

Notice

Hewlett-Packard to Agilent Technologies Transition

This documentation supports a product that previously shipped under the Hewlett-Packard company brand name. The brand name has now been changed to Agilent Technologies. The two products are functionally identical, only our name has changed. The document still includes references to Hewlett-Packard products, some of which have been transitioned to Agilent Technologies.



Agilent Technologies

**Handbook Supplement
Using HP Instrument BASIC
with the HP 8711C/12C/13C/14C
RF Network Analyzers
and the HP 8730A Tuner Analyzer**



**HP Part No. 08712-90058
Printed in USA October 1996**

Microwave Instruments Division

© Copyright Hewlett-Packard Company 1996
All Rights Reserved. Reproduction, adaptation, or translation without prior written permission
is prohibited, except as allowed under the copyright laws.
1400 Fountaingrove Parkway, Santa Rosa, CA 95403-1799, USA

Contents

1. Introduction	
Overview of HP Instrument BASIC	1-2
Using HP Instrument BASIC	1-2
Allocating Internal Memory for IBASIC Use	1-3
Typographical Conventions	1-3
2. Recording Programs	
Keystroke Recording	2-1
What is Keystroke Recording?	2-1
IBASIC Programs and the HP-IB Buffer	2-2
What's in a Recorded Program	2-2
The OUTPUT Statement	2-3
The ASSIGN Statement	2-3
SCPI Mnemonics	2-3
How Recording Works	2-4
Operations That Do Not Record	2-5
Front Panel Operations Without Mnemonics	2-5
HP Instrument BASIC Operations	2-6
Operations Requiring Additional Programming	2-6
Synchronization	2-6
Active Control of the HP-IB Interface	2-7
Mnemonics With No Corresponding Front Panel Operation	2-7
Avoiding Recording Errors	2-7
Use Instrument Preset	2-7
Specifically Select Parameters	2-8
Use HP-IB Echo	2-8
3. Running, Pausing and Stopping Programs	
Starting Programs Automatically	3-1
Running and Continuing a Program	3-1
Pausing a Program	3-2
Using LOCAL LOCKOUT 8 to Disable the Pause Key	3-3
Stopping a Program	3-4
4. Saving and Recalling Programs	
Selecting a Disk	4-1
Saving a Program	4-2
AUTOST Programs	4-3
Recalling a Program	4-3
CAT to a String Array Exception	4-3

5. Developing Programs	
External Editors	5-2
HP BASIC	5-2
ASCII Word Processors	5-2
Editing Your Program Using Edit	5-3
The IBASIC Editor Softkeys	5-4
Recording into an Existing Program	5-4
Editing with an External Keyboard	5-5
Inserting Lines	5-5
Editing Lines	5-6
Entering Program Lines	5-6
Editing from the Front Panel	5-6
Character Entry	5-7
The Label Window	5-7
Inserting Lines	5-7
Removing Program Text	5-8
Deleting Characters	5-8
Deleting Lines	5-8
Recalling a Deleted Line	5-8
Renumbering, Copying, Moving, and Indenting Lines	5-9
Using IBASIC Display	5-9
Using Utilities	5-12
6. Debugging Programs	
Setting Breakpoints	6-2
Examining Variables	6-2
Examining Strings	6-3
Examining Arrays	6-3
Displaying the Last Error Encountered	6-4
7. Graphics and Display Techniques	
Using the Display Partitions	7-1
Allocating Display Partitions	7-2
De-Allocating Display Partitions	7-3
Operation with No Display Partition	7-3
Displaying Text	7-4
Pop-up Message Windows and Custom Annotations	7-5
Graphics Initialization and Scaling	7-5
Using Graphics	7-6
Drawing Figures	7-7
Graphics Exceptions	7-9
GRID, RECTANGLE, POLYGON, and POLYLINE scaling differences	7-9
Labeling with Different Partitions	7-9
SCPI Graphics Commands	7-10

8. Interfacing with External Devices

Communication with Devices	8-1
HP-IB Device Selectors	8-1
Moving Data Through the HP-IB	8-2
General Structure of the HP-IB	8-2
The System Controller	8-3
Using the Serial and Parallel Ports	8-3
Using the Analyzer Ports in IBASIC programs	8-3
Writeable Ports	8-4
Readable Ports [I=READIO(A,B)]	8-4
General Bus Management	8-5
REMOTE	8-6
Host Instrument	8-6
LOCAL LOCKOUT	8-6
Host Instrument	8-6
LOCAL	8-7
Host Instrument	8-7
TRIGGER	8-7
Host Instrument	8-7
CLEAR	8-7
Host Instrument	8-7
ABORT	8-8
Aborting the Internal Bus	8-8
HP-IB Service Requests	8-8
Setting Up and Enabling SRQ Interrupts	8-8
Servicing SRQ Interrupts	8-9
Example	8-9
Conducting a Serial Poll	8-10
Passing and Regaining Control	8-10
The IBASIC HP-IB Model	8-11
External and Internal Busses	8-11
Service Request Indicators	8-11
IBASIC as the Active Controller	8-12
Passing Active Control to the Instrument	8-12
IBASIC as a Non-Active Controller	8-13
Interfacing with an External Controller	8-13
Synchronizing IBASIC with an External Controller	8-14
Using OUTPUT and ENTER statements	8-14
Using Status information	8-14
Design Rules	8-15
Transferring Data Between Programs	8-15
Using OUTPUT and ENTER statements	8-15
Setting and Querying Variables	8-16
Downloading and Uploading Programs	8-17
Downloading	8-17
Uploading	8-18

9. Using Subprograms	
User-Created Subprograms	9-1
Built-In High-Speed Subprograms	9-2
Example Programs	9-4
Avoiding Multiple Loads of Subprograms	9-4
10. IBASIC Keyword Summary	
11. Example Programs	
Example Program Summaries	11-1
DATA_EXT — Data transfer between internal and external programs	11-1
DATA_INT — Data transfer between internal and external programs	11-1
DOWNLOAD — Download program to analyzer	11-1
DRAW871X — Drawing setup diagrams	11-1
DUALCTRL — Two controller operation	11-2
REPORT — Using the parallel port	11-2
TRICTRL — External controller with local IBASIC controllers	11-2
UPLOAD — Upload program from analyzer	11-2
USERBEG — Set up user-defined User BEGIN softkeys	11-2
USERBEG1 — The default User BEGIN program	11-3
USERBEG2 — Fast recall of instrument states	11-3
USER_BIT — Using the USER bit	11-3
USERKEYS — Customized softkeys	11-3
USR_FLOC — Fault location measurements	11-3
BARCODE, STATS, DATALOG — Bar Code Programs	11-4
Example Program Listings	11-6
DATA_EXT — Data transfer between internal and external programs	11-6
DATA_INT — Data transfer between internal and external programs	11-9
DOWNLOAD — Download program to analyzer	11-11
DRAW871X — Drawing setup diagrams	11-13
DUALCTRL — Two controller operation	11-17
REPORT — Using the parallel port	11-20
TRICTRL — External controller with local IBASIC controllers	11-23
UPLOAD — Upload program from analyzer	11-26
USERBEG — Set up user-defined User BEGIN softkeys	11-28
USERBEG1 — The default User BEGIN program	11-31
USERBEG2 — Fast recall of instrument states	11-33
USER_BIT — Using the USER bit	11-35
USERKEYS — Customized softkeys	11-37
USR_FLOC — Fault location measurements	11-40
BARCODE — Using Bar Code Reader	11-42
STATS — Using Bar Code Reader	11-46
DATALOG — Using Bar Code Reader	11-49

Index

Figures

4-1. The <code>(SAVE RECALL)</code> Screen	4-2
5-1. The HP IBASIC Program Editor	5-3
5-2. The PC Keyboard	5-5
5-3. The IBASIC Display Partitions	5-11
7-1. Display Partitions on the Analyzer	7-1
7-2. Using INPUT with No Display Partition	7-3
7-3. Printing to a Display Partition	7-4
7-4. Pixel Dimensions with Available Display Partitions	7-6
7-5. "HELP" Program Output	7-8
11-1. Sample Bar Codes	11-53

Tables

5-1. IBASIC Display Partitions	5-10
7-1. IBASIC Display Partitions	7-2
7-2. SCPI Graphics Commands	7-11
9-1. Built-in Subprogram Description (Filenames found in :MEM,0,0)	9-3
10-1. Alphabetical List of IBASIC Keywords	10-1
10-2. Categorical List of IBASIC Keywords	10-8

Introduction

A built-in HP Instrument BASIC (IBASIC) controller (Option 1C2) can be ordered with your analyzer. An upgrade kit (HP model no. 86224C) is also available to add the controller to an analyzer that was purchased without this option.

This manual describes creating and using 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 become familiar with the operation of the analyzer before programming it. This manual introduces the IBASIC operating and programming environment and provides examples of intermediate and advanced IBASIC programs. It assumes familiarity with the analyzer and HP BASIC.

Related information can be found in the following references. Contact a Hewlett-Packard sales or service office if you wish to order any of these documents. A list of HP sales and service offices can be found in the "Specifications" chapter of your *User's Guide*.

- Information on the IBASIC language, including keyword descriptions, error messages, interface specifics and programming techniques is available in the *HP Instrument BASIC Users Handbook*.
- Information on operating the analyzer is available in the analyzer's *User's Guide*.
- Information on programming the analyzer, including example programs, is available in the analyzer's *Programmer's Guide*.
- Information on the analyzer's HP-IB command mnemonics is also available in the analyzer's *Programmer's Guide*.
- Information on the SCPI (Standard Commands for Programmable Instruments) programming language is available in *A Beginners Guide to SCPI*.
- Information on using the HP-IB is available in the *Tutorial Description of the Hewlett-Packard Interface Bus*.

Overview of HP Instrument BASIC

When installed in your analyzer, 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 HP-IB commands over an internal interface bus (select code 8). It can also communicate with other instruments, computers, and peripherals using the external HP-IB 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 HP-IB address of 16. When using IBASIC to control other instruments, no other device should use the same address.

The external HP-IB address can be changed using either the front panel keys under the **(SYSTEM OPTIONS)** **HP-IB** 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 HP Instrument BASIC (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 controlling and automating your analyzer.

IBASIC's programming interface includes an editor. Softkeys are available to allow you to run or continue a program or configure the display.

The IBASIC command set is a subset of the command set of HP BASIC. 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.

Using IBASIC, you can:

- Create on screen graphics
- Control other instruments and peripherals
- Create interactive prompts
- Simplify keystrokes with the **(BEGIN)** key
- Keystroke record programs
- 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 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:

<code>(HARDKEY)</code>	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.
<code>Softkey</code>	The label of a softkey.
<code>Softkey ON off</code>	Upper case selection in a softkey indicates the state AFTER the softkey is pressed.
<code>(HARDKEY) Softkey 1 Softkey 2</code>	A series of hardkeys and softkeys represents the path to a given softkey or menu.
<code><element></code>	Angle brackets are used to signify a syntax element in a statement.

Recording Programs

IBASIC programs for the analyzer can be created from the instrument's front panel using an external PC keyboard (Option 1CL) on an HP controller running HP BASIC, or on a workstation or PC using a text editor.

Keystroke recording, described in this chapter, is ideal for creating simple programs or measurement sequences for instrument control. If a program requires data processing, decision making, or prompts for an operator or graphical setup diagrams, these must be entered using another technique. Alternative methods of program development may be used to supplement keystroke recording and create more sophisticated programs. These methods are covered in Chapter 5, "Developing Programs."

Keystroke Recording

Of all the available methods of creating IBASIC programs, keystroke recording is by far the easiest. It requires only a couple of steps to set up and run, and can be accomplished with very little knowledge of programming.

What is Keystroke Recording?

Keystroke recording is a way to automatically create IBASIC measurement sequence programs. To enable recording, simply press **(SYSTEM OPTIONS) IBASIC Key Record ON off**. Then press the normal key sequences of a measurement on the analyzer. Press **(SYSTEM OPTIONS) IBASIC Key Record on OFF** to terminate the recording. The resulting program can then be run by pressing **Run** in the **(SYSTEM OPTIONS) IBASIC** menu.

IBASIC programs communicate with the analyzer over an internal bus. They use the same set of commands used by external controllers for remote operation of the instrument. Keystroke recording works by finding the bus command, called a SCPI mnemonic, that fits each operation performed from the front panel and then building a program line to perform that operation when 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 simply 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 HP-IB Buffer

Recorded programs work by sending HP-IB commands to the instrument.

These commands are queued into 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 completed.

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” chapter in the *Programmer’s Guide*.

What’s in a Recorded Program

If you look at any program created using keystroke recording you will find that it 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 The `ASSIGN` statement, which is automatically created, will vary depending on the model of analyzer you have:

HP 8711C	<code>ASSIGN @Hp8711 TO 800</code>
HP 8712C	<code>ASSIGN @Hp8712 TO 800</code>
HP 8713C	<code>ASSIGN @Hp8713 TO 800</code>
HP 8714C	<code>ASSIGN @Hp8714 TO 800</code>
HP 8730A	<code>ASSIGN @Hp8730 TO 800</code>

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). The data can take several forms but in recorded IBASIC programs it is a string containing commands for the instrument (a mnemonic).

Although the OUTPUT command is very flexible it is used only one way when generated by a recording. 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 HP-IB interface, which is only used 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:

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

SCPI is a standard instrument control programming language providing commands that are common from one product to another, 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 HP-IB 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 HP-IB interfacing using IBASIC refer to Chapter 8, "Interfacing with the HP-IB." The SCPI mnemonics for the analyzer are documented in the *Programmer's Guide*.

How Recording Works

To fully understand IBASIC recording, it is important to understand the relationship between front panel instrument operation and the program that is generated to emulate that operation.

Note SCPI mnemonics entered in a program during a recording session do not have a one-to-one correlation with the actual keys that are pressed during that session.

The fact that the generated SCPI mnemonics do not exactly correspond to the keys actually pressed is important to remember. As you press a sequence of keys to perform an operation, the corresponding SCPI mnemonic for that **operation** is generated. The operation may take one keystroke or several, but the mnemonic is not generated until after a valid sequence of keystrokes is completed.

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

For example, recording the simple key sequence: **POWER** **Level** **[-]** **[1]** **[0]** **Enter** requires six keystrokes and produces only one mnemonic, "SOUR1:POW -10 dBm", which is generated **after** the sequence is completed. This is then automatically formed into the command:

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

and inserted into the program.

This means that if you accidentally press the wrong key in a sequence, it may not show up in the recorded program. Additionally, you cannot exactly mimic keystrokes to leave the instrument in a specific front panel state, unless it is a state that 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, there are some operations that cannot be captured or can only be partially captured using this method. These generally fall into one of the following areas:

- Front panel operations with no corresponding SCPI mnemonic (such as transitional key sequences).
- IBASIC front panel operations (such as some of the softkey operations found under the **SYSTEM OPTIONS** **IBASIC** menu).
- Operations requiring additional programming steps (such as passing control of the HP-IB to the instrument for hardcopy output).
- HP-IB operations with no front panel equivalent (such as HP-IB query commands or data transfer).
- Service menu keys (in general)

Note Do not recall programs in keystroke record mode; doing so will overwrite previously recorded program steps.

Front Panel Operations Without Mnemonics

There are some areas of front panel operation which have no corresponding SCPI mnemonics.

- Most operations on the front panel that require numeric entry allow you to use the knob to increment or decrement the current value. This will not record as a program line. You must 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 you setup prior to recording or 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 the same exact state when the program is run as when it was recorded. This is discussed further in the "Avoiding Recording Errors" section of this chapter.

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 the HP-IB.”

Operations Requiring Additional Programming

Some operations that work well when performed from the front panel have circumstances that require special attention when used in a program. This is due to two kinds of problems, synchronization and active control.

Synchronization

Timing and synchronization must always be anticipated where one event must complete before another can occur. One example of this is when you need to detect a state in the instrument before issuing the next command. For example, 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.

Note

Synchronization is only a problem with **overlapped** commands (such as the command to trigger a sweep), that is commands that don't hold off the processing of subsequent commands. The analyzer adds an extra command *WAI when an overlapped command is created using keystroke recording. *WAI prevents the analyzer from executing any further commands until the overlapped command has finished. For more information on synchronization see the “Synchronizing the Analyzer and a Controller” chapter in the *Programmer's Guide*.

Active Control of the HP-IB Interface

Some operations require the analyzer to be the active controller on the external HP-IB bus. This generally means that the analyzer must be the System Controller (or active control must be passed to it from an external controller, if one is connected). When an IBASIC program begins running, however, the instrument's active 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 HP-IB devices. Note that active control of the HP-IB 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 HP-IB.

(HARD COPY) Start

(HARD COPY) Abort

You can keystroke record 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 the "Passing and Regaining Control" section of Chapter 8 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 which, therefore, cannot be recorded. These include operations such as querying instrument status, transferring data over HP-IB, setting and clearing status registers and general HP-IB housekeeping.

These operations are useful for the more advanced HP-IB 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 HP-IB command set. See also "Built-In High Speed Subprograms" in Chapter 9.

Avoiding Recording Errors

Use Instrument Preset

In most cases, it is recommended that the **(PRESET)** key/operation be recorded as the first keystroke recorded. This sets the instrument to its default state and avoids the risk of creating a program that depends on instrument settings that were present at the time of the keystroke recording but may be different when the program is run.

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 the instrument preset before a recorded program is run (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 HP-IB Echo

HP-IB 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 HP-IB Echo, press **SYSTEM OPTIONS** **HP-IB** and **HP-IB 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 is the exact mnemonic that is generated in your recorded program during a recording session.

Using HP-IB 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.

Running, Pausing and Stopping Programs

Program control — running, pausing and stopping an IBASIC program — can be managed from the analyzer front panel using various hardkeys and softkeys. These actions and their corresponding keys are described in this chapter.

A special case is an autostart program which runs automatically on power-up if it exists on the analyzer's built-in floppy disk drive or RAM disk.

IBASIC programs may also be remotely controlled via SCPI commands over the HP-IB. For information on running, pausing and stopping programs from an external controller see Chapter 8, "Interfacing with External Devices."

Starting Programs Automatically

When the analyzer is powered up, it automatically searches first 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.

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 either of two 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.

Note When an external keyboard is connected, its 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 was supplied with your analyzer. (HP part number 08712-80028.)

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

The prerun initialization phase consists of:

- Reserving memory space 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 only initialized the first time a program is run.

- Checking 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 has been successfully completed, the program will begin the execution phase. Program lines will be 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 a Program

When an IBASIC program is running on the analyzer a softkey menu is always available. 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. This menu automatically appears when a program is paused. 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: **Pause** key locked-out from within a program.

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

Example: 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 until it is cleared. 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 Users Handbook*, contained in this binder.

Saving and Recalling Programs

IBASIC programs can reside 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 **(SAVE RECALL) Programs** menu. To access the **(SAVE RECALL) 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.

Another mode of program transfer is between the analyzer and an external controller, such as an HP Series 200/300/700 controller. Using an external controller, you can combine the convenience of keystroke recording in IBASIC with the ease of program editing in a dedicated external workstation by recording the measurement sequence and then uploading the program to the external controller for further editing. Fully developed programs may be downloaded from an external controller as well. The methods of transferring programs between the analyzer and an external controller are 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. If desired, specify the type of file, binary or ASCII, with the **File Type** softkey; default 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 being changed for each new file. For portability, save files in ASCII. See "User-Created Subprograms" in Chapter 9 for when it is necessary to save files in binary format.

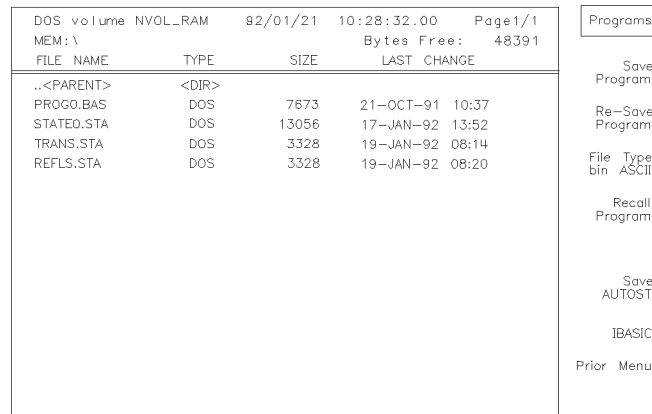


Figure 4-1. The **(SAVE RECALL) Programs** Screen

If you are re-saving a program — that is, saving a file to a disk that already contains the file name — press **(SAVE RECALL) 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.

Use ASCII Format Whenever Possible

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.
- Binary format may not be supported in future releases of firmware or analyzers.

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 first 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 Any program recalled to the program buffer using the **(SAVE RECALL) 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." The difference is that 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 of 80).

Developing Programs

For many applications, you can use keystroke recording to create and run programs without needing to alter the program code that is generated. 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 programs from the ground up.

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 places you in the editor where you can make changes to your program on a line-by-line basis.

IBASIC Display menu allows you to select what part, if any, of the CRT display is available for the use of IBASIC. An IBASIC display partition provides you with a command line you can use to execute 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 using the built-in IBASIC editor, programs can be developed in the following external environments.

- HP BASIC editors
- ASCII word processors

The external editing environments provide many advantages, the most notable being 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).

