Tien Chi Chen

Automatic Computation of Exponentials, Logarithms, Ratios and Square Roots*

Abstract: It is shown how a relatively simple device can evaluate exponentials, logarithms, ratios and square roots for fraction arguments, employing only shifts, adds, high-speed table lookups, and bit counting. The scheme is based on the cotransformation of a number pair (x,y) such that the $F(x,y) = f(x_0)$ is invariant; when x is driven towards a known value x_ω, y is driven towards the result. For an N-bit fraction about $N/4$ iterations are required, each involving two or three adds; then a termination algorithm, based on an add and an abbreviated multiply, completes the process, for a total cost of about one conventional multiply time. Convergence, errors and simulation using APL are discussed.

Introduction

The present paper shows a hardware technique to evaluate we^x, $w + \ln x$, w/x and $w/x^{\frac{1}{2}}$, where w and x are given numbers. Each function is evaluated in about one conventional multiply time, using a unified apparatus based on adds, shifts, bit counting and the use of a table of $\ln (1 + 2^{-m})$. This technique is the subject of a recently issued patent [1].

The scheme is based on the iterative cotransformation of a number pair (x,y) such that a bivariate function $F(x,y)$ remains invariant. In doing so, x is directed towards x_ω, where the corresponding y_ω is the desired result. When x is sufficiently close to x_ω, a termination algorithm using an add and an abbreviated multiply completes the evaluation. The iteration aims to remove the leading one bit in the fraction $|x_k - x_\omega|$, and zero bits are automatically ignored.

Our divide algorithm resembles the IBM System/360 Model 91 scheme [2], which cotransforms the dividend and divisor by limited multiplications, until the divisor is close to unity. For $\ln x$ and e^x, the use of decimal digit-by-digit schemes dates back to Briggs three centuries ago [3]. Meggitt's unified study of pseudo division and pseudo multiplication methods [4] includes the mechanization of Brigg's scheme, square roots and trigonometric functions, each costing about three conventional multiplication times. A recent study by Sarkar and Krishnamurthy [5] showed that a further improvement in speed is possible.

Walther [6] recently showed a unifying algorithm containing the CORDIC schemes first proposed by Volder. The elementary functions treated include ratios and square roots; included also are exponentials and logarithms, as inferred from hyperbolic functions. The scheme is based on stepwise rotations in any of three coordinate systems (circular, linear, and hyperbolic) and usually employs shift sequences that are ascending integers. For hyperbolic functions a basic shift sequence needs to be invoked repeatedly.

The unified automatic evaluation of logarithms and exponentials of binary numbers has been considered in 1962 by Cantor, Estrin, and Turn [7], within the context of the "fixed-plus-variable" binary computer design. The method requires rather large tables (at least several hundred words for each function) to transform the argument into the vicinity of 1 and 0, respectively.

Specker [8] suggested the use of a table of $\ln(1 + 2^{-m})$ for the evaluation of $\ln x$ and e^x, and commented on the possible advantage of hardware implementation. The use of a given table entry here depends on the outcome of a comparison against $|x_k - x_\omega|$.

Very recently, de Lugish [9] discussed the unified mechanization for multiply, divide, $\ln x$, e^x, square root and trigonometric functions. His technique is based on the redundancy recoding technique in fast division algorithms, and involves the systematic one-bit left shift of x or $(1 - x)$ at every iteration. The shifted quantity, matched against two comparands, determines a value $s_k = -1$, 0 or $+1$; $(1 + s_k 2^{-k})$ is then used to transform

*A brief version of this work was presented at the IEEE Computer Society Symposium on Computer Arithmetic, University of Maryland, May 15–16, 1972.

(x_k, y_k). There is no termination algorithm. To guarantee convergence with the steady left shift of x, an initiation scheme is often invoked. The redundancy recoding does diminish the number of add-type operations (to about one for every three bit positions); and although one shift and two comparisons are required for every bit position, the de Lugish unified method can be quite advantageous when the add time is the overriding cost factor.

