DESIGNED FOR HP-IB SYSTEMS

Programming Note

8566A,8568A/9835,9845-51 NOVEMBER 1981

SUPERCEDES: NONE

Subprogram Library
for the 8566A or 8568A Spectrum Analyzer with the 9835 or 9845 Desktop Computer

Volume 1
- Program Form
- Error
- Save/Recall
- Olbits
- Yes/No
- Entry

HEWLETT PACKARD

Required reading . . .

8566A,8568A/9835,9845-1 Introductory Operating Guide
for the 8566A/8568A Spectrum Analyzers
with the 9835/9845 Desktop Computers
(P/N 5952-9356)

Also of interest. . .

Subprogram Library Tape Cartridge for the HP 8566A or 8568A
with the 9835 or 9845 (P/N 08566-10002 REV A)

8566A Spectrum Analyzer Remote Operation (P/N 08566-90003)

8568A Spectrum Analyzer Remote Operation (P/N 08568-90003)*

8568A/9825A-99 Program Execution Time Information for the
8568A/9825A Automatic Spectrum Analyzer System**(P/N 5952-9284)

*The 8568A Learn String is documented only in 8568A Remote manuals dated October, 1981 and later.

**Contains execution time data for the 8568A alone.

INTRODUCTION

The HP 8566A or 8568A Spectrum Analyzer can be controlled over HP-IB* with a computing controller such as the 9835 or 9845 Desktop Computer. Basic operation and programming of such automated spectrum analyzers is described in the 8566A,8568A/9835,9845-1 Introductory Operating Guide. The present series of Programming Notes, beginning here with Volume 1, will describe a number of subprograms which extend the value of the spectrum analyzer's built-in firmware and make it easier for you to develop your own custom software.

The subprograms perform tasks in several functional areas. These include determining the analyzer's current state, special marker and trace functions, calibrating equivalent analyzer bandwidths, and manipulating the analyzer settings for accurate amplitude measurements. The complexity ranges from high, as in the **Peaks** subprogram which can be used for sophisticated interpretation of trace data, to low, as in the **Save** subprogram, which simply reads the analyzer's Learn String into the controller's memory.

All, however, are high *level* subprograms in the sense that it is easy and convenient to refer to them in the context of a measurement program written in the System 35 or 45B BASIC language. Transparent to the user are the details of formatting and transferring data, reading analyzer functions without disturbing the current state, and verifying legal conditions (or providing error messages for illegal usage); these are left to the code within the subprograms.

The subprograms contained in Volume 1 perform elementary utility tasks and provide a foundation for the measurement oriented subprograms which are presented in later volumes. In addition, a **Program Form** is provided as an aid in developing well-structured main programs of your own, from which the various subprograms can be called as needed.

The **Error** program is used by other subprograms as a means of reporting error messages. **Save** and **Recall** provide high speed transfers of the Learn String out of and into the analyzer; **Save** also serves to identify the analyzer as an 8566A or 8568A. Having defined a Learn String either by calling **Save** or by loading a previously stored Learn String from a mass storage file, **Olbits** allows easy access to any bit field within the Learn String. **Yes, No,** and **Entry** are provided as convenient methods to respond to questions at the controller keyboard, or to enter values from the analyzer keypad.

Equipment Required

Operation of the subprograms described in this series of device subroutine procedures requires the following equipment:

1. 8566A or 8568A Spectrum Analyzer

2. 9835A/B Desktop Computer with 98332A I/O ROM, or
   9845B/T Desktop Computer with 98412A I/O ROM (Opt. 312)

3. 98034A/B HP-IB Interface.

Getting Started

The first step is to assemble the automatic spectrum analyzer components. Use the Introductory Operating Guide to help you set up and check-out the system, and begin programming.

To start programming for your specific signal analysis application, study the section below entitled "Writing Programs", which provides details on constructing programs with the subprograms presented in this series. It is assumed that you are familiar with the 8566A or 8568A Spectrum Analyzer Operation and Remote Operation Manuals.

*Hewlett-Packard Interface Bus, the Hewlett-Packard implementation of IEEE STD 488-1978 and ANSI STD MC 1.1, "Digital Interface for Programmable Instrumentation."

Refer to section entitled Subprogram Descriptions for detailed definitions of parameters and error codes, and examples of subprogram usage.

For advanced proramming, the operation of each subprogram is discussed on a line-by-line basis. Also refer to the 8566A or 8568A Spectrum Analyzer Remote Operation manuals for a summary of the contents of the analyzer's Learn String.*

Although the subprograms can be entered into the controller from the keyboard, it is recommended that you obtain the tape cartridge, P/N 08566-10002, which contains listings of the subprograms in the Subprogram Library. If you have this "master cartridge," duplicate it onto a blank cartridge. Save the master cartridge as a backup copy and use the new copy as a "working" cartridge.

WRITING PROGRAMS USING THE LIBRARY SUBPROGRAMS

The subprograms in this series provide high-level generalized routines which can be called upon to solve some of your individual measurement requirements. This section will show you how to assemble these subprograms with your main program and subprograms into a standard format, using the Program Form as a framework.

The Subprogram Library consists of two kinds of subprograms: *calls* and *functions*. These are described in detail in the System 35 or System 45B Desktop Computer Operating and Programming manual. Briefly, the difference between these two types of subprograms is the manner in which each passes values back and forth between itself and the calling program.