1. Partition an IBASIC display (as described later in this chapter).
2. Use an external keyboard to enter the command GET "PR0G0:,4" (this command loads a program file PR0G0 from the internal floppy disk drive).
3. Watch the IBASIC display as the program is loaded — syntax errors result in error messages 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 no syntax checking occurs until the program is loaded by the instrument. Another complication with using a word processor is that 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.

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 somewhat similar to the instrument's IBASIC editor; others should find it easy to learn and use. This section tells you how to edit and enter an IBASIC program.

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.

CALL	PRINT	ABORT	SUB	SUBEND	DATA	LOCAL	DIM	BIT	ABCDEFGHIJKLM	Edit
------	-------	-------	-----	--------	------	-------	-----	-----	---------------	------

```

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"

```

Prior Menu

Figure 5-1. The HP IBASIC Program Editor

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 in the “Editing from the Front Panel” section of this chapter.

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 which you want the recorded statements inserted.
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 **(SYSTEM OPTIONS) IBASIC Key Record on OFF** to conclude the recording session.

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

Note 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 analyzer and the IBASIC editor work with IBM PC-AT compatible keyboards (US only) that have a mini-DIN interface. 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, HP part number 1252-4141.

The PC-AT keyboard, Figure 5-2, has four major key areas: the typewriter keypad, the numeric keypad, the cursor keypad, and the function keys. Alphanumeric text can be entered using the typewriter and numeric keypads as needed. The cursor keypad can be used to move the cursor up/down a line or left/right to the next character positions. The function keys of the keyboard map to the softkeys on the analyzer front panel.

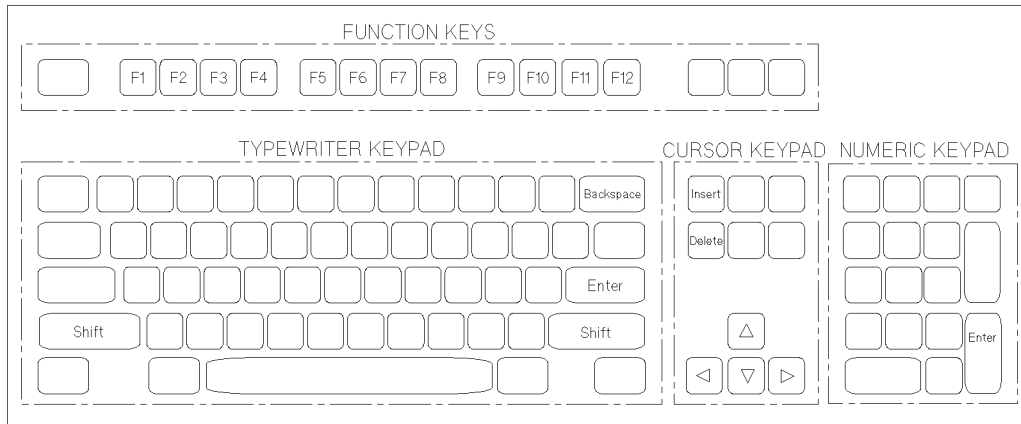


Figure 5-2. The PC Keyboard

Connect the keyboard to the rear panel DIN KEYBOARD connector of the analyzer with the power off. Turn on the power and load the IBASIC program to be edited. Select the **(SYSTEM OPTIONS) IBASIC Edit** menu and use the cursor keypad to position the cursor within the program for editing operations. The Page Up and Page Down keys on the keyboard scroll through the program quickly and easily.

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 renumbered one higher to accommodate the inserted line.

To stop inserting lines either 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. Remember 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 while the up and down arrow keys move between lines. The alphanumeric keypad on the keyboard can be used for entering or editing text. Another key that is useful is the **(Delete)** key, which 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 then stores the line in the program buffer.

Entering Program Lines

When you finish entering or changing a program line, to store it into the program buffer you must ENTER 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.

Note 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.

Editing from the Front Panel

Use the step keys to move the cursor up and down the lines in the program. When the cursor is located at the beginning of a 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:

<code>Select Char/Word</code>	<code>(F1)</code>	Inserts the character or word highlighted by the label window cursor at the position marked by the program cursor.
<code>Space</code>	<code>(F2)</code>	Inserts a space at the position marked by the program cursor.
<code>Delete Char</code>	<code>(F3)</code>	Deletes the character highlighted by the program cursor.
<code>Backspace</code>	<code>(F4)</code>	Deletes the last character before the program cursor.
<code>Enter</code>	<code>(F5)</code>	Enters the edited program line.
<code>Prior Menu</code>	<code>(F8)</code>	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 from within 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 functions the same 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 maintains its column-relative position on 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 resting 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, simply move the cursor up one line.)

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 buffered 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, simply 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 simply moves the cursor over line numbers without deleting the number.

Note

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.
 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.

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 CRT 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 When the UPPER or LOWER display partition is selected, the measurement display automatically selects the “split-screen” format. This format uses half of the CRT 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*.

Most display allocation should be handled by your program via the SCPI mnemonics. These softkeys are best utilized 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 shows the relative size and location of the different IBASIC partitions and their command and display lines.

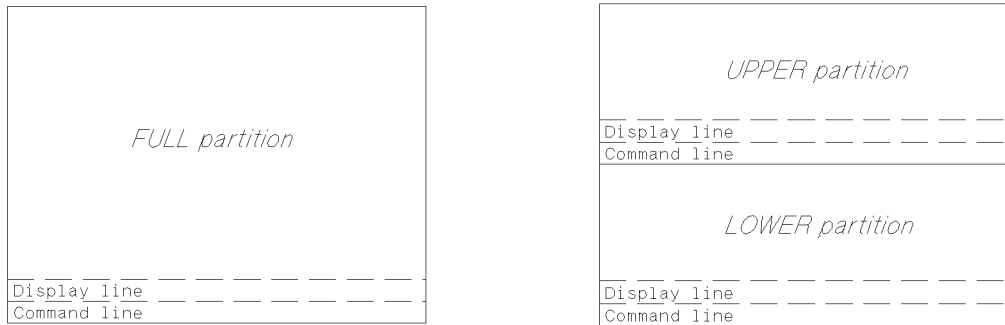


Figure 5-3. The IBASIC Display Partitions

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

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

Executing the **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.

Memory Size allows you to set 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 **Memory 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:

- 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 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.

Debugging Programs

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

Of course bugs can and do appear in even the best designed programs and IBASIC contains some features that can help you to track them down. Some IBASIC capabilities useful for program debugging are simple and, used properly, can be very helpful. Some of these capabilities are:

- 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 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 interface. 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, HP 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 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. Pausing the program can be accomplished by pressing the **Pause** softkey (**F8**) on an external keyboard) that is available 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" in Chapter 5, "Developing Programs.") 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 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:

```
100 100 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 using the `OUTPUT` command to control the analyzer, when a command syntax is unrecognized or incorrect. For more details on SCPI errors, refer to the *Programmer's Guide*.

Graphics and Display Techniques

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 "Automating Measurements" in the *User's Guide*.

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 the three measurement display areas, called display partitions, can be used by an IBASIC program. These partitions are shown in Figure 7-1.

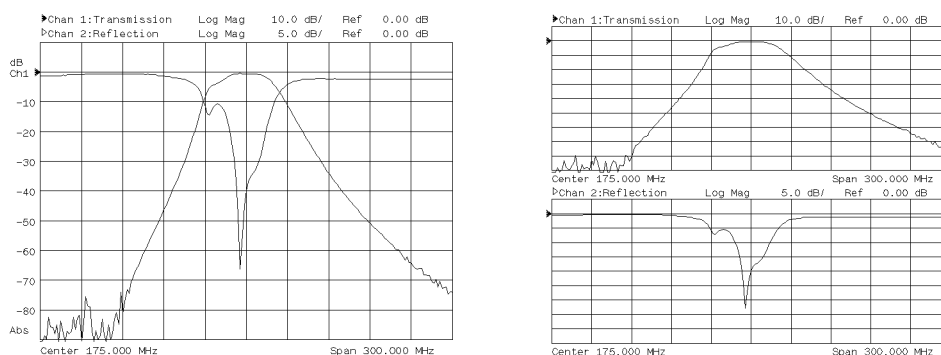


Figure 7-1. Display Partitions on the Analyzer

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 the 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:PROGm OFF	No Display
Full (F2)	DISPlay:PROGm FULL	The Whole Display
Upper (F3)	DISPlay:PROGm UPPer	Upper Measurement Channel Area
Lower (F4)	DISPlay:PROGm 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 the upper partition, "DISP:PROG LOWer" allocates the lower partition, and "DISP:PROG FULL" allocates the 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, even better, 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:PROG UPPer"
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 and then 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 to the right of the measurement display. This area is reserved for softkey labels. It can be 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 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 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.

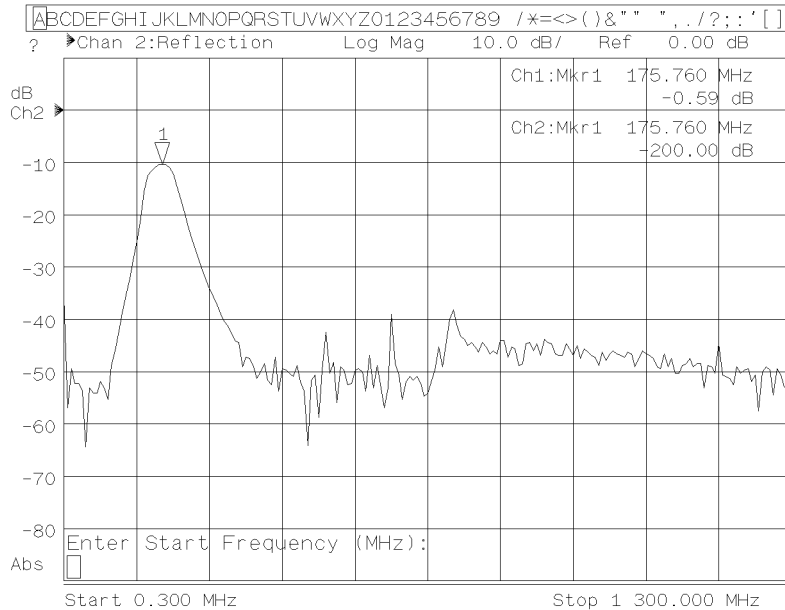


Figure 7-2. Using INPUT with No Display Partition

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 the “SCPI Graphics Commands” section of this chapter.

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 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.

This information is useful 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).

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

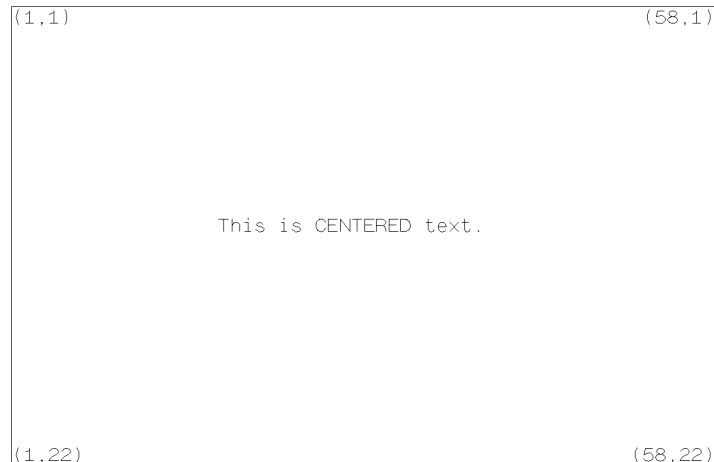


Figure 7-3. Printing to a Display Partition

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 Temp$;"This is the first line of text"
80 Big$=Big$&Temp$
90 OUTPUT Temp$;"This is the second line of text"
100 Big$=Big$&Temp$
110 PRINTER IS CRT; WIDTH 2000
120 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 150 below writes a string to the screen.

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

Pop-up Message Windows and Custom Annotations

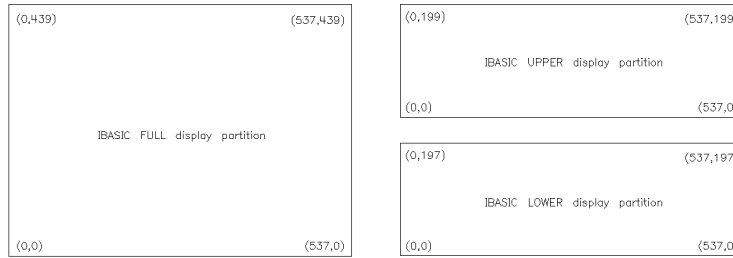
From your IBASIC program, you can replace instrument annotations with user-defined annotations. You can change the X-axis labels and measurement channel annotations to customize the display. Pop-up messages may also be used to display permanent or temporary messages. Refer to "Automating Measurements"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

Graphics Initialization and Scaling

In all partitions, display coordinate 0,0 is at the bottom left corner and clipping occurs automatically 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 GDU's (Graphical Display Units) high and 122 GDU's wide (assuming full partition). This gives a RATIO result of 1.22 and provides the same results as issuing a WINDOW 0,122,0,100 command. In order to prevent circles from appearing oval 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, allowing integers only to be used; thus speeding execution. These are also the same values that are returned by utilizing 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.



hp61c

Figure 7-4. Pixel Dimensions with Available Display Partitions

Using Graphics

IBASIC's 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 also, the section titled "Graphics Exceptions," later in this chapter.

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

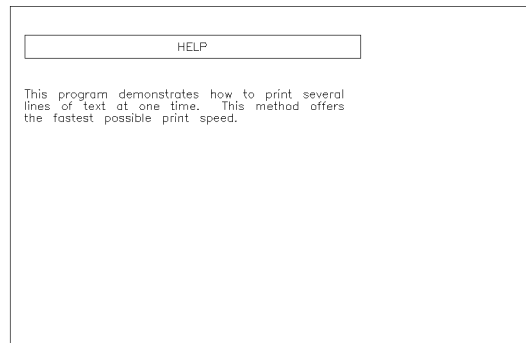
LORG - Defines the relative origin of a label

These keywords are used in the "BARCODE" program example listed in Chapter 11, "Example Programs", and on the IBASIC 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
```



pd62a

Figure 7-5. "HELP" Program Output

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.

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

LINE TYPE - The analyzer does not support different line types.

GRID, RECTANGLE, POLYGON, and POLYLINE scaling differences

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

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

Note

When converting programs from previous models (such as HP 8711A/B and family) to run on an HP 8711C/12C/13C/14C or HP 8730A, scaling differences will affect all graphics commands. The "C" family of network analyzers as well as the HP 8730A tuner analyzer 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 may be used to label graphs, however, the following should be noted. 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 in that they do not require an IBASIC display partition. This means that they can be used to write or draw directly to a measurement window.

These commands, listed in Table 7-2 are SCPI mnemonics and are programmable from an external controller as well as from IBASIC. The commands are of the 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 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	Draw 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	Clear the user graphics and graphics buffer for the specified window.
DISPlay:WINDow[1 2 10]:GRAPhics:COLor <num>	NR1	Set 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	Draw 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	Draw a label with the lower left corner at the current pen location.
DISPlay:WINDow[1 2 10]:GRAPhics:LABel:FONT 	CHAR	Select 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	Move 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	Draw 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>		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,” earlier in this chapter.)

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 Chapter 8 of the *Programmer’s Guide*.

Interfacing with External Devices

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

The HP-IB interface is Hewlett-Packard's implementation of the IEEE-488.2 Digital Interface for Programmable Instrumentation. The acronym HP-IB stands for "Hewlett-Packard 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 an HP-IB instrument controller residing inside an instrument. It uses the instrument's HP-IB interface for external communication and an internal HP-IB interface to communicate with the instrument. This unique arrangement presents a few differences between IBASIC's implementation of HP-IB control and HP BASIC controllers. A description of the interaction of IBASIC with the host instrument and the external HP-IB interface is given in the section entitled "The IBASIC HP-IB Model", later in this chapter.

Communication with Devices

HP-IB Device Selectors

Since the HP-IB allows several devices to be interconnected, each device must be uniquely identified. Specifying the select code of the HP-IB interface (such as 7 or 8) 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 HP-IB 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 HP-IB 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 HP-IB interface (select code 7).

The second part of an HP-IB 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 HP-IB 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 device's address must be unique. The analyzer is shipped from the factory with a primary address of 16. No other device on the bus should use the same address.

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

Moving Data Through the HP-IB

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

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

Devices that communicate over the HP-IB 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 the communication occurring on the bus. It specifies which device talks, which device listens and when the exchange of data takes place.

An HP-IB 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** **HP-IB** 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 HP-IB 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, 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 two additional ports that can be used to control peripherals, material handlers or other devices. Active control of the HP-IB interface is not needed when these ports are being used. These ports are a parallel port and a serial port for use with hardcopy output to non-HP-IB printers and plotters.

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 use 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 HP-IB commands with `READIO` and `WRITEIO`. `READIO` and `WRITEIO` are faster than HP-IB commands.

Writeable Ports

WRITEIO	15,0;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). Checks Centronics status lines for <ul style="list-style-type: none">• 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.)

Readable Ports [I=READIO(A,B)]

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 <ul style="list-style-type: none">• D0 – Cent_acknowledge• D1 – Cent_busy• D2 – Cent_out_of_paper• D3 – Cent_on_line• D4 – Cent_printer_err

An example program, REPORT, demonstrating peripheral control over the parallel port is provided in Chapter 11, “Example Programs.”

Refer to “Automating Measurements” in the *Users Guide* for further explanation and examples of how to access the analyzer’s I/O ports.

General Bus Management

The HP-IB standard provides several mechanisms that allow managing 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 HP-IB states.

CLEAR — sets selected (or all) devices to a pre-defined, device-dependent HP-IB 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).

These statements (and functions) are described in the following discussion. However, the actions that a device takes upon receiving each of the above commands are, in general, 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 active controller on the internal interface (select code 8). For the program to be active controller on the external interface (select code 7), the instrument must either be set as system controller or have control passed to it from an external controller. The program automatically assumes the controller status of the host instrument. For more information refer to "The IBASIC HP-IB Model" section later in this chapter.

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

REMOTE

Most HP-IB devices can be controlled either 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 HP-IB, it is in the Remote state. Unless operating in the Local Lockout mode, each HP-IB 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 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 HP-IB 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 HP-IB 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 HP-IB 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.

HP-IB Service Requests

Most HP-IB 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 (executes remedial action).

Setting Up and Enabling SRQ Interrupts

In order for an HP-IB 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:

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

```
ASSIGN @Device T0 700
Status_byte=SPOLL(@Device)
```

```
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 HP-IB 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 HP-IB interface. The concept of using pass control with IBASIC is discussed in the next section, “The IBASIC HP-IB Model.”

The IBASIC HP-IB 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.

External and Internal Busses

There is physically only one HP-IB port and one HP-IB address for the analyzer. IBASIC has access to two HP-IB 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 SRI's will be set to their OR'd value when a program starts, and again when it finishes. This assures that any pending SRQ's 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 example in Chapter 11, “Example Programs.”)

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 HP-IB 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 active controller on the external interface:

- When the host instrument is set as System Controller and the program has not passed control
- 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 HP-IB 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 HP-IB hardware.
-------------	---

IBASIC as a Non-Active Controller

IBASIC programs are always the active controller on the internal interface. There are two cases where an IBASIC program does not have control of the external HP-IB interface:

- When the host instrument is set as Talker/Listener and active control has **NOT** been passed from an external device
- When the host instrument is set as System Controller and the program has passed control to either the host instrument 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 the “Interfacing with an External Controller” section that follows).

Interfacing with an External Controller

So far, we have discussed the ability to interface IBASIC programs with a network of external devices using the HP-IB. 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 HP-IB 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 HP-IB model, presented earlier in this chapter.

Second, the external program can make use of the extensive set of analyzer HP-IB 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 HP-IB 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 then may 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 Chapter 11.

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>; *0PC?"
```

```
ENTER 716; 0pc
```

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 *0PC, *0PC?, *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 such that 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. This means that 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 HP-IB while acting as a non-active controller.