In contrast to the other techniques, the present method relies heavily on automatic bit-counting to avoid redundant actions. It also uses a terminal linear approximation which speeds up the computation and, incidentally, halves both the rounding errors and the table requirements.

Theory

Our algorithm to evaluate $z_0 = f(x_0)$ is based conceptually on the introduction of a parameter y to form a two-variable function $F(x,y)$ such that

1) There is a known starting value $y = y_0$ with $F(x_0,y_0) = z_0$.
2) There are convenient, known mechanisms to cotransform the number pair (x_k,y_k), $k \geq 0$, into (x_{k+1},y_{k+1}) leaving F invariant, namely $F(x_{k+1},y_{k+1}) = F(x_k,y_k)$.
3) The sequence of x-transformations tends to a known goal $x = x_\omega$; the corresponding y-transformations then tends to $y = y_\omega$. F is so defined that $F(x_\omega,y_\omega) = y_\omega$.
4) The iterative part of the algorithm is essentially the repeated application of the cotransformation procedure, carried out in the absence of detailed knowledge about the value z_0.

In terms of solid coordinate geometry, F is easily represented by a curve confined to the $z = z_0$ plane and passing through $P_0(x_0,y_0,z_0)$ as shown in Fig. 1. We have

$$F(x,y) = z_0 = f(x_0) \tag{1}$$

and the transformation invariance is merely the requirement that if the point $P_k(x_k,y_k,z_0)$ is on the curve F, then so is $P_{k+1}(x_{k+1},y_{k+1},z_0)$. Clearly, then,

$$f(x_0) = z_0 = F(x_0,y_0) = F(x_1,y_1) = \cdots = F(x_k,y_k). \tag{2}$$

Requirement 3) means that F passes through the point $Q(x_\omega,y_\omega,z_0)$. To ascertain this, we adopt the functional form

$$F(x,y) = yg(x) + h(x) \tag{3a}$$

with

$$g(x_\omega) = 1, \qquad h(x_\omega) = 0 \tag{3b}$$

then

$$F(x_\omega,y) = y. \tag{4a}$$

even if (x_ω,y,z_0) does not lie on the curve F in Eq. (1).

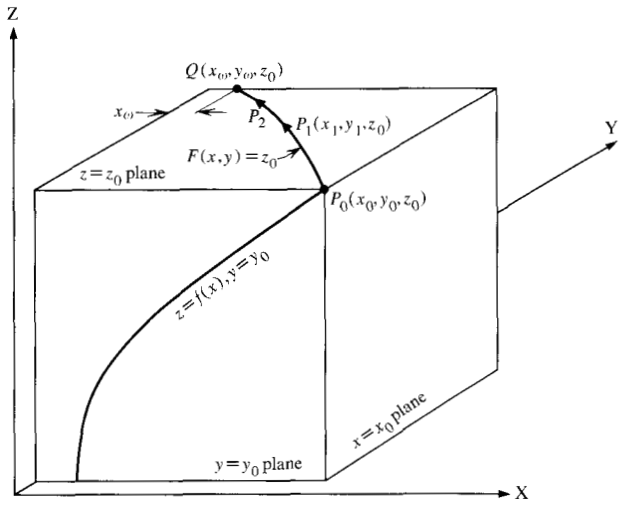

Figure 1 Evaluation of $z_0 = f(x_0)$.

The invariance, Eq. (2), then guarantees that $Q(x_\omega,y_\omega,z_0)$ indeed lies on F, and

$$z_0 = F(x_\omega,y_\omega) = y_\omega; \tag{4b}$$

and the iterative transformations of the number pair (x,y) in 2) is just a methodology to move a point along the curve F, starting from P_0, through P_1, P_2, etc., towards Q.

We shall limit the transformations to those based only on the current values of (x,y), thus

$$x_{k+1} = \phi(x_k,y_k) \tag{5a}$$

and

$$y_{k+1} = \psi(x_k,y_k)\,. \tag{5b}$$

The choice of ϕ is dictated mainly by cost and efficiency considerations. It is convenient to specify ϕ to be a function of x_k alone, operative for x_k being in some closed interval $[a,b]$; the x-transformation should yield an x_{k+1} much closer to x_ω than x_k is, and further should lie in the same interval, to facilitate further transformations.

