turn on the network analyzer and verify its address by pressing **[LOCAL]** *[SET ADDRESSES]* and *[ADDRESS: 875x]*. If the address has been changed from the default value (16), return it to 16 to perform the examples in this document by pressing **[1] [6] [x1] [PRESET]**. Make sure the instrument is in *[TALKER/LISTENER]* mode, as indicated under the **[LOCAL]** key, since this is the only mode in which the network analyzer and an HP Vectra can communicate over HP-IB.

4. **Connection.** Type the following on the computer in the immediate portion of the display and all on one line:

```
CALL IOOUTPUTS(716&, "PRES;",
LEN("PRES;")): IF PCIB.ERR <> NOERR THEN
ERROR PCIB.BASERR
```

This presets the network analyzer. If a preset does not occur, there is a problem. Since many HP-IB problems are caused by an incorrect address or bad or loose HP-IB cables, check all HP-IB addresses and connections.

Notes on QuickBASIC

In QuickBASIC, multiple statements are allowed per line, and line numbers are not required. In the examples in this programming guide, line numbers are included for clarity. Each line is preceded by a line number, and each line number is followed by a complete one-line statement. No carriage returns are used in the statements although it may appear that way on the following pages.
The following error trapping line should follow every call to an I/O routine:

```
IF PCIB.ERR <> NOERR THEN ERROR PCIB.BASERR
```

In the following example programs, this line is generally made into a separate routine that can easily be executed after every call to an I/O routine:

```
CALL IOXXXX: GOSUB ERRORTRAP
```

If an error occurs, the number corresponding to that error is assigned to the variable `PCIB.ERR` and the program branches to an HP-IB Command Library subprogram for error handling which displays a message on the computer screen stating the error number and type.

Since the IOOUTPUTS command library routine to send a command from the computer to the analyzer is called so often and is so long, it is worthwhile to make it into a separate routine (called IOOUTS here) that can be executed with a GOSUB statement. If this is done, the line to preset the analyzer becomes

```
A$ = "PRES;": GOSUB IOOUTS
```

and the program END is followed by the ERRORTRAP and IOOUTS routines.

```
:
END

ERRORTRAP:
  IF PCIB.ERR <>NOERR THEN ERROR PCIB.BASERR
RETURN

IOOUTS:
  CALL IOOUTPUTS(716&, A$, LEN(A$)): GOSUB
  ERRORTRAP
RETURN
```

The construction of the IOOUTPUTS call is as follows:

```
CALL IOOUTPUTS(716&, A$, LEN(A$)): GOSUB
ERRORTRAP
```

CALL IOOUTPUTS: command. Execute the HP-IB string data output command.

716&: address. The data is directed to interface 7 (HP-IB) and out to the device at address 16 (the network analyzer). The appended "&" is required by the IO routine, which expects a long-integer.

A$: HP-IB command string. A$ should be set equal to the mnemonic corresponding to the desired operation before the GOSUB IOOUTS command that will execute the call to IOOUTPUTS is given.

LEN(A$): length. The IOOUTPUTS routine must know the length (in characters) of the command string it is sending so that it can append an appropriate line terminator.

GOSUB ERRORTRAP: error trap. The call to an error trapping routine that must follow every call to an I/O routine.

Just as there are I/O commands to send data to the analyzer, there are I/O commands to receive data from the analyzer. For more information on this topic, see the section entitled *Transferring Data.*

Basic Instrument Control

Preparation for HP-IB control

At the beginning of a program, the network analyzer has to be taken from an unknown state and brought under computer control. One way to do this is with an abort/clear sequence, which prepares the bus for activity and the analyzer for receiving HP-IB commands. In addition, a time-out should be set (IOTIMEOUT), and, if the program will be transferring data, the end-or-identify mode should be disabled (IOEOI). Because a known initial instrument state makes programs more reliable, the next step is generally to put the network analyzer into a known state. The most convenient way to do this is to send PRES, which returns the analyzer to the preset state. If preset is not desired and the status reporting mechanism is going to be used, CLES can be sent to clear all of the status reporting registers and their enabled bits.

For an example of the necessary preparation for HP-IB control in QuickBASIC programs, load the following program (stored on the Example Programs disk as **IPGI1.BAS)**. Note that the first four I/O commands are to the address 7&, the interface bus. Only the IOOUTPUTS command is actually to the analyzer, address 716&.

```
10   CALL IOTIMEOUT(7&, 10!): GOSUB ERRORTRAP
```

Define a system time-out of 10 seconds. (This value is chosen because most sweeps and calibration calculations are completed in under 10 seconds.) Time-out allows recovery from I/O operations that are not completed in the allowed number of seconds.

```
20   CALL IOABORT(7&): GOSUB ERRORTRAP
```

Halt any bus activity and return active control to the computer.

```
30   CALL IOCLEAR(7&): GOSUB ERRORTRAP
```

Clear syntax errors, the input command buffer, and any messages waiting to be sent out. This command does not affect the status reporting system.

```
40   CALL IOEOI(7&, 0): GOSUB ERRORTRAP
```

Disable the end-or-identify mode for transferring data. This prevents both a write operation from setting the EOI line on the last byte of the write and a read operation from terminating upon sensing that the EOI line has been set.

```
50   A$ = "PRES;": GOSUB IOOUTS
```

Send the HP-IB mnemonic PRES to the network analyzer (address = 716) via the IOOUTS subroutine. This presets the instrument, clears the status reporting system, and resets all front panel settings except the HP-IB mode and the HP-IB addresses.

```
60   END
```

End program execution.

```
70   ERRORTRAP:
80   IF PCIB.ERR <> NOERR THEN ERROR
     PCIB.BASERR
90   RETURN
100  IOOUTS:
110  CALL IOOUTPUTS(716&, A$, LEN(A$)):
     GOSUB ERRORTRAP
120  RETURN
```

This program brings the network analyzer to a known state and prepares it to respond to HP-IB control.
The network analyzer will not respond to HP-IB commands unless the remote line is asserted. When the remote line is asserted and the analyzer is addressed to listen, it automatically goes into remote mode. Remote mode means that all front panel keys except **[LOCAL]** and the line power switch are disabled. The command IOABORT asserts the remote line, which remains asserted until the command IOLOCAL is executed. Another way to assert the remote line is to execute

```
CALL IOREMOTE(716&): GOSUB ERRORTRAP
```

This statement asserts the remote line and addresses the network analyzer to listen, thereby putting it into remote mode. Now no front panel key will respond until **[LOCAL]** is pressed.

The local key can also be disabled with the following sequence:

```
CALL IOREMOTE(716&): GOSUB ERRORTRAP
CALL IOLLOCKOUT(7&): GOSUB ERRORTRAP
```

Now no front panel key (including **[LOCAL]**) except the line power switch will respond. The analyzer can be returned to local mode temporarily with the following command:

```
CALL IOLOCAL(716&): GOSUB ERRORTRAP
```

However, as soon as the analyzer is next addressed to listen, it goes back into local lockout. The only way to clear local lockout, other than cycling power, is to execute

```
CALL IOLOCAL(7&): GOSUB ERRORTRAP
```

This disables the remote line on the interface, puts the instrument into local mode, and clears local lockout.

Commands

A computer controls the network analyzer by sending it commands over HP-IB. Each command is specific to the network analyzer and is executed automatically, taking precedence over analyzer manual control. A command applies only to the active channel unless functions are coupled between channels, just as with front panel operation. Most commands are equivalent to front panel functions.

No operand commands

The simplest command that the network analyzer accepts is one that requires no operand. For example, AUTO is a no operand command. Leave the previous program in the main window and put the cursor in the immediate window. Now execute

```
A$ = "AUTO;": GOSUB IOOUTS
```

In response, the network analyzer autoscales the active channel just as it would if **[SCALE REF]** *[AUTO SCALE]* were pressed on the analyzer's front panel.
The semicolon following AUTO terminates the command inside the network analyzer. It clears the active entry area and prepares the network analyzer for the next command. If there is a syntax error in a command, the network analyzer will ignore the command and look for the terminating semicolon. When it finds this terminator, the network analyzer starts processing incoming commands normally. Characters between the syntax error and the next terminator are lost. A line feed can also act as terminator. The QuickBASIC IOOUTPUTS routine, which is called from the user-defined subprogram IOOUTS, automatically transmits a carriage return/line feed following the data if there is not a semicolon at the end of the statement.

The IOOUTPUTS routine will transmit all commands listed, as long as they are separated by commas or semicolons. All the information enclosed in quotes will be transmitted literally. A carriage return/line feed is transmitted after each command, but this can be prevented by separating commands with semicolons instead of commas.

The network analyzer does not distinguish between upper and lower case letters. For example, execute

```
A$ = "auto;": GOSUB IOOUTS
```

On/off commands

The network analyzer also accepts a command that turns a function on and off. Execute

```
A$ = "DUACON;": GOSUB IOOUTS
```

This activates dual channel display mode on the network analyzer. To restore single channel display mode, execute

```
A$ = "DUACOFF;": GOSUB IOOUTS
```

The command is composed of the root mnemonic DUAC (dual channel) and ON or OFF.

In addition, the network analyzer has a debug mode to aid in troubleshooting systems. When debug mode is on, the network analyzer scrolls incoming HP-IB commands across the display. To turn this mode on manually, press **[LOCAL]** *[HP-IB DIAG ON]*. To turn it on over HP-IB, execute

```
A$ = "DEBUON;": GOSUB IOOUTS
```

Parameter setting commands

The analyzer also accepts commands that set parameters. For example, execute

```
A$ = "STAR 10 MHZ;": GOSUB IOOUTS
```

The network analyzer now has a start frequency of 10 MHz. The STAR 10 MHZ command performs the same function as keying in **[START] [1] [0] [M/u]** from the network analyzer's front panel. STAR is the root mnemonic for the start key, 10 is the data, and MHZ is the units. The network analyzer's root mnemonics are derived from the equivalent key label if possible and from the common name for the function if not. The *HP-IB Quick Reference* lists all the root mnemonics and all the different units accepted.

Notice that the front panel remote (R) and listen (L) HP-IB status indicators are on. The network analyzer automatically goes into remote mode when it is sent a command with the IOOUTPUTS statement.

Interrogate instrument state commands

Each instrument parameter can be interrogated to find its current state or value with query commands. If a question mark is appended to the root mnemonic of a command, the network analyzer will send out the value of that parameter. For example, the command POWE 5 DB sets the analyzer's output power to +5 dBm, and the command POWE? tells the analyzer to send out the current RF output power value at the test port to the computer. The program in the main window can be modified to show the use of this command by deleting line 50 and inserting the following lines before the END at line 60.

```
45   A$ = "POWE?;": GOSUB IOOUTS
50   CALL IOENTER(716&, REPLY!): GOSUB
     ERRORTRAP
55   PRINT REPLY!
```

This modified program is stored on the Example Programs disk as **IPGI2.BAS**.

Now run the program, and the computer will display the source power level in dBm. The preset level is 0 dBm for the 8753C and −10 dBm for the 8752A. Next change the power level by pressing **[LOCAL] [MENU]** *[POWER]* **[1] [x1]**, and run the program again.

When the network analyzer receives the command POWE?, it prepares to send out the current RF source power level. The QuickBASIC statement CALL IOENTER(716&, REPLY!): GOSUB ERRORTRAP addresses the analyzer to talk, thereby allowing it to transmit information to the computer. This turns the network analyzer front panel talk light (T) on. The computer places the data transmitted by the network analyzer into the variable listed in the IOENTER statement. In this case, the network analyzer transmits the output power value, and this gets placed in the real number variable REPLY!.

The IOENTER statement takes the binary data sent out from the network analyzer and formats it into a real number. There are other I/O routines for entering a string (IOENTERS), an array of real numbers (IOENTERA), and unformatted data (IOENTERAB, IOENTERB). The data being requested is determined by the I/O routine and must correspond to the variable being received.

On/off commands can be also be interrogated. The reply is 1 if the function is on and 0 if it is off. Similarly, if a command controls a function that is underlined on the network analyzer display when active, interrogating that command yields 1 if the command is underlined and 0 if it is not. For example, there are nine options in the format menu, and only one is underlined at a time. Of the nine, only the underlined option will return 1 when interrogated.

For instance, rewrite line 45 as

```
45   A$ = "DUAC?;": GOSUB IOOUTS
```

Run the program once and note the result. Then press **[LOCAL] [DISPLAY]** *[DUAL CHAN]* to toggle the display mode, and run the program again to observe the difference.

Another example is to rewrite line 45 as

```
45   A$ = "PHAS?;": GOSUB IOOUTS
```

In this case, the computer will display 1, only if phase is currently being displayed on the network analyzer. Since the command only applies to the active channel, the response to the PHAS? inquiry depends on which channel is active.

Held commands

A held command is one that cannot be interrupted during its execution. When the network analyzer is executing a held command, it holds off processing new HP-IB commands, halting HP-IB operation until the held command completes execution. Some examples of held commands are DONE, PRES, and SING.

While a held command is executing, the network analyzer will still service the HP-IB interface routines, such as IOSPOLL, IOCLEAR, and IOABORT, all of which must be called and followed by error trapping. Executing a call to IOCLEAR will abort a held command, leaving its execution to be completed as if it had been begun from the front panel. These routines (IOSPOLL, IOCLEAR, and IOABORT) also clear the input buffer, destroying any commands received after the held command. If the network analyzer has halted the bus because its input buffer was full, executing a call to the routine IOABORT will release the bus.

Operation complete (OPC)

The operation complete (OPC) function allows synchronization of the program by requiring the current command to complete execution before the next command can begin. For instance, a program should not have the operator connect the next calibration standard while the network analyzer is still measuring the current one. To provide OPC information, the network analyzer uses its OPC reporting mechanism, which indicates when the execution of certain key commands has been completed. The function is activated by sending either OPC or OPC? immediately before an OPC'able command. When the command completes execution, bit 0 of the Event Status Register is set. If OPC? is interrogated, the network analyzer outputs 1 when the command completes execution.

The program in the main window can be modified to show the use of the OPC? command by deleting lines 45 through 55 and inserting the following lines before the END at line 60.

```
44   A$ = "SWET 3 S; OPC?; SING;": GOSUB
     IOOUTS
```

Set the sweep time to 3 seconds, and OPC? a single sweep.

```
48   PRINT "SWEEPING"
52   CALL IOENTER(716&,REPLY!): GOSUB
     ERRORTRAP
```

The program will halt until the network analyzer completes the sweep and sends out 1.

```
56   PRINT "DONE"
```

The modified program is stored on the Example Programs disk as **IPGI3.BAS**.

When it is run, the computer displays the sweeping message as the analyzer executes the sweep, and the computer displays DONE when the analyzer finishes the sweep. When DONE appears, the program can continue with a valid data trace ensured in the analyzer. Without a single sweep, it takes more than one sweep time to ensure good data.

Measurement Programming

The previous section of this document outlined the process to get commands into the network analyzer. The next step is to organize the commands into a measurement sequence. A typical measurement sequence consists of the following steps:

1. Prepare the instrument.
2. Calibrate the instrument.
3. Connect the device under test.
4. Make the measurement.
5. Process the data.
6. Transfer the data.

Prepare the instrument

Define the measurement by setting the basic measurement parameters. These include all the stimulus parameters (sweep type, span, sweep time, number of points, and RF power level) as well as the parameter to be measured, IF averaging, and IF bandwidth. These parameters define how data is gathered and processed within the instrument. Changing any parameter requires that a new sweep be taken.

Other parameters can be set within the instrument, such as smoothing, trace scaling, or trace math, that do not directly affect data gathering. These functions are classified as post processing functions: they can be changed with the instrument in hold mode, and the data will correctly reflect the new state.

The save/recall registers and the learn string are two rapid ways of setting up an entire instrument state. The learn string is a string summary of the instrument state that can be read into and sent out from the computer, as shown in Example 6A: *Using the learn string.*

Calibrate the instrument

Measurement calibration is normally performed once the instrument state has been defined. Although it is not required to make a measurement, calibration improves the accuracy of the data.

There are several ways to calibrate the instrument. The simplest way is to stop the program and have the operator perform the calibration from the front panel. Alternatively, the computer can be used to guide the operator through the calibration, as shown in Examples 2A: *1-port calibration* and 2B: *Full 2-port calibration (HP 8753C only).* Lastly, calibration data saved from a previous calibration can be transmitted back into the instrument, as shown in Example 6B: *Reading calibration data.* This should only be done if the hardware configuration has not changed.

Connect the device under test

The computer can be used to verify that the device is connected properly and to speed up the adjustment process. Useful functions for this purpose include limit testing, bandwidth searches, and trace statistics. All device adjustments should take place at this stage and be finished before taking data.

Make the measurement

Once the device is connected and adjusted, measure its frequency response and hold the data within the instrument so that there is a valid trace to analyze. The single sweep command SING is designed to do this. All stimulus changes are completed before the sweep is started, and the HP-IB hold state is not released until the formatted trace is displayed. When the sweep is complete, the instrument is put into hold mode, which freezes the data inside the instrument. Because single sweep is OPC'able, it is easy to determine when the sweep has been completed.

The number of groups command NUMGn is similar to SING, but it triggers n sweeps. This is useful, for example, in making a measurement with an averaging factor n (n can range from 1 to 999). Both SING and NUMGn commands restart averaging.

Process the data

With valid data to operate on, the post-processing functions can be used. Referring ahead to the data processing chain in Figure 1 (page 20), notice that any function that affects the data after the error correction stage can be used. The most useful functions are trace statistics, marker searches, electrical delay offset, time domain, and gating. If a 2-port calibration is active, then any of the four S-parameters can be viewed without taking a new sweep.

Transfer the data

Lastly, transmit the results out of the instrument. Each data output command is designed to ensure that transmitted data reflects the current state of the instrument.

- The commands OUTPDATA, OUTPRAWn, and OUTPFORM will transmit data only after all formatting functions have completed.
- The commands OUTPLIML, OUTPLIMM, and OUTPLIMF will transmit data only after a limit test has occurred (if limit testing is on).
- The command OUTPMARK will activate a marker (if one is not already selected) and will transmit data only after any current marker searches have completed.
- The command OUTPMSTA will transmit data only after marker statistics for the current trace have been calculated. If the statistics function is not on, it will be turned on to update the current values and then turned off.
- The command OUTPMWID will transmit data only after a bandwidth search has been executed for the current trace. If the bandwidth search function is not on, it will be turned on to update the current values and then turned off.

Data transfer is discussed further in Examples 3A through 3D: *Transferring data.*

Basic Programming Examples

Making measurements

The procedure for setting up measurements on the network analyzer via HP-IB follows the same sequence as when the setup is performed manually. As long as the desired frequency range, number of points, and power level are set prior to performing the calibration, there is no required order.

Example 1: Setting up a basic measurement

The following program illustrates how to set up a basic measurement on the network analyzer. The program will select the desired parameter, measurement format, and frequency range. Performing calibrations is described in later examples.

This example program is stored on the Example Programs disk as **IPG1.BAS**.

```
10  REM $INCLUDE: 'QBSETUP'
```

Call the QuickBASIC initialization file QBSETUP, the setup program for the MS-DOS HP-IB Command Library. This command should appear before the body of the program whenever calls to the HP-IB Command Library are to be made.

```
20  CLS
```

Clear the computer CRT.

```
30  ISC& = 7
```

Assign the interface select code to a variable. This select code is set on the HP 82335A HP-IB interface card.

```
40  VNA& = 716
```

Assign the address of the HP 8753C/8752A to a variable.

```
50  CALL IOTIMEOUT(ISC&, 10!):
    GOSUB ERRORTRAP
```

Define a system time-out of 10 seconds and perform error trapping. Time-out allows recovery from I/O operations that are not completed in under 10 seconds.

```
60  CALL IOABORT(ISC&): GOSUB
    ERRORTRAP
```

Abort any HP-IB transfers and perform error trapping.

```
70  CALL IOCLEAR(ISC&): GOSUB
    ERRORTRAP
```

Clear the analyzer's HP-IB interface and perform error trapping.

```
80  CALL IOEOI(ISC&, 0): GOSUB
    ERRORTRAP
```

Disable the End-Or-Identify mode for transferring data and perform error trapping.

```
90  A$ = "PRES; MENUOFF;":
    GOSUB IOOUTS
```

Preset the network analyzer and turn its softkey menu off.

```
100 A$ = "CHAN1; S21; LOGM;":
    GOSUB IOOUTS
```

Make channel 1 the active channel and measure the forward transmission parameter, displaying its magnitude in decibels. The mnemonic for this parameter is the same for both analyzers (S21) although it is called TRANSMISSION on the HP 8752A.

```
110 A$ = "CHAN2; S21; PHAS;":
    GOSUB IOOUTS
```

Make channel 2 the active channel and measure the phase of the forward transmission parameter.

```
120 A$ = "DUACON;": GOSUB
    IOOUTS
```

Display both channels simultaneously.

```
130 LOCATE 1, 1: INPUT "ENTER
    START FREQUENCY (MHz): ",
    F.START!
```

Position the cursor on the computer CRT at (row,column) = (1,1), and read in a real start frequency, F.START!.

```
140 LOCATE 1, 41: INPUT "ENTER
    STOP FREQUENCY (MHz): ",
    F.STOP!
```

Read in a real stop frequency, F.STOP!.