Note	Moving data through the HP-IB 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, that are listed in Chapter 11, "Example Programs," 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 name of the drive to catalog, and then outputs the catalogued string to the external program and 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 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 "PROG:SELEcted]:NUMBer" and "PROG:SELEcted]:STRing" mnemonics (and their query counterparts) are part of the analyzer HP-IB 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
```

Downloading and Uploading Programs

Programs can be transferred between an external controller and program memory using the HP-IB download command "PROG:SELected:DEFine" and its upload query "PROG: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 HP-IB 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 HP-IB control lines.

Refer to Chapter 11, "Example Programs" 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 CRT 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).

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 Chapter 11, “Example Programs” 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 to save the uploaded program to. The number of 256 byte records declared in the `CREATE ASCII` statement (line 730) is simply 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.

Although this simple method works for many uploaded programs, there may still be a problem with the file size caused by the `OUTPUT` statement in line 490. This is because every ASCII line in a LIF file contains a two byte length header and possibly one additional pad byte to make the length an even number of bytes. These extra bytes are not included in the definite length block header information. You can account for this extra overhead by allocating an extra 10 to 15 percent of space when you create the ASCII file. For example, the `Openfile` subroutine could be rewritten as:

```
570 SUB Openfile(@File,Filename$,Fsize)
680  ON ERROR GOTO Openerr
715  Fsize=Fsize+(Fsize*0.15)
720  IF Fsize MOD 256>0 THEN Fsize=Fsize+256
730  CREATE ASCII Filename$,Fsize DIV 256
```

Using Subprograms

Analyzer products shipped with the IBASIC option can run subprograms. The subprograms may be user-created or built-in.

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 RMB manual. 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 Chapter 9 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.
Built-in Subprogram Description (Filenames found in : MEM,0,0)

Filename	Subprogram Name (parameter list)	Description
XFER	Read_fdata(INTEGER Chan,REAL A(*))	Read real formatted data
	Read_fmем(INTEGER Chan,REAL A(*))	Read real formatted mem
	Read_cdata(INTEGER Chan,REAL A(*))	Read complex data
	Read_cmем(INTEGER Chan,REAL A(*))	Read complex memory
	Write_fdata(INTEGER Chan,REAL A(*))	Write real formatted data
	Write_fmем(INTEGER Chan,REAL A(*))	Write real formatted mem.
	Write_cdata(INTEGER Chan,REAL A(*))	Write complex data
	Write_cmем(INTEGER Chan,REAL A(*))	Write complex memory
	Read_rdata(INTEGER Chan,Input\$, REAL A(*))	Read raw complex data
	Write_rdata(INTEGER Chan,Input\$,REAL A(*))	Write raw complex data
	Read_corr(INTEGER Chan, N,REAL A(*))	Read complex error coef.
	Write_corr(INTEGER Chan, N,REAL A(*))	Write complex error coef.
	MATH	
Define Complex Array Operations	Cmplx_mag(REAL Cdata(*),Mag(*),INTEGER Sz)	Mag of complex array
	Cmplx_arg(REAL Cdata(*),Arg(*),INTEGER Sz) !Arg of complex array	
	Cmplx_conjg(REAL A(*),B(*)) !Complex conj of array A to B	
Define Complex Number Operations	Cadd(REAL Op1(*),INTEGER Row1,REAL Op2(*),INTEGER Row2,REAL Ans(*),INTEGER Rowans)	Complex Ans = Op1 + Op2
	Csub(REAL Op1(*),INTEGER Row1,REAL Op2(*),INTEGER Row2,REAL Ans(*),INTEGER Rowans)	Complex Ans = Op1 - Op2
	Cmul(REAL Op1(*),INTEGER Row1,REAL Op2(*),INTEGER Row2,REAL Ans(*),INTEGER Rowans)	Complex Ans = Op1 * Op2
	Cdiv(REAL Op1(*),INTEGER Row1,REAL Op2(*),INTEGER Row2,REAL Ans(*),INTEGER Rowans)	Complex Ans = Op1 / Op2
RPG	SUB Rpg_function (INTEGER function)	Redirect RPG & STEP keys function 0 = normal 1 = Actv MKR & INPUT 2 = Actv MKR & LABELS

Example Programs

```
1
10 LOADSUB Read_fdata FROM "XFER:MEM,0,0"
```

*Example use of built in subprograms
Appends Read_data sub program to end of this program.
This subprogram can now be called.*

```
20
30
40
50 REAL Trace_array(1:201)
60 Read_fdata( 1, Trace_array(*))
70 LOADSUB ALL FROM "MATH:MEM,0,0"
```

*Reads Channel 1 data into
Trace_array(*)
Appends all math subprograms
defined in "MATH" to the end
of this program.*

```
80 END
90 SUB Read_fdata(INTEGER Chan,REAL A(*))
100 SUB Cmplx_mag(REAL Cdata(*),Mag(*),INTEGER Sz)
110 SUB Cmplx_arg(REAL Cdata(*),Arg(*),INTEGER Sz)
120 ...
```

*Read real formatted data.
Mag of complex array.
Arg of complex array.*

Note 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.

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.

Avoiding Multiple Loads of Subprograms

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

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


IBASIC Keyword Summary

This chapter summarizes the HP Instrument BASIC keyword implementation in the analyzer. Table 10-1 is alphabetical. It indicates the type of support for each entry and notes exceptions, if any. Exceptions are major differences between the keywords descriptions in the “HP Instrument BASIC Language Reference” and their implementation in the analyzer. When differences are too extensive to be summarized, see the “HP Instrument BASIC Language Reference.”

Table 10-2 contains the same information as Table 10-1, but is organized by category.

Table 10-1. Alphabetical List of IBASIC Keywords

HP IBASIC Keyword	Support FP = Front Panel EK = External Keyboard P = Programmable	Exceptions
&	FP,EK,P	
*	EK,P	
+	EK,P	
-	EK,P	
/	EK,P	
<.<=,<>,>,>=	P	
ABORT	EK,P	Select Code = 7,8,9,15
ABS	EK,P	
ACS	FP,EK,P	
ALLOCATE	EK,P	
AND	FP,EK,P	
ASN	FP,EK,P	
ASSIGN	EK,P	
ATN	FP,EK,P	
AXES	EK,P	
BEEP	EK,P	
BINAND	FP,EK,P	
BINCMP	FP,EK,P	
BINEOR	FP,EK,P	
BINIOR	FP,EK,P	
BIT	FP,EK,P	

Table 10-1. Alphabetical List of IBASIC Keywords (continued)

HP IBASIC Keyword	Support FP = Front Panel EK = External Keyboard P = Programmable	Exceptions
CALL	EK,P	
CASE	P	
CASE ELSE	P	
CAT	FP,EK,P	Supports 58 columns. See manual.
CHR\$	FP,EK,P	
CLEAR	EK,P	Select Code = 7,8,9,15
CLEAR SCREEN	EK,P	
CLS	EK,P	
COM	P	
CONT	EK,FP	Line number support from EK only
COPY	FP,EK,P	
COPYLINES	EK	
COS	FP,EK,P	Abs vals less than 1.7083127722e+10
CREATE		
CREATE ASCII	EK,P	
CREATE BDAT	EK,P	
CREATE DIR		
CRT	EK,P	ENTER CRT(ENTER 1) not supported
CSIZE	EK,P	
DATA	P	
DATE	EK,P	
DATE\$	EK,P	
DEALLOCATE	EK,P	
DEF FN	P	
DEG	FP,EK,P	
DEL	FP,EK	Front Panel deletes only 1 line
DELSUB	EK,P	
DET	EK,P	
DIM	P	
DISABLE	P	
DISABLE INTR	P	Interface Select Code = 7 or 8
DISP	EK,P	
DIV	EK,P	
DOT	EK,P	
DRAW	EK,P	

Table 10-1. Alphabetical List of IBASIC Keywords (continued)

HP IBASIC Keyword	Support FP = Front Panel EK = External Keyboard P = Programmable	Exceptions
DROUND	EK,P	
DUMP ALPHA	none	Use HPIB command
DVAL	FP,EK,P	
DVAL\$	FP,EK,P	
EDIT	FP,EK	Front Panel EDITs default line #
ELSE	P	
ENABLE	P	
ENABLE INTR	P	Interface Select Code = 7 or 8
END	P	
END IF	P	
END LOOP	P	
END SELECT	P	
END WHILE	P	
ENTER	EK,P	
ERRL()	P	
ERRLN()	EK,P	
ERRM\$	EK,P	
ERRN	EK,P	
EXIT IF	P	
EXOR	FP,EK,P	
EXP	EK,P	
FN	P	
FNEND	P	
FOR NEXT	P	
FRACT	EK,P	
FRAME	EK,P	
GCLEAR	EK,P	
GET	FP,EK,P	
GINIT	EK,P	
GOSUB	P	
GOTO	P	
GRID	EK,P	
IDRAW	EK,P	
IF THEN	P	

Table 10-1. Alphabetical List of IBASIC Keywords (continued)

HP IBASIC Keyword	Support FP = Front Panel EK = External Keyboard P = Programmable	Exceptions	
IMAGE	P	See Manual.	
IMOVE	EK,P		
INDENT	EK		
INITIALIZE	FP,EK,P		
INPUT	P		
INT	EK,P		
INTEGER	P		
ILOT	EK,P		
IVAL	FP,EK,P		
IVAL\$	FP,EK,P		
KBD	P		Returns select code =2.
LABEL	EK,P		
LDIR	EK,P		
LEN	FP,EK,P		
LET	EK,P		
LGT	EK,P		
LIST	EK,P		Valid Device Selectors
LOAD	FP,EK,P		
LOADSUB	FP,EK,P		
LOADSUB ALL FROM ...	FP,EK,P		
LOCAL	EK,P	Select Code 7 only.	
LOCAL LOCKOUT	EK,P	Select Code 7 only.	
LOG	EK,P		
LOOP	P		
LORG	EK,P		
LWC\$	FP,EK,P		
MAT	EK,P		
MAT REORDER	EK,P		
MAT REORDER ... BY	EK,P		
MAT foo=CSUM(bar)	EK,P		
MAT foo=IDN	EK,P		
MAT foo=INV(bar)	EK,P		
MAT foo=RSUM(bar)	EK,P		

Table 10-1. Alphabetical List of IBASIC Keywords (continued)

HP IBASIC Keyword	Support FP = Front Panel EK = External Keyboard P = Programmable	Exceptions
MAX	EK,P	
MAXLEN	FP,EK,P	
MAXREAL	EK,P	
MIN	EK,P	
MINREAL	EK,P	
MOD	EK,P	
MODULO	EK,P	
MOVE	EK,P	
MOVELINES	EK	
MSI	FP,EK,P	MSI may be altered by the instr.
NOT	FP,EK,P	
NUM	FP,EK,P	
ON OFF CYCLE	P	
ON OFF ERROR	P	
ON OFF INTR	P	Interface Select Code = 7 or 8
ON OFF KEY	P	Key selectors 1 through 7
ON OFF TIMEOUT	P	Interface Select Code = 7 or 8
OPTION BASE	P	
OR	FP,EK,P	
OUTPUT	EK,P	Select Code 1,7,8,9,15
PASS CONTROL	EK,P	Select Code 7 or 8
PAUSE	EK,FP,P	
PDIR	EK,P	
PEN	EK,P	0 = erase 1 = draw
PENUP	EK,P	
PI	EK,P	
PIVOT	EK,P	
PLOT	EK,P	
POLYGON	EK,P	FILL not supported. Scaling diffs.
POLYLINE	EK,P	
POS	FP,EK,P	
PRINT	EK,P	
PRINTER IS	EK,P	
PROUND	EK,P	

Table 10-1. Alphabetical List of IBASIC Keywords (continued)

HP IBASIC Keyword	Support FP = Front Panel EK = External Keyboard P = Programmable	Exceptions
PRT	EK,P	
PURGE	FP,EK,P	
RAD	FP,EK,P	
RANDOMIZE	EK,P	
RATIO	EK,P	
RE-SAVE	FP,EK,P	
RE-STORE	FP,EK,P	
READ	EK,P	
READIO	EK,P	Select Code 9 or 15. See manual.
REAL	P	
RECTANGLE	EK,P	FILL not supported. Scaling diffs.
REDIM	EK,P	
REM	P	
REMOTE	EK,P	Select Code 7
REN	EK	
RENAME	FP,EK,P	
REPEAT UNTIL	P	
RESTORE	P	
RETURN	P	
REV\$	FP,EK,P	
RND	EK,P	
ROTATE	FP,EK,P	
RPLOT	EK,P	Fill not supported. Scaling diffs.
RPT\$	FP,EK,P	
RUN	EK,FP,P	
SAVE	FP,EK,P	
SCRATCH	FP,EK	Front Panel executes SCRATCH A.
SECURE	FP,EK	
SELECT	P	
SET TIME foo	FP,EK,P	
SET TIMEDATE foo	EK,P	
SGN	EK,P	
SHIFT	FP,EK,P	
SHOW	EK,P	

Table 10-2. Categorical List of IBASIC Keywords

HP Instrument BASIC Keyword	Support FP = Front Panel EK = External Keyboard P = Programmable	Exceptions
<p>Program Entry/Editing</p> <p>COPYLINES DEL DELSUB EDIT</p> <p>INDENT LIST</p> <p>MOVELINES REM REN SECURE</p> <p>Program Debugging</p> <p>ERRL() ERRLN() ERRM\$ ERRN STEP</p> <p>Memory Allocation</p> <p>ALLOCATE COM DEALLOCATE DELSUB DIM INTEGER LOADSUB OPTION BASE REAL SCRATCH</p> <p>Relational Operators</p> <p><.<=,<>,=>,>=</p> <p>General Math</p> <p>* + - /</p>	<p>EK FP,EK EK FP,EK</p> <p>EK EK,P</p> <p>EK P EK FP,EK</p> <p>P EK,P EK,P EK,P FP,EK</p> <p>EK,P P EK,P EK,P P P P FP,EK</p> <p>P</p> <p>EK,P EK,P EK,P EK,P</p>	<p>Front Panel deletes only 1 line.</p> <p>Front Panel EDITS default line #. See Manual.</p> <p>Valid Device Selectors #7xx, #7xxxx, #9, #15.</p> <p>Front Panel executes SCRATCH A.</p>

Table 10-2. Categorical List of IBASIC Keywords (continued)

HP Instrument BASIC Keyword	Support FP = Front Panel EK = External Keyboard P = Programmable	Exceptions
ABS	EK,P	
DIV	EK,P	
DROUND	EK,P	
EXP	EK,P	
FRACT	EK,P	
INT	EK,P	
LET	EK,P	
LGT	EK,P	
LOG	EK,P	
MAX	EK,P	
MAXREAL	EK,P	
MIN	EK,P	
MINREAL	EK,P	
MOD	EK,P	
MODULO	EK,P	
PI	EK,P	
PROUND	EK,P	
RANDOMIZE	EK,P	
RND	EK,P	
SGN	EK,P	
SQR	EK,P	
SQRT	EK,P	
~	EK,P	
Binary Functions		
BINAND	FP,EK,P	
BINCMP	FP,EK,P	
BINEOR	FP,EK,P	
BINIOR	FP,EK,P	
BIT	FP,EK,P	
ROTATE	FP,EK,P	
SHIFT	FP,EK,P	
Trigonometric Operations		
ACS	FP,EK,P	
ASN	FP,EK,P	
ATN	FP,EK,P	
COS	FP,EK,P	Abs vals less than 1.7083127722e+10
DEG	FP,EK,P	

Table 10-2. Categorical List of IBASIC Keywords (continued)

HP Instrument BASIC Keyword	Support FP = Front Panel EK = External Keyboard P = Programmable	Exceptions
RAD SIN TAN	FP,EK,P FP,EK,P FP,EK,P	
String Operations		
&	FP,EK,P	
CHR\$	FP,EK,P	
DVAL\$	FP,EK,P	
DVAL	FP,EK,P	
IVAL\$	FP,EK,P	
IVAL	FP,EK,P	
LEN	FP,EK,P	
LWC\$	FP,EK,P	
MAXLEN	FP,EK,P	
NUM	FP,EK,P	
POS	FP,EK,P	
REV\$	FP,EK,P	
RPT\$	FP,EK,P	
TRIM\$	FP,EK,P	
UPC\$	FP,EK,P	
VAL\$	FP,EK,P	
VAL	FP,EK,P	
Logical Operations		
AND	FP,EK,P	
EXOR	FP,EK,P	
NOT	FP,EK,P	
OR	FP,EK,P	
Mass Storage		
CAT	FP,EK,P	Supports 58 columns. See manual.
COPY	FP,EK,P	
CREATE		
CREATE ASCII	EK,P	
CREATE BDAT	EK,P	
CREATE DIR		
GET	FP,EK,P	

Table 10-2. Categorical List of IBASIC Keywords (continued)

HP Instrument BASIC Keyword	Support FP = Front Panel EK = External Keyboard P = Programmable	Exceptions
INITIALIZE LOAD LOADSUB LOADSUB ALL FROM ... MSI	FP,EK,P FP,EK,P FP,EK,P FP,EK,P FP,EK,P	MSI may be altered by the instr. When save/recalling programs to/from DOS subdirectories.
PURGE RE-SAVE RENAME RE-STORE SAVE STORE	FP,EK,P FP,EK,P FP,EK,P FP,EK,P FP,EK,P FP,EK,P	
Program Control		
CALL	EK,P	
CASE	P	
CASE ELSE	P	
CONT	EK,FP	Line number support from EK only
DEF FN	P	
ELSE	P	
END	P	
END IF	P	
END LOOP	P	
END SELECT	P	
END WHILE	P	
EXIT IF	P	
FN	P	
FNEND	P	
FOR NEXT	P	
GOSUB	P	
GOTO	P	
IF THEN	P	
LOOP	P	
PAUSE	EK,FP,P	
REPEAT UNTIL	P	

Table 10-2. Categorical List of IBASIC Keywords (continued)

HP Instrument BASIC Keyword	Support FP = Front Panel EK = External Keyboard P = Programmable	Exceptions
RETURN RUN SELECT STOP SUB SUBEND SUBEXIT SYSTEM\$ WAIT WHILE	P EK,FP,P P FP,P P P P EK,P EK,P P	
Event Initiated Branching		
DISABLE DISABLE INTR ENABLE ENABLE INTR	P P P P	Interface Select Code = 7 or 8.
ON OFF CYCLE ON OFF ERROR ON OFF INTR	P P P	Interface Select Code = 7 or 8. Must not precede an ON INTR statement.
ON OFF KEY ON OFF TIMEOUT SYSTEM PRIORITY	P P P	Interface Select Code = 7 or 8. Must precede ENABLE INTR statement. Key selectors 1 through 7 Interface Select Code = 7 or 8
Graphics Control		
GCLEAR GINIT RATIO SHOW VIEWPORT WHERE WINDOW	EK,P EK,P EK,P EK,P EK,P EK,P EK,P	
Graphics Plotting		
DRAW IDRAW IMOVE	EK,P EK,P EK,P	

Table 10-2. Categorical List of IBASIC Keywords (continued)

HP Instrument BASIC Keyword	Support FP = Front Panel EK = External Keyboard P = Programmable	Exceptions
IPLOT MOVE PDIR PEN PENUP PIVOT PLOT POLYGON POLYLINE RECTANGLE RPLOT	EK,P EK,P EK,P EK,P EK,P EK,P EK,P EK,P EK,P EK,P	0 = erase 1 = draw FILL not supported. Scaling diffs. FILL not supported. Scaling diffs. FILL not supported. Scaling diffs.
Graphics Axis and Labeling		
AXES CSIZE FRAME GRID LABEL LDIR LORG	EK,P EK,P EK,P EK,P EK,P EK,P EK,P	
HP-IB Control		
ABORT CLEAR LOCAL LOCAL LOCKOUT PASS CONTROL REMOTE SPOLL TRIGGER	EK,P EK,P EK,P EK,P EK,P EK,P EK,P EK,P	Select Code = 7,8,9,15 Select Code = 7,8,9,15 Select Code 7 only. Select Code 7 only. Select Code 7 or 8 Select Code 7 Select Code 7 Select Code 7
Clock and Calendar		
DATE DATE\$ SET TIME foo SET TIMEDATE foo TIME TIME\$	EK,P EK,P FP,EK,P EK,P EK,P EK,P	

Example Programs

Example Program Summaries

This chapter contains listings of the example programs referred to throughout this manual. These programs are all available on the *IBASIC Example Programs Disk* that accompanies this manual.

In addition to these example programs, there are two additional disks of examples for the analyzer. These disks are the *Example Programs Disk — DOS Format* and the *Example Programs Disk — LIF Format*. These disks are included with the analyzer when it is delivered. All the programs on these disks are designed to run on the analyzer's internal IBASIC controller.

The example programs on the *IBASIC Example Programs Disk*, some of which are listed in this chapter, include the following:

DATA_EXT — Data transfer between internal and external programs

This program is designed to run on an external controller — either HP IBASIC for Windows running on a PC or HP BASIC running on an HP workstation.

This program demonstrates how to transfer data from an IBASIC program running on the analyzer to an HP BASIC or IBASIC program running externally. It loads a program into the analyzer, runs it, sets a variable and then gives it control of the bus. This program then acts as a device on the bus (sending and receiving data).

DATA_INT — Data transfer between internal and external programs

This program is designed to run on the analyzer's internal IBASIC controller.

This program demonstrates how to transfer data to and from an external controller. In this example a catalog listing is transferred from the analyzer to the external controller. A numeric variable value is also downloaded from the external controller to the analyzer's program.

DOWNLOAD — Download program to analyzer

This program demonstrates how to download an IBASIC program to the analyzer. It is designed (in HP BASIC or HP IBASIC for Windows) to run on an external workstation or PC.

DRAW871X — Drawing setup diagrams

This program draws the analyzer and a device under test to the full screen IBASIC display partition. The drawing can be scaled to fit the application. This program uses the analyzer's graphics commands for drawing. To use the IBASIC drawing commands, see the "BARCODE" program.

DUALCTRL — Two controller operation

This program demonstrates how the external controller and HP IBASIC can work together. It is designed to run on an external controller (in HP BASIC or HP IBASIC for Windows). The program downloads an IBASIC program to the analyzer and runs it twice. After each run, two program variables are read from the analyzer and displayed.

REPORT — Using the parallel port

This program uses the analyzer to generate a report, making a hardcopy on a printer connected to the parallel port. It uses a subprogram to send the output to the parallel port one line at a time. Before using this program, be sure that your printer is configured to ignore the Printer_select Centronics signal, since the WRITEIO command does not assert this signal.

TRICTRL — External controller with local IBASIC controllers

This example program demonstrates how an external controller can be used with two instruments running IBASIC. Run this program on an external controller. Connect two instruments via HP-IB cables to the external controller. Set one instrument to address 16, set the other instrument to address 18.

This program insures that only the analyzer or the local IBASIC is sending SCPI commands at one time. This is one possible implementation of synchronizing the analyzer and a controller. Refer to “Automating Measurements” in the *User’s Guide*.

The external controller is responsible for downloading the IBASIC program to each analyzer. The external controller sets the status reporting to send a SRQ whenever a user requested service request occurs.

When all instrument configuration has completed, the external controller sends a “run program” command to each analyzer and then goes into an idle loop. The external controller remains in the idle loop until either instrument sends an SRQ.

While the external controller is idle, each instrument can freely send various SCPI commands. Each instrument may ask for service by triggering an SRQ. Once a SRQ has been triggered, the instrument must remain in an idle loop, until the external controller indicates it is done servicing the SRQ. This is done using the program variable “Ctrl_flag”. The flag is cleared when the external controller is done and has returned to its idle loop.

UPLOAD — Upload program from analyzer

This program uploads the IBASIC program in the analyzer’s program buffer to an ASCII file on the external controller’s current mass storage device.

USERBEG — Set up user-defined User BEGIN softkeys

This program creates User BEGIN softkeys which allow the user to Save or Recall one of two instrument states, set the marker to maximum, set the scale/div, and compute some measurement statistics at the marker.

USERBEG1 — The default User BEGIN program

The default User BEGIN program is created automatically when there is no IBASIC program installed. In this default program, softkey 3 is defined to be the marker-to-max function; softkey 4 prompts the user for a title, and also enables the clock. You may edit this program to change the functions you need.

USERBEG2 — Fast recall of instrument states

This example program demonstrates the fast recall of previously defined instrument states. The instrument states SETUP1, SETUP2, and SETUP3 must have been previously saved to the analyzer's internal non-volatile RAM disk. Load the program into the analyzer. Then press the **BEGIN** key. Enable the user-defined **BEGIN** by pressing User BEGIN ON. When User BEGIN is enabled, the **BEGIN** softkeys will be labeled "Setup1," "Setup2," and "Setup3." To recall each setup, select the appropriate softkey.

USER_BIT — Using the USER bit

This program demonstrates how to read and write to the USER bit. The USER bit is a TTL signal accessible by a BNC connector on the analyzer's rear panel. IBASIC's graphics commands are used to draw the USER bit value to the display.

USERKEYS — Customized softkeys

This program provides an example of how the analyzer's softkeys can be customized. The example demonstrates how to set up six instrument states, store them to the analyzer's internal memory, and setup two interactive softkey menus to choose between them.

USR_FLOC — Fault location measurements

This example program is for analyzers with Option 100 only. This program creates User BEGIN softkeys that can be used for making a fault location measurement.

BARCODE, STATS, DATALOG — Bar Code Programs

You may use bar code readers to simplify your measurement setups. The HPCCK-1210 KeyWand scanner or compatible bar code scanner will work with the analyzer. Connect your bar code scanner to the DIN keyboard connector. You may connect a keyboard or other DIN key input device in parallel with the bar code scanner. The bar code scanner will work in place of, or in addition to, your keyboard.

The INPUT statement is used to read the bar code from the scanner. When the input statement is encountered, the program will wait until the user has completed an input. The input is completed whenever a carriage return is received from the keyboard or a bar code has been successfully scanned by the bar code scanner.

The following three programs, designed to run on the analyzer's internal IBASIC controller, demonstrate the use of bar code scanner applications as well as other useful applications. While a bar code scanner is useful in demonstrating these programs, it is not required; one can simply press ENTER and the program will input default values. Sample bar codes are provided for experimentation at the end of this chapter.

The three programs are as follows:

BARCODE - This program demonstrates basic bar code scanning to select one of three filter setups depending upon what is scanned. RF stimulus is set and response limits are read, set and tested for each device. Depending upon result, the program prints "PASS" or "FAIL" on the CRT. Most useful in this program is a subprogram to draw an analyzer representation on the CRT. This code can be re-used in any user application that may require a guided setup.

The analyzer image (and DUT image) can be both scaled to any size, and offset in the X or Y axis as required. This is an excellent program to familiarize yourself with graphic routines using IBASIC graphics commands.

Although most of the other example programs can be used on an external computer, this program is intended to be used in the analyzer's internal IBASIC environment only.

STATS - This program first reads a DUT bar code and sets the RF stimulus accordingly. It then displays a running average of all similar devices and constantly updates the display with both the current DUT and the current average of all devices tested so far. Also demonstrates the use of two of the built-in CSUB routines for reading and writing trace data from/to the analyzer.

DATALOG - This program will very quickly store measured trace data for one of three filters to internal analyzer memory in a format that can be read by spreadsheet programs for further analysis. Because the data is stored to RAM, the time delay inherent with disks is not an issue; trace data can be stored in a fraction of a second. With 101 data points per trace selected, the internal memory will hold over 200 device test results. At this point, the program automatically transfers the data to disk. Of course, more data points will take longer to store and fill the memory sooner. The program will read the bar code and select the stimulus accordingly. It then measures the device and upon request, stores it under a unique name dependent upon model number and serial number. Once the internal memory is full, or at any user requested time, all trace data is transferred to disk.

Note

These three programs use a different command to preset the analyzer. This service command is `DIAG:PRES:SERV` instead of the more usual `SYST:PRES` command. There are two reasons for this:

1. Since the preset power level can be user defined, the normal preset command will not guarantee a 0 dBm level. This service command will force the power level to be set to 0 dBm.
 2. This service command is required to force the HP 8730A to preset to the same state as an HP 871xC. Failure to do this will prevent proper operation of these programs.
-

Example Program Listings

DATA_EXT — Data transfer between internal and external programs