Equation (5b) follows Eq. (5a), such that $P_{k+1}(x_{k+1}, y_{k+1},z_0)$ stays on the curve F. The function ψ can be derived from Eq. (1) and Eq. (3a). In order that P_k and P_{k+1} both lie on F, we have

$$y_{k+1}\, g(x_{k+1}) - y_k\, g(x_k) = -h(x_{k+1}) + h(x_k)$$

hence

$$y_{k+1} = \{y_k g(x_k) + h(x_k) - h(\phi(x_k,y_k))\}/g(\phi(x_k,y_k)) \tag{5c}$$

is a known function of (x_k,y_k), independent of the unknown z_0. The point P_{k+1} will be a step closer than P_k to the destination $Q(x_\omega,y_\omega,z_0)$.

It will be seen in the next section that the actual choices of ϕ and ψ are quite straightforward. In any case, starting from $(x,y) = (x_0,y_0)$, the repeated use of Eqs. (5a) and (5b) should show a steady displacement along the curve F towards the point Q:

$$\begin{aligned} z_0 &= f(x_0) = F(x_0,y_0) \\ &= F(x_1,y_1) = F(x_2,y_2) = \cdots = F(x_k,y_k) = \cdots \\ &= F(x_\omega,y_\omega) = y_\omega. \end{aligned} \tag{6}$$

if the x-process converges to x_ω. The corresponding y_ω should then equal z_0.

In reality, it may be difficult or impossible to reach x_ω exactly, and extrapolation by a Taylor series would be desirable, if $F(x,y)$ is differentiable with respect to x near x_ω. Thus,

$$\begin{aligned} F(x_\omega \pm \mu, y_n) &= F(x_\omega,y_n) \pm \mu\, \partial F/\partial x]_{x_\omega} + O(\mu^2) \\ &\sim y_n \pm \mu\, \partial F/\partial x]_{x_\omega}, \text{ using Eq. (4a).} \end{aligned} \tag{7}$$

For an N-bit fraction, if $|\mu| < 2^{-N}$, just the first term will usually be adequate.

On the other hand, both the iteration cost and roundoff error can be halved by including the term linear in μ, with $|\mu| \lesssim 2^{-N/2}$. Within this termination algorithm, the multiplication (if any) involves a half-precision number (μ) as one of the operands, costing half of a standard multiply time; high-performance multipliers elsewhere in the same machine can now be exploited.

In what follows, we shall assume that linear extrapolation will be used when $|\mu| < 2^{-\tilde{N}/2}$ with $\tilde{N} \sim N \gg 1$.

Specific cases

The functions being considered here are

we^x, $0 \le x < \ln 2 = 0.69314\cdots$, with $F = ye^x$, $x_\omega = 0$;

$w + \ln x$, $1/2 \le x < 1$, with $F = y + \ln x$, $x_\omega = 1$;

w/x for $1/2 \le x < 1$, with $F = y/x$, $x_\omega = 1$;

$w/x^{\frac{1}{2}}$ for $1/4 \le x < 1$, with $F = y/x^{\frac{1}{2}}$, $x_\omega = 1$.

Binary and hexadecimal floating-point arguments can easily be mapped onto the ranges given (the algorithms below actually work for $0 < x < 1$, but the convergence may be slower). We note that $x^{\frac{1}{2}}$ can be obtained by setting $w = x$ in the inverse square root case.

• *Exponential.* $(0 \le x < \ln 2)$

$$\begin{aligned} z_0 &= we^{x_0} = y_0 e^{x_0} \\ &= (y_0 a_0)\, e^{x_0 - \ln a_0} = y_1\, e^{x_1} = \cdots \\ &= y_n\, e^{\mu} \sim y_n + y_n \mu. \end{aligned}$$

Initiation: $y_0 = w$.

Function: $F(x,y) = ye^x$.

Transformations: $x_{k+1} = x_k - \ln a_k$, $y_{k+1} = y_k\, a_k$.