150 A$ = "STAR" + STR$(F.START!) + "MHz;": GOSUB IOOUTS	Set the start frequency on the network analyzer to F.START!. In QuickBASIC, the "+" is used to concatenate strings.
160 A$ = "STOP" + STR$(F.STOP!) + "MHz;": GOSUB IOOUTS	Set the stop frequency on the network analyzer to F.STOP!.
170 A$ = "AUTO;": GOSUB IOOUTS	Autoscale the network analyzer's active channel (2).
180 A$ = "CHAN1; AUTO;": GOSUB IOOUTS	Activate and autoscale channel 1.
190 A$ = "MENUON;": GOSUB IOOUTS	Turn the network analyzer's softkey menu back on.
200 CALL IOLOCAL(ISC&): GOSUB ERRORTRAP	Return the network analyzer to local mode and perform error trapping.
210 END	End program execution.
220 ERRORTRAP:	Define a routine to trap errors.
230 IF PCIB.ERR <> NOERR THEN ERROR PCIB.BASERR	Perform error trapping.
240 RETURN	Return from the ERRORTRAP routine.
250 IOOUTS:	Define a routine to send a command string from the computer to the analyzer.
260 CALL IOOUTPUTS(VNA&, A$, LEN(A$)): GOSUB ERRORTRAP	Send the command string A$ out to the analyzer and perform error trapping.
270 RETURN	Return from the IOOUTS routine.

Running the program

1. The computer sets up a measurement of transmission log magnitude on channel 1 and transmission phase on channel 2, displaying both measurements simultaneously by using the dual channel display mode.

2. Enter any valid value in MHz when prompted for start and stop frequencies.

3. The computer will enter the specified start and stop frequencies into the network analyzer, and they will be the frequency limits of the analyzer's display.

Performing calibrations

Coordinating a measurement calibration over HP-IB follows the keystrokes required to calibrate from the front panel in that there is a command for every step. The general key sequence is to select the calibration, to measure the calibration standards, and then to declare the calibration done. The actual sequence depends on the calibration kit and changes slightly for 2-port calibrations*, which are divided into three calibration sub-sequences.

The calibration kit tells the network analyzer which standards to expect at each step of the calibration. The set of standards associated with a given calibration is termed a class. For example, measuring the short during a 1-port calibration is one calibration step. All of the shorts that can be used for this calibration step make up the class, which is called class S11B. For the 7 mm and the 3.5 mm cal kits, class S11B has only one standard, so selecting *[SHORT]* automatically measures the short. For type-N cal kits, however, class S11B has two standards: male and female test ports. Selecting *[SHORTS]* brings up a second menu, allowing the operator to select which standard in the class is to be measured. The sex listed refers to the test port.

To do a 1-port calibration over HP-IB using the 7 mm or 3.5 mm cal kits, sending the command `CLASS11B` will automatically measure the short. For the type-N cal kit, sending `CLASS11B` brings up the menu with the male and female test port options. To select one of these standards, use either the command `STANA` or the command `STANB`. The `STAN` command can be appended with the letters `A` through `G`, corresponding to the standards listed under softkeys 1 through 7, softkey 1 being the uppermost softkey. The `STAN` command is always OPC'able, but a `CLASS` command is OPC'able only if the class has just one standard in it, which is then automatically measured. This is because when there is more than one standard in a class, the command that calls the class simply brings up another menu.

Each full 2-port measurement calibration is divided into three subsequences: transmission, reflection and isolation. Each subsequence is treated like a calibration in its own right: each must be opened, all of its standards must be measured, and then it must be declared done. The opening and closing commands for the subsequences are similar.

Transmission subsequence: `TRAN` and `TRAD`
Reflection subsequence: `REFL` and `REFD`
Isolation subsequence: `ISOL` and `ISOD`

*HP 8753C only.

Example 2A: 1-port calibration

The following program illustrates how to perform a 1-port measurement calibration on the network analyzer over HP-IB. The program does the calibration using the HP 85032B 50 ohm type-N calibration kit. It steps the operator through the calibration by giving explicit directions on the network analyzer display and allowing the user to continue the program from the network analyzer front panel. The desired instrument state should be set up before the program is run.

This example program is stored on the Example Programs disk as **IPG2A.BAS**.

`10 DECLARE SUB ERRORTRAP ()`	Define a subroutine to trap errors.
`20 DECLARE SUB IOOUTS (A$, ADDRESS&)`	Define a subroutine to send a command string from the computer to the analyzer.
`30 DECLARE SUB WAITFORKEY (LABEL$, VNA&, DISPLAY&, ISC&)`	Define a subroutine to display a message on the analyzer and wait for the operator to press a key.
`40 REM $INCLUDE: 'QBSETUP'`	Call the QuickBASIC initialization file `QBSETUP`.
`50 CLS`	Clear the computer CRT.
`60 ISC& = 7`	Assign the interface select code to a variable.
`70 VNA& = 716`	Assign the analyzer's address to a variable.
`80 DISPLAY& = 717`	Assign the analyzer's display address to a variable.
`90 CALL IOTIMEOUT(ISC&, 10!): CALL ERRORTRAP`	Define a system time-out of ten seconds and perform error trapping.
`100 CALL IOABORT(ISC&): CALL ERRORTRAP`	Abort any HP-IB transfers and perform error trapping.
`110 CALL IOCLEAR(ISC&): CALL ERRORTRAP`	Clear the analyzer's HP-IB interface and perform error trapping.
`120 CALL IOEOI(ISC&, 0): CALL ERRORTRAP`	Disable the End-Or-Identify mode for transferring data and perform error trapping.
`130 CALL IOOUTS("CALKN50; MENUOFF; CLES; ESE64;", VNA&)`	Select the 50 ohm type-N cal kit, turn off the softkey menu, clear the status byte, and set up the status reporting system so that bit 6, User Request, of the Event Status Register is summarized by bit 5 of the status byte, allowing a key press to be detected by a serial poll. For more information about setting up status reporting systems, refer to *Example 7: Interrupt generation.*
`140 CALL IOOUTS("WAIT;", VNA&)`	Wait for a clean sweep on the analyzer so that the following command will have the proper effect.
`150 CALL IOOUTS("ENTO;", VNA&)`	Clear the analyzer's entry area.
`160 CALL IOOUTS("CALIS111;", VNA&)`	Open the calibration by calling the S11 1-port calibration.
`170 CALL WAITFORKEY("CONNECT OPEN AT PORT 1", VNA&, DISPLAY&, ISC&)`	Ask for an open and wait for the operator to connect it.
`180 CALL IOOUTS("CLASS11A; OPC?; STANB;", VNA&)`	Measure the open. Identify the specific standard (female test port) within the class using the command `STANB`, indicating the option at the second softkey from the top.
`190 CALL IOENTER(VNA&, REPLY!): CALL ERRORTRAP`	Wait for the standard to be measured.

`200 CALL WAITFORKEY("CONNECT SHORT AT PORT 1", VNA&, DISPLAY&, ISC&)`	Ask for a short and wait for the operator to connect it.
`210 CALL IOOUTS("CLASS11B; OPC?; STANB;", VNA&)`	Measure the short. Identify the specific standard (female test port) within the class.
`220 CALL IOENTER(VNA&, REPLY!): CALL ERRORTRAP`	Wait for the standard to be measured.
`230 CALL WAITFORKEY("CONNECT LOAD AT PORT 1", VNA&, DISPLAY&, ISC&)`	Ask for a load and wait for the operator to connect it.
`240 CALL IOOUTS("OPC?; CLASS11C;", VNA&)`	Measure the load. There are no options within this class, so `OPC?`, which always precedes the last command, comes first.
`250 CALL IOENTER(VNA&, REPLY!): CALL ERRORTRAP`	Wait for the standard to be measured.
`260 CALL IOOUTS("PG;", DISPLAY&)`	Clear the user graphics by removing the last prompt.
`270 CLS : PRINT "COMPUTING CALIBRATION COEFFICIENTS"`	Display program progress on the computer CRT.
`280 CALL IOOUTS("DONE; OPC?; SAV1;", VNA&)`	Complete the calibration and save it.
`290 CALL IOENTER(VNA&, REPLY!): CALL ERRORTRAP`	Wait until the network analyzer has computed the calibration coefficients before continuing.
`300 CLS : PRINT "1-PORT CALIBRATION COMPLETED. CONNECT TEST DEVICE."`	Display program progress and instructions on the computer CRT.
`310 CALL IOOUTS("MENUON;", VNA&)`	Turn the softkey menu back on.
`320 CALL IOLOCAL(ISC&): CALL ERRORTRAP`	Return the analyzer to local mode and perform error trapping.
`330 END`	End program execution.
`340 SUB ERRORTRAP`	Define a subroutine to trap errors.
`350 IF PCIB.ERR <> NOERR THEN ERROR PCIB.BASERR`	Perform error trapping.
`360 END SUB`	Return from the `ERRORTRAP` subroutine.
`370 SUB IOOUTS (A$, ADDRESS&) STATIC`	Define a subroutine to send a command string from the computer to the analyzer.
`380 CALL IOOUTPUTS(ADDRESS&, A$, LEN(A$)): CALL ERRORTRAP`	Send the command string `A$` out to the analyzer and perform error trapping.
`390 END SUB`	Return from the `IOOUTS` subroutine.
`400 SUB WAITFORKEY (LABEL$, VNA&, DISPLAY&, ISC&) STATIC`	Define a subroutine to display a message on the analyzer and wait for the operator to press a key.
`410 CLS : PRINT LABEL$`	Display instructions on the computer CRT.

```
420 CALL IOOUTS("PG; PU; PA
    390,3600; PD; LB" +
    LABEL$ + "; PRESS ANY KEY
    WHEN READY." + CHR$(3),
    DISPLAY&)
```

Write on the network analyzer's display:
PG : PaGe; clears old user graphics.
PU : Pen Up; prevents anything from being drawn.
PA : Pen At; positions the logical pen.
PD : Pen Down; enables drawing.
LB : LaBel; writes the message on the display. The label must always be terminated by the ETX symbol, CHR$(3).

```
430 CALL IOOUTS("ESR?;",
    VNA&)
```

Request the Event Status Register value from the analyzer.

```
440 CALL IOENTER(VNA&,
    ESTAT!): CALL ERRORTRAP
```

Receive the Event Status Register value from the analyzer, thereby clearing the latched User Request bit so that old key presses will not trigger a measurement.

```
450 CALL IOOUTS("ESE64;",
    VNA&)
```

Ensure that the proper status reporting system is still in effect.

```
460 STAT% = 0
```

Initialize STAT% for entry into the DO UNTIL loop.

```
470 DO UNTIL ((STAT% MOD 64)
    >31)
```

Wait for a key press to be indicated by the setting of bit 5 of the status byte. MOD 64 removes the effect of all higher value bits (bit 6 is equivalent to 64 in decimal), and >31 ensures that bit 5, which is equivalent to 32 in decimal, is set.

```
480  CALL IOSPOLL(VNA&, STAT%):
     CALL ERRORTRAP
```

Read in the status byte as an integer.

```
490 LOOP
```

```
500 END SUB
```

Return from the WAITFORKEY subroutine.

Running the program

1. The computer assumes that the port being calibrated is a 50 ohm type-N female test port and prompts the operator to connect each standard.
2. Connect the standards as prompted, and press any key on the front panel of the network analyzer to continue the program and measure the standard.
3. The program will display a message when the measurement calibration is complete.

Example 2B: Full 2-port calibration (HP 8753C only)

The following program illustrates how to perform a full 2-port measurement calibration on the network analyzer over HP-IB. The program does the calibration using the HP 85032B calibration kit. It steps the operator through the calibration by giving explicit directions on the network analyzer display and allowing the user to continue the program from the network analyzer front panel. The desired instrument state should be set up before the program is run. The main difference between this example and Example 2A is that in this case the calibration process allows removal of both the forward and reverse error terms. This permits measurement of all four S-parameters of the device under test. Port 1 is a female test port and port 2 is a male test port.

This example program is stored on the Example Programs disk as **IPG2B.BAS**.

10 DECLARE SUB ERRORTRAP ()	Define a subroutine to trap errors.
20 DECLARE SUB IOOUTS (A$, ADDRESS&)	Define a subroutine to send a command string from the computer to the analyzer.
30 DECLARE SUB WAITFORKEY (LABEL$, VNA&, DISPLAY&, ISC&)	Define a subroutine to display a message on the analyzer and wait for the operator to press a key.
40 REM $INCLUDE: 'QBSETUP'	Call the QuickBASIC initialization file QBSETUP.
50 CLS	Clear the computer CRT.
60 ISC& = 7	Assign the interface select code to a variable.
70 VNA& = 716	Assign the analyzer's address to a variable.
80 DISPLAY& = 717	Assign the analyzer's display address to a variable.
90 CALL IOTIMEOUT(ISC&, 10!): CALL ERRORTRAP	Define a system time-out of ten seconds and perform error trapping.
100 CALL IOABORT(ISC&): CALL ERRORTRAP	Abort any HP-IB transfers and perform error trapping.
110 CALL IOCLEAR(ISC&): CALL ERRORTRAP	Clear the analyzer's HP-IB interface and perform error trapping.
120 CALL IOEOI(ISC&, 0): CALL ERRORTRAP	Disable the End-Or-Identify mode for transferring data and perform error trapping.
130 CALL IOOUTS("CALKN50; MENUOFF; CLES; ESE64;", VNA&)	Select the 50 ohm type-N cal kit, turn off the softkey menu, clear the status byte, and set up the status reporting system so that bit 6, User Request, of the Event Status Register is summarized by bit 5 of the status byte, allowing a key press to be detected by a serial poll.
140 CALL IOOUTS("CALIFUL2;", VNA&)	Open the calibration by calling for a full two-port calibration.
150 CALL IOOUTS("REFL;", VNA&)	Open the reflection calibration subsequence.
160 CALL WAITFORKEY("CONNECT OPEN AT PORT 1", VNA&, DISPLAY&, ISC&)	Ask for an open at port 1 and wait for the operator to connect it.
170 CALL IOOUTS("CLASS11A; OPC?; STANB;", VNA&)	Measure the open. Identify the specific standard (female test port) within the class using the command STANB, indicating the option at the second softkey from the top.
180 CALL IOENTER(VNA&, REPLY!): CALL ERRORTRAP	Wait for the standard to be measured.
190 CALL WAITFORKEY("CONNECT SHORT AT PORT 1", VNA&, DISPLAY&, ISC&)	Ask for a short at port 1 and wait for the operator to connect it.

```
200 CALL IOOUTS("CLASS11B;
    OPC?; STANB;", VNA&)
```

Measure the short. Identify the specific standard (female test port) within the class.

```
210 CALL IOENTER(VNA&,
    REPLY!): CALL ERRORTRAP
```

Wait for the standard to be measured.

```
220 CALL WAITFORKEY("CONNECT
    LOAD AT PORT 1", VNA&,
    DISPLAY&, ISC&)
```

Ask for a load at port 1 and wait for the operator to connect it.

```
230 CALL IOOUTS("OPC?;
    CLASS11C;", VNA&)
```

Measure the load. There are no options within this class, so OPC?, which always precedes the last command, comes first.

```
240 CALL IOENTER(VNA&,
    REPLY!): CALL ERRORTRAP
```

Wait for the standard to be measured.

```
250 CALL WAITFORKEY("CONNECT
    OPEN AT PORT 2", VNA&,
    DISPLAY&, ISC&)
```

Ask for an open at port 2 and wait for the operator to connect it.

```
260 CALL IOOUTS("CLASS22A;
    OPC?; STANA;", VNA&)
```

Measure the open. Identify the specific standard (male test port) within the class using the command STANA, indicating the option at the first softkey from the top.

```
270 CALL IOENTER(VNA&,
    REPLY!): CALL ERRORTRAP
```

Wait for the standard to be measured.

```
280 CALL WAITFORKEY("CONNECT
    SHORT AT PORT 2", VNA&,
    DISPLAY&, ISC&)
```

Ask for a short at port 2 and wait for the operator to connect it.

```
290 CALL IOOUTS("CLASS22B;
    OPC?; STANA;", VNA&)
```

Measure the short. Identify the specific standard (male test port) within the class.

```
300 CALL IOENTER(VNA&,
    REPLY!): CALL ERRORTRAP
```

Wait for the standard to be measured.

```
310 CALL WAITFORKEY("CONNECT
    LOAD AT PORT 2", VNA&,
    DISPLAY&, ISC&)
```

Ask for a load at port 2 and wait for the operator to connect it.

```
320 CALL IOOUTS("OPC?;
    CLASS22C;", VNA&)
```

Measure the load, noting that there are no options within this class.

```
330 CALL IOENTER(VNA&,
    REPLY!): CALL ERRORTRAP
```

Wait for the standard to be measured.

```
340 CALL IOOUTS("OPC?;
    REFD;", VNA&)
```

Close the reflection calibration subsequence.

```
350 CLS : PRINT "COMPUTING
    REFLECTION CALIBRATION
    COEFFICIENTS"
```

Display program progress on the computer CRT.

```
360 CALL IOENTER(VNA&,
    REPLY!): CALL ERRORTRAP
```

Wait for the analyzer to finish calculating the reflection calibration coefficients before continuing.

```
370 CALL IOOUTS("TRAN;",
    VNA&)
```

Open the transmission calibration subsequence.

```
380 CLS : PRINT "OPENING
    TRANSMISSION CALIBRATION
    SUBSEQUENCE"
```

Display program progress on the computer CRT.

```
390 CALL WAITFORKEY("CONNECT
    THRU (PORT 1 TO PORT 2)",
    VNA&, DISPLAY&, ISC&)
```

Ask for a thru and wait for the operator to connect it.

```
400 CLS : PRINT "MEASURING
    FORWARD TRANSMISSION"
```

Display program progress on the computer CRT.

410 CALL IOOUTS("OPC?; FWDT;", VNA&)	Measure forward transmission.
420 CALL IOENTER(VNA&, REPLY!): CALL ERRORTRAP	Wait for the standard to be measured.
430 CALL IOOUTS("OPC?; FWDM;", VNA&)	Measure forward load match.
440 CALL IOENTER(VNA&, REPLY!): CALL ERRORTRAP	Wait for the standard to be measured.
450 CLS : PRINT "MEASURING REVERSE TRANSMISSION"	Display program progress on the computer CRT.
460 CALL IOOUTS("OPC?; REVT;", VNA&)	Measure reverse transmission.
470 CALL IOENTER(VNA&, REPLY!): CALL ERRORTRAP	Wait for the standard to be measured.
480 CALL IOOUTS("OPC?; REVM;", VNA&)	Measure reverse load match.
490 CALL IOENTER(VNA&, Reply!): CALL ERRORTRAP	Wait for the standard to be measured.
500 CALL IOOUTS("TRAD;", VNA&)	Close the transmission calibration subsequence.
510 CLS : INPUT "SKIP ISOLATION CALIBRATION? (Y/N) ", ANSWER$	Ask the operator if the isolation part of the calibration is to be skipped.
520 IF ((ANSWER$ = "Y") OR (ANSWER$ = "y")) THEN	Skip the isolation part of the calibration.
530 CALL IOOUTS("OMII;", VNA&)	Tell the analyzer to omit the isolation part of the calibration.
540 ELSE	Do the isolation part of the calibration.
550 CALL WAITFORKEY("ISOLATE TEST PORTS", VNA&, DISPLAY&, ISC&)	Ask the operator to isolate the test ports.
560 CALL IOOUTS("ISOL; AVERFACT10; AVERON;", VNA&)	Open the isolation calibration subsequence. Turn averaging on with an averaging factor of ten.
570 CLS : PRINT "MEASURING REVERSE ISOLATION"	Display program progress on the computer CRT.
580 CALL IOOUTS("OPC?; REVI;", VNA&)	Measure reverse isolation.
590 CALL IOENTER(VNA&, REPLY!): CALL ERRORTRAP	Wait for the standard to be measured.
600 CLS : PRINT "MEASURING FORWARD ISOLATION"	Display program progress on the computer CRT.
610 CALL IOOUTS("OPC?; FWDI;", VNA&)	Measure forward isolation.
620 CALL IOENTER(VNA&, REPLY!): CALL ERRORTRAP	Wait for the standard to be measured.
630 END IF	
640 CALL IOOUTS("ISOD; AVEROFF;", VNA&)	Close the isolation calibration subsequence. Turn off averaging.
650 CALL IOOUTS("PG;", DISPLAY&)	Ensure that the user graphics are cleared by removing the last prompt.
660 CLS : PRINT "COMPUTING CALIBRATION COEFFICIENTS"	Display program progress on the computer CRT.