```
10  !-----
20  !
30  ! BASIC program:  DATA_EXT - Data transfer (external)
40  !
50  ! This program demonstrates how to transfer data from
60  ! an IBASIC program running on the HP 8711 to an
70  ! HP BASIC program (or an IBASIC program running
80  ! externally).  This program was designed to run on a
90  ! computer or PC.  It loads a program into the HP 8711,
100 ! runs it, and then gives it control of the bus.
110 ! This program then acts as a device on the bus;
120 ! sending and receiving data.
130 !
140 ! Before running this program, a disc with the program
150 ! 'DATA_INT' should be in the HP 8711's internal drive.
160 !
170 !-----
180 !
190 ! Initialize variables for the interface select code
200 ! and the HP-IB address of the HP 8711.
210 !
220 Scode=7
230 Address=16
240 Na=Scode*100+Address
250 !
260 ! Abort any bus traffic, clear the input/output queues
270 ! of the analyzer, clear the analyzer's status
280 ! registers and the display.
290 !
300 ABORT Scode
310 CLEAR Na
320 OUTPUT Na;"*CLS"
330 CLEAR SCREEN
340 !
350 ! Dimension an array to hold the catalog listing.
360 !
370 DIM Directory$(1:100)[85]
380 !
390 ! Prompt the operator to insert the disk in the
400 ! HP 8711, load the program and wait until done.
410 !
420 INPUT "Put disc with program 'DATA_INT' into the HP 8711. Press <ENTER>",$
430 DISP "Loading program on HP 8711 . . ."
440 OUTPUT Na;"PROG:EXEC 'GET ""DATA_INT:INTERNAL""'"
450 OUTPUT Na;"*OPC?"
460 ENTER Na;0pc
470 !
480 ! Read the analyzer's event status register and
490 ! check for any errors when loading file.
```

```

500  !
510  OUTPUT Na;"*ESR?"
520  ENTER Na;Esr
530  IF Esr>0 THEN
540      BEEP
550      DISP "Error occurred while loading 'DATA_INT' . . . Program stopped."
560      STOP
570  END IF
580  !
590  ! Determine the HP-IB address of the controller
600  ! and set the pass control back address.
610  !
620  INTEGER Stat,Addr
630  STATUS 7,3;Stat
640  Addr=BINAND(Stat,31)
650  OUTPUT Na;"*PCB ";Addr
660  !
670  ! Send the command to run the DATA_INT program.
680  !
690  DISP "Running the program..."
700  OUTPUT Na;"PROG:STAT RUN"
710  !
720  ! Monitor the program's status.  When it has
730  ! paused, set the variable for the controller's
740  ! HP-IB address.
750  !
760  OUTPUT Na;"PROG:STAT?"
770  ENTER Na;Prog$
780  IF Prog$<>"PAUS" THEN GOTO 760
790  OUTPUT Na;"PROG:NUMB 'Host'," ;Scode*100+Addr
800  OUTPUT Na;"PROG:STAT CONT"
810  !
820  ! Pass control of the bus to the HP 8711.
830  !
840  PASS CONTROL Na
850  !
860  ! Wait until addressed to talk by the HP 8711,
870  ! then send the name of the disk to catalog.
880  !
890  OUTPUT Scode;" :INTERNAL"
900  !
910  ! Wait until addressed to listen by the HP 8711,
920  ! then read the directory from the analyzer.
930  !
940  DISP "Reading data . . ."
950  ENTER Scode;Directory$(*)
960  !
970  ! Print the catalog to the controller's display.
980  !
990  FOR I=1 TO 100
1000     IF LEN(Directory$(I))>0 THEN PRINT Directory$(I)
1010  NEXT I
1020  !
1030  ! Try to return the HP 8711 to LOCAL control.
1040  ! If the analyzer is still the active controller

```

```
1050 ! an error will be generated and the program
1060 ! will loop until control of the bus is received.
1070 !
1080 ON ERROR GOTO 1090
1090 LOCAL Na
1100 DISP ""
1110 END
```

DATA_INT — Data transfer between internal and external programs

```
10  !-----
20  !
30  ! IBASIC program:  DATA_INT - Data transfer (internal)
40  !
50  ! This program demonstrates how to transfer data to
60  ! and from an external controller.  In this example a
70  ! catalog listing is transferred from the HP 8711 to
80  ! the external controller.  For more information look
90  ! at the program listing for 'DATA_EXT'.
100 !
110 ! This IBASIC program is intended to run on the
120 ! HP 8711's internal controller.
130 !
140 !-----
150 !
160 ! Dimension an array to hold the catalog listing.
170 !
180 DIM Directory$(1:100)[85]
190 !
200 ! Pause the program and wait for the controller to
210 ! set the 'Host' variable with its' HP-IB address.
220 ! The controller continues this program after the
230 ! variable has been passed.
240 !
250 Host=0
260 PAUSE
270 !
280 ! Address the external controller to talk, read
290 ! the device to catalog.  If the HP 8711 is not
300 ! active controller on the bus an error will occur
310 ! and the program will loop until control is
320 ! received.
330 !
340 ON ERROR GOTO 340
350 ENTER Host;Stor_dev$
360 OFF ERROR
370 !
380 ! Catalog the requested storage device into
390 ! the string array.
400 !
410 DISP "Reading catalog..."
420 CAT Stor_dev$ TO Directory$(*)
430 !
440 ! Address the external controller to listen,
450 ! send the catalog array to the controller.
460 !
470 DISP "Transferring data..."
480 OUTPUT Host;Directory$(*)
490 !
500 ! Pass control back to the external controller.
510 !
520 PASS CONTROL Host
530 DISP "DONE"
```

540 END

DOWNLOAD — Download program to analyzer

```
10  !-----
20  !
30  ! BASIC program:  DOWNLOAD - Download program to Rfna
40  !
50  ! This program demonstrates how to download an IBASIC
60  ! program to the 871x.  This program is designed to
70  ! run on an external controller.
80  !
90  !-----
100 !
110 ! Initialize variables for the interface select code
120 ! and the HP-IB address of the HP 8711.
130 !
140 Scode=7
150 Address=16
160 Na=Scode*100+Address
170 !
180 ! Initialize variables, abort any bus traffic and
190 ! clear the input/output queues of the analyzer.
200 !
210 DIM Line$[255]
220 ABORT Scode
230 CLEAR Na
240 !
250 ! Get the program's filename and open the file.
260 !
270 Get_filename: INPUT "Program to be transferred?",Filename$
280 ON ERROR GOTO No_file
290 DISP "Checking file . . ."
300 ASSIGN @Basic_prog TO Filename$;FORMAT ON
310 OFF ERROR
320 !
330 ! Clear the contents of the analyzer's program buffer.
340 !
350 OUTPUT Na;"PROG:DEL:ALL"
360 !
370 ! Change the EOL (end of line) character to line feed
380 ! and initialize the line counter.
390 !
400 Transfer: ASSIGN @Prog TO Na;EOL CHR$(10)
410 Line_count=0
420 !
430 ! Initiate the program transfer (an indefinite length
440 ! block data transfer).
450 !
460 OUTPUT @Prog;"PROG:DEF #0";
470 !
480 ! Read each program line from the file and send it to
490 ! the HP 8711.  Loop until the end of file is reached.
500 !
510 ON ERROR GOSUB End_file
```

```

520 LOOP
530 ENTER @Basic_prog;Line$
540 OUTPUT @Prog;Line$
550 Line_count=Line_count+1
560 DISP "Lines transferred: ";Line_count
570 END LOOP
580 !
590 ! End the data transfer (output a line feed with EOI)
600 ! and close the file. Return the analyzer to LOCAL
610 ! control and stop this program.
620 !
630 End_block: OUTPUT @Prog;CHR$(10) END
640 ASSIGN @Basic_prog TO *
650 DISP "Transfer complete"
660 LOCAL Na
670 STOP
680 !
690 ! This subroutine is the error handler for opening
700 ! the file - if the file won't open it returns to
710 ! get a new file name.
720 !
730 No_file: BEEP
740 DISP "CAN'T OPEN: "";Filename$;"" -- ";
750 GOTO Get_filename
760 RETURN
770 !
780 ! This subroutine is the error handler for the
790 ! data transfer. When the end of file is reached
800 ! it generates an error. Execution is resumed
810 ! outside of the transfer loop.
820 !
830 End_file: IF ERRN=59 THEN GOTO End_block
840 DISP ERRM$;" occurred during data transfer"
850 STOP
860 RETURN
870 END

```

DRAW871X — Drawing setup diagrams

```
1 !-----
2 !
3 ! IBASIC program:  DRAW871X - Drawing setup diagrams
4 !
5 ! This program draws the HP 871X network analyzer
6 ! and a device under test to the full screen IBASIC
7 ! display partition.  The drawing can be scaled to
8 ! fit the application.  Setting the scale factor to
9 ! 1.0 creates a drawing of about 400 pixels wide
10 ! (1/2 screen width) and 100 pixels high (1/3 screen
11 ! height).
12 !
13 !-----
14 !
15 ! Setup an I/O path name for the internal bus and
16 ! declare variables.
17 !
18 INTEGER X0,Y0
19 REAL Scale
20 ! Make @Hp87xx common to all subroutines
21 COM /Sys_state/ @Hp87xx,Scode
22 ! Identify the computer we are running on
23 ! and assign the i/o port address to @Hp87xx
24 CALL Iden_port
25 !
26 !
27 ! Preset the analyzer and wait until it is done.
28 !
29 OUTPUT @Hp87xx;"SYST:PRES;*OPC?"
30 ENTER @Hp87xx;0pc
31 ! Allocate the full screen as an IBASIC display
32 ! and clear the graphics buffer.
33 !
34 OUTPUT @Hp87xx;"DISP:PROG FULL"
35 OUTPUT @Hp87xx;"DISP:WIND10:GRAP:CLEAR"
36 !
37 ! Rescale the display window for the new VGA display
38 output @Hp87xx;"DISP:WIND10:GRAP:SCAL 0,1023,0,383"
39 !
40 ! Setup the origin and scale parameters for the
41 ! drawing.  Draw the network analyzer and dut.
42 !
43 X0=100
44 Y0=100
45 Scale=1.
46 CALL Draw_na(X0,Y0,Scale)
47 CALL Draw_dut(X0,Y0,Scale)
48 END
49 !-----
50 SUB Draw_na(INTEGER X0,INTEGER Y0,REAL Sc)
51 !-----
```

```

52 !
53 ! This subroutine draws the HP 8711 at origin XO,YO
54 ! and scale Sc. The drawing is done to the IBASIC
55 ! display (window 10) using the HP 8711's user
56 ! graphics commands.
57 !
58 !-----
59 COM /Sys_state/ @Hp87xx,Scode
60 OUTPUT @Hp87xx;"DISP:WIND10:GRAP:MOVE "&VAL$(XO)&","&VAL$(YO)
61 OUTPUT @Hp87xx;"DISP:WIND10:GRAP:RECT "&VAL$(INT(Sc*350))&","
&VAL$(INT(Sc*100))
62 OUTPUT @Hp87xx;"DISP:WIND10:GRAP:MOVE "&VAL$(XO+INT(Sc*10))&","
&VAL$(YO+INT(Sc*10))
63 OUTPUT @Hp87xx;"DISP:WIND10:GRAP:RECT "&VAL$(INT(Sc*180))&","
&VAL$(INT(Sc*80))
64 OUTPUT @Hp87xx;"DISP:WIND10:GRAP:MOVE "&VAL$(XO+INT(Sc*200))&""
&VAL$(YO+INT(Sc*80))
65 OUTPUT @Hp87xx;"DISP:WIND10:GRAP:RECT "&VAL$(INT(Sc*15))&","&
VAL$(INT(Sc*8))
66 OUTPUT @Hp87xx;"DISP:WIND10:GRAP:MOVE "&VAL$(XO+INT(Sc*200))&""
&VAL$(YO+INT(Sc*70))
67 OUTPUT @Hp87xx;"DISP:WIND10:GRAP:RECT "&VAL$(INT(Sc*15))&","&
VAL$(INT(Sc*8))
68 OUTPUT @Hp87xx;"DISP:WIND10:GRAP:MOVE "&VAL$(XO+INT(Sc*200))&""
&VAL$(YO+INT(Sc*60))
69 OUTPUT @Hp87xx;"DISP:WIND10:GRAP:RECT "&VAL$(INT(Sc*15))&","&
VAL$(INT(Sc*8))
70 OUTPUT @Hp87xx;"DISP:WIND10:GRAP:MOVE "&VAL$(XO+INT(Sc*200))&""
&VAL$(YO+INT(Sc*50))
71 OUTPUT @Hp87xx;"DISP:WIND10:GRAP:RECT "&VAL$(INT(Sc*15))&","&
VAL$(INT(Sc*8))
72 OUTPUT @Hp87xx;"DISP:WIND10:GRAP:MOVE "&VAL$(XO+INT(Sc*200))&""
&VAL$(YO+INT(Sc*40))
73 OUTPUT @Hp87xx;"DISP:WIND10:GRAP:RECT "&VAL$(INT(Sc*15))&","&
VAL$(INT(Sc*8))
74 OUTPUT @Hp87xx;"DISP:WIND10:GRAP:MOVE "&VAL$(XO+INT(Sc*200))&""
&VAL$(YO+INT(Sc*30))
75 OUTPUT @Hp87xx;"DISP:WIND10:GRAP:RECT "&VAL$(INT(Sc*15))&","&
VAL$(INT(Sc*8))
76 OUTPUT @Hp87xx;"DISP:WIND10:GRAP:MOVE "&VAL$(XO+INT(Sc*200))&""
&VAL$(YO+INT(Sc*20))
77 OUTPUT @Hp87xx;"DISP:WIND10:GRAP:RECT "&VAL$(INT(Sc*15))&","&
VAL$(INT(Sc*8))
78 OUTPUT @Hp87xx;"DISP:WIND10:GRAP:MOVE "&VAL$(XO+INT(Sc*200))&""
&VAL$(YO+INT(Sc*10))
79 OUTPUT @Hp87xx;"DISP:WIND10:GRAP:RECT "&VAL$(INT(Sc*15))&","&
VAL$(INT(Sc*8))
80 OUTPUT @Hp87xx;"DISP:WIND10:GRAP:MOVE "&VAL$(XO+INT(Sc*265))&""
&VAL$(YO+INT(Sc*80))
81 OUTPUT @Hp87xx;"DISP:WIND10:GRAP:RECT "&VAL$(INT(Sc*70))&","&
VAL$(INT(Sc*13))
82 OUTPUT @Hp87xx;"DISP:WIND10:GRAP:MOVE "&VAL$(XO+INT(Sc*230))&""
&VAL$(YO+INT(Sc*81))
83 OUTPUT @Hp87xx;"DISP:WIND10:GRAP:RECT "&VAL$(INT(Sc*20))&","&
VAL$(INT(Sc*10))

```