1. *Calls* pass values *only* by way of their parameter list. For example, a call with the name "Sub" having three parameters would be invoked from a program (or subprogram) by code such as:

```
10    CALL Sub(X,Y,Z)
```

2. *Functions* may also pass values between the calling program (or subprogram) and themselves by a parameter list. In addition, a function *always* returns one value by way of the *name* of the function itself; this is done by using the function in an arithmetic or logical expression in the calling program or subprogram:

```
   10    Z=FNSub(X,Y)+6.3

or 10    IF FNSub(X,Y)=Z THEN 150
```

The section entitled Subprogram Descriptions includes memory requirements; the memory requirement does not include 24 bytes corresponding to the "`! END OF PROGRAM`" statement at the end of each subprogram.

The Program Form and the subprograms, and their respective file names as they appear on the Subprogram Library Tape Cartridge, P/N 08566-10002, are listed in the following table:

Table 1. Subprogram Library Files

File Name	Subprogram	Type	EXTERNALS*
PROGRM	Program Form		______
ERROR	Error	CALL	______
SA/RE	Save; Recall	FN's	Error
OLBITS	Olbits	FN	Error; Save
YES/NO	Yes; No	FN's	______
ENTRY	Entry	FN	Interrupt (in file PROGRM)

*EXTERNALS are other subprograms which may be called by this subprogram and which therefore must also be appended.

*The 8568A Learn String is documented only in 8568A Remote manuals dated October, 1981 and later.

General Program Structure

The recommended structure for your programs is shown in Figure 1. This structure organizes the program to make it easy to troubleshoot and make additions and/or deletions.

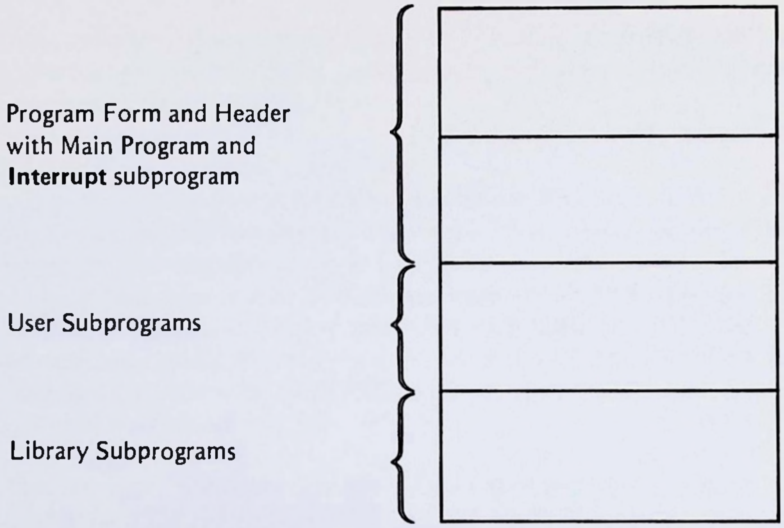
Program Form and Header with Main Program and **Interrupt** subprogram
User Subprograms
Library Subprograms

Figure 1. Recommended Program Structure

Program Form

The program form is comprised of program comments which contain general information about the program such as the title, date, author, and a statement describing what the program does. Controller initialization statements necessary for the main program's execution such as address assignments and format statements, as well as a basic interrupt-handling subprogram are also included. Program Form requires 1174 bytes of memory (including the "`! END OF PROGRAM`" statement at the end of the listing).

The program form provides a simple way to start writing your program. It provides generalized labels to assist your program documentation and defines the standard device name and address used by the Subprogram Library. It assures that the default format is STANDARD and the array indexing begins with one, not zero (OPTION BASE 1). The controller interrupt branching is defined and enabled. As you write additional software and add to the software library, you will want to make additions to, or modifications of the program form to suit your particular requirements.

Following the documentation/definition section of the form, a section of code is included that forcibly aborts all operations on the HP-IB, clears the analyzer, and locks the analyzer and other instruments on the HP-IB to prevent local (manual) operation. This assures a clean starting point for the hardware and controller to commence execution of your program.

When using the form for your own programs, be sure to delete any lines of code that are not relevant to your program. This will prevent wasting controller memory on unnecessary program statements.

To load the Program Form file from the Subprogram Library Tape Cartridge, insert the cartridge into the tape transport and type:

`GET"PROGRM"` (Press EXECUTE)

```
101    ! Program name/description
102    ! Author: File Name, YYMMDD
103    !  Program description comments...line 1
104    !  Program description comments...line 2
105    !
106    !
107        OPTION BASE 1
108        STANDARD
109    !
110        COM R,Sa,E$[20],L$[80]
111    !
112        Sa=718              ! 8566A/8568A
113        PRINTER IS 16       ! System printer
114    !
115        ON INT #7 CALL Interrupt
116           CONTROL MASK 7;128
117           CARD ENABLE 7
118    !
119        ABORTIO 7
120           CLEAR Sa
121           LOCAL LOCKOUT 7
122           OUTPUT Sa;"TS"
123    ! Program starts here...
1000   END
1001   !
1002   !
1003   ! ***************** Subprograms *********************
1004   !
1005   !
1006   SUB Interrupt
1007       OPTION BASE 1
1008       COM R,Sa,E$[20],L$[80]
1009       STATUS Sa;R
1010         IF NOT BIT(R,6) THEN Re_enable
1011         IF BIT(R,1) THEN Re_enable
1012         IF BIT(R,3) THEN DISP "HARDWARE BROKEN!"
1013         IF BIT(R,5) THEN DISP "ILLEGAL COMMAND!"
1014       PAUSE
1015 Re_enable: CONTROL MASK 7;128
1016            CARD ENABLE 7
1017   SUBEND
1018   ! End Interrupt
1019   !
1020   !
1021   ! END OF PROGRAM
```

101 – 104: General program information including the program name, author, file name, date, and program description.