670 CALL IOOUTS("DONE; OPC?; SAV2;", VNA&)	Affirm the completion of the calibration and save it.
680 CALL IOENTER(VNA&, REPLY!): CALL ERRORTRAP	Wait until the network analyzer has computed the calibration coefficients before continuing.
690 CLS : PRINT "FULL 2-PORT CALIBRATION COMPLETED. CONNECT TEST DEVICE.";	Display program progress and instructions on the computer CRT.
700 CALL IOOUTS("MENUON;", VNA&)	Turn the softkey menu back on.
710 CALL IOLOCAL(ISC&): CALL ERRORTRAP	Return the analyzer to local mode and perform error trapping.
720 END	End program execution.
730 SUB ERRORTRAP	Define a subroutine to trap errors.
740 IF PCIB.ERR <> NOERR THEN ERROR PCIB.BASERR	Perform error trapping.
750 END SUB	Return from the ERRORTRAP subroutine.
760 SUB IOOUTS (A$, ADDRESS&) STATIC	Define a subroutine to send a command string from the computer to the analyzer.
770 CALL IOOUTPUTS(ADDRESS&, A$, LEN(A$)): CALL ERRORTRAP	Send the command string A$ out to the analyzer and perform error trapping.
780 END SUB	Return from the IOOUTS subroutine.
790 SUB WAITFORKEY (LABEL$, VNA&, DISPLAY&, ISC&) STATIC	Define a subroutine to display a message on the analyzer and wait for the operator to press a key.
800 CLS : PRINT LABEL$	Display instructions on the computer CRT.
810 CALL IOOUTS("PG; PU; PA 390,3600; PD; LB" + LABEL$ + "; PRESS ANY KEY WHEN READY." + CHR$(3), DISPLAY&)	Write on the network analyzer's display: PG : PaGe; clears old user graphics. PU : Pen Up; prevents anything from being drawn. PA : Pen At; positions the logical pen. PD : Pen Down; enables drawing. LB : LaBel; writes the message on the display. The label must always be terminated by the ETX symbol, CHR$(3).
820 CALL IOOUTS("ENTO;", VNA&)	Clear the analyzer's entry area.
830 CALL IOCLEAR(VNA&): CALL ERRORTRAP	Clear the analyzer's HP-IB interface and perform error trapping.
840 CALL IOOUTS("ESR?;", VNA&)	Request the Event Status Register value from the analyzer.
850 CALL IOENTER(VNA&, ESTAT!): CALL ERRORTRAP	Receive the Event Status Register from the analyzer, thereby clearing the latched User Request bit so that old key presses will not trigger a measurement.
860 CALL IOOUTS("ESE64;", VNA&)	Ensure that the proper status reporting system is still in effect.
870 STAT% = 0	Initialize STAT% for entry into the DO UNTIL loop.
880 DO UNTIL ((STAT% MOD 64) >31)	Wait for a key press to be indicated by the setting of bit 5 of the status byte. MOD 64 removes the effect of all higher value bits (bit 6 is equivalent to 64 in decimal), and >31 ensures that bit 5, which is equivalent to 32 in decimal, is set.

```
890  CALL IOSPOLL(VNA&, STAT%):    Read in the status byte as an integer.
     CALL ERRORTRAP

900 LOOP

910 CALL IOOUTS("PG;",             Clear the user graphics on the analyzer.
    DISPLAY&)

920 END SUB                        Return from the WAITFORKEY subroutine.
```

Running the program

1. The computer assumes that the test ports being calibrated are 50 ohm type-N, port 1 being a female test port and port 2 being a male test port. Prompts to connect each standard appear just above the message line on the HP 8753C display.

2. Connect the standards as prompted, and press any key on the front panel of the network analyzer to continue the program and measure the standard. When the option of omitting the isolation calibration is given, press "Y" or "N" on the computer keyboard. If the isolation cal is performed, averaging is automatically employed to ensure a good calibration.

3. The program will display a message when the measurement calibration is complete.

Transferring data

Trace information can be read out of the analyzer in two ways. First, trace data can be read selectively using markers. This is preferable if only specific information is needed. Secondly, the entire trace can be read out. This is only necessary if all the trace data is needed. The process of transferring data can be divided into the following three steps:

1. Set up the receiving array. Trace data is represented inside the network analyzer as a real/imaginary component pair for each point. The receiving array for marker data must store three values: this real/imaginary component pair as well as a stimulus value. See Table 1 to identify the first two values according to the current display format and marker mode. The receiving array for reading in an entire trace must be two components wide and the number of points long in order to accommodate all of the trace data. Since QuickBASIC stores data by column and therefore fills the first array dimension first, make the first dimension of the receiving array correspond to the number of elements per point (e.g. 2) and the second dimension correspond to the number of points (e.g. 201). In addition, because a four-byte header is sent out before the trace data when reading in an entire trace in all formats except *form 4*, at least one extra real number or two extra integers must be allocated at the beginning of the receiving array in order to maintain data order. Although this four-byte header can be read in as one real number or as two integers, the four bytes are actually meant to be two ASCII characters and one integer. The first two bytes are the ASCII characters "#A" that indicate that a fixed length block transfer follows. The last two bytes form an integer containing the number of bytes in the block to follow.

2. Request the data from the network analyzer. For marker data, this is always done by the command `OUTPMARK`. For an entire trace, the desired data format and level must be specified. The analyzer can transmit data over HP-IB in five different formats, three of which are shown in the following example programs. The level of the data is determined by the `OUTPxxxx` command used. (See Figure 1.) The different data levels are as follows:

   - Raw data is the basic measurement data. It reflects the stimulus parameters, IF averaging, and IF bandwidth, and is read out with the four `OUTPRAWx` commands. Normally, only `OUTPRAW1` is available, and it sends out the current active parameter; however, if a full 2-port measurement calibration is on, all four `OUTPRAWx` commands are available. The four arrays correspond to S11, S21, S12, and S22, respectively, and the data is in real/imaginary component pairs.

   - Error-corrected data is the raw data with error correction applied. This data is read out with the command `OUTPDATA`, which reads active trace data, or the command `OUTPMEMO`, which reads the error corrected trace memory, if available. The data is for the current active parameter and is in real/imaginary component pairs. Neither raw nor error-corrected data reflect such post-processing functions as electrical delay offset, trace math, or time domain gating.

   - Formatted data, read out by the command `OUTPFORM`, is the data being displayed by the analyzer and reflects all post-processing functions. See Table 1 to identify the array values according to the current display format and marker mode.

   - Calibration coefficient data is the error correction arrays resulting from a calibration. Each array corresponds to a specific error term in the error model, and the data is stored as real/imaginary component pairs. The HP-IB Quick Reference details which error coefficients are used for specific calibration types and which arrays those coefficients are to be found in. Not all calibration types use all twelve arrays.

Because formatted data is seen on the analyzer display, it is generally the most useful. However, if post-processing is not necessary, as may be the case with smoothing, error-corrected data is more desirable.

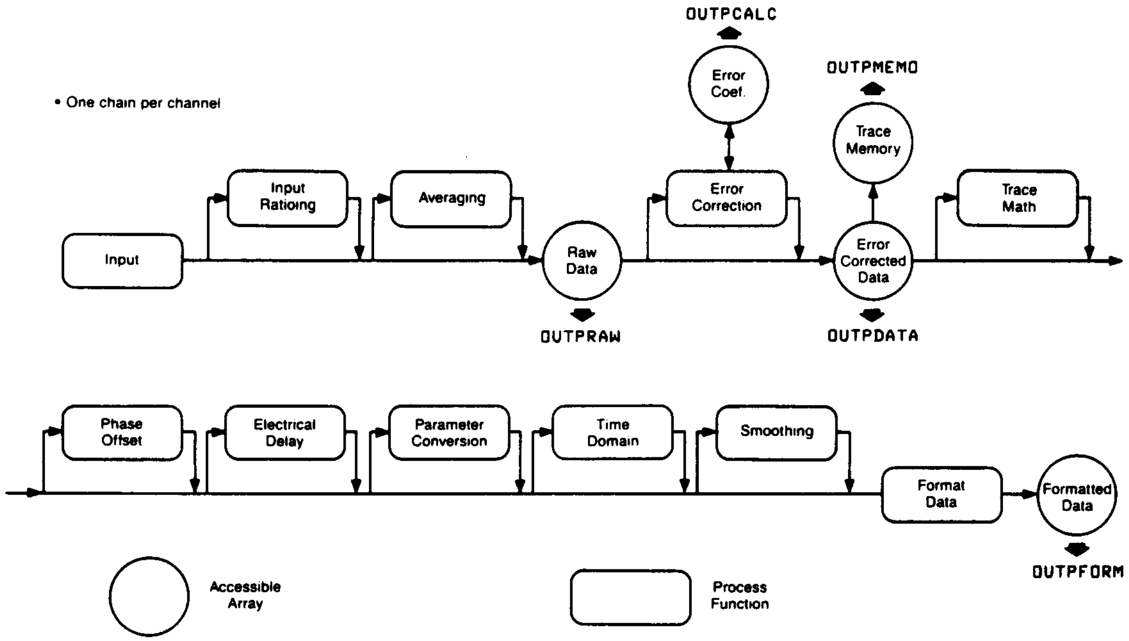


Figure 1. Data processing chain

3. Set all receiving parameters, and receive the data into the array. The receiving parameters and the type of data read in depend on which I/O routine will be used to receive the array. The three parameters in the computer that it may be necessary to initialize are as follows:

   - `MAX%`: the maximum number of items to be read. This includes the data and the header for all data formats except *form 4*. See Table 2 to determine whether `MAX%` is to specify a number of real numbers or a number of bytes according to the entering I/O routine used.

   - `ACTUAL%`: the actual number of items read. This is set by the I/O routine and should be initialized to zero.

   - `FLAG%`: the code set to indicate how transferred bytes are to be placed into memory. For example, `FLAG% = 1` means that bytes will be put into consecutive memory locations; `FLAG% = 4` means that every four bytes will be reversed in memory. See Table 2 to identify the entering I/O routines that use `FLAG%` as a parameter.

In general, the entering I/O routine must be sent a segment address indicating the place in memory to start storing data. If there is a four-byte header to be read in, this address should be one real number or two integers (four bytes) before the desired destination of the true data. For example, an array to hold the data for a 201-point trace with two real numbers per point might be allocated as `DAT!(1 TO 2, 1 TO 201)`. In order to account for the header, it should instead be dimensioned as `DAT!(1 TO 2, 0 TO 201)`, which will add two real numbers to the beginning of the array. Since only one of these is needed to store the four-byte header, the starting address specified in the entering I/O routine should only include one of them in the array: `SEG DAT!(2, 0)`. The result of this is that `DAT!(1, 0)` will be empty, `DAT!(2, 0)` will store the header, and `DAT!(1, 1)` will store the first real number of the data. See Table 2 for a summary of all entering I/O routines. For more information, refer to the *HP-IB Command Library Manual.*

Table 1. Units as a Function of Display Format

DISPLAY FORMAT	MARKER MODE	OUTPMARK value 1, value 2	OUTPFORM value 1, value 2	MARKET READOUT** value, aux value
LOG MAG		dB,*	dB,*	dB,*
PHASE		degrees,*	degrees,*	degrees,*
DELAY		seconds,*	seconds,*	seconds,*
SMITH	LIN MKR	lin mag, degrees	real, imag	lin mag, degrees
CHART	LOG MKR	dB, degrees	"	dB, degrees
	Re/Im	real, imag	"	real, imag
	R + jX	real, imag ohms	"	real, imag ohms
	G + jB	real, imag Siemens	"	real, imag Siemens
POLAR	LIN MKR	lin mag, degrees	real, imag	lin mag, degrees
	LOG MKR	dB, degrees	"	dB, degrees
	Re/Im	real, imag	"	real, imag
LIN MAG		lin mag,*	lin mag,*	lin mag,*
REAL		real,*	real,*	real,*
SWR		SWR,*	SWR,*	SWR,*

* Value not significant in this format, but is included in data transfers.

** The marker readout values are the marker values displayed in the upper right-hand corner of the display. They also correspond to the value and aux value associated with the fixed marker.

Table 2. Entering IO Routine Summary

ROUTINE	DATA TYPE	MAX%	FLAG %
IOENTER	one real	—	no
IOENTERA	array of reals	number of reals	no
IOENTERAB	unformatted	number of bytes*	yes
IOENTERB	unformatted	number of bytes	yes
IONETERS	character string	number of characters	no

* IOENTERAB will only read out as many bytes as are indicated by the last two bytes of the header (the number of bytes in the block to follow). However, if MAX% is less than this number, the transfer will terminate once MAX% bytes have been read out (MAX% is used as a safeguard to prevent longer-than-anticipated data from over-running the data array).

Example 3A: Data transfer using form 4, ASCII transfer format

The following program illustrates how to transfer data using *form 4*. *Form 4* transfers two numbers for each trace point, each number of the transfer data as a 24-character string, each character being a digit, sign, or decimal point. *Form 4* does not use a header. The first of two eleven-point transfers uses OUTPFORM to read out magnitude data. This eleven-point transfer with two real numbers per point and 24 bytes per point takes 528 (11*2*24) bytes. The second transfer uses OUTPLIML to read out limit data. (OUTPLIML reads out the stimulus frequency, result, upper limit, and lower limit of limit data.) Note that stimulus values can be read using this command even though no limits have been set. This eleven-point transfer with four real numbers per point and 24 bytes per point takes 1056 (11*4*24) bytes.

This example program is stored on the Example Programs disk as **IPG3A.BAS**.

`10 REM $INCLUDE: 'QBSETUP'`	Call the QuickBASIC initialization file QBSETUP.
`20 CLS`	Clear the computer CRT.
`30 ISC& = 7`	Assign the interface select code to a variable.
`40 VNA& = 716`	Assign the analyzer's address to a variable.
`50 CONST SIZE% = 11`	Set a constant to the number of points to be used in the trace.
`60 CALL IOTIMEOUT(ISC&, 10!): GOSUB ERRORTRAP`	Define a system time-out of ten seconds and perform error trapping.
`70 CALL IOABORT(ISC&): GOSUB ERRORTRAP`	Abort any HP-IB transfers and perform error trapping.
`80 CALL IOCLEAR(ISC&): GOSUB ERRORTRAP`	Clear the analyzer's HP-IB interface and perform error trapping.
`90 CALL IOEOI(ISC&, 0): GOSUB ERRORTRAP`	Disable the End-Or-Identify mode for transferring data and perform error trapping.
`100 A$ = "PRES;": GOSUB IOOUTS`	Preset the network analyzer.
`110 DIM DAT!(1 TO 2, 1 TO SIZE%), STIM!(1 TO 4, 1 TO SIZE%)`	Prepare arrays to receive the data. All IOENTER routines that fill arrays do so column by column. For example DAT! will be filled in the order DAT!(1,1), DAT!(2,1), DAT!(1,2), etc. Noting this, dimension the array such that the data will be properly grouped.
`120 A$ = "POIN " + STR$(SIZE%) + "; SING; FORM4; OUTPFORM;": GOSUB IOOUTS`	Set the number of points in the trace to SIZE%, sweep once, and then hold. Tell the analyzer to send out formatted data in *form 4*, the ASCII transfer format.
`130 MAX% = 2 * SIZE%`	The maximum number of real numbers to be read in is two per point with SIZE% points.
`140 ACTUAL% = 0`	Initialize the actual number of real numbers read in. This variable is given a value by IOENTERA.
`150 CALL IOENTERA(VNA&, SEG DAT!(1, 1), MAX%, ACTUAL%): GOSUB ERRORTRAP`	Read the trace data into the array. The first field is the magnitude in dB.
`160 A$ = "OUTPLIML;": GOSUB IOOUTS`	Tell the analyzer to send out the limit test data for each point.
`170 MAX% = 4 * SIZE%`	The maximum number of real numbers to be read in during the next transfer is four per point with SIZE% points.
`180 ACTUAL% = 0`	Re-initialize the actual number of real numbers read in.

`190 CALL IOENTERA(VNA&, SEG STIM!(1, 1), MAX%, ACTUAL%): GOSUB ERRORTRAP`	Read the trace data into the array. The first field is the frequency in Hz.
`200 PRINT TAB(5); "#"; TAB(13); "MAGNITUDE"; TAB(27); "FREQUENCY"`	Display the table heading.
`210 PRINT TAB(15); "(dB)"; TAB(29); "(Hz)": PRINT`	
`220 FOR I% = 1 TO SIZE%`	Display the data for each trace point in a table on the computer CRT.
`230 PRINT USING "######"; I%;`	Display the trace point index in the desired format. For an explanation of QuickBASIC format statements, see the section entitled *Formatting Numbers* in *Microsoft QuickBASIC: Basic Language Reference*.
`240 PRINT "  "; : PRINT USING "+###.######";DAT!(1,I%);`	Display the trace point magnitude in the desired format.
`250 PRINT "  "; : PRINT USING "##.##^^^^";STIM!(1,I%)`	Display the trace point frequency in the desired format.
`260 NEXT I%`	
`270 CALL IOLOCAL(ISC&): GOSUB ERRORTRAP`	Return the network analyzer to local mode and perform error trapping.
`280 END`	End program execution.
`290 ERRORTRAP:`	Define a routine to trap errors.
`300 IF PCIB.ERR <> NOERR THEN ERROR PCIB.BASERR`	Perform error trapping.
`310 RETURN`	Return from the `ERRORTRAP` routine.
`320 IOOUTS:`	Define a routine to send a command string from the computer to the analyzer.
`330 CALL IOOUTPUTS(VNA&, A$, LEN(A$)): GOSUB ERRORTRAP`	Send the command string `A$` out to the analyzer and perform error trapping.
`340 RETURN`	Return from the `IOOUTS` routine.

Running the program

1. The computer presets the analyzer and resets the trace to eleven points.
2. The computer reads in the trace data requested by `OUTPFORM`. The first number for each point is the magnitude in dB. Regardless of the number of significant digits transmitted, the network analyzer only measures magnitude to a resolution of 0.001 dB, phase to 0.01 degrees, and group delay to 0.01 psec.
3. The computer reads in the trace data read out by `OUTPLIML`. The first number for each point is the frequency in Hz.
4. The computer displays the magnitude and frequency at the eleven points of the trace in a table.

Example 3B: Data transfer using form 5, PC-DOS 32-bit floating point format

The following program illustrates how to transfer data using *form 5. Form 5* transfers two numbers for each trace point, each number as a four-byte real number, and it uses a header, so the receiving array DAT! is set up to accommodate it. One 201-point transfer is done using `OUTPFORM` to read out magnitude data. This 201-point transfer with two real numbers per point and four bytes per point plus a four-byte header takes 1612 (201*2*4+4) bytes. Note that this same transfer in *form 4* would take 9648 (201*2*24) bytes.

This example program is stored on the Example Programs disk as **IPG3B.BAS**.

10	`REM $INCLUDE: 'QBSETUP'`	Call the QuickBASIC initialization file `QBSETUP`.
20	`CLS`	Clear the computer CRT.
30	`ISC& = 7`	Assign the interface select code to a variable.
40	`VNA& = 716`	Assign the analyzer's address to a variable.
50	`CALL IOTIMEOUT(ISC&, 10!): GOSUB ERRORTRAP`	Define a system time-out of ten seconds and perform error trapping.
60	`CALL IOABORT(ISC&): GOSUB ERRORTRAP`	Abort any HP-IB transfers and perform error trapping.
70	`CALL IOCLEAR(ISC&): GOSUB ERRORTRAP`	Clear the analyzer's HP-IB interface and perform error trapping.
80	`CALL IOEOI(ISC&, 0): GOSUB ERRORTRAP`	Disable the End-Or-Identify mode for transferring data and perform error trapping.
90	`DIM DAT!(1 TO 2, 0 TO 201)`	Prepare an array to receive the data, leaving at least four bytes of space before the desired data destination to account for the two-integer header.
100	`A$ = "SING; FORM5; OUTPFORM;": GOSUB IOOUTS`	Sweep once and then hold. Tell the analyzer to send out formatted data in *form 5,* PC-DOS 32-bit floating point.
110	`MAX% = 201 * 4 * 2 + 4`	The maximum number of bytes to be read in is two 4-byte real numbers per point with 201 points plus a four-byte (two-integer) header.
120	`ACTUAL% = 0`	Initialize the actual number of bytes read in. This variable is given a value by `IOENTERB`.
130	`FLAG% = 1`	No swapping of bytes is desired.
140	`CALL IOENTERB(VNA&, SEG DAT!(2, 0), MAX%, ACTUAL%, FLAG%): GOSUB ERRORTRAP`	Read in the data, specifying the beginning array address as one real number (four bytes) before the desired destination of the true data in order to account for the header and therefore maintain data grouping.
150	`PRINT USING "+###.#####"; DAT!(1, 1); DAT!(1, 201)`	Display the first and last data point values. Only the first value of the pair of numbers for each point (the magnitude in dB) is significant.
160	`A$ = "CONT;": GOSUB IOOUTS`	Restore continuous sweep trigger mode to the analyzer.
170	`CALL IOLOCAL(ISC&): GOSUB ERRORTRAP`	Return the network analyzer to local mode and perform error trapping.
180	`END`	End program execution.
190	`ERRORTRAP:`	Define a routine to trap errors.
200	`IF PCIB.ERR <> NOERR THEN ERROR PCIB.BASERR`	Perform error trapping.
210	`RETURN`	Return from the `ERRORTRAP` routine.
220	`IOOUTS:`	Define a routine to send a command string from the computer to the analyzer.

230 CALL IOOUTPUTS(VNA&, A$, LEN(A$)): GOSUB ERRORTRAP	Send the command string A$ out to the analyzer and perform error trapping.
240 RETURN	Return from the IOOUTS routine.

Running the program

1. The computer reads in the trace data requested by OUTPFORM in *form 5*. The first number for each point is the magnitude in dB.

2. The computer displays the first and last magnitude values read in.

   Now go to the analyzer and press **[MENU]** *[NUMBER OF POINTS]* **[4] [0] [1] [x1]**. Run the program again. Note that although the program does not generate an error, only half of the data was read in since the computer only expected the data for 201 points. In this case the analyzer is still waiting to transfer data.

   Now change the number of points to 101. Run the program again. Note that a QuickBASIC error was generated since the analyzer ran out of data to transmit before the computer received the data from 201 points that it was expecting.

   It is imperative that the receiving array be correctly dimensioned. Fortunately, this is easy to ensure because not only is the number of points in the analyzer's trace readily available through POIN?, but the size of the transfer block is also easily determined from the header. In addition, QuickBASIC allows dimension statements anywhere in a program, so it is possible to wait until the size of the transfer is known to dimension the receiving array.

The above example program can be modified to take advantage of this by making the following changes:

- Change line 90 to the following:

90 DIM HEADER%(0 TO 1)	Prepare an array to receive the two-integer header.

- Delete line 110.
- Insert the following lines between lines 100 and 120:

102 MAX% = 4	The maximum number of bytes to be read in is only the four byte header.
105 ACTUAL% = 0	Initialize the actual number of bytes read in. This variable is given a value by IOENTERB.
108 FLAG% = 1	No swapping of bytes is desired.
110 CALL IOENTERB(VNA&, SEG HEADER%(0), MAX%, ACTUAL%, FLAG%): GOSUB ERRORTRAP	Read in the header as two integers. The second integer is the number of bytes of the trace data that would follow if MAX% were not set to read in only the header.
112 DIM DAT!(1 TO 2, 0 TO HEADER%(1) / 8)	Prepare an array to receive the data. The necessary size of the array can be determined from the known number of bytes of the trace data. (There are HEADER%(1) bytes with four bytes per real number and two real numbers per point.)
115 A$ = "OUTPFORM;": GOSUB IOOUTS	Tell the analyzer to send out data formatted data in *form 5*, PC-DOS 32-bit floating point.
118 MAX% = HEADER%(1) + 4	The maximum number of bytes to be read in is the number of bytes following the header, given by HEADER%(1), plus the four bytes in the header.