```

84 OUTPUT @Hp87xx;"DISP:WIND10:GRAP:MOVE "&VAL$(XO+INT(Sc*275))&"
,"&VAL$(YO+INT(Sc*85))
85 OUTPUT @Hp87xx;"DISP:WIND10:GRAP:RECT "&VAL$(INT(Sc*50))&","&
VAL$(INT(Sc*3))
86 OUTPUT @Hp87xx;"DISP:WIND10:GRAP:MOVE "&VAL$(XO+INT(Sc*295))&"
,"&VAL$(YO+INT(Sc*50))
87 OUTPUT @Hp87xx;"DISP:WIND10:GRAP:CIRC "&VAL$(INT(Sc*8))
88 OUTPUT @Hp87xx;"DISP:WIND10:GRAP:MOVE "&VAL$(XO+INT(Sc*245))&"
,"&VAL$(YO+INT(Sc*15))
89 OUTPUT @Hp87xx;"DISP:WIND10:GRAP:CIRC "&VAL$(INT(Sc*4))
90 OUTPUT @Hp87xx;"DISP:WIND10:GRAP:MOVE "&VAL$(XO+INT(Sc*325))&"
,"&VAL$(YO+INT(Sc*15))
91 OUTPUT @Hp87xx;"DISP:WIND10:GRAP:CIRC "&VAL$(INT(Sc*4))
92 SUBEND
93 !-----
94 SUB Draw_dut(INTEGER XO,INTEGER YO,REAL Sc)
95 COM /Sys_state/ @Hp87xx,Scode
96 !-----
97 !
98 ! This subprogram draws a device under test (dut)
99 ! and connects it to the HP 8711 that was drawn
100 ! with an origin at XO,YO and a scale of Sc.
101 !
102 !-----
103 OUTPUT @Hp87xx;"DISP:WIND10:GRAP:MOVE "&VAL$(XO+INT(Sc*245))&"
,"&VAL$(YO+INT(Sc*15))
104 OUTPUT @Hp87xx;"DISP:WIND10:GRAP:DRAW "&VAL$(XO+INT(Sc*245))&"
,"&VAL$(YO-INT(Sc*20))
105 OUTPUT @Hp87xx;"DISP:WIND10:GRAP:DRAW "&VAL$(XO+INT(Sc*265))&"
,"&VAL$(YO-INT(Sc*20))
106 OUTPUT @Hp87xx;"DISP:WIND10:GRAP:MOVE "&VAL$(XO+INT(Sc*265))&"
,"&VAL$(YO-INT(Sc*22))
107 OUTPUT @Hp87xx;"DISP:WIND10:GRAP:RECT "&VAL$(INT(Sc*40))&","&
VAL$(INT(Sc*4))
108 OUTPUT @Hp87xx;"DISP:WIND10:GRAP:MOVE "&VAL$(XO+INT(Sc*305))&"
,"&VAL$(YO-INT(Sc*20))
109 OUTPUT @Hp87xx;"DISP:WIND10:GRAP:DRAW "&VAL$(XO+INT(Sc*325))&"
,"&VAL$(YO-INT(Sc*20))
110 OUTPUT @Hp87xx;"DISP:WIND10:GRAP:DRAW "&VAL$(XO+INT(Sc*325))&"
,"&VAL$(YO+INT(Sc*15))
111 SUBEND
112 !
113 !*****
114 ! Iden_port: Identify io port to use
115 !*****
116 SUB Iden_port
117 COM /Sys_state/ @Hp87xx,Scode
118 !
119 IF POS(SYSTEM$("SYSTEM ID"),"HP 87")<>0 THEN
120 ASSIGN @Hp87xx TO 800
121 Scode=8
122 ELSE
123 ASSIGN @Hp87xx TO 716
124 Scode=7
125 END IF

```

```
126 !  
127 SUBEND !Iden_port  
128 !
```

DUALCTRL — Two controller operation

```
10  !-----
20  !
30  ! BASIC program:  DUALCTRL - Two controller operation
40  !
50  ! This program is designed to run on an external
60  ! controller.  It demonstrates how the external
70  ! controller and HP IBASIC can work together.  The
80  ! program downloads an IBASIC program to the HP 871X
90  ! and runs it twice.  After each run, two program
100 ! variables are read from the analyzer and displayed.
110 !
120 !-----
130 !
140 ! Initialize the variables for the interface select
150 ! code and the HP-IB address of the HP 871X.
160 !
170 Scode=7
180 Address=16
190 Na=Scode*100+Address
200 !
210 ! Prepare the analyzer for remote operation, clear
220 ! the analyzer's input/output queues, the display
230 ! and scratch any program in the buffer.
240 !
250 CLEAR Na
260 CLEAR SCREEN
270 OUTPUT Na;"PROG:DEL:ALL"
280 !
290 ! Download the program as an indefinite block length
300 ! data transfer, terminate the data transfer by
310 ! sending a carriage return and EOI.
320 !
330 DISP "Downloading the program..."
340 ASSIGN @Prog TO Na
350 OUTPUT @Prog;"PROG:DEF #0";
360 OUTPUT @Prog;"10 COM INTEGER Times_run,Test$[10]"
370 OUTPUT @Prog;"20 Times_run=Times_run+1"
380 OUTPUT @Prog;"30 IF Times_run=1 THEN Test$=""PASS""
390 OUTPUT @Prog;"40 IF Times_run=2 THEN Test$=""FAIL""
400 OUTPUT @Prog;"50 FOR I= 1 TO 20"
410 OUTPUT @Prog;"60 BEEP"
420 OUTPUT @Prog;"70 NEXT I"
430 OUTPUT @Prog;"80 END"
440 OUTPUT @Prog;CHR$(10) END
450 !
460 ! Initialize interrupt registers - clear the status byte,
470 ! the service request enable register, the standard event
480 ! enable register, and preset the other status registers.
490 !
500 OUTPUT Na;"*CLS"
510 OUTPUT Na;"*SRE 0"
520 OUTPUT Na;"*ESE 0"
530 OUTPUT Na;"STAT:PRES"
```

```

540  !
550  ! Set up the status registers to generate an interrupt
560  ! on negative transition of the Program Running bit
570  ! (bit 14 in the Operational Status register).
580  !
590  OUTPUT Na;"STAT:OPER:NTR #HFFFF"
600  OUTPUT Na;"STAT:OPER:ENAB 16384"
610  OUTPUT Na;"*CLS"
620  OUTPUT Na;"*SRE 128"
630  !
640  ! Run the program, read and display the variables.
650  !
660  DISP "Running the program..."
670  OUTPUT Na;"PROG:EXEC 'RUN'"
680  Display_res(Na,Scode)
690  OUTPUT Na;"PROG:EXEC 'RUN'"
700  Display_res(Na,Scode)
710  !
720  ! Return the analyzer to front panel control, this
730  ! is the end of the program.
740  !
750  LOCAL Na
760  DISP "DONE !"
770  END
780  !
790  !-----
800  !
810  SUB Display_res(Na,Scode)
820  !-----
830  !
840  ! This subprogram waits for an SRQ interrupt to
850  ! signal that an IBASIC program running on the
860  ! analyzer has finished. It then reads and clears
870  ! the HP-IB status registers. The values of two
880  ! program variables are then read and displayed.
890  !
900  !-----
910  !
920  ! Setup branching to an interrupt handling routine,
930  ! enable the interrupts and wait until one occurs.
940  !
950  ON INTR Scode GOTO Read_results
960  ENABLE INTR Scode;2
970  Idle: GOTO Idle
980  Read_results: !
990  !
1000 ! The program has finished running - read and clear
1010 ! the operational status register and status byte.
1020 !
1030 A=SPOLL(Na)
1040 OUTPUT Na;"STAT:OPER:EVENT?"
1050 ENTER Na;Event
1060 OUTPUT Na;"*CLS"
1070 !
1080 ! Read a numeric variable (Times_run) and a string

```



```
1090    ! variable (Test$) and display the values.
1100    !
1110    OUTPUT Na;"PROG:NUMB? 'Times_run'"
1120    ENTER Na USING "X,K";Times_run
1130    OUTPUT Na;"PROG:STR? 'Test$'"
1140    ENTER Na USING "X,K";Test$
1150    DISP "Times_run: ";Times_run,"Test$: ";Test$
1160    PRINT "Times_run: ";Times_run,"Test$: ";Test$
1170    SUBEND
```

REPORT — Using the parallel port

```
10  !-----
10  !
20  !
30  ! IBASIC program:  REPORT - Using the parallel port
40  !
50  ! This program uses the 871X to generate a report,
60  ! making a hardcopy on a printer connected to the
70  ! parallel port.  It uses a subprogram to send the
80  ! output to the parallel port one line at a time.
90  !
100 ! This example uses five different font types that
110 ! may or may not be supported for your printer.
120 ! These character fonts are available for HP LaserJet
130 ! printers.  Refer to your printer manual to modify
140 ! the example fonts for your printer.
150 !
160 !-----
170 !
180 ! Assign an I/O path name for the internal bus and
190 ! declare and initialize variables.
200 !
210 COM /Cset/ Block$[50],Title$[50],Slant$[50],Banner$[50],Medium$[50]
220 ASSIGN @Rfna TO 800
230 Esc$=CHR$(27)
240 !
250 ! Preset the analyzer, put it in Trigger HOLD mode,
260 ! allocate the full IBASIC display and clear the
270 ! screen.
280 !
290 OUTPUT @Rfna;"SYST:PRES;*WAI"
300 OUTPUT @Rfna;"ABOR;:INIT:CONT OFF;*WAI"
310 OUTPUT @Rfna;"DISP:PROG FULL"
320 CLEAR SCREEN
330 !
340 ! Define the escape sequence for each font that is
350 ! used.  Refer to your printer manual.
360 !
370 Block$=Esc$&"&l00"&Esc$&"(8U"&Esc$&"(s1p10h12v0s0b0T"
380 Title$=Esc$&"&l00"&Esc$&"(8U"&Esc$&"(s1p8h12v0s0b0T"
390 Slant$=Esc$&"&l00"&Esc$&"(7J"&Esc$&"(s0p6h14v1s0b0T"
400 Banner$=Esc$&"&l00"&Esc$&"(7J"&Esc$&"(s0p4h24v0s0b0T"
410 Medium$=Esc$&"&l00"&Esc$&"(7J"&Esc$&"(s0p8h14v0s0b0T"
420 !
430 ! Select the font to use writing the company name
440 ! and address, send the company name and address.
450 !
460 CALL Send_line(Title$,1)
470 CALL Send_line("COMPANY NAME",1)
480 CALL Send_line("CITY, STATE, COUNTRY",1)
490 CALL Send_line(" ",1)
500 !
510 ! Select the font to use writing the device name,
520 ! send the device name.
```

```

530  !
540  CALL Send_line(Banner$,1)
550  CALL Send_line(" ",0)
560  CALL Send_line(" ",1)
570  CALL Send_line(" ",1)
580  CALL Send_line(" ",1)
590  CALL Send_line("  BPF-175 Bandpass Filter",1)
600  CALL Send_line(" ",1)
610  CALL Send_line(" ",1)
620  CALL Send_line(" ",0)
630  CALL Send_line(" ",1)
640  CALL Send_line(" ",1)
650  !
660  ! Select the font to use writing the device
670  ! specifications, send the information.
680  !
690  CALL Send_line(Slant$,1)
700  CALL Send_line(" ",1)
710  CALL Send_line("PASS BAND (MHZ)          3 dB          60 +/- 5",1)
720  CALL Send_line(" ",1)
730  CALL Send_line("                          20 dB          90 +/- 5",1)
740  CALL Send_line(" ",1)
750  CALL Send_line("                          40 dB          120 +/- 5",1)
760  CALL Send_line(" ",1)
770  CALL Send_line("SWR PASSBAND (typical)    1.8:1",1)
780  CALL Send_line(" ",1)
790  CALL Send_line("SWR STOPBAND (typical)   1.8:1",1)
800  CALL Send_line(" ",1)
810  CALL Send_line("Cost per unit:           36.95",1)
820  !
830  ! Select the font to use for the performance data
840  ! title, send the title.
850  !
860  CALL Send_line(Block$,1)
870  CALL Send_line("                                ",0)
880  CALL Send_line("                                Transmission Characteristics",1)
890  !
900  ! Return the display to the analyzer.
910  !
920  OUTPUT @Rfna;"DISP:PROG OFF"
930  !
940  ! Setup the device measurement.  This example
950  ! measures the transmission response of a
960  ! bandpass filter at 175 MHz.
970  !
980  OUTPUT @Rfna;"DISP:ANN:FREQ1:MODE SSTOP"
990  OUTPUT @Rfna;"SENS1:FREQ:STAR 10 MHz;STOP 400 MHz;*WAI"
1000 OUTPUT @Rfna;"DISP:WIND1:TRAC:Y:PDIV 20 dB;RLEV -50 dB;RPOS 5"
1010 OUTPUT @Rfna;"DISP:ANN:TITL ON;TITL1:DATA 'HP 8711 RF NETWORK ANALYZER'"
1020 !
1030 ! Take a measurement sweep and wait for it to
1040 ! complete.  Perform a -3 dB bandwidth search.
1050 !
1060 OUTPUT @Rfna;"INIT1;*OPC?"
1070 ENTER @Rfna;Opc

```

```

1080 OUTPUT @Rfna;"CALC1:MARK1 ON;MARK:BWID -3"
1090 !
1100 ! Select the parallel port and the printer's
1110 ! control language as the hardcopy device.
1120 ! Set the printer resolution and margins -
1130 ! turn off automatic form feed.
1140 !
1150 OUTPUT @Rfna;"HCOP:DEV:LANG PCL;PORT CENT"
1160 OUTPUT @Rfna;"HCOP:DEV:RES 300"
1170 OUTPUT @Rfna;"HCOP:PAGE:MARG:LEFT 40"
1180 OUTPUT @Rfna;"HCOP:PAGE:WIDT 110"
1190 OUTPUT @Rfna;"HCOP:ITEM1:FFE:STAT OFF"
1200 !
1210 ! Send the measurement data (graph and marker
1220 ! values) to the printer.
1230 !
1240 OUTPUT @Rfna;"HCOP"
1250 !
1260 ! Select the fonts and send the "footer"
1270 ! information for the report.
1280 !
1290 CALL Send_line(Banner$,1)
1300 CALL Send_line(" ",1)
1310 CALL Send_line(" ",1)
1320 CALL Send_line("IN STOCK IMMEDIATE DELIVERY!",1)
1330 CALL Send_line(Medium$,1)
1340 CALL Send_line(" ",0)
1350 CALL Send_line("For more information: Call 1-800-Filter",1)
1360 !
1370 ! Send a form feed to the printer.
1380 !
1390 WRITEIO 15,0;12
1400 END
1410 !-----
1420 SUB Send_line(String$,INTEGER Crlf)
1430 !-----
1440 !
1450 ! The subprogram sends a string to the parallel port
1460 ! (I/O port 15). The Crlf flag determines whether
1470 ! a carriage return (ASCII 13) and line feed (ASCII
1480 ! 10) are needed at the end of the string.
1490 !
1500 !-----
1510 INTEGER Length
1520 Length=LEN(String$)
1530 FOR I=1 TO Length
1540 WRITEIO 15,0;NUM(String$[I;1])
1550 NEXT I
1560 IF Crlf=1 THEN
1570 WRITEIO 15,0;10
1580 WRITEIO 15,0;13
1590 END IF
1600 SUBEND

```

TRICTRL — External controller with local IBASIC controllers

```
10  !-----
20  !
30  ! BASIC program:  TRICTRL - Three controller operation
40  !   One controller, Two IBASIC instruments
50  !
60  ! This program is designed to run on an external
70  ! controller.  It demonstrates how the external
80  ! controller and multiple instruments running IBASIC
90  ! programs can be synchronized to work together.
100 !
110 ! Run this program on an external controller.  Two HP871x
120 ! are needed.  Set one HP871x to address 16.  Set the
130 ! other to address 18.  Connect HP-IB cables between
140 ! the controller and the two analyzers.
150 !
160 ! The program downloads IBASIC programs to two HP 871Xs,
170 ! then runs each program.  Pressing softkey 1 on either
180 ! instrument triggers a sweep.  Pressing softkey 3 on
190 ! either instrument will trigger an SRQ.  The controller
200 ! will poll the instrument over the HP-IB bus, determine
210 ! which instrument has requested service, log the SRQ,
220 ! and release the instrument for more measurements by
230 ! setting the IBASIC variable Ctrl_flag.
240 !-----
250 !
260 ! Initialize the variables for the interface select
270 ! code and the HP-IB address of the HP 871X.
280 !
290  Scode=7
300  Address1=16
310  Address2=18
320  Na1=Scode*100+Address1
330  Na2=Scode*100+Address2
340  Dev_count1=0
350  Dev_count2=0
360  !
370  ! Prepare the analyzer for remote operation, clear
380  ! the analyzer's input/output queues and scratch
390  ! any program in the buffer.
400  !
410  ABORT 7
420  CLEAR Na1
430  CLEAR Na2
440  CLEAR SCREEN
450  !
460  OUTPUT Na1;"SYST:PRES;*OPC?"           ! Preset analyzer #1
470  ENTER Na1;Opc
480  OUTPUT Na2;"SYST:PRES;*OPC?"           ! Preset analyzer #2
490  ENTER Na2;Opc
500  !
510  OUTPUT Na1;"PROG:STAT STOP"           ! Stop all programs
520  REMOTE Na1
530  OUTPUT Na1;"PROG:DEL:ALL"             ! Scratch the programs
```

```

540  OUTPUT Na2;"PROG:STAT STOP"
550  REMOTE Na2
560  OUTPUT Na2;"PROG:DEL:ALL"
570  !
580  ! Initialize interrupt registers - clear the status byte,
590  ! the service request enable register, the standard event
600  ! enable register, and preset the other status registers.
610  !
620  OUTPUT Na1;"*CLS"
630  OUTPUT Na1;"*SRE 0"
640  OUTPUT Na1;"*ESE 0"
650  OUTPUT Na1;"STAT:PRES;*OPC?"
660  ENTER Na1;Opc
670  !
680  OUTPUT Na2;"*CLS"
690  OUTPUT Na2;"*SRE 0"
700  OUTPUT Na2;"*ESE 0"
710  OUTPUT Na2;"STAT:PRES;*OPC?"
720  ENTER Na2;Opc
730  !
740  ON INTR 7,2 GOSUB User_srq      ! Define the SRQ service routine
750  GOSUB Usermask                ! Enable the user SRQ
760  ENABLE INTR 7;2
770  !
780  ! Download the program as an indefinite block length
790  ! data transfer, terminate the data transfer by
800  ! sending a carriage return and EOI.
810  !
820  DISP "Downloading the programs..."
830  ASSIGN @Prog TO Na1
840  GOSUB Dnld
850  ASSIGN @Prog TO Na2
860  GOSUB Dnld
870  !
880  ! Run the programs
890  DISP "Running the programs..."
900  OUTPUT Na1;"PROG:STAT RUN;*OPC?"
910  ENTER Na1;Opc
920  !
930  OUTPUT Na2;"PROG:STAT RUN;*OPC?"
940  ENTER Na2;Opc
950  !
960  BEEP
970  DISP "Waiting for srq..."
980  !
990  LOCAL Na1
1000 LOCAL Na2
1010 Idle:GOTO Idle
1020 STOP
1030 ! Enable SRQs to occur when the user_srq bit is set
1040 Usermask:      !
1050 OUTPUT Na1;"*ESE 64;*SRE 32"
1060 OUTPUT Na1;"*OPC?"
1070 ENTER Na1;Opc
1080 OUTPUT Na2;"*ESE 64;*SRE 32"

```

```

1090 OUTPUT Na2;"*0PC?"
1100 ENTER Na2;Opc
1110 RETURN
1120 !
1130 User_srq:      ! This routine is called to service SRQs
1140 Stb=SPOLL(Na1)      ! Poll the first instrument
1150 IF (BINAND(Stb,64)<>0) THEN
1160   OUTPUT Na1;"*ESR?"
1170   ENTER Na1;Stat
1180   Dev_count1=Dev_count1+1
1190   PRINT "Inst:",Na1,"Dev:",Dev_count1
1200   OUTPUT Na1;"PROG:NUMB 'Ctrl_flag',0"  ! Clear the IBASIC flag
1210   LOCAL Na1
1220 ELSE
1230 !
1240   Stb=SPOLL(Na2)      ! Poll the second instrument
1250   IF (BINAND(Stb,64)<>0) THEN
1260     OUTPUT Na2;"*ESR?"
1270     ENTER Na2;Stat
1280     Dev_count2=Dev_count2+1
1290     PRINT "Inst:",Na2,"Dev:",Dev_count2
1300     OUTPUT Na2;"PROG:NUMB 'Ctrl_flag',0"  ! Clear the IBASIC flag
1310     LOCAL Na2
1320   END IF
1330 END IF
1340 !
1350 ENABLE INTR 7
1360 RETURN
1370 !
1380 Dnld: ! Download example program to analyzer
1390 OUTPUT @Prog;"PROG:DEF #0";
1400 OUTPUT @Prog;"10 COM INTEGER Ctrl_flag"
1410 OUTPUT @Prog;"20 OUTPUT 800;""ABOR;:INIT1:CONT OFF""
1420 OUTPUT @Prog;"30 ON KEY 1 LABEL ""Test 1"" GOSUB Do_test"
1430 OUTPUT @Prog;"40 ON KEY 3 LABEL ""Done Test"" GOSUB Send_srq"
1440 OUTPUT @Prog;"50 Idle:GOTO Idle"
1450 OUTPUT @Prog;"60 STOP"
1460 OUTPUT @Prog;"70 Send_srq: !"
1470 OUTPUT @Prog;"80 BEEP"
1480 OUTPUT @Prog;"90 Ctrl_flag=1"
1490 OUTPUT @Prog;"100 OUTPUT 800;""SYST:KEY:USER""
1500 OUTPUT @Prog;"110 DISP ""Waiting for CTRL...""
1510 OUTPUT @Prog;"120 Stall: IF Ctrl_flag=1 THEN GOTO Stall"
1520 OUTPUT @Prog;"130 DISP "" ""
1530 OUTPUT @Prog;"140 RETURN"
1540 OUTPUT @Prog;"150 DO_TEST: OUTPUT 800;""INIT1;*0PC?""
1550 OUTPUT @Prog;"160 ENTER 800;Opc"
1560 OUTPUT @Prog;"170 RETURN"
1570 OUTPUT @Prog;"180 END"
1580 OUTPUT @Prog;CHR$(10) END
1590 OUTPUT @Prog;"*opc?"
1600 ENTER @Prog;Opc
1610 RETURN
1620 END
1630 !

```

UPLOAD — Upload program from analyzer