Termination: $x_n = \mu$; $z_0 \sim y_n + y_n\, x_n$.

Relative error: $0 \le \epsilon \le \mu^2/2(1 + 2\mu/3) < 2^{-\tilde{N}-1}$.

• *Logarithm.* $(1/2 \le x < 1)$

$$\begin{aligned} z_0 &= w + \ln x_0 = y_0 + \ln x_0 \\ &= (y_0 - \ln a_0) + \ln x_0 a_0 = y_1 + \ln x_1 = \cdots \\ &= y_n + \ln(1 - \mu) \sim y_n - \mu. \end{aligned}$$

Initiation: $y_0 = w$.

Function: $F(x,y) = y + \ln x$.

Transformations: $x_{k+1} = x_k a_k$, $y_{k+1} = y_k - \ln a_k$.

Termination: $1 - x_n = \mu$; $z_0 \sim y_n - (1 - x_n)$.

Absolute error: $0 \ge \delta \ge -\mu^2/2(1 - 2\mu/3) \sim -2^{-\tilde{N}-1}$

• *Ratio.* $(1/2 \le x < 1)$

$$\begin{aligned} z_0 &= w/x_0 = y_0/x_0 \\ &= (y_0 a_0)/(x_0 a_0) = y_1/x_1 = \cdots \\ &= y_n/(1 - \mu) \sim y_n + y_n \mu. \end{aligned}$$

Initiation: $y_0 = w$.

Function: $F(x,y) = y/x$.

Transformations: $x_{k+1} = x_k a_k$, $y_{k+1} = y_k a_k$.

Termination: $1 - x_n = \mu$; $z_0 \sim y_n + y_n(1 - x_n)$.

Relative error: $0 \le \epsilon = \mu^2/(1 - \mu^2) \sim 2^{-\tilde{N}}$.

• *Inverse square root.* $(1/4 \le x < 1)$

$$\begin{aligned} z_0 &= w/x_0^{\frac{1}{2}} = y_0/x_0^{\frac{1}{2}} \\ &= (y_0 a_0)/(x_0 a_0^2)^{\frac{1}{2}} = y_1/x_1^{\frac{1}{2}} = \cdots \\ &= y_n/(1 - \mu)^{\frac{1}{2}} \sim y_n + y_n \mu/2. \end{aligned}$$

Initiation: $y_0 = w$.

Function: $F(x,y) = y/x^{\frac{1}{2}}$.

Transformations: $x_{k+1} = x_k a_k^2$, $y_{k+1} = y_k a_k$.

Termination: $1 - x_n = \mu$; $z_0 \sim y_n + y_n(1 - x_n)/2$.

Relative error: $0 \le \epsilon = -1 + 1/(1 - \mu)^{\frac{1}{2}}(1 + \mu/2)$
$\le 3\mu^2/8 < 2^{-N-1}$.

Choice of a_k

We select a_k to be of the form $(1 + 2^{-m})$, such that multiplications by a_k can be replaced by a shift and an add.

Table 1 Summary of the algorithms.

Range of x_0	$f(x_0)$	$F(x_k,y_k)$	x_k	x_{k+1}	y_{k+1}	x_n	*Termination algorithm*
[0,ln 2)	we^{x_0}	$y_k e^{x_k}$	$2^{-m}+p$	$x_k-\ln(1+2^{-m})\sim p$	$y_k+2^{-m}y_k$	$0\le\mu<2^{-N/2}$	$y_n e^{\mu}\sim y_n+y_n\mu$
[1/2,1)	$w+\ln x_0$	$y_k+\ln x_k$	$1-(2^{-m}+p)$	$x_k+2^{-m}x_k\sim 1-p$	$y_k-\ln(1+2^{-m})$	$1-\mu$	$y_n+\ln(1-\mu)\sim y_n-\mu$
[1/2,1)	w/x	y_k/x_k	$1-(2^{-m}+p)$	$x_k+2^{-m}x_k\sim 1-p$	$y_k+2^{-m}y_k$	$1-\mu$	$y_n/(1-\mu)\sim y_n+y_n\mu$
[1/4,1)	$w/x^{\frac{1}{2}}$	$y_k/x_k^{\frac{1}{2}}$	$1-2(2^{-m}+p)$	$x_k(1+2^{-m})^2\sim 1-2p$	$y_k+2^{-m}yk$	$1-\mu$	$y_n/(1-\mu)^{\frac{1}{2}}\sim y_n+y_n\mu/2$

The value m is usually chosen as the position number* of the leading 1-bit in $|x_k-x_\omega|$; but an increase by 1 is needed for the inverse square root.

