HP Instrument BASIC User's Handbook Supplement

Agilent Technologies
8712ET/ES and 8714ET/ES
RF Network Analyzers

Agilent Technologies

Part No. 08714-90019

Printed in USA

Print Date: June 2000

Supersedes October 1998

Firmware Revision

This manual documents analyzers with firmware revisions E.06.00 and above.

Contents

1 Introduction

Introduction

This manual describes how to create and use HP Instrument BASIC (IBASIC) software in the analyzer. It demonstrates how to use IBASIC's programming, editing and debugging features. It also describes how to save and recall programs and how certain instrument-specific IBASIC features are implemented in the analyzer. The reader should be familiar with the operation of the analyzer and BASIC before programming the analyzer.

Related information can be found in the following references that were shipped with your analyzer:

- The *Programmer's Guide,* a companion to this book, provides general-purpose programming information. The *Programmer's Guide* describes all valid instrument commands.
- The *User's Guide* provides information on making measurements with the analyzer.
- The *HP Instrument BASIC User's Handbook* provides information on HP Instrument BASIC (IBASIC) .
- The *Example Programs Disk* — DOS format: part number 08714-10003, is included with the analyzer. This disk provides copies of all of the example programs in the *Example Programs Guide*.
- Additional information on HP BASIC programming is available in the manual set for the BASIC revision being used. For example: *BASIC 7.0 Programming Techniques* and *BASIC 7.0 Language Reference*.

Other information that may be helpful can be found in the following reference materials:

- The *Tutorial Description of the Hewlett-Packard Interface Bus* (literature no. 5021-1927) provides information on using the GPIB.

Contact the nearest Agilent sales office for ordering information. A list of sales and service offices can be found in the “Specifications and Characteristics” chapter of the *User's Guide*.

Overview of HP Instrument BASIC

HP Instrument BASIC (IBASIC) can be used for a wide range of applications, from simple recording and playback of measurement sequences to remote control of other instruments. IBASIC is a complete language with over 200 keywords.

IBASIC is a complete system controller residing inside your analyzer. It communicates with the analyzer via GPIB commands over an internal interface bus (select code 8). It can also communicate with other instruments, computers, and peripherals using the external GPIB interface (select code 7) or the serial (select code 9) or parallel (select code 15) I/O ports.

NOTE The analyzer can also be controlled by an external controller. It has a factory default external GPIB address of 16. When using IBASIC to control other instruments, no other device should use the same address.The external GPIB address can be changed using either the front panel keys under the SYSTEM OPTIONS, **GPIB** menu, or the SCPI mnemonic `SYST:COMM:GPIB:ADDR`.

Using HP Instrument BASIC

You need not be proficient in a programming language to successfully use IBASIC. In keystroke recording mode, IBASIC automatically builds an executable program by capturing measurement sequences as they are performed. With little or no editing of these program lines, you can immediately put your program to work to control and automate your analyzer.

The IBASIC language is a subset of the HP BASIC language. In fact, IBASIC programs can be run on any HP BASIC workstation with very few changes. When an external PC keyboard (with a DIN connector) is connected to the analyzer, the IBASIC user interface emulates the user interface of the HP BASIC. The PC keyboard can be used for command entry, editing and program inputs.

You can use IBASIC to do the following:

- create on screen graphics
- control other instruments and peripherals
- create interactive prompts
- simplify keystrokes with the BEGIN key
- create programs by recording keystrokes
- run applications

IBASIC also works in conjunction with an external controller which can download and run programs, query variables and respond to service requests (SRQs).

Allocating Internal Memory for IBASIC Use

Your analyzer contains a volatile RAM disk that is configured for use with IBASIC. The default condition set at the factory allocates most of this disk's memory for IBASIC use. To see what the current allocations are for this disk, press SAVE RECALL **Select Disk** **Configure VOL_RAM**. A message will appear on the analyzer's display that shows total memory available, and how the memory is currently allocated.

To change the allocations, use the **Modify Size** softkey. The number you enter with the **Modify Size** softkey, is the percentage of memory that will be used for normal disk functions (such as storing instrument states). The remainder will be allocated for use with IBASIC.

NOTE Cycle power after changing the allocations. You must cycle the power on the analyzer for new allocations to take effect.

Typographical Conventions

The following conventions are used in this manual when referring to various parts of the HP Instrument BASIC and analyzer operation environments:

HARDKEY	The name of a hardkey on the front panel of the analyzer. This notation is also used to represent keys on an external keyboard connected to the analyzer's DIN interface.
Softkey	The label of a softkey.
Softkey ON off	Upper case selection in a softkey indicates the state *after* the softkey is pressed.
HARDKEY **Softkey 1** **Softkey 2**	A series of hardkeys and softkeys represents the path to a given softkey or menu.
`<element>`	Angle brackets are used to signify a syntax element in a statement.
`computer font`	User input and computer output is shown with a mono-spaced font.

2 Recording Programs

Keystroke recording, described in this chapter, is ideal for creating simple programs or measurement sequences for instrument control. Other methods of program development may be used to supplement keystroke recording and create more sophisticated programs. If a program requires data processing, decision making, operator input, or a display of graphical diagrams, other program development methods are needed. These methods are covered in Chapter 5, “Developing Programs.”

IBASIC programs for the analyzer can be created and edited using the following methods:

- the front panel keys and knob
- an external keyboard (Option 1CL)
- an Agilent controller running HP BASIC
- a workstation or PC using a text editor or programming editor

Keystroke Recording

Keystroke recording is the easiest way to create IBASIC programs. It requires only a couple of steps to set up and run, and very little programming knowledge is required.

What Is Keystroke Recording?

Keystroke recording is a way to automatically create IBASIC measurement sequence programs. To record a program, press the following keys:

- SYSTEM OPTIONS **IBASIC** **Key Record on OFF**
- Normal key sequences for the measurement
- SYSTEM OPTIONS **IBASIC** **Key Record on OFF** to terminate recording

The resulting program can then be run by pressing **Run** in the SYSTEM OPTIONS **IBASIC** menu.

Keystroke recording works by finding the SCPI mnemonic that fits each *operation* performed by the instrument; then it builds a program line to perform that operation when the recorded program is executed. All program lines built by keystroke recording are entered into the analyzer's program buffer. If the buffer contains no existing lines, a complete executable program will be created. If there is a program in the buffer when recording is turned on, the recorded statements are inserted into the existing program. Refer to Chapter 5, “Developing Programs,” for a description of how to record into existing programs.

IBASIC Programs and the Input Buffer

Recorded programs send commands to the instrument, and use the same set of commands used by external controllers for remote operation of the instrument.

These commands are stored in an input buffer by the instrument. An IBASIC program generally outputs the commands much faster than the instrument can execute them. This often causes the program to complete while the instrument is still executing commands in the input buffer. The instrument continues processing these commands until the buffer is empty.

This may have some side-effects if you are not aware of this interaction. For example, it may not be immediately obvious that the program has actually finished, since the instrument is still functioning “remotely.” This could cause confusion if you try to pause and continue a program that has actually ended.

You can clear the buffer from within your program by inserting the statement `CLEAR 8` at the beginning of your program (see Chapter 5 for information on editing programs).

Another side-effect of the speed with which the analyzer processes commands is that it is possible for a command to execute before a previous command has completed execution. The most common example of this is a data query that executes before a measurement sweep is complete. This interaction can lead to erroneous data being collected. For more information on synchronizing the execution of commands, refer to “Synchronizing the Analyzer and a Controller” in the *Programmer's Guide*.

What's in a Recorded Program

A program created using keystroke recording is composed of three fundamental IBASIC statements: ASSIGN, OUTPUT and END. The following simple program demonstrates these statements:

```
1    ASSIGN @Rfna TO 800
2    OUTPUT @Rfna;”SOUR1:POW -10 dBm”
10   END
```

The ASSIGN and END statements are automatically created when keystroke recording is used to create a new program (as opposed to modifying an existing one).

There will only be one ASSIGN statement at the beginning of a program and one END statement at the end, but in a typical program there will be many OUTPUT statements. Since the OUTPUT statement does the actual work of controlling the analyzer, let's take a closer look at how it is used.

NOTE Each analyzer model requires a unique ASSIGN statement. The ASSIGN statement, which is automatically created, will vary depending on the model of the analyzer:

8712ES/ET	ASSIGN @Hp8712 To 800
8714ES/ET	ASSIGN @Hp8714 TO 800

The OUTPUT Statement

The IBASIC statement

```
OUTPUT <destination>;<data>
```

tells the internal computer to send some information(`data`) to a device at a specific address(`destination`). The destination can be a device selector number (example: `OUTPUT 800`), or a name representing a number, called a path name (example: `OUTPUT@Rfna`). In recorded IBASIC programs, the data are strings containing commands for the instrument (mnemonics).

The following represents a typical `OUTPUT` command from a recording session:

```
OUTPUT @Rfna;"SOUR1:POW -10 dBm"
```

Notice that the `OUTPUT` command is followed by a name representing a device selector (`@Rfna`), followed by a semicolon and the data (`SOUR1:POW -10 dBm`).

The ASSIGN Statement

The destination in an `OUTPUT` statement specifies the address of the device. In recorded programs this address is represented by the I/O path name `@Rfna`. The following line appears in all recorded programs before any `OUTPUT` statements:

```
ASSIGN @Rfna to 800
```

The `ASSIGN` statement allows you to substitute an I/O path name (a variable preceded by the @ symbol) for a device selector number. Therefore, after the above `ASSIGN` statement, the program line

```
OUTPUT @Rfna;"SOUR1:POW -10 dBm"
```

is equivalent to

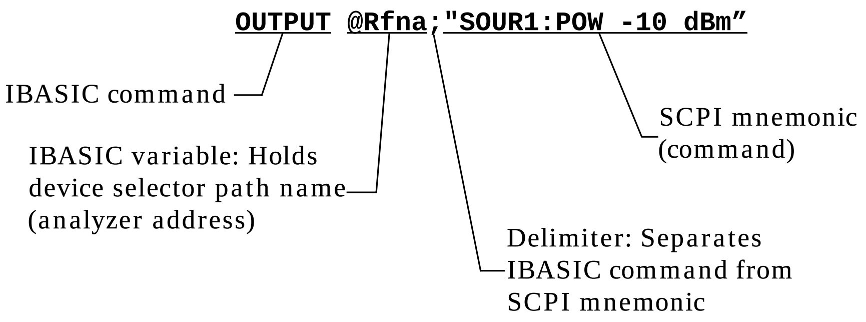
```
OUTPUT 800;"SOUR1:POW -10 dBm"
```

The device selector 800 specifies the host instrument as the destination of any data sent by the `OUTPUT` command. The program communicates with the analyzer via select code 8, the internal GPIB interface, which is used only for communication between IBASIC programs and the analyzer. The analyzer will respond to any address on the internal interface from 800 to 899 (800 is typically used).

SCPI Mnemonics

The data sent to the analyzer by the `OUTPUT` command is called a SCPI (Standard Commands for Programmable Instruments) mnemonic and is found in quotes following the device selector path name and semicolon. See Figure 2-1.

Figure 2-1 **Example IBASIC `OUTPUT` Command Syntax**


OUTPUT @Rfna;"SOUR1:POW -10 dBm"
IBASIC command
IBASIC variable: Holds device selector path name (analyzer address)
Delimiter: Separates IBASIC command from SCPI mnemonic
SCPI mnemonic (command)

SCPI is a standard instrument control programming language that provides commands that are common from one product to another, thus reducing the number of device specific commands. It uses easy-to-learn, self-explanatory syntax that provides flexibility for both novice and expert programmers.

The SCPI mnemonic codes used by IBASIC are the same ones used to control the instrument remotely via an external computer. External computers communicate with the analyzer over the external GPIB bus while IBASIC programs communicate with it over the internal bus. In our example, the mnemonic “`SOUR1:POW -10 dBm`” tells the instrument to set the source power to –10 dBm.

For more information on GPIB interfacing using IBASIC, refer to Chapter 8, “Interfacing with External Devices.” The SCPI mnemonics for the analyzer are documented in the *Programmer's Guide*.

How Recording Works

Keystroke recording works by recording the functional *operation* of the instrument, *not* by recording every keystroke. A sequence of SCPI mnemonics forms the recorded sequence of operations.

As you press a sequence of keys to perform an operation, SCPI mnemonics for that operation are generated and recorded. The operation may take one keystroke or several keystrokes, but the mnemonic is not generated until the *operation* is complete.

In other words, it is the functional operation of the instrument that is recorded as a mnemonic. The separate, individual keystrokes to perform that operation are not recorded.

For example, recording the simple key sequence: POWER **Level** – 1 0 **Enter** requires six keystrokes and produces a single mnemonic that is generated after the keystroke sequence is completed:

```
"SOUR1:POW -10dBm"
```

This mnemonic is then automatically formed into the command

```
   OUTPUT @Rfna;"SOUR1:POW -10 dBm"
```

and is inserted into the program.

If you accidentally press an incorrect key in a sequence, it may not show up in the recorded program. Also, you cannot exactly mimic keystrokes to leave the instrument in a specific front panel state, unless this state appears as a natural consequence of a completed operation.

As shown in the above example, pressing the POWER hardkey in a recording session has the effect of bringing up the POWER menu, but does not, by itself, generate a program line. You could not, therefore, leave the instrument with the POWER menu displayed.

Operations That Do Not Record

Although keystroke recording works automatically in most situations, some operations cannot be captured or can only be partially captured using this method. This is true for one of the following reasons:

- There is no corresponding SCPI mnemonic for the front panel keystrokes (such as transitional key sequences).
- The keystrokes are IBASIC front panel operations (such as some of the softkey operations found under the SYSTEM OPTIONS **IBASIC** menu).
- The operation requires additional programming steps (such as passing control of the GPIB to the instrument for hardcopy output).
- The GPIB operation has no front panel equivalent (such as GPIB query commands or data transfer).
- The keystrokes are service menu keys (in general).

NOTE Do not recall programs in keystroke record mode. This will overwrite previously recorded program steps.

Front Panel Operations without Mnemonics

The following descriptions indicate areas of front panel operation which have no corresponding SCPI mnemonics:

- Most operations on the front panel requiring numeric entry allow use of the knob to increment or decrement the current value. This will not record as a program line. *Always use the numeric keypad or step keys to enter any value if you want the operation to be recorded.*
- During a measurement sequence, it may take several key presses to cause an operation that will generate a mnemonic. The transitional sequences between actual instrument events are not recordable. For example, pressing the SCALE key displays the scale numeric entry, but nothing is recorded until you enter a value for the scale parameter.

- Any default states which you set up prior to recording or which you encounter while recording (and consequently do not select) are not recorded.
- Use of step keys is not recommended because the results may depend on the function's step size, which may change as other parameters change.

NOTE Instrument states that are not specifically selected or changed are not recorded. Since these default states are not recorded, you must either actively select them to generate a program statement, or make sure the instrument is in exactly the same state when the program is run as when it was recorded. This is discussed further in "Avoiding Recording Errors" on page 2-13.

HP Instrument BASIC Operations

Some softkeys under the (SYSTEM OPTIONS) **IBASIC** menu cannot be recorded. Operations on programs, such as **Run**, **Continue**, **Edit** and (SAVE RECALL) **Programs**, do not record. You can, however, record display partitions and all other save and recall operations not having to do with IBASIC programs.

Although IBASIC operations cannot be recorded, many do have corresponding SCPI mnemonics that allow an external controller to control and communicate with internal IBASIC programs. For more information, refer to Chapter 8, "Interfacing with External Devices."

Operations Requiring Additional Programming

Some operations that work well when performed manually from the front panel require special attention when used in a program. This is due to two interactions: synchronization and active control.

Synchronization

Often, one event must be completed before another can occur. Suppose you want your program to perform a limit test on data, but only after a sweep has been completed. You can record the command to perform the limit test by pressing key sequences. However, to detect when the instrument has completed a sweep, you must edit the program and include a routine that waits for a status register to indicate the end of the sweep.

CAUTION

Synchronization is required only with overlapped commands. Overlapped commands, such as the command to trigger a sweep, don't hold off the processing of subsequent commands, and are not necessarily completed when the next command begins. The analyzer adds a "wait" command (`*WAI`) when an overlapped command is created using keystroke recording. `*WAI` delays processing of subsequent commands until previous commands have been parsed. `*WAI` *does not insure previous commands have completed*. For more information on synchronization, see "Synchronizing the Analyzer and a Controller" in the *Programmer's Guide*.

Active Control of the GPIB Interface

Some operations require the analyzer to be the active controller on the external GPIB bus. The analyzer must be the System Controller (or active control must be passed to it from an external controller). When an IBASIC program begins running, control of the external interface is automatically passed to the program, so active control must be passed back to the analyzer before these operations can be performed.

These operations include all of the following actions when they are directed to GPIB devices.

- HARDCOPY **Start**
- HARDCOPY **Abort**

NOTE

Active control of the GPIB interface is only a problem if that bus is being used. Hardcopy output to devices on the serial or parallel ports do not require control of the GPIB.

The example program "PASSCTRL" in the *Example Programs Guide* and on the *Example Programs Disk* illustrates these concepts.

You can use keystroke recording for any of these operations but you will not be able to successfully run the program that is generated. You will need to enter the program lines necessary to first pass control to the analyzer and then wait for control to be passed back to the program.

See "Passing and Regaining Control" on page 8-16 for an example of passing control to the analyzer.

Mnemonics with No Corresponding Front Panel Operation

Several of the analyzer SCPI mnemonics for the instrument perform operations that are not available from the front panel and cannot be recorded. These include operations such as querying instrument status, transferring data over GPIB, setting and clearing status registers, and general GPIB housekeeping.

These operations are useful for the more advanced GPIB programmer using IBASIC. Because they fall outside the direct operating realm of the analyzer, they cannot be recorded. They can be added to a recorded program using the built-in editor or another editing environment. See your analyzer's *Programmer's Guide* for a complete description of the analyzer's GPIB command set. See also "Built-In High-Speed Subprograms" on page 9-3.

Avoiding Recording Errors

Use Instrument Preset

In most cases, the (PRESET) key/operation should be the first keystroke recorded. This sets the instrument to its default state and avoids the risk of creating a program that depends on prior instrument settings.

You can include the command to perform a preset in your program by pressing (PRESET) immediately after turning recording on. This inserts the following line prior to all other `OUTPUT` statements in your program:

```
OUTPUT @Rfna;"SYST:PRES;*WAI"
```

See your analyzer's *User's Guide* to determine the specific preset state for your particular analyzer.

Specifically Select Parameters

If you do not want to preset the instrument before running a recorded program (for example, you may be recording a section of a larger measurement sequence), be sure to specifically activate every instrument setting that you will need in your automated sequence. For example, if you want the data format to be Log Mag, press (FORMAT) and then **Log Mag** , even though Log Mag is already the default setting. This will generate a program line to specifically set the data format to Log Mag.

In some cases you may have to select another setting first and then re-select the original setting in order to generate the correct program line. For example, if you want to generate a program line to set the sweep trigger to Continuous, and you discover that it is already set to Continuous when you start recording, press (MENU) **Trigger** **Hold** first—then press **Continuous** . You can easily remove unwanted program lines generated by this procedure in the editor.

NOTE

Do not rely on the step keys or the front panel knob to set parameters. Use of step keys is not recommended because the results may depend on the function's step size, which may change as other parameters change.