```
10  !-----
20  !
30  ! BASIC program: UPLOAD - Upload program from HP 871X
40  !
50  ! This program uploads the current IBASIC program
60  ! in the HP 871X's program buffer to an ASCII file
70  ! on the controller's current mass storage device.
80  !
90  !-----
100 !
110 ! Assign an I/O path name to the HP 8711, initialize
120 ! the variables, and clear the analyzer's input/output
130 ! queues.
140 !
150 ASSIGN @Rfna TO 716
160 DIM Prog_line$[256]
170 CLEAR @Rfna
180 !
190 ! Enter the name of the file to be created.
200 !
210 INPUT "ENTER NAME OF FILE TO UPLOAD PROGRAM TO ",Filename$
220 PRINT Filename$
230 !
240 ! Query the HP 8711 for the contents of its
250 ! program buffer.
260 !
270 OUTPUT @Rfna;"PROG:DEF?"
280 !
290 ! Read the block header, the number of digits in
300 ! the file size, and the file size.
310 !
320 ENTER @Rfna USING "#,A,D";Prog_line$,Ndigits
330 ENTER @Rfna USING "#,&VAL$(Ndigits)&"D";Nbytes
340 !
350 ! Create the target ASCII file on the current mass
360 ! storage device and assign it an I/O path name.
370 !
380 Openfile(@File,Filename$,Nbytes)
390 ASSIGN @File TO Filename$;FORMAT ON
400 !
410 ! Read the program one line at a time, and write
420 ! it to the new file. Print each line on the
430 ! display as it is read.
440 !
450 LOOP
460     ENTER @Rfna;Prog_line$
470     EXIT IF LEN(Prog_line$)=0
480     PRINT Prog_line$
490     OUTPUT @File;Prog_line$
500 END LOOP
510 !
520 ! Close the new file.
530 !
```



```

540  ASSIGN @File TO *
550  END
560  !-----
570  SUB Openfile(@File,Filename$,Fsize)
580  !-----
590  !
600  ! This subprogram creates an ASCII file with the
610  ! name 'Filename$' of the specified size 'Fsize'.
620  ! Error trapping is used to detect any errors in
630  ! opening the file.  If the controller is HP IBASIC
640  ! for Windows a DOS file is created, otherwise the
650  ! LIF format is used.
660  !
670  !-----
680      ON ERROR GOTO Openerr
690      IF SYSTEM$("SYSTEM ID")="IBASIC/WINDOWS" THEN
700          CREATE Filename$,1
710      ELSE
720          IF Fsize MOD 256>0 THEN Fsize=Fsize+256
730          CREATE ASCII Filename$,Fsize DIV 256
740      END IF
750  !
760  Openerr:IF ERRN<>54 THEN PRINT ERRM$
770  SUBEND

```

USERBEG — Set up user-defined User BEGIN softkeys

```
10 !Filename:  USERBEG
20 !
70  !
80  ! Description:  Program to set up the User Begin softkeys.
90  !
100 !  A) This program creates User Begin softkeys which allow
110 !      the user to:  Save or Recall one of two instrument
120 !      states, set the marker to maximum, set the scale/div,
130 !      and compute some measurement statistics at the marker.
140 !
150 !  B) In order to run this program, do the following -
160 !      1) Load this program into the 871x
170 !      2) Press the "BEGIN" (hardkey) and the "User Begin on/OFF" (softkey).
180 !      3) The "User Begin" function is now enabled (which runs this
190 !      program).  This program re-defines the softkeys displayed
200 !      whenever the BEGIN hardkey is pressed.  The functions
210 !      performed by these softkeys are defined by this
220 !      program.  Note that all front panel keys in the analyzer are
230 !      active (as if there were no program running).
240 !      4) Use the instrument as you normally would.  However, when
250 !      the BEGIN hardkey is pressed, the menu defined
260 !      by this program will be displayed instead of the usual
270 !      BEGIN softkeys, until the "User Begin ON/off" (softkey)
280 !      is pressed, turning off the "User Begin" mode.
290 !
300 !*****
310 ! Initialize
320 !
330 User_begin:ASSIGN @Hp871x TO 800      !REQUIRED - first line for
User Begin program
340 !
350 REAL Vert_scale,Mrkr_data(1:30),Mrkr_mean,Mrkr_sdev
360 REAL Mrkr_max,Mrkr_min,I
370 DIM Message$[124]
380 !
390 !-----
400 ! Write the softkey labels.  Maximum label length=20characters
410 !
420 OUTPUT @Hp871x;"DISP:MENU2:KEY8  '';*WAI"      !clear all labels
430 OUTPUT @Hp871x;"DISP:MENU2:KEY1  '   Save   State 1';*WAI"
440 OUTPUT @Hp871x;"DISP:MENU2:KEY2  '   Recall  State 1';*WAI"
450 OUTPUT @Hp871x;"DISP:MENU2:KEY3  '   Save   State 2';*WAI"
460 OUTPUT @Hp871x;"DISP:MENU2:KEY4  '   Recall  State 2';*WAI"
470 OUTPUT @Hp871x;"DISP:MENU2:KEY5  'Mkr -> Max';*WAI"
480 OUTPUT @Hp871x;"DISP:MENU2:KEY6  'Scale/Div';*WAI"
490 OUTPUT @Hp871x;"DISP:MENU2:KEY7  '  MarkerStatistics';*WAI"
500 !
510 User_pause:PAUSE      !pause the program until a softkey is pressed
520 GOTO User_pause      !return to program pause after a softkey press
530 !
540 !-----
550 ! Define softkey routines
```

```

560  !
570 User_key1:      ! Define softkey 1:  save state 1
580  OUTPUT @Hp871x;"MMEM:STOR:STAT 1,'MEM:UBEGN1.STA'" !save state 1
590  GOTO User_pause !return to softkey loop
600  !
610 User_key2:      ! Define softkey 2:  recall state 1
620  OUTPUT @Hp871x;"MMEM:LOAD:STAT 1,'MEM:UBEGN1.STA'" !recall state 1
630  GOTO User_pause !return to softkey loop
640  !
650 User_key3:      ! Define softkey 3:  save state 2
660  OUTPUT @Hp871x;"MMEM:STOR:STAT 1,'MEM:UBEGN2.STA'" !save state 2
670  GOTO User_pause !return to softkey loop
680  !
690 User_key4:      ! Define softkey 4:  recall state 2
700  OUTPUT @Hp871x;"MMEM:LOAD:STAT 1,'MEM:UBEGN2.STA'" !recall state 2
710  GOTO User_pause !return to softkey loop
720  !
730 User_key5:      ! Define softkey 5:  set marker to max
740  OUTPUT @Hp871x;"CALC1:MARK:FUNC MAX"      !marker -> max
750  GOTO User_pause
760  !
770 User_key6:      ! Define softkey 6:  adjust the scale, dB/Div,
of the trace
780  INPUT "Enter the scale (dB/Div)",Vert_scale !ask user for scale
790  OUTPUT @Hp871x;"DISP:WIND1:TRAC:Y:PDIV "&VAL$(Vert_scale)
!set the scale
800  GOTO User_pause
810  !
820 User_key7:      ! Define softkey 7:  compute statistics for marker.
830  OUTPUT @Hp871x;"DISP:ANN:MESS:DATA 'Computing marker statistics...'"
840  OUTPUT @Hp871x;"CALC1:MARK1 ON"          !ensure marker is on
850  FOR I=1 TO 30
!read marker 30 times
860  OUTPUT @Hp871x;"CALC1:MARK1:Y?"        !get marker reading
870  ENTER @Hp871x;Mrkr_data(I)
880  NEXT I
890  Mrkr_mean=SUM(Mrkr_data)/30              !compute mean
900  !
910  Mrkr_sdev=0                               !initialize standard deviation
920  Mrkr_min=Mrkr_data(1)                    !initialize min
930  Mrkr_max=Mrkr_data(1)                    !initialize max
940  FOR I=1 TO 30
!compute std dev, min, max
950  Mrkr_sdev=Mrkr_sdev+(Mrkr_data(I)-Mrkr_mean)^2 !sum squares of deviation
960  Mrkr_min=MIN(Mrkr_min,Mrkr_data(I))      !find min
970  Mrkr_max=MAX(Mrkr_max,Mrkr_data(I))      !find max
980  NEXT I
990  Mrkr_sdev=SQRT(Mrkr_sdev/29)             !finish computation of std dev
1000 !
1010 Message$="Marker Statistics:"&CHR$(10)    !1st line of message
1020 Message$=Message$&" Mean ="&VAL$(Mrkr_mean)&CHR$(10) !2nd line of message
1030 Message$=Message$&" Min ="&VAL$(Mrkr_min)&CHR$(10)   !3rd line of message
1040 Message$=Message$&" Max ="&VAL$(Mrkr_max)&CHR$(10)   !4th line of message
1050 Message$=Message$&" Standard Deviation ="&VAL$(Mrkr_sdev)
!5th line of message
1060 OUTPUT @Hp871x;"DISP:ANN:MESS:DATA '&Message$&'", MEDIUM" !display message
1070 GOTO User_pause                          !return to softkey loop

```

```
1080 !  
1090 END
```

USERBEG1 — The default User BEGIN program

```
10      ! -----
20      !
30      ! BASIC program: USERBEG1
40      !
50      ! This is the default User Defined BEGIN program. This program
60      ! will automatically install if the [User BEGIN] key is
70      ! selected, and a program has not been previously loaded.
80      !
90      ! The following line is required. DO NOT REMOVE!
100     User_begin:ASSIGN @Rfna TO 800      ![User Begin] Program
110     !
120     ! To Modify:
130     ! Use [IBASIC][EDIT] or [IBASIC][Key Record]
140     !
150     !
160     ! Delclare storage for variables.
170     DIM Name$[60],Str1$[60],Str2$[60],Str3$[60]
180     !
190     ! Clear the softkey labels
200     OUTPUT @Rfna;"DISP:MENU2:KEY8  ' ';*WAI"
210     !
220     ! Re-define softkey labels here.
230     OUTPUT @Rfna;"DISP:MENU2:KEY1  '*';*WAI"
240     OUTPUT @Rfna;"DISP:MENU2:KEY2  '*';*WAI"
250     OUTPUT @Rfna;"DISP:MENU2:KEY3  'Mkr -> Max';*WAI"
260     OUTPUT @Rfna;"DISP:MENU2:KEY4  'Title and Clock';*WAI"
270     OUTPUT @Rfna;"DISP:MENU2:KEY5  '*';*WAI"
280     OUTPUT @Rfna;"DISP:MENU2:KEY6  '*';*WAI"
290     OUTPUT @Rfna;"DISP:MENU2:KEY7  '*';*WAI"
300     !
310     !The following 2 lines are required. DO NOT REMOVE!
320     User_pause:PAUSE
330     GOTO User_pause
340     !
350     User_key1:      ! Define softkey 1 here.
360     GOSUB Message ! Remove this line.
370     GOTO User_pause
380     !
390     User_key2:      ! Define softkey 2 here.
400     GOSUB Message ! Remove this line
410     GOTO User_pause
420     !
430     User_key3:      ! Example Marker Function
440     OUTPUT @Rfna;"CALC1:MARK1 ON"
450     OUTPUT @Rfna;"CALC1:MARK:FUNC MAX"
460     GOTO User_pause
470     !
480     User_key4:      ! Example Title Entry
490     INPUT "Enter Title Line 1. Press [Enter] when done.",Name$
500     OUTPUT @Rfna;"DISP:ANN:TITL1:DATA  '&Name$&''"
510     OUTPUT @Rfna;"DISP:ANN:TITL ON"
520     GOTO User_pause
```

```

530  !
540 User_key5:      ! Define softkey 5 here.
550  GOSUB Message ! Remove this line.
560  GOTO User_pause
570  !
580 User_key6:      ! Define softkey 6 here.
590  GOSUB Message ! Remove this line.
600  GOTO User_pause
610  !
620 User_key7:      ! Define softkey 7 here.
630  GOSUB Message ! Remove this line.
640  GOTO User_pause
650  !
660 Message:      !
670  Str1$="This key is programmable."
680  Str2$="To modify, select"
690  Str3$="[System Options], [IBASIC], [Edit]."
700  OUTPUT @Rfna;"DISP:ANN:MESS '&Str1$&CHR$(10)&Str2$&CHR$(10)&Str3$&'
, MEDIUM"
710  RETURN
720  !
730  END

```

USERBEG2 — Fast recall of instrument states

```
10      ! -----
20      !
30      ! BASIC program: USERBEG2
40      !
50      ! This is an example User Defined BEGIN program.  This program
60      ! will recall the named file.  Demonstrates fast recall
70      ! of a previously defined files SETUP1, SETUP2, and SETUP3.
80      !
90      ! The following line is required. DO NOT REMOVE!
100     User_begin:ASSIGN @Rfna TO 800      ![User Begin] Program
110     !
120     ! To Modify:
130     ! Use [IBASIC][EDIT] or [IBASIC][Key Record]
140     !
150     !
160     ! Delclare storage for variables.
170     DIM Name$[60],Str1$[60],Str2$[60],Str3$[60]
180     !
190     ! Clear the softkey labels
200     OUTPUT @Rfna;"DISP:MENU2:KEY8 '';*WAI"
210     !
220     ! Re-define softkey labels here.
230     OUTPUT @Rfna;"DISP:MENU2:KEY1 'Setup1';*WAI"
240     OUTPUT @Rfna;"DISP:MENU2:KEY2 'Setup2';*WAI"
250     OUTPUT @Rfna;"DISP:MENU2:KEY3 'Setup3';*WAI"
260     OUTPUT @Rfna;"DISP:MENU2:KEY4 '*';*WAI"
270     OUTPUT @Rfna;"DISP:MENU2:KEY5 '*';*WAI"
280     OUTPUT @Rfna;"DISP:MENU2:KEY6 '*';*WAI"
290     OUTPUT @Rfna;"DISP:MENU2:KEY7 '*';*WAI"
300     !
310     !The following 2 lines are required. DO NOT REMOVE!
320     User_pause:PAUSE
330     GOTO User_pause
340     !
350     User_key1:      ! Define softkey 1 here.
360     OUTPUT @Rfna;"MMEM:LOAD:STAT 1,'MEM:SETUP1'"
370     GOTO User_pause
380     !
390     User_key2:      ! Define softkey 2 here.
400     OUTPUT @Rfna;"MMEM:LOAD:STAT 1,'MEM:SETUP2'"
410     GOTO User_pause
420     !
430     User_key3:      ! Example Marker Function
440     OUTPUT @Rfna;"MMEM:LOAD:STAT 1,'MEM:SETUP3'"
450     GOTO User_pause
460     !
470     User_key4:      ! Example Title Entry
480     GOTO User_pause
490     !
500     User_key5:      ! Define softkey 5 here.
510     GOTO User_pause
520     !
530     User_key6:      ! Define softkey 6 here.
```

```
540 GOTO User_pause
550 !
560 User_key7:      ! Define softkey 7 here.
570 GOTO User_pause
580 !
590 END
```


USER_BIT — Using the USER bit

```
10  !-----
20  !
30  ! IBASIC program:  USER_BIT - Using the USER bit
40  !
50  ! This program reads and writes to the USER bit.
60  ! IBASIC's graphics commands are used to draw the
70  ! USER bit value to the display.
80  !
90  !-----
100 !
110 ! Assign an I/O path name to the internal bus and
120 ! initialize variables.
130 !
140 ASSIGN @Rfna TO 800
150 INTEGER Beeper,Count
160 Count=0
170 Beeper=0
180 !
190 ! Preset the analyzer, setup measurement and display
200 ! parameters for a measurement and put the analyzer
210 ! in Trigger HOLD mode.
220 !
230 OUTPUT @Rfna;"SYST:PRES;*WAI"
240 OUTPUT @Rfna;"DISP:ANN:FREQ1:MODE SSTOP"
250 OUTPUT @Rfna;"SENS1:FREQ:STAR 100 MHz;STOP 400 MHz;*WAI"
260 OUTPUT @Rfna;"DISP:WIND1:TRAC:Y:PDIV 20 dB;RLEV -60 dB;RPOS 5"
270 OUTPUT @Rfna;"SENS1:SWE:POIN 101;TIME .1 s;*WAI"
280 OUTPUT @Rfna;"ABOR;:INIT1:CONT OFF;*WAI"
290 !
300 ! Wait for all the setup operations to be complete
310 ! before continuing the program.
320 !
330 OUTPUT @Rfna;"*0PC?"
340 ENTER @Rfna;Opc
350 !
360 ! Allocate the lower display partition.
370 !
380 OUTPUT @Rfna;"DISP:PROG LOW"
390 !
400 ! Setup a softkey menu to enable and disable the
410 ! beeper.  Clear the analyzer's input/output queues.
420 !
430 ON KEY 1 LABEL "Beep      Enable" GOSUB Beep_on
440 ON KEY 2 LABEL "Beep      Disable" GOSUB Beep_off
450 CLEAR @Rfna
460 !
470 ! Trigger 100 sweeps.  Beep (if the beeper flag is set)
480 ! and toggle the USER bit after each sweep.
490 !
500 DISP "USER bit example program. End of sweep toggles USER bit."
510 PRINT "Draw the end of sweep USER bit value..."
520 MOVE 0,20
530 FOR I=1 TO 100
```

```

540     OUTPUT @Rfna;"INIT1;*OPC?"
550     ENTER @Rfna;0pc
560     GOSUB Toggle
570     NEXT I
580     DISP "End program"
590     STOP
600     !
610     ! The Odd flag's value alternates between 1 and 0
620     ! depending on the number of sweeps that have been
630     ! taken. It is the value that is written to the
640     ! USER bit.
650     !
660 Toggle: !
670     IF Odd=0 THEN
680         WRITEIO 15,1;0
690         Odd=1
700     ELSE
710         WRITEIO 15,1;1
720         Odd=0
730     END IF
740     IF Beeper=1 THEN
750         BEEP
760     END IF
770     !
780     ! Read the value of the USER bit and draw it to the
790     ! IBASIC display.
800     !
810     Val=READIO(15,1)
820     Val=Val*12
830     DRAW 6*(I-1),Val+20
840     DRAW 6*I,Val+20
850     RETURN
860     !
870     ! These two subroutines set a flag that is used
880     ! to turn on or off the beeper.
890     !
900 Beep_on: Beeper=1
910     RETURN
920     !
930 Beep_off: Beeper=0
940     RETURN
950     END

```

USERKEYS — Customized softkeys

```
10  !-----
20  !
30  ! IBASIC program:  USERKEYS - Customized softkeys
40  !
50  ! This program provides an example template for use
60  ! in customizing the HP 871X's softkeys.  The example
70  ! demonstrates how to set up six instrument states,
80  ! store them to the analyzer's internal memory, and
90  ! setup two interactive softkey menus to choose
100 ! between them.
110 !
120 !-----
130 !
140 ! Assign an I/O path name to the internal bus, preset
150 ! the analyzer, wait until the preset is complete,
160 ! turn on Trigger HOLD mode and set the display scale
170 ! and reference values.
180 !
190 ASSIGN @Rfna TO 800
200 OUTPUT @Rfna;"SYST:PRES;*OPC?"
210 ENTER @Rfna;Opc
220 OUTPUT @Rfna;"ABOR;:INIT1:CONT OFF;*WAI"
230 OUTPUT @Rfna;"DISP:WIND1:TRAC:Y:PDIV 20 dB;RLEV -60 dB;RPOS 5"
240 !
250 ! Setup six instrument states and store them to the
260 ! internal memory.
270 !
280 GOSUB Save_1
290 GOSUB Save_2
300 GOSUB Save_3
310 GOSUB Save_4
320 GOSUB Save_5
330 GOSUB Save_6
340 !
350 ! Setup the Main Menu keys.
360 !
370 GOSUB Menu_1
380 !
390 ! Wait until a softkey is pressed.
400 !
410 Suspend: !
420 WAIT 100000
430 GOTO Suspend
440 STOP
450 !
460 ! This subroutine sets up the softkey menus -
470 ! Menu1 sets up the main menu, Menu2 sets up
480 ! the second level menu.
490 !
500 Menu_1: BEEP
510 DISP "MAIN MENU"
520 ON KEY 1 LABEL "Setup #1" GOSUB Load_1
530 ON KEY 2 LABEL "Setup #2" GOSUB Load_2
```