1) For we^x, m is the position number of the leading 1-bit in x_k. We have thus

$$x_k=2^{-m}+p\,,\qquad 0\le p<2^{-m}. \tag{8}$$

Here p represents the bit pattern after the mth bit. The latter, among all bits in x_k, is responsible for over half of the value of $|x_k-x_\omega|$, and is the leading candidate for removal. With the help of a small table $\{T_m=-\ln(1+2^{-m})\}$, we have

$$\begin{aligned}x_{k+1}&=(2^{-m}+p)-\ln(1+2^{-m})\\&=(2^{-m}+p)-(2^{-m}-2^{-2m}/2+2^{-3m}/3-\cdots)\\&=p+O(2^{-2m})\,,\end{aligned} \tag{9a}$$

where the most objectionable bit is replaced by a second-order disturbance, and p is largely intact. The y-transformation is done by a right shift of m places, and then by adding to the unshifted y_k

$$y_{k+1}=y_k+2^{-m}y_k\,. \tag{9b}$$

The iteration requires a table lookup, a shift and two adds. With m varying from 1 to $\tilde{N}/2$, one needs a total of $\tilde{N}/2$ tabular entries; even if $\tilde{N}=60$, the table requirement is only 30 words.

2) For $w+\ln x$, m is selected to be the position number of the leading 1-bit in $(1-x_k)$; here we have

$$x_k=1-(2^{-m}+p)\,,\qquad 0\le p<2^{-m} \tag{10}$$

and

$$\begin{aligned}x_{k+1}&=x_k+2^{-m}x_k\\&=1-p-2^{-m}(2^{-m}+p)\\&=1-p+O(2^{-2m})\,,\end{aligned} \tag{11a}$$

again the most objectionable bit is replaced by a second-order disturbance. Also,

$$y_{k+1}=y_k-\ln(1+2^{-m})\,, \tag{11b}$$

using the same tabular values in 1).

*The kth bit after the binary point is said to have position number equaling k.

3) For w/x, the x-transformation, hence the bit pattern preservation, is the same as in 2); the y-transformation is the same as in 1).

$$x_{k+1}=x_k+2^{-m}x_k \tag{12a}$$

$$y_{k+1}=y_k+2^{-m}y_k\,. \tag{12b}$$

4) For $w/x^{\frac{1}{2}}$, m is chosen as 1 + the position number of the leading 1-bit in $(1-x_k)$.

$$x_k=1-2(2^{-m}+p)\,,\qquad 0\le p<2^{-m} \tag{13}$$

then

$$t_k=x_k+2^{-m}x_k \tag{13a}$$

$$\begin{aligned}x_{k+1}&=t_k+2^{-m}t_k\\&=1-2p(1+2\cdot 2^{-m}+2^{-2m})-2^{-2m}(3+2\cdot 2^{-m})\\&=1-2[p+O(2^{-2m})]\,.\end{aligned} \tag{13b}$$

The y-transformation is the same as in 1), namely

$$y_{k+1}=y_k+2^{-m}y_k\,. \tag{13c}$$

The explicit algorithms are summarized in Table 1.

Convergence and accuracy

Our simple choice $a_k=1+2^{-m}$ thus transforms the most significant 1-bit in $|x_j-x_\omega|$ to zero, leaving the remaining bit pattern largely unaffected. The larger the value of m (i.e., the greater the position number of the most significant bit) the more negligible is the perturbation on p.

Thus, the number of iterations required to convert the leading $N/2$ bits of x to zero is measured by the number of 1 bits in $|x_0-x_\omega|$.

