Personal Calculator Algorithms IV: Logarithmic Functions

A detailed description of the algorithms used in Hewlett-Packard hand-held calculators to compute logarithms.

by William E. Egbert

BEGINNING WITH THE HP-35,[1,2] all HP personal calculators have used essentially the same algorithms for computing complex mathematical functions in their BCD (binary-coded decimal) microprocessors. While improvements have been made in newer calculators,[3] the changes have affected primarily special cases and not the fundamental algorithms.

This article is the fourth in a series that examines these algorithms and their implementation.[4,5,6] Each article presents in detail the methods used to implement a common mathematical function. For simplicity, rigorous proofs are not given and special cases other than those of particular interest are omitted.

Although tailored for efficiency within the environment of a special-purpose BCD microprocessor, the basic mathematical equations and the techniques used to transform and implement them are applicable to a wide range of computing problems and devices.

The Logarithmic Function Algorithm

This article will discuss the method of generating the ln(x) and $\log_{10}(x)$ functions. To minimize program length, a single function, ln(x), is always computed first. Once ln(x) is calculated, $\log_{10}(x)$ is found by the formula

$$\log_{10}(x) = \frac{\ln(x)}{\ln(10)} .$$

ln(x) is generated using an approximation process much the same as the one used to compute trigonometric functions.[5] The fundamental equation used in this case is the logarithmic property that

$$\ln(a_1 \cdot a_2 \cdot a_3 \cdot \ldots \cdot a_n) = \ln(a_1) + \ln(a_2) + \ln(a_3) + \ldots + \ln(a_n) \quad (1)$$

This algorithm simply transforms the input number x into a product of several terms whose logarithms are known. The sum of the logarithms of these various partial-product terms forms ln(x).

Exponent

Numbers in HP calculators are stored in scientific notation in the form $x = M \cdot 10^K$. M is a number whose magnitude is between 1.00 and 9.999999999 and K is an integer between −99 and +99. Using equation 1, it is easy to see that

$$\ln(M \cdot 10^K) = \ln(M) + \ln(10^K)$$

At this point, another logarithmic property becomes useful, which is

$$\ln(A^b) = b \cdot \ln(A).$$

Using this relationship

$$\ln(M \cdot 10^K) = \ln(M) + K \cdot \ln(10).$$

Thus to find the logarithm of a number in scientific notation, one calculates the logarithm of the mantissa of the number and adds that to the exponent times ln(10).

Mantissa

The problem of finding ln(x) is now reduced to finding the logarithm of its mantissa M.

Let P = 1/M. Then

$$\begin{aligned} \ln(PM) &= \ln(P) + \ln(M) \\ \ln(1) &= \ln(P) + \ln(M) \\ 0 &= \ln(P) + \ln(M) \\ -\ln(P) &= \ln(M) \end{aligned} \quad (2)$$

This may appear to be a useless exercise since at first glance −ln(P) seems to be as hard to compute as ln(M).

Suppose, however, that a new number P_n is formed by multiplying P by r which is a small number close to 1.

$$P_n = P \cdot r$$

In addition, let P_n be defined as a product of powers of numbers a_j whose natural logarithms are known.

$$P_n = a_0^{K_0} \cdot a_1^{K_1} \cdot \ldots \cdot a_j^{K_j} \cdot \ldots \cdot a_n^{K_n}$$

Thus

$$P = P_n/r$$

$$\ln(P) = \ln(P_n) - \ln(r)$$

Using equation 2

$$\ln(M) = \ln(r) - \ln(P_n)$$

Finally

$$\ln(M) = \ln(r) - (K_0\ln(a_0) + K_1\ln(a_1) + \ldots + K_j\ln(a_j) + \ldots + K_n\ln(a_n))$$

Thus to find ln(M) one simply multiplies M by the carefully selected numbers a_j so that the product MP_n is forced to approach 1. If all the logarithms of a_j are added up along the way to form $\ln(P_n)$ then ln(M) is the logarithm of the remainder r minus this sum. Notice that the remainder r is nothing more than the final product MP_n.

Implementation

How is this algorithm implemented in a special-purpose microprocessor? First of all, the terms of P_n were chosen to reduce computation time and minimize the amount of ROM (read-only memory) needed to store a_j and its logarithm. The numbers chosen for the a_j terms are of the form $a_j = (1 + 10^{-j})$, where j = 0-4 (see Table 1).

