HEWLETT-PACKARD JOURNAL
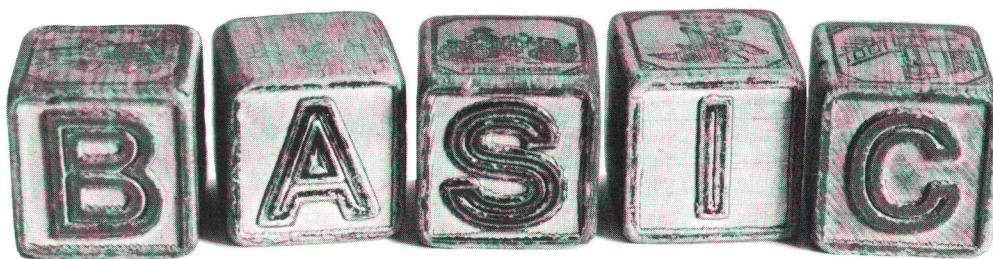
BASIC

The Language of Time Sharing

A computer language designed for the beginner and the once-in-a-while programmer, BASIC is powerful, yet easy to learn. Try it and see.

By Gerald L. Peterson

SINCE THE INVENTION OF THE FIRST ELECTRONIC COMPUTER some 20 years ago, men have been searching for better ways to communicate with these wonderful thinking machines. One approach has been to design computer languages that can be easily understood both by men and by computers. The most successful of these languages has been FORTRAN, in which computations and program steps are specified using a mixture of English and algebra. Other languages, such as ALGOL and PL-1, while not as widely used as FORTRAN, are good examples of how computer languages are evolving; they are becoming more flexible and they are putting more powerful features at the user's disposal.

Unfortunately, the built-in flexibility and power of these languages gives them an inherent disadvantage for the 'once-in-a-while' programmer. Generally speaking, the more flexible a language becomes, the more conventions and options a person has to remember to use the language efficiently. Often a user finds that, even though he has once mastered the language, if he hasn't used it for a few months he will have forgotten the many rules necessary to write a working program, and will essentially have to relearn the language. What's more, most programs are run today in 'batch' mode; the programmer leaves his program at the computation center with hundreds of other programs and returns several hours later to claim the results of his run. Often a simple error or two has kept his program from executing correctly, and he has to repeat the cycle. For the beginning programmer, this can be particularly frustrating.

Time Sharing

In the past five years, a new phenomenon called time sharing has appeared on the computer scene. The time-sharing system is a complex system of computer hardware and specialized software which allows several users to sit at typewriter-like terminals and communicate with the computer simultaneously. These terminals may be coupled to a central computer many miles away through telephone circuits. A user sitting at his terminal has the illusion that he has the computer to himself, even though the computer is miles away and several other people are using it at the same time.

With time sharing, almost anyone can have access to a computer. However, the computational power available through time sharing isn't really useful to someone who can't write programs. To make this computational power useful to large numbers of people, simpler computer languages had to be developed.

Several time-sharing languages have now been designed. These languages are not only easy to learn but also conversational, that is, the user can interact with his program and with the time-sharing system from the remote terminal. The time-sharing system checks the pro-

gram for errors as soon as it is composed; then, after it is 'debugged', it may be run immediately and the answers obtained.

BASIC

The conversational language that has proved most popular is called BASIC. It was developed at Dartmouth College. In September of 1963, Professors John G. Kemeny and Thomas E. Kurtz of Dartmouth launched a project that was to have a major effect on the computer industry. The project, which was supported in part by the National Science Foundation, was to build and operate a time-sharing system using General Electric Company computers and a language that was to be developed at Dartmouth as part of the project. Two short-term goals were in mind at the beginning of the program; the first was to introduce the computer to a majority of the students at Dartmouth, and the second was to make the computer more readily accessible to the faculty.

Work began on the project and by the spring of 1964, BASIC, the 'Beginner's All-purpose Symbolic Instruction Code', was born. Students and faculty alike used the new system so enthusiastically that four years later, in their final report on the Dartmouth Time Sharing Computing System, dated June 1967, Kemeny and Kurtz reported that they had introduced some 2000 students to BASIC and to the computer, and that 40% of the faculty used the system for a wide variety of projects.

The underlying motivation for the Dartmouth project went somewhat deeper than the short-term goals indicate. With the increased dependence of our modern society on computers, it is vitally important that the typical attitude of the intelligent person toward computers be changed. This attitude, a kind of superstitious awe of the machine, can best be dispelled by having people use the computer to solve day-to-day problems, whether they be large or small. The prerequisite to doing this is to have the computer easily accessible and easily programmable.

At Dartmouth this has been achieved. Even the project directors admit to a major change in their approach to computers. They have learned to work with the computer in solving a problem, instead of just submitting a problem to the computer for machine solution.

One factor responsible for the success of the Dartmouth project was the time-sharing concept. However, the major factor was the easy-to-use language, BASIC.

BASIC was originally designed to be such a simple language that students could learn to do useful things with it after two one-hour lectures. The language has since been expanded into a more powerful, general-purpose programming language, but the simple core has been retained. The novice can use the simple statements of elementary BASIC whether or not he is aware of the additional features available to the more expert programmer.

How to Speak BASIC

The best way to learn BASIC is to sit down at the teletypewriter, Fig. 1, and do some BASIC programming. The teletypewriter keyboard is similar to a typewriter's, but there are no lower-case letters. On some teletypewriters, zeros have a slash through them to differentiate them from the letter 'oh'. There is no backspace key, but BASIC does allow the programmer to correct errors as he types. Typing a left arrow (←) effectively backspaces
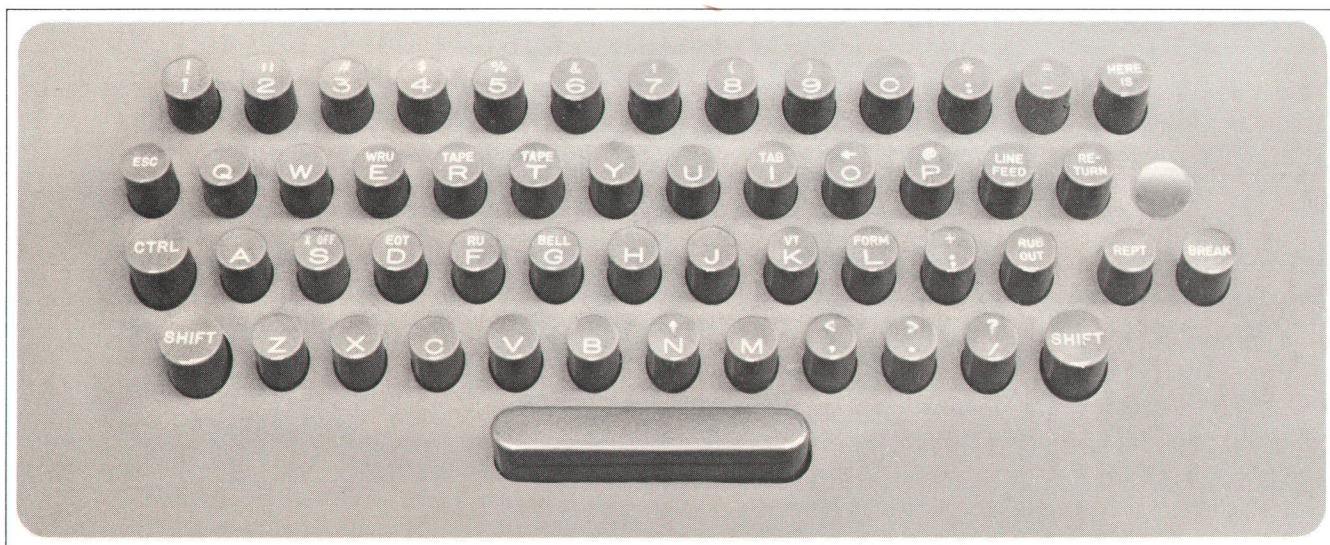

Fig. 1. *The computer and the BASIC programmer talk to each other via the teletypewriter.*

the teletypewriter one space. For example:

```
110 FER←←OR I = 1 TO 10
```

would be recognized as

```
110 FOR I = 1 TO 10
```

Hitting the key marked RETURN signifies the end of the statement and returns the teletypewriter to the left edge of the paper; the computer will send a LINE FEED to the teletypewriter to advance the paper one line and indicate that it is ready for the next statement.

Here is an example of a simple program in BASIC; it's a program for calculating the hypotenuse of a right triangle.

```
10 LET X = 5
20 LET Y = 10
30 LET H1 = SQR(X↑2+Y↑2)
40 PRINT "THE HYPOTENUSE IS",H1
50 END
```

Two things are immediately evident in this complete BASIC program. The first is that all statements begin with a number which serves to identify that particular statement and shows the position of that statement in relation to the rest of the program statements. Second, the statement itself, which follows the statement number, always begins with an English word that serves to identify what type of statement it is.

Statements need not be entered in order as the program is composed, but when the program is executed the statements will be executed in order of ascending statement numbers. Experienced programmers usually choose statement numbers which are multiples of five or ten; this is to leave room for inserting additional statements later, should they be needed.