107: Declare array indexing to begin at 1 (rather than 0).

108: Declare 9835/45 STANDARD (default) format.

110: Declare **R** (the analyzer status byte), **Sa** (the analyzer address), **E$** (the error code string), and **L$** (the learn string) to be accessible from the COMmon storage block.

112: Set spectrum analyzer address **Sa.**

113: Set system printer address.

115: Define label **Interrupt** as the name of HP-IB interrupt-handling subroutine.
116–117: Enable interrupts from the bus. If any interrupts (SRQ's) originate from Select Code 7 (the HP-IB), the program will branch to **Interrupt**.
119–122: Clear HP-IB; perform Instrument Preset on analyzer; execute LOCAL LOCKOUT on HP-IB; take one sweep.

123—1000: Insert your main program between these lines (see "Loading Your Program" for instructions).
1006: Commence interrupt processing. Refer to the Spectrum Analyzer Remote Operation Manual for more information on Service Requests (SRQ's).
1007: Declare array indexing to begin at 1.
1008: Declare COMmon block (identical to line 110).
1009: Read the status byte from the spectrum analyzer and save it as variable **R**.
1010: If **R**=0, the interrupt originated at some instrument other than the spectrum analyzer. The code provided here ignores such interrupts, goes to line 1015, re-enables the interrupt capability of the HP-IB, and returns to your program. If your system has other HP-IB instruments capable of generating interrupts, you will want to insert code here to process such interrupts.
1011: If bit 1 of the spectrum analyzer status byte is set, a units key or SHIFT r has been pressed on the analyzer. (Note that this interrupt, SQR 102, can occur only if you have programmed the spectrum analyzer to generate such interrupts with "R4.")

Interrupt ignores such interrupts, except to store the spectrum analyzer status byte in variable **R**; subroutine **Entry** (see "SUBPROGRAM DESCRIPTIONS") is an example of how this particular interrupt can be processed using the status byte information provided by **Interrupt.**

1012: If bit 3 is set, an SRQ 110 has been generated. Normally this occurs if a phase-lock error occurs, and there will be an appropriate error mssage (such as YTO UNLOCK, M/N UNLOCK, etc.) displayed on the spectrum analyzer CRT. Refer to the 8566A or 8568A Operation and Service Manual for more information on these messges. The program displays the error message "HARDWARE BROKEN!", goes to line 1014 and stops.
1013: If bit 5 is set, an SRQ 140 has been generated. This occurs when an illegal string of characters is transmitted over the HP-IB to the spectrum analyzer. The program displays "ILLEGAL COMMAND!" and stops.
1014: An SRQ 110 or SRQ 140 has occurred (line 1012 or line 1013).
1021: Append subprograms to this line (see "Appending Subprograms" for instructions).

Appending Subprograms

Once you have decided which subprograms to use in your program, you can append those subprograms to the program form. Table 1 shows the subprogram file names in the CATalog of the subprogram cartridge. Any or all of the subprograms can be appended with the following procedure:

1. Locate the number of the last line of the program:

System 35:	System 45:
EDIT 9999 (Press EXECUTE)	EDIT 32766 (Press EXECUTE)
(Press ⇩)	(Press ⇩)

For example, the controller might show:

```
1021 ! END OF PROGRAM
```

If you are using the Program Form and appending subprograms provided with the Subprogram Library, the last line will be "! END OF PROGRAM" (this is a non-functional comment statement) and the last line number is therefore 1021. If you are not using the program form, the last line should be any non-functional statement which can be written over.

If the last line of your program is *functional*, such as

```
1090 END
```

simply add one to the last program line number (in this case, 1090+1=1091) and continue with step 2.

2. Insert the Subprogram Library Tape Cartridge, P/N 08566-10002, into the controller's tape transport and append the desired subprogram. For example, to load the subprograms stored as file "SA/RE", beginning at line 1021 of the existing program, enter:

```
GET"SA/RE",1021        (Press EXECUTE)
```

If, after executing the GET command, one or more lines appear on the controller display with the message IMPROPER LINE NUMBER, it might be that there are insufficient additional line numbers available to accommodate the entire file which is being appended. In this case, the RENumber command can be used to condense the existing line numbers. Refer to the System 35 or System 45B Desktop Computer Operating and Programming manual for a description of this command.

3. The subprogram is now appended to your program. Additional subprograms can be appended to your new program (which now includes the subprogram just appended) by noting the new last program line and following the procedure outlined above.

   For example, find the last program line after appending the "SA/RE" file onto the Program Form; this should be line 1067. Now, append file "OLBITS":

```
GET"OLBITS",1067    (Press EXECUTE)
```

4. Most of the subprograms in the Subprogram Library use one or more other subprograms from the Subprogram Library. For example, the Save and Recall subprograms call subprogram Error, and so this subprogram must also be appended using the procedure in steps 1 and 2. Refer to Table 1 and add the required subprograms. The dependency of one program on others can also be determined by examining the program listing for EXTERNALS listed as a comment in the first few lines of code.