Use GPIB Echo

GPIB echo is a useful analyzer feature that allows you to view the SCPI mnemonic or mnemonics corresponding to any operation executed from the front panel. To turn on GPIB echo, press SYSTEM OPTIONS **GPIB** and **GPIB Echo on OFF**. After doing this, you will see a mnemonic appear in a dialogue box on the screen as you complete any key sequence that has a matching SCPI mnemonic.

This mnemonic is used in your recorded program during a recording session.

Using GPIB echo, you can preview the SCPI mnemonic commands that will be stored in your program before you actually record them. While this is not essential, it can be very useful when you are in doubt as to what a particular key sequence will record, or precisely when a key sequence corresponding to a mnemonic is completed.

3 Running, Pausing, and Stopping Programs

This chapter describes how to start, stop, and pause a program from the front panel. Automatic execution of a program is described.

IBASIC programs can also be remotely controlled via SCPI commands over the GPIB. For information on running, pausing and stopping programs using an external controller, see Chapter 8, “Interfacing with External Devices.”

Starting Programs Automatically

When the analyzer is powered up, it automatically searches for a program named `AUTOST` or `AUTOST.BAS`. When an `AUTOST` program is found, it is automatically loaded and executed. The analyzer searches in the following order:

- internal non-volatile RAM disk
- internal floppy disk drive

The `AUTOST` program can be used for anything from configuring the analyzer for specific measurements, much like an internal instrument state Save/Recall register, to diagramming measurement setups using graphics commands, as in a guided measurement sequence.

Refer to Chapter 4, “Saving and Recalling Programs,” for information on using the analyzer to name programs before they are saved.

Running and Continuing a Program

To run an IBASIC program that is already in the analyzer program buffer, press the **Run** softkey in the SYSTEM OPTIONS **IBASIC** menu. The RUN command can also be executed from an external keyboard in three ways:

- Press the function key F1 that corresponds to the **Run** softkey (see note below).
- Type RUN on a command line and press **Enter**. A command line is always available when an IBASIC display is partitioned. (See Chapter 5, "Developing Programs," for information about display partitions.) You can also activate a command line from an external keyboard with no IBASIC displays partitioned by pressing the ESC key on your external keyboard.
- Press the function key F9 when the command line is available.

NOTE

The keyboard function keys duplicate softkey actions.
When an external keyboard is connected, the function keys F1 through F8 always represent the analyzer's eight softkeys. The analyzer's hardkeys are each represented by a combination of Shift or Ctrl and one of the function keys. Refer to your analyzer's *User's Guide* for more information on the external keyboard interface. The SYSTEM OPTIONS **IBASIC** menu can be accessed from an external keyboard using Ctrl + F3 (for SYSTEM OPTIONS) and F1 (for **IBASIC**). A keyboard template showing which keys to press for specific analyzer functions is supplied with your analyzer (part number 08712-80028).

The RUN command is executed in two phases: prerun initialization and program execution.

The following occurs during the prerun initialization phase:

- Memory is reserved for variables specified in COM (both labeled and blank), DIM, REAL or INTEGER statements, or implied in the main program segment. Numeric variables are initialized to 0; string variables are initialized to the null string.

NOTE *Variables in COM are initialized only once—the first time a program is run.*

- The program is checked for syntax errors that require more than one program line to detect. Included in this are errors such as incorrect array references, and mismatched parameter or COM lines.

After prerun is successfully completed, the program begins the execution phase. Program lines are executed until one of the following events occurs:

1. An END or STOP statement is encountered in the program.
2. The PRESET hardkey is pressed to reset the instrument.
3. The **Pause** softkey is pressed to pause the program.
4. A PAUSE statement is encountered in the program.

Pausing and Continuing a Program

The program control softkey menu is always available when an IBASIC program is running. This “Program Running” menu has seven user-defined softkeys and a **Pause** softkey. Press the **Pause** softkey to suspend execution of a program. **Pause** is the eighth softkey and is represented by F8 on an external keyboard.

The program can also be paused by inserting a PAUSE statement in the program. The instrument responds as if you had pressed the **Pause** softkey. Refer to Chapter 5, “Developing Programs,” to learn how to insert statements in your recorded program. Note that PAUSE is one of the IBASIC keywords included in the editor's label window (also described in Chapter 5).

To continue the program from a paused state, press the **Continue** softkey in the SYSTEM OPTIONS **IBASIC** menu or F2 on an external keyboard. Continuing a paused program resumes program operation from where it was paused, retaining the current program context (variable values, etc.).

Pausing a program does not close any files that have been opened by the program. You will not be able to perform any of the following disk operations after pausing a program that has left a file open on that medium:

- RENAME FILE
- DELETE FILE
- DELETE ALL FILES
- COPY FILES
- COPY DISK
- FORMAT DISK

To close all open files, you must complete the execution of the program or perform an IBASIC RESET. This can be done by pressing the PRESET hardkey. The PRESET hardkey is represented by Ctrl + F4 on an external keyboard. Keystroke recorded programs do not open files and therefore avoid this problem.

Using LOCAL LOCKOUT 8 to Disable the Pause Key

The **Pause** key is always enabled when a program is first run and remains enabled until a LOCAL LOCKOUT 8 command is executed.

Example 1. **Pause** key locked-out from within a program.

```
10 !
20 LOCAL LOCKOUT 8
30 !
40 ...
```

Example 2. The **Pause** key can be re-enabled from within the program with the LOCAL 8 command at any time.

```
 90 !
100 Re-enable_pause:
110   LOCAL 8
120 RETURN
130 !
```

The PRESET key can still be used to stop a program even if the LOCAL LOCKOUT 8 command has been executed.

Stopping a Program

To stop a program completely, press the (PRESET) hardkey at any time while the program is running. This causes an IBASIC RESET. Placing a STOP statement in your program will also terminate the program, but does not perform an IBASIC RESET operation. The END statement can also be used to stop program execution, but it must be the last line in the main program segment.

The program remains in the program buffer after execution stops. To clear the program buffer, press (SYSTEM OPTIONS) **IBASIC** **Utilities** **Clear Program** or turn off the instrument.

For more information on the PAUSE and STOP statements, see the "HP Instrument BASIC Language Reference" section of the *HP Instrument BASIC User's Handbook*.

4 Saving and Recalling Programs

IBASIC programs are stored in memory, on disk, or in an external computer.

To transfer a program between the instrument's buffer and a disk mass storage device, use the Programs menu. To access the Programs menu using an external keyboard, use Ctrl + F1 (for SAVE RECALL) and F5 (for Programs).

The `GET`, `SAVE`, `LOAD`, `STORE`, `RE-STORE`, and `RE-SAVE` commands can be used within a program or from an IBASIC command line to transfer program files to and from mass storage. An autoload feature also exists to allow for a program (named `AUTOST` or `AUTOST.BAS`) to be automatically recalled from the internal non-volatile RAM disk or the built-in floppy disk and run at power-up.

Using an external controller, such as an HP Series 200/300/700 workstation, you can combine the convenience of keystroke recording in IBASIC with the ease of program editing on a dedicated external workstation. Record the measurement sequence and transfer the program to the workstation for further editing. In addition, fully developed programs may be transferred from an external controller to the analyzer. Transferring programs between the analyzer and an external computer is described in detail in Chapter 8, “Interfacing with External Devices.”

This chapter describes all program transfer operations between the program buffer and the analyzer's internal non-volatile RAM disk, internal volatile RAM disk, and internal floppy disk drive.

Selecting a Disk

When the SAVE RECALL **Programs** menu is selected, the analyzer automatically catalogs the selected disk or memory. The selected disk is one of the following mass storage devices:

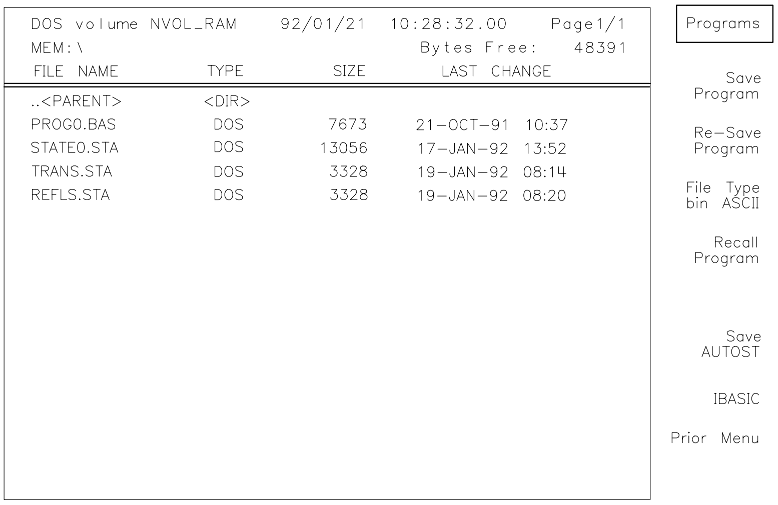
- Internal Non-Volatile RAM Disk **Non-Vol RAM Disk**
- Internal Volatile RAM Disk **Volatile RAM Disk**
- Internal Floppy Disk Drive **Internal 3.5” Disk**

To select a mass storage device, press the **Select Disk** softkey in the SAVE RECALL menu. Then press the key corresponding to your choice.

Saving a Program

To save the current contents of the analyzer's program buffer to a file, press **Save Program** in the SAVE RECALL **Programs** menu. Specify the type of file with the **File Type** softkey. The default choice is ASCII. The program is saved to an ASCII file with a default name on the currently selected mass storage device or disk. Each time the **Save Program** key is used a new file is created. These files are named `PROG0.BAS`, `PROG1.BAS`,... with the number incrementing for each new file. For portability, save files in ASCII. For more information regarding when to save files in binary format, see "User-Created Subprograms" on page 9-2.

Figure 4-1 **The SAVE RECALL Screen**



```
DOS volume NVOL_RAM      92/01/21   10:28:32.00      Page1/1
MEM:\                                    Bytes Free:    48391
FILE  NAME          TYPE        SIZE        LAST  CHANGE

..<PARENT>          <DIR>
PROG0.BAS            DOS        7673      21-OCT-91  10:37
STATE0.STA           DOS       13056      17-JAN-92  13:52
TRANS.STA            DOS        3328      19-JAN-92  08:14
REFLS.STA            DOS        3328      19-JAN-92  08:20
```

cp61b

If you are re-saving a program — that is, saving a file to a disk that already contains the file name — press **Programs** and use the arrow keys to highlight the name of the file to be re-saved. Then press **Re-Save Program** and the file is saved. The disk is automatically catalogued when the SAVE RECALL menu is selected.

The **Re-Save Program** softkey can also be used to save a new program with a non-default file name. Press **Re-Save Program**. Enter the new program's name using the external keyboard or the internal label maker. If no file with that name exists on the disk, a new file is created.

NOTE Whenever possible use ASCII as the file type for the following reasons:

- ASCII format is faster.
- Binary format is not compatible from one model of analyzer to another.
- HP BASIC cannot read a binary file from the analyzer.

AUTOST Programs

IBASIC allows you to designate a program to be automatically loaded and run when the instrument is first powered up. To make an autoloading program, save it with the file name AUTOST on the internal floppy disk drive or internal non-volatile RAM disk. This can be done from the SAVE RECALL **Programs** menu by pressing **Save AUTOST** or by using the **Re-Save Program** softkey and entering the file name AUTOST.

When the analyzer is powered up, it automatically searches the internal non-volatile RAM disk and then the built-in floppy disk drive for a program named AUTOST or AUTOST.BAS. When an AUTOST program is found, it is automatically loaded and executed.

Recalling a Program

To recall a program file from mass storage to the program buffer, use the SAVE RECALL **Programs** menu to catalog the disk. Select the desired mass storage device or disk, use the arrow keys to highlight the file and press **Recall Program**.

The recalled program file is entered into the program buffer one line at a time and checked for syntax errors. Lines with syntax errors are commented out and the IBASIC syntax error is displayed briefly in an error message and written to the CRT at the same time. To view error messages logged to the CRT, use the SYSTEM OPTIONS **IBASIC** **IBASIC Display** menu to allocate a screen partition for IBASIC.

NOTE Recalled programs overwrite the current program. Any program recalled to the program buffer using the **Programs** menu will overwrite the current contents of the program buffer. Be sure to save your current program before recalling another program from disk.

CAT to a String Array Exception

The analyzer's treatment of CAT to a string array is not the standard as documented in the *HP IBASIC Language Reference*. If you send the catalog to a string array, the array must contain at least 59 characters for a directory listing rather than the standard 80.

Accessing Mass Storage within Programs

The following mass storage specifiers may be used with many of the commands listed in Table 10-10, “Mass Storage Keywords.”

MEMORY,0,0 = non-volatile RAM disk

MEMORY,0,1 = volatile RAM disk

INTERNAL = floppy disk drive

Example

```
GET “PROG1.TXT:MEMORY,0,1”
```

Copies the file named PROG1.TXT from the volatile RAM disk to IBASIC program memory.

NOTE Access to NFS mounted file system is not supported directly under IBASIC. SCPI commands must be used to access NFS mounted file systems.

5 Developing Programs

For many applications, keystroke recording alone is sufficient to create and run programs. However, with some knowledge of the IBASIC language and the program development capabilities of the analyzer, you can significantly increase the power of your recorded programs or create your own original programs.

This chapter describes the operation of the following keys in the SYSTEM OPTIONS **IBASIC** menu, and any softkeys found in their underlying menus:

- **Edit**
- **IBASIC Display**
- **Utilities**

Edit starts the editor. You can make changes to your program on a line-by-line basis, or create a new program.

The **IBASIC Display** menu allows you to select the part of the CRT display available for use by IBASIC. An IBASIC display partition provides you with a place to enter IBASIC commands from an external keyboard. It also provides an area for viewing graphics and program output.

Utilities allows you to clear programs from the program buffer, allocate memory for program use, or secure program lines.

External Editors

In addition to the built-in IBASIC editor, programs can be developed in the following external environments:

- HP BASIC editors
- ASCII word processors
- programming editors

These external editing environments provide many advantages, the most notable are improved speed and flexibility. Precautions must be taken when using ASCII word processors because they do not provide the syntax checking available when using the internal editor.

After editing a program in an external environment, the best practice is to `GET` the program from an IBASIC command line using the following procedure (instead of using the SAVE RECALL keys described in Chapter 4, “Saving and Recalling Programs”).

1. Partition an IBASIC display (as described later in this chapter).
2. Use an external keyboard to enter the command `GET "PROG0:,4"` (this command loads a program file `PROG0` from the internal floppy disk drive).
3. Watch the IBASIC display as the program is loaded — syntax errors result in error messages which are displayed on the screen.
4. Edit the program to correct any errors found.

HP BASIC

The HP BASIC editor checks for the syntax of the version of HP BASIC being used. Because IBASIC is a subset of HP BASIC, it may not find all of the errors — the most common error is the use of HP BASIC commands that are not supported by IBASIC. For a listing of the commands supported by IBASIC, refer to Chapter 10, “IBASIC Keyword Summary.”

ASCII Word Processors

When an ASCII word processor is used to edit a program, syntax checking does not occur until the program is loaded by the instrument. Also, program line numbers are not automatically renumbered when new lines are inserted.

It is recommended that you renumber the program, as described later in this chapter, to reduce the possibility of errors. Errors in numbering lines usually do not result in a syntax error; they write over other program lines.

Programming Editors

Editors designed for computer programming offer some advantages:

- methods to visually distinguish language elements with color or other type style
- complex search-and-replace utilities

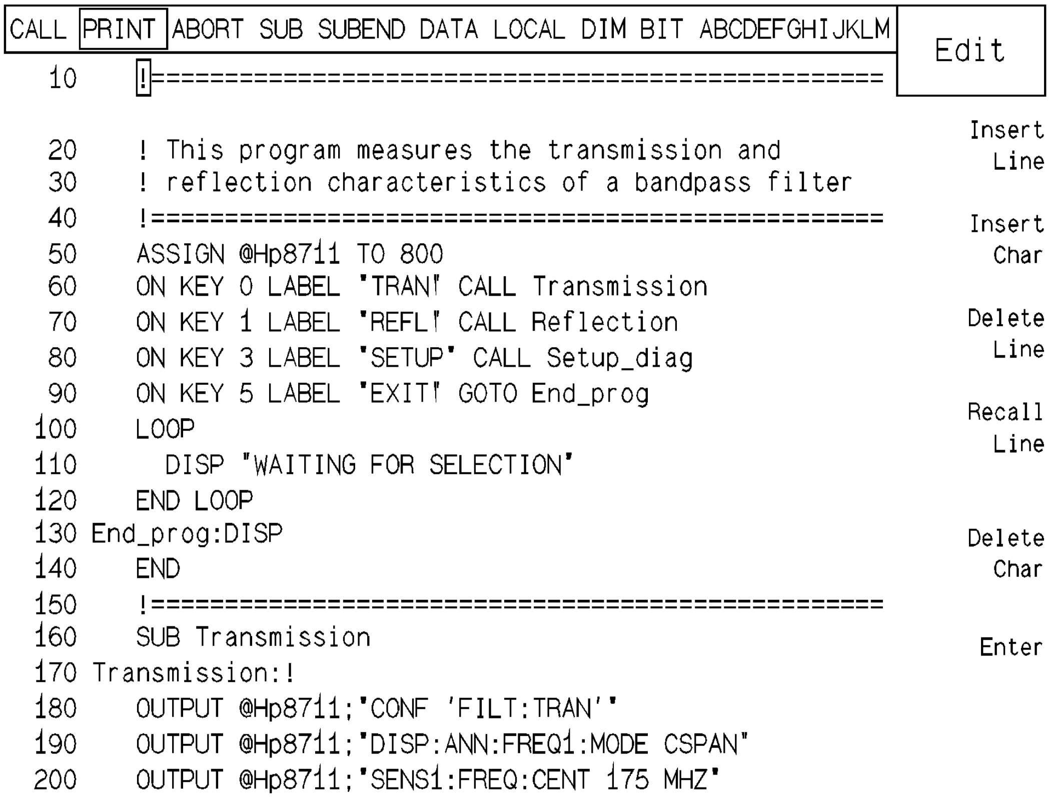
These editors are available as part of some programming language products and as individual products.

Editing Your Program Using Edit

The built-in editor may be used for creating and altering lines in an IBASIC program. Those familiar with the editor found in HP BASIC will find it similar to the instrument's IBASIC editor. It is easy to learn and use.

To start the editor, press the **Edit** softkey in the SYSTEM OPTIONS **IBASIC** menu or F4 on an external keyboard. You will see the program appear on the display with a cursor on the first line of the program, as shown in Figure 5-1. If the program buffer is empty, the first line number `10` appears with the cursor positioned to begin entering text.

Figure 5-1 The HP IBASIC Program Editor



```
 10     !==================================================

 20     ! This program measures the transmission and
 30     ! reflection characteristics of a bandpass filter
 40     !==================================================
 50     ASSIGN @Hp8711 TO 800
 60     ON KEY 0 LABEL "TRAN" CALL Transmission
 70     ON KEY 1 LABEL "REFL" CALL Reflection
 80     ON KEY 3 LABEL "SETUP" CALL Setup_diag
 90     ON KEY 5 LABEL "EXIT" GOTO End_prog
100     LOOP
110       DISP "WAITING FOR SELECTION"
120     END LOOP
130 End_prog:DISP
140     END
150     !==================================================
160     SUB Transmission
170 Transmission:!
180     OUTPUT @Hp8711;"CONF 'FILT:TRAN'"
190     OUTPUT @Hp8711;"DISP:ANN:FREQ1:MODE CSPAN"
200     OUTPUT @Hp8711;"SENS1:FREQ:CENT 175 MHZ"
```

The analyzer editor is accompanied by a "Label Window" at the top of the screen. This window is filled with characters and IBASIC keyword commands and has its own cursor.