A statement can be deleted from a program by typing the statement number and then hitting the RETURN key.

Variables, Constants, and Functions

There are three variables in the above program: X, Y, and H1. A variable may be named in BASIC with a single letter of the alphabet or with a letter of the alphabet followed by a single digit, 0 through 9.

The three constants in the program (2, 5, and 10) all happen to be integers. However, decimal fractions may also be used as constants, and very large or very small numbers may be entered in 'E' format. For instance, 1.2 E6 is equivalent to 1.2×10^6 or 1,200,000, and 2E-3 is equivalent to 2×10^{-3} or 0.002.

Within the parentheses of statement 30 are the operations ↑ and +. Since there is no way of writing X^2 on

```
10 LET X = 5
20 LET Y = 10
30 LET H1 = SQR(X↑2+Y↑2)
40 PRINT "THE HYPOTENUSE IS",H1
50 END



RUN
THE HYPOTENUSE IS                    11.1803

READY
```

Fig. 2. *A complete BASIC program and computer solution. This program finds the hypotenuse of a right triangle whose sides are 5 and 10.*

the teletypewriter the symbol ↑ was chosen to signify exponentiation. Thus $X ↑ 2$ means X^2. The + sign, of course, signifies addition. The arithmetic operators in BASIC are:

* *multiplication*	+ *addition*
/ *division*	− *subtraction.*

The letters SQR in statement 30 identify one of the ten standard functions in BASIC, the square root function. The other functions available to the programmer are:

SIN (X)	*Sine of X*
COS (X)	*Cosine of X*
TAN (X)	*Tangent of X*
ATN (X)	*Arc tangent of X*
EXP (X)	e^X
LOG (X)	*Natural logarithm of X*
ABS (X)	*Absolute value of X*
INT (X)	*Integer part of X*
SGN (X)	*Sign of X.*

The arguments of these functions may be variables, constants, or a combination of variables, constants, and operators.

Types of Statements

What types of statements are allowed in BASIC? Actually, there are not very many; this is what makes BASIC easy to learn. In the example program there are three: LET, PRINT, and END.

The LET Statement: The form of this statement is:
    LET <variable> = <expression>.

When a LET statement is executed the variable on the left side of the equals sign is assigned the value of the expression on the right side. In the sample program LET statements are used to assign the values 5 and 10 to X and Y; then H1 is assigned the value $\sqrt{125}$.

4

```
10  READ X,Y
20  PRINT "THE HYPOTENUSE IS", SQR(X↑2+Y↑2)
30  GO TO 10
40  DATA 1,1,5,10,3,4
50  END


RUN
THE HYPOTENUSE IS                1.41421
THE HYPOTENUSE IS                11.1803
THE HYPOTENUSE IS                5

ERROR 56 IN LINE  10
```

Fig. 3. *The hypotenuse program modified to compute the hypotenuses of three right triangles having sides (1, 1), (5, 10), and (3, 4). The error message indicates that the program ran out of data.*

The PRINT Statement: This statement causes the system to print output from the program on the teletypewriter. Its general form is:

$$PRINT \left\langle \begin{array}{c} expression \\ or \\ message \end{array} \right\rangle, \left\langle \begin{array}{c} expression \\ or \\ message \end{array} \right\rangle \cdots$$

To print a message we simply enclose the message within quotation marks in the PRINT statement. We may print the value of any variable, constant, or formula by including it in the PRINT statement. Thus, in executing the example program, the system will first print the message, THE HYPOTENUSE IS, and then the value of the variable H1. The word PRINT followed by no list of parameters causes the teletype to space one line upon execution.

The END Statement: Every BASIC program must have an END statement and it must have the largest statement number of any in the program.

Running the Program

To run the sample program we first get the system's attention, using whatever procedure is appropriate to the particular system. When the system is ready to receive inputs, it types READY. We then type in the program, as listed above. After the END statement, we type RUN. The results are shown in Fig. 2.

More Statements

Let us now change the example program slightly to demonstrate some new statements.

```
10  READ X,Y
20  PRINT "THE HYPOTENUSE IS", SQR(X↑2+Y↑2)
30  GO TO 10
40  DATA 1,1,5,10,3,4
50  END
```

The GO TO Statement: This statement alters the normal sequential execution of program statements and transfers control to a specified statement number. The general form is:

GO TO <statement number>.

In the example, the GO TO statement starts the program over again by transferring control to line 10 after the answer is printed.

READ and DATA Statements: The general form of these statements is:

READ <variable list>
DATA <number list>.

Whenever a READ statement is used in a program there must be at least one DATA statement in the program. When the first READ statement is executed, the first number in the lowest numbered DATA statement is assigned to the first variable in the READ list, the second number to the second variable, and so on until the READ variable list is satisfied. Subsequent READ statements will begin reading data where the previous READ left off. In the example, the program would execute statements 10 through 30 three times, using X and Y values of (1,1), (5,10), and (3,4). On the fourth attempt to read data, no more data would be found and the program would halt; the system would print a diagnostic message to indicate that the program ran out of data.

The RESTORE Statement: This statement, which isn't illustrated in the example program, is used whenever it is necessary to read the same data more than once in a program. When the statement

RESTORE

appears in a program, a subsequent READ statement will cause the computer to begin reading the data all over again starting with the first number of the first DATA statement.

Fig. 3 shows the results of running the second example program.

Inputs, Branches, and Loops

The INPUT Statement: This statement is used when the programmer wants to input numbers into his program from the keyboard as the program executes. The general form is:

INPUT <variable, variable, ...>.

Let us use this statement to write a program that inputs initial and final values for a table of numbers. The pro-

5

```
10 PRINT "INPUT INITIAL AND FINAL VALUES"
20 INPUT I,F
30 IF I<0 THEN 10
40 FOR J = I TO F
50 PRINT J,J↑2,SQR(J)
60 NEXT J
70 GO TO 10
80 END


RUN
INPUT INITIAL AND FINAL VALUES
?1,5
 1              1              1
 2              4              1.41421
 3              9              1.73205
 4              16             2
 5              25             2.23607
INPUT INITIAL AND FINAL VALUES
?
```

Fig. 4. *This program asks the operator for initial and final values of a sequence of integers, then prints the integers and their squares and square roots.*

```
10   DIM A[3,3],B[3,3],C[3,1],X[3,1]
20   MAT READ A
30   DATA 1,-2,-2
40   DATA 1,1,1
50   DATA 1,1,-6
60   MAT READ C
70   DATA -15,117,40
80   MAT B=INV(A)
90   MAT X=B*C
100  MAT PRINT X
110  END


RUN
 73

 33

 11.


READY
```

Fig. 5. *Matrix operations are a powerful feature of BASIC. Here some of them are used to solve a set of simultaneous equations (see text).*

gram will check that the initial value is non-negative and then print the numbers, their squares, and their square roots.

```
10 PRINT "INPUT INITIAL AND FINAL VALUES"
20 INPUT I,F
30 IF I<0 THEN 10
40 FOR J = I TO F
50 PRINT J,J↑2,SQR(J)
60 NEXT J
70 GO TO 10
80 END
```

When statement 20 of this program is executed the teletypewriter will type a question mark, indicating that input is to be typed. We respond with two numbers separated by a comma, then press the RETURN key. The program then continues.

The IF-THEN Statement: This statement tests for equality or inequality between two expressions and transfers control depending on whether the test is true or false. The general form is:

IF <*expression*> *relational operator* <*expression*>
    THEN <*statement number*>.

The relational operators allowed are:

<	less than
>	greater than
<=	less than or equal
>=	greater than or equal
=	equal
<>	not equal (some systems use #)

If the test is true, control will transfer to the statement number following the THEN. If the test is false, control transfers to the next sequential statement.

Statement 30 of the example tests whether I is nega-tive; if it is, the program asks for new values by jumping back to statement 10.

The FOR and NEXT Statements: Often in a program it is necessary to loop through a group of statements several times to perform a calculation or a printout. The FOR statement does this with a minimum of programming effort. Upon entry to the FOR loop in statement 40 of the example, the variable J is set to the value I. Statement 50 is executed, and then the NEXT J statement at 60 directs control back to the beginning of the loop at statement 40. J is now incremented by 1 and a check is made to see if J is greater than F; if it is, control is transferred to the statement following 60; if it is not, the loop is executed again. The cycle repeats until J becomes greater than F. The general form of the two statements is:

FOR <*variable*> = <*expression*> *TO* <*expression*>
    STEP <*expression*>
NEXT <*variable*>.

The values of the expressions for the initial and final values are computed once upon entry into the loop, as is the step size. If the step size is omitted, as it was in the example, it is assumed to be one.

Fig. 4 shows the results of running our third example program.

Matrix Operations

BASIC has the very powerful feature of having built-in functions to manipulate matrices. A matrix is defined in a BASIC program with a DIM statement. Thus, the statement

```
10   DIM A[3,3],B[10,10]
```

tells the BASIC System to allocate storage for a 3-by-3

matrix called A and a 10-by-10 matrix called B. The name of a matrix is always a single letter. Particular elements of a matrix may now be referenced by calling out the desired row and column of that element. For example, `20 LET A[2,1]=0` assigns the value 0 to the element of matrix A in the second row, first column.