Loading Your Program

Once you have loaded the Program Form and appended any Library Subprograms that are required, you will want to add your main program and subprograms. The procedure for doing this utilizes the insert line INS LN editing key to add program lines. Your program lines should be inserted between lines 123 and 1000 of the form (see Figure 2).*

Example:

A program using the subprograms Save and Olbits to determine whether the sweep time is in the COUPLED ("AUTO") mode will be inserted into the Program Form between lines 123 and 1000.

1. Go to the line just below where the new program line is to be inserted.

```
EDIT 1000      (Press EXECUTE)
```

   Press INS LN

   The controller displays:

```
124  _
```

2. Type the line to be inserted:

```
DIM A$[9]             (Press STORE after each line)
A$="COUPLED"
CALL Save(Analyzer)
Flag=FNOlbits(18,0,1)
IF Flag THEN A$="UN"&A$
DISP "Sweep time is "&A$
```

*The gap between 123 and 1000 is arbitrary and is intended to allow you to conveniently insert a large number of program lines. Consult the System 35 or System 45B Operating and Programming manual on the usage of the line renumbering command REN.

The listing of this program segment will now be:

```
123   ! Program starts here...
124   DIM A$[9]
125   A$="COUPLED"
126   CALL Save(Analyzer)
127   Flag=FN01bits(18,0,1)
128   IF Flag THEN A$="UN"&A$
129   DISP "Sweep time is "&A$
1000  END
```

In lines 124 and 125, the string **A$** is dimensioned and initially assigned the string "COUPLED." The Learn String is obtained in line 126 by calling **Save** (with parameter **Analyzer**, not used here). In lines 127 and 128, the function subprogram **Olbits** is used to test a bit in byte 18 of the Learn String; if true, concatenate "UN" as a prefix for **A$** to form the message "UNCOUPLED." The message is displayed in line 129.

Programming Conventions

NOTE

1. Do not use the variable **R** or the strings **E$** and **L$** in the main program. Use the variable **Sa** only as the spectrum analyzer's address, as defined in line 112 of the Program Form. These variables and strings are used extensively in the Library Subprograms and reside at the beginning of the COMmon storage.

2. If you use COMmon storage to provide access to variables and strings among the main program and your subprograms, use a separate COM statement following, *not preceding,* the COMmon declaration in line 110 of the Program Form. This will ensure that the Library Subprograms will reference **R**, **Sa**, **E$**, and **L$** at the correct locations within the COMmon storage. Each time you use your own COMmon statement, be sure to precede it with:

```
COM R,Sa,E$[20],L$[80]
```

General Recommendations:

1. Use the Program Form (File: PROGRM) provided on the Subprogram Library Tape Cartridge, P/N 08566-10002.

2. Use a local lockout while remotely operating the HP 8566A or 8568A.

3. At the end of the program, return the HP 8566A or 8568A to local in an instrument preset (IP) condition.

4. Use the recommended mnemonics for device codes shown below.

5. Set the HP-IB select code to 7.

Recommended Device Codes

Sa	spectrum analyzer
Sg	signal generator
Lp	line printer
Pm	power meter
Gp	graphic plotter

SUBPROGRAM DESCRIPTIONS

Error

CALL
Outputs the subprogram error code on controller display.
File: ERROR

Description

This subprogram is called by another subprogram when it is unable to complete its tasks. The error message output by **Error** consists of two parts: (1) the name of the subprogram that detected the error, and (2) a numeric error code (as some subprograms have more than one detectable error condition). The error codes are established by the subprogram in which the error occurs. The CALL Error(Ern) statement in a subprogram must be followed immdiately by a PAUSE. Then, **Error** simply outputs the message and program execution is halted.

For example, if an error occurred in **PEAKS** the controller could display:

```
ERROR Save-1
```

This message, from the **Save** error code table, means that **Save** was passed an illegal 8566A or 8568A Learn String, and cannot continue.

When a program (or subprogram) invokes a subprogram incorrectly, **Error** prints the appropriate error message on the controller display and suspends program execution.

Press STEP the next line to be executed will be displayed. Note the line number

Press EDIT and enter the line number just noted.

Press EXECUTE to see the line immediately following the line containing the error.

The previous line should contain a CALL or FN reference to the subprogram in which the error occurred:

If this is not the case, a branch of the form

IF. . . THEN. . . or
ON . . . GOSUB . . . or
ON . . . GOTO . . .

might have branched to the displayed line. This can only be true if the offending subprogram is a function (FN) subprogram.

Parameters

Passed:
Ern = error number passed by the subprogram detecting the error.

Returned: none

Memory Required

426 bytes

Error Codes

None

Listing and Annotation for Error

```
101    ! Error  (File: ERROR)
102    ! HP 08566-10002, 810702
103    ! For HP 8566A or 8568A with System 35 or 45
104    ! CALL: Output error name from E$ and error #
105    !     Ern=error number
106    ! EXTERNALS: none
```

```
107    !
108    SUB Error(Ern)
109        COM R,Sa,E$[20],L$[80]
110        BEEP
111        DISP "ERROR "&E$;-ABS(Ern)
112    SUBEND
113    ! End Error
114    !
115    !
116    ! END OF PROGRAM
```

101 – 105: Description.
106: Requires no other subprogram.
108: Declare subroutine subprogram **Error** with parameter **Ern.**
109: Declare COMmon storage.
110: Signal operator with a BEEP.
111: Display error message on the controller. The subprogram name is stored in **E$** (accessed from COMmon). The error number is **Ern**, passed from the subprogram which detects the error.
112: Returns program execution to the calling program segment, at the line following the CALL Error (Ern) statement. This is normally a PAUSE statement. See the above "Description" of **Error** for instructions on responding to an error.