This modified program is stored on the Example Programs disk as **IPG3BX.BAS**.

Two transfers are done using OUTPFORM. The first transfer reads in only the four-byte header (as two integers) before it terminates. The second of these integers is the size in bytes of the block of data to follow, and with this the receiving array can be correctly dimensioned regardless of the number of points in the trace.

Example 3C: Data transfer using form 1, network analyzer internal format

The following program illustrates how to transfer data using *form 1. Form 1* transfers a six-byte binary string of data for each trace point. The six bytes can be represented as three integers, and *form 1* uses a four-byte header, which can be read in as two integers, so the receiving array `DAT!` is set up to accommodate this. One transfer is done using `OUTPDATA` to determine the size of the data block. The receiving array is then correctly dimensioned, and a second transfer is done using `OUTPDATA` to receive all of the trace data. If there is a 201-point trace, with six-bytes per point plus a four-byte header, this transfer takes only 1210 (201*6+4) bytes. This is considerably faster than the same transfer in either *form 4* or *form 5.*

However, the data received in *form 1* is difficult to decode. Real/imaginary data uses the first two bytes for the imaginary fraction mantissa, the middle two bytes for the real fraction mantissa, the fifth byte for additional resolution when transferring raw data, and the last byte as the common power of two. The data could be recombined and displayed on the computer, but since this requires reformatting time, *form 1* is most useful for getting data to store on disk, as shown in the following program.

This example program is stored on the Example Programs disk as **IPG3C.BAS**.

`10 REM $INCLUDE: 'QBSETUP'`	Call the QuickBASIC initialization file `QBSETUP`.
`20 CLS`	Clear the computer CRT.
`30 ISC& = 7`	Assign the interface select code to a variable.
`40 VNA& = 716`	Assign the analyzer's address to a variable.
`50 CALL IOTIMEOUT(ISC&, 10!): GOSUB ERRORTRAP`	Define a system time-out of ten seconds and perform error trapping.
`60 CALL IOABORT(ISC&): GOSUB ERRORTRAP`	Abort any HP-IB transfers and perform error trapping.
`70 CALL IOCLEAR(ISC&): GOSUB ERRORTRAP`	Clear the analyzer's HP-IB interface and perform error trapping.
`80 CALL IOEOI(ISC&, 0): GOSUB ERRORTRAP`	Disable the End-Or-Identify mode for transferring data and perform error trapping.
`90 DIM HEADER%(0 TO 1)`	Prepare an array to receive the four-byte header as two integers.
`100 A$ = "SING; FORM1; OUTPDATA;": GOSUB IOOUTS`	Sweep once and then hold. Tell the analyzer to send out corrected data in *form 1,* instrument internal binary.
`110 MAX% = 4`	The maximum number of bytes to be read in is only the four-byte header.
`120 ACTUAL% = 0`	Initialize the actual number of bytes read in. This variable is given a value by `IOENTERB`.
`130 FLAG% = 4`	Reverse every four bytes.
`140 CALL IOENTERB(VNA&, SEG HEADER%(0), MAX%, ACTUAL%, FLAG%): GOSUB ERRORTRAP`	Read in the header as two integers. The first integer is the number of bytes of the trace data that would follow if `MAX%` were not set to read in only the header.
`150 DIM DAT%(1 TO 3, 0 TO HEADER%(0) / 6)`	Prepare an array to receive the data. The necessary size of the array can be determined from the known number of bytes of the trace data. (In addition to one four-byte header, there are six bytes per point in *form 1,* so allocate three integers per point.)
`160 A$ = "OUTPDATA;": GOSUB IOOUTS`	Tell the analyzer to send out corrected data in *form 1,* instrument internal binary.

`170 MAX% = HEADER%(0) + 4`	The maximum number of bytes to be read in is the number of bytes following the header, given by HEADER%(0), plus four bytes in the header.
`180 ACTUAL% = 0`	Re-initialize the actual number of bytes read in.
`190 FLAG% = 1`	Because the data is only going to be stored in a file and not seen, no swapping of bytes is necessary.
`200 CALL IOENTERB(VNA&, SEG DAT%(2, 0), MAX%, ACTUAL%, FLAG%): GOSUB ERRORTRAP`	Read in the data, specifying the beginning array address as two integers (four bytes) before the desired destination of the true data in order to account for the header and therefore maintain data grouping.
`210 OPEN "TESTDATA" FOR BINARY AS #1`	Open the binary storage file.
`220 PUT #1, , HEADER%(0)`	Store the number of bytes of the trace data in the storage file.
`230 PUT #1, , DAT%(2, 0)`	Store the four-byte header in the storage file as two integers.
`240 PUT #1, , DAT%(3, 0)`	
`250 FOR I% = 1 TO HEADER%(0) / 6`	
`260  PUT #1, , DAT%(1, I%)`	Store the trace data in the storage file.
`270  PUT #1, , DAT%(2, I%)`	
`280  PUT #1, , DAT%(3, I%)`	
`290 NEXT I%`	
`300 CLOSE #1`	Close the storage file.
`310 PRINT "CHANGE SETUP AND PRESS <ENTER>."`	Display instructions on the computer CRT.
`320 DO UNTIL INKEY$ = CHR$(13): LOOP`	Wait for the operator to change the trace.
`330 OPEN "TESTDATA" FOR BINARY AS #1`	Open the binary storage file.
`340 GET #1, , HEADER%(0)`	Read the number of bytes of trace data from the storage file.
`350 GET #1, , DAT%(2, 0)`	Read the header from the storage file.
`360 GET #1, , DAT%(3, 0)`	
`370 FOR I% = 1 TO (HEADER%(0) / 6)`	
`380  GET #1, , DAT%(1, I%)`	Read the trace data from the storage file.
`390  GET #1, , DAT%(2, I%)`	
`400  GET #1, , DAT%(3, I%)`	
`410 NEXT I%`	
`420 CLOSE #1`	Close the storage file.
`430 A$ = "SING;": GOSUB IOOUTS`	Sweep once to view the current setup's trace on the analyzer and then hold.
`440 PRINT "PRESS <ENTER> TO CONTINUE.": DO UNTIL INKEY$ = CHR$(13): LOOP`	Allow the operator to view the current setup's trace before continuing.
`450 A$ = "INPUDATA;": GOSUB IOOUTS`	Prepare the analyzer to read in corrected data.
`460 MAX% = HEADER%(0) + 4`	The maximum number of bytes to be sent out is the number of bytes following the header, given by HEADER%(0), plus the four bytes in the header.
`470 FLAG% = 1`	No swapping of bytes is desired.

480 CALL IOOUTPUTB(VNA&, SEG DAT%(2, 0), MAX%, FLAG%): GOSUB ERRORTRAP	Send out the data, specifying the beginning array address as two integers (four bytes) before the address where the true data is stored in order to account for the header.
490 KILL "TESTDATA"	Delete the data file.
500 CALL IOLOCAL(ISC&): GOSUB ERRORTRAP	Return the network analyzer to local mode and perform error trapping.
510 END	End program execution.
520 ERRORTRAP:	Define a routine to trap errors.
530 IF PCIB.ERR <> NOERR THEN ERROR PCIB.BASERR	Perform error trapping.
540 RETURN	Return from the ERRORTRAP routine.
550 IOOUTS:	Define a routine to send a command string from the computer to the analyzer.
560 CALL IOOUTPUTS(VNA&, A$, LEN(A$)): GOSUB ERRORTRAP	Send the command string A$ out to the analyzer and perform error trapping.
570 RETURN	Return from the IOOUTS routine.

Running the program

1. The computer initiates a transfer using OUTPDATA, reads in the four-byte header as two integers, and terminates the transfer. The second of these integers is the size in bytes of the block of data to follow, and with this, the receiving array is correctly dimensioned.

2. The computer reads in all the trace data requested by OUTPDATA.

3. The computer stores the size of the block of data and the data in the hard disk file TESTDATA. If a hard disk is not available, change the file name on lines 210 and 330 to A:TESTDATA, and make sure that there is a formatted non-write-protected) disk in the A: drive.

4. Change the setup on the analyzer as prompted by the computer by, for example, disconnecting the test device.

5. The computer reads the trace data back in from the storage file, sends the data out to the analyzer, and deletes the storage file.

Example 3D: Data transfer using markers

The following program illustrates how to transfer data using markers and the command OUTPMARK. In order to read data off a trace using a marker, the marker must first be made active and put at the desired frequency using a command to select a specific stimulus value, like MARK1 133.15MHZ, or a command to do a marker search, like MARK3; SEAMIN. The command OUTPMARK tells the network analyzer to transmit three numbers: marker value one, marker value two, and marker stimulus value. See Table 1 (page 20) to identify the first two marker values according to the current display format. The third marker value, the stimulus value, is either frequency or time, depending on the network analyzer's active domain. These three values can be read in as an array of real numbers using the routine IOENTERA. In this case, there is no header, and MAX% is the maximum number of real numbers to read in (3).

This Example Program is stored on the Example Programs disk as **IPG3D.BAS**.

`10 REM $INCLUDE: 'QBSETUP'`	Call the QuickBASIC initialization file QBSETUP.
`20 CLS`	Clear the computer CRT.
`30 ISC& = 7`	Assign the interface select code to a variable.
`40 VNA& = 716`	Assign the analyzer's address to a variable.
`50 DISPLAY& = 717`	Assign the analyzer's display address to a variable.
`60 CALL IOTIMEOUT(ISC&, 10!): GOSUB ERRORTRAP`	Define a system time-out of ten seconds and perform error trapping.
`70 CALL IOABORT(ISC&): GOSUB ERRORTRAP`	Abort any HP-IB transfers and perform error trapping.
`80 CALL IOCLEAR(ISC&): GOSUB ERRORTRAP`	Clear the analyzer's HP-IB interface and perform error trapping.
`90 CALL IOEOI(ISC&, 0): GOSUB ERRORTRAP`	Disable the End-Or-Identify mode for transferring data and perform error trapping.
`100 DIM VALU!(0 TO 2)`	Allocate space to hold data read in from the analyzer.
`110 ADDRESS& = VNA&`	Initialize the output address to the address of the network analyzer.
`120 A$ = "PRES;": GOSUB IOOUTS`	Preset the network analyzer.
`130 A$ = "CHAN1; S21; LOGM;": GOSUB IOOUTS`	Make channel 1 the active channel and measure the magnitude of forward transmission parameter S21 in decibels.
`140 A$ = "CENT 134MHz;": GOSUB IOOUTS`	Set the center frequency to 134 MHz.
`150 A$ = "SPAN 25MHz;": GOSUB IOOUTS`	Set the frequency span to 25 MHz.
`160 A$ = "AUTO;": GOSUB IOOUTS`	Autoscale the resulting trace.
`170 A$ = "SING; MARK3; SEAMIN;": GOSUB IOOUTS`	Sweep once, hold, and set marker three at the minimum magnitude value of the trace.
`180 A$ = "MARK4; SEAMAX;": GOSUB IOOUTS`	Set marker four at the maximum magnitude value of the trace.
`190 A$ = "MARK1 133.15MHz; OUTPMARK;": GOSUB IOOUTS`	Set marker one at 133.15 MHz, sweep once, and request marker data from marker one. Since the format is log magnitude, only the first value (the magnitude at the marker in dB) and the third value (the frequency in Hz) read in are significant. → See Table 1.

200 MAX% = 3	Set the maximum number of real numbers to be read in from the analyzer.
210 ACTUAL% = 0	Initialize the actual number of real numbers read in. This variable is given a value by IOENTERA.
220 CALL IOENTERA(VNA&, SEG VALU!(0), MAX%, ACTUAL%): GOSUB ERRORTRAP	Read in marker data from the analyzer.
230 PRINT " MARKER AT 133.15 MHz:"	
240 PRINT " FROM LOG MAGNITUDE PLOT:"	Display a heading.
250 PRINT TAB(15); VALU!(0); " DB"	Display the magnitude value just read in.
260 GOSUB WAITING	Wait for the user to press any network analyzer key before continuing.
270 A$ = "PHAS; AUTO;": GOSUB IOOUTS	Display the phase of the active transmission parameter and autoscale the resulting trace.
280 A$ = "MARK1; OUTPMARK;": GOSUB IOOUTS	Request marker data from marker one. Since the format is phase, only the first value (the phase at the marker in degrees) and the third value (the frequency in Hz) read in are significant. → See Table 1. Note that a single sweep / hold is not necessary here because only format has changed.
290 ACTUAL% = 0	Re-initialize the actual number of real numbers read in.
300 CALL IOENTERA(VNA&, SEG VALU!(0), MAX%, ACTUAL%): GOSUB ERRORTRAP	Read in marker data from the analyzer.
310 PRINT " FROM PHASE PLOT:"	Display a heading.
320 PRINT TAB(15); VALU!(0); " DEGREES"	Display the phase value just read in.
330 GOSUB WAITING	Wait for the user to press any network analyzer key before continuing.
340 A$ = "LINM; AUTO;": GOSUB IOOUTS	Display the linear magnitude of the active transmission parameter and autoscale the resulting trace.
350 A$ = "MARK1; OUTPMARK;": GOSUB IOOUTS	Request marker data from marker one. Since the format is linear magnitude, only the first value (the linear magnitude) and the third value (the frequency in Hz) read in are significant. → See Table 1.
360 ACTUAL% = 0	Re-initialize the actual number of real numbers read in.
370 CALL IOENTERA(VNA&, SEG VALU!(0), MAX%, ACTUAL%): GOSUB ERRORTRAP	Read in marker data from the analyzer.
380 PRINT " FROM LINEAR MAGNITUDE PLOT:"	Display a heading.
390 PRINT TAB(15); VALU!(0); " UNITS"	Display the magnitude value just read in.
400 GOSUB WAITING	Wait for the user to press any network analyzer key before continuing.

410 A$ = "SMIC; AUTO; SMIMRX;": GOSUB IOOUTS	Display the Smith chart of the active transmission parameter and autoscale the trace. Set the marker data to be given in the form R + jX.
420 A$ = "MARK1; OUTPMARK;": GOSUB IOOUTS	Request marker data from marker one. In this configuration, the first value (real in ohms), the second value (imaginary in ohms), and the third value (the frequency in Hz) read in are significant. → See Table 1.
430 ACTUAL% = 0	Re-initialize the actual number of real numbers read in.
440 CALL IOENTERA(VNA&, SEG VALU!(0), MAX%, ACTUAL%): GOSUB ERRORTRAP	Read in marker data from the analyzer.
450 PRINT " FROM SMITH CHART:"	Display a heading.
460 PRINT TAB(15); VALU!(0); " + j "; VALU!(1); " OHMS"	Display the normalized impedance values just read in.
470 GOSUB WAITING	Wait for the user to press any network analyzer key before continuing.
480 A$ = "POLA; AUTO; POLMRI;": GOSUB IOOUTS	Display the active transmission parameter in polar form and autoscale the trace. Set the marker data to be in the form real/imaginary.
490 A$ = "MARK1; OUTPMARK;": GOSUB IOOUTS	Request marker data from marker one. In this configuration, the first value (real), the second value (imaginary), and the third value (the frequency in Hz) read in are significant. → See Table 1.
500 ACTUAL% = 0	Re-initialize the actual number of real numbers read in.
510 CALL IOENTERA(VNA&, SEG VALU!(0), MAX%, ACTUAL%): GOSUB ERRORTRAP	Read in marker data from the analyzer.
520 PRINT " FROM POLAR PLOT:"	Display a heading.
530 PRINT TAB(15); VALU!(0); " + j "; VALU!(1); " UNITS"	Display the values just read in.
540 CALL IOLOCAL(ISC&): GOSUB ERRORTRAP	Return the network analyzer to local mode and perform error trapping.
550 END	Perform error trapping.
560 ERRORTRAP:	Define a routine to trap errors.
570 IF PCIB.ERR <> NOERR THEN ERROR PCIB.BASERR	Perform error trapping.
580 RETURN	Return from the ERRORTRAP routine.
590 IOOUTS:	Define a routine to send a command string from the computer to the analyzer.
600 CALL IOOUTPUTS(ADDRESS&, A$, LEN(A$)): GOSUB ERRORTRAP	Send the command string A$ out to the analyzer and perform error trapping.
610 RETURN	Return from the IOOUTS routine.
620 WAITING:	Define a routine to display a prompt on the network analyzer's display and wait for the user to press any key before continuing.
630 ADDRESS& = DISPLAY&	Reset the output address to the network analyzer's display.

640 A$ = "PU; PA 390,3600; PD; LBPRESS ANY KEY TO CONTINUE" + CHR$(3): GOSUB IOOUTS	Write a prompt on the network analyzer's display.
650 ADDRESS& = VNA&	Return the output address to the network analyzer.
660 A$ = "CLES; ESE64;": GOSUB IOOUTS	Set up the status reporting system so that bit 6, User Request, of the Event Status Register is summarized by bit 5 of the Status Byte, allowing a key press to be detected by a serial poll.
670 A$ = "ESR?;": GOSUB IOOUTS	Request the Event Status Register value from the analyzer.
680 CALL IOENTER(VNA&, ESTAT!): GOSUB ERRORTRAP	Receive the Event Status Register value from the analyzer, thereby clearing the latched User Request bit so that old key presses will not trigger a measurement.
690 STAT% = 0	Initialize STAT% for entry into the DO UNTIL loop.
700 DO UNTIL ((STAT% MOD 64) > 31)	Wait for a key press to be indicated by the setting of bit 5 in the status byte. MOD 64 removes the effect of all higher value bits (bit 6 is equivalent to 64 in decimal), and > 31 ensures that bit 5, which is equivalent to 32 in decimal, is set.
710 CALL IOSPOLL(VNA&, STAT%): GOSUB ERRORTRAP	Read in the status byte as an integer.
720 LOOP	
730 ADDRESS& = DISPLAY&	Reset the output address to the network analyzer's display.
740 A$ = "PG;": GOSUB IOOUTS	Clear old user graphics from the network analyzer's display.
750 ADDRESS& = VNA&	Return the output address to the network analyzer.
760 RETURN	Return from the WAITING routine.

Running the program

1. The computer sets up a trace on the analyzer and puts markers at the maximum and minimum log magnitudes of the trace as well as at a specific frequency.

2. The computer reads in the data from marker one read out by OUTPMARK. Press any key on the analyzer front panel to continue the program, go on to a new display format, and read in its data from marker one. Note that only the identity of the first two marker data values varies with the current display format and marker mode; the command to read out the marker data, OUTPMARK and the number of values to be read (3) is always the same.

Advanced Programming Examples

Using list frequency mode

The network analyzer normally takes data points spaced at regular intervals across the overall frequency range of the measurement. For a 2 GHz linear frequency sweep with 201 points, data will be taken at intervals of 10 MHz. The list frequency mode, however, lets you select the specific points or frequency spacing between points at which measurements are to be made. This allows flexibility in setting up tests, and it reduces measurement time since device performance is not measured at frequencies not needed.

The following examples illustrate the use of the network analyzer's list frequency mode to perform arbitrary frequency testing. Example 4A constructs a table of list frequency segments which is then loaded into the network analyzer's list frequency table. Each segment stipulates a start frequency, a stop frequency, and the number of data points to be taken over that frequency range. The command sequence for entering a list frequency table imitates the key sequence followed when entering a table from the front panel in that there is a command for every key press. Editing a segment is also the same as the key sequence, and the network analyzer automatically reorders each edited segment in order of increasing start frequency.

Example 4B selects a specific segment of the list frequency table to "zoom-in" on. This is useful when a single instrument is being used to measure several different devices, each with its own frequency range. Using a single calibration performed with all of the segments active, each specific device can be measured by selecting the appropriate segment for that device.

The list frequency segments can be overlapped, but the number of points in all the segments must not exceed 1632 points. Also, the list frequency table is carried as part of the learn string. While it cannot be modified in this form, it can easily be stored and recalled.

Example 4A: List frequency sweep

The following program illustrates how to create a list frequency table on the computer and transmit it to the analyzer. It takes advantage of the computer's ability to simplify creating, adding to, and editing the table. The table is entered and completely edited before it is transmitted to the analyzer. For simplicity, the options to enter center, span, and step size are not given.

This example program is stored on the Example Programs disk as **IPG4A.BAS**.

10	`REM $INCLUDE: 'QBSETUP'`	Call the QuickBASIC initialization file QBSETUP.
20	`CLS`	Clear the computer CRT.
30	`ISC& = 7`	Assign the interface select code to a variable.
40	`VNA& = 716`	Assign the analyzer's address to a variable.
50	`CALL IOTIMEOUT(ISC&, 10!): GOSUB ERRORTRAP`	Define a system time-out of ten seconds and perform error trapping.
60	`CALL IOABORT(ISC&): GOSUB ERRORTRAP`	Abort any HP-IB transfers and perform error trapping.
70	`CALL IOCLEAR(ISC&): GOSUB ERRORTRAP`	Clear the analyzer's HP-IB interface and perform error trapping.
80	`CALL IOEOI(ISC&, 0): GOSUB ERRORTRAP`	Disable the End-Or-Identify mode for transferring data and perform error trapping.
90	`LOCATE 1, 1: INPUT "NUMBER OF SEGMENTS? ", NUMBER%`	Read in the desired number of segments from the operator's input.
100	`DIM TABLE!(1 TO 3, 1 TO NUMBER%)`	Create an array to hold the segment data (start frequency, stop frequency, and number of points for each segment).
110	`GOSUB CLEARLINES`	Clear the CRT lines being used for data entry.
120	`LOCATE 5, 1: PRINT "SEGMENT"; TAB(15); "START(MHz)"; TAB(32); "STOP(MHz)"; TAB(49); "NUMBER OF POINTS";`	Display the segment table header on the computer CRT.
130	`FOR I% = 1 TO NUMBER%`	Repeat for each segment in the segment list.