The matrix functions defined in BASIC are listed in the table at right.

To demonstrate some of these statements, we will solve the following system of simultaneous equations. The equations:

$$X_1 - 2X_2 - 2X_3 = -15$$
$$X_1 + X_2 + X_3 = 117$$
$$X_1 + X_2 - 6X_3 = 40$$

MAT READ A	*Read numbers into matrix A row by row from a DATA statement*
MAT A = ZER	*Fill A with zeros*
MAT A = CON	*Fill A with ones*
MAT A = IDN	*Set up A as an identity matrix*
MAT PRINT A	*Print A, row by row*
MAT B = A	*Set B equal to A*
MAT C = A + B	*Add matrices A and B*
MAT C = A − B	*Subtract matrix B from matrix A*
MAT C = A∗B	*Multiply matrix B times matrix A*
MAT C = TRN(A)	*Transpose matrix A*
MAT C = INV(A)	*Invert matrix A*
MAT C = (K)∗A	*Multiply matrix A by K, where K is any expression*

The program:

```
10   DIM A[3,3],B[3,3],C[3,1],X[3,1]
20   MAT   READ A
30   DATA 1,-2,-2
40   DATA 1,1,1
50   DATA 1,1,-6
60   MAT   READ C
70   DATA -15,117,40
80   MAT B=INV(A)
90   MAT X=B*C
100  MAT   PRINT X
110  END
```

Fig. 5 shows the results of running this program.

Miscellaneous Statements

The REM Statement: The REM statement is used to insert comments into programs, for explanation and future documentation. For example:

```
      •
      •
      •
10    LET Z=3
20    REM THIS IS A REMARK
30    PRINT Z
      •
      •
      •
```

The DEF Statement: The DEF statement is used to define functions in a program. The name of a function must be three letters, the first two of which must be FN. Thus, functions may be named FNA, FNB, etc. For example, if we want to do a cube root calculation several times in a program we could say:

```
10 DEF FNA(X) = X↑(1/3)
```

```
10    FOR I=1 TO 10
20    PRINT RND(0);
30    NEXT I
40    END


RUN
 1.52602E-05    .500092     .500412      1.64799E-03     6.17992E-03
 .522248      .577867     .266971     .901025      .503415
READY




20 PRINT INT(25*RND(0))+1;



RUN
 1       13      13      1       1       14      15      7       23      13
READY
```

Fig. 6. *The RND function generates pseudo-random numbers or integers.*

Later in the program this function could be called by writing:

```
        •
        •
        •
450 LET Z = FNA(3.7)*SQR(Y)
        •
        •
        •
575 LET T1 = FNA(I↑2)
```

The RND Function: The RND function will automatically generate random numbers in the range from 0 to 1. The form of this function requires an argument, although the argument has no significance. Thus, the program

```
10    FOR I=1 TO 10
20    PRINT RND(0);
30    NEXT I
40    END
```

will print the first 10 random numbers. Running the program twice will produce the same set of numbers; this is useful for debugging purposes.

As a further example, if we need ten random integers ranging from 1 to 25 we could change line 20 to:

```
20 PRINT INT(25*RND(0))+1;
```

Notice that we have ended the last two PRINT statements with semicolons (;). This changes the format of the printout. Had the semicolon been omitted, each random number would be printed on a separate line, whereas now the numbers will be printed several to a line, as shown in Fig. 6.

The GOSUB and RETURN Statements: The GOSUB and RETURN statements allow a part of a program that must be executed at several points in the overall program to be executed as a subroutine, so it doesn't have to be typed in several times. For example, suppose that the teletypewriter must be spaced three lines at three different points in a program. This can be done as follows:

```
        •
        •
        •
100  GOSUB 900
        •
        •
        •
320  GOSUB 900
        •
        •
        •
540  GOSUB 900
        •
        •
        •
900  PRINT
901  PRINT
902  PRINT
903  RETURN
1000 END
```

Each time the RETURN is executed, control is transferred to the statement following the GOSUB which last called the PRINT routine.

Control Commands

There are several commands that may be given to the computer by typing the command at the start of a new line (no line number) and following the command with a CARRIAGE RETURN.

STOP. Stops all operations at once, even when the teletypewriter is typing. When the system is ready to accept further input, it types READY.

RUN. Begins the computation of a program.

SCRATCH. Destroys the problem currently being worked on; it gives the user a 'clean sheet' to work on. When the system is ready to accept a new program, it types READY.

LIST. Causes an up-to-date listing of the program to be typed out.

LIST XXXX. Causes an up-to-date listing of the program to be typed out beginning at line number XXXX and continuing to the end.

That's All There Is To It

Programs written according to the rules given in this article will run on any system using the BASIC language. However, individual systems may have additional features (statement types, etc.) that will further simplify programming or perform special functions. This is true of the Hewlett-Packard version of BASIC. The next article explains why and how HP BASIC differs from basic BASIC. ⓗⓟ

Gerald L. Peterson

Developing an acquaintance with a broad expanse of California has occupied much of Jerry Peterson's leisure time since he arrived from Wisconsin in July 1967. Church activities, golf, and tennis help to fill the remaining free hours.

Currently product manager for the 2000A Time-Sharing System, Jerry came directly to HP from the University of Wisconsin (BSEE) to work in market development for the 2116 computer family. He is a member of IEEE and Eta Kappa Nu.

BASIC at Hewlett-Packard

*Previously available only on large time-sharing systems,
BASIC has been adapted by Hewlett-Packard programmers
for HP computers and instrumentation systems.*

By Richard M. Moley

HEWLETT-PACKARD ENGINEERS have long been enthusiastic users of computers as engineering design aids. Before time-sharing services became available, considerable use was made of the ALGOL and FORTRAN computer languages on computers operated as batch processors. The advent of time-sharing systems and the development of simpler conversational languages, of which BASIC is an excellent example, provided a great stimulus to the further use of computers. The growth in the use by HP engineers of commercially available time-sharing services, and particularly the use of the BASIC language, was phenomenal.

A related development at Hewlett-Packard was that of a family of 16-bit general-purpose computers particularly suited for the scientific and instrumentation fields (Fig. 1).[1] Not long after the announcement of the first of these computers, serious consideration was given to the development of a BASIC compiler to run on this family of machines to satisfy the following objectives:

- To provide a conversational compiler to supplement the FORTRAN and ALGOL compilers developed for the family.

- To provide a powerful, flexible, and convenient method of controlling complex computer instrumentation systems.

The word 'compiler' may be new to some readers. A compiler is a language-processing computer program which translates high-level language statements (ALGOL, FORTRAN, BASIC, or whatever) into a form which the computer can 'understand' and execute directly. This process is analogous to translating a text from a foreign language into English so that a person who knows only English can read and understand it.

HP BASIC

A project was undertaken in May 1967 to develop a single-terminal BASIC compiler to run on any HP computer having at least an 8K core memory and an ASR-33 teletypewriter. The design criteria called for the compiler
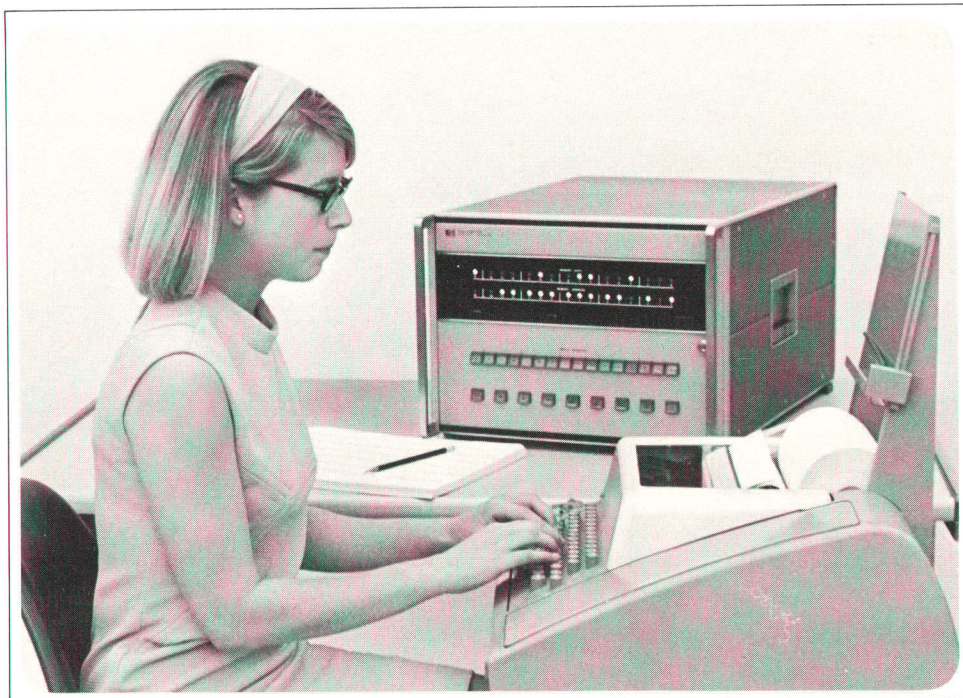