Save/Recall

CALLS

Transfers Learn String from the analyzer to **L$**, or from **L$** to the analyzer

File: SA/RE

Description

Subprogram **Save** provides a high speed transfer of the 8566A or 8568A "Learn String" to the string **L$** in the system controller. I/O ROM transfer type BFHS (byte-by-byte fast handshake) is used. The string **L$** (dimensioned for 80 bytes) resides in COMmon to provide ready access by other program segments which require the use of the "Learn String." Once the "Learn String" is in the system controller, it can be transferred to another string for storage, using simple string assignment statements. The string array **R$** ("Recall String") is easy to remember for this purpose. The string **L$** can be transmitted back to the analyzer via the **Recall** subprogram.

Parameters

Passed:
none

Returned:
Analyzer = 8566 or 8568, depending on type of analyzer at specified address

Example

```
150   DIM R$[80]
160   CALL Save(Analyzer)
170   R$=L$
.
.
.
250   L$=R$
260   CALL Recall
```

150 **R$** dimensioned for 80 bytes as a buffer to store the current **L$** Learn String (and therefore, the analyzer state) until it is required later in the program.
160 Call **Save**. (Return parameter **Analyzer** is not used here.)
170 **R$** buffers the current **L$** until required later.
250 **L$** reset to **R$** in preparation for **Recall**.
260 Call **Recall** to restore analyzer to earlier state.

Memory Required

1430 bytes

Error Codes

Recall-1: String **L$** does not contain an 8566A or 8568A Learn String.

Save-1: String read was not an 8566A or 8568A Learn String.

Listing and Annotation for Save/Recall

```
101    ! Save/Recall   (File: SA/RE)
102    ! HP 08566-10002, 810702
103    ! For HP 8566A or 8568A with System 35 or 45
104    ! CALL's: Buffered Save/Recall of 8566A control state via L$
105    !    Analyzer=analyzer identification, 8566 or 8568  (R)
106    ! EXTERNALS: Error
107    !
108    SUB Save(Analyzer)
109        OPTION BASE 1
110        COM R,Sa,E$[20],L$[80]
111        OUTPUT Sa;"OL"
112        ENTER Sa BFHS 80 NOFORMAT;L$
113        Last_byte=FNLstr(80,0)
114        IF Last_byte=162 THEN Analyzer=8566
115        IF Last_byte=165 THEN Analyzer=8568
116        IF (Last_byte<>162) AND (Last_byte<>165) THEN Err
117        SUBEXIT
118 Err:   E$="Save"
119        CALL Error(1)
120        PAUSE
121    SUBEND
122    !
123    !
124    SUB Recall
125        OPTION BASE 1
126        COM R,Sa,E$[20],L$[80]
```

101 – 105: Description.
106: Requires **Error**.
108: Declare subroutine subprogram **Save** with parameter **Analyzer**.
109: Declare array indexing to begin at 1.
110: Declare COMmon storage.
111 – 112: Transfer Learn String from spectrum analyzer to string **L$** in controller using a byte-by-byte fast handshake.
113: Save the last byte of the Learn String as the variable **Last__byte**.
114: If **Last__byte** equals 162, then the spectrum analyzer must be an 8566A. Save the value 8566 in the parameter **Analyzer**.
115: If **Last__byte** equals 165, then the spectrum analyzer must be an 8568A. Save the value 8568 in **Analyzer**.
116: If **Last__byte** does not equal 162 or 165, then the Learn String did not come from either an 8566A or 8568A. An error message is returned by branching to line 133.
117: Return from the subroutine to the calling program segment.
118: Define **E$** to be the literal string "Save."
119: Call subroutine **Error** with **Ern** = 1.
120: Stop program execution. (Required after calling **Error**.)
121: Return from the subroutine to the calling program segment.
124: Declare subroutine **Recall**.
125: Declare array indexing to begin at 1.
126: Declare COMmon storage.

```
127          IF FNLstr(1,0)<>31 THEN Err
128          Last_byte=FNLstr(80,0)
129          IF (Last_byte<>162) AND (Last_byte<>165) THEN Err
130          OUTPUT Sa BFHS  NOFORMAT;L$
131          OUTPUT Sa;"HD"
132          SUBEXIT
133 Err:     E$="Recall"
134          CALL Error(1)
135          PAUSE
136     SUBEND
137     !
138     !
139     DEF FNLstr(Bnum,Shift)
140        OPTION BASE 1
141        COM R,Sa,E$[20],L$[80]
142        RETURN SHIFT(NUM(L$[Bnum;1]),Shift)
143     FNEND
144     ! End Save/Recall
145     !
146     !
147     ! END OF PROGRAM
```

127 – 129: Test for errors: the value of the first byte of the Learn String must be 31, and the value of the last byte either 162 or 165. (Refer to the 8566A or 8568A Spectrum Analyzer Remote Operation Manual for more information on the contents of the Learn Strings.)

130: Output the string **L$** from the controller to the spectrum analyzer (the analyzer will recognize it as a Learn String by the first and last bytes, as verified in lines 127 – 129).