Statistically, half of these $N/2$ bits are zeros; hence, the average number of iterations is about $N/4$, and the average advance of m between two successive iterations is $\Delta m\sim 2$.

A convenient measure of convergence is the relative reduction of the distance to x_ω

$$r\equiv(x_{k+1}-x_\omega)/(x_k-x_\omega)\,. \tag{14}$$

The sequence $\{x_k\}$ converges if $|r|\le r_0<1$ for large k. Using the results of the previous section, it is easy to show that $0<r\le r_0<1$ and thus the sequence converges from the same side, with x_{k+1} lying in (x_k,x_ω).

This one-sided convergence is extremely desirable for the smooth applications of iterative transformations requiring $\{x_k\}$ to be suitably bounded. Here $\{x_k\}$ all lie in (x_0, x_ω), the latter being a valid interval for the x-transformations.

From the previous section one easily obtains

$$r = p/(2^{-m} + p) + O(2^{-m}) . \tag{15}$$

Substituting the bounds $0 \leq p < 2^{-m}$, and the average value $\langle p \rangle = 2^{-m-1}$, we obtain

$$1/2 + O(2^{-m}) > r \geq O(2^{-m}) , \tag{16}$$

and

$$\langle r \rangle \sim 1/3 . \tag{17}$$

A detailed analysis in the Appendix shows that $\langle r \rangle \sim 1 - \ln 2 = 0.307$.

The truncation errors have been given in specific cases in the third section. Should these be deemed too large, an extra iteration could be taken. Better still, as these errors bear a definite sign, one could reduce roughly half of the maximum error by adding one bit to μ. Thus,

$$y_n e^{\mu} \sim y_n + y_n(\mu + 2^{-N-2}) , \quad |\epsilon| \leq 2^{-N-2} \tag{18}$$

$$y_n + \ln(1-\mu) \sim y_n - (\mu + 2^{-\tilde{N}-2}) , \quad |\delta| \leq 2^{-\tilde{N}-2} \tag{19}$$

$$y_n/(1-\mu) \sim y_n + y_n(\mu + 2^{-\tilde{N}-1}) , \quad |\epsilon| \leq 2^{-N-1} \tag{20}$$

$$y_n/(1-\mu)^{\frac{1}{2}} \sim y_n + \tfrac{1}{2} y_n(\mu + 2^{-\tilde{N}-2}) , \quad |\epsilon| \leq 2^{-\tilde{N}-2} . \tag{21}$$

For $|\epsilon| \leq 2^{-N-1}$, $\tilde{N}$ never needs to exceed N, and $\tilde{N} = N$ will be assumed below.

A major source of inaccuracy is rounding error, which can be controlled by using longer intermediate results, rounded tabular values, rounded arithmetic, fewer arithmetic steps, or combinations of all of these. Using J guard bits in the fraction, the total absolute rounding error in computing y_n is

$$\rho \leq i(j + k)\, 2^{-N-J} , \tag{22}$$

where

i = the number of iterations,

$j \cdot 2^{-N-J}$ = the rounding error due to each tabular or shifted operand,

and

$k \cdot 2^{-N-J}$ = the rounding error due to an add (or subtract) operation.

We have

$j = 1/2$ if operands in question are all rounded,

$= 1$ if unrounded;

$k = 1/2$ if the add (subtract) operations include rounding,

$= 1$ if unrounded arithmetic is practiced.

The total absolute error in computing μ is the same as ρ in Eq. (22) except for the inverse square root case, where 2ρ is used because of the extra work required. Ignoring the rounding error generated by the termination algorithm, the total absolute rounding error is bounded by $\rho(1 + q\partial F/\partial x]_{x_\omega})$ with $q = 2$ for the inverse square root, and $q = 1$ otherwise.

It turns out that i, expected to be about $N/4$, is never known to exceed $N/2$, and the choice $J = 6$ should ensure $\rho \leq 2^{-(N+1)}$ for $N \leq 64$, using rounded operands and rounded operations, or for $N \leq 32$, using unrounded operands and unrounded operations.