The current program line (the line containing the cursor) always appears as two lines on the screen, allowing you to enter up to 108 characters if needed. All other lines have only their first 51 characters displayed (excluding line numbers).

Each line has a numeric field in the first 6 columns in which program line numbers are right justified. Although program lines are automatically numbered by the editor, you can edit the current line number to copy or move it to a different location in the program. The range of line numbers is from 1 to 32767. To end an editing session, press the **Prior Menu** softkey in the edit menu or (F8) on an external keyboard. This will return you to the (SYSTEM OPTIONS) **IBASIC** menu.

The IBASIC Editor Softkeys

The editor has two sets of softkey menus, the **Edit** keys and the **Character Entry** keys. The edit menu is activated when you press (SYSTEM OPTIONS) **IBASIC** **Edit** . The menu box above the softkeys shows the label **Edit**.

The edit menu provides the following softkeys:

Insert Line	((F1))
Insert Char	((F2))
Delete Line	((F3))
Recall Line	((F4))
Delete Char	((F5))
Enter	((F6))
	((F7))
Prior Menu	((F8))

The character entry menu is described "Editing from the Front Panel" on page 5-11.

Recording into an Existing Program

One way to enter lines into your program is to use the keystroke recording capabilities of IBASIC. To record measurement sequences or other front panel operations into your program, follow the procedure described below.

1. Activate the editor by pressing (SYSTEM OPTIONS) **IBASIC** **Edit**.
2. Use the step keys on the analyzer or the cursor keypad on an external keyboard to position the cursor on the line above where you want to insert the recorded statements.
3. Press **Prior Menu** to exit the editor.
4. Press **Key Record on OFF** to activate keystroke recording.
5. Record the measurement sequence or front panel operation.
6. Press **IBASIC** **Key Record on OFF** to conclude the recording session.

The inserted recording acts the same as if you had pressed **Insert Line** in the editor, and generated `OUTPUT` statements in insert mode.

NOTE An `ASSIGN` statement is required. The `ASSIGN` statement (for example, `ASSIGN; @Hp8711 to 800`) is *not* generated when you are recording into an existing program and *must* be included in your program prior to any recorded `OUTPUT` commands. If you initially created the program using recording, this statement should already exist. If it does not exist, you will need to enter it.

Editing with an External Keyboard

With an external keyboard connected to the analyzer, it is easy to edit or create an IBASIC program using the internal editor. Note that the front panel editor described in the next section is always available, even when an external keyboard is in use.

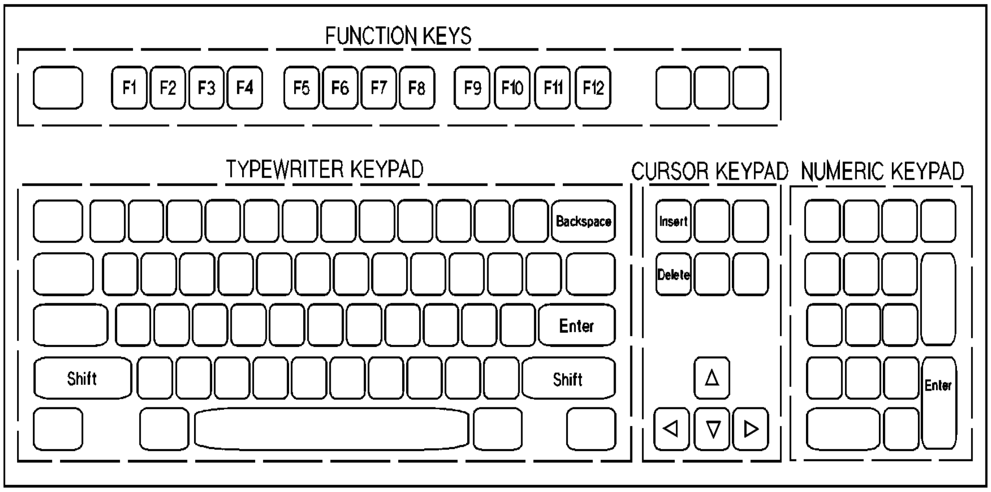
NOTE

The external PC-AT compatible keyboard requires a mini-DIN connector The analyzer and the IBASIC editor work with IBM PC-AT compatible keyboards (US only) that have a mini-DIN connector. Non-US language keyboards will not cause an error, they simply will not be recognized as different from the US keyboard. A compatible keyboard can be purchased by ordering option 1CL with the analyzer. Keyboards with a standard DIN connector will need a mini-DIN to DIN adapter, part number 1252-4141.

The PC-AT keyboard, Figure 5-2, has four major key areas:

Typewriter keypad	Used to enter text and numeric characters.
Numeric keypad	Used to enter numeric characters.
Cursor keypad	Used to move to cursor up/down a line, or left/right to the next or previous character position.
Function keys	F1 through F8 correspond to the analyzer’s eight softkeys.

Figure 5-2 The PC Keyboard



Installing an External Keyboard

1. Remove power from the analyzer.
2. Connect the keyboard to the rear panel DIN KEYBOARD connector.
3. Turn on power to the analyzer.
4. Test the keyboard:
   a. Select the (SYSTEM OPTIONS) **IBASIC** **Edit** menu and use the cursor keypad to position the cursor within the program for editing operations.
   b. Use the Page Up and Page Down keys on the keyboard to scroll through the program listing

Inserting Lines

Insert one or more program lines above an existing line by placing the cursor on that line and pressing (Shift) + (Insert) on the keyboard. This key combination functions as a toggle to turn insert mode on and off.

As an example, assume you want to insert some lines between two adjacent program lines numbered 90 and 100. Place line 100 in the current line position and press (Shift) + (Insert). The program display "opens" and a new line, number 91, appears between line 90 and line 100. Enter the inserted line and another inserted line, number 92, will appear. If, after continuing to enter lines in this manner, the inserted line number increments to 100, then the current line 100 will be incremented one higher to accommodate the inserted line.

To stop inserting lines, press (Shift) + (Insert) again or use the cursor keys to move to another program line. Make sure you have entered any changes to your final inserted line (with the **Enter** key) before exiting the insert mode.

CAUTION

Any changes you have made to the current line will be lost if you move the cursor to another line without pressing **Enter**.

Editing Lines

Use the cursor keypad on the keyboard to move around the program for editing. The left and right arrow keys move within a program line and the up and down arrow keys move between lines. The alphanumeric keypad on the keyboard can be used for entering or editing text. The Delete key deletes the character highlighted by the cursor.

When you finish editing or changing a program line, store it into the program by pressing **Enter** on the keyboard. The computer checks the line for syntax errors and converts letter case to the required form for names and keywords (IBASIC commands). If no errors are detected, it stores the line in the program buffer.

Entering Program Lines

When you finish entering or changing a program line, it must be stored into the program buffer by entering it in one of four ways:

1. Use the ENTER key on the front panel of the analyzer.
2. Use the **Enter** softkey on the instrument.
3. Use the Enter or Return key on the external keyboard.
4. Use the function key on the keyboard (F6) that represents the analyzer's **Enter** softkey.

The computer checks the line for syntax errors and converts letter case to the required form for names and keywords (IBASIC commands).

If no errors are detected, it then stores the line.

CAUTION

If you move to another line without pressing ENTER, changes are lost. If you edit or enter text on the current program line and then move off the line without pressing ENTER, all editing on the line will be lost.

Accessing the IBASIC Command Line

When an external keyboard is attached, pressing the ESC key will toggle the IBASIC command line on or off. The IBASIC command line is active when a box shaped cursor is displayed near the lower left-hand corner of the display.

Editing from the Front Panel

Use the step keys to move the cursor up and down to select the line in the program. With the cursor located at the beginning of the line you want to change, use the knob to position the cursor within the line.

Character Entry

The character entry menu and the associated label window are activated by pressing the **Insert Line** or **Insert Char** softkeys. The knob and step keys now move the cursor in the label window.

Use the knob or step keys to move the label window's cursor until it highlights the desired letter or keyword and press **Select Char/Word**. Continue editing until the line is correct. Press **Enter**. The computer checks the line for syntax and then stores it in the program if the syntax is correct. Press **Prior Menu** to return to the edit menu.

The character entry menu provides the following softkeys:

Select Char/Word	(F1)	Inserts the character or word highlighted by the label window cursor at the position marked by the program cursor.
Space	(F2)	Inserts a space at the position marked by the program cursor.
Delete Char	(F3)	Deletes the character highlighted by the program cursor.
Backspace	(F4)	Deletes the last character before the program cursor.
Enter	(F5)	Enters the edited program line.
Prior Menu	(F8)	Returns to the edit menu and de-activates the label window.

The Label Window

The label window is a scrolling list of the most common characters, symbols, and keywords used in IBASIC programming. It contains the uppercase alphabet, the numbers 0 to 9, symbols such as single and double quotation marks, parentheses, signs for mathematical and string operations as well as numerous other characters and symbols.

It also contains the following IBASIC keywords:

ABORT	ENTER	NOT
ASSIGN	FOR	OUTPUT
BIT	GOTO	PAUSE
CALL	IF	PRINT
CLEAR	INPUT	SUB
DATA	INTEGER	SUBEND
DIM	LIST	THEN
DISP	LOCAL	TO
END	NEXT	WAIT

Inserting Lines

To insert one or more program lines above any existing line, place the cursor on the existing line and press **Insert Line**. This causes the cursor to move to a new line that appears above the existing one. Enter and store the inserted line and another inserted line will appear. Remember, each line must be ENTERed or any changes will be lost when the cursor is moved to a different line.

Removing Program Text

You can remove individual characters or entire lines with the editor.

Deleting Characters

The **Delete Char** softkey removes the character under the cursor and moves all characters to the left one place. Repeatedly pressing **Delete Char** will cause text to the right of the cursor to be removed one character at a time. The **Delete Char** softkey works the same way in both the line number and program statement fields. When used in the line number field, it deletes only line numbers to the right of the cursor (not program statement characters).

When using an external keyboard, there are other keys that perform the same function as the **Delete Char** softkey. These are the (Delete) key in the cursor keypad and the function key that maps to the appropriate softkey, (F5) for the edit menu or (F3) for the character entry menu.

Another way to remove text on a line is by backspacing. Pressing the (– / <—) hardkey or the **Backspace** softkey on the front panel of the analyzer removes the letter to the left of the cursor and moves the cursor (and all characters to the right of the cursor) one space to the left. The (F4) function key or the **Backspace** key on the typewriter keypad of the external keyboard perform the same function. When the cursor is on a line number, using backspace simply moves the cursor back one position without deleting the number.

Deleting Lines

The **Delete Line** softkey allows you to remove the current program line. When the current program line disappears, all subsequent lines in the display move up one line, but are not renumbered. The cursor stays in the same column and moves to the next-highest numbered line.

If **Delete Line** is pressed when the cursor is on the last program line, the line text is removed but the line number remains with the cursor in the first column of line. This puts the editor in insert mode on the last line of the program (see “Inserting Lines”). (To get out of insert mode, move the cursor up one line with the arrow key.)

Pressing **Delete Line** will *not* remove a subprogram line with the SUB keyword in it unless all program lines belonging to that subprogram have already been deleted. A block of program lines can be deleted by executing the command `DELETE x,y` from an IBASIC command line (where `x` is the first line number in the block and `y` is the last line number).

When using an external keyboard, there are other keys that perform the same function as the **Delete Line** softkey. These are Shift + Delete in the cursor keypad, and the function key (F3) that maps to the **Delete Line** softkey in the edit menu.

Recalling a Deleted Line

The last line that was deleted using **Delete Line** is saved in the analyzer. To recall this line, press the **Recall Line** softkey or F4 on an external keyboard. Press **Enter** to restore the line to the program.

Renumbering, Copying, Moving, and Indenting Lines

If you want to change the line number of an edited program line, move the cursor to the line number field and enter the line number you want. Changing the line number causes a copy operation, not a move. Therefore, if you only want to move the line, change the line number first, press **Enter** and then delete the original line. If you want to create an edited copy of the current line, edit the line and then change the line number and press **Enter**. The edits will only appear in the copied line.

If you are inserting a program line and you change the line number, the line will move to its new location when you ENTER it. The editor will remain in insert mode at the new location in the program.

You will notice that when the cursor is in the line number field, entries operate in an overtype fashion rather than in the insert fashion as in the text portion of the program line. Also the ← (backspace) key moves the cursor over line numbers without deleting the number.

NOTE

Renumbering the entire program with RENumber. To renumber the entire program, IBASIC supports the RENumber command *but* you need an external keyboard to execute it. The command can be executed by following the steps listed below.

1. Exit the edit mode by pressing **Prior Menu** until the (SYSTEM OPTIONS) **IBASIC** menu is active.
2. Partition an IBASIC display as described next in this chapter, or press the **Esc** (Escape) key on the keyboard to enable the command line.
3. Enter the command `REN x,y` (where x is the new beginning line number and y is the increment) from the command line of the IBASIC display
4. Another way to "renumber" program lines with an external keyboard is to use the `COPYLINES` and `MOVELINES` commands. Use the `INDENT` command to make your code more readable.

Using IBASIC Display

Pressing the SYSTEM OPTIONS **IBASIC** **IBASIC Display** softkey (F7 on an external keyboard) allows you to allocate a partition of the analyzer's display to be used by your program or, alternately, to return any allocated partition to the analyzer.

The analyzer display is divided into two small partition areas (Upper and Lower) or one large area (Full), which encompasses both the Upper and Lower partition areas. See Figure 5-3 on page 5-18.

All screen output commands, such as PRINT and DRAW, require that you allocate a partition of the screen in order to view the results of the command. This can be performed in your program or interactively using the **IBASIC Display** softkey. Allocating display partitions can be accomplished from within your program using the SCPI mnemonic "DISP:PROG" and specifying the parameter UPPER, LOWER or FULL. For example, the statement

```
OUTPUT 800;"DISP:PROG FULL"
```

allocates the entire display, corresponding to selecting **Full** from the **IBASIC Display** menu.

An IBASIC display partition cannot occupy the same location as a measurement channel display. When an IBASIC display is partitioned, it limits the amount of the screen available to simultaneously show measurement data. Table 5-1 shows the **IBASIC Display** menu softkeys, their corresponding SCPI mnemonics, their functions, and the measurement data that can be viewed when the display partition is allocated.

Table 5-1 IBASIC Display Partitions

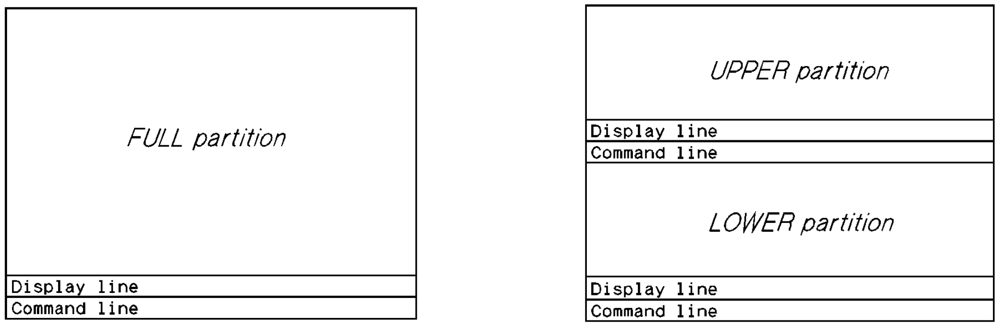
Softkey	SCPI Mnemonic	Allocates	Visible Data
None	`DISPlay:PROGram OFF`	No Display	Measurement Channels 1 and 2
Full	`DISPlay:PROGram FULL`	The Whole Display	None
Upper	`DISPlay:PROGram UPPer`	Upper Measurement Channel Area	Measurement Channel 2 only
Lower	`DISPlay:PROGram LOWer`	Lower Measurement Channel Area	Measurement Channel 1 only

NOTE Split-screen format is automatically selected if UPPER or LOWER is selected. When the UPPER or LOWER display partition is selected, the measurement display automatically selects the “split-screen” format. This format uses half of the screen to display each measurement channel's measurement data. Measurement channel 1 data is always shown on the upper half of the screen, and measurement channel 2 data is shown on the lower half. The split-screen format allows measurement data to be viewed simultaneously with IBASIC program output. For more information about the split-screen format, or other parts of the measurement display, refer to your analyzer's *User's Guide*.

Display allocation should be managed by your program using SCPI commands. The softkeys are best used during program development.

An IBASIC partition can be very useful during program development. It can be used to view program output, to query variables, and to execute IBASIC commands (such as `GET` and `REN`) outside of your program. Figure 5-3 on page 5-18 shows the relative size and location of the different IBASIC partitions and their command and display lines.

Figure 5-3 The IBASIC Display Partitions



More information about using display partitions within a program is available in Chapter 7, “Graphics and Display Techniques.”

Using UTILITIES

Pressing the SYSTEM OPTIONS **IBASIC** **Utilities** (F6 on an external keyboard) allows you to clear the program buffer, allocate memory for program use, or secure your program.

- Clear Program (F1)
- Stack Size (F2)
- Secure (F3)

Executing **Clear Program** erases the current program buffer and frees all memory currently allocated. Memory size (see below) is reset to 8192 bytes. You will be prompted to ensure you do not accidentally erase the program.

Stack Size allows you to set the stack memory to be used by your program. At power up it is set by default to 8192 bytes. However, when a program is RUN, the analyzer will try to automatically set the memory size large enough to accommodate the program's stack and COM memory requirements.

For some programs the automatic memory sizing will be too small and you will get the message:

```
Error 2 in 100 Memory overflow
```

When this error occurs, you must manually set the **Stack Size** to the value in bytes required by your program, up to the available memory in your system.

NOTE

The total amount of memory available for IBASIC and VolRAMdisk may change from firmware revision to revision. To determine the current available memory, press SAVE RECALL **Select Disk** **Configure VOL_RAM**. A memory report will appear on the analyzer's display.

Secure is used to secure lines of your program. Secured lines cannot be listed, edited, or displayed. After you press this key you will see the following:

- Start Line # (softkey 1)
- End Line # (softkey 2)
- Perform Secure (softkey 4)

  After you have set the start and stop line numbers, execute the **Perform Secure** operation.

CAUTION Do not secure the *only* copy of a program. Once you have secured your program lines, there is no way to remove the security. Therefore, do not secure the only copy of your program. Make a copy of your original program, **Secure** the copy, and keep the original in a safe place. This prevents unauthorized users from listing your program.

6 Debugging Programs

The process of creating programs usually involves correcting errors. You can minimize errors by using keystroke recording for measurements and other front panel sequences and by writing structured, well-designed programs.

IBASIC includes features that can help you find problems in a program. You can do the following:

- RUN or CONTINUE your program
- STEP through your program, executing one line at a time
- display the last error encountered in your program
- examine program variables

By examining the values assigned to variables at various places in the program, you can get a much better idea of what is really happening in your program.

By inserting a PAUSE statement in your program, you can pause the program at any line and then examine the values of variables at that point in the program. You can then press **Continue** in the (SYSTEM OPTIONS) **IBASIC** menu to resume operation to the next PAUSE statement (or the program end).