131: Disable the active function area of the CRT.

132: Return to the calling program segment.

133: Define **E$** as the literal string "Recall."

134: Call the **Error** subroutine with parameter 1.

135: Stop program execution after sending error message.

136: Return to the calling program segment.

139 – 143: Define internal function subprogram **Lstr** with parameters **Bnum** and **Shift**. Subprogram **Lstr** fetches a byte from the Learn String (byte number **Bnum**), and converts it to an equivalent decimal numeric value. The resulting bit pattern is shifted by the number of positions specified by **Shift**. The result of these operations is returned to the calling program segment. Note that the subprogram **Lstr** is identified as an internal subprogram. It is used only by other subprograms in the Subprogram Library. The more generalized subprogram, **Olblts**, is recommended for general use.

Olbits

FUNCTION

Returns a bit field from the 8566A or 8568A "Learn String."

File: OLBITS

Description

Function subprogram **Olbits** is used by a program or subprogram to read specific instrument state information from bytes of the 80-byte OL learn string. **Olbits** provides access to any byte of the learn string and supplements **Functions** and **State**. The data structure of the OL learn string is discussed in the 8566A or 8568A Spectrum Analyzer Remote Operation manual.*

Parameters

Passed:

Bnum = byte number in learn string, L$, 1 to 80.
Bit = least significant bit of field to be returned, 0 to 7 (where 0 is least significant bit of byte Bnum).
Width = number of bits (width of field) to be returned, 1 to (8-**Bit**).

Returned:

FNOlbits = value of bit field, in decimal

*The 8568A Learn String is documented only in 8568A Remote manuals dated October, 1981 and later.

Example

To return all 8 bits of byte number 1 in the OL Learn String, use the following:

```
           PROGRAM                              DISPLAY
  40    A=FNOlbits(1,0,8)
  50    DISP A                                    31
```

Memory Required

886 bytes

Error Code

Olbits-1: parameter **Bnum** must be in range 1-80; parameter **Bit** must be in range 0-7; parameter **Width** must be in range 1 to (8-**Bit**).

Listing and Annotation for Olbits

```
101    ! Olbits  (File: OLBITS)
102    ! HP 08566-10002, 810702
103    ! For HP 8566A or 8568A with System 35 or 45
104    ! FN: Return bit field from learn string in L$
105    !    Bnum=byte # in L$, 1 to 80
106    !     Bit=least significant bit of field, 0 to 7
107    !   Width=# of bits (width of field), 1 to 8-Bit
108    ! EXTERNALS: Error; Save
109    !
110    DEF FNOlbits(Bnum,Bit,Width)
111        OPTION BASE 1
112        COM R,Sa,E$[20],L$[80]
113        IF (Bnum<1) OR (Bnum>80) THEN Err
114        IF (Bit<0) OR (Bit>7) THEN Err
115        IF (Width<1) OR (Width>8-Bit) THEN Err
116        CALL Save(Analyzer)
117        RETURN SHIFT(BINAND(FNLstr(Bnum,Bit+Width-8),255),8-Width)
118 Err:   E$="Olbits"
119        CALL Error(1)
120        PAUSE
121        RETURN 0
122    FNEND
123    ! End Olbits
124    !
125    !
126    ! END OF PROGRAM
```

101 – 107: Description.
108: Requires **Error** and **Save**.
110: Define function subprogram **Olbits** with parameters **Bnum, Bit**, and **Width**.
111: Declare array indexing to begin at 1.
112: Declare COMmon storage.
113 – 115: Test for illegal parameter values.
116: Load **L$** with the Learn String via **Save**.
(Return parameter **Analyzer** is not used here.)
117: Get decimal numeric equivalent of bits 0 through (**Bit** + **Width** – 1) of byte **Bnum**, using the internal subprogram **Lstr** (in subprogram file SA/RE). Note that the selected bits are shifted left so that the most significant bit of the specified field is shifted to bit position 7. The result will occupy the eight least significant bits of a 16 bit word in the controller. Zero the eight most significant bits of the 16 bit word

while preserving the other eight bits by "band"ing the word with the mask 255 (255_{10} = 0000000011111111_2). Shift the result to the right by (8 – **Width**) positions so the 1st bit of the requested field is in position 0. Return this result.

118: Store the literal string "Olbits" in **E$**.
119 – 120: Send error message and stop program execution.
121: Return 0 as the function **Olbits** value if error encountered.
122: End of function subprogram **Olbits**.

Yes/No

FUNCTION
Returns logical value (0 or 1) when operator presses Y or N on controller keyboard.
File: YES/NO

Description

These function subprograms allow program questions to be answered from the system controller keyboard.

The function **FNYes** will return 1 for 1 or Y and 0 for 0 or N followed by CONTINUE .

The function **FNNo** will return 1 for 0 or N and 0 for 1 or Y followed by CONTINUE .

Pressing any other alphanumeric key will cause an error/information message to appear on the controller's display.

```
Answer Y or N.
```

Note: Y,y,1,N,n, or 0 may be pressed.

Parameters

Passed: none

Returned:
FNYes = 1 for y, Y, or 1 followed by CONTinue on controller keyboard.
= 0 for n, N or 0
FNNo = 1 for n, N or 0 followed by CONTinue on controller keyboard.
= 0 for y, Y, or 1

Example

To ask whether the program should continue, consider the following branch program:

```
190  DISP "Continue?"
200  IF FNNo THEN PAUSE
210  Continue:  !
```

N CONTINUE on the controller keyboard stops the program.