Table 1 Values of a_j Terms

j	a_j	ln a_j
0	2	0.6931
1	1.1	0.09531
2	1.01	0.009950
3	1.001	0.0009995
4	1.0001	0.000099995

To achieve high accuracy using relatively few a_j terms, an approximation is used when $r = MP_n$ approaches 1. For numbers close to 1, $\ln(r) \approx r-1$. This yields

$$\ln M \approx (r-1) - \sum_{j=0}^{n} K_j\ln(a_j) \qquad (3)$$

Since all of the a_j terms are larger than 1, M must be between 0 and 1 if the product P_nM is to approach 1. As M is defined to be between 1 and 10, a new quantity A is formed by dividing M by 10. A is now in the proper range $(0.1 \leq A < 1)$ so that using the a_j terms as defined will cause the product AP_n to approach 1 without exceeding 1.

The product P_n can now be formally defined as a series, where j goes from 0 to n. Each partial product AP_j has the form

$$A \cdot P_j = A \cdot P_{j-1}(1+10^{-j})^{K_j}, j = 0, 1, 2, \ldots, n$$

$P_{-1} = 1$, and K_j is the largest integer such that $P_j < 1$.

In practice, each $A \cdot P_j$ is formed by multiplying $A \cdot P_{j-1}$ by $(1 + 10^{-j})$, K_j times. There is one intermediate product, T_i, for each count of K_j, as shown below.

$$T_0 = A(1 + 10^{-0})^1$$

$$T_1 = A(1 + 10^{-0})^2$$

$$T_{K_0} = A(1 + 10^{-0})^{K_0}$$

$$T_{K_0+1} = A(1 + 10^{-0})^{K_0}(1 + 10^{-1})^1$$

$$T_m = A(1 + 10^{-0})^{K_0}(1 + 10^{-1})^{K_1} \ldots (1 + 10^{-n})^{K_n} = AP_n$$

$$m = K_0 + K_1 + \ldots + K_n$$

$$T_i = T_{i-1}(1 + 10^{-j}) \text{ for some } j \qquad (4)$$

Notice that each multiplication of the intermediate product T_{i-1} by a_j simply amounts to shifting T_{i-1} right the number of digits denoted by the current value of j and adding the shifted value to the original T_{i-1}. This very efficient multiplication method is similar to the pseudo-multiplication of the trigonometric algorithm.[5]

An Example

A numeric example to illustrate this process is now in order. Let A = 0.155. To compute ln(A), A must be multiplied by factors of a_j until AP_n approaches 1. To begin the process A = 0.155 is multiplied by $a_0 = 2$ to form the intermediate product $T_0 = 0.31$. Another multiplication by a_0 gives $T_1 = 0.62$. A third multiplication by 2 results in 1.24, which is larger than 1. Thus $K_0 = 2$ and $AP_0 = 0.62$. The process is continued in Table 2.

Table 2 Generation of ln(0.155)

j	a_j	AP_j	K_j	T_i	$\ln(a_j)$
−1		0.155		0.155	
0	2		1	0.31	0.6931
0	2	0.62	2	0.62	0.6931
					*
1	1.1		1	0.682	0.0953
1	1.1		2	0.7502	0.0953
1	1.1		3	0.82522	0.0953
1	1.1		4	0.9077	0.0953
1	1.1	0.9985	5	0.9985	0.0953
2	1.01	0.9985	0		**
3	1.001	0.9995	1	0.9995	0.00099
4	1.0001	0.9996	1	0.9996	0.00009

$0.9996 = A \cdot P_4 = r$ $\qquad 1.8638 = \sum \ln(a_j)$

*Another ×2 would result in $AP_3 > 1$. Thus a_j is changed to 1.1.
**The 1.01 constant is skipped entirely.

Applying the values found in Table 2 to equation 3 results in

$$\begin{aligned}\ln(0.155) &= (0.9996 - 1) - 1.8638 \\ &= -1.8642\end{aligned}$$

This answer approximates very closely the correct 10-digit answer of −1.864330162.

This example demonstrates the simplicity of this method of logarithm generation. All that is required is a multiplication (shift and add) and a test for 1. To implement this process using only three working registers, a pseudo-quotient similar to the one generated in the trigonometric algorithm is formed.[5] Each digit represents the number of successful multiplications by a particular a_j. For the preceding example, the pseudo-quotient would be

2	5	0	1	1
↑	↑	↑	↑	↑
j = 0	j = 1	j = 2	j = 3	j = 4

With $-\ln(r) = (r - 1)$ as the first term, the appropriate logarithms of (a_j) are then summed according to the count in the pseudo-quotient digit corresponding to the proper a_j. The final sum is $-\ln(A)$.