140 GOSUB LOADPOINT	Load the data for the current segment, TABLE!(1 TO 3,I%). Since LOADPOINT is a subroutine, I% is used as a global variable.
150 GOSUB CLEARDATA	Clear the current segment data from the CRT lines being used for data entry.
160 NEXT I%	
170 GOSUB CLEARLINES	Clear the CRT lines being used for data entry.
180 LOCATE 1, 1: INPUT "DO YOU WANT TO EDIT (Y/N)? ", ANSWER$	Determine if editing is initially desired.
190 DO UNTIL ((ANSWER$ = "N") OR (ANSWER$ = "n"))	Repeat until all editing has been done.
200 INPUTENTRY: LOCATE 1, 40: INPUT "ENTRY NUMBER? ", I%	Get the number of the segment to be edited.
210 IF ((I% <1) OR (I%> NUMBER%)) THEN GOTO INPUTENTRY	Make sure the segment number is valid.
220 GOSUB LOADPOINT	Re-enter the segment data.
230 GOSUB CLEARLINES	Clear the CRT lines being used for data entry.
240 LOCATE 1, 1: INPUT "DO YOU WANT TO EDIT (Y/N)? ", ANSWER$	Determine if more editing is desired.
250 LOOP	
260 A$ = "EDITLIST; CLEL;": GOSUB IOOUTS	To begin sending the table to the analyzer, open the analyzer's list frequency table for editing, and delete any existing segments.
270 FOR I% = 1 TO NUMBER%	Loop for each segment.
280 A$ = "SADD; STAR " + STR$(TABLE!(1, I%)) + "MHz;": GOSUB IOOUTS	
290 A$ = "STOP " + STR$(TABLE!(2, I%)) + "MHz;": GOSUB IOOUTS	
300 A$ = "POIN " + STR$(TABLE!(3, I%)) + ";": GOSUB IOOUTS	
310 A$ = "SDON;": GOSUB IOOUTS	Add a segment, specifying its start frequency, its stop frequency, and the number of points it is made up of. Then declare the current frequency list segment done.
320 NEXT I%	
330 A$ = "EDITDONE; LISFREQ;": GOSUB IOOUTS	Close the edit frequency list table and activate the list frequency mode.
340 A$ = "AUTO;": GOSUB IOOUTS	Autoscale the trace.
350 CALL IOLOCAL(ISC&): GOSUB ERRORTRAP	Return the network analyzer to local mode and perform error trapping.
360 END	End program execution.
370 ERRORTRAP:	Define a routine to trap errors.
380 IF PCIB.ERR <> NOERR THEN ERROR PCIB.BASERR	Perform error trapping.
390 RETURN	Return from the ERRORTRAP routine.

400 IOOUTS:	Define a routine to send a command string from the computer to the analyzer.
410 CALL IOOUTPUTS(VNA&, A$, LEN(A$)): GOSUB ERRORTRAP	Send the command string A$ out to the analyzer and perform error trapping.
420 RETURN	Return from the IOOUTS routine.
430 LOADPOINT:	Define a routine to read in all of one segment's data from the operator and load it into the data table on the computer.
440 GOSUB CLEARLINES	Clear the CRT lines being used for data entry.
450 LOCATE 1, 1: PRINT "SEGMENT: "; STR$(I%); TAB(40); "STOP FREQUENCY (MHz)?"	
460 LOCATE 2, 1: PRINT "START FREQUENCY (MHz)?"; TAB(40); "NUMBER OF POINTS?"	Display the input labels.
470 IF ((TABLE!(1, I%) <> 0) OR (TABLE!(2, I%) <> 0) OR (TABLE!(3, I%) <> 0)) THEN	If the segment contains valid data, display it at the entry locations.
480 LOCATE 2, 23: PRINT TABLE!(1, I%);	
490 LOCATE 1, 61: PRINT TABLE!(2, I%);	
500 LOCATE 2, 57: PRINT TABLE!(3, I%);	
510 END IF	
520 SAVE! = TABLE!(1, I%)	Save the start frequency of the current table entry.
530 LOCATE 2, 22: INPUT TABLE!(1, I%)	Read the start frequency of the segment.
540 IF TABLE!(1, I%) = 0 THEN TABLE!(1, I%) = SAVE!	If no value or 0 was entered, return the start frequency to its previous value.
550 LOCATE 2, 23: PRINT SPACE$(16): LOCATE 2, 23: PRINT TABLE!(1, I%);	Display the new start frequency.
560 SAVE! = TABLE!(2, I%)	Save the stop frequency of the current table entry.
570 LOCATE 1, 60: INPUT TABLE!(2, I%)	Read the stop frequency of the segment.
580 IF TABLE!(2, I%) = 0 THEN TABLE!(2, I%) = SAVE!	If no value or 0 was entered, return the stop frequency to its previous value.
590 LOCATE 1, 61: PRINT SPACE$(19): LOCATE 1, 61: PRINT TABLE!(2, I%);	Display the new stop frequency.
600 SAVE! = TABLE!(3, I%)	Save the number of points of the current table entry.
610 TABLE!(3, I%) = 0	Set TABLE!(3, I%) for entry into the DO UNTIL loop.
620 DO UNTIL (TABLE!(3, I%) > 0)	Repeat until a valid number of points has been entered.
630 LOCATE 2, 56: INPUT TABLE!(3, I%)	Read the number of points in the segment.

640 IF ((TABLE!(3, I%) = 0) AND (SAVE! <> 0)) THEN TABLE!(3, I%) = SAVE!	If no value or 0 was entered and the previous value was valid, return the number of points to that previous value.
650 LOOP	
660 LOCATE 2, 57: PRINT SPACE$(23): LOCATE 2, 57: PRINT TABLE!(3, I%);	Display the new number of points.
670 IF (TABLE!(3, I%) = 1) THEN TABLE!(2, I%) = TABLE!(1, I%)	If there is only one point in the segment, let the stop frequency equal the start frequency to avoid ambiguity.
680 LOCATE I% + 5, 3: PRINT I%; TAB(17); TABLE!(1, I%); TAB(34); TABLE!(2, I%); TAB(54); TABLE!(3, I%);	Display the new data in the table.
690 RETURN	Return from the LOADPOINT routine.
700 CLEARLINES:	Define a routine to clear the CRT lines used for data entry.
710 FOR J% = 1 TO 3	Clear each line.
720 LOCATE J%, 1: PRINT SPACE$(80);	
730 NEXT J%	
740 RETURN	Return from the CLEARLINES routine.
750 CLEARDATA:	Define a routine to clear only the data (not the prompts) from the CRT lines used for data entry.
760 LOCATE 1, 61: PRINT SPACE$(19): LOCATE 2, 23: PRINT SPACE$(16);	
770 RETURN	Return from the CLEARDATA routine.

Running the program

1. The computer clears the analyzer's list frequency table. If this is not desired, remove the CLEL command from line 90.

2. Enter the number of segments and then the parameters of each segment as prompted.

3. Edit the computer's list frequency table until it is satisfactory. Pressing <ENTER> at a prompt during editing leaves the parameter at its current value.

4. The computer sends the completed list frequency table out to the analyzer, which orders the segments, activates the list frequency mode, and displays an all-segment sweep.

Example 4B: Single segment selection

The following program illustrates how to read the list frequency table data out of the network analyzer and choose a single segment out of this table of segments to be the operating frequency range of the network analyzer. It is assumed that a list frequency table has already been entered into the analyzer, either manually or over HP-IB as shown in the previous example.

This example program is stored on the Example Programs disk as **IPG4B.BAS**.

10 REM $INCLUDE: 'QBSETUP'	Call the QuickBASIC initialization file QBSETUP.
20 CLS	Clear the computer CRT.
30 ISC& = 7	Assign the interface select code to a variable.
40 VNA& = 716	Assign the analyzer's address to a variable.
50 CALL IOTIMEOUT(ISC&, 20!): GOSUB ERRORTRAP	Define a system time-out of twenty seconds and perform error trapping. This time-out is longer than usual because when there are many points, the HP 8752A factory correction takes more than 10 seconds to adjust to a new frequency range. If the timeout is set to only 10 seconds, a time-out error may be generated when nothing is wrong.
60 CALL IOABORT(ISC&): GOSUB ERRORTRAP	Abort any HP-IB transfers and perform error trapping.
70 CALL IOCLEAR(ISC&): GOSUB ERRORTRAP	Clear the analyzer's HP-IB interface and perform error trapping.
80 CALL IOEOI(ISC&, 0): GOSUB ERRORTRAP	Disable the End-Or-Identify mode for transferring data and perform error trapping.
90 LOCATE 2, 1: PRINT TAB(4); "SEGMENT"; TAB(22); "START (MHz)"; TAB(42); "STOP (MHz)"; TAB(59); "NUMBER OF POINTS"	Display the table heading.
100 A$ = "EDITLIST; SEDI30; SEDI?": GOSUB IOOUTS	Request segment 30, the largest possible segment number, and the analyzer will automatically select the last segment. Then output its number to the computer.
110 CALL IOENTER(VNA&, NUMSEGS!): GOSUB ERRORTRAP	Because there in no HP-IB Command Library routine to read in an integer, read the last segment number into the real variable NUMSEGS!.
120 NUMSEGS% = INT(NUMSEGS!)	Convert the number of segments to an integer.
130 DIM TABLE!(1 TO 3, 1 TO NUMSEGS%)	Create an array to hold all of the segment parameters.
140 FOR I% = 1 TO NUMSEGS%	Read the segment parameters from the analyzer for each segment.
150 GOSUB READLIST	
160 NEXT I%	
170 LOCATE 1, 1: INPUT "SELECT SEGMENT NUMBER (0 TO EXIT): ", SEGMENT%	Determine which segment the operator wishes to activate. Entering 0 exits the loop.
180 DO UNTIL (SEGMENT% = 0)	Repeat until the operator enters 0.
190 LOCATE 3, 1: PRINT SPACE$(80);	Clear the current segment display line on the computer CRT.
200 IF ((NUMSEGS% > 20) AND (SEGMENT% < 21)) THEN	Display the desired segment's data at the top of the table if it is not already on the display screen.
210 LOCATE 3, 1: PRINT USING "##"; TAB(6); SEGMENT%;	

220 PRINT USING "#####.##"; TAB(23); TABLE!(1, SEGMENT%) / 1000000; TAB(42); TABLE!(2, SEGMENT%) / 1000000;	
230 PRINT USING "####"; TAB(65); TABLE!(3, SEGMENT%)	
240 END IF	
250 A$ = "EDITDONE; SSEG" + STR$(SEGMENT%) + ";": GOSUB IOOUTS	Make the desired segment the new operating frequency range of the measurement.
260 A$ = "AUTO;": GOSUB IOOUTS	Autoscale the trace.
270 LOCATE 1, 36: PRINT SPACE$(10);	Clear the segment number entry display.
280 LOCATE 1, 1: INPUT "SELECT SEGMENT NUMBER (0 TO EXIT): ", SEGMENT%	Determine which segment the operator wishes to activate.
290 LOOP	
300 A$ = "ASEG;": GOSUB IOOUTS	Resume operation using all list frequency segments.
310 A$ = "AUTO;": GOSUB IOOUTS	Autoscale the trace.
320 CALL IOLOCAL(ISC&): GOSUB ERRORTRAP	Return the network analyzer to local mode and perform error trapping.
330 END	End program execution.
340 ERRORTRAP:	Define a routine to trap errors.
350 IF PCIB.ERR <> NOERR THEN ERROR PCIB.BASERR	Perform error trapping.
360 RETURN	Return from the ERRORTRAP routine.
370 IOOUTS:	Define a routine to send a command string from the computer to the analyzer.
380 CALL IOOUTPUTS(VNA&, A$, LEN(A$)): GOSUB ERRORTRAP	Send the command string A$ out to the analyzer and perform error trapping.
390 RETURN	Return from the IOOUTS routine.
400 READLIST:	Define a routine to read all of one segment's parameters from the analyzer and display them on the computer CRT.
410 A$ = "EDITLIST; SEDI" + STR$(I%) + ";": GOSUB IOOUTS	Activate the I%th segment.
420 A$ = "STAR?;": GOSUB IOOUTS	Interrogate the start frequency of the analyzer.
430 CALL IOENTER(VNA&, TABLE!(1, I%)): GOSUB ERRORTRAP	Read the start frequency into the computer's list table.
440 A$ = "STOP?;": GOSUB IOOUTS	Interrogate the stop frequency of the analyzer.
450 CALL IOENTER(VNA&, TABLE!(2, I%)): GOSUB ERRORTRAP	Read the stop frequency into the computer's list table.

460 A$ = "POIN?;": GOSUB IOOUTS	Interrogate the number of points of the analyzer.
470 CALL IOENTER(VNA&, TABLE!(3, I%)): GOSUB ERRORTRAP	Read the number of points into the computer's list table.
480 IF (I% < 21) THEN	The first twenty segments will fit on the screen at once.
490 ROW% = 3 + I%	Set the segment data display row accordingly.
500 ELSEIF (I% = 21) THEN	There are too many segments to fit on the screen at once.
510 LOCATE 24, 1: PRINT "PRESS <ENTER> TO CONTINUE";	
520 DO UNTIL INKEY$ = CHR$(13): LOOP	Wait for the user to continue before clearing the screen.
530 FOR J% = 4 TO 24	Clear the lines used to display the data from the first twenty segments.
540 LOCATE J%, 1: PRINT SPACE$(80);	
550 NEXT J%	
560 ROW% = 3 + (I% MOD 20)	Set the segment data display row accordingly.
570 ELSE	This is not one of the first twenty segments, so set the segment data display row accordingly.
580 ROW% = 3 + (I% MOD 20)	
590 END IF	
600 LOCATE ROW%, 1: PRINT USING "##"; TAB(6); I%;	
610 PRINT USING "#####.##"; TAB(23); TABLE!(1, I%) / 1000000; TAB(42); TABLE!(2, I%) / 1000000;	
620 PRINT USING "####"; TAB(65); TABLE!(3, I%)	Display the segment parameters.
630 RETURN	Return from the READLIST routine.

Running the program

1. The computer reads in the frequency list table segments from the analyzer and displays the data in a table. (It is assumed that a list frequency table has already been entered into the analyzer.)

2. Enter a segment number, as prompted, to view only that segment on the analyzer.

3. Continue entering and viewing single segments. Enter 0 at the prompt to exit the loop.

4. The computer restores all the segments on the analyzer by displaying an all-segment sweep.

Using limit lines

To perform limit testing on the network analyzer over HP-IB, limits must first be loaded into the network analyzer. Then the limits can be activated and the device measured. The device's performance to the specified limits is signaled by a pass or fail message on the network analyzer display.

The following examples illustrate the use of the network analyzer to perform limit testing. Example 5A constructs a table of limit segments which is then loaded into the network analyzer's limit table. Each segment stipulates an upper limit, lower limit, limit type, and stimulus frequency. The command sequence for entering a limit table imitates the key sequence followed when entering a table from the front panel in that there is a command for every key press. Editing a limit is also the same as the key sequence, and the network analyzer automatically reorders the edited segments in order of increasing start frequency.

Example 5B performs limit testing by examining the limit/search fail bits which are set and latched when limit testing or a marker search fails. There are four bits, one for each channel for both limit testing and marker search. Their purpose is to allow the computer to determine whether the test/search just executed was successful. The sequence of their use is to clear Event Status Register B, to trigger the limit test or marker search, and then to check the appropriate fail bit.

The best ways to trigger the limit test are with a single sweep (SING) or with a set number of sweeps (NUMGn). Marker searches (max, min, target, and widths), however, are automatically triggered by reading out related marker or bandwidth values. Regardless of how the limit/search was triggered, the results can be found simply by checking the fail bit.

The limit table is carried as part of the learn string. While it cannot be modified in this form, it can easily be stored and recalled.

Example 5A: Limit line setup

The following program illustrates how to create a limit table and transmit it to the network analyzer. It takes advantage of the computer's ability to simplify creating and editing the table. The table is entered and completely edited before being transmitted to the network analyzer. For simplicity, the option of entering offsets is not given.

This program is stored on the Example Programs disk as **IPG5A.BAS**.

10	REM $INCLUDE: 'QBSETUP'	Call the QuickBASIC initialization file QBSETUP.
20	CLS	Clear the computer CRT.
30	ISC& = 7	Assign the interface select code to a variable.
40	VNA& = 716	Assign the analyzer's address to a variable.
50	CALL IOTIMEOUT(ISC&, 10!): GOSUB ERRORTRAP	Define a system time-out of ten seconds and perform error trapping.
60	CALL IOABORT(ISC&): GOSUB ERRORTRAP	Abort any HP-IB transfers and perform error trapping.
70	CALL IOCLEAR(ISC&): GOSUB ERRORTRAP	Clear the analyzer's HP-IB interface and perform error trapping.
80	CALL IOEOI(ISC&, 0): GOSUB ERRORTRAP	Disable the End-Or-Identify mode for transferring data and perform error trapping.
90	LOCATE 1, 1: INPUT "NUMBER OF LIMIT SEGMENTS? ", NUMBER%	Read in the desired number of limits from the operator.
100	DIM TABLE!(1 TO 4, 1 TO NUMBER%)	Create an array to hold the limit data (stimulus frequency value, upper limit value, lower limit value, and limit type code).
110	DIM LIMITTYPE$(1 TO NUMBER%)	Create an array to hold the limit type string.
120	CLS	Clear the computer CRT.

```
130 LOCATE 6, 1: PRINT TAB(3);
    "SEGMENT"; TAB(15);
    "STIMULUS (MHz)";
    TAB(33); "UPPER (dB)";
    TAB(49); "LOWER (dB)";
    TAB(68); "TYPE";
```

Display the limit table header on the computer CRT.

```
140 FOR I% = 1 TO NUMBER%
```

Repeat for each segment in the limit table.

```
150  GOSUB LOADLIMIT
```

Load the data for the current segment, `TABLE!(1 to 4, I%)`. Since `LOADLIMIT` is a subroutine, `I%` is used as a global variable.

```
160 NEXT I%
170 GOSUB CLEARLINES
```

Clear the CRT lines being used for data entry.

```
180 LOCATE 1, 1: INPUT "DO YOU
    WANT TO EDIT (Y/N)? ",
    ANSWER$
```

Determine if editing is initially desired.

```
190 DO UNTIL ((ANSWER$ = "N")
    OR (ANSWER$ = "n"))
```

Repeat until all editing has been done.

```
200  INPUTENTRY: LOCATE 1, 40:
     INPUT "ENTRY NUMBER? ",
     I%
```

Get the number of the segment to be edited.

```
210  IF ((I% < 1) OR (I% >
     NUMBER%)) THEN GOTO
     INPUTENTRY
```

Make sure the segment number is valid.

```
220  GOSUB LOADLIMIT
```

Re-enter the segment data.

```
230  GOSUB CLEARLINES
```

Clear the CRT lines being used for data entry.

```
240  LOCATE 1, 1: INPUT "DO YOU
     WANT TO EDIT (Y/N)? ",
     ANSWER$
```

Determine if more editing is desired.

```
250 LOOP
260 A$ = "EDITLIML; CLEL;":
    GOSUB IOOUTS
```

To begin sending the table to the analyzer, open the analyzer's limit line table for editing, and delete any existing segments.

```
270 FOR I% = 1 TO NUMBER%
```

Loop for each segment.

```
280  A$ = "SADD; LIMS" +
     STR$(TABLE!(1, I%)) +
     "MHZ;": GOSUB IOOUTS
290  A$ = "LIMU" +
     STR$(TABLE!(2, I%)) +
     "DB;": GOSUB IOOUTS
300  A$ = "LIML" +
     STR$(TABLE!(3, I%)) +
     "DB;": GOSUB IOOUTS
310  A$ = "LIMT" +
     LIMITTYPE$(I%) + ";":
     GOSUB IOOUTS
320  A$ = "SDON;": GOSUB
     IOOUTS
```

Add a segment, specifying its stimulus frequency value, upper limit value, lower limit value, and limit type. Then declare the current limit line segment done.

```
330 NEXT I%
340 A$ = "EDITDONE;
    LIMILINEON;
    LIMITESTON;": GOSUB
    IOOUTS
```

Close the edit limit line table, display the limit lines on the analyzer, and activate limit testing.