These capabilities can be used together to effectively examine the program's operation and solve your particular problems.

NOTE

An external keyboard is required for effective program debugging. Most of the debugging techniques described in this chapter make use of an external keyboard. The analyzer and the IBASIC editor work with IBM PC-AT compatible keyboards (US only) that have a mini-DIN connector. Non-US language keyboards will not cause an error, they simply will not be recognized as different from the US keyboard. A compatible keyboard can be purchased by ordering option 1CL with the analyzer. Keyboards with a standard DIN connector will need a mini-DIN to DIN adapter, part number 1252-4141.

Setting Breakpoints

A common method of debugging a program involves the use of breakpoints. A breakpoint causes the program to stop before executing a specified line so that you can examine the program state at that point. In IBASIC this can be accomplished by inserting PAUSE statements in the program code. Note that PAUSE is one of the IBASIC keywords included in the editor's label window (described in Chapter 5, “Developing Programs”). When the program is then run, you can use the command line to check or change variable values.

Execution of the program can be resumed in one of two ways.

- Press **Step** (F3 on an external keyboard) to execute the next program line.
- Press **Continue** (F2 on an external keyboard) to continue the program until the next PAUSE, STOP or END statement is encountered.

Examining Variables

To examine a variable, it is necessary to pause the program. Pause the program by pressing the **Pause** softkey (F8 on an external keyboard) when a program is running, or by inserting a PAUSE statement in your program.

A command line becomes active when an IBASIC program is paused or stopped and an IBASIC display partition is present. (For information on creating an IBASIC display partition, see "Using IBASIC Display" on page 5-16.) You may also activate the command line when no IBASIC window is partitioned by pressing the ESC key on the external keyboard. A cursor will appear in the lower left portion of the screen when the command line is active. Strike the ESC key again to de-activate. Once the command line is active, a variable can be examined in two ways. Both methods require the use of an external keyboard.

1. Enter the variable name (without a line number) on the command line. This results in the value assigned to that variable being shown in the display line of the IBASIC window.
2. Execute the command `PRINT Value` from the command line (where `Value` is the name of the variable being examined). This results in the value assigned to that variable being shown on the print screen of the IBASIC window.

To examine a variable without accessing a command line, it is necessary to add the statement `PRINT Value` (or `DISP Value`) to the program before the PAUSE statement that temporarily stops the program. PAUSE, PRINT and DISP are all keywords that are included in the IBASIC editor's label window (see Chapter 5, "Developing Programs," for a description of the label window).

NOTE

An IBASIC display partition must be active to view the results of a PRINT statement or to access a command line. The display line (accessed with the DISP command) is available even when no IBASIC display is present.

Examining Strings

Enter string variables as you would any other variable. Any string variable entered without delimiters will display as much of the string as will fit on the display line of the screen (up to 58 characters).

To select only a section of a string, use the IBASIC substring syntax (see the “HP Instrument BASIC Programming Techniques” section of the *HP Instrument BASIC Users Handbook*). For example, to examine the 7 character substring starting at the second character of A$, enter `A$[2;7]` on the command line or execute the command `PRINT A$[2;7]`.

Examining Arrays

To select an array to be examined, you can either select individual elements or the entire array. For example, the following entry:

```
I_array(1),I_array(2),I_array(3)
```

selects the elements 1 through 3 of the array `I_array` to be displayed.

You may select an entire array to be examined by entering the array variable name and specifying a wildcard (*) for the element (such as `I_array(*)`). If I_array(20) is an integer array, and the first and second elements are set to 100, entering `I_array(*)` would display the following:

```
100 100 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

Individual array elements (e.g., I_array(17)) can also be specified in the same way as any other single variable.

Displaying the Last Error Encountered

It is sometimes useful to review the last error encountered by a program that is being run. This is done from the command line by examining the value assigned to the variable name ERRM$. This value will include the error number and message of the last error encountered by the program.

An additional method of displaying the error message is to use an error trapping subroutine.

For example, insert the following line at the beginning of a program.

```
ON ERROR GOSUB Errormsg
```

The subroutine Errormsg should then be included at the end of the program (after execution is stopped but before the END command).

```
100    Errormsg:!
110    DISP ERRM$
120    PAUSE
130    RETURN
```

The error message is automatically shown on the display line of the IBASIC window and program execution is paused when an error message is encountered.

You may also encounter SCPI errors in addition to IBASIC errors. SCPI errors can occur when a command syntax is unrecognized or incorrect. For more details on SCPI errors, refer to the *Programmer's Guide*.

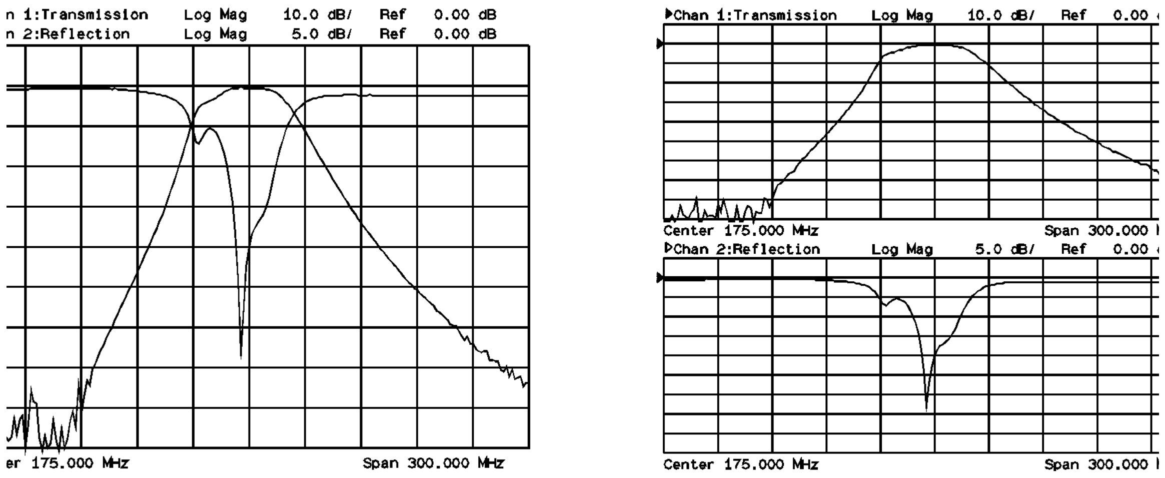
7 Graphics and Display Techniques

Display Partitions

IBASIC programs have the ability to allocate portions of the instrument's display for program output, including text and graphics. This section provides a description of the various programming techniques used to do both. Any of three measurement display areas, called display partitions, can be used by an IBASIC program. These partitions are shown in Figure 7-1.

The analyzer has two measurement channels which can be displayed simultaneously. The instrument's screen can be split into two trace areas for this purpose (upper for measurement channel 1 and lower for measurement channel 2). Additionally, the two measurements can be overlaid onto one full size screen (the default setting). For more information, refer to the *Automating Measurements User's Guide Supplement* for more information.

Figure 7-1 Display Partitions on the Analyzer


n 1:Transmission Log Mag 10.0 dB/ Ref 0.00 dB
n 2:Reflection Log Mag 5.0 dB/ Ref 0.00 dB
er 175.000 MHz
Span 300.000 MHz
▶Chan 1:Transmission Log Mag 10.0 dB/ Ref 0.00
Center 175.000 MHz
Span 300.000
▷Chan 2:Reflection Log Mag 5.0 dB/ Ref 0.00
Center 175.000 MHz
Span 300.000

Using the Display Partitions

Many IBASIC commands (such as `PRINT`, `DISP`, `CLEAR SCREEN`, `MOVE`, `DRAW` and `GCLEAR`) require a display as an output device. These commands output data to the screen by writing to a screen buffer. Since IBASIC programs share all the hardware resources with the instrument, the display must be shared for instrument and program use.

In order to view this output buffer, a portion of the display must be released from the instrument. When no program is running, you can do this manually using the SYSTEM OPTIONS **IBASIC** **IBASIC Display** softkey menu. To do this within a running program requires sending a command to the analyzer both to borrow a part of the display and again to return it for the instrument's use.

This process is called *allocation of display partitions*. Manual allocation of display partitions is described in Chapter 5, "Developing Programs." Table 7-1 below includes a summary of the available partitions, their locations and the SCPI mnemonic used to select each partition.

Table 7-1 IBASIC Display Partitions

Softkey	SCPI Mnemonic	Allocates
None (F1)	`DISPlay:PROGram OFF`	No Display
Full (F2)	`DISPlay:PROGram FULL`	The Whole Display
Upper (F3)	`DISPlay:PROGram UPPer`	Upper Measurement Channel Area
Lower (F4)	`DISPlay:PROGram LOWer`	Lower Measurement Channel Area

Allocating Display Partitions

To request a display partition from the analyzer for use by an IBASIC program, send the instrument the corresponding SCPI mnemonic.

`DISP:PROG UPPer`	Allocates upper partition
`DISP:PROG LOWer`	Allocates lower partition
`DISP:PROG FULL`	Allocates full-screen partition

For example, to print a message to the upper partition area, you might use a program segment like this:

```
30    ASSIGN @Hp8711 TO 800
40    OUTPUT @Hp8711;"DISP:PROG UPPer"
50    CLEAR SCREEN
60    PRINT "This is the upper partition"
```

To be sure that you are not writing to a partition that has not yet been assigned, you should include a `WAIT` statement or, preferably, add a SCPI query command followed by an `ENTER` statement to synchronize the program with the instrument. The previous example might then look like this:

```
30    ASSIGN @Hp8711 TO 800
40    OUTPUT @Hp8711;"DISP:PROGUPPer"
42    OUTPUT @Hp8711;"DISP:PROG?"
44    ENTER @Hp8711;Screen$
46    IF Screen$<>"UPP" THEN GOTO 42
50    CLEAR SCREEN
60    PRINT "This is the upper partition"
```

The mnemonic `DISP:PROG?` (line 42 above) requests the instrument to send the current partition status. The `ENTER` statement on the next line reads that status, assigns the value to the variable `Screen$`, and continues.

De-Allocating Display Partitions

To return the display partition to the analyzer for use as a measurement screen, use the “`DISP:PROG OFF`” mnemonic. This should be done before the termination of any program that has allocated a display partition. It may also be required within the program to allow the user to view instrument measurement data. The following example demonstrates this command:

```
   830   OUTPUT @Hp8711;”DISP:PROG OFF”
```

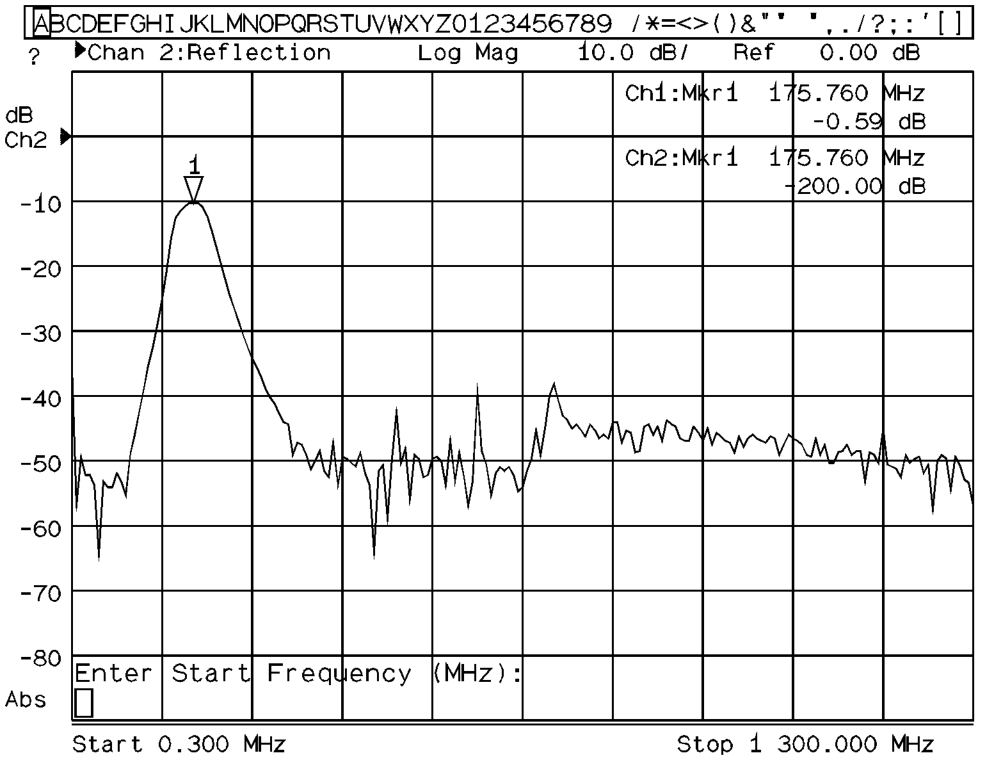
Operation with No Display Partition

IBASIC programs can also access the analyzer's display when no partition has been allocated. This can be done through the use of certain areas of the screen. One of these areas is the area reserved for softkey labels. It is accessed using the `ON KEY` statement.

A second area is a display line (or command line) that appears when no part of the display is allocated for use by IBASIC. This display line, which is located at the lower left corner of the active measurement channel graticule, appears when needed by the `INPUT` or `DISP` commands or when activated. To activate the command line, press ESC on an external keyboard.

Figure 7-2 on page 7-6 shows an example of the use of this display line. When the `INPUT` command is being used, the IBASIC editor's label window and character entry softkey menu appear. Refer to Chapter 5, “Developing Programs,” for a description of the IBASIC editor.

Figure 7-2 **Using INPUT with No Display Partition**



In addition to the commands described above, the analyzer has "User Graphics" commands that can write to any of the display partitions. These commands can be used to write to measurement windows as well as the IBASIC window. These commands are described in "SCPI Graphics Commands" on page 7-16.

Displaying Text

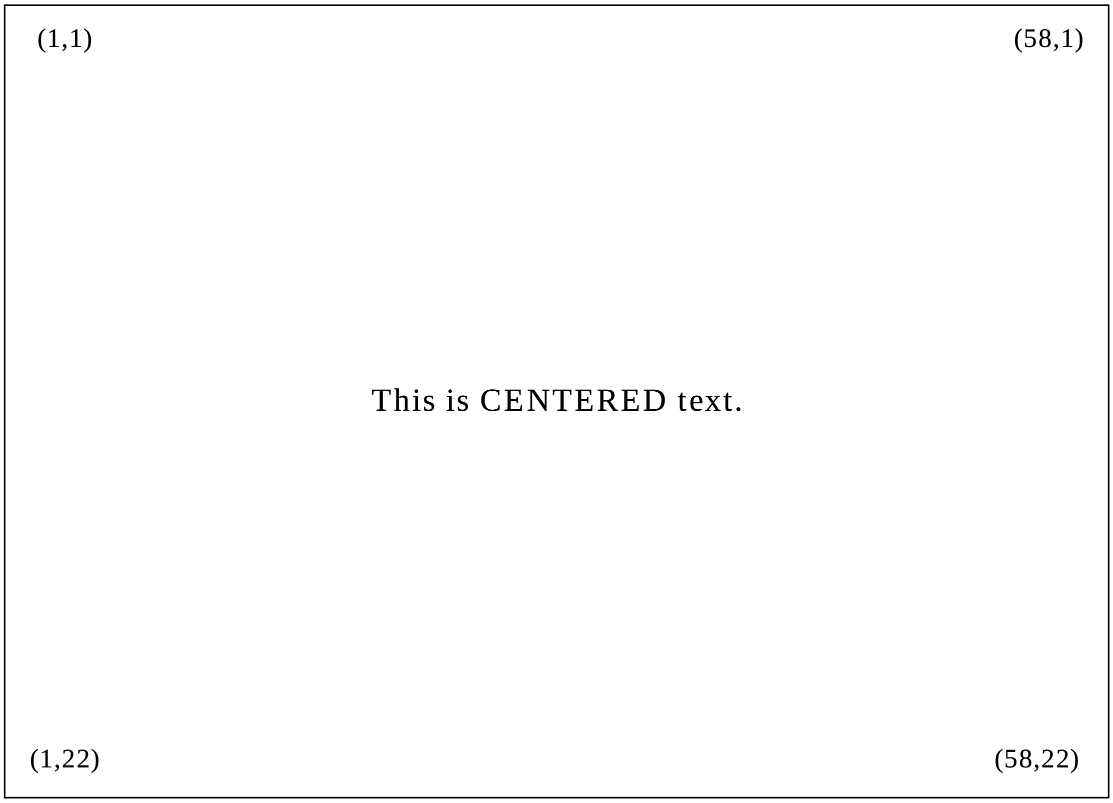
Most of IBASIC's text capabilities are covered in detail in the “HP Instrument BASIC Programming Techniques” section of the *HP Instrument BASIC Users Handbook*. The PRINT statement works the same way in every display partition. Information is printed starting at the top left corner of the current partition and continues until the display line of the partition is reached. The screen then scrolls up to allow additional lines to be printed. Figure 7-3 on page 7-8 shows the different display partitions and the location of text printed to them. Note that causing the screen to scroll does not affect any graphics displayed on the screen, since text and graphics are written to different planes of the display.

All partitions have a width of 58 characters. The height varies according to partition. Both upper and lower partitions contain 10 lines, while the full partition contains 22 lines.

Display partition coordinates are important if you are using the PRINT TABXY statement to position text. For example, the following program segment prints a message in the center of the full partition (assuming it has been allocated earlier in the program). See Figure 7-3, below.

```
100   Maxlines=22
110   Tabx=(58-LEN(“This is CENTERED text.”))/2
120   PRINT TABXY(Tabx,Maxlines/2);”This is CENTERED text.”
```

Figure 7-3 Printing to a Display Partition



Getting Text to the Screen Quickly

A useful technique to get text onto the screen quickly is to write your display message to a long string using the OUTPUT statement, and then print the string to the screen. For large amounts of text, this speeds up screen display time considerably. The following program segment demonstrates this:

```
60     DIM Temp$[100],Big$[2000]
70     OUTPUT Big$;"This is the first line of text"
80     OUTPUT Temp$;"This is the second line of text"
90     Big$=Big$&Temp$
100    PRINTER IS CRT; WIDTH 2000
110    PRINT Big$
```

The OUTPUT statements in this example are used to copy each line of the message into the variable Temp$ and append a carriage return.

You can also print to the screen using the OUTPUT statement in conjunction with the display address (1). For example, line 120 below writes a string to the screen.