Fig. 1. *HP BASIC is a version of the BASIC language which has special commands for controlling instruments and for making better use of limited memory space. It can be used on any HP computer which has an 8K memory. Shown here is the HP 2114A Computer.*
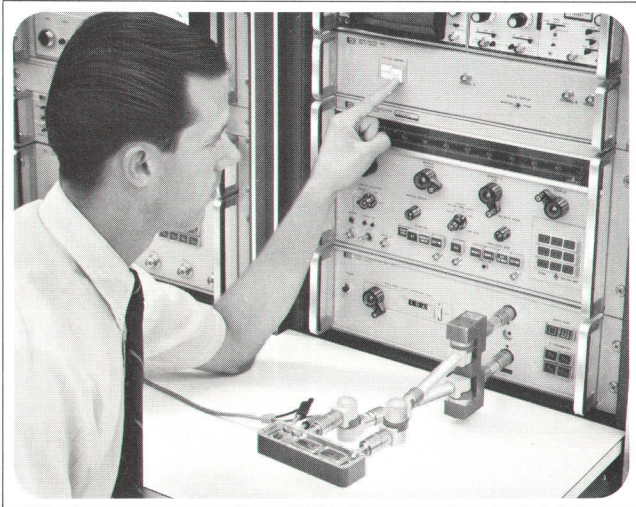
[1] K. B. Magleby, 'A Computer for Instrumentation Systems,' **Hewlett-Packard Journal,** March 1967. This article describes the HP 2116A Computer; the 2115A, 2114A, and 2116B Computers were developed later.

Fig. 2. *The thin-film amplifier in the foreground is about to have its s-parameters measured automatically by the HP 8541A Automatic Network Analyzer, a computer-controlled system programmed in BASIC or FORTRAN. Programs for performing many tests are supplied with the system. Special tests are programmed by the user.*

to occupy less than 6K of the core, leaving at least 2K for the user, and for the language to be compatible with commonly available time-sharing implementations of BASIC, so that programs developed on one system could be run on another.

To make the language useful in the instrumentation environment, it was necessary to expand it. Two statements were added to those described in the preceding article.

The WAIT statement: This statement extends BASIC to allow the introduction of delays into a program. Execution of

$WAIT\ (<formula>)$

causes the program to wait for the number of milliseconds specified by the value of the formula.

The CALL statement: This statement extends BASIC to allow the transfer of parameters to and from instrument-control subroutines. Such subroutines can be added to BASIC to create a specialized BASIC which has all the features of the standard BASIC, along with the capability to control instruments. Subroutines which have been appended to BASIC can be accessed through a statement of the form:

$CALL\ (<subroutine\ number>,\ <parameter\ list>)$

The subroutine number is a positive integer specifying the desired subroutine. The parameter list contains a number of parameters appropriate to the subroutine being called.

An Instrumentation-System Example

An example may help to clarify the use of the CALL and WAIT statements. Let us suppose we have an instrumentation system consisting of a programmable voltage source and a digital voltmeter (DVM). With these we wish to test the gain of a slowly settling amplifier. We connect the amplifier's input to the voltage source and its output to the DVM. The voltage source is controlled by the computer program by means of subroutine number 1, which has two parameters, voltage and current limit. The DVM is controlled by means of subroutine number 2, which has three parameters, the function and range settings for the DVM, and the voltage measured by the DVM. The test might proceed as follows:

```
10    INPUT V,I,R,F,G,T
20    CALL (1,V,I)
30    WAIT (T)
40    CALL (2,R,F,Y)
50    IF Y=V*G THEN 90
60    PRINT "GAIN CHECK FAILED ";
70    PRINT "INPUT="V;"OUTPUT="Y
80    GOTO 10
90    PRINT "GAIN CHECK PASSED"
100   GOTO 10
110   END
```

Statement 10 requests input values for the voltage (V) and current limit (I) for the voltage source, the range (R) and function (F) settings for the DVM, the specified gain (G) and the settling time (T) of the amplifier.

Statement 20 sets up the voltage source.

Statement 30 allows time for the amplifier to settle.

Statement 40 causes the DVM to read the voltage on the output of the amplifier and return its value as variable Y.

Statement 50 checks whether the amplifier output is equal to its input times the specified gain. If it is, statement 90 is executed and the message 'Gain Check Passed' is printed. Statement 100 then returns control to statement 10, and the system awaits the next input values. If the amplifier output is not equal to the input times the specified gain, statements 60 and 70 are executed; the message 'Gain Check Failed Input = XX Output = XX' is printed, with the actual values of V and Y inserted in the XX positions. Statement 80 then returns control to statement 10.

This program is greatly simplified, of course. It is intended only as an illustration of the use of the CALL and WAIT statements. In an actual system there would probably be a switching matrix, controlled by another subroutine, which would make it possible to apply the voltage source and the DVM to any point in the test fixture. The program would also be modified to accept amplifier outputs within a specified tolerance of the ideal.

AND, OR, and NOT

The requirement for an error tolerance is common in instrumentation systems. Tests for combinations of conditions are also common. These can be accomplished in BASIC by a series of IF statements. However, to make it simpler to program tolerances and combined tests, the logical operations AND, OR, and NOT were added to the HP version of BASIC. These operators have the same meaning as they do in a Boolean expression. Going back to our example program, a tolerance band of 10% around the ideal output could be introduced by changing statement 50 to read

```
50  IF (Y<(1.1*V*G)) AND (Y>(.9*V*G)) THEN 80
```

so that the amplifier gain will be considered acceptable if $0.9VG < Y < 1.1VG$.

Interprogram Transfers

In BASIC each program, including its subroutines and data, is treated as a complete entity. It is not possible to split a program into separate segments, each of which first processes the data partially and then passes the partially processed data on to another segment. This can be a crippling restriction on a system which has limited memory available for the user's program (for example, an HP 8K system, which has 2K available for the user). To overcome this restriction, another statement was added to HP BASIC.

The COM Statement: The COM statement makes it possible for one program to store information in memory for retrieval by a subsequent program. The form of the statement is the same as that of the DIM statement described in the preceding article. Thus the common-area information is accessible only as a matrix. If the COM statement is present in a program, it must be the first entered and the lowest numbered statement of the program.

Using the COM statement, it is possible to run a program which, although too large to fit into 2K of memory when written as a complete program, is of such a nature that it can be split into separate segments which communicate through the COM statement. Such a problem might be encountered in an instrumentation system. The

```
            FORTRAN                                        BASIC
    PROGRAM EXAMP                          1    COM D[10],C[53,3]
    COMMON  D(10),CAL(3,125)               10   PRINT "CONN DEVICE",
10  WRITE (2,11)                           20   CALL (7,A1)
11      FORMAT (//"CONN DEVICE←")          30   PRINT "FREQ","VSWR"
    NDMY = NYES1(NDMY))                    40   PRINT
    WRITE (2,12)                           50   LET M=1
12      FORMAT (//"  FREQ",5X,"VSWR"//)    60   GOSUB 9800
    M = 1                                  70   IF F<0 THEN 10
20  CALL CALF3(2,M,F)         Find the     80   GOSUB 9900
    IF (F) 10,30             Mth Frequency 90   CALL (16,R,V1,A)
30      CALL  CORM1(50,M,X,Y)    Find      100  LET V2=(1+V1)/(1-V1)
        T1 = SQRT(X*X+Y*Y)    Complex      110  PRINT F,V2
        VSWR = (1.0+T1)/(1.0-T1) Reflection 120  LET M=M+1
        WRITE (2,31) F,VSWR   Coefficient  130  GOTO 50
31          FORMAT ( F6.1, F10.3 )  Convert
        M = M + 1                 to       9800 REM: FIND F GIVEN M
        GO TO 20               VSWR        9810 IF M<1 THEN 9850
    END                    Print and       9820 IF M>D[3] THEN 9850
                          Go to Next       9830 LET F=D[1]+M*D[2]
                          Frequency        9840 RETURN
                                           9850 LET F=-1
                                           9860 RETURN

                                           9900 REM:  FIND CORR REFL AT FREQ F
                                           9910 CALL (6,F,X,Y)
                                           9920 CALL (14,R,X,Y)
                                           9930 CALL (11,R,C[M,1],R)
                                           9940 CALL (12,R,C[M,3],T1)
                                           9950 CALL (10,C[M,2],T1,T1)
                                           9960 CALL (13,R,T1,R)
                                           9970 RETURN
                                           9999 END
```

Fig. 3. *FORTRAN and HP BASIC listings of the same program. This program directs the HP 8541A Automatic Network Analyzer to measure and list the corrected VSWR of the device being tested.*

first program segment might acquire data and convert the data to engineering units while a second segment analyzes the converted data.