350 CALL IOLOCAL(ISC&): GOSUB ERRORTRAP	Return the network analyzer to local mode and perform error trapping.
360 END	End program execution.
370 ERRORTRAP:	Define a routine to trap errors.
380 IF PCIB.ERR <> NOERR THEN ERROR PCIB.BASERR	Perform error trapping.
390 RETURN	Return from the ERRORTRAP routine.
400 IOOUTS:	Define a routine to send a command string from the computer to the analyzer.
410 CALL IOOUTPUTS(VNA&, A$, LEN(A$)): GOSUB ERRORTRAP	Send the command string A$ out to the analyzer and perform error trapping.
420 RETURN	Return from the IOOUTS routine.
430 LOADLIMIT:	Define a routine to read in all of one segment's data from the operator's input and load it into the data table on the computer.
440 GOSUB CLEARLINES	Clear the CRT lines being used for data entry.
450 LOCATE 1, 1: PRINT "SEGMENT: "; STR$(I%);	
460 LOCATE 2, 1: PRINT "STIMULUS VALUE (MHz)?";	
470 LOCATE 3, 1: PRINT "UPPER LIMIT VALUE (dB)?";	
480 LOCATE 4, 1: PRINT "LOWER LIMIT VALUE (dB)?";	
490 LOCATE 1, 40: PRINT "LIMIT TYPE (1,2,3)?";	
500 LOCATE 2, 42: PRINT "1 = FLAT";	
510 LOCATE 3, 42: PRINT "2 = SLOPED";	
520 LOCATE 4, 42: PRINT "3 = SINGLE POINT";	Display the input labels.
530 IF ((TABLE!(1, I%) <> 0) OR (TABLE!(2, I%) <> 0) OR (TABLE!(3, I%) <> 0) OR (TABLE!(4, I%) <> 0)) THEN	If the segment contains valid data, display it at the entry locations.
540 LOCATE 2, 22: PRINT TABLE!(1, I%);	
550 LOCATE 3, 25: PRINT TABLE!(2, I%);	
560 LOCATE 4, 25: PRINT TABLE!(3, I%);	
570 LOCATE 1, 59: PRINT TABLE!(4, I%);	
580 END IF	
590 SAVE! = TABLE!(1, I%)	Save the stimulus frequency value of the current table entry.
600 LOCATE 2, 21: INPUT TABLE!(1, I%)	Read the stimulus frequency value of the segment.

610 IF TABLE!(1, I%) = 0 THEN TABLE!(1, I%) = SAVE!	If no value or 0 was entered, return the stimulus frequency to its previous value.
620 LOCATE 2, 22: PRINT SPACE$(17)	
630 LOCATE 2, 22: PRINT TABLE!(1, I%);	Display the new stimulus frequency.
640 SAVE! = TABLE!(2, I%)	Save the upper limit value of the current table entry.
650 LOCATE 3, 23: INPUT TABLE!(2, I%)	Read the upper limit value of the segment.
660 IF TABLE!(2, I%) = 0 THEN TABLE!(2, I%) = SAVE!	If no value or 0 was entered, return the upper limit to its previous value.
670 LOCATE 3, 24: PRINT SPACE$(15): LOCATE 3, 25: PRINT TABLE!(2, I%);	Display the new upper limit.
680 SAVE! = TABLE!(3, I%)	Save the lower limit value of the current table entry.
690 LOCATE 4, 23: INPUT TABLE!(3, I%)	Read the lower limit value of the segment.
700 IF TABLE!(3, I%) = 0 THEN TABLE!(3, I%) = SAVE!	If no value or 0 was entered, return the lower limit to its previous value.
710 LOCATE 4, 24: PRINT SPACE$(15): LOCATE 4, 25: PRINT TABLE!(3, I%)	Display the new lower limit.
720 SAVE! = TABLE!(4, I%)	Save the limit type integer code of the current table entry.
730 TABLE!(4, I%) = 0	Set TABLE!(4, I%) for entry into the DO UNTIL loop.
740 DO UNTIL ((TABLE!(4, I%) > 0) AND (TABLE!(4, I%) < 4))	Repeat until a valid limit type integer code has been entered.
750 LOCATE 1, 58: INPUT TABLE!(4, I%)	Read the limit type integer code of the segment.
760 IF (TABLE!(4, I%) = 0) THEN TABLE!(4, I%) = SAVE!	If no value or 0 was entered and the previous value was valid, return the limit type integer code to that previous value.
770 LOOP	
780 LOCATE 1, 59: PRINT SPACE$(28): LOCATE 1, 59: PRINT TABLE!(4, I%)	Display the new limit type integer code.
790 LOCATE I% + 6, 1: PRINT SPACE$(80): LOCATE I% + 6, 1: PRINT TAB(5); I%; TAB(19); TABLE!(1, I%); TAB(36); TABLE!(2, I%); TAB(52); TABLE!(3, I%); TAB(68);	Display the new data in the table.
800 SELECT CASE TABLE!(4, I%)	Display the limit type corresponding to the limit type integer code in the table. Set the current LIMITTYPE$ entry to the proper two-character code for transmission to the network analyzer.
CASE 1	A limit type integer code of 1 indicates "FLAT LINE".
810 PRINT "FLAT";	

```
820  LIMITTYPE$(I%) = "FL"
     CASE 2                       A limit type integer code of 2 indicates
                                  "SLOPING LINE".
830  PRINT "SLOPED";
840  LIMITTYPE$(I%) = "SL"
     CASE 3                       A limit type integer code of 3 indicates "SINGLE
                                  POINT".
850  PRINT "SINGLE POINT";
860  LIMITTYPE$(I%) = "SP"
870 END SELECT
880 RETURN                        Return from the LOADLIMIT routine.
890 CLEARLINES:                   Define a routine to clear the CRT lines used for
                                  data entry.
900 FOR J% = 1 TO 4               Clear each line.
910  LOCATE J%, 1: PRINT
     SPACE$(80)
920 NEXT J%
930 RETURN                        Return from the CLEARLINES routine.
```

Running the program

1. The computer clears the analyzer's limit line table. If this is not desired, remove the CLEL command from line 90.

2. Enter the number of segments and then the parameters of each segment as prompted.

3. Edit the computer's limit line table until it is satisfactory. Pressing <ENTER> at a prompt during editing leaves the parameter at its current value.

4. The computer sends the completed limit line table out to the analyzer, which orders the segments, activates limit testing, and displays the limit lines.

Example 5B: PASS/FAIL tests

The following program illustrates how to perform limit testing using the limit/search fail bits in Event Status Register B. The requirement that several sweeps in a row must pass is used in order to ensure that the limit test pass was not extraneous due to the device settling or the operator tuning during the sweep.

The program assumes that an appropriate calibration has been performed, that limit lines have been defined, and that limit testing is on prior to running the program.

This program is stored on the Example Programs disk as **IPG5B.BAS**.

10 REM $INCLUDE: 'QBSETUP'	Call the QuickBASIC initialization file QBSETUP.
20 CLS	Clear the computer CRT.
30 ISC& = 7	Assign the interface select code to a variable.
40 VNA& = 716	Assign the analyzer's address to a variable.
50 CALL IOTIMEOUT(ISC&, 10!): GOSUB ERRORTRAP	Define a system time-out of ten seconds and perform error trapping.
60 CALL IOABORT(ISC&): GOSUB ERRORTRAP	Abort any HP-IB transfers and perform error trapping.
70 CALL IOCLEAR(ISC&): GOSUB ERRORTRAP	Clear the analyzer's HP-IB interface and perform error trapping.
80 CALL IOEOI(ISC&, 0): GOSUB ERRORTRAP	Disable the End-Or-Identify mode for transferring data and perform error trapping.
90 INPUT "NUMBER OF CONSECUTIVE PASSED SWEEPS FOR QUALIFICATION? ", QUAL%	Enter the number of sweeps that must pass before the device is considered to have passed the limit test.
100 STARTTEST: PASSES% = 0	Initialize the counter holding the number of sweeps that have passed the limit test.
110 CLS : PRINT "TUNE DEVICE"	Display instructions on the computer CRT.
120 CONTINUE: A$ = "OPC?;SING;": GOSUB IOOUTS	Sweep once and thus perform a limit test.
130 CALL IOENTER(VNA&, REPLY!): GOSUB ERRORTRAP	Wait for the end of the sweep.
140 A$ = "ESB?;": GOSUB IOOUTS	Request the Event Status Register B value from the analyzer.
150 CALL IOENTER(VNA&, ESTAT!): GOSUB ERRORTRAP	Receive the Event Status Register B value from the analyzer in order to check the fail bit.
160 IF ((ESTAT! MOD 32) > 15) THEN	Check if bit 4, the channel 1 limit fail bit, is set, indicating that the device failed the current sweep.
170 IF (PASSES% <> 0) THEN SOUND 300, 5	If sweeps have been passing, audibly warn the operator that the device is now failing.
180 GOTO STARTTEST	Restart the test sequence.
190 END IF	
200 SOUND 1000, 1	Indicate audibly that the device passed the current sweep.
210 PASSES% = PASSES% + 1	Increment the sweeps passed counter.
220 IF PASSES% = 1 THEN	The device just passed its first sweep, encourage the operator to stop tuning the device.
230 CLS : PRINT "STOP TUNING"	
240 END IF	

250 IF PASSES% < QUAL% THEN GOTO CONTINUE	Loop until enough consecutive sweeps have passed that the device is considered to have passed the limit test.
260 CLS : PRINT "DEVICE PASSED"	Display program progress on the computer CRT.
270 FOR INDEX% = 1 TO 5	Indicate audibly that the device has passed the limit test.
280 SOUND 500, 1	
290 SOUND 1000, 1	
300 NEXT INDEX%	
310 SOUND 2000, 1	
320 PRINT "PRESS <ENTER> TO TEST NEXT DEVICE, <ESC> TO END."	Display instructions on the computer CRT.
330 CHAR$ = CHR$(0)	Initialize CHAR$ for entry into the DO UNTIL loop.
340 DO UNTIL ((CHAR$ = CHR$(13)) OR (CHAR$ = CHR$(27)))	Wait until a valid key (<ENTER> or <ESC>) is pressed.
350 CHAR$ = INKEY$	
360 LOOP	
370 IF (CHAR$ = CHR$(13)) THEN	If <ENTER> was pressed, return to the beginning of the test cycle to test the next device.
380 GOTO STARTTEST	
390 END IF	
400 CALL IOLOCAL(ISC&): GOSUB ERRORTRAP	Return the network analyzer to local mode and perform error trapping.
410 END	End program execution.
420 ERRORTRAP:	Define a routine to trap errors.
430 IF PCIB.ERR <> NOERR THEN ERROR PCIB.BASERR	Perform error trapping.
440 RETURN	Return from the ERRORTRAP routine.
450 IOOUTS:	Define a routine to send a command string from the computer to the analyzer.
460 CALL IOOUTPUTS(VNA&, A$, LEN(A$)): GOSUB ERRORTRAP	Send the command string A$ out to the analyzer and perform error trapping.
470 RETURN	Return from the IOOUTS routine.

Running the program

1. Set up a limit table on channel 1 for a specific device either manually or using Example 5A: *Limit line setup*.

2. Run the program. Specify the number of sweeps that must pass for qualification. For very slow sweeps, as few as two sweeps is appropriate. For very fast sweeps, as many as six or more sweeps may be needed.

3. Connect the filter. The computer beeps to indicate the test status.

4. When enough consecutive sweeps pass, the computer warbles and requests a new device.

Storing/recalling instrument states

It is possible to store and recall entire instrument states over HP-IB using the commands to read the learn string and the calibration data out of the analyzer. The learn string is up to 3000 bytes long and is in form 1, instrument internal binary. It includes all front panel settings, the list frequency table, and the limit table for each channel. It is read out with `OUTPLEAS` and sent back with `INPULEAS`.

Although the learn string contains the identity of the current active calibration, it does not contain the calibration data. Therefore, in order to get the entire instrument state, it is necessary to read out the learn string and the calibration data. This calibration data is stored inside the network analyzer in up to twelve calibration coefficient arrays. Each array is a specific error coefficient and is stored and transmitted as a data array of which each point is specified as a real/imaginary pair of real numbers. The number of points in the array is the same as the number of points in the sweep. For more information about which calibration coefficients correspond to which calibration types, see the section entitled *Calibration Arrays* in the *HP-IB Quick Reference*.

The computer can read out the error coefficient arrays using the commands `OUTPCALC01`, `OUTPCALC02`, ... `OUTPCALC12`. Each calibration type uses only as many arrays as are needed, starting with array 1. Hence, it is necessary to know the calibration type and therefore the number of arrays before trying to read them out. Although the calibration type is in the learn string, it is difficult to extract. Instead, it can be determined if a calibration type is active by sending the mnemonic of the type in question followed by a question mark (`CALIRESP?`). The analyzer will then respond with `1` if that type is active and `0` if it is not.

Calibration data can also be sent from the computer to the analyzer. The calibration type mnemonic must be sent first to prepare the analyzer. Then the calibration coefficient arrays can be transferred using the `INPUCALCnn` commands. Once all the coefficients are in the analyzer, the command sequence `SAVC; CONT` will create a calibration set and put the analyzer in continuous sweep trigger mode, thereby activating the calibration.

Example 6A: Learn string

The following program makes use of the learn string to transfer the instrument state between the analyzer and the computer. It demonstrates the use of the commands `OUTPLEAS` and `INPULEAS`. Note that character matching must be disabled by calling the HP-IB Command Library routine `IOMATCH` before the learn string is read in by the routine `IOENTERS`. This prevents the computer from terminating on a linefeed when the string is read because the learn string may contain linefeeds as part of its information.

This example program is stored on the Example Programs disk as **IPG6A.BAS**.

`10 REM $INCLUDE: 'QBSETUP'`	Call the QuickBASIC initialization file `QBSETUP`.
`20 CLS`	Clear the computer CRT.
`30 ISC& = 7`	Assign the interface select code to a variable.
`40 VNA& = 716`	Assign the analyzer's address to a variable.
`50 CALL IOTIMEOUT(ISC&, 10!): GOSUB ERRORTRAP`	Define a system time-out of ten seconds and perform error trapping.
`60 CALL IOABORT(ISC&): GOSUB ERRORTRAP`	Abort any HP-IB transfers and perform error trapping.
`70 CALL IOCLEAR(ISC&): GOSUB ERRORTRAP`	Clear the analyzer's HP-IB interface and perform error trapping.
`80 MATCH$ = CHR$(10)`	Define the match character as the linefeed.
`90 ENABLE% = 1: DISABLE% = 0`	Initialize flag values to enable and disable character matching.
`100 CALL IOMATCH(ISC&, MATCH$, DISABLE%): GOSUB ERRORTRAP`	Disable character matching for the current match character, the linefeed. This prevents termination on a linefeed when a string is read since the linefeed could actually be part of the learn string information.

`110 A$ = "OUTPLEAS;": GOSUB IOOUTS`	Request the learn string from the analyzer.
`120 MAX% = 3000`	Set the maximum number of characters to read in.
`130 LEARNSTRING$ = SPACE$(MAX%)`	Set aside space to receive the learn string.
`140 ACTUAL% = 0`	Initialize the actual number of characters read in. This variable is given a value by `IOENTERS`.
`150 CALL IOENTERS(VNA&, LEARNSTRING$, MAX%, ACTUAL%): GOSUB ERRORTRAP`	Receive the learn string from the analyzer.
`160 LEARNSTRING$ = LEFT$(LEARNSTRING$, ACTUAL%)`	Redefine the learn string to contain only the information read in from the analyzer.
`170 CALL IOMATCH(ISC&, MATCH$, ENABLE%): GOSUB ERRORTRAP`	Enable character matching. This results in termination on a linefeed when a string is read.
`180 CALL IOLOCAL(ISC&): GOSUB ERRORTRAP`	Put the analyzer in local mode.
`190 PRINT "CHANGE STATE AND PRESS <ENTER>"`	
`200 DO UNTIL INKEY$ = CHR$(13): LOOP`	Allow the operator to connect a new analyzer or to modify the state of the present analyzer from the front panel.
`210 A$ = "INPULEAS" + LEARNSTRING$ + ";": GOSUB IOOUTS`	Restore the state defined in the learn string to the analyzer.
`220 PRINT "INITIAL INSTRUMENT STATE RESTORED."`	Display program progress on the computer CRT.
`230 CALL IOLOCAL(ISC&): GOSUB ERRORTRAP`	Return the network analyzer to local mode and perform error trapping.
`240 END`	End program execution.
`250 ERRORTRAP:`	Define a routine to trap errors.
`260 IF PCIB.ERR <> NOERR THEN ERROR PCIB.BASERR`	Perform error trapping.
`270 RETURN`	Return from the `ERRORTRAP` routine.
`280 IOOUTS:`	Define a routine to send a command string from the computer to the analyzer.
`290 CALL IOOUTPUTS(VNA&, A$, LEN(A$)): GOSUB ERRORTRAP`	Send the command string `A$` out to the analyzer and perform error trapping.
`300 RETURN`	Return from the `IOOUTS` routine.

Running the program

1. The computer reads the learn string in from the analyzer, thereby storing its state.

2. Change the state of the analyzer from its front panel as prompted.

3. The computer sends the learn string back to the analyzer, thereby restoring it to its original state.

Example 6B: Reading calibration data

The following program illustrates how to determine which calibration is active, how to read measurement calibration data out of the network analyzer, and how to put it back into the instrument.

The two-dimensional calibration coefficient arrays are transferred in *form 5,* PC-DOS 32-bit floating point format. They are stored in one three-dimensional array from which they can be examined, modified, stored, and put back into the instrument. If the data is only to be stored and put back, it is most efficient to read it in *form 1,* instrument internal binary format.

This example program is stored on the Example Programs disk as **IPG6B.BAS**.

10 REM $INCLUDE: 'QBSETUP'	Call the QuickBASIC initialization file QBSETUP.
20 CLS	Clear the computer CRT.
30 ISC& = 7	Assign the interface select code to a variable.
40 VNA& = 716	Assign the analyzer's address to a variable.
50 CALL IOTIMEOUT(ISC&, 10!): GOSUB ERRORTRAP	Define a system time-out and perform error trapping.
60 CALL IOABORT(ISC&): GOSUB ERRORTRAP	Abort any HP-IB transfers and perform error trapping.
70 CALL IOCLEAR(ISC&): GOSUB ERRORTRAP	Clear the analyzer's HP-IB interface and perform error trapping.
80 CALL IOEOI(ISC&, 0): GOSUB ERRORTRAP	Disable the End-Or-Identify mode for transferring data and perform error trapping.
90 DIM CALTYPE$(1 TO 6), NUMBER%(1 TO 6)	Set up parallel arrays of possible calibrations and the number of arrays associated with each calibration.
100 CALTYPE$(1) = "CALIRESP": NUMBER%(1) = 1	
110 CALTYPE$(2) = "CALIRAI": NUMBER%(2) = 2	
120 CALTYPE$(3) = "CALIS111": NUMBER%(3) = 3	
130 CALTYPE$(4) = "CALIS221": NUMBER%(4) = 3	
140 CALTYPE$(5) = "CALIFUL2": NUMBER%(5) = 12	
150 CALTYPE$(6) = "NOOP": NUMBER%(6) = 0	
160 LOCATE 5, 25: PRINT "CALIBRATION NUMBER OF"	
170 LOCATE 6, 25: PRINT " TYPE ARRAYS"	Display the calibration table heading.
180 FOR I% = 1 TO 6	Display a table of possible calibrations on the computer CRT.
190 LOCATE 7 + I%, 18: PRINT USING "#"; I%;	
200 PRINT "."; TAB(27); CALTYPE$(I%); TAB(45);	
210 PRINT USING "##"; NUMBER%(I%)	
220 NEXT I%	
230 ACTIVE! = 0	Initialize ACTIVE! for entry into the DO UNTIL loop.

240 DO UNTIL ACTIVE!	Repeat until the active calibration type is selected by the user.
250 INDEX% = 0	Initialize INDEX% for entry into the DO UNTIL loop.
260 DO UNTIL ((INDEX% > 0) AND (INDEX% < 7))	Get a valid calibration type selection from the user.
270 LOCATE 15, 25: INPUT "ENTER SELECTION: ", INDEX%	
280 LOOP	
290 IF (NUMBER%(INDEX%) = 0) THEN	If no calibration was active, clear the computer CRT and go to the end of the program.
300 CLS : GOTO FINISH	
310 END IF	
320 A$ = CALTYPE$(INDEX%) + "?;": GOSUB IOOUTS	Ask the network analyzer if the user-chosen calibration is active.
330 CALL IOENTER(VNA&, ACTIVE!): GOSUB ERRORTRAP	Get the response from the analyzer.
340 LOOP	
350 CLS	Clear the computer CRT.
360 PRINT "CALIBRATION TYPE: "; CALTYPE$(INDEX%)	Confirm that the analyzer's active calibration has been found by displaying it and its corresponding number of arrays on the computer CRT.
370 PRINT "NUMBER OF ARRAYS: "; NUMBER%(INDEX%)	
380 A$ = "FORM5; POIN?;": GOSUB IOOUTS	Set data to be transferred in form 5, PC-DOS floating point and request the number of points from the analyzer.
390 CALL IOENTER(VNA&, POINTS!): GOSUB ERRORTRAP	Receive the number of points from the analyzer.
400 POINTS% = INT(POINTS!)	Convert the number of points to an integer.
410 DIM CAL!(1 TO 2, 0 TO POINTS%, 1 TO NUMBER%(INDEX%))	Allocate space for a three-dimensional array to hold all the calibration coefficients. Think of CAL! as a data structure with a two-dimensional array for each of the calibration type's corresponding arrays. These two-dimensional arrays are read in one at a time, and each is preceded by a four-byte header. Space is allocated for these headers by extending CAL!'s second dimension by one and thus adding two real numbers (eight bytes) to the beginning of each two-dimensional array.
420 DIM DIGIT$(1 TO NUMBER%(INDEX%))	Dimension an array to hold two-digit integers from 1 to the number of arrays, each integer with a leading zero if necessary. These are used with OUTPCALC and INPUCALC commands.
430 LOCATE 1, 41: PRINT "ARRAYS RECEIVED: "	Display a heading for program progress information.
440 MAX% = 4 * 2 * POINTS% + 4	The maximum number of bytes to read in for each two-dimensional array is two four-byte numbers per point with POINTS% points plus a four-byte header.
450 FLAG% = 1	Set FLAG% for no swapping of bytes.
460 FOR I% = 1 TO NUMBER%(INDEX%)	Read in each of the two-dimensional arrays making up CAL! one at a time.