```
120    OUTPUT CRT;"OUTPUT 1 WORKS WELL TOO"
```

Pop-Up Message Windows and Custom Annotations

From your IBASIC program, you can replace instrument annotations with user-defined annotations. You can change the X-axis labels and measurement channel annotations to customize the display. Pop-up messages can also be used to display permanent or temporary messages. Refer to the *Automating Measurements User's Guide Supplement*.

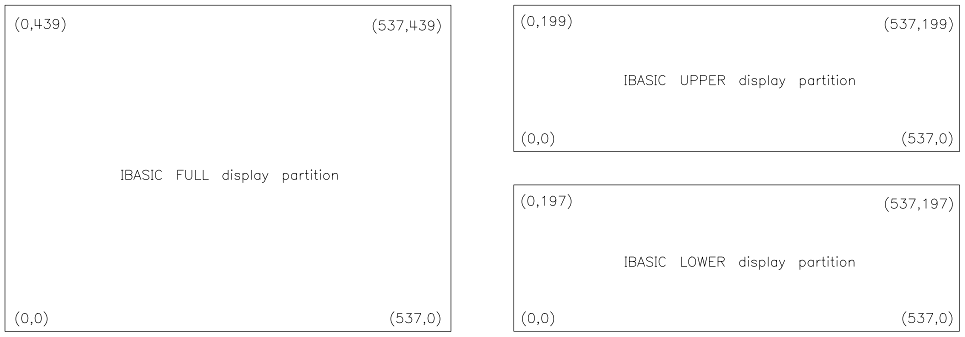
Graphics Initialization and Scaling

In all partitions, display coordinate 0,0 is at the bottom left corner and the image is cropped to fit the display if the X,Y coordinate exceeds the displayable range of the current partition. Figure 7-4 shows the different partitions and the pixel dimensions (GESCAPE values) for each.

After a GINIT command, the display is dimensioned as 100 GDUs (Graphical Display Units) high and 122 GDUs wide (assuming a full partition). This gives a RATIO (aspect ratio) result of 1.22 and provides the same results as issuing a WINDOW 0,122,0,100 command. To prevent circles from appearing oval in shape, this ratio should be maintained. You can also issue a WINDOW 0,537,0,439 command. This will maintain the same ratio but the display will now be dimensioned in actual pixel units. This may be more useful than the default GINIT values since fractional display units are not needed; it allows integers only to be used, thus speeding execution. These are also the same values that are returned by using the GESCAPE command (see BARCODE program example). The GESCAPE command will always set the current pixel dimension sizes. Because the results of this command can vary drastically with partition size, you must first partition the display *before* executing the GINIT and GESCAPE commands.

NOTE Upon power up, the default display coordinates are 0,537,0,439 and will remain that until a GINIT is performed. It is recommended that a GINIT command always be part of any graphics program and that it be executed only after the display partition is set.

Figure 7-4 Pixel Dimensions with Available Display Partitions


(0,439)
(537,439)
IBASIC FULL display partition
(0,0)
(537,0)
(0,199)
(537,199)
IBASIC UPPER display partition
(0,0)
(537,0)
(0,197)
(537,197)
IBASIC LOWER display partition
(0,0)
(537,0)

hp61c

Using Graphics

IBASIC graphics commands are easy to understand and use. You can use the `MOVE` statement to move the “pen” to a specific pixel location (without drawing) and then draw a line from the current pen location to another pixel coordinate using the `DRAW` statement. The `GCLEAR` statement removes all graphics.

The `PEN` command provides an easy method of erasing lines drawn by the `DRAW` command. When `PEN 1` is issued (the default state), all `DRAW` commands act normally, drawing a line with the full intensity. When `PEN 0` is issued, all `DRAW` commands erase any pixels their path encounters. Where there are no lines in the path, no change is visible. As an example of using the `MOVE` and `DRAW` commands, the following statement moves the logical pen to a point 100 units to the right of, and 150 units above, the lower left corner of the display:

```
100   MOVE 100,150
```

This statement then draws a line to coordinates (200,10):

```
110   DRAW 200,10
```

Finally, these two statements erase the previously drawn line:

```
120   PEN 0
130   DRAW 100,150
```

Although text and graphics appear together, you can clear them separately. Use `CLEAR SCREEN` to clear the text. Use `GCLEAR` to clear the graphics.

Drawing Figures

Some IBASIC keywords listed below may be used to simplify drawings and setup diagrams. See “Graphics Exceptions” on page 7-14.

POLYGON	Draws all or part of a regular polygon.
RECTANGLE	Draws a rectangle.
LABEL	Produces alphanumeric labels.
CSIZE	Sets size and aspect ratio of labels.
LDIR	Defines the angle at which a label is to be drawn.
LORG	Defines the relative origin of a label.

These keywords are used in the “BARCODE” program example listed in the *Example Programs Guide*, and on the *Example Programs Disk*. The keywords appear in the subprograms “Box”, “Circle”, and “Label” described below.

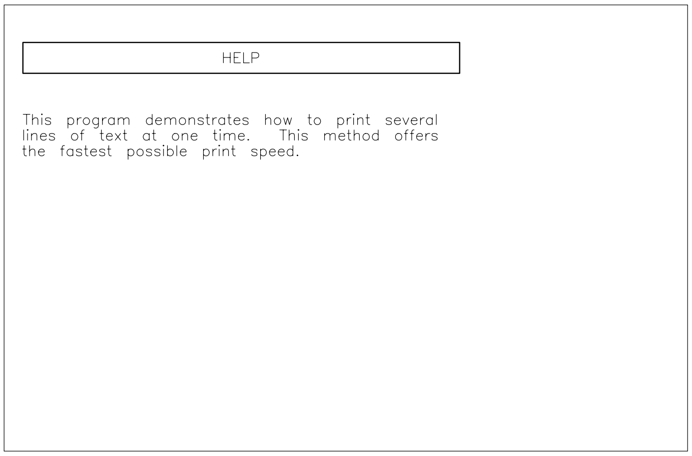
```
1700 Box:SUB Box(Xpos,Ypos,Xsize,Ysize)
1710     COM /Scale/ Sc,INTEGER X,Y
1720     MOVE X+(Xpos-Xsize/2)*Sc,Y+(Ypos-Ysize/2)*Sc
1730     RECTANGLE Xsize*Sc,Ysize*Sc
1740  SUBEND
1750  !
1760 Circle:SUB Circle(Xpos,Ypos,Radius)
1770     COM /Scale/ Sc,INTEGER X,Y
1780     MOVE X+Xpos*Sc,Y+Ypos*Sc
1790     POLYGON Radius*Sc,16,16
1800  SUBEND
1810  !
1820 Connect:SUB Connect(X1,Y1,X2,Y2,How)
1830     COM /Scale/ Sc,INTEGER X,Y
1840     MOVE X+X1*Sc,Y+Y1*Sc
1850     SELECT How
1860     CASE 1      !...diagonal
1870        DRAW X+X2*Sc,Y+Y2*Sc
1880     CASE 0
1890        DRAW X+X1*Sc,Y+Y2*Sc
1900        DRAW X+X2*Sc,Y+Y2*Sc
1910     CASE -1
1920        DRAW X+X2*Sc,Y+Y1*Sc
1930        DRAW X+X2*Sc,Y+Y2*Sc
1940     END SELECT
1950  SUBEND
1960  !
1970 Label:SUB Label(Text$,Xpos,Ypos,Size,Lorg,Ldr,Pen)
1980     COM /Scale/ Sc,INTEGER X,Y
1990     LORG Lorg
2000     LDIR Ldr
```

```
2010     CSIZE Size*Sc,.55
2020     MOVE X+Xpos*Sc,Y+Ypos*Sc
2030     PEN Pen
2040     LABEL Text$
2050     PEN 1
2060  SUBEND
2070  !
```

The following program displays a “HELP” screen and demonstrates many of the techniques discussed so far. Running this program produces the screen display shown in Figure 7-5.

```
10     DIM A$[58],String$[1000]
20     ASSIGN @Hp8711 TO 800
30     OUTPUT @Hp8711;"DISP:PROG FULL;*WAI"
40     GINIT
50     GCLEAR
60     MOVE 0,89
70     RECTANGLE 200,14
80     PRINT TABXY(24,2);"HELP"
90     OUTPUT A$;"This program demonstrates how to print several"
100    String$=String$A$
110    OUTPUT A$;"lines of text at one time.  This method 
offers"
120    String$=String$A$
130    OUTPUT A$;"the fastest possible print speed."
140    String$=String$A$
150    PRINTER IS CRT;WIDTH 1000  ! Prevent auto cr/lf
160    PRINT TABXY(1,5);String$
170    END
```

Figure 7-5 “HELP” Program Output



```
HELP

This program demonstrates how to print several
lines of text at one time.  This method offers
the fastest possible print speed.
```

pd62a

Graphics Exceptions

The following graphics commands do not conform to the keyword description found in the *HP Instrument BASIC Users Handbook*:

`VIEWPORT`	Does not create isotropic units that are physically square. Does not soft clip the display area.
`CLIP`	The analyzer does not support graphics clipping.
`SHOW`	Does not create isotropic units.
`LINE TYPE`	The analyzer does not support different line types.

`POLYLINE, POLYGON, RECTANGLE, RPLOT`—The analyzer does not support the FILL or EDGE options. Also see next paragraph.

GRID, RECTANGLE, POLYGON, and POLYLINE scaling differences

When the display is initialized using `GINIT`, the display will be scaled to a height of 100 GDUs and a width of 122 GDUs.

The ratio is 1.22 and the pixel height-to-width ratio is fixed at 1.0 (square pixels).

NOTE Previous analyzer models may have scaling differences. When converting programs from previous models (such as 8711A/B and family) to run on an 8712ET/ES or 8714ET/ES, scaling differences will affect all graphics commands. The ET/ES family of network analyzers (as well as the 8711C family) have square pixels. The older “A” and “B” family network analyzers had a non-square pixel height-to-width ratio of 1.79.

Labeling with Different Partitions

The LABEL command can be used to label graphs, however, note that labels that may be of the correct size for a full screen partition will appear half as big if a GINIT is performed after the analyzer has been set to either the upper or lower half partition. This is because the CSIZE command scales according to display height, not width. Since the display height is one-half, the character size will also be one-half. Labels that are scaled properly for full screen displays will not be scaled properly for half screen displays and vice-versa.

SCPI Graphics Commands

In addition to the commands described earlier in this chapter, there are several SCPI mnemonics that can be used to create graphics and messages on the display of the analyzer.

These commands are instrument-specific mnemonics, not standard IBASIC commands. They are also different from the previously described IBASIC commands because an IBASIC display partition is not required. This means that they can be used to write or draw directly to a measurement window.

These commands, listed in Table 7-2 on page 7-17, are SCPI mnemonics and are programmable from an external controller as well as from IBASIC. The commands are of the following form

```
DISPlay:WINDow[1|2|10]:GRAPhics:<command>
```

The number specified in the `WINDow` part of the command selects where the graphics are to be written.

`WINDow1` draws the graphics to the channel 1 measurement window.

`WINDow2` draws the graphics to the channel 2 measurement window.

`WINDow10` draws the graphics to an IBASIC display partition.

TIP

Graphics written directly to a measurement window are not redrawn sweep-to-sweep. When SCPI graphics commands are used to write directly to a measurement window, they write to the static graphics plane (the same plane where the graticule is drawn). There is no sweep-to-sweep speed penalty once the graphics have been drawn.

Table 7-2 SCPI Graphics Commands

SCPI COMMAND	FORM	DESCRIPTION
`DISPlay:WINDow[1|2|10]:GRAPhics :CIRCle <radius>`	command only	Draws a circle of the specified Y-axis radius centered at the current pen location — `radius` is in pixels.
`DISPlay:WINDow[1|2|10]:GRAPhics :CLEar`	command only	Clears the user graphics and graphics buffer for the specified window.
`DISPlay:WINDow[1|2|10]:GRAPhics :COLor <num>`	NR1	Sets the color of the user graphics pen — choose from `0` for erase, `1` for bright, and `2` for dim.
`DISPlay:WINDow[1|2|10]:GRAPhics [:DRAW] <x>,<y>`	command only	Draws a line from the current pen position to the specified new pen position — `x` and `y` are the new absolute X and Y coordinates in pixels.
`DISPlay:WINDow[1|2|10]:GRAPhics :LABel <string>`	command only	Draws a label with the lower left corner at the current pen location.
`DISPlay:WINDow[1|2|10]:GRAPhics :LABel:FONT <font>`	CHAR	Selects the user graphics label font — choose from `SMALl|HSMall|NORMal|HNORmal |BOLD|HBOLd|SLANt|HSLant`.
`DISPlay:WINDow[1|2|10]:GRAPhics :MOVE <x>,<y>`	NR1,NR1	Moves the pen to the specified new pen position — `x` and `y` are the new absolute X and Y coordinates in pixels.
`DISPlay:WINDow[1|2|10]:GRAPhics :RECTangle <xsize>,<ysize>`	command only	Draws a rectangle of the specified size with lower left corner at the current pen position — `xsize` and `ysize` are the width and height in pixels.
`DISPlay:WINDow[1|2|10]:GRAPhics :SCALe <xmin>,<xmax>,<ymin>,<ymax>`	command only	Specifies a new coordinate system for the specified window. Subsequent graphics commands will use these new coordinates. This command may be useful for converting older programs to work on your analyzer. (See "Graphics Exceptions" on page 7-14.)

For More Information

For more information about the analyzer's user graphics commands, refer to Chapter 7 of the *Programmer's Guide*. Refer also to the example program titled “GRAPHICS” in the *Example Programs Guide*.

8 Interfacing with External Devices

This chapter describes the techniques necessary for programming the GPIB interface. It describes how this interface works and how to use it to control or interface with systems containing various GPIB devices. It also describes how to interface with external devices using the serial and parallel interfaces.

The GPIB interface is Agilent's implementation of the IEEE-488.2 Digital Interface for Programmable Instrumentation. The acronym GPIB stands for “General Purpose Interface Bus,” and is often referred to as the “bus.” The interface is easy to use and allows great flexibility in communicating data and control information between an HP Instrument BASIC program and external devices.

IBASIC is a GPIB instrument controller residing inside an instrument. It uses the instrument's GPIB interface for external communication and an internal GPIB interface to communicate with the instrument. This unique arrangement presents a few differences between IBASIC's implementation of GPIB control and HP BASIC controllers. A description of the interaction of IBASIC with the host instrument and the external GPIB interface is given in the section entitled “The IBASIC GPIB Model” on page 8-17.

Communication with Devices

GPIB Device Selectors

Since the GPIB allows several devices to be interconnected, each device must be uniquely identified. Specifying the select code (such as 7 or 8) of the GPIB interface to which a device is connected, is not enough to uniquely identify each specific device on the bus.

Each device on the bus has a primary address that identifies it. This address can be set by the user. It must be unique to allow individual access of each device. When a particular GPIB device is to be accessed, it must be identified with both its interface select code and its bus address.

The interface select code is the first part of an GPIB device selector. IBASIC programs run inside an instrument and communicate with it over an internal bus (interface select code 8). IBASIC programs can also communicate with external devices using the instrument's GPIB interface (select code 7).

The second part of an GPIB device selector is the device's primary address, an integer in the range of 0 through 30. For example, to specify the device on the interface at select code 7 with a primary address of 22, use device selector 722. Secondary GPIB addressing is also supported for those devices requiring it. These devices will have at least 5-digit service selection such as 72201.

Since the analyzer is the only device on the internal interface, its primary address on that interface is arbitrary and the instrument will respond to any primary address with a select code equal to 8 (e.g., 800, 811, 822, etc.).

NOTE Each GPIB device address must have a unique GPIB address. The analyzer is shipped from the factory with a primary address of 16. No other device on the bus should use the same address.

Setting Device Addresses

The procedure for setting the address of an GPIB device is given in the installation manual for each device. To set the address of the analyzer, use the softkeys in the SYSTEM OPTIONS **GPIB** menu, or the SCPI mnemonic `SYST:COMM:GPIB:ADDR`.

Moving Data through the GPIB

Data is output and entered into the program through the GPIB with the `OUTPUT` and `ENTER` statements, respectively. The only difference between the `OUTPUT` and `ENTER` statements for the GPIB and those for other interfaces is the addressing information within GPIB device selectors.

The following examples show several different syntax styles which you can use.

```
100   Hpib=7
110   Device_addr=22
120   Device_selector=Hpib * 100 + Device_addr
130    !
140   OUTPUT Device_selector;”F1R7T2T3”
150   ENTER Device_selector;Reading