```

540  ON KEY 3 LABEL "Setup #3" GOSUB Load_3
550  ON KEY 5 LABEL "Autoscale" GOSUB Autoscale
560  ON KEY 6 LABEL " Next Menu" GOSUB Menu_2
570  RETURN
580  !
590 Menu_2: BEEP
600  DISP "MORE MENU"
610  ON KEY 1 LABEL "Setup #4" GOSUB Load_4
620  ON KEY 2 LABEL "Setup #5" GOSUB Load_5
630  ON KEY 3 LABEL "Setup #6" GOSUB Load_6
640  ON KEY 5 LABEL "Autoscale" GOSUB Autoscale
650  ON KEY 6 LABEL "Prior Menu" GOSUB Menu_1
660  RETURN
670  !
680  ! This subroutine automatically sets the scale and
690  ! reference values of the display.
700  !
710 Autoscale: OUTPUT @Rfna;"DISP:WIND1:TRAC:Y:AUTO ONCE"
720  OUTPUT @Rfna;"DISP:WIND2:TRAC:Y:AUTO ONCE"
730  RETURN
740  !
750  ! These six subroutines each set up the analyzer to
760  ! make a different measurement and store that setup
770  ! to the instrument's internal memory.
780  !
790 Save_1: OUTPUT @Rfna;"SENS1:STAT ON;*WAI"
800  OUTPUT @Rfna;"DISP:ANN:FREQ1:MODE SSTOP"
810  OUTPUT @Rfna;"SENS1:FREQ:STAR 100 MHz;STOP 400 MHz;*WAI"
820  OUTPUT @Rfna;"INIT1;*WAI"
830  OUTPUT @Rfna;"MMEM:STOR:STAT 1,'MEM:STATE1.STA'"
840  RETURN
850  !
860 Save_2: OUTPUT @Rfna;"SENS2:STAT ON;*WAI"
870  OUTPUT @Rfna;"SENS2:FUNC 'XFR:POW:RAT 1,0';DET NBAN;*WAI"
880  OUTPUT @Rfna;"DISP:ANN:FREQ2:MODE CSPAN"
890  OUTPUT @Rfna;"SENS2:FREQ:CENT 200 MHz;SPAN 300 MHz;*WAI"
900  OUTPUT @Rfna;"INIT2;*WAI"
910  OUTPUT @Rfna;"MMEM:STOR:STAT 1,'MEM:STATE2.STA'"
920  RETURN
930  !
940 Save_3: OUTPUT @Rfna;"CALC2:FORM SWR"
950  OUTPUT @Rfna;"INIT2;*WAI"
960  OUTPUT @Rfna;"MMEM:STOR:STAT 1,'MEM:STATE3.STA'"
970  OUTPUT @Rfna;"CALC2:FORM MLOG"
980  RETURN
990  !
1000 Save_4:OUTPUT @Rfna;"SENS2:STAT OFF"
1010 OUTPUT @Rfna;"SENS1:SWE:POIN 1601;*WAI"
1020 OUTPUT @Rfna;"INIT1;*WAI"
1030 OUTPUT @Rfna;"MMEM:STOR:STAT 1,'MEM:STATE4.STA'"
1040 RETURN
1050 !
1060 Save_5:OUTPUT @Rfna;"CALC1:MARK:BWID -3;FUNC:TRAC ON"
1070 OUTPUT @Rfna;"INIT1;*WAI"
1080 OUTPUT @Rfna;"MMEM:STOR:STAT 1,'MEM:STATE5.STA'"

```

```

1090 RETURN
1100 !
1110 Save_6:OUTPUT @Rfna;"SENS1:BWID 250 Hz;*WAI"
1120 OUTPUT @Rfna;"SENS1:SWE:POIN 101;*WAI"
1130 OUTPUT @Rfna;"INIT1;*WAI"
1140 OUTPUT @Rfna;"MMEM:STOR:STAT 1,'MEM:STATE6.STA'"
1150 RETURN
1160 !
1170 ! These six subroutines each recall one of the
1180 ! measurement setups that were stored earlier.
1190 !
1200 Load_1:DISP "Setup 1"
1210 OUTPUT @Rfna;"MMEM:LOAD:STAT 1,'MEM:STATE1.STA';*WAI"
1220 OUTPUT @Rfna;"INIT1;*WAI"
1230 RETURN
1240 !
1250 Load_2:DISP "Setup 2"
1260 OUTPUT @Rfna;"MMEM:LOAD:STAT 1,'MEM:STATE2.STA';*WAI"
1270 OUTPUT @Rfna;"INIT2;*WAI"
1280 RETURN
1290 !
1300 Load_3:DISP "Setup 3"
1310 OUTPUT @Rfna;"MMEM:LOAD:STAT 1,'MEM:STATE3.STA';*WAI"
1320 OUTPUT @Rfna;"INIT2;*WAI"
1330 RETURN
1340 !
1350 Load_4:DISP "Setup 4"
1360 OUTPUT @Rfna;"MMEM:LOAD:STAT 1,'MEM:STATE4.STA';*WAI"
1370 OUTPUT @Rfna;"INIT1;*WAI"
1380 RETURN
1390 !
1400 Load_5:DISP "Setup 5"
1410 OUTPUT @Rfna;"MMEM:LOAD:STAT 1,'MEM:STATE5.STA';*WAI"
1420 OUTPUT @Rfna;"INIT1;*WAI"
1430 RETURN
1440 !
1450 Load_6:DISP "Setup 6"
1460 OUTPUT @Rfna;"MMEM:LOAD:STAT 1,'MEM:STATE6.STA';*WAI"
1470 OUTPUT @Rfna;"INIT1;*WAI"
1480 RETURN
1490 END

```

USR_FLOC — Fault location measurements

```
10  ! -----
20  !
30  ! BASIC program: USR_FLOC
40  !
50  ! Fault Location measurements require option 100.
60  ! User BEGIN requires option 1C2, IBASIC.
70  !
80  !
90  ! This is an example user BEGIN program for fault location.
100 !
110 ! Load this program into the analyzer. Then press [BEGIN]
120 ! [User BEGIN ON].
130 !
140 ! The following line is required. DO NOT REMOVE!
150 User_begin:ASSIGN @Rfna TO 800      ![User Begin] Program
160   ASSIGN @Hp8712 TO 800
170 !
180 ! To Modify:
190 ! Use [IBASIC][EDIT] or [IBASIC][Key Record]
200 !
210 !
220 ! Delclare storage for variables.
230   DIM Name$[60],Str1$[60],Str2$[60],Str3$[60]
240 !
250 ! Clear the softkey labels
260   OUTPUT @Rfna;"DISP:MENU2:KEY8 '';*WAI"
270 !
280 ! Re-define softkey labels here.
290   OUTPUT @Rfna;"DISP:MENU2:KEY1 'Test End of Cable';*WAI"
300   OUTPUT @Rfna;"DISP:MENU2:KEY2 '*';*WAI"
310   OUTPUT @Rfna;"DISP:MENU2:KEY3 'Mkr -> Max';*WAI"
320   OUTPUT @Rfna;"DISP:MENU2:KEY4 'Next Peak Left';*WAI"
330   OUTPUT @Rfna;"DISP:MENU2:KEY5 'Next Peak Right';*WAI"
340   OUTPUT @Rfna;"DISP:MENU2:KEY6 'Zoom on Marker';*WAI"
350   OUTPUT @Rfna;"DISP:MENU2:KEY7 '*';*WAI"
360 !
370 !The following 2 lines are required. DO NOT REMOVE!
380 User_pause:PAUSE
390   GOTO User_pause
400 !
410 User_key1:      ! Example Set Stop Distance to 1100ft
420   OUTPUT @Hp8712;"SENS1:STAT ON; *WAI"
430   OUTPUT @Hp8712;"SENS1:FUNC 'FLOC 1,0';DET NBAN; *WAI"
440   OUTPUT @Hp8712;"SENS1:DIST:STOP 1100; *opc?"
450   ENTER @Hp8712;0pc
460   OUTPUT @Hp8712;"SENS1:CORR:RVEL:COAX 0.89"
470   OUTPUT @Hp8712;"DISP:WIND1:TRAC:Y:AUTO ONCE"
480   GOTO User_pause
490 !
500 User_key2:      ! Define softkey 2 here.
510   GOSUB Message ! Remove this line
520   GOTO User_pause
```

```

530 !
540 User_key3:          ! Example Marker Function
550   OUTPUT @Rfna;"CALC1:MARK1 ON"
560   OUTPUT @Rfna;"CALC1:MARK:FUNC MAX"
570   GOTO User_pause
580 !
590 User_key4:          ! Define softkey 6 here.
600   OUTPUT @Rfna;"CALC1:MARK1 ON"
610   OUTPUT @Hp8712;"CALC1:MARK:MAX:LEFT"
620   GOTO User_pause
630 !
640 User_key5:          ! Define softkey 5 here.
650   OUTPUT @Hp8712;"CALC1:MARK1 ON"
660   OUTPUT @Hp8712;"CALC1:MARK:MAX:RIGHT"
670   GOTO User_pause
680 !
690 User_key6:          ! Zoom on Cable
700   OUTPUT @Hp8712;"SENS1:STAT ON; *WAI"
710   OUTPUT @Hp8712;"SENS1:FUNC 'FLOC 1,0';DET NBAN; *WAI"
720   OUTPUT @Hp8712;"calc1:mark1:x?"
730   ENTER @Hp8712;Distance
740   New_start=Distance-20
750   IF (New_start<0) THEN New_start=0
760   OUTPUT @Hp8712;"sens1:dist:start "&VAL$(New_start)
770   OUTPUT @Hp8712;"sens1:dist:stop "&VAL$(Distance+20)
780   OUTPUT @Hp8712;"*opc?"
790   ENTER @Hp8712;0pc
800   GOTO User_pause
810 !
820 User_key7:          ! Define softkey 7 here.
830   GOSUB Message ! Remove this line.
840   GOTO User_pause
850 !
860 Message:           !
870   Str1$="This key is programmable."
880   Str2$="To modify, select"
890   Str3$="[System Options], [IBASIC], [Edit]."
900   OUTPUT @Rfna;"DISP:ANN:MESS '"&Str1$&CHR$(10)&Str2$&CHR$(10)&Str3$&"'
, MEDIUM"
910   RETURN
920 !
930   END

```

BARCODE — Using Bar Code Reader

```
10  !-----
20  !
30  ! IBASIC program:  BARCODE - Using barcode reader
40  ! For 871xC and 8730A only!
50  !
60  ! This IBASIC program was written for a barcode
70  ! reader, but it is not required.  Sets the 871x's
80  ! state depending on model # of DUT being measured.
90  ! Expects to see BARCODE with the following format:
100 ! Model Number (6char), space, Serial Number (5char)
110 ! Valid Models:  BPF175, BPF200, SAW134
120 ! REV C.03.00    961009.JVV
130 !
140 ! -----
150 !
160 COM /Hpip/ @Rfna
170 COM /Scale/ Sc,INTEGER X,Y
180 DIM Name$[50],Stat$[50],Scan$[90],Lim$(1:3,1:5)[30],Test$(0:1)[4]
190 INTEGER Tab,Fail_flg,G(1:4)
200 !
210 Init: !
220   Test$(0)="PASS"
230   Test$(1)="FAIL"
240   ASSIGN @Rfna TO 800
250   Sc=1      ! Scales the 8711 drawing and DUT
260   X=3      ! Starting X posn of 8711 plot
270   Y=44     !      "      Y      "
280   Tab=38   ! Tab position for text
290   OUTPUT @Rfna;"DIAG:PRES:SERV;*OPC?" !Presets to 871xC state
300   ENTER @Rfna;Opc
310   OUTPUT @Rfna;"DISP:PROG UPP"
320   GINIT
330   GCLEAR
340   GESCAPE 1,3;G(*)
350   WINDOW G(1),G(3),G(2),G(4)
360   OUTPUT @Rfna;"SENS1:STAT OFF;:SENS2:STAT ON"
370   OUTPUT @Rfna;"DISP:WIND2:TRAC:Y:RPOS 9"
380   OUTPUT @Rfna;"ABOR;:INIT:CONT OFF"
390 Setup: !
400   BEEP 500,.1
410   INPUT "Enter Operator's Name:",Name$
420   BEEP 3000,.03
430   INPUT "Enter Station Number:",Stat$
440   BEEP 3000,.03
450   OUTPUT @Rfna;"SYST:DATE?"
460   ENTER @Rfna;Year,Month,Day
470   CALL Draw_na      ! Draw HP8711
480   Box(418,44,212,163) ! Draw text box
490   PRINT TABXY(Tab,3);"Oper: ";Name$[1,15]
500   PRINT TABXY(Tab,4);"Station: ";Stat$[1,11]
510   PRINT TABXY(Tab,5);"Date: ";Year;Month;Day
520 Meas_dev: !
530   LOOP
```



```

540     CALL Draw_dut(1)
550     CALL Scan_dut(Scan$,Cent$,Span$,Loss$,Lim$(*))
560     PRINT TABXY(Tab,7);"Model: "&Scan$[1,6]
570     PRINT TABXY(Tab,8);"Serial: "&Scan$[8,12]
580     GOSUB Set_stim
590     DISP "MEASURING THE DEVICE"
600     OUTPUT @Rfna;"*CLS"
610     OUTPUT @Rfna;"ABOR;:INIT2:CONT OFF;:INIT2;*OPC?"
620     ENTER @Rfna;0pc ! wait for end of sweep
630     OUTPUT @Rfna;"CALC2:MARK1 ON;MARK:FUNC MAX"
640     OUTPUT @Rfna;"CALC2:MARK1:Y?"
650     ENTER @Rfna;Loss
660     OUTPUT @Rfna;"STAT:QUES:LIM:EVEN?"
670     ENTER @Rfna;Fail_flg
680 Disp_result:  !
690     PRINT TABXY(Tab,9);"Loss (dB): ";Loss
700     Fail_flg=BIT(Fail_flg,1) ! Bit 1 is for Ch2
710     IF Fail_flg THEN BEEP 2100,.5
720     Label(Test$(Fail_flg),78,58,22,5,0,1)
730 Continue:  !
740     CALL Draw_dut(0)
750     BEEP 300,.05
760     INPUT "Disconnect DUT. Measure another? (Y/n)",Ans$
770     EXIT IF UPC$(Ans$[1,1])="N"
780     Label(Test$(Fail_flg),78,58,22,5,0,0)
790     END LOOP
800     OUTPUT @Rfna;"ABOR;:INIT:CONT ON"
810     STOP
820     !
830 Set_stim: ! Set Freqs and Limit lines
840     OUTPUT @Rfna;"DISP:ANN:FREQ:MODE CSPAN"
850     OUTPUT @Rfna;"SENS:FREQ:CENT "&Cent$&" MHZ;SPAN "&Span$&" MHZ"
860     OUTPUT @Rfna;"DISP:WIND2:TRAC:Y:RLEV ";-PROUND(VAL(Loss$),1);"DB"
870     FOR I=1 TO 3     ! SET LIMIT LINES
880         OUTPUT @Rfna;"CALC2:LIM:SEGM"&VAL$(I)&":TYPE "&Lim$(I,1)&";STAT
ON"
890         OUTPUT @Rfna;"CALC2:LIM:SEGM"&VAL$(I)&":FREQ:STAR "&Lim$(I,2)&"
MHZ;STOP "&Lim$(I,3)&" MHZ"
900         OUTPUT @Rfna;"CALC2:LIM:SEGM"&VAL$(I)&":AMPL:STAR "&Lim$(I,4)&"
;STOP "&Lim$(I,5)
910     NEXT I
920     OUTPUT @Rfna;"CALC2:LIM:DISP ON;STAT ON"
930     RETURN
940     END
950     ! ##### SUBPROGRAMS #####
960     !
970 Draw_na:SUB Draw_na
980     ! This draws HP 8711 at origin X,Y
990     Box(144,62,287,125) ! Frame
1000    Box(144,62,289,127)
1010    Box(78,62,112,90) ! CRT
1020    Box(78,62,114,94)
1030    FOR I=25 TO 102 STEP 11! Keys
1040        Box(146,I,9,6)
1050    NEXT I

```

```

1060     Box(178,110,9,9)      ! BEGIN
1070     Box(234,110,65,15)   ! Drive
1080     Box(234,110,47,5)
1090     Circle(228,75,12)    ! Knob
1100     Circle(187,19,7)     ! Out
1110     Circle(256,19,7)     ! in
1120     Box(10,25,4,12)      ! Switch
1130     Circle(10,41,3)
1140     Label("RF OUT",187,35,8,5,0,1)
1150     Label("RF IN",256,35,8,5,0,1)
1160     SUBEND
1170     !
1180     Draw_dut:SUB Draw_dut(INTEGER Pen)
1190     ! This connects DUT to HP 8711
1200     PEN Pen
1210     Connect(187,19,200,-25,0)
1220     Box(221,-25,44,19)
1230     Connect(256,19,243,-25,0)
1240     PEN 1
1250     SUBEND
1260     !
1270     Scan_dut:SUB Scan_dut(Scan$,Cent$,Span$,Loss$,Lim$(*))
1280     LOOP
1290     Invalid=0
1300     Scan$="BPF175 12345"! Default model/serial
1310     BEEP 500,.05
1320     INPUT "Connect and scan the Device.",Scan$ ! SCAN BARCODE HERE
1330     IF LEN(Scan$)<12 THEN ! Valid device needs 12 char.
1340         Invalid=1
1350     ELSE
1360         Model$=Scan$[1,6]
1370         SELECT UPC$(TRIM$(Model$))
1380         CASE "BPF175","BPF177"
1390             RESTORE F1
1400         CASE "BPF200"
1410             RESTORE F2
1420         CASE "SAW134"
1430             RESTORE F3
1440         CASE ELSE
1450             Invalid=1
1460         END SELECT
1470     END IF
1480     EXIT IF NOT Invalid
1490     DISP Scan$;" <<--is INVALID! Try again."
1500     BEEP 1500,.2
1510     WAIT 1
1520     END LOOP
1530     BEEP 3000,.03
1540     READ Cent$,Span$,Loss$,Lim$(*)
1550     ! Limit lines format: Center, Span, Loss, (LIM TYPE, STRT, STP,
STRTdB, STPdB)
1560     F1: DATA 175,250,2 ! 175 MHz BPF
1570     DATA "LMIN", 160,190,-5,-5
1580     DATA "LMAX", 100,140,-50,-9
1590     DATA "LMAX", 210,240,-7,-30

```

```

1600 F2: DATA 200,100,1      ! 200 MHz BPF
1610   DATA "LMIN", 196,204,-3,-3
1620   DATA "LMAX", 180,190,-40,-10
1630   DATA "LMAX", 210,220,-10,-40
1640 F3: DATA 134,40,22     ! 134 MHz SAW BPF
1650   DATA "LMIN", 128,140,-27,-27
1660   DATA "LMAX", 123,125,-65,-30
1670   DATA "LMAX", 143,145,-30,-65
1680 SUBEND
1690 !
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 !
2080 Amp:SUB Amp(Xpos,Ypos,Size) ! Draws > Triangle
2090   COM /Scale/ Sc,INTEGER X,Y
2100   MOVE X+(Xpos+Size/2)*Sc,Y+Ypos*Sc
2110   POLYGON Size*Sc,3,3
2120 SUBEND

```

STATS — Using Bar Code Reader

```
10  ! -----
20  !
30  ! IBASIC program:  STATS - Collects statistics.
40  !
50  ! This HP 8711 IBASIC program uses a barcode reader
60  ! but it can be bypassed by simply pressing ENTER.
70  ! Displays running average of selected BPF passbands.
80  ! Finds linear avg of log data (ie Avg of 1dB & 5dB=3)
90  ! Expects to see BARCODE with the following format:
100 ! Model Number (6char), space, Serial Number (5char)
110 ! Valid Models:  BPF175, BPF200, SAW134
120 ! For 871xC and 8730A only
130 ! REV C.01.00    961009.JVV
140 !
150 ! -----
160 !
170 Init: !
180  COM /Hplib/ @Rfna
190  COM Csub_loaded
200  DIM A(1:1601),M(1:1601)
210  INTEGER Points,N,I,Chan
220  Points=201 ! # of trace points
230  Chan=2
240  ASSIGN @Rfna TO 800
250  IF NOT Csub_loaded THEN
260      LOADSUB Read_fdata FROM "XFER:MEM 0,0"
270      LOADSUB Write_fmем FROM "XFER:MEM 0,0"
280      Csub_loaded=1
290  END IF
300  OUTPUT @Rfna;"DIAG:PRES:SERV;*0PC?"
310  ENTER @Rfna;0pc
320  OUTPUT @Rfna;"DISP:PROG UPP"
330  GINIT
340  GCLEAR
350  OUTPUT @Rfna;"DISP:ANN:MESS:STAT 0"
360  OUTPUT @Rfna;"SENS1:STAT OFF;;SENS2:STAT ON"
370  OUTPUT @Rfna;"SENS2:SWE:POIN ";Points    ! points
380  OUTPUT @Rfna;"DISP:WIND2:TRAC:Y:RPOS 9;PDIV 1 DB;*0PC?"
390  ENTER @Rfna;0pc
400  N=0
410 Setup: !
420  LOOP
430      GOSUB Scan_next
440      ! Softkey titles formatted for 871x usage
450      ON KEY 1 LABEL " AVER THIS DATA" GOSUB Avg_this
460      ON KEY 3 LABEL "SCAN      ANOTHER" GOSUB Scan_next
470      ON KEY 5 LABEL "DONE" GOSUB Exit
480      LOOP
490          DISP "SELECT A SOFTKEY."
500          WAIT 1
510          DISP
520          WAIT .3
```