To use the COM statement, a program must be organized into separate segments, each of which starts with a similar COM statement. The first such program segment is entered into the computer and executed. Upon completion, this program segment will have stored the variables of interest in the common area. Subsequent program segments can then be entered into the computer, each one retrieving information left in the common area by a previous segment. A two-segment problem requiring one hundred common variables might be organized as follows:

Segment Number 1	Segment Number 2
10 COM A[100]	10 COM A[100]
•	•
•	•
•	•
100 LET A[I]=X↑2+Y↑2	300 MAT PRINT A
•	•
•	•
•	•
500 END	400 END

Ignored or Deleted Lines

A powerful feature of BASIC is that it is conversational; it is always listening to the teletypewriter and will interpret anything typed in as either a system command or a program statement. If it is neither of these a diagnostic message is printed. This can be annoying if the programmer merely wants to type something on the sheet for his own information but not for the computer's. For example, the programmer might want to type his name, the date, or the serial number of a device under test. To make this possible, a further extension was made to the normal operating characteristics of BASIC. HP BASIC will ignore any message terminated by pressing the ESC or ALT key on the teletypewriter. When this key is pressed BASIC responds with a '\', CARRIAGE RETURN, LINE FEED to indicate that the line has been ignored. This feature is also useful for suppressing an erroneous program statement when the programmer realizes an error has been made and wants to avoid the diagnostic message.

An Interpretive Compiler

When the development of HP BASIC started, the only conversational BASIC systems were those provided by the commercial time-sharing systems. These generally use very large computers, with large fast-access core

memories, large bulk memories in the form of disc or drum, and sophisticated instruction repertoires and memory-protection schemes. Such systems typically store the program statements on a disc as they are entered. Then, when the complete program has been entered and the RUN command given, they compile the program into a machine-executable form. If any errors are detected in the form of the statements (syntax errors) during the compilation process, the compilation is terminated and an error diagnostic is printed on the teletypewriter. The user must then correct the errors and attempt to run again.

In these large systems there are two versions of the user's program: one is the original, stored on the disc, and the other is the translation, which is executed in the computer's main core memory. The original must be retained for editing and listing purposes.

To keep a logical error in a user's program from destroying the BASIC system during a program's execution, the large computer's memory-protection scheme automatically terminates the user's program should it try to destroy the system.

On a small computer with restricted core memory size and no bulk memory (remember that the design criteria called for the HP system to occupy no more than 6K on an 8K computer), it is not possible to store both the orig-

Fig. 4. *The HP 9500A is a computer-controlled system designed to test a variety of electronic components, modules, and assemblies, over a frequency range of dc–500 MHz. It provides stimuli, measures responses, processes data, and records results, all automatically. It is programmed in HP BASIC.*

12

inal and translated versions of the user's program. Therefore, for HP BASIC, an *interpretive* technique was adopted. The computer stores only the original copy of the program (or a version very close to it), and executes one statement at a time by calling subroutines which perform the operations requested by the statements. This process is analogous to that of a foreign-language interpreter who translates sentence-by-sentence for an English-speaking person. This technique is slower than the conventional technique, but for many problems it is not markedly so. Another advantage of the interpretive technique is that the interpreter can preserve the integrity of the system by making sure before performing each operation that the operation will not destroy the system. This eliminates the need for a hardware memory-protection feature.

Deleting Spaces Saves Space

To make its use of core memory more efficient, the HP BASIC system deletes the spaces from incoming statements and converts the condensed statements into a compact form, shorter than the original and convenient for interpretation. When the programmer calls for a listing of his program, the original program is reconstructed from the compact form and made readable by the insertion of spaces at appropriate places.

The HP system was made more convenient for the user by having it check each statement for syntax errors as it is entered, instead of waiting for the entire program to be entered, as do many large-scale systems. Errors are diagnosed as soon as a statement is entered, so the user can correct them immediately.

To take advantage of the high-speed photoreaders and tape punches which exist in many small computer installations, special commands were included in HP BASIC to enable a program to be read in via a photoreader or to be listed on a high-speed tape punch. Using tape, large programs can be entered and punched at speeds up to 30 times that of the teletypewriter.

Uses of HP BASIC

The single-terminal HP BASIC system has been available since March 1968 and is being used extensively both within HP and by purchasers of our computers. It provides an attractive alternative to the use of a time-shared terminal for those who have access to a computer which is not entirely dedicated to another function. It is being used within HP in the control of experiments, in the incoming inspection of parts, and in the testing of instruments. The HP 8541A Computerized Network Analyzer

Richard M. Moley

A 1961 graduate of the University of Manchester with a B.Sc. (Honors) in electrical engineering, Dick Moley designed computers and developed process control software systems before joining HP in 1966. At HP he served as project manager for a data acquisition executive program and the HP BASIC system prior to his appointment as manager of general purpose systems programming. Dick, a member of IEEE and ACM, received his MSEE from Stanford this year.

Dick enjoys golf, bridge, reading, and sea travel. A veteran of three transatlantic crossings by ship, he hopes someday to spend a long vacation cruising from San Francisco to Southampton.

(Fig. 2 and 3) is programmed in BASIC, as are several HP automatic test systems (Fig. 4).

One of the most exciting outgrowths of the BASIC project has been its extension into a special-purpose time-sharing system which services up to sixteen active users.[2] This system has an executive control program, a communication control program, and program library capabilities. Its version of BASIC doesn't have CALL and WAIT statements because there are no instruments to be controlled, and it doesn't have the COM statement because there is much more memory available for the user; however, it has other features, not found in standard BASIC, that are particularly useful for scientific computation.

Acknowledgments

[2] T. C. Poulter, Jr., 'A Practical Time-Shared Computer System,' **Hewlett-Packard Journal,** July 1968.

How to Correct for Errors in High-Frequency Oscilloscope Measurements

Knowing the parameters of various input configurations, it is possible to plot curves to determine error with frequency.

By Wayne A. Kohl

CHARACTERISTICS OF DISPLAYS ON OSCILLOSCOPES are related to source impedance, oscilloscope input impedance and system bandwidth. Limited bandwidth of the measurement system results in a reduction of amplitude of high-frequency signals. Also, the input capacitance of the oscilloscope and the source resistance of the circuit under test form an RC low-pass filter, Fig. 1(a). A frequency-dependent load on the circuit under test affects the circuit performance and introduces an error which increases with frequency. These affect both pulse and CW measurements.

CW measurement errors introduced may be represented by a term called 'transfer ratio' where unity transfer ratio (infinite input impedance and bandwidth) results in no error. Transfer ratio of the scope input is defined as:

$$TR = \frac{V_{in}}{V_g} = \frac{1}{\sqrt{1 + (2\pi f R_g C_{in})^2}} \qquad (1)$$

where V_{in} is the voltage at the input terminals of the oscilloscope, V_g is the source voltage, f is frequency, R_g is the resistance of the source and C_{in} is the input capacitance of the oscilloscope system. R_{in} is assumed to be large and does not affect the calculation.

Since all oscilloscopes and CRT systems have finite input impedance, the transfer ratio is always less than 1. How much less depends upon the magnitude of the impedance and frequency. Transfer ratio versus frequency may be plotted, to determine the error resulting from various source and input impedance combinations.

Fig. 2 is the transfer ratio plot for the scope input of Fig. 1(a) with typical values. Fig. 1(b) is a scope input
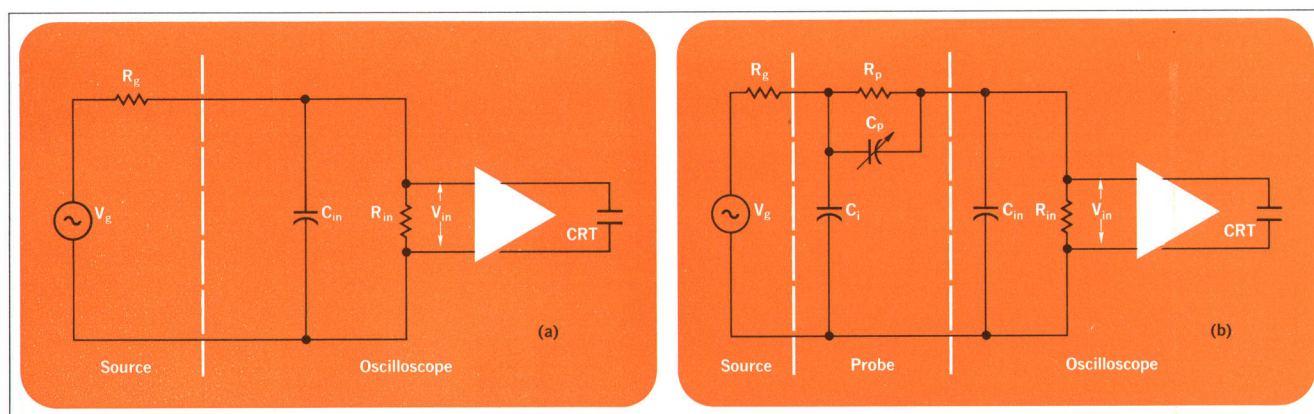
Fig. 1. *Typical oscilloscope input (a) is a low pass filter whose characteristics depend upon source resistance and input capacitance. Addition of a probe (b) enables input capacitance to be reduced and source resistance to be increased thus improving high-frequency performance of the typical input.*