320   ASSIGN @Hpib_device TO 702
330   OUTPUT @Hpib_device;”Data message”
340   ENTER @Hpib_device;Number

440   OUTPUT 800;”SOUR1:POW -10 dBm”
480   ENTER 724;Readings(*)
```

General Structure of the GPIB

Communications through the GPIB are made according to a precisely defined standard, IEEE 488.1. The rules set by IEEE 488.1 ensure that orderly communication takes place on the bus. For more information about the structure of the GPIB and the IEEE 488.1 standard, refer to the *Tutorial Description of the Hewlett-Packard Interface Bus*.

Devices that communicate over the GPIB perform one or more of the following three functions:

Talk — send data over the bus

Listen — receive data over the bus

Control — control the exchange of data on the bus

The System Controller

The controller is a device that has been designated to control all communication occurring on the bus. It specifies which device talks, which device listens, and when the exchange of data takes place.

A GPIB system can have more than one device with the ability to control the bus, but only one of these devices is allowed to control the exchange of data at any given time. The device that is currently controlling the exchange of data is called the **Active Controller**.

One device must be able to take control of the bus even if it is not the active controller. The device designated as the **System Controller** is the only device with this ability. To designate the analyzer as the system controller, use the **System Controller** softkey in the SYSTEM OPTIONS **GPIB** menu.

The system controller is generally designated before running a program and should not be changed under program control. An exception to this is when an IBASIC program is running on the analyzer's internal controller. If the IBASIC program controls other GPIB devices, the analyzer must be designated as the system controller.

A SCPI mnemonic `SYST:COMM:GPIB:CONT <ON|OFF>` can be used to make the analyzer the system controller. Program execution should be carefully synchronized by using the Operation Complete command (`*OPC?`) and waiting for a reply before any `OUTPUT 7xx` command is sent. (Refer to the “Synchronizing the Analyzer and a Controller” chapter in the *Programmer's Guide* for more information on the `*OPC?` command.)

Using the Serial and Parallel Ports

The analyzer has a parallel port and a serial port for use with peripherals (like printers and plotters), material handlers and other devices. Active control of the GPIB interface is not needed when these ports are being used.

In addition to the serial and parallel ports, there are also two BNC connectors on the rear panel of the analyzer. These connectors provide access (using TTL signal levels) to two programmable bits:

Limit Test TTL bit — indicates the results of a pass/fail limit test

User TTL bit — to be used as needed (for example to be used with a foot pedal)

Using the Analyzer Ports in IBASIC Programs

IBASIC can directly control the serial port, the parallel port, the Pass/Fail TTL bit, and the User bit without using the GPIB commands along with `READIO` and `WRITEIO` commands. However, `READIO` and `WRITEIO` are faster than GPIB commands.

Table 8-1 Writable Ports [I=WRITEIO(A,B)]

IO Type	Port, Register, Data	Description
`WRITEIO`	`15,0;A`	Outputs 8-bit data to the Cent_DO through D7 lines of the Centronics port. Cent_DO is the least significant bit, Cent_D7 is the most significant bit. Sets Printer_select signal high (de-select). Checks Centronics status lines for the following: • OUT OF PAPER • PRINTER NOT ON LINE • BUSY • ACKNOWLEDGE
`WRITEIO`	`15,1;A`	Sets/clears the “user” bit according to the least significant bit of A. A least significant bit equal to 1 sets the user bit high. A least significant bit of 0 clears the user bit.
`WRITEIO`	`15,2;A`	Sets/clears the limit pass/fail bit according to the least significant bit of A. A least significant bit equal to 1 sets the pass/fail bit high. A least significant bit of 0 clears the pass/fail bit.
`WRITEIO`	`15,3;A`	Outputs 8-bit data to the Cent_D0 through D7 lines of the Centronics port. Cent_D0 is the least significant bit, Cent_D7 is the most significant bit. Sets Printer_select signal high (de-select). Does not check Centronics status lines.
`WRITEIO`	`9,0;A`	Outputs a byte to the serial port. The byte is output serially according to the configuration for the serial port. (See above.)

Table 8-2 Readable Ports [I=READIO(A,B)]

IO Type	Register	Description
READIO	9,0	Reads the serial port.
READIO	15,0	Reads the 8-bit data port, Cent_D0 through D7.
READIO	15,1	Reads the user bit.
READIO	15,2	Reads the limit test pass/fail bit.
READIO	15,10	Reads the 8-bit status port . • D0 – Cent_acknowledge • D1 – Cent_busy • D3 – Cent_on_line • D2 – Cent_out_of_paper • D4 – Cent_printer_err

An example program, REPORT, demonstrating peripheral control over the parallel port is provided in the *Example Programs Guide.*

Refer to the *Automating Measurements User's Guide Supplement* for further explanation and examples of how to access the analyzer's I/O ports.

General Bus Management

The GPIB standard provides several mechanisms to manage the bus and the devices on the bus. Here is a summary of the IBASIC statements that use these control mechanisms:

`ABORT`	abruptly terminates all bus activity and resets all devices to their power-on GPIB states.
`CLEAR`	sets selected (or all) devices to a pre-defined, device-dependent GPIB state.
`LOCAL`	returns selected (or all) devices to local (front panel) control.
`LOCAL LOCKOUT`	disables selected (or all) devices' front panel controls.
`REMOTE`	puts selected (or all) devices into their device-dependent, remote modes.
`SPOLL`	performs a serial poll of the specified device (which must be capable of responding).
`TRIGGER`	sends the trigger message to a device (or selected group of devices).

The actions that a device takes upon receiving each of the above commands are different for each device. For external devices, refer to the particular device's manuals to determine how it will respond.

All of the bus management commands, with the exception of `ABORT`, require that the program be the active controller on the interface. A running IBASIC program is always the active controller on the internal interface (select code 8). The instrument must either be set as system controller or have control passed to it from an external controller for the program to be the active controller on the external interface (select code 7). The program automatically assumes the controller status of the host instrument. For more information, refer to "The IBASIC GPIB Model" on page 8-17.

NOTE In this section, the term Host Instrument refers to the instrument where the IBASIC controller is located.

REMOTE

Most GPIB devices can be controlled from the front panel or from the bus. If the device's front panel controls are currently functional, it is in the Local state. If it is being controlled through the GPIB, it is in the Remote state. Unless operating in the Local Lockout mode, each GPIB device has method (usually a key) to return itself to Local (front panel) control.

When the analyzer is being controlled by a program running on an external controller, the **Return to Local** softkey is always available to return the analyzer to Local control.

The Remote message is automatically sent to all devices whenever the system controller is powered on, reset, or when it sends the Abort message. A device also enters the Remote state automatically whenever it is addressed. The `REMOTE` statement also outputs the Remote message, which causes all (or specified) devices on the bus to change from local control to remote control. The host instrument must be designated as the system controller before an IBASIC program can execute the `REMOTE` statement on select code 7.

Host Instrument

The `REMOTE` statement has no effect on the host instrument since it is always in remote control whenever an IBASIC program is running. Specifying the internal interface in a `REMOTE` statement will not generate an error, but will have no effect.

LOCAL LOCKOUT

The Local Lockout message effectively locks out the “local” switch present on most GPIB device front panels. It maintains system integrity by preventing a user from interfering with system operations by pressing buttons. As long as Local Lockout is in effect, no bus device can be returned to local control from its front panel.

The Local Lockout message is sent by executing the `LOCAL LOCKOUT` statement. This message can be sent to all devices on the external interface by specifying the bus address (7). Specifying a single address on the bus (i.e. 722) sends the command to only the device at that address. The Local Lockout message is cleared when the Local message is sent by executing the `LOCAL` statement. However, executing the `ABORT` statement does not cancel the Local Lockout message.

Host Instrument

The Local Lockout message is not supported for the host instrument since front panel control is always necessary in order to pause or abort the program. Specifying the internal interface in a `LOCAL LOCKOUT` statement will not generate an error, but will have no effect.

LOCAL

During system operation, it may be necessary for an operator to interact with one or more external devices. For instance, an operator might need to work from the front panel to make special tests or to troubleshoot. It is also good systems practice to return all devices to local control when remote-control operations are complete. Executing the `LOCAL` statement returns the specified devices to local (front panel) control.

If primary addressing is specified, the Go-to-Local message is sent only to the specified device(s). However, if only the interface select code is specified (`LOCAL 7`), the Local message is sent to all devices on the external interface and any previous Local Lockout message (which is still in effect) is automatically cleared.

Host Instrument

The `LOCAL` statement has no effect on the host instrument since it is always in remote control whenever an IBASIC program is running. Specifying the internal interface in a `LOCAL` statement will not generate an error.

TRIGGER

The `TRIGGER` statement sends a Trigger message to a selected device or group of devices. The purpose of the Trigger message is to initiate some device-dependent action; for example, it can be used to trigger a digital voltmeter to perform its measurement cycle. Because the response of a device to a Trigger message is strictly device-dependent, neither the Trigger message nor the interface indicates what action is initiated by the device.

Specifying only the interface select code outputs a Trigger message to all devices currently addressed to listen on the bus. Including a device address in the statement triggers only the device addressed by the statement.

Host Instrument

The TRIGGER statement is supported by the analyzer. Issuing a TRIGGER command will initiate a single sweep assuming the analyzer is in TRIGGER hold mode. TRIGGER is ignored if not in hold mode.

CLEAR

The CLEAR statement provides a means of “initializing” a device to its predefined device-dependent state. When the CLEAR statement is executed, the Clear message is sent either to all devices or to the specified device, depending on the information contained within the device selector. If only the interface select code is specified, all devices on the specified GPIB interface are cleared. If primary-address information is specified, the Clear message is sent only to the specified device. Only the active controller can send the Clear message.

Host Instrument

The CLEAR statement is fully compatible on the internal interface.

ABORT

This statement may be used to terminate all activity on the external bus and return the GPIB interfaces of all devices to reset (or power-on) condition. Whether this affects other modes of the device depends on the device itself. The IBASIC program must be either the active or the system controller to perform this function. If it is the system controller and has passed active control to another device, executing this statement causes active control to be returned. Only the interface select code may be specified; primary-addressing information (such as 724) must not be included.

Aborting the Internal Bus

ABORT is not supported for select code 8. Executing ABORT 8 will not generate an error.

GPIB Service Reqsts

Most GPIB devices, such as voltmeters, frequency counters, and analyzers, are capable of generating a “service request” when they require the active controller to take action. Service requests are generally made after the device has completed a task (such as making a measurement) or when an error condition exists (such as a printer being out of paper). The operating and/or programming manuals for each device describe the device's capability to request service and conditions under which the device will request service. To request service, the device sends a Service Request message (SRQ) to the active controller. The mechanism by which the active controller detects these requests is the SRQ interrupt. Interrupts allow an efficient use of system resources, because the system may be executing a program until interrupted by an event's occurrence. If enabled, the external event initiates a program branch to a routine which “services” the event by performing some action.

Setting Up and Enabling SRQ Interrupts

In order for an GPIB device to be able to initiate a service routine in the active controller, two prerequisites must be met: the SRQ interrupt event must have a service routine defined, and the SRQ interrupt must be enabled to initiate the branch to the service routine.

The following program segment shows an example of setting up and enabling an SRQ interrupt.

```
100   Hpib=7
110   ON INTR Hpib GOSUB Service_routine
120   !
130   Mask=2
140   ENABLE INTR Hpib;Mask
```

Since IBASIC recognizes only SRQ interrupts, the value assigned to the mask is meaningless. However, a mask value may be present as a placeholder for compatibility with HP Series BASIC programs.

When an SRQ interrupt is generated by any device on the bus, the program branches to the service routine when the current line is exited (either when the line's execution is finished or when the line is exited by a call to a user-defined function). The service routine, in general, must perform the following operations in order:

1. Determine which device(s) are requesting service.
2. Determine what action is requested.
3. Clear the SRQ line.
4. Perform the requested action.
5. Re-enable interrupts.
6. Return to the former task (if applicable).

NOTE The `ON INTR` statement must always precede the `ENABLE INTR` statement when the two are used in the same program.

Servicing SRQ Interrupts

The SRQ is a level-sensitive interrupt; in other words, if an SRQ is present momentarily but does not remain long enough to be sensed by the controller, an interrupt will not be generated. The level-sensitive nature of the SRQ line also has further implications, which are described in the following paragraphs.

Example

Assume that only one device is currently on the bus. The following service routine serially polls the device requesting service and clears the interrupt request. In this case, the controller does not have to determine which device was requesting service because only one device is present. Since only service request interrupts are enabled in IBASIC, the type of interrupt does not need to be determined either. The service is performed, and the SRQ event is re-enabled to generate subsequent interrupts.

```
500   Serv_rtn:Ser_poll=SPOLL(@Device)
510   ENTER @Device;Value
520   PRINT Value
530   ENABLE INTR 7  ! Use previous mask.
540   RETURN
```

The IEEE standard states that when an interrupting device is serially polled, it is to stop interrupting until a new condition occurs (or the same condition occurs again). To “clear” the SRQ line, a serial poll must be performed on the device. By performing this serial poll, the controller acknowledges to the device that it has seen the request for service and is responding. The device then removes its request for service (by releasing SRQ).

If the SRQ line had not been released, the controller would have branched to the service routine immediately upon re-enabling interrupts on this interface. This is due to the level-sensitive nature of the SRQ interrupt.

Also note that once an interrupt is sensed and logged, the interface cannot generate another interrupt until the first interrupt is serviced. The controller disables all subsequent interrupts from an interface until a pending interrupt is serviced.

Conducting a Serial Poll

A sequential poll of individual devices on the bus is known as a Serial Poll. A byte of device-specific status is returned in response to a Serial Poll. This byte is called the “Status Byte” message and, depending on the device, may indicate an overload, a request for service, or a printer being out of paper. The particular response of each device depends on the device.

The `SPOLL` function performs a Serial Poll of the specified device; the program must currently be the active controller in order to execute this function.

Examples

```
Status_byte=SPOLL(@Device)

ASSIGN @Device TO 700

Spoll_724=SPOLL(724)
```

The Serial Poll is meaningless for an interface since it must poll individual devices on the interface. Therefore, primary addressing must be used with the `SPOLL` function.

Passing and Regaining Control

Active control of the bus can be passed between controllers using the PASS CONTROL command. The following statements first define the GPIB interface's select code and the new active controller's primary address and then pass control to that controller.

```
100    Hp_ib=7
110    New_ac_addr=20
120    PASS CONTROL 100*Hp_ib+New_ac_addr
```

Once the new active controller has accepted active control, the controller passing control assumes the role of a non-active controller on the specified GPIB interface. The concept of using pass control with IBASIC is discussed in the next section, "The IBASIC GPIB Model."

The IBASIC GPIB Model

The fact that IBASIC resides in, and coexists with an instrument poses a large set of possible interactions, both internal to the instrument and externally with other controllers and instruments. This section defines the principal players and rules of order when IBASIC is running within the host instrument.

The External Bus and the Internal Bus

There is physically only one GPIB port and one GPIB address for the analyzer. IBASIC has access to two GPIB ports: the “real” external port (select code 7) and a “virtual” internal port (select code 8), through which it communicates with the analyzer.

The analyzer has only one output buffer, one input buffer, and one set of status registers. Commands and data from both ports are placed in the same input buffer and data read out of both ports comes from the same output buffer. The instrument will not provide any kind of arbitration between an external controller and an IBASIC program.

The analyzer always behaves as if there is only one controller. If an IBASIC program is running, it is assumed to be the controller and therefore will receive all SRQs from the host instrument (via the internal port).

Service Request Indicators

An external controller may perform a serial poll (`SPOLL`) at any time without affecting a running IBASIC program. There are two Service Request Indicators (SRI) — one for the external port and one for the internal port. The internal SRI can only be cleared by an IBASIC program performing an `SPOLL` on device 800. The external SRI can only be cleared by an `SPOLL` from an external controller and can only be set when there is not an active IBASIC program.

The two SRIs will be set to their OR'd value when a program starts, and again when it finishes. This assures that any pending SRQs can be serviced by the instrument's new controller.

The pausing or termination of a program will cause the Program Running bit in the Device Status register to go low. This can be used to generate an external SRQ. (For an example, see the DUALCTRL program in the *Example Programs Guide*.)

IBASIC as the Active Controller

The IBASIC program is always the active controller on the internal interface (select code 8). When a program starts running, the GPIB controller status of the instrument is automatically passed to the program. For example, if the instrument is set as System Controller, a program running in the instrument automatically becomes system controller and active controller on the external bus, and the instrument relinquishes active control. When the program stops, the instrument regains active control.

Also, if an instrument set as Talker/Listener is passed control from an external controller, any program running in the instrument becomes active controller on the external interface.

Thus, there are two cases where a program running in an instrument can be the active controller on the external interface:

1. When the host instrument is set as System Controller and the program has not passed control.
2. When the host instrument is set as Talker/Listener and the instrument has been passed control from an external controller.

Passing Active Control to the Instrument

The only way that the analyzer can gain active control of the external interface while a program is running, is if the program is currently the active controller on select code 7 and passes control to the instrument. Normally, the active controller on the 7 bus can pass control to any device on the interface by using the statement

```
PASS CONTROL 7xx
```

where “xx” represents the address of the device on the bus. Because an IBASIC program does not interface with the host instrument over select code 7, a different method is used to pass control in this case. To pass active control of the external interface from an IBASIC program to the host instrument, use the statement

```
PASS CONTROL 8xx
```

where “xx” represents any two digit number from 00 to 99. This allows the instrument to control external plotters, printers and disk drives. When the instrument is finished with its GPIB control activity, it automatically passes control back to the program.

NOTE

Control over the internal bus is used to govern access to the external bus. When the instrument is given control over the internal bus, it is actually given access to the external GPIB hardware.

IBASIC as a Non-Active Controller

The IBASIC program is always the active controller on the internal interface. There are two cases where an IBASIC program does not have control of the external GPIB interface:

1. When the host instrument is set as Talker/Listener and active control has *not* been passed from an external device.
2. When the host instrument is set as System Controller and the program has passed control to either the analyzer or another device on the external interface.

In both of these cases, the program cannot perform activities of any kind on the external interface.

NOTE An IBASIC program cannot act as a device on the external bus. To communicate with an external controller, the IBASIC program must be active controller and the external controller must act as the device (see "Interfacing with an External Controller" on page 8-21).

Interfacing with an External Controller

So far, we have discussed the ability to interface IBASIC programs with a network of external devices using the GPIB. The idea of including an external controller in that network, and interfacing an IBASIC program with a program running in that computer, presents some new possibilities.

External controller programs can interface with IBASIC programs (referred to as “internal programs”) over GPIB in two basic ways:

First, the two programs can pass data back and forth using simple `OUTPUT` and `ENTER` statements. This requires coordination of both the internal and external programs and also requires that the internal program be the active controller during the interaction. To get an internal program and an external program to work together successfully, you should have a good understanding of the GPIB model, presented earlier in this chapter.

Second, the external program can make use of the extensive set of analyzer GPIB commands that interface with IBASIC programs. These mnemonics fall under the subsystem `PROGram` and allow the external controller to remotely perform many of the IBASIC front panel activities. This includes the ability to run, stop, pause, continue and delete an internal program. You can also remotely query or set the values of numeric and string variables.

Also included in the analyzer GPIB command set are commands that allow you to transfer programs and program data to and from the instrument. Programs can be transferred (uploaded and downloaded) between an external controller and the program buffer in the instrument, and data can be transferred between an external program and a non-running internal program by setting and querying internal program variables. These SCPI mnemonics are described in the *Programmer's Guide*.

Also, refer to example programs included on the *IBASIC Example Programs Disk*: `DUALCTRL, TRICTRL, UPLOAD` and `DOWNLOAD`. These programs demonstrate using IBASIC with an external controller.

Synchronizing IBASIC with an External Controller

Using OUTPUT and ENTER Statements

Commands sent to the analyzer with OUTPUT and ENTER statements from IBASIC and from the external controller at the same time must be synchronized by the programmer. These commands cannot be allowed to overlap. Overlapped commands sent from the external controller and IBASIC will result in unpredictable behavior or deadlocks.

For example:

If the external controller executes:

```
OUTPUT 716;"<command>"
```

and IBASIC simultaneously executes:

```
OUTPUT 800;"<command>"
```

the results are unpredictable.

To avoid overlapped commands, you must ensure that only the controller is allowed to send commands or only IBASIC is allowed to send commands at any one time. One possible method to avoid overlap is described below. For this method, when the respective controller is done sending commands, the other controller is informed. The alternate controller may then begin sending commands. After each set of commands is completed, the alternate controller is informed and given a signal to send commands to the analyzer. See example program `TRICTRL` in the *Example Programs Guide*.

Using Status Information

Status information must also be synchronized between the IBASIC program and a program running on an external controller. The status information is shared between these programs. Commands which affect the status information should not overlap between the IBASIC program and the external controller.

For example:

From an external controller:

```
OUTPUT 716;”<command>;*OPC?”
```

```
ENTER 716;Opc
```

From IBASIC, simultaneously execute:

```
OUTPUT 800;”*CLS”
```

may cause the external controller to not complete execution. The `*CLS` command clears status information which the external controller may be waiting for. The commands which affect status information include `*OPC`, `*OPC?`, `*WAI`, `*CLS`, `*RST`, `*SRE`, `*ESE`, and `STAT:PRES`.

Design Rules

Design your IBASIC and External Controller with the following rules:

- Do not overlap commands between the external controller and IBASIC.
- Do not change status information which is expected by the alternate controller. Design programs so status information does not overlap.

See the example program, `TRICTRL`, which implements a synchronization protocol between an external controller and two instruments running IBASIC programs.

Transferring Data between Programs

Using OUTPUT and ENTER Statements

All data sent from an external controller to the instrument's external port is received by the instrument and not by any program running in it. Therefore, a non-active controller IBASIC program can never enter or output data via the external interface. In order to pass data between an external controller and an internal program using `OUTPUT` and `ENTER` statements, the internal program must be given active control and the external controller must become the non-active controller. HP IBASIC for Windows and HP BASIC controllers have the ability to enter and output data via GPIB while acting as a non-active controller.

NOTE Moving data through the GPIB and running a measurement in the host instrument at the same time can slow both operations significantly. It is recommended that you do not perform these operations simultaneously.

One method of passing data between the two controllers is to set the instrument as Talker/Listener and run a program on the external controller that starts the IBASIC program and passes control to it. The IBASIC program can then output data to, and enter data from, the external controller. Two programs listed in the *Example Programs Guide* demonstrate how to transfer data between an internal program and an external controller program. The first program, DATA_EXT, is run from an external controller. It assumes that a disk containing the corresponding IBASIC program `DATA_INT` is in the disk drive of the analyzer. It remotely loads the IBASIC program, starts it, and then transfers active control to it. The IBASIC program `DATA_INT`, with active control of the interface, queries the external program for the name of the drive to catalog, and then outputs the catalogued string to the external program, and then passes active control back. After receiving the catalog data, the external program goes into a loop (line 1080) executing a command that continues to generate an error until the host computer again becomes the active controller, when control is passed back.

Setting and Querying Variables

Another means of transferring data between an internal and an external program involves the ability to set and query internal program variables from an external program. The “`PROGram[:SELected]:NUMBer`” and “`PROGram[:SELected]:STRing`” mnemonics (and their query counterparts) are analyzer GPIB commands. The internal program must not be running when these commands are executed.

The command