At this point one more transformation is needed to optimize this algorithm perfectly to the microprocessor's capabilities. Recall that the factors a_j were chosen to force the product P_nA towards 1. Suppose $B_i = T_i - 1$. Forcing B_m towards 0 causes P_nA to be forced to 1. Substituting B_i into (4) and simplifying yields

$$(B_i + 1) = (B_{i-1} + 1)(1 + 10^{-j}) \text{ for some } j$$

$$B_i + 1 = B_{i-1}(1 + 10^{-j}) + 1 + 10^{-j}$$

$$B_i = B_{i-1}(1 + 10^{-j}) + 10^{-j}$$

Multiplying through by −1 results in the following equation, which is equivalent to equation 4.

$$-B_i = -B_{i-1}(1 + 10^{-j}) - 10^{-j} \text{ for some } j \qquad (5)$$

This expression is now in a very useful form, since the a_j term is the same as before, but the zero test is performed automatically when the 10^{-j} subtraction is done. A test for a borrow is all that is required. An additional benefit of this transformation is that accuracy can be increased by shifting $-B_i$ left one digit for each a_j term after it has been applied the maximum number of times possible. This increases accuracy by replacing zeros generated as B_i approaches zero with significant digits that otherwise would have been lost out of the right end of the register. This shifting, which is equivalent to a multiplication by 10^j, gives yet another benefit. Multiplying equation 5 by 10^j and simplifying,

$$-B_i \times 10^j = (-B_{i-1}(1 + 10^{-j}) - 10^{-j}) \times 10^j$$

$$-B_i \times 10^j = -B_{i-1} \times 10^j (1 + 10^{-j}) - 1 \text{ for some } j \qquad (6)$$

Notice that the 10^{-j} subtraction reduces to a simple −1 regardless of the value of j. The formation of the initial $-B_0$ is also easy since $-B_0 = -(A - 1) = 1 - A$. This is formed by taking the 10's complement of M (the original mantissa), creating 10 − M. A right shift divides this by 10 to give $1 - M/10 = 1 - A = -B_0$. A final, almost incredible, benefit of the B_i transformation is that the final remainder $-B_m \times 10^j$ is in the exact form required to be the first term of the summation process of equation 4 without further modification. The correct $\ln(a_j)$ constants are added directly to $-B_m \times 10^j$, shifting the sum right one digit after each pseudo-quotient digit to preserve accuracy and restore the proper normalized form disrupted by equation 6. The result is $-\ln(A)$.

Finally, the required ln(M) is easily found by subtracting the computed result $-\ln(A)$ from ln(10).

$$\begin{aligned} \ln(10) - (-\ln(A)) &= \ln(10) + \ln(M/10) \\ &= \ln(10 \cdot M/10) \\ &= \ln(M) \end{aligned}$$

Once ln(M) is computed, $K \cdot \ln(10)$ is added as previously discussed to form ln(x). At this point log(x) can be generated by dividing ln(x) by ln(10).

Summary

In summary, the computation of logarithmic functions proceeds as follows:

1. Find the logarithm of 10^K using $K \cdot \ln(10)$.
2. Transform the input mantissa to the proper form required by $-B_0$.
3. Apply equation 6 repeatedly and form a pseudo-quotient representing the number of successful multiplications by each a_j.
4. Form $-\ln(A)$ by summing the $\ln(P_j)$ constants corresponding to the pseudo-quotient digits with the remainder $-B_m \times 10^j$ as the first term in the series.
5. Find ln(x) or log(x) using simple arithmetic operations.
6. Round and display the answer.

The calculator is now ready for another operation.

References

1. T.M. Whitney, F. Rodé, and C.C. Tung, "The 'Powerful Pocketful': An Electronic Calculator Challenges the Slide Rule," Hewlett-Packard Journal, June 1972.
2. D.S. Cochran, "Algorithms and Accuracy in the HP-35," Hewlett-Packard Journal, June 1972.
3. D.W. Harms, "The New Accuracy: Making $2^3 = 8$," Hewlett-Packard Journal, November 1976.
4. W.E. Egbert, "Personal Calculator Algorithms I: Square Roots," Hewlett-Packard Journal, May 1977.
5. W.E. Egbert, "Personal Calculator Algorithms II: Trigonometric Functions," Hewlett-Packard Journal, June 1977.
6. W.E. Egbert, "Personal Calculator Algorithms III: Inverse Trigonometric Functions," Hewlett-Packard Journal, November 1977.

William E. Egbert

Bill Egbert is a project manager at HP's Corvallis, Oregon Division. He produced this series of algorithm articles as part of his work on the HP-67 and HP-97 Programmable Calculators. He was project leader for the HP-67 and did microprogramming for both calculators. More recently, he was project leader for the firmware development of the HP-19C and the HP-29C. Bill received his BSEE degree from Brigham Young University in 1973 and his MSEE from Stanford University in 1976. He's been with HP since 1973. Born in Fallon, Nevada, he's married, has two small children, and lives in Corvallis.