470	ACTUAL% = 0	Initialize or re-initialize the actual number of bytes read in.
480	DIGIT$(I%) = STR$(I%)	Create the current two-digit number string corresponding to I%.
490	IF (LEN(DIGIT$(I%)) = 2) THEN	Since strings corresponding to positive numbers are preceded by a space, one-digit numbers are two characters long. These must be converted to 0 followed by the one digit in order to be the required two digits long.
500	DIGIT$(I%) = "0" + RIGHT$(DIGIT$(I%), 1)	
510	ELSE	
520	DIGIT$(I%) = RIGHT$(DIGIT$(I%), 2)	The number is already two digits long, so simply remove the preceding space.
530	END IF	
540	A$ = "OUTPCALC" + DIGIT$(I%) + ";": GOSUB IOOUTS	Request the current two-dimensional calibration coefficient array from the analyzer.
550	CALL IOENTERB(VNA&, SEG CAL!(2, 0, I%), MAX%, ACTUAL%, FLAG%): GOSUB ERRORTRAP	Read in the current two-dimensional array, specifying the beginning array address as one real number (four bytes) before the desired destination of the true data in order to read in the header.
560	LOCATE 1, 60: PRINT I%	Display program progress on the computer CRT.
570	NEXT I%	
580	LOCATE 4, 1: PRINT "PRESS <ENTER> TO RE-TRANSMIT CALIBRATION."	Display instructions on the computer CRT.
590	DO UNTIL INKEY$ = CHR$(13): LOOP	Wait for the operator to continue.
600	LOCATE 4, 1: PRINT SPACE$(80)	Clear the instruction display line on the computer CRT.
610	A$ = CALTYPE$(INDEX%) + ";": GOSUB IOOUTS	Prepare the analyzer to receive the correct calibration type from the computer.
620	LOCATE 2, 41: PRINT "ARRAYS TRANSMITTED: ";	Display a heading for program progress information.
630	FOR I% = 1 TO NUMBER%(INDEX%)	Send out each of the two-dimensional arrays making up CAL! separately.
640	A$ = "INPUCALC" + DIGIT$(I%) + ";": GOSUB IOOUTS	Prepare the analyzer to receive the current two-dimensional calibration coefficient array.
650	CALL IOOUTPUTB(VNA&, SEG CAL!(2, 0, I%), MAX%, FLAG%)	Send the current two-dimensional calibration coefficient array to the analyzer.
660	LOCATE 2, 60: PRINT I%	Display program progress on the computer CRT.
670	NEXT I%	
680	A$ = "SAVC;": GOSUB IOOUTS	Create a cal set using the current calibration data.
690	A$ = "CONT;": GOSUB IOOUTS	Trigger a sweep so that the calibration becomes active.
700	FINISH: LOCATE 4, 1: PRINT "DONE"	Display program progress on the computer CRT.

`710 CALL IOLOCAL(ISC&): GOSUB ERRORTRAP`	Return the network analyzer to local mode and perform error trapping.
`720 END`	End program execution.
`730 ERRORTRAP:`	Define a routine to trap errors.
`740 IF PCIB.ERR <> NOERR THEN ERROR PCIB.BASERR`	Perform error trapping.
`750 RETURN`	Return from the `ERRORTRAP` routine.
`760 IOOUTS:`	Define a routine to send a command string from the computer to the analyzer.
`770 CALL IOOUTPUTS(VNA&, A$, LEN(A$)): GOSUB ERRORTRAP`	Send the command string `A$` out to the analyzer and perform error trapping.
`780 RETURN`	Return from the `IOOUTS` routine.

Running the program

1. When the computer displays the calibration type table, enter the number corresponding to the active calibration on the analyzer. Before continuing, the computer ensures that the correct type was chosen by questioning the analyzer.

2. The computer reads the up to twelve calibration coefficient arrays from the network analyzer one at a time into one three-dimensional array.

3. Press <ENTER> on the computer CRT as prompted.

4. The computer sends the up to twelve calibration coefficient arrays back to the network analyzer one at a time.

Miscellaneous Programming Examples

Example 7: Interrupt generation

The following program illustrates how to use the HP-IB Command Library routine `IOPEN` and QuickBASIC's `PEN` statements to generate interrupts. A call to `IOPEN`:

```
CALL IOPEN(ISC&, 0): GOSUB ERRORTRAP
```

will enable a Service Request (SRQ) to generate an interrupt that can be detected by QuickBASIC's `PEN` statements. Through these statements, QuickBASIC has the ability to enable (`PEN ON`) and disable (`PEN OFF`) HP-IB interrupts and execute an interrupt handling routine every time one occurs (`ON PEN GOSUB xxxx`).

In order for the analyzer to generate an SRQ when a specific event occurs, both the desired Event Status Register bit and the desired status byte bit must be enabled. The status reporting system can be set up using HP-IB commands, and it must be reset every time the status is cleared (`CLES`). For example, `ESE 64; SRE 32` enables the User Request bit (6; 64 = 2^6) of the Event Status Register and the Event Status Register summary bit (5; 32 = 2^5) of the status byte (refer ahead to Figure A.1 on page 65). This means that when the User Request bit is set, the Event Status Register summary bit in the status byte is set. Likewise when the Event Status Register summary bit in the status byte is set, an SRQ is generated. With this status reporting system, a key press will generate an SRQ. By then using the above described `PEN` statements, an SRQ can be made to generate an interrupt, which will cause a special interrupt handling routine to be executed.

The following program uses the HP-IB command `WRSKn` to re-label the softkeys. The interrupt generation system is then set up so that when a key is pressed, the computer processes the generated interrupt by identifying which key was pressed. If full use of this method is made, an automatic system would no longer require a computer keyboard and would instead be as easy to use as a manual instrument.

This example program is stored on the Example Programs disk as **IPG7.BAS**.

`10 REM $INCLUDE: 'QBSETUP'`	Call the QuickBASIC initialization file `QBSETUP`.
`20 CLS`	Clear the computer CRT.
`30 ISC& = 7`	Assign the interface select code to a variable.
`40 VNA& = 716`	Assign the analyzer's address to a variable.
`50 CALL IOTIMEOUT(ISC&, 10!): GOSUB ERRORTRAP`	Define a system time-out of ten seconds and perform error trapping.
`60 CALL IOABORT(ISC&): GOSUB ERRORTRAP`	Abort any HP-IB transfers and perform error trapping.
`70 CALL IOCLEAR(ISC&): GOSUB ERRORTRAP`	Clear the analyzer's HP-IB interface and perform error trapping.
`80 CALL IOEOI(ISC&, 0): GOSUB ERRORTRAP`	Disable the End-Or-Identify mode for transferring data and perform error trapping.
`90 A$ = "PRES;": GOSUB IOOUTS`	Preset the network analyzer.
`100 A$ = "CLES; ESE64; SRE32;": GOSUB IOOUTS`	Clear the status byte and set the status reporting system to the following: 1) Bit 6, User Request, of the Event Status Register is summarized by bit 5 of the status byte. This allows a key press to be detected by a serial poll. 2) Bit 5 of the status byte, the Event Status Register, is enabled. This allows the Event Status Register to generate service requests.
`110 A$ = "MENUMRKF;": GOSUB IOOUTS`	Activate a menu that uses all of the softkeys in order to ensure that each softkey is active and may be written to.
`120 A$ = "MENUOFF;": GOSUB IOOUTS`	Turn the built-in softkey menu off so that the softkeys may be labeled by the computer.

```
130 A$ = "WRSK1 " + CHR$(34) +
    "CAL #1" + CHR$(34) +
    ";": GOSUB IOOUTS
```

Label the softkeys. The label must be preceded and followed by double quotes. To put double quotes within a string in QuickBASIC, use CHR$(34).

```
140 A$ = "WRSK2 " + CHR$(34) +
    "TEST #1" + CHR$(34) +
    ";": GOSUB IOOUTS
150 A$ = "WRSK3 " + CHR$(34) +
    "CAL #2" + CHR$(34) +
    ";": GOSUB IOOUTS
160 A$ = "WRSK4 " + CHR$(34) +
    "TEST #2" + CHR$(34) +
    ";": GOSUB IOOUTS
170 A$ = "WRSK8 " + CHR$(34) +
    "ABORT" + CHR$(34) + ";":
    GOSUB IOOUTS
```

```
180 PRINT "SOFTKEYS LOADED"
```

Display program progress on the computer CRT.

```
190 PEN OFF
```

Disable HP-IB interrupts.

```
200 ON PEN GOSUB GETSRQ
```

Set up the interrupt system so that whenever an HP-IB interrupt occurs, a routine that gets a service request will be executed.

```
210 PEN ON
```

Enable HP-IB interrupts.

```
220 CALL IOPEN(ISC&, 0): GOSUB
    ERRORTRAP
```

Let an SRQ generate an interrupt.

```
230 WAITSRQ:
```

Continue to let key presses generate interrupts until the eighth softkey, labeled <ABORT>, is pressed.

```
240 IF KEYCODE% <> 10 THEN
    GOTO WAITSRQ
```

```
250 PEN OFF
```

Disable HP-IB interrupts.

```
260 A$ = "MENUON;": GOSUB
    IOOUTS
```

Turn the softkey menu back on.

```
270 CALL IOLOCAL(ISC&): GOSUB
    ERRORTRAP
```

Return the network analyzer to local mode and perform error trapping.

```
280 END
```

End program execution.

```
290 ERRORTRAP:
```

Define a routine to trap errors.

```
300 IF PCIB.ERR <> NOERR THEN
    ERROR PCIB.BASERR
```

Perform error trapping.

```
310 RETURN
```

Return from the ERRORTRAP routine.

```
320 IOOUTS:
```

Define a routine to send a command string from the computer to the analyzer.

```
330 CALL IOOUTPUTS(VNA&, A$,
    LEN(A$)): GOSUB ERRORTRAP
```

Send the command string A$ out to the analyzer and perform error trapping.

```
340 RETURN
```

Return from the IOOUTS routine.

```
350 GETSRQ:
```

Define a routine to get a service request.

```
360 CALL IOSPOLL(VNA&, STAT%):
    GOSUB ERRORTRAP
```

Perform a serial poll to read in the status byte and thereby clear it.

```
370 A$ = "CLES; ESE64;
    SRE32;": GOSUB IOOUTS
```

Ensure that the status byte was cleared and that the proper status reporting system is in operation.

```
380 A$ = "OUTPKEY;": GOSUB
    IOOUTS
```

Request the code of the last analyzer key pressed from the analyzer.

```
390 CALL IOENTER(VNA&,
    KEYCODE!): GOSUB ERRORTRAP
```

Receive the key code from the analyzer.

```
400 KEYCODE% = INT(KEYCODE!)            Convert the key code to an integer.
410 SELECT CASE KEYCODE%:
    CASE 60                              The first softkey is labeled CAL #1.
420  CLS : LOCATE 1, 1: PRINT
     "CALIBRATION #1"
     CASE 61                             The second softkey is labeled TEST #1.
430  CLS : LOCATE 1, 1: PRINT
     "TEST #1"
    CASE 56                              The third softkey is labeled CAL #2.
440  CLS : LOCATE 1, 1: PRINT
     "CALIBRATION #2"
    CASE 59                              The fourth softkey is labeled TEST #2.
450  CLS : LOCATE 1, 1: PRINT
     "TEST #2"
    CASE 10                              The eighth softkey is labeled ABORT.
460  CLS : LOCATE 1, 1: PRINT
     "ABORT"
    CASE ELSE                            No other keys are defined.
470  CLS : LOCATE 1, 1: PRINT
     "***UNDEFINED***"
480 END SELECT
490 RETURN                               Return from the GETSRQ routine.
```

Running the program

1. The computer presets the network analyzer, relabels the softkeys, and sets up the desired network analyzer status reporting and interrupt generation systems.

2. When a key is pressed, an interrupt is generated and the interrupt handling routine, which displays the identity of the key pressed on the computer, is executed.

3. Press the network analyzer softkey labeled ABORT to end the program.

Example 8: User interface

The following example program illustrates how to create a custom user interface involving only the front panel keys and the display of the network analyzer. Graphics can be drawn by sending HP-GL (Hewlett-Packard Graphics Language) commands to the network analyzer display. See the section entitled *Display Graphics* in the *HP-IB Quick Reference* for a list of accepted HP-GL commands and their functions.

It is possible to customize a user interface by taking over the network analyzer's front panel keys. The User Request bit in the Event Status Register is set whenever a front panel key is pressed or the knob is turned regardless of the current mode (local or remote) of the analyzer. Each key has its own number, as shown in Figure E.4, *Front Panel Keycodes*, of the *HP-IB Quick Reference*. The number of the key last pressed can be read with OUTPKEY? or KOR?. With OUTPKEY?, a knob turn is always reported as negative one. With KOR?, a knob turn is reported as a negative number encoded with the number of counts turned. There are 120 counts per knob rotation. Clockwise rotations are reported as numbers from −1 to −64, −1 being a very small rotation. Counter-clockwise rotations are reported as numbers from −32767 to −32701, −32767 being a very small rotation. Hence, clockwise rotations do not need any decoding at all; counter-clockwise rotations can be decoded by adding 32768.

This example uses the knob and the up and down keys on the network analyzer to adjust the size and position of a grid on the display. Pressing **[ENTRY OFF]** on the network analyzer selects the current size or position and continues the program.

This example program is stored on the Example Programs disk as **IPG8.BAS**.

`10 REM $INCLUDE: 'QBSETUP'`	Call the QuickBASIC initialization file QBSETUP.
`20 CLS`	Clear the computer CRT.
`30 ISC& = 7`	Assign the interface select code to a variable.
`40 VNA& = 716`	Assign the analyzer's address to a variable.
`50 DISPLAY& = 717`	Assign the analyzer display's address to a variable.
`60 CALL IOTIMEOUT(ISC&, 10!): GOSUB ERRORTRAP`	Define a system time-out of ten seconds and perform error trapping.
`70 CALL IOABORT(ISC&): GOSUB ERRORTRAP`	Abort any HP-IB transfers and perform error trapping.
`80 CALL IOCLEAR(ISC&): GOSUB ERRORTRAP`	Clear the analyzer's HP-IB interface and perform error trapping.
`90 CALL IOEOI(ISC&, 0): GOSUB ERRORTRAP`	Disable the End-Or-Identify mode for transferring data and perform error trapping.
`100 ADDRESS& = VNA&: A$ = "AUTO; CLES; ESE64; POIN?;": GOSUB IOOUTS`	Prepare the analyzer by scaling the trace for plotting, clearing the status byte, and setting up the status reporting system so that bit 6, User Request, of the Event Status Register is summarized by bit 5 of the status byte (allowing a key press to be detected by a serial poll). Then request the number of points from the analyzer.
`110 CALL IOENTER(VNA&, POINTS!): GOSUB ERRORTRAP`	Receive the number of points from the analyzer.
`120 POINTS% = INT(POINTS!)`	Convert the number of points to an integer.
`130 DIM DAT!(1 TO 2, 0 TO POINTS%)`	Prepare an array to receive the data.
`140 ADDRESS& = VNA&: A$ = "SING; FORM2; OUTPFORM;": GOSUB IOOUTS`	Sweep once and then hold. Tell the analyzer to send out formatted data in form 2, IEEE 32-bit floating point.

`150 MAX% = POINTS% * 2 * 4 + 4`	The maximum number of bytes to be read in is two 4-byte real numbers per point with POINTS% points plus the four-byte (two-integer) header.
`160 ACTUAL% = 0`	Initialize the actual number of bytes read in. This variable is given a value by IOENTERB.
`170 FLAG% = 4`	Swap every four bytes.
`180 CALL IOENTERB(VNA&, SEG DAT!(2, 0), MAX%, ACTUAL%, FLAG%): GOSUB ERRORTRAP`	Read in the data from the analyzer.
`190 ADDRESS& = VNA&: A$ = "SCAL?;": GOSUB IOOUTS`	Request the scale factor from the network analyzer.
`200 CALL IOENTER(VNA&, SCAL!): GOSUB ERRORTRAP`	Receive the scale factor.
`210 ADDRESS& = VNA&: A$ = "REFP?;": GOSUB IOOUTS`	Request the reference position from the network analyzer.
`220 CALL IOENTER(VNA&, REFP!): GOSUB ERRORTRAP`	Receive the reference position.
`230 ADDRESS& = VNA&: A$ = "REFV?;": GOSUB IOOUTS`	Request the value at the reference position from the network analyzer.
`240 CALL IOENTER(VNA&, REFV!): GOSUB ERRORTRAP`	Receive the value at the reference position.
`250 XMAX% = 5850: YMAX% = 4094`	Set maximum limits for x and y values.These are the corner coordinate values given in the section entitled *Display Graphics* in the *HP-IB Quick Reference;* YMAX% is rounded to an even number for simplicity.
`260 XCENTER% = XMAX% / 2: YCENTER% = YMAX% / 2`	Initialize the center values for x and y to reasonable values.
`270 SIZE% = 750`	Initialize the size of the square to a reasonable value.
`280 ADDRESS& = DISPLAY&: A$ = "CS; SP2;": GOSUB IOOUTS`	Turn off the analyzer's measurement display and set its color to that of channel 1 memory using display graphics commands.
`290 PRINT "ADJUST SIZE OF VIEWPORT. PRESS [ENTRY OFF] TO CONTINUE."`	Display instructions on the computer CRT.
`300 KEYCODE% = 0: OLDSIZE% = 0`	Initialize KEYCODE for entry into the DO UNTIL loop, and initialize OLDSIZE% for entry into the IF...THEN loop. This ensures that the square is drawn the first time.
`310 DO UNTIL (KEYCODE% = 34)`	Continue to adjust the size of the square until **[ENTRY OFF]** is pressed on the analyzer.
`320  IF (SIZE% <> OLDSIZE%) THEN`	If the size of the square has been changed, redraw it.
`330   GOSUB DRAWSQUARE`	
`340   OLDSIZE% = SIZE%`	Keep track of the previous size setting.
`350  END IF`	If the size has not changed, the square does not need to be redrawn.
`360  GOSUB GETKEY`	Wait for an analyzer key to be pressed, and get its code.
`370  IF KEYCODE% < 0 THEN`	KEYCODE% indicates a knob count if it is negative.

380	IF (KEYCODE% < −64) THEN KEYCODE% = KEYCODE% + 32768	If the knob count is less than −64, add 32768 (2^15) to recover it. If the knob count is greater than −64, no decoding is needed.
390	SIZE% = SIZE% − KEYCODE% * 15	Adjust the size of the square according to the knob count, multiplying the knob count to make the size change significant.
400	ELSE	KEYCODE% indicates a key press if it is positive.
410	IF (KEYCODE% <> 34) THEN	If the key press was not **[ENTRY OFF]**, it was not a valid key, so display an appropriate message on the computer CRT.
420	PRINT "ONLY <ENTRY OFF> AND KNOB TURNING ARE VALID ENTRIES"	
430	END IF	
440	END IF	
450	IF (SIZE% < 100) THEN	Enforce the minimum size limit.
460	SIZE% = 100	
470	ELSE	
480	IF (SIZE% > ((YMAX% / 2) − 2)) THEN	Enforce the maximum size limit.
490	SIZE% = ((YMAX% / 2) − 2)	
500	END IF	
510	END IF	
520	LOOP	The size of the square has now been adjusted.
530	CLS	Clear the computer CRT.
540	ADDRESS& = DISPLAY&: A$ = "SP4;": GOSUB IOOUTS	Set the analyzer display's color to that of channel 2 memory by using a display graphics command.
550	PRINT "ADJUST POSITION OF VIEWPORT. PRESS <ENTRY OFF> TO STOP."	Display operator instructions on the computer CRT.
560	KEYCODE% = 0: OLDXCENTER% = 0: OLDYCENTER% = 0	Initialize variables for entry into the DO UNTIL and IF...THEN loops. This ensures that the square is drawn the first time.
570	DO UNTIL (KEYCODE% = 34)	Continue to adjust the position of the square until **[ENTRY OFF]** is pressed on the analyzer.
580	IF ((OLDXCENTER% <> XCENTER%) OR (OLDYCENTER% <> YCENTER%)) THEN	If the position of the square has been changed, redraw it.
590	GOSUB DRAWSQUARE	
600	OLDXCENTER% = XCENTER%: OLDYCENTER% = YCENTER%	Keep track of the previous center settings.
610	END IF	If the position has not changed, the square does not need to be redrawn.
620	GOSUB GETKEY	Wait for an analyzer key to be pressed, and get its code.
630	SELECT CASE KEYCODE%	Reposition the square according to KEYCODE%.
	CASE 26	**[UP ARROW]** was pressed.

```
640    YCENTER% = YCENTER% + 150    Move the square up.
      CASE 18                       [DOWN ARROW] was pressed.
650    YCENTER% = YCENTER% - 150    Move the square down.
      CASE IS < 0                   The knob was turned.
660    IF (KEYCODE% < -64) THEN     Recover the knob count, if necessary.
       KEYCODE% = KEYCODE% +
       32768
670    XCENTER% = XCENTER% -        Move the square to the left or to the right
       KEYCODE% * 20                according to the knob count, multiplying it to
                                    make the position change significant.
      CASE 34                       [ENTRY OFF] was pressed, so accept the key as
                                    valid and do not move the square.
      CASE ELSE                     An invalid key was pressed.
680    PRINT "ONLY [UP ARROW],      Display an appropriate message on the computer
       [DOWN ARROW], [ENTRY         CRT.
       OFF], AND KNOB TURNING
       ARE VALID"

690   END SELECT
700   IF XCENTER% > (XMAX% -        Enforce the right side limit.
      SIZE% - 2) THEN
710    XCENTER% = (XMAX% - SIZE%
       - 2)
720   ELSE
730    IF XCENTER% < (SIZE% + 2)    Enforce the left side limit.
       THEN
740     XCENTER% = (SIZE% + 2)
750    ELSE
760     IF YCENTER% >(YMAX% -       Enforce the top side limit.
        SIZE% - 2) THEN
770      YCENTER% = (YMAX% -
         SIZE% - 2)
780     ELSE
790      IF YCENTER% < (SIZE% +     Enforce the bottom side limit.
         2) THEN
800       YCENTER% = (SIZE% + 2)
810      END IF
820     END IF
830    END IF
840   END IF
850  LOOP                           The position of the square has now been adjusted.
860  CLS                            Clear the computer CRT.
870  ADDRESS& = DISPLAY&: A$ =      Erase the user graphics display, and set the
     "AF; SP5;": GOSUB IOOUTS       analyzer display's color to that of the graticule by
                                    using a display graphics command.
880  GOSUB DRAWSQUARE               Redraw the square in its final color.
890  FOR I% = 1 TO 9                Draw a grid with ten divisions along each axis in
                                    the square.
900   OFFSET% = (2 * SIZE% * I% /   Determine the distance between the I%th grid line
      10) - SIZE%                   and the zero axis.
```