```
PROG:NUMB <string>, <value>
```

sets the value of a numeric variable in the program. The command

```
PROG:STR <string>, <value>
```

sets the value of a string variable in the program. In both the `PROG:NUMB` and `PROG:STR` commands and queries, `<string>` is the variable name and must be string data (in quotes). In the `PROG:STR` command, `<value>` is also string data (in quotes).

Numeric and string parameters can also be queried. The query

```
PROG:NUMBer? <string>
```

returns the value of the specified numeric variable.

Arrays of `REAL` or `INTEGER` type may be sent or queried but arrays of strings are not allowed. Array elements are separated by commas.

Examples

```
OUTPUT 716;”PROG:NUMBER 'Test',99”
OUTPUT @Ibasic;”PROG:STRING 'A$','String Data'”
OUTPUT 716;”PROG:NUMB? 'Iarray(*)'”
```

The following program segment sends both numeric and string variable queries and enters the resulting data:

```
10 ASSIGN @Prog TO 716
20 OUTPUT @Prog;"FORM ASCII,3"
30 OUTPUT @Prog;"PROG:NUMB? 'Test'"
40 ENTER @Prog;Testval
50 PRINT "The value of the variable Test=";Testval
60 OUTPUT @Prog;"PROG:STR? 'A$'"
70 ENTER @Prog;STR$
80 PRINT "A$=";Str$
90 End
```

Transferring Programs

Programs can be transferred between an external controller and program memory using the GPIB download command "PROGram[:SELected]:DEFine" and its upload query "PROGram[:SELected]:DEFine?". Programs that use these mnemonics are run in the external controller.

Downloading

Program data transferred (downloaded) from the external controller to the instrument is always transferred as an "arbitrary block." The arbitrary block may be a definite length or indefinite length block. The indefinite length block is by far the easiest and is simply a block of data that begins with the characters "#0" preceding the first line, and ends with a line-feed character accompanied by an EOI signal on the GPIB interface.

When using the mnemonic PROG:DEF to download program lines, the #0 must not be followed by a line-feed. Each program line must have a line number at its beginning and a line-feed at its end. To end the arbitrary block of program lines, a single line-feed must be output with the OUTPUT END parameter, which sends the EOI (End or Identify) signal on the GPIB control lines.

Refer to the *Example Programs Guide* for a listing of the example program DOWNLOAD.

Notice that the `OUTPUT` statement on line 460 is terminated with a semicolon. This suppresses the line-feed that would otherwise occur.

As each line of the program is downloaded, it is checked for syntax.

If an error is found, the error message is displayed on the screen and the line is commented and checked for syntax again. If it still causes an error (for example the line may be too long), the line is discarded.

Any lines that currently exist in the memory buffer will remain unless they are overwritten by downloaded program lines. This makes it easy to edit lines in an external controller and then download only the edited lines into an existing program. If you want to completely overwrite the current program in memory, you must delete the program first. This can be done remotely using the extended command `PROG:DEL:ALL` (see line 350 in the example program `DOWNLOAD`).

Uploading

The mnemonic `PROG:DEF?` is used to upload a program from the program buffer. The entire program is then returned as a definite length arbitrary block. A definite length block starts with the "#" character followed by a single digit defining the number of following digits to read as the block length.

Refer to the *Example Programs Guide* for a listing of the example program `UPLOAD`, which demonstrates an uploading routine run on an external controller.

The subroutine `Openfile` (lines 570 through 770) creates an ASCII file in which to save the uploaded program. The number of 256 byte records declared in the `CREATE ASCII` statement (line 730) is the file size (declared in the definite block header) divided by 256. Line 720 accommodates any remainder in this calculation by increasing the file size number by one record, if any remainder exists.

IBASIC Communication across the LAN

You may need a way for an IBASIC program running on the analyzer to signal a remote computer that it has completed some operation.

IBASIC cannot communicate directly across LAN using the ASSIGN and OUTPUT or ENTER commands. However, IBASIC can use the following SCPI command to send a message to a remote computer via LAN:

```
DIAGnostic:COMMunicate:LAN:SEND <IP_ADDR>,<PORT_NUM>,<STRING>
```

This command opens a socket to the remote computer, and sends the specified string. The `<IP_ADDR>` argument specifies the IP address of the remote computer. The `<PORT_NUM>` argument specifies the port number to use. The `<STRING>` is the message to be sent.

For example:

```
DIAGnostic:COMMunicate:LAN:SEND ‘15.4.40.49’,8001,’Ready!’
```

If the remote computer is not listening for a LAN connection at the specified port, this command will block, and wait for the remote computer to accept the connection. After about 75 seconds, it will time out. This is the standard TCP/IP timeout period.

See the *LAN Interface User’s Guide Supplement* for more information.

9 Using Subprograms

User-Created Subprograms

You can use the LOADSUB keyword with subprograms of your own creation. LOADSUB enables you to append subprograms to other programs and is supported as described in the *IBASIC Guide*. When using LOADSUB, keep in mind the following:

- Subprograms must be stored to files using the STORE keyword when first created.
- Subprograms may be stored from the external keyboard or from the front panel if the [File Type] format is BIN.
- BIN type files are generally not transportable between the analyzer and other development systems (only ASCII files are compatible with other systems).

Typical examples of LOAD/STORE:

From an external keyboard:

LOAD “MYFILE”

STORE “MYFILE”

From the front panel:

SAVE RECALL **Programs** **File Type BIN** **Save Program**

SAVE RECALL **Programs** **File Type BIN** **Recall Program**

Typical examples of LOADSUB:

LOADSUB subprogram_name FROM “filename”

LOADSUB ALL FROM “filename”

User-created subprograms are appended to the end of the BASIC program currently stored in the EDIT buffer.

Built-In High-Speed Subprograms

You can use LOADSUB to access pre-compiled routines stored as instrument firmware in internal memory. Any IBASIC program running on the analyzer can access these subprograms; programs running on external computers cannot. The external program must use the equivalent code listed in this chapter in place of a built-in subprogram.

IBASIC programs which use the built-in subprograms are simpler and run faster. For example, most data transfer operations run twice as fast when using the built-in subprograms; math operations run many times faster. Built-in subprograms are stored in memory designated as “MEM,0,0”.

To access a subprogram, the subprogram first must be loaded into the main program using the LOADSUB keyword. The LOADSUB keyword requires a filename be specified from which to load the subprogram. Three built-in files are “XFER”, “MATH”, and “RPG”.

- “XFER” file adds support to transfer trace data between the instrument and the IBASIC program.
- “MATH” file adds high speed support for complex array operations.
- “RPG” file adds fast RPG (rotary pulse generator — front panel knob) response for markers.

LOADSUB `<Subprogram name>` FROM `<Filename:MEM,0,0>` loads the named subprogram from the built-in file “FILENAME”.

LOADSUB ALL FROM `<Filename:MEM,0,0>` loads all the subprograms in the named built-in file “FILENAME”. See the following table for subprogram names within the files “XFER”, “MATH”, and “RPG”.

Table 9-1 XFER Subprograms

Filename	Subprogram Name (parameter list)	Description
XFER	`Read_fdata` `(INTEGER Chan,REAL A(*))`	Reads real formatted data.
	`Read_fmem` `(INTEGER Chan,REAL A(*))`	Reads real formatted mem.
	`Read_cdata` `(INTEGER Chan,REAL A(*))`	Reads complex data.
	`Read_cmem` `(INTEGER Chan,REAL A(*))`	Reads complex memory.
	`Write_fdata` `(INTEGER Chan,REAL A(*))`	Writes real formatted data.
	`Write_fmem` `(INTEGER Chan,REAL A(*))`	Writes real formatted mem.
	`Write_cdata` `(INTEGER Chan,REAL A(*))`	Writes complex data.
	`Write_cmem` `(INTEGER Chan,REAL A(*))`	Writes complex memory.
	`Read_rdata` `(INTEGER Chan,Input$, REAL A(*))`	Reads raw complex data.
	`Write_rdata` `(INTEGER Chan,Input$,REAL A(*))`	Writes raw complex data.
	`Read_corr` `(INTEGER Chan, N,REAL A(*))`	Reads complex error coef.
	`Write_corr` `(INTEGER Chan, N,REAL A(*))`	Writes complex error coef.

Table 9-2 MATH Subprograms, Complex Arrays

Filename	Subprogram Name (parameter list)	Description
MATH Define Complex Array Operations	`Cmplx_mag(REAL Cdata(*),Mag(*),INTEGER Sz)`	Mag of complex array
	`Cmplx_arg(REAL Cdata(*),Arg(*),INTEGER Sz)` $\arg(i) = \operatorname{atan}\left(\frac{C_{imag}(i)}{C_{real}(i)}\right)$ in the range $-\pi, \pi$	
	`Cmplx_conjg(REAL A(*),B(*))` $\bar{A} \Rightarrow B$, complex conjugate of array A goes into array B	

Table 9-3 MATH Subprograms, Complex Numbers (1 of 2)

Filename	Subprogram Name (parameter list)
MATH Define Complex Number Operation	`Cadd(REAL Op1(*),INTEGER Row1,REAL Op2(*), INTEGER Row2,REAL Ans(*),INTEGER Rowans)` complex Ans = complex Op1 + complex Op2 $Ans_{real}(\text{Rowans}) = Op1_{real}(\text{row1}) + Op2_{real}(\text{row2})$ $Ans_{imag}(\text{Rowans}) = Op1_{imag}(\text{row1}) + Op2_{imag}(\text{row2})$
	`Csub(REAL Op1(*), INTEGER Row1, REAL Op2(*), INTEGER Row2, REAL Ans(*), INTEGER Rowans)` complex Ans = complex Op1 - complex Op2 $Ans_{real}(Rowans) = Op1_{real}(row1) - Op2_{real}(row2)$ $Ans_{imag}(Rowans) = Op1_{imag}(row1) - Op2_{imag}(row2)$

Table 9-4 MATH Subprograms, Complex Numbers (2 of 2)

Filename	Subprogram Name (parameter list)
MATH Define Complex Number Operation	`Cmul(REAL Op1(*),INTEGER Row1,REAL Op2(*), INTEGER Row2,REAL Ans(*),INTEGER Rowans)` complex Ans = complex Op1 × complex Op2 $Ans_{real}(Rowans) = Op1_{real}(row1) \bullet Op2_{real}(row2) - Op1_{imag}(row1) \bullet Op2_{imag}(row2)$ $Ans_{imag}(Rowans) = Op1_{real}(row1) \bullet Op2_{imag}(row2) + Op2_{real}(row2) \bullet Op1_{imag}(row1)$
MATH Define Complex Number Operation	`Cdiv(REAL Op1(*),INTEGER Row1,REAL Op2(*), INTEGER Row2,REAL Ans(*),INTEGER Rowans)` complex Ans = complex Op1 ⁄ complex Op2 $den = Op2_{real}(row2) \bullet Op2_{real}(row2) + Op2_{imag}(row2) \bullet Op2_{imag}(row2)$ $num1 = Op1_{real}(row1) \bullet Op2_{real}(row2) + Op1_{imag}(row1) \bullet Op2_{imag}(row2)$ $num2 = Op1_{imag}(row1) \bullet Op2_{real}(row2) - Op1_{real}(row1) \bullet Op2_{imag}(row2)$ $Ans_{real}(Rowans) = \frac{num1}{den}$ $Ans_{imag}(Rowans) = \frac{num2}{den}$

Table 9-5 RPG Subprograms

Filename	Subprogram Name (parameter list)	Description
RPG	`SUB Rpg_function (INTEGER function)`	Redirects RPG &STEP keys function. 0=normal 1=Actv MKR &INPUT

Table 9-6 Example Illustrating Use of Built-In Subroutines

#	Program Statement	Description
10	LOADSUB Read_fdata FROM "XFER:MEM,0,0"	*Appends Read_data sub program to end of this program. This subprogram can now be called.*
20	(Additional program statements here)	
30		
40		
50	REAL Trace_array(1:201)	*Reads Channel 1 data into Trace_array(*).*
60	Read_fdata(1, Trace_array(*))	
70	LOADSUB ALL FROM "MATH:MEM,0,0"	*Appends all math subprograms defined in "MATH" to the end of this program.*
80	END	
90	SUB Read_fdata(INTEGER Chan,REAL A(*))	*Reads real formatted data.*
100	SUB Cmplx_mag(REAL Cdata(*),Mag(*),INTEGER Sz)	*Mag of complex array.*
110	SUB Cmplx_arg(REAL Cdata(*),Arg(*),INTEGER Sz)	*Arg of complex array.*
120	...(program continues)	

Table 9-7 RUNTIME Built-In Subprogram Errors

Number	Description
8,9,16	Improper or inconsistent dimensions found which specify array size. Using the wrong number of subscripts when referencing an array element.
983	Wrong type or number of parameters. An improper parameter list for a machine resident function.

NOTE Built in subprograms cannot be edited. Built in subprograms cannot be edited since they are compiled and built into the firmware. However, any subprogram can be deleted by the DELSUB keyword support in revision 2 IBASIC.

Avoiding Multiple Loads of Subprograms

To avoid multiple LOADS of a subprogram which has already been loaded, the following code should be used.