Fig. 2. *Curves of transfer ratio for a typical oscilloscope with 1 meg input impedance and 25 pF input capacitance compared to a 10 meg, 10 pF passive probe input. With the probe, sensitivity is decreased by a factor of 10, but measurement error is substantially reduced at high frequencies.*
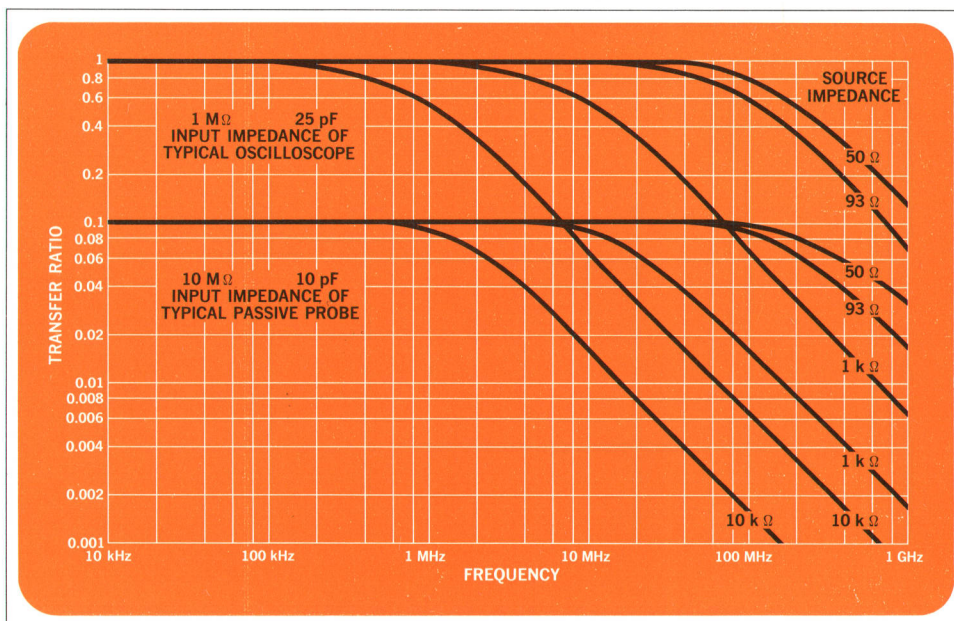
with a passive probe attached. These curves of transfer ratio versus frequency indicate less error when a high impedance probe is used, but sensitivity is reduced by a factor of 10.

Transfer ratio curves may be used to correct inaccuracies at high frequencies for various source impedances. For example, a measurement taken with a 1 megohm, 25 pF probe at 10 MHz from a 10 kilohm source impedance would read only 6.3% of the actual unloaded circuit voltage.

Resistive Dividers

The passive probe used with a typical scope input, Fig. 1(b), requires that C_{in} be compensated by C_p. If the resistance R_{in} is low and the effect due to C_{in} can be neglected, a simple resistive divider may be used as a probe.

DC Case. Fig. 3 shows the equivalent circuit of a resistive divider from which this expression for the dc division may be derived:

$$\frac{V_{in}}{V_g} = \frac{R_{in}}{R_{in} + R_d + R_g}. \qquad (2)$$

Division is dependent upon source resistance and is correct for only one source resistance. Error may be introduced at other source resistances, but this variation provides a method for determining R_g in situations where it might otherwise be difficult to measure.

Division from divider tip to scope input is unaffected by source resistance, but the voltage at the probe tip is

Wayne A. Kohl

Wayne Kohl graduated from the University of Colorado in 1962 with a BSEE. Prior to joining HP in 1965, he worked on aerospace guidance and control systems. At HP he has worked on the development of the Model 180A Oscilloscope and the Model 1801A Vertical Amplifier. He was project leader on the 1802A Dual Channel 100 MHz Vertical Amplifier plug-in. Wayne is a member of Tau Beta Pi and Eta Kappa Nu.
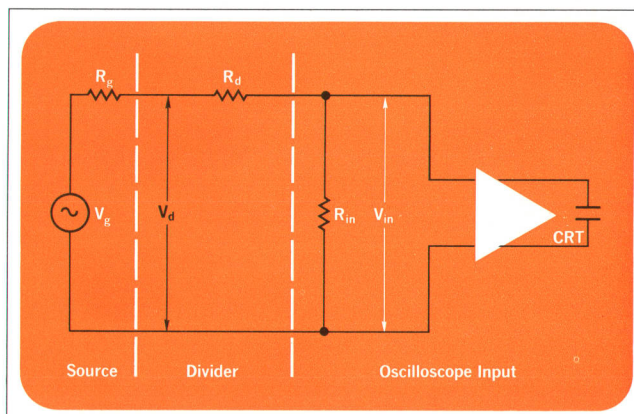
Fig. 3. *Division ratio in a typical resistive divider probe is determined by R_d, R_g and R_{in}.*

15
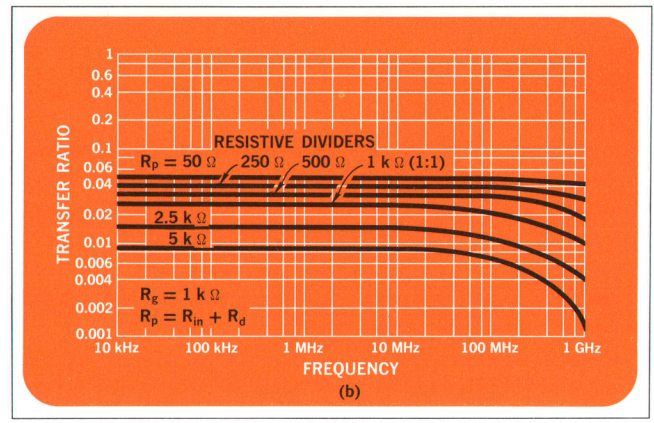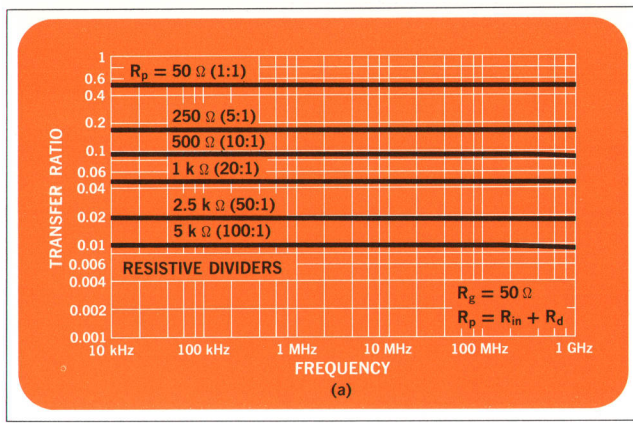
Fig. 4. *Plots for a resistive divider probe with 0.7 pF input capacitance operating into a 50 ohm termination. Source impedances are 50 ohms (a) and 1k ohms (b).*

dependent on source resistance. From the expression for dc division (Eq. 2) with two measurements using different dividers, it is possible to calculate source resistance R_g. After the source resistance is determined, it is then possible to calculate open circuit or unloaded source voltage V_g by:

$$V_g = \frac{V_{in}(R_{in} + R_d + R_g)}{R_{in}}. \qquad (3)$$

This dc example of the technique can be extended to pulse or CW work in determining impedance at a specific frequency or as an aid in identifying complex quantities. **High-Frequency Case.** Fig. 4 shows plots for various divider probes with 0.7 pF tip capacitance operating into a 50 ohm resistive termination. The transfer ratio is more constant than the example of Fig. 2, indicating less error variation with frequency.

Active Probes

Active probes have lower input capacitance without sacrificing sensitivity, but at reduced dynamic range. Transfer ratio of an active probe on a scope input, Fig. 5, is given by:

$$TR = \frac{V_{in}}{V_g} = \frac{\dfrac{R_p}{R_p + R_g}}{\sqrt{1 + \left(2\pi f \dfrac{R_p R_g}{R_p + R_g} C_i\right)^2}}. \qquad (4)$$

The plot of transfer ratio, Fig. 6, shows an improvement in accuracy over the passive probe plot of Fig. 2.

Pulse Measurement Problems

Although this discussion has been primarily concerned with CW measurements, pulse measurements likewise are affected by transfer ratios. Attenuation of high frequency components of a pulse are generally interpreted as a loss of rise time. Thus, the maximum rise time associated with a circuit with bandwidth limited by R and C is expressed as $t_r = 2.2\ RC$. Each of the examples discussed here has a rise-time limit resulting from the generator source resistance and circuit input capacitance and requires the user to determine rise time by:

$$t_r = \sqrt{(t_r)^2_{\text{input}} + (t_r)^2_{\text{gen}}}. \qquad (5)$$

Acknowledgment

I would like to thank Ralph Reiser for his assistance in preparing this material. 🔷

Fig. 5. *Equivalent circuit of a typical active probe.*
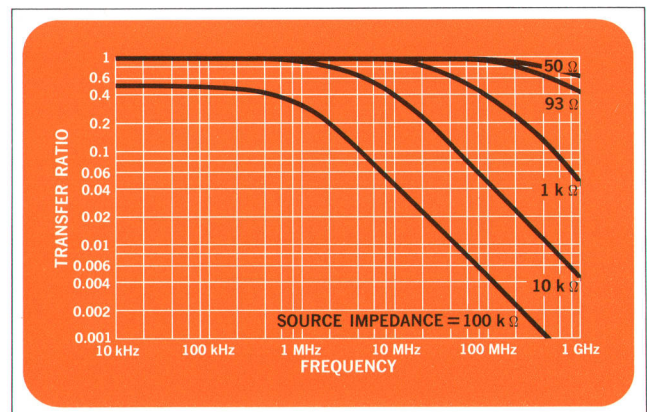
Fig. 6. *Transfer ratio plots of an active probe with an input impedance of 100 k ohms and 3.5 pF input capacitance.*

16

Extending Precision Oscilloscope Measurements into the High Frequencies

Growing requirements for more accurate high-frequency measurements and real time display of fast rise time phenomena demand very sophisticated oscilloscope systems.