Y CONTINUE continues the program on line 210.

A meaningless answer such as Z CONTINUE will display

```
Answer Y or N.
```

and cause the program to wait for an acceptable answer.

Memory Required

1434 bytes

Error Codes

None

Listing and Annotation for Yes/No

```
101    ! Yes/No  (File: YES/NO)
102    ! HP 08566-10002, 810702
103    ! For HP 8566A or 8568A with System 35 or 45
104    ! FN's: Read Y,y,1 or N,n,0 from keyboard
105    !     FN=1 if answer is true; 0 if false (R)
106    ! EXTERNALS: none
107    !
108    DEF FNYes
109        OPTION BASE 1
110        DIM S$[10]
111 Input: INPUT "",S$
112        IF LEN(S$)=0 THEN Err
113        S=NUM(S$[1;1])
114        IF (S<>121) AND (S<>89) AND (S<>49) THEN 118
115           Ans=1
116           DISP "YES"
117           GOTO Exit
118        IF (S<>110) AND (S<>78) AND (S<>48) THEN Err
119           Ans=0
120           DISP "NO"
121 Exit:  WAIT 500
122        RETURN Ans
123 Err:   BEEP
124        DISP "Answer Y or N."
125        WAIT 1000
126        GOTO Input
127    FNEND
128    !
129    !
130    DEF FNNo
131        OPTION BASE 1
132        DIM S$[10]
```

101 – 105: Description.
106: No other subprogram required.
108: Define function subprogram **Yes**.
109: Declare array indexing to begin at 1.
110: Dimension **S$** for up to 10 characters; **S$** will be used to store the keyboard entry.
111: Read controller keyboard entry, terminated by CONTINUE, into **S$**.
112: If at least one key was pressed before CONTINUE, go to line 113. Otherwise, go to line 123.
113: Convert the ASCII character of the first key pressed before CONTINUE to its numeric equivalent, **S**.
114 – 117: If **S** matches Y, y, or 1, set **Ans** = 1, display "YES", and go to line 121.
118 – 120: If **S** matches N, n, or 0, set **Ans** = 0, display "NO", and go to line 121. Otherwise, go to line 123.
121 – 122: Return parameter **Ans** to calling program segment.
123 – 126: No key was pressed before CONTINUE (see line 112), or a key other than Y, y, 1, N, n, or 0 was pressed. Display error message ("Answer Y or N"), then go to line 111 for another entry.
127: End of function subroutine **Yes**.
130: Define function subprogram **No**.
131: Declare array indexing to begin at 1.
132: Dimension **S$** for up to 10 characters; **S$** will be used to store the keyboard entry.

```
133 Input:  INPUT "",S$
134         IF LEN(S$)=0 THEN Err
135         S=NUM(S$[1;1])
136         IF (S<>121) AND (S<>89) AND (S<>49) THEN 140
137             Ans=0
138             DISP "YES"
139             GOTO Exit
140         IF (S<>110) AND (S<>78) AND (S<>48) THEN Err
141             Ans=1
142             DISP "NO"
143 Exit:   WAIT 500
144         RETURN Ans
145 Err:    BEEP
146         DISP "Answer Y or N."
147         WAIT 1000
148         GOTO Input
149    FNEND
150    ! End Yes/No
151    !
152    !
153    ! END OF PROGRAM
```

133: Read controller keyboard entry, terminated by CONTINUE, into **S$**.
134: If at least one key was pressed before CONTINUE, go to line 135. Otherwise, go to line 145.
135: Convert the ASCII character of the first key pressed before CONTINUE to its numeric equivalent, **S**.
136 – 139: If **S** matches Y, y, or 1, set **Ans** = 0, display "YES", and go to line 143.
140 – 142: If **S** matches N, n, or 0, set **Ans** = 1, display "NO", and go to line 143. Otherwise, go to line 145.
143 – 144: Return parameter **Ans** to calling program segment.
145 – 148: No key was pressed before CONTINUE (see line 134), or a key other than Y, y, 1, N, n, or 0 was pressed. Display error message ("Answer Y or N"), then go to line 133 for another entry.
149: End of function subprogram **No**.

Entry

FUNCTION
Returns analyzer's number keyboard entry.
File: ENTRY

Description

The 8566A or 8568A Spectrum Analyzer data keyboard can be used to enter integer numbers to the 9835 or 9845 controller program via function **Entry. Entry** has two modes of operation:

1. Press a sequence of 8566A or 8568A DATA numeric keys, terminate entry with a Units key. Values returned can be any integer between 1 and 999999999999 ($10^{12}-1$).

2. Single key pressed (non-zero numeric or Units key).

   Values returned can be 1, 2, 3, 4, 5, 6, 7, 8, 9, 10^3 (kHz key), 10^6 (MHz key) or 10^9 (GHz key).

   Note: **Entry** leaves the ENTRY mode ("EE") as the active function on the CRT of the analyzer. To clear this from the display and disable the keyboard of the analyzer after using **Entry**, send "HD" to the analyzer (or enable any other function):

```
10   A=FNEntry(0)
20   OUTPUT Sa;"HD"
```

Parameters

Passed:

Mode: Specifies data entry mode.

Mode = 0: Data entry on the analyzer keyboard is terminated by pressing a Units key. Values entered can be integers from 1 to 999999999999 ($10^{12}-1$). The Units key pressed multiplies the entered value by the frequency unit (Hz = x1; kHz = x10^3, MHz = x10^6, GHz = x10^9).