```
10  ON ERROR GOTO 30
20  DELSUB Read_fdata
30  LOADSUB Read_fdata FROM "XFER:MEM,0,0"
40  OFF ERROR
```

10 IBASIC Keyword Summary

Keywords and Keyword Support

This chapter summarizes the HP Instrument BASIC keyword implementation in the analyzer. Table 10-1 lists the keywords by name in alphabetic order. The type of support for each keyword is shown and any exceptions are noted. Exceptions are important differences between the keyword description in the *HP Instrument BASIC Language Reference* and the implementation in the analyzer. If a keyword is not listed in Table 10-1, it may not be supported in the analyzer.

Keyword and Operator Tables

Table 10-2 through Table 10-20 contain the same information as Table 10-1, with the keywords and operators organized by category.

Keywords in Alphabetic Order

Table 10-1

Table 10-1 Keywords by Name in Alphabetic Order

HP IBASIC Keyword or Operator	Support*			Exceptions and Comments
	FP	EK	P	
&	✓	✓	✓	
*		✓	✓	
+		✓	✓	
-		✓	✓	
/		✓	✓	
^		✓		
<,<=,<>,=,>,>=		✓	✓	
ABORT		✓	✓	Select Code = 7, 8, 9, 15
ABS		✓	✓	
ACS	✓	✓	✓	
ALLOCATE		✓	✓	
AND	✓	✓	✓	
ASN	✓	✓	✓	
ASSIGN		✓	✓	
ATN	✓	✓	✓	
AXES		✓	✓	
BEEP		✓	✓	

*** FP= Front Panel, EK= External Keyboard, P=Programmable**

Table 10-1 (Continued) Keywords by Name in Alphabetic Order

HP IBASIC Keyword or Operator	Support*			Exceptions and Comments
	FP	EK	P	
BINAND	✓	✓	✓	
BINCMP	✓	✓	✓	
BINEOR	✓	✓	✓	
BINIOR	✓	✓	✓	
BIT	✓	✓	✓	
CALL		✓	✓	
CASE			✓	
CASE ELSE			✓	
CAT	✓	✓	✓	Supports 58 columns. See the *HP Instrument BASIC Language Reference*.
CHR$	✓	✓	✓	
CLEAR		✓	✓	Select Code = 7, 8, 9, 15
CLEAR SCREEN		✓	✓	
CLIP		✓	✓	Select Code = 7
CLS		✓	✓	
COM			✓	
CONT		✓	✓	Line number support from KB only.
COPY	✓	✓	✓	
COPYLINES		✓		
COS	✓	✓	✓	Abs val less than 1.7083127722e+10

* FP= Front Panel, EK= External Keyboard, P=Programmable

Table 10-1 (Continued) Keywords by Name in Alphabetic Order

HP IBASIC Keyword or Operator	Support*			Exceptions and Comments
	FP	EK	P	
CREATE				
CREATE ASCII		✓	✓	
CREATE BDATE		✓	✓	
CREATE DIR	✓	✓	✓	
CRT		✓	✓	ENTER CRT(ENTER 1) not supported.
CSIZE		✓	✓	
DET		✓	✓	
DOT	✓	✓	✓	
DRAW				
DROUND		✓	✓	
DUMP ALPHA		✓	✓	
DATA	✓	✓	✓	
DATE		✓	✓	
DATE$			✓	
DEALLOCATE			✓	
DEF FN	✓	✓	✓	
DEG	✓	✓	✓	
DEL		✓	✓	Front Panel deletes a single line only.
DELSUB		✓	✓	
DET		✓	✓	

*** FP= Front Panel, EK= External Keyboard, P=Programmable**

Table 10-1 (Continued) Keywords by Name in Alphabetic Order

HP IBASIC Keyword or Operator	Support*			Exceptions and Comments
	FP	EK	P	
`DIM`		✓	✓	
`DISABLE`			✓	
`DISABLE INTR`		✓	✓	Interface Select Code = 7 or 8
`DISP`	✓	✓	✓	
`DIV`		✓		
`DVAL`	✓	✓	✓	
`DVAL$`		✓	✓	
`EDIT`		✓	✓	Front Panel edits default line number.
`ELSE`			✓	
`ENABLE`			✓	
`ENABLE INTR`			✓	Interface Select Code = 7 or 8
`END`			✓	
`END IF`			✓	
`END LOOP`			✓	
`END SELECT`			✓	
`END WHILE`			✓	
`ENTER`		✓	✓	
`ERRL()`			✓	
`ERRLN()`		✓	✓	
`ERRM$`		✓	✓	

* FP= Front Panel, EK= External Keyboard, P=Programmable

Table 10-1 (Continued) Keywords by Name in Alphabetic Order

HP IBASIC Keyword or Operator	Support*			Exceptions and Comments
	FP	EK	P	
ERRN		✓	✓	
EXIT IF			✓	
EXOR	✓	✓	✓	
EXP		✓	✓	
FN			✓	
FNEND			✓	
FOR NEXT			✓	
FRACT		✓	✓	
FRAME		✓	✓	
GCLEAR		✓	✓	
GET	✓	✓	✓	See Chapter 4 for usage details.
GINIT		✓	✓	
GOSUB			✓	
GOTO			✓	
GRID		✓	✓	
IDRAW		✓	✓	
IF THEN			✓	
IMAGE			✓	
IMOVE		✓	✓	
INDENT		✓		

* FP= Front Panel, EK= External Keyboard, P=Programmable

Table 10-1 (Continued) Keywords by Name in Alphabetic Order

HP IBASIC Keyword or Operator	Support*			Exceptions and Comments
	FP	EK	P	
INITIALIZE	✓	✓	✓	
INPUT			✓	See the *HP Instrument BASIC Language Reference.*
INT		✓	✓	
INTEGER			✓	
IPLOT		✓	✓	
IVAL	✓	✓	✓	
IVAL$	✓	✓	✓	
KBD			✓	Returns select code = 2
LABEL		✓	✓	
LDIR		✓	✓	
LEN	✓	✓	✓	
LET		✓	✓	
LGT		✓	✓	
LIST		✓	✓	Valid Device Selectors
LOAD	✓	✓	✓	See Chapter 4 for usage details.
LOADSUB	✓	✓	✓	
LOADSUB ALL FROM ...	✓	✓	✓	
LOCAL		✓	✓	Select Code 7 only

*** FP= Front Panel, EK= External Keyboard, P=Programmable**

Table 10-1 (Continued) Keywords by Name in Alphabetic Order

HP IBASIC Keyword or Operator	Support*			Exceptions and Comments
	FP	EK	P	
`LOCAL LOCKOUT`		✓	✓	Select Code 7 = all keys Select Code 8 = all keys except Preset
`LOG`		✓	✓	
`LOOP`			✓	
`LORG`		✓	✓	
`LCW$`	✓	✓	✓	
`MAT`		✓	✓	
`MAT REORDER`		✓	✓	
`MAT REORDER...BY`		✓	✓	
`MAT foo=CSUM(bar)`		✓	✓	
`MAT foo=IDN`		✓	✓	
`MAT foo=INV(bar)`		✓	✓	
`MAT foo=RSUM(bar)`		✓	✓	
`MAX`		✓	✓	
`MAXLEN`	✓	✓	✓	
`MAXREAL`		✓	✓	
`MIN`		✓	✓	
`MINREAL`		✓	✓	
`MOD`		✓	✓	
`MODULO`		✓	✓	
`MOVE`		✓	✓	

* FP= Front Panel, EK= External Keyboard, P=Programmable

Table 10-1 (Continued) Keywords by Name in Alphabetic Order

HP IBASIC Keyword or Operator	Support*			Exceptions and Comments
	FP	EK	P	
MOVELINES		✓		Valid Device Selectors #7xx, #7xxxx, #9, #15
MSI	✓	✓	✓	MSI may be altered by the instrument.
NOT	✓	✓	✓	
NUM	✓	✓	✓	
NUT	✓	✓	✓	
ON|OFF CYCLE			✓	
ON|OFF ERROR			✓	
ON|OFF INTR			✓	Interface Select Code = 7 or 8. Must precede ENABLE INTR statement.
ON|OFF KEY			✓	Key selectors 1 through 7
ON|OFF TIMEOUT			✓	Interface Select Code = 7 or 8
OPTION BASE			✓	
OR	✓	✓	✓	
OUTPUT		✓	✓	Select Code 1,7, 8, 9, 15
PASS CONTROL		✓	✓	Select Code 7 or 8
PAUSE	✓	✓	✓	
PDIR		✓	✓	
PEN		✓	✓	0=erase, 1=draw
PENUP		✓	✓	

* FP= Front Panel, EK= External Keyboard, P=Programmable

Table 10-1 (Continued) Keywords by Name in Alphabetic Order

HP IBASIC Keyword or Operator	Support*			Exceptions and Comments
	FP	EK	P	
PI		✓	✓	
PIVOT		✓	✓	
PLOT		✓	✓	
POLYGON		✓	✓	FILL not supported, scaling diffs.
POLYLINE		✓	✓	
POS	✓	✓	✓	
PRINT		✓	✓	
PRINTER IS		✓	✓	
PROUND		✓	✓	
PRT		✓	✓	
PURGE	✓	✓	✓	
RAD	✓	✓	✓	
RANDOMIZE		✓	✓	
RATIO		✓	✓	
RE-SAVE	✓	✓	✓	
RE-STORE	✓	✓	✓	
READ		✓	✓	
READIO		✓	✓	Select Code 9 or 15. See the *HP Instrument BASIC Language Reference.*
REAL			✓	

* FP= Front Panel, EK= External Keyboard, P=Programmable

Table 10-1 (Continued) Keywords by Name in Alphabetic Order

HP IBASIC Keyword or Operator	Support*			Exceptions and Comments
	FP	EK	P	
RECTANGLE		✓	✓	FILL not supported, scaling diffs.
REDIM		✓	✓	
REM			✓	
REMOTE		✓	✓	Select Code 7
REN		✓		
RENAME	✓	✓	✓	
REPEAT UNTIL			✓	
RESTORE			✓	
RETURN			✓	
REV$	✓	✓	✓	
RND		✓	✓	
ROTATE	✓	✓	✓	
RPLOT		✓	✓	FILL not supported, scaling diffs.
RPT$	✓	✓	✓	
RUN	✓	✓	✓	
SAVE	✓	✓	✓	
SCRATCH	✓	✓		
SECURE	✓	✓		
SELECT			✓	
SET TIME foo	✓	✓	✓	

* FP= Front Panel, EK= External Keyboard, P=Programmable

Table 10-1 (Continued) Keywords by Name in Alphabetic Order

HP IBASIC Keyword or Operator	Support*			Exceptions and Comments
	FP	EK	P	
`SET TIMEDATE foo`		✓	✓	
`SGN`		✓	✓	
`SHIFT`	✓	✓	✓	
`SHOW`		✓	✓	Select Code 7
`SIN`	✓	✓	✓	
`SPOLL`		✓	✓	
`SQR`		✓	✓	
`SQRT`		✓	✓	
`STEP`	✓	✓		
`STOP`	✓		✓	
`STORE`	✓	✓	✓	
`SUB`			✓	
`SUBEND`			✓	
`SUBEXIT`			✓	
`SUM`		✓	✓	
`SYSTEM PRIORITY`			✓	
`SYSTEM$`		✓	✓	
`TAB()`		✓	✓	
`TABXY()`		✓	✓	
`TAN`	✓	✓	✓	Select Code 7

* FP= Front Panel, EK= External Keyboard, P=Programmable

Table 10-1 (Continued) Keywords by Name in Alphabetic Order

HP IBASIC Keyword or Operator	Support*			Exceptions and Comments
	FP	EK	P	
TIME		✓	✓	
TIME$		✓	✓	
TRIGGER		✓	✓	
TRIM$	✓	✓	✓	
TRN		✓	✓	
UPC$	✓	✓	✓	
USING		✓	✓	
VAL	✓	✓	✓	
VAL$	✓	✓	✓	
VIEWPORT		✓	✓	
WAIT		✓	✓	
WHERE		✓	✓	
WHILE			✓	
WIDTH		✓	✓	
WINDOW		✓	✓	
WRITEIO		✓	✓	Select Code 9 or 15. See the *HP Instrument BASIC Language Reference.*

*** FP= Front Panel, EK= External Keyboard, P=Programmable**

Keywords in Category Order

Table 10-2 **Program Entry and Editing Commands**

HP IBASIC Keyword or Operator	Support*			Exceptions and Comments
	FP	EK	P	
COPYLINES		✓		
DEL	✓	✓		Front Panel deletes 1 line only.
DELSUB		✓		
EDIT	✓	✓		Front Panel edits default line #. See the *HP Instrument BASIC Language Reference*.
INDENT		✓		
LIST		✓	✓	
MOVELINES		✓		Valid Device Selectors #7xx, #7xxxx, #9, #15
REM			✓	
REN		✓		
SECURE	✓	✓		
*** FP= Front Panel, EK= External Keyboard, P=Programmable**				

Table 10-3 Program Debugging Keywords

HP IBASIC Keyword or Operator	Support*			Exceptions and Comments
	FP	EK	P	
`ERRL()`			✓	
`ERRLN()`		✓	✓	
`ERRM$`		✓	✓	
`ERRN`		✓	✓	
`STEP`	✓	✓		
* FP= Front Panel, EK= External Keyboard, P=Programmable				

Table 10-4 Memory Allocation Keywords

HP IBASIC Keyword or Operator	Support*			Exceptions and Comments
	FP	EK	P	
`ALLOCATE`		✓	✓	
`COM`			✓	
`DEALLOCATE`		✓	✓	
`DELSUB`		✓	✓	
`DIM`			✓	
`INTEGER`			✓	
`LOADSUB`		✓	✓	
`OPTION BASE`			✓	
`REAL`			✓	
`SCRATCH`	✓	✓		Front Panel executes `SCRATCH A`.
*** FP= Front Panel, EK= External Keyboard, P=Programmable**				

Table 10-5 Relational Operators and Basic Math Keywords

Relational Operators and Basic Math Keywords				
HP IBASIC Keyword or Operator	**Support***			**Exceptions and Comments**
	FP	**EK**	**P**	
&	✓	✓	✓	
<,<=,<>,=,>,>=			✓	
*		✓	✓	
+		✓	✓	
-		✓	✓	
/		✓	✓	
^		✓		
ABS		✓	✓	
DIV		✓	✓	
DROUND		✓	✓	
EXP		✓	✓	
FRACT		✓	✓	
INT		✓	✓	
LET		✓	✓	
LGT		✓	✓	
LOG		✓	✓	
MAX		✓	✓	
MAXREAL		✓	✓	
MIN		✓	✓	
* FP= Front Panel, EK= External Keyboard, P=Programmable				

Relational Operators and Basic Math Keywords (Continued)				
HP IBASIC Keyword or Operator	**Support***			**Exceptions and Comments**
	FP	**EK**	**P**	
MINREAL		✓	✓	
MOD		✓	✓	
MODULO		✓	✓	
PI		✓	✓	
PROUND		✓	✓	
RANDOMIZE		✓	✓	
RND		✓	✓	
SGN		✓	✓	
SQR		✓	✓	
SQRT		✓	✓	
* FP= Front Panel, EK= External Keyboard, P=Programmable				

Table 10-6 Binary Math Keywords

HP IBASIC Keyword or Operator	Support*			Exceptions and Comments
	FP	EK	P	
`BINAND`	✓	✓	✓	
`BINCMP`	✓	✓	✓	
`BINEOR`	✓	✓	✓	
`BINIOR`	✓	✓	✓	
`BIT`	✓	✓	✓	
`ROTATE`	✓	✓	✓	
`SHIFT`	✓	✓	✓	
* FP= Front Panel, EK= External Keyboard, P=Programmable				

Table 10-7 Trigonometric Keywords

HP IBASIC Keyword or Operator	Support*			Exceptions
	FP	EK	P	
`ACS`	✓	✓	✓	
`ASN`	✓	✓	✓	
`ATN`	✓	✓	✓	
`COS`	✓	✓	✓	Abs. val. less than 1.7083127722e+10
`DEG`	✓	✓	✓	
`RAD`	✓	✓	✓	
`SIN`	✓	✓	✓	
`TAN`	✓	✓	✓	
* FP= Front Panel, EK= External Keyboard, P=Programmable				

Table 10-8 String Operation Keywords

HP IBASIC Keyword or Operator	Support*			Exceptions
	FP	EK	P	
CHR$	✓	✓	✓	
DVAL$	✓	✓	✓	
IVAL$	✓	✓	✓	
IVAL	✓	✓	✓	
LEN	✓	✓	✓	
LWC$	✓	✓	✓	
MAXLEN	✓	✓	✓	
NUM	✓	✓	✓	
POS	✓	✓	✓	
REV$	✓	✓	✓	
RPT$	✓	✓	✓	
TRIM$	✓	✓	✓	
UPC$	✓	✓	✓	
VAL$	✓	✓	✓	
VAL	✓	✓	✓	

* FP= Front Panel, EK= External Keyboard, P=Programmable

Table 10-9 Logical Operation Keywords

HP IBASIC Keyword or Operator	Support*			Exceptions
	FP	EK	P	
`AND`	✓	✓	✓	
`EXOR`	✓	✓	✓	
`NOT`	✓	✓	✓	
`OR`	✓	✓	✓	

* FP= Front Panel, EK= External Keyboard, P=Programmable

Table 10-10 Mass Storage Keywords

HP IBASIC Keyword or Operator	Support*			Exceptions
	FP	EK	P	
CAT	✓	✓	✓	Supports 58 columns. See the *HP Instrument BASIC Language Reference*.
COPY	✓	✓	✓	
CREATE	✓	✓	✓	
CREATE ASCII		✓	✓	
CREATE BDAT		✓	✓	
CREATE DIR		✓	✓	
GET	✓	✓	✓	See Chapter 4.
INITIALIZE	✓	✓	✓	
LOAD	✓	✓	✓	See Chapter 4.
LOADSUB	✓	✓	✓	
LOADSUB ALL FROM...	✓	✓	✓	
MSI	✓	✓	✓	MSI may be altered by the instrument.
PURGE	✓	✓	✓	
RE-SAVE	✓	✓	✓	
RENAME	✓	✓	✓	
RE-STORE	✓	✓	✓	
SAVE	✓	✓	✓	
STORE	✓	✓	✓	
*** FP= Front Panel, EK= External Keyboard, P=Programmable**				

Table 10-11 Program Control Keywords

Program Control Keywords				
HP IBASIC Keyword or Operator	Support*			Exceptions
	FP	EK	P	
CALL		✓	✓	
CASE			✓	
CASE ELSE			✓	
CONT		✓	✓	Line number support EK only
DEF FN			✓	
ELSE			✓	
END			✓	
END IF			✓	
END LOOP			✓	
END SELECT			✓	
END WHILE			✓	
EXIT IF			✓	
FN			✓	
FNEND			✓	
FOR NEXT			✓	
GOSUB			✓	
GOTO			✓	
IF THEN			✓	
LOOP			✓	
* FP= Front Panel, EK= External Keyboard, P=Programmable				

Program Control Keywords (Continued)				
HP IBASIC Keyword or Operator	Support*			Exceptions
	FP	EK	P	
PAUSE			✓	
REPEAT UNTIL			✓	
RETURN			✓	
RUN	✓	✓	✓	
SELECT			✓	
STOP	✓		✓	
SUB			✓	
SUBEND			✓	
SUBEXIT			✓	
SYSTEM$		✓	✓	
WAIT		✓	✓	
WHILE			✓	

* FP= Front Panel, EK= External Keyboard, P=Programmable

Table 10-12 Program Control: Event Initiated Branching Keywords

HP IBASIC Keyword or Operator	Support*			Exceptions
	FP	EK	P	
`DISABLE`			✓	
`DISABLE INTR`			✓	Interface Select Code = 7 or 8
`ENABLE`			✓	
`ENABLE INTR`			✓	Interface Select Code = 7 or 8. Must not precede ON INTR statement.
`ON|OFF CYCLE`			✓	
`ON|OFF ERROR`			✓	
`ON|OFF INTR`			✓	Interface Select Code = 7 or 8. Must precede ENABLE INTR statement.
`ON|OFF KEY`			✓	Key selectors 1 through 7
`ON|OFF TIMEOUT`			✓	Interface Select Code = 7 or 8
`SYSTEM PRIORITY`			✓	
`TRIGGER`		✓	✓	
* FP= Front Panel, EK= External Keyboard, P=Programmable				

Table 10-13 Graphics Control Keywords

HP IBASIC Keyword or Operator	Support*			Exceptions
	FP	EK	P	
CLIP		✓	✓	Select Code 7
GCLEAR		✓	✓	
GINIT		✓	✓	
RATIO		✓	✓	
SHOW		✓	✓	Select Code 7
VIEWPORT		✓	✓	
WHERE		✓	✓	
WINDOW		✓	✓	

*** FP= Front Panel, EK= External Keyboard, P=Programmable**

Table 10-14 Graphics Plotting Keywords

HP IBASIC Keyword or Operator	Support*			Exceptions
	FP	EK	P	
DRAW		✓	✓	
IDRAW		✓	✓	
IMOVE		✓	✓	
IPLOT		✓	✓	
MOVE		✓	✓	
PDIR		✓	✓	
PEN		✓	✓	0=erase, 1=draw
PENUP		✓	✓	
PIVOT		✓	✓	
PLOT		✓	✓	
POLYGON		✓	✓	FILL not supported, scaling diffs.
POLYLINE		✓	✓	
RECTANGLE		✓	✓	FILL not supported, scaling diffs.
RPLOT		✓	✓	FILL not supported, scaling diffs.
*** FP= Front Panel, EK= External Keyboard, P=Programmable**				

Table 10-15 Graphics Axes and Labeling Keywords

HP IBASIC Keyword or Operator	Support*			Exceptions
	FP	EK	P	
AXES		✓	✓	
CSIZE		✓	✓	
FRAME		✓	✓	
GRID		✓	✓	
LABEL		✓	✓	
LDIR		✓	✓	
LORG		✓	✓	
* FP= Front Panel, EK= External Keyboard, P=Programmable				

Table 10-16 GPIB Control Keywords

HP IBASIC Keyword or Operator	Support*			Exceptions
	FP	EK	P	
ABORT		✓	✓	Select Code = 7, 8, 9, 15
CLEAR		✓	✓	Select Code = 7, 8, 9, 15
LOCAL		✓	✓	Select Code = 7 only
LOCAL LOCKOUT		✓	✓	Select Code 7 = all keys Select Code 8 = all keys except Preset
PASS CONTROL		✓	✓	Select Code = 7 or 8
REMOTE		✓	✓	Select Code = 7
SPOLL		✓	✓	Select Code = 7
* FP= Front Panel, EK= External Keyboard, P=Programmable				

Table 10-17 Clock and Calendar Keywords

HP IBASIC Keyword or Operator	Support*			Exceptions
	FP	EK	P	
DATE		✓	✓	
DATE$		✓	✓	
SET TIME foo	✓	✓	✓	
SET TIMEDATE foo		✓	✓	
TIME		✓	✓	
TIME$		✓	✓	
* FP= Front Panel, EK= External Keyboard, P=Programmable				

Table 10-18 General Device Input/Output Keywords

General Device Input/Output Keywords				
HP IBASIC Keyword or Operator	Support*			Exceptions
	FP	EK	P	
ASSIGN		✓	✓	
BEEP		✓	✓	
CRT		✓	✓	ENTER CRT(ENTER 1) not supported
DATA			✓	
DISP		✓	✓	
DUMP ALPHA		✓	✓	
DVAL	✓	✓	✓	
ENTER		✓	✓	
* FP= Front Panel, EK= External Keyboard, P=Programmable				

General Device Input/Output Keywords (Continued)				
HP IBASIC Keyword or Operator	**Support***			**Exceptions**
	FP	**EK**	**P**	
IMAGE			✓	
INPUT			✓	See the *HP Instrument BASIC Language Reference.*
KBD			✓	Returns Select Code = 2
OUTPUT		✓	✓	Select Code = 1, 7, 8, 9, 15
PRINT		✓	✓	
PRINTER IS		✓	✓	
PRT		✓	✓	
READ		✓	✓	
READIO		✓	✓	Select Code 9 or 15. See the *HP Instrument BASIC Language Reference.*
RESTORE			✓	
TAB()		✓	✓	
TABXY()		✓	✓	
USING		✓	✓	
WIDTH		✓	✓	
WRITEIO		✓	✓	Select Code 9 or 15. See the *HP Instrument BASIC Language Reference.*
*** FP= Front Panel, EK= External Keyboard, P=Programmable**				

Table 10-19 Display and Keyboard Keywords

HP IBASIC Keyword or Operator	Support*			Exceptions
	FP	EK	P	
CLEARSCREEN		✓	✓	
CLS		✓	✓	
* FP= Front Panel, EK= External Keyboard, P=Programmable				

Table 10-20 Array Operation Keywords

HP IBASIC Keyword or Operator	Support*			Exceptions
	FP	EK	P	
DET		✓	✓	
DOT		✓	✓	
FRACT		✓	✓	
MAT		✓	✓	
MAT foo=IDN		✓	✓	
MAT foo=INV(bar)		✓	✓	
MAT foo=CSUM(bar)		✓	✓	
MAT foo=RSUM(bar)		✓	✓	
MAT REORDER		✓	✓	
MAT REORDER...BY		✓	✓	
REDIM		✓	✓	
SUM		✓	✓	
TRN		✓	✓	
* FP= Front Panel, EK= External Keyboard, P=Programmable				

Index

Index

Index