The new HP Model 1802A Dual Channel Vertical Amplifier, Fig. 1, extends the usefulness of the HP Model 180A Oscilloscope System [1] [2] to 100 MHz. Pulse rise time of the 1802A is less than 3.5 ns. The plug-in has the standard capability of displaying either or both inputs, or the algebraic sum of the inputs plus the capability to cascade channels for increased display sensitivity. Horizontal sweep may be triggered by either or both signals.

Sync Amplifier

A significant difference between conventional sync amplifiers and the 1802A sync amplifiers is the latter's ability to trigger from a signal other than that being displayed by using one channel for triggering and the other for display. This uses the extra gain of the amplifier to give better sensitivity than conventional external triggering.

Flexibility in triggering is enhanced by the ability to choose these modes with a front panel switch rather than changing connections. This is possible because the sync amplifiers are connected in parallel with the vertical amplifiers at their inputs, Fig. 2.

For the same reason it is possible to use either sync channel as a wideband (100 MHz) ×10 amplifier. The amplified front-panel output may be used to synchronize or drive other equipment, or may be fed back into the other channel for increased display sensitivity up to 75 MHz. A separate front panel attenuator is provided to calibrate the front panel output.

Input Circuit

Both the π-type fixed attenuators and the Bridged-T variable attenuators have been carefully designed to maintain a constant 50 ohm impedance when properly
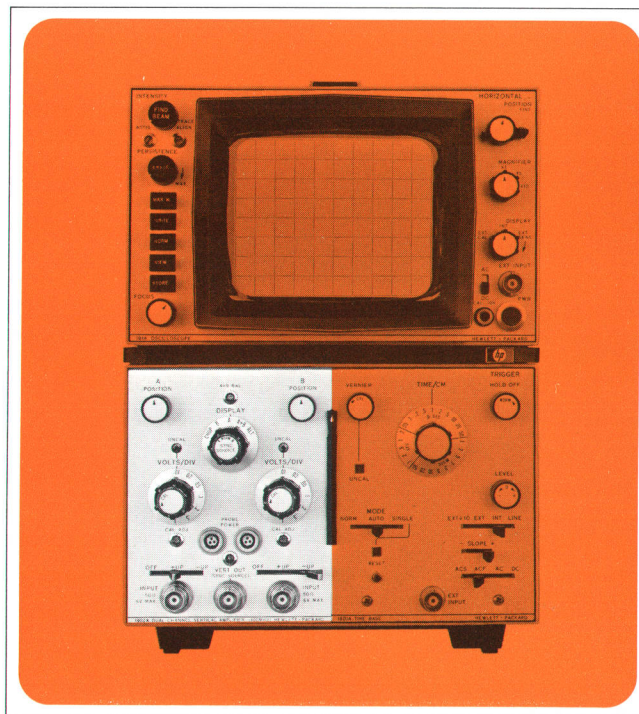

Fig. 1. *The new HP Model 1802A Dual Channel Vertical Amplifier plug-in extends the range of the Model 180 Oscilloscope System to 100 MHz.*
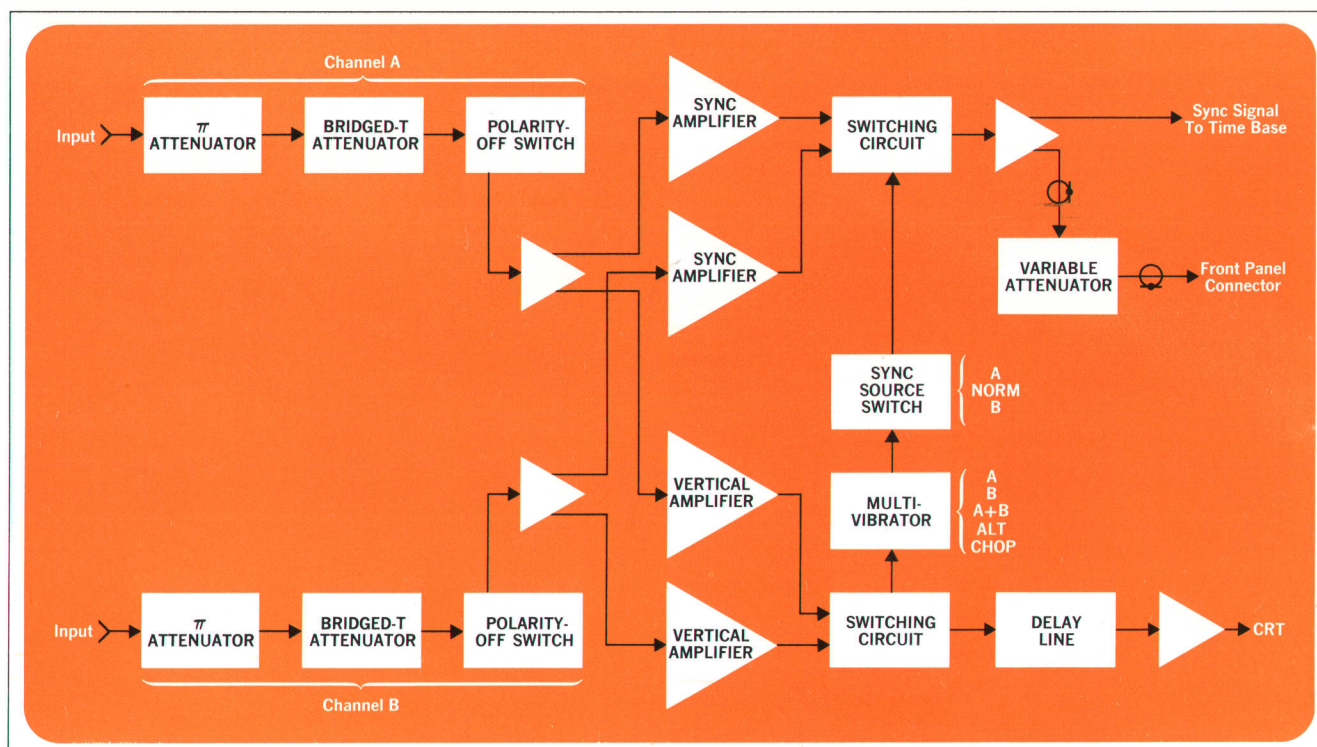
Fig. 2. *In the Model 1802A, switching is arranged to provide maximum flexibility in use. The separate front-panel attenuator allows adjusting the front panel output amplitude, either for external use or for calibrating the 1802A for cascade operation.*

terminated, and to provide a uniform attenuation over the wide frequency range. Mechanical design and careful location of components of the input system has resulted in the elimination of capacitive effects as a major source of error.

In the article on page 14, the effects of capacitance on CW measurement errors and pulse rise-time degradation are discussed. Loading due to large values of capacitance with the 1802A 50 Ω input may be neglected.

The 1802A input system has been designed to minimize reflections in a 50 Ω transmission system and eliminate disturbance to a system under test. Switches are mechanically and electrically symmetrical to provide constant terminating impedance for both sides of the input amplifier and the circuit under test.

For other than 50 Ω impedance levels, use of probes such as the Model 1123A or the new Model 10020A Miniature Resistive Dividers will provide a wide range of input impedances which can be selected to suit individual requirements.

DC Drift

Constant impedance at the amplifier inputs eliminates dc shifts with polarity switching, because any currents at the amplifier inputs develop equal voltages regardless of polarity. Leakage currents back into the low impedance termination result in very low offset voltages and low dc drift. Thus, there is no necessity for channel balance or

attenuator balance adjustments. The only balance control needed maintains the 0 volt dc reference level centered on the CRT in the $A+B$ mode.

Low dc drift is an advantage when measuring low-repetition rate, high-frequency signals with the variable persistence Model 181A Oscilloscope.[2] Visible displays are built up by integrating from trace to trace. Good stability assures that subsequent traces will be precisely superimposed over previous traces.

Good dc stability also enables setting up a system in advance of a one-shot transient and being certain that the display will appear positioned as preset. The sync amplifier front panel output can provide a triggering signal to other equipment.

Acknowledgments

I would like to express my thanks to all of the people who contributed to the design of the 1802A, and especially to Don Watson, group leader, Carl Glitzke for the product design, Bill Mordan and Johan Sverdrup for circuit contributions, and Art Porter and Bill Ford for their technical assistance. 🔆

References

[1] Floyd G. Siegel, 'A New DC–50 MHz Oscilloscope,' **'Hewlett-Packard Journal,'** Vol. 17, No. 12, Aug., 1966.