Nonstandard choices for m

The standard choices of m for this paper have been given in the fourth section. They are closely related to the left-zeros count of the fraction $|x_k - x_\omega|$, thus

$$m = 1 + (\text{the left-zeros count of } x_k) \text{ for } we^x \tag{23}$$

$$m = 1 + [\text{the left-zeros count of } (1 - x_k)] \text{ for } w + \ln x , \; w/x \tag{24a}$$

$$m = 2 + [\text{the left-zeros count of } (1 - x_k)] \text{ for } w/x^{\frac{1}{2}} . \tag{25}$$

The counting of left-zeros being a regular activity in normalized floating-point arithmetic, the implementability of a "standard m-finder" is not an issue.

However, nonstandard integer choices, $\tilde{m}$, may be desirable for added efficiency and/or algorithm simplicity. We remark here that the choice of $\tilde{m} > m$ may increase the number of iterations and hence the rounding error, but the algorithm nevertheless still converges. Conversely, the choice of $\tilde{m} < m$ may speed up the convergence in some cases, yet may lead to divergence if carelessly applied. The standard choices on the other hand are simple and reliable, and are asymptotically optimum (for $m \to \infty$).

The convergence for the inverse square root algorithm is slow for x_0 below $4/9 = (0.111000\cdots)_2$; several iterations are often needed to make the leading fraction bit of x_k into a 1. A brief study reveals that for x_k in (1/4, 4/9) the optimum choice is not $m = 2$ (standard choice) but $\tilde{m} = 1$; with it x_{k+1} is guaranteed to be at least 0.56.

In Eqs. (24a) and (25) the standard m is formed from $(1 - x_k)$, calling for a complementation process. Alternatively, one could count the left-ones in x_k directly. Instead of Eq. (24a) we now have

$$m = q + (\text{the left-ones count of } x_k) \tag{24b}$$

with $q = 1$ if $p \neq 0$ and $q = 0$ otherwise. To simplify

controls, one could persist in using $q = 1$; even for the unlikely event of $p = 0$, where $q = 0$ would be an unusually good choice, with $q = 1$ rapid convergence is still guaranteed. A similar situation occurs for the inverse square root, with $(1 + q)$ in place of q.

A unified apparatus

The four iteration algorithms can be handled by one unified apparatus (see Fig. 2), consisting of an adder A, a shifting mechanism S, and an $N/2$-word fast memory M holding $\{-\ln(1 + 2^{-m})\}$, $1 \leq m \leq N/2$. A memory word has $N + J$ fraction bits. Internal computation is also done using $N + J$ fraction bits, plus one integer bit to handle the range $0 \leq y_k < 2$; this is so because y_k can exceed unity, but need not exceed 2 if w is properly prenormalized, say to within $[0,1]$. The apparatus is capable of the following elementary operations; these, of course, do not consume equal time:

a) Load $C(\mathrm{X})$ (the contents of register X), into T.
b) Deduce m from $C(\mathrm{T})$, store it in V.
c) Use m as address to fetch $C(m)$ from M; add $C(m)$ and $C(\mathrm{T})$.
d) Store the sum in X.
e) Load $C(\mathrm{Y})$ into T.
f) Shift $C(\mathrm{T})$ to the right by m places, put result into U; add $C(\mathrm{T})$ and $C(\mathrm{U})$.
g) Store the sum in Y.
h) Go to step a) if $m \leq \hat{m}$, else enter the termination algorithm.

Starting by putting x_0 in X , $w(= y_0)$ in Y, the sequencing for the evaluations are:

we^{x_0}: (abcdefgh) ; then $C(\mathrm{Y}) + C(\mathrm{Y}) \times C(\mathrm{X}) \sim we^{x_0}$.

$w + \ln x_0$: (abfdecgh) ; then $C(\mathrm{Y}) + 1 - C(\mathrm{X}) \sim w + \ln x_0$.

w/x_0: (abfdefgh) ; then $C(\mathrm{Y}) + C(\mathrm{Y}) \times [1 - C(\mathrm{X})] \sim w/x_0$.