Mode #0: Data entry from the analyzer keyboard by pressing a single, non-zero key (either numeric or a Units key). The value entered can be 1, 2, 3, 4, 5, 6, 7, 8, 9, 10^3, 10^6, or 10^9.

Returned:

FNEntry = value entered on analyzer DATA keyboard.

Examples

The following examples show the two entry types. The displayed output at the right is the result of the corresponding analyzer keyboard entry:

In the first type, the value is read and printed only after the units key is pressed.

```
140   E=FNEntry(0)          1  2  3  .  4  6  6  kHz mV msec
150   DISP E                      123456
```

or

```
140   E=FNEntry(0)          1  2  3  .  4  5  6  Hz µV µsec
150   DISP E                        123
```

Note that the actual value is always an integer; fractional parts of the number entered are truncated.

In the second type, the value is displayed when any non-zero DATA key (*or* a units key) is pressed.

```
140   E=FNEntry(1)          2
150   DISP E                2
```

or

```
140   E=FNEntry(1)          kHz mV msec
150   DISP E                1000
```

Memory Required

1022 bytes

Error Codes

None

Listing and Annotation for Entry

```
101   ! Entry  (File: ENTRY)
102   ! HP 08566-10002, 810702
103   ! For HP 8566A or 8568A with System 35 or 45
104   ! FN: Suppress other interrupts, return 8566A/8568A DATA key(s) entry
105   !         as an integer.
106   !      FN=Keyboard entry (R)
107   !    Mode=0: Requires units key as terminator.
108   !         #0 read one key only (1-9 or units multiplier); 0 not allowed
109   !      (R=Status byte from analyzer)
110   ! EXTERNALS: Interrupt
```

```
111    !
112    DEF FNEntry(Mode)
113          COM R,Sa,E$[20],L$[80]
114          ON INT #7 CALL Interrupt
115 Start:   OUTPUT Sa;"EE OA"
116          ENTER Sa;Entry
117          IF Entry AND NOT Type THEN Start
118          Type=Mode
119          IF NOT Type THEN Units
120          IF Entry THEN Exit
121          GOTO Start
122 Units:   R=BINAND(R,253)
123          OUTPUT Sa;"R1 R3 R4"
124 Idle:    IF NOT BIT(R,1) THEN Idle
125          OUTPUT Sa;"OA"
126          ENTER Sa;Entry
127 Exit:    RETURN Entry
128    FNEND
129    ! End Entry
130    !
131    !
132    ! END OF PROGRAM
```

101 – 109: Description.
110: **Interrupt** subroutine required (see Program Form).
112: Define function subroutine **Entry** with parameter **Mode**.
113: Declare COMmon storage.
114: Define label **Interrupt** as the name of HP-IB interrupt-handling subroutine.
115 – 117: Verify that no key is pressed as **Entry** commences (i.e., "debounce" keys): Enable keyboard entry from analyzer and read value into **Entry**. If a non-zero value is read, and **Type** is still zero (it is initialized to zero by the controller when the **Entry** subroutine program segment is entered), the operator must be holding down a key (either 1-9 or a Units key). Execute from line 115 again until a "clean" start can be made.
118: Assign entry type **Mode** to **Type**.
119: If **Type** is non-zero, single key entry type was requested; go to line 120. If **Type** is zero, then an entry sequence terminated with a units key was requested; go to line 122.
120 – 121: If a non-zero key has been pressed since the debouncing in lines 115-117, then go to line 127 to return the entered value **Entry**. Otherwise, go to line 115 to read another data value.
122: An entry sequence with a units key terminator has been requested. Set bit 1 of the spectrum analyzer status byte, **R**, to 0.
123: Disable interrupts except for Illegal Command interrupt, "R1". Enable Hardware Broken interrupt, "R3" and Units Key interrupt, "R4."
124: Enter an "idle" mode, repeatedly executing line 124 until an interrupt occurs. When an interrupt appears on the interface bus, the program branches to the defined interrupt routine. Line 115 of the **Program Form** defines such a branch; the **Interrupt** subroutine is listed in lines 1006-1017 of the Program Form. If the Units Key interrupt has occurred, the **Interrupt** program will, after reading the status byte into **R** (accessible from COMmon storage), recognize that bit 1 has been set and re-enable the controller to respond to HP-IB service requests. Control is then returned to the **Entry** subprogram at the beginning of line 124. Now, bit 1 of **R** is set, so go to line 125.
125 – 126: Output the value of the active function (keyboard entry) from the spectrum analyzer to the controller.
127: Return the entered value **Entry** to the calling program segment.

HEWLETT
PACKARD

For more information, call your local HP Sales Office or nearest Regional Office: Eastern (201) 265-5000; Midwestern (312) 255-9800; Southern (404) 955-1500; Western (213) 970-7500; Canadian (416) 678-9430. Ask the operator for instrument sales. Or write Hewlett-Packard, 1501 Page Mill Road, Palo Alto, CA 94304. In Europe: Hewlett-Packard S.A., 7, rue du Bois-du-Lan, P.O. Box, CH 1217 Meyrin 2, Geneva, Switzerland. In Japan: Yokogawa-Hewlett-Packard Ltd., 29-21, Takaido-Higashi 3-chome, Suginami-ku, Tokyo 168.

Printed in U.S.A.