[2] Charles A. Donaldson, Charles A. Gustafson, 'Easier and Brighter Display of HF Signals,' **'Hewlett-Packard Journal,'** Vol. 19, No. 9, pp. 10–15, May 1968.

Voltage Probe for High-Frequency Measurements

By Eddie A. Evel

WHEN MEASURING HIGH-FREQUENCY SIGNALS, accuracy can be adversely affected by circuit loading caused by the measuring instrument. Since loading may change the characteristics of the circuit under test or the waveform of the signal, it is often necessary to use a high-impedance, low-capacitance input device. The new HP Model 1123A Voltage Probe, Fig. 1, is a wide-bandwidth, active probe designed for making accurate high-frequency measurements.

Low Input Capacitance

Using active circuitry in the probe tip, Fig. 2, the probe input capacitance is approximately 3.5 pF. Because of the low capacitance, it has a high input impedance at high frequencies. Thus it allows accurate measurement of high frequency signals having a high source impedance.

50 Ω Output Impedance

Output impedance of the probe is 50 ohms, making it useful for driving a variety of instruments. A matched output impedance is necessary to maintain good pulse fidelity over a variety of load conditions. If the probe is used to drive a pulse into improperly terminated 50 Ω coax, the reflection will be absorbed by the probe and not reflected again. Therefore the reflection will not appear at the coax output. With the 50 ohm output impedance, long lengths of cable may be used to connect the probe to the instrument as a remote signal input, while still preserving good signal fidelity.

DC Output Level Control

An external screwdriver adjustment adjusts dc offset at the probe output over a ±0.5 volt range. The output dc level may be adjusted to —0.5 volts for example, in which case the dynamic range at the probe input is shifted to 0 to +1 volt. Thus by setting the offset control to the appropriate level, a 0 to +1 volt or 0 to —1 volt signal can be observed with the probe even though the dynamic range is restricted to ±0.5 volts at the probe output.

DC Stabilization Circuit

By utilizing a unique dc stabilization circuit, the dc vs. temperature drift common in most active probes is almost eliminated. In addition the circuit requires no adjustment to optimize temperature drift performance.

The stabilization circuit (Fig. 3) consists of amplifiers 1 and 2 connected such that the input signal being fed through R1 is compared to the output signal being fed back through R4. If the input voltage is different than the output voltage, amplifier 2 generates a correction signal
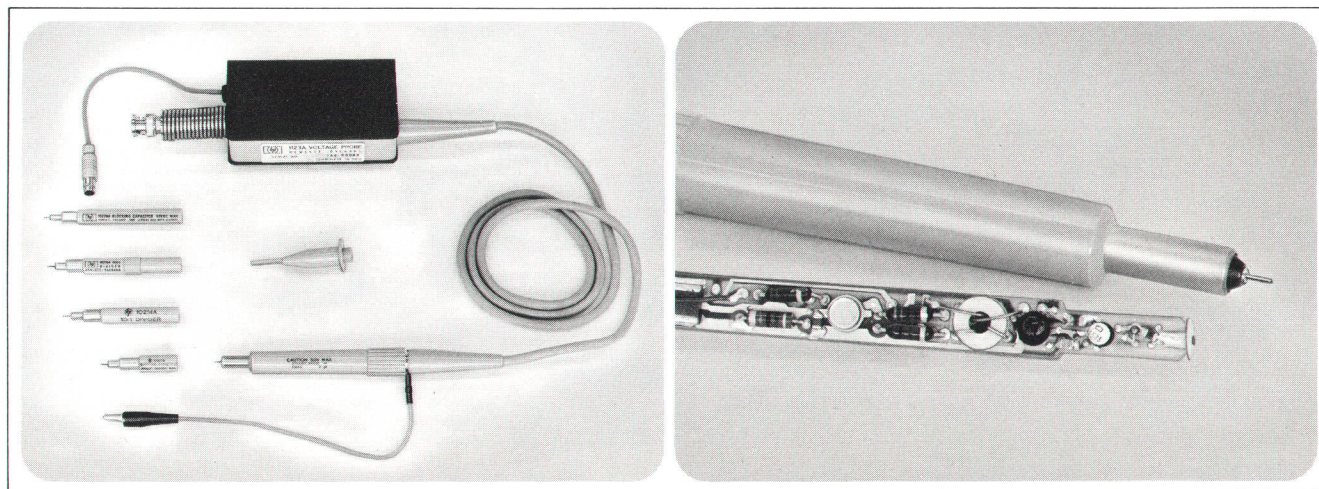

Fig. 1. *Various tips (left) may be used with the HP Model 1123A Voltage Probe to increase dynamic range or ac couple the probe. The spring mounting of the amplifier assembly reduces the possibility of connector damage.* **Fig. 2.** *By mounting a FET in the probe tip and using miniature components, the input capacitance is kept low.*

which is fed to amplifier 4. Therefore, dc temperature drift is determined by amplifiers 1 and 2 only. At high frequencies, R5 and C1 cut off the output of amplifier 2 and the input signal is amplified by 3 and 4 only.

This circuit permits the probe to be designed with no active circuitry in the probe tip that will affect dc temperature drift. Further, amplifiers 3 and 4 can be optimized for high frequency performance without regard for dc drift performance.

Mechanical Features

The amplifier assembly is spring mounted, Fig. 1, by its output BNC connector to keep it conveniently out of the way while providing stress relief for the BNC connector to which it is mounted. The probe tip pin is designed to be easily replaceable.
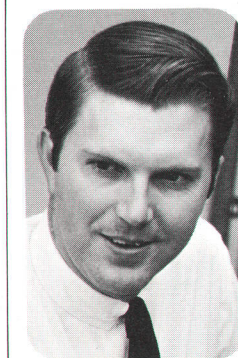
Accessories

Accessories which can be used with the 1123A Voltage Probe include a 10:1 Divider Tip and a 100:1 Divider Tip. These slip over the tip of the probe to increase dynamic range. Also included are two blocking capacitors to allow observation of signals which have a large dc component. A hook tip may be used which facilitates attachment of the probe to a circuit component.

Acknowledgments

The assistance and guidance of our group leader, Don Watson, and the product design done by Dan Paxton are gratefully acknowledged. I would also like to thank Al DeVilbiss for his helpful suggestions and Bob Beamer for his assistance in the electrical design, and all others who contributed to the Model 1123A Project.

Fig. 3. *This dc stabilization circuit provides excellent dc vs. temperature drift performance. The system uses a feedback amplifier arrangement.*

Eddie A. Evel

Ed Evel is a graduate of Kansas State University where he received his BSEE in 1962. After graduation he worked on missile guidance systems. Since joining HP in 1965, he has worked on the design of the Models 191A and 193A TV Waveform Oscilloscopes. Ed was responsible for the electrical design of the Model 1123A Voltage Probe.
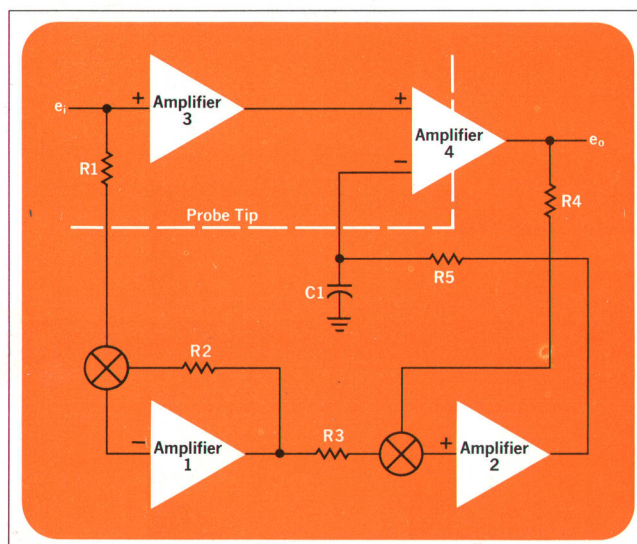
1969 UTC Offset Announced

The International Bureau of Time, Paris, has announced that the fractional frequency offset for Coordinated Universal Time for 1969 will continue to be −300 parts in 10^{10}. This offset from the atomic time (and frequency) scale is annually selected so that Coordinated Universal Time (UTC) can approximate UT2, a time scale related to the rotation of the earth.

For the past three years the offset has also been −300 parts in 10^{10}. Before that it was −150 parts in 10^{10} and, earlier still, −130 parts in 10^{10}.

HEWLETT-PACKARD JOURNAL *hp* **NOVEMBER 1968** *Volume 20 · Number 3*

TECHNICAL INFORMATION FROM THE LABORATORIES OF THE HEWLETT-PACKARD COMPANY   PUBLISHED AT 1501 PAGE MILL ROAD, PALO ALTO, CALIFORNIA 94304
Editorial Staff: F. J. BURKHARD, R. P. DOLAN, L. D. SHERGALIS, R. H. SNYDER   Art Director: R. A. ERICKSON

HP Archive

This vintage Hewlett-Packard document was
preserved and distributed by

www.hparchive.com

Please visit us on the web!