`910 A$ = "PU; PA" + STR$(XCENTER% + OFFSET%) + "," + STR$(YCENTER% - SIZE%) + ";": GOSUB IOOUTS`	
`920 A$ = "PD; PA" + STR$(XCENTER% + OFFSET%) + "," + STR$(YCENTER% + SIZE%) + ";": GOSUB IOOUTS`	Draw the I%th vertical grid line.
`930 A$ = "PU; PA" + STR$(XCENTER% - SIZE%) + "," + STR$(YCENTER% + OFFSET%) + ";": GOSUB IOOUTS`	
`940 A$ = "PD; PA" + STR$(XCENTER% + SIZE%) + "," + STR$(YCENTER% + OFFSET%) + ";": GOSUB IOOUTS`	Draw the I%th horizontal grid line.
`950 NEXT I%`	
`960 ADDRESS& = DISPLAY&: A$ = "SP1;": GOSUB IOOUTS`	Set the analyzer display's color to that of channel 1 data by using a display graphics command.
`970 BOTTOM! = REFV! - REFP! * SCAL!`	Calculate the value of the bottom grid line.
`980 FULL! = 10 * SCAL!`	Calculate the value of the full scale span across the grid.
`990 X% = XCENTER% - SIZE%`	Determine the x-position of the first point to plot.
`1000 Y% = ((DAT!(1, 1) - BOTTOM!) / FULL! * 2 * SIZE%) + YCENTER% - SIZE%`	Determine the y-position of the first point to plot.
`1010 ADDRESS& = DISPLAY&: A$ = "PU; PA" + STR$(X%) + "," + STR$(Y%) + ";": GOSUB IOOUTS`	Position the graphics pen at the first point to plot.
`1020 FOR I% = 2 TO POINTS%`	Draw the trace, point by point, using display graphics commands.
`1030 X% = (((I% - 1) / (POINTS% - 1)) * 2 * SIZE%) + XCENTER% - SIZE%`	
`1040 Y% = (((DAT!(1, I%) - BOTTOM!) / FULL!) * 2 * SIZE%) + YCENTER% - SIZE%`	
`1050 A$ = "PD; PA" + STR$(X%) + "," + STR$(Y%) + ";": GOSUB IOOUTS`	
`1060 NEXT I%`	
`1070 CALL IOLOCAL(ISC&): GOSUB ERRORTRAP`	Return the network analyzer to local mode and perform error trapping.
`1080 END`	End program execution.
`1090 ERRORTRAP:`	Define a routine to trap errors.
`1100 IF PCIB.ERR <> NOERR THEN ERROR PCIB.BASERR`	Perform error trapping.

`1110 RETURN`	Return from the ERRORTRAP routine.
`1120 IOOUTS:`	Define a routine to send a command string from the computer to the analyzer.
`1130 CALL IOOUTPUTS(ADDRESS&, A$, LEN(A$)): GOSUB ERRORTRAP`	Send the command string A$ out to the analyzer and perform error trapping.
`1140 RETURN`	Return from the IOOUTS routine.
`1150 DRAWSQUARE:`	Define a routine to draw a square on the analyzer's display.
`1160 ADDRESS& = DISPLAY&: A$ = "AF;": GOSUB IOOUTS`	Erase the old square using display graphics commands.
`1170 A$ = "PU; PA" + STR$(XCENTER% – SIZE%) + "," + STR$(YCENTER% – SIZE%) + ";": GOSUB IOOUTS`	Position the "pen" at the lower left corner of the square.
`1180 A$ = "PD; PA" + STR$(XCENTER% – SIZE%) + "," + STR$(YCENTER% + SIZE%) + ";": GOSUB IOOUTS`	Draw the left side of the square.
`1190 A$ = "PD; PA" + STR$(XCENTER% + SIZE%) + "," + STR$(YCENTER% + SIZE%) + ";": GOSUB IOOUTS`	Draw the top side of the square.
`1200 A$ = "PD; PA" + STR$(XCENTER% + SIZE%) + "," + STR$(YCENTER% – SIZE%) + ";": GOSUB IOOUTS`	Draw the right side of the square.
`1210 A$ = "PD; PA" + STR$(XCENTER% – SIZE%) + "," + STR$(YCENTER% – SIZE%) + ";": GOSUB IOOUTS`	Draw the bottom side of the square.
`1220 RETURN`	Return from the DRAWSQUARE routine.
`1230 GETKEY:`	Define a routine to wait for an analyzer key to be pressed and to get the key's code.
`1240 STAT% = 0`	Initialize STAT% for entry into the DO UNTIL loop.
`1250 DO UNTIL ((STAT% MOD 64) > 31)`	Wait for a key press to be indicated by the setting of bit 5 of the status byte.
`1260  CALL IOSPOLL(VNA&, STAT%): GOSUB ERRORTRAP`	Read in the status byte as an integer.
`1270 LOOP`	
`1280 ADDRESS& = VNA&: A$ = "ESR?;": GOSUB IOOUTS`	Now that a key press has occurred, request the Event Status Register value from the analyzer.
`1290 CALL IOENTER(VNA&, ESTAT!)`	Receive the Event Status Register value from the analyzer, thereby clearing the latched User Request bit so that old key presses will not trigger a measurement.
`1300 ADDRESS& = VNA&: A$ = "KOR?;": GOSUB IOOUTS`	Request the key code or knob count from the analyzer.

`1310 CALL IOENTER(VNA&,`
`KEYCODE!)`
	Receive the key code or knob count.

`1320 KEYCODE% = INT(KEYCODE!)`	Convert the key code or knob count to an integer.
`1330 RETURN`	Return from the `GETKEY` routine.

Running the program

1. Set up the analyzer to make a measurement before running the program.
2. Adjust the size of the display box from the network analyzer front panel using the knob. Press **[ENTRY OFF]** when the size is satisfactory.
3. Adjust the position of the display box from the network analyzer front panel using the knob and the up and down keys. Press **[ENTRY OFF]** when the position is satisfactory.
4. The computer sends the analyzer commands that draw a grid and the trace in the box on the analyzer's display.

Appendix A: Status Reporting

The status reporting mechanism of the network analyzer gives information about specific functions and events inside the network analyzer. The status byte is an 8-bit register, each bit of which summarizes the state of one aspect of the instrument. For example, the error queue summary bit will always be set if there are any errors in the queue. The value of the status byte can be read in two ways. The first way is to send the command OUTPSTAT. The second is to call the HP-IB Command Library routine IOSPOLL:

```
CALL IOSPOLL(VNA&, STAT%): GOSUB ERRORTRAP
```

The advantage of using this instead of the command OUTPSTAT is that this does not put the analyzer into the remote mode, and it thus gives the operator access to the network analyzer front panel functions. Reading the status byte does not affect its value.

In addition to the error queue, the status byte also summarizes the two Event Status Registers that monitor specific instrument conditions. Furthermore, the status byte has a bit that is set when the analyzer is issuing a service request over HP-IB and a bit that is set when the network analyzer is prepared to transmit data over HP-IB. For a definition of the status registers, see Figure A.1, *Status Reporting System*.

To tell if a bit of the status byte is set, it is necessary to determine the integer value corresponding to that bit (bit n is equivalent to 2^n). MOD can be used to remove the effect of all bits of higher value than the one of interest, and ≥ can be used to see if the bit of interest is set. For example, bit 4 corresponds to an integer value of 16, and bit 5 corresponds to an integer value of 32. If STAT% is the integer representation of the status byte, the following IF...THEN loop will only be entered if bit 4 is set:

```
IF ((STAT% MOD 32) > 15) THEN...
```

Example A1: Error queue

The following program illustrates how to monitor the analyzer's error queue from the computer. The error queue holds up to twenty instrument errors and warnings in the order that they occurred. Each time the network analyzer detects an error condition, it writes a message to its display and puts the error in the error queue. If there are any errors in the queue, bit 3 of the status byte will be set. Once the computer detects than bit 3 is set, the error can be requested from the queue with OUTPERRO, which commands the network analyzer to transmit the number and message of the oldest error in the queue.

Because the error queue will keep up to twenty errors until either all the errors are read out or the instrument is preset, it is important to clear out the error queue whenever errors are detected. Only errors, not prompts, are put in the error queue.

This example program is stored on the Example Programs disk as **IPGA1.BAS**.

`10 REM $INCLUDE: 'QBSETUP'`	Call the QuickBASIC initialization file QBSETUP.
`20 CLS`	Clear the computer CRT.
`30 ISC& = 7`	Assign the interface select code to a variable.
`40 VNA& = 716`	Assign the analyzer's address to a variable.
`50 CALL IOTIMEOUT(ISC&, 10!): GOSUB ERRORTRAP`	Define a system time-out of ten seconds and perform error trapping.
`60 CALL IOABORT(ISC&): GOSUB ERRORTRAP`	Abort any HP-IB transfers and perform error trapping.
`70 CALL IOCLEAR(ISC&): GOSUB ERRORTRAP`	Clear the analyzer's HP-IB interface and perform error trapping.
`80 CALL IOEOI(ISC&, 0): GOSUB ERRORTRAP`	Disable the End-Or-Identify mode for transferring data and perform error trapping.
`90 LENGTH% = 50`	Set a maximum length for the string to hold the error data.
`100 ERRDATA$ = SPACE$(LENGTH%)`	Prepare a string to hold the error data.

110 STATUSPOLL: STAT% = 0	Initialize the status byte for entry into the DO UNTIL loop.
120 DO UNTIL ((STAT% MOD 16) > 7)	Loop until bit three of the status byte, the error queue summary, is set.
130 CALL IOSPOLL(VNA&, STAT%): GOSUB ERRORTRAP	Read the status byte into the variable STAT% using a serial poll. The serial poll is an HP-IB function dedicated specifically to getting the status byte of an instrument quickly without causing the instrument to go into remote mode.
140 LOOP	
150 A$ = "OUTPERRO;": GOSUB IOOUTS	Now that the error queue has something in it, instruct the analyzer to output the error data, which consists of an error number and an error message string. This communication with the network analyzer puts it in remote mode.
160 ACTUAL% = 0	Initialize the actual number of bytes read in. This variable is set during IOENTERS.
170 CALL IOENTERS(VNA&, ERRDATA$, LENGTH%, ACTUAL%): GOSUB ERRORTRAP	Read the error data into one string. This will then consist of the error number (as a string) and the error message string.
180 ERRNUM% = VAL(LEFT$(ERRDATA$, 5))	Extract the error number from the string read in.
190 I% = 9	Initialize the string counter to begin after the error number.
200 ERRID$ = ""	Initialize the error message string.
210 DO UNTIL MID$(ERRDATA$, I%, 1) = CHR$(34)	Repeat until the end of the string has been reached.
220 ERRID$ = ERRID$ + MID$(ERRDATA$, I%, 1)	Extract the error message from the error data string one character at a time.
230 I% = I% + 1	Increment the counter at the next character.
240 LOOP	
250 PRINT ERRNUM%; ": "; ERRID$	Display the error number and error message string on the computer CRT.
260 CALL IOLOCAL(ISC&): GOSUB ERRORTRAP	Return the network analyzer to local mode so that the front panel is available to the operator. Perform error trapping.
270 SOUND 550, 2	Indicate audibly that an error occurred.
280 GOTO STATUSPOLL	Continue polling for errors.
290 END	End program execution.
300 ERRORTRAP:	Define a routine to trap errors.
310 IF PCIB.ERR <> NOERR THEN ERROR PCIB.BASERR	Perform error trapping.
320 RETURN	Return from the ERRORTRAP routine.
330 IOOUTS:	Define a routine to send a command string from the computer to the analyzer.
340 CALL IOOUTPUTS(VNA&, A$, LEN(A$)): GOSUB ERRORTRAP	Send the command string A$ out to the analyzer and perform error trapping.
350 RETURN	Return from the IOOUTS routine.

Running the program

1. Preset the network analyzer and run the program.

2. Nothing happens until an error occurs, so generate one. Three possible ways to do this on the network analyzer are the following:

   a. Press a blank softkey.

   b. Loosen the R connection.

   c. Press **[CAL]** *[CALIBRATE MENU] [RESPONSE] [DONE: RESPONSE].*

3. Once an error occurs, the computer will continue to beep and to display the error number and message until the error queue is empty (until the error number 0 and the error message NO ERRORS are received).

4. The computer will continue to monitor the network analyzer's error queue until the operator ends the program by pressing <CTRL-Break> on the computer keyboard.

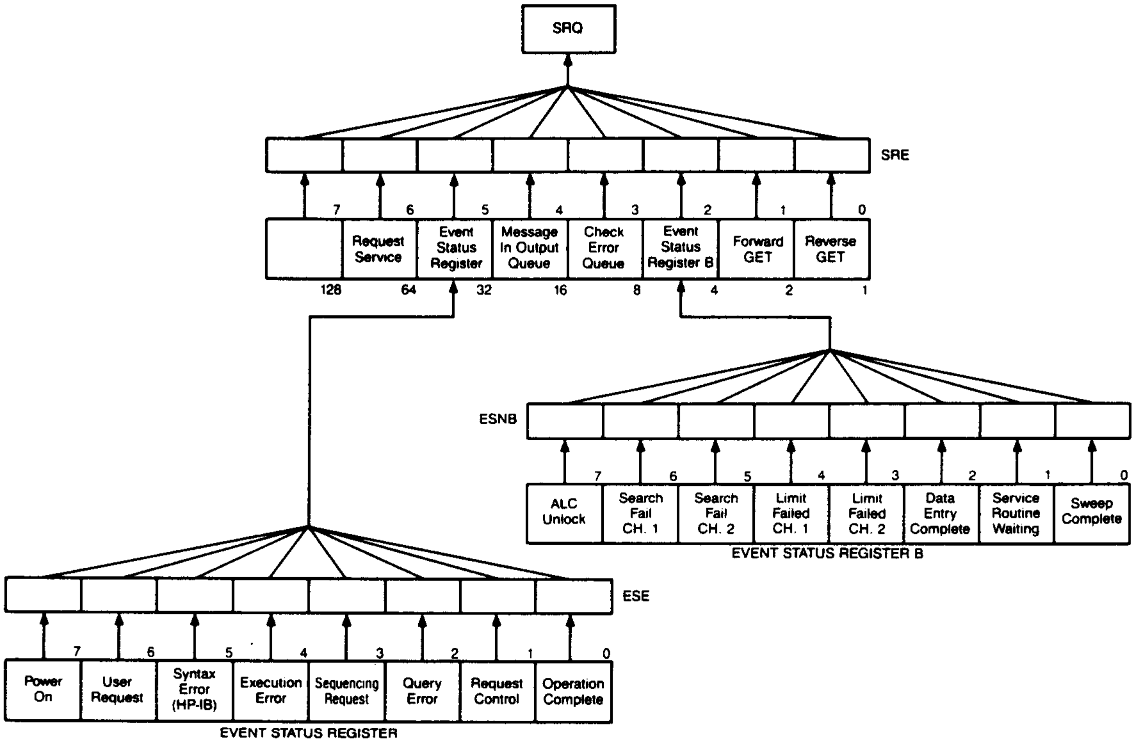


Figure A.1. Status reporting system

Example A2: Status registers

The following program illustrates how to monitor the analyzer's Event Status Register from the computer. The Event Status Registers are 8-bit registers which consist of latched event bits. A latched bit is set at the onset of the monitored condition. It is cleared when the register is read or when the command CLES (clear status) is sent.

Each time the network analyzer detects a key press or knob turn, it sets bit 6 of the Event Status Register. Once the computer detects that bit 6 is set, the key code or knob count can be requested from the analyzer with KOR?. Note that since the network analyzer is in remote mode, the normal function of the key pressed is not executed. In effect, the front panel has been taken over, and the keys could now be redefined.

This example program is stored on the Example Programs disk as **IPGA2.BAS**.

10 REM $INCLUDE: 'QBSETUP'	Call the QuickBASIC initialization file QBSETUP
20 CLS	Clear the computer CRT.
30 ISC& = 7	Assign the interface select code to a variable.
40 VNA& = 716	Assign the analyzer's address to a variable.
50 CALL IOTIMEOUT(ISC&, 10!): GOSUB ERRORTRAP	Define a system time-out of ten seconds and perform error trapping.
60 CALL IOABORT(ISC&): GOSUB ERRORTRAP	Abort any HP-IB transfers and perform error trapping.
70 CALL IOCLEAR(ISC&): GOSUB ERRORTRAP	Clear the analyzer's HP-IB interface and perform error trapping.
80 CALL IOEOI(ISC&, 0): GOSUB ERRORTRAP	Disable the End-Or-Identify mode for transferring data and perform error trapping.
90 GETKEY: ESTAT! = 0	Initialize ESTAT! for entry into the DO UNTIL loop.
100 DO UNTIL ((ESTAT! MOD 128) >63)	Wait for a key press to be indicated by the setting of bit 6, User Request, of the Event Status Register. MOD 128 removes the effect of all higher value bits (bit 7 is equivalent to 128 in decimal), and >63 ensures that bit 6, which is equivalent to 64 in decimal, is set.
110 A$ = "ESR?;": GOSUB IOOUTS	Request the Event Status Register value from the analyzer.
120 CALL IOENTER(VNA&, ESTAT!): GOSUB ERRORTRAP	Receive the Event Status Register value from the analyzer, thereby clearing the latched User Request bit so that old key presses will not trigger a measurement.
130 LOOP	
140 A$ = "KOR?;": GOSUB IOOUTS	Since the User Request bit has been set, request the key code or knob count from the analyzer.
150 CALL IOENTER(VNA&, KEYCODE!): GOSUB ERRORTRAP	Receive the key code or knob count from the analyzer.
160 IF KEYCODE! > = 0 THEN	If the code is positive, it was a key press rather than a knob turn.
170 PRINT "KEY CODE = ";	
180 ELSE	The code is negative, so it was a knob turn.
190 PRINT "KNOB TURN = ";	
200 IF KEYCODE! <-400 THEN	If the turn was a counter-clockwise rotation, the code needs to be recovered.
210 KEYCODE! = KEYCODE! + 32768	

220 END IF	
230 END IF	
240 PRINT KEYCODE!	Display the code or knob count on the computer CRT.
250 GOTO GETKEY	Wait for the next key press or knob turn.
260 CALL IOLOCAL(ISC&): GOSUB ERRORTRAP	Return the network analyzer to local mode and perform error trapping.
270 END	End program execution.
280 ERRORTRAP:	Define a routine to trap errors.
290 IF PCIB.ERR <> NOERR THEN ERROR PCIB.BASERR	Perform error trapping.
300 RETURN	Return from the ERRORTRAP routine.
310 IOOUTS:	Define a routine to send a command string from the computer to the analyzer.
320 CALL IOOUTPUTS(VNA&, A$, LEN(A$)): GOSUB ERRORTRAP	Send the command string A$ out to the analyzer and perform error trapping.
330 RETURN	Return from the IOOUTS routine.

Running the program

1. Preset the network analyzer and run the program.
2. Nothing happens until a key is pressed, so press one.
3. The computer will detect the key press or knob turn and display its code.
4. The computer will continue to monitor the network analyzer's key presses and knob turns until the operator ends the program by pressing <CTRL-Break> on the computer keyboard.

For more information, call your local HP sales office listed in your telephone directory or an HP regional office listed below for the location of your nearest sales office.

United States:
Hewlett-Packard Company
4 Choke Cherry Road
Rockville, MD 20850
(301) 670-4300

Hewlett-Packard Company
5201 Tollview Drive
Rolling Meadows, IL 60008
(312) 255-9800

Hewlett-Packard Company
5161 Lankershim Blvd.
No. Hollywood, CA 91601
(818) 505-5600

Hewlett-Packard Company
2015 South Park Place
Atlanta, GA 30339
(404) 955-1500

Canada:
Hewlett-Packard Ltd.
6877 Goreway Drive
Mississauga, Ontario L4V1M8
(416) 678-9430

Australia/New Zealand:
Hewlett-Packard Australia Ltd.
31-41 Joseph Street,
Blackburn, Victoria 3130
Melbourne, Australia
(03) 895-2895

Europe/Africa/Middle East:
Hewlett-Packard S.A.
Central Mailing Department,
P.O. Box 529
1180 AM Amstelveen,
The Netherlands
(31) 20/547 9999

Far East:
Hewlett-Packard Asia Ltd.
22/F Bond Centre
West Tower
89 Queensway
Central, Hong Kong
(5) 8487777

Japan:
Yokogawa-Hewlett-Packard Ltd.
29-21, Takaido-Higashi 3-chome
Suginami-ku, Tokyo 168
(03) 331-6111

Latin America:
Latin American Region Headquarters
Monte Pelvoux Nbr. 111
Lomas de Chapultepec
11000 Mexico, D.F. Mexico
(905) 596-79-33

November 1, 1989

Printed in U.S.A.
08753-90168

HEWLETT
PACKARD