```

530     END LOOP
540 END LOOP
550 !
560 Exit: !
570 CLEAR SCREEN
580 DISP "PROGRAM PAUSED"
590 LOCAL @Rfna
600 PAUSE
610 RETURN
620 !
630 Scan_next: !
640 LOOP
650     Scan_dut(Model$,Serial$,Cent$,Span$,Loss$)
660     IF Model$="ABORT" THEN GOTO Exit
670     IF NOT N THEN Curr_model$=Model$
680 EXIT IF Model$=Curr_model$
690     DISP "Inconsistent Model #, Try again!"
700     BEEP 2100,.1
710     WAIT 1
720 END LOOP
730 CLEAR SCREEN
740 PRINT TABXY(1,4);"Device currently under test:"
750 PRINT "Model # ";Model$;" Serial # ";Serial$
760 PRINT TABXY(1,6);"# Avg'd:";N
770 PRINT TABXY(1,7);"Status of Serial # "&Serial$&": MEASURING  "
780 GOSUB Set_stim
790 RETURN
800 !
810 Avg_this: !
820 PRINT TABXY(1,7);"Status of Serial # "&Serial$&": READING DATA"
830 Read_fdata(Chan,A(*))
840 N=N+1
850 PRINT TABXY(1,7);"Status of Serial # "&Serial$&": AVERAGING  "
860 IF N=1 THEN
870     MAT M=A
880     OUTPUT @Rfna;"TRAC CH2SMEM,CH2SDATA;*WAI"
890     OUTPUT @Rfna;"CALC2:MATH (IMPL);:DISP:WIND2:TRAC1 ON;TRAC2
ON;*WAI"
900     OUTPUT @Rfna;"ABOR;:INIT2:CONT ON;*WAI"
910 ELSE
920     FOR I=1 TO Points
930         M(I)=(N-1)/N*M(I)+A(I)/N
940     NEXT I
950 END IF
960 PRINT TABXY(1,6);"# Averaged:";N
970 PRINT TABXY(1,7);"Status of Serial # "&Serial$&": WRITING DATA"
980 Write_fmem(Chan,M(*))
990 PRINT TABXY(1,7);"Status of Serial # "&Serial$&": AVG COMPLETE"
1000 GOSUB Scan_next
1010 RETURN
1020 !
1030 Set_stim:! Set Freqs
1040 OUTPUT @Rfna;"DISP:ANN:FREQ:MODE CSPAN"
1050 OUTPUT @Rfna;"SENS:FREQ:CENT "&Cent$&" MHZ;SPAN "&Span$&" MHZ"
1060 OUTPUT @Rfna;"DISP:WIND2:TRAC:Y:RLEV -"&Loss$&" DB;*OPC?"

```

```

1070 ENTER @Rfna;0pc
1080 RETURN
1090 !
1100 END
1110 !
1120 ! ##### SUBPROGRAMS #####
1130 !
1140 Scan_dut:SUB Scan_dut(Model$,Serial$,Cent$,Span$,Loss$)
1150     ALLOCATE Scan$[80]
1160     LOOP
1170         Invalid=0
1180         Scan$="ABORT"
1190         Scan$="BPF175 12345" !##### These 3 lines for demo only
1200         S$=VAL$(RND*1.E+9) !##### Generates random S/N
1210         Scan$[8,12]=S$[3,7] !##### Delete all to enable abort.
1220         BEEP 500,.05
1230         INPUT "Connect & scan DUT or leave blank to exit.",Scan$ !SCAN
BARCODE
1240         IF LEN(Scan$)<12 THEN ! Valid device needs 12 char.
1250             Invalid=1
1260         ELSE
1270             Model$=Scan$[1,6]
1280             SELECT UPC$(TRIM$(Model$))
1290             CASE "BPF175","BPF177"
1300                 RESTORE F1
1310             CASE "BPF200"
1320                 RESTORE F2
1330             CASE "SAW134"
1340                 RESTORE F3
1350             CASE ELSE
1360                 Invalid=1
1370             END SELECT
1380         END IF
1390     EXIT IF NOT Invalid
1400     IF POS(UPC$(Scan$),"ABORT") THEN
1410         Model$="ABORT"
1420         SUBEXIT
1430     END IF
1440     DISP Scan$;" <<--is INVALID! Try again."
1450     BEEP 1500,.2
1460     WAIT 1
1470     END LOOP
1480     BEEP 3000,.03
1490     Serial$=Scan$[8,12]
1500     READ Cent$,Span$,Loss$ ! Data format: Center, Span, Nom Loss
1510 F1: DATA 175,50,2 ! 175 MHz BPF
1520 F2: DATA 200,12,1 ! 200 MHz BPF
1530 F3: DATA 134,15,22 ! 134 MHz SAW BPF
1540 SUBEND

```

DATALOG — Using Bar Code Reader

```
1000 ! -----
1010 !
1020 ! IBASIC program:  DATALOG - Logs trace data
1030 !
1040 ! This HP 8711 IBASIC program uses a barcode reader
1050 ! but it can be bypassed by simply pressing ENTER.
1060 ! Stores ASCII trace data in internal NonVol memory
1070 ! until full, then copies stored files to floppy.
1080 ! Expects to see BARCODE with the following format:
1090 ! Model Number (6char), space, Serial Number (5char)
1100 ! Valid Models:  BPF175, BPF200, SAW134
1110 ! For 871xC and 8730A only
1120 ! REV C.01.00      961009.JVV
1130 !
1140 ! -----
1150 !
1160 Init:  !
1170 !
1180 !
1190 !  Make @Hp87xx common to all subroutines
1200   COM /Sys_state/ @Hp87xx,Scode
1210 !  Identify the computer we are running on
1220 !  and assign the i/o port address to @Hp87xx
1230   CALL Iden_port
1240 !
1250 !
1260   OUTPUT @Hp87xx;"DIAG:PRES:SERV;*OPC?"! Presets even 8730 to 871xC
state
1270   ENTER @Hp87xx;0pc
1280   OUTPUT @Hp87xx;"DISP:PROG UPP"
1290   GINIT
1300   GCLEAR
1310   GOSUB Warning    ! Warning may be deleted if desired
1320   OUTPUT @Hp87xx;"DISP:ANN:MESS:STAT 0"
1330   OUTPUT @Hp87xx;"SENS1:STAT OFF;:SENS2:STAT ON"
1340   OUTPUT @Hp87xx;"SENS2:SWE:POIN 201"! 201 points
1350   OUTPUT @Hp87xx;"DISP:WIND2:TRAC:Y:RPOS 9"
1360   OUTPUT @Hp87xx;"MMEM:MSIS 'MEM:'"
1370   OUTPUT @Hp87xx;"MMEM:INIT 'MEM:',DOS"
1380   OUTPUT @Hp87xx;"MMEM:STOR:STAT:IST OFF;CORR OFF;TRAC OFF;*OPC?"
1390   ENTER @Hp87xx;0pc
1400 Setup:  !
1410   LOOP
1420     GOSUB Scan_next
1430     ! Softkey titles designed for 871x format
1440     ON KEY 1 LABEL "STORE THIS DATA" GOSUB Stor_mem
1450     ON KEY 2 LABEL "TRANSFER TO FLOPPY" CALL Store_disk
1460     ON KEY 3 LABEL "SCAN      ANOTHER" GOSUB Scan_next
1470     ON KEY 5 LABEL "DONE" GOSUB Exit
1480     LOOP
1490     DISP "SELECT A SOFTKEY"
1500     WAIT 1
```

```

1510         DISP
1520         WAIT .3
1530     END LOOP
1540 END LOOP
1550 !
1560 Exit:    !
1570     Store_disk
1580     CLEAR SCREEN
1590     DISP "PROGRAM PAUSED"
1600     LOCAL @Hp87xx
1610     PAUSE
1620     RETURN
1630 !
1640 Scan_next:    !
1650     Scan_dut(Model$,Serial$,Cent$,Span$,Loss$)
1660     IF Model$="ABORT" THEN GOTO Exit
1670     CLEAR SCREEN
1680     PRINT TABXY(1,3);"Device currently under test:"
1690     PRINT
1700     PRINT "Model # ";Model$;"   Serial # ";Serial$
1710     PRINT TABXY(1,7);"Status of Serial # "&Serial$&": MEASURING      "
1720     GOSUB Set_stim
1730     RETURN
1740 !
1750 Stor_mem:    !
1760     PRINT TABXY(1,7);"Status of Serial # "&Serial$&": STORING TO RAM"
1770     Store_ram(Model$,Serial$)
1780     PRINT TABXY(1,7);"Status of Serial # "&Serial$&": STORING DONE  "
1790     GOSUB Scan_next
1800     RETURN
1810 !
1820 Set_stim:    ! Set Freqs
1830     OUTPUT @Hp87xx;"DISP:ANN:FREQ:MODE CSPAN"
1840     OUTPUT @Hp87xx;"SENS:FREQ:CENT "&Cent$&" MHZ;SPAN "&Span$&" MHZ"
1850     OUTPUT @Hp87xx;"DISP:WIND2:TRAC:Y:RLEV -"&Loss$&" DB;*OPC?"
1860     ENTER @Hp87xx;0pc
1870     RETURN
1880 !
1890 Warning:    !
1900     BEEP 3000,.3
1910     PRINT TABXY(15,4);"WARNING!"
1920     PRINT "This program will initialize the INTERNAL memory."
1930     PRINT "All internally saved files will be lost!"
1940     PRINT
1950     PRINT "Do you wish to continue? (y/N)"
1960     INPUT "Continue?",Ans$
1970     CLEAR SCREEN
1980     IF UPC$(Ans$[1,1])="Y" THEN RETURN
1990     END
2000 !
2010 ! ##### SUBPROGRAMS #####
2020 !
2030 Scan_dut:SUB Scan_dut(Model$,Serial$,Cent$,Span$,Loss$)
2040     ALLOCATE Scan$[80]
2050     LOOP

```



```

2060         Invalid=0
2070         Scan$="ABORT"
2080         Scan$="BPF175 12345"##### These 3 lines for demo only
2090         S$=VAL$(RND*1.E+9)##### Generates random S/N
2100         Scan$[8,12]=S$[3,7]##### Delete all to enable abort.
2110         BEEP 500,.05
2120         INPUT "Connect & scan DUT or leave blank to exit.",Scan$!SCAN
BARCODE
2130         IF LEN(Scan$)<12 THEN ! Valid device needs 12 char.
2140             Invalid=1
2150         ELSE
2160             Model$=Scan$[1,6]
2170             SELECT UPC$(TRIM$(Model$))
2180             CASE "BPF175","BPF177"
2190                 RESTORE F1
2200             CASE "BPF200"
2210                 RESTORE F2
2220             CASE "SAW134"
2230                 RESTORE F3
2240             CASE ELSE
2250                 Invalid=1
2260             END SELECT
2270         END IF
2280     EXIT IF NOT Invalid
2290         IF POS(UPC$(Scan$),"ABORT") THEN
2300             Model$="ABORT"
2310             SUBEXIT
2320         END IF
2330         DISP Scan$;" <<--is INVALID! Try again."
2340         BEEP 1500,.2
2350         WAIT 1
2360     END LOOP
2370     BEEP 3000,.03
2380     Serial$=Scan$[8,12]
2390     READ Cent$,Span$,Loss$
2400 ! Data format: Center, Span, Loss
2410 F1: DATA 175,300,2 ! 175 MHz BPF
2420 F2: DATA 200,100,1 ! 200 MHz BPF
2430 F3: DATA 134,30,22 ! 134 MHz SAW BPF
2440     SUBEND
2450 !
2460     SUB Store_ram(Model$,Serial$)
2470         COM /Sys_state/ @Hp87xx,Scode
2480         Id$=Model$[3,4]&"_"&Serial$! 2 unique chars + Ser
2490         ALLOCATE Err$[80]
2500         DISABLE
2510         REPEAT
2520             OUTPUT @Hp87xx;"*CLS"
2530             OUTPUT @Hp87xx;"MMEM:MSIS 'MEM:'"
2540             OUTPUT @Hp87xx;"MMEM:STOR:TRAC CH2FDATA,'"&Id$&'"';*WAI"
2550             OUTPUT @Hp87xx;"SYST:ERR?"
2560             ENTER @Hp87xx;Err$
2570             SELECT VAL(Err$)
2580             CASE 0! No Problem
2590             CASE -254! Internal Mem full

```

```

2600         CALL Store_disk
2610     CASE -257! dupl file name
2620         OUTPUT @Hp87xx;"MMEM:DEL '&Id$&'";*WAI" !ERASE OLD
2630     CASE ELSE
2640         BEEP 2000,.5
2650         DISP Err$;
2660         INPUT " Fix, Press ENTER",Ans$
2670     END SELECT
2680     UNTIL VAL(Err$)=0
2690     ENABLE
2700 SUBEND
2710 !
2720 SUB Store_disk
2730     COM /Sys_state/ @Hp87xx,Scode
2740     ALLOCATE Err$[80]
2750     BEEP 700,.1
2760     DISP "Standby: Transferring internal files to disk."
2770     LOOP
2780         OUTPUT @Hp87xx;"*CLS"
2790         OUTPUT @Hp87xx;"MMEM:COPY '*.*' , 'INT:';*WAI"
2800         OUTPUT @Hp87xx;"SYST:ERR?"
2810         ENTER @Hp87xx;Err$
2820     EXIT IF NOT VAL(Err$)
2830         GOSUB Trap_err
2840     END LOOP
2850     OUTPUT @Hp87xx;"MMEM:MSIS 'MEM:';DEL '*.*'"
2860     SUBEXIT
2870 !
2880 Trap_err:    !
2890     IF VAL(Err$)=-250 THEN SUBEXIT! no file to xfer
2900     BEEP 2000,.5
2910     CLEAR SCREEN
2920     PRINT TABXY(1,4);"DISK ERROR DETECTED"
2930     PRINT "*** "&Err$&" ***"
2940     INPUT "Fix above problem, then press ENTER",Ans$
2950     CLEAR SCREEN
2960 SUBEND
2970 !
2980 SUB Iden_port
2990 Iden_port:    ! Identify IO port to use
3000     COM /Sys_state/ @Hp87xx,Scode
3010     !
3020     IF POS(SYSTEM$("SYSTEM ID"),"HP 87")<>0 THEN
3030         ASSIGN @Hp87xx TO 800
3040         Scode=8
3050     ELSE
3060         ASSIGN @Hp87xx TO 716
3070         Scode=7
3080     END IF
3090 SUBEND! Iden_port

```



YES



BPF175 00123



NO



BPF175 00156



RUN



BPF175 00560



PAUSE



BPF175 00758



ABORT



BPF175 01023



CONT



BPF200 00045



ENTER (*blank*)



BPF200 00059



BENCH 4500 (*Station*)



BPF200 00077



BENCH 5300 (*Station*)



SAW134 00707



J VALLELUNGA



SAW134 00721



LENE CORRIGAN



SAW134 00776

Figure 11-1. Sample Bar Codes

Index

A

ABORT, 8-8
active controller, 2-7
allocating memory, 1-3
annotations
 custom, 7-5
arrays, 6-3
ASCII file, 4-2
ASCII word processors, 5-2
ASSIGN, 2-3
AUTOSTART, 4-3
AUTOSTART programs, 3-1

B

BARCODE
 example program, 11-4
bar codes
 sample, 11-53
binary file, 4-2
breakpoints
 setting, 6-2
buffer
 HP-IB, 2-2
 program, 2-1
bus
 external, 8-11
 internal, 8-11
bus management
 general, 8-5

C

character entry, 5-7
CLEAR, 8-7
codes
 bar, 11-53
continue command, 3-1
control
 passing, 8-10
controller
 active, 2-7
 synchronization, 8-14
 system, 2-7
controller, external
 interfacing with, 8-13
conventions
 typographical, 1-3

D

DATA_EXT
 example program, 11-1
DATA_INT
 example program, 11-1
DATALOG
 example program, 11-4
data transfer, 8-15
debugging, 6-1
deleted text
 to recall, 5-8
deleting text, 5-8
disabling the **Pause** key, 3-3
disk
 to select, 4-1
displaying text, 7-4
display partitions, 5-9, 7-1
DOWNLOAD
 example program, 11-1
downloading programs, 8-17
DRAW871X
 example program, 11-1
drawing figures, 7-7
DUALCTRL
 example program, 11-2

E

echo
 HP-IB, 2-8
Edit, 5-3
editing, 5-2, 5-6
editor
 HP BASIC, 5-2
errors
 displaying, 6-4
example program
 BARCODE, 11-4
 DATA_EXT, 11-1
 DATA_INT, 11-1
 DATALOG, 11-4
 DOWNLOAD, 11-1
 DRAW871X, 11-1
 DUALCTRL, 11-2
 REPORT, 11-2

- STATS, 11-4
- TRICTRL, 11-2
- UPLOAD, 11-2
- USERBEG, 11-2
- USERBEG1, 11-3
- USERBEG2, 11-3
- USER_BIT, 11-3
- USERKEYS, 11-3
- external bus, 8-11
- external controller
 - interfacing with, 8-13
- external keyboard, 3-2, 5-5

F

- file
 - ASCII, 4-2
 - binary, 4-2
- File Type, 4-2

G

- GET, 4-1
- graphics, 7-5
 - SCPI commands, 7-10

H

- HP BASIC editor, 5-2
- HP-IB
 - general structure, 8-2
- HP-IB buffer, 2-2
- HP-IB device selectors, 8-1
- HP-IB echo, 2-8

I

- IBASIC
 - synchronize with an external controller, 8-14
- IBASIC Display, 5-9
- IBASIC editor, 5-3
- IBASIC operations, 2-6
- IBASIC Utilities, 5-12
- instrument preset, 2-7
- internal bus, 8-11

K

- keyboard
 - external, 3-2, 5-5
- keyboard overlay, 3-2
- keystroke recording, 2-1

L

- label window, 5-7
- LINE TYPE, 7-9
- LOAD, 4-1
- local, 8-7

- LOCAL 8, 3-3
- local lockout, 8-6
- LOCAL LOCKOUT 8, 3-3
- lockout
 - local, 8-6

M

- memory allocation, 1-3, 3-2
- message windows, 7-5
- mnemonics
 - SCPI, 2-3

O

- operations
 - IBASIC, 2-6
- OUTPUT, 2-3
- overview, 1-2

P

- parallel port, 8-3
- partitions
 - display, 5-9, 7-1
- PASS CONTROL, 8-12
- passing control, 8-10
- PAUSE, 3-2
- Pause key, to disable, 3-3
- pausing a program, 3-2
- pop-up messages, 7-5
- ports
 - serial and parallel, 8-3
- PRESET, 2-7
- preset operation, 2-7
- program
 - to pause, 3-2
 - to recall, 4-3
 - to run, 3-1
 - to stop, 3-4
- program buffer, 2-1
- program recording, 2-1
- programs
 - downloading and uploading, 8-17

Q

- querying variables, 8-16

R

- READIO, 8-3
- recalling a deleted line, 5-8
- recalling a program, 4-3
- Recall Program, 4-3
- recording
 - keystroke, 2-1
- recording programs, 2-1
- reference material, 1-1

remote, 8-6
renumbering, 5-9
REPORT
 example program, 11-2
RE-SAVE, 4-1
RE-STORE, 4-1
RUN, 3-1
running a program, 3-1

S

sample bar codes, 11-53
SAVE, 4-1
 Save Program, 4-2
 SAVE RECALL, 4-1
SCPI mnemonics, 2-3
selecting a disk, 4-1
serial poll, 8-10
serial port, 8-3
service requests, 8-8
setting variables, 8-16
SRQs, 8-8
STATS
 example program, 11-4
status information, 8-14
STOP, 3-4
stopping a program, 3-4
STORE, 4-1
strings, 6-3
subprograms, 9-1
synchronization, 2-6, 8-14
system controller, 2-7

T

template
 keyboard, 3-2
text

 to display, 7-4
timing, 2-6
transferring data, 8-15
TRICTRL
 example program, 11-2
trigger, 8-7
typographical conventions, 1-3

U

UPLOAD
 example program, 11-2
uploading programs, 8-17
USERBEG
 example program, 11-2
USERBEG1
 example program, 11-3
USERBEG2
 example program, 11-3
USER_BIT
 example program, 11-3
USERKEYS
 example program, 11-3
Utilities, 5-12

V

variables, 6-2
 setting and querying, 8-16

W

window
 label, 5-7
word processors, 5-2
WRITEIO, 8-3