$w/x_0^{\frac{1}{2}}$: (abfdafdefgh) ; then $C(\mathrm{Y}) + C(\mathrm{Y}) \times [1 - C(\mathrm{X})]/2 \sim w/x_0^{\frac{1}{2}}$.

The apparatus invokes the termination algorithm by shipping $C(\mathrm{X})$, $C(\mathrm{Y})$ away, to be processed by more conventional equipment. For details of the sequencing see Table 2. If self-contained operation is desired, the iterations can be allowed to run until $\mu \leq 2^{-N}$. It would then be necessary to double the table size and add one more guard bit for accuracy.

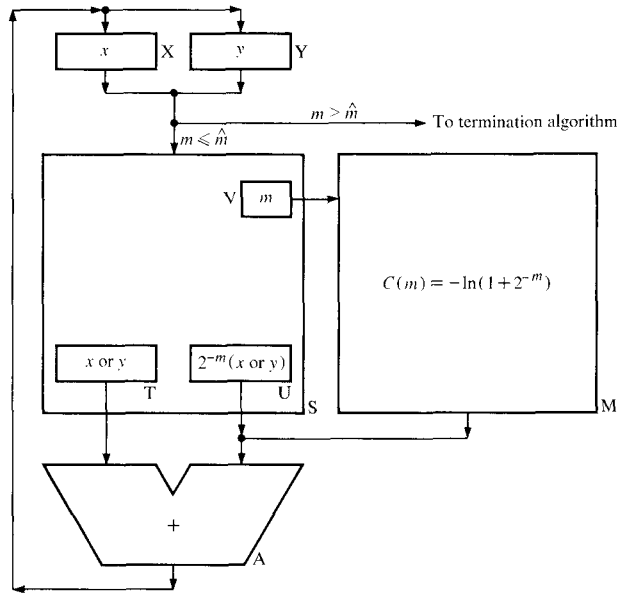

Figure 2 The unified apparatus.

The adder, the shifter, and the sequential bit-counter are within the state of art. A small high-speed memory is needed for we^x and $(w + \ln x)$; this is easily achieved using modern memory devices in an integrated circuit memory technology compatible with computation hardware. Also, this apparatus need not stand alone and can share equipment with other parts of the same system.

To derive a rough timing estimate we equate memory access time with shift time, and note that a conventional multiplier takes time $T = N$ shift-add times. The expected $N/4$ iterations involve $3N/4$ shift-adds for the inverse square root and $N/2$ for the other three functions; the time estimate is, therefore, $3T/4$ and $T/2$, respectively. To these one has to add the timing cost of the terminal algorithm.

For logarithms the complement-add required adds little to the iteration cost. The total timing cost is still measured by $T/2$.

For exponentials, ratios, and inverse square roots a half-multiply is needed. If this is done with a conventional multiplier, the cost would again be $T/2$ for a total cost of T (exponential, ratio) and $1.25T$ (inverse square root). However, if a powerful fast multiplier is invoked here, the half-multiply may cost little more time than an add.

APL simulation

The algorithms have been simulated by a nest of interlocking subroutines written in APL, which handles the bit-pattern manipulations with clarity and precision. The results of a run on the IBM APL terminal system is reported here.

Table 2 Sequencing operations.

	a) *For* we^x	b) *For* $w + \ln x$	c) *For* w/x	d) *For* $w/x^{\frac{1}{2}}$
PREP	$C(X) \leftarrow x_0$ $C(Y) \leftarrow w$	$C(X) \leftarrow x_0$ $C(Y) \leftarrow w$	$C(X) \leftarrow x_0$ $C(Y) \leftarrow w$	$C(X) \leftarrow x_0$ $C(Y) \leftarrow w$
XNOW	$C(T) \leftarrow C(X)$ $C(V) \leftarrow m$ computed from $C(T)$ $C(X) \leftarrow C(T) + C(m)$	$C(T) \leftarrow C(X)$ $C(V) \leftarrow m$ computed from $C(T)$ $C(U) \leftarrow 2^{-m} \times C(T)$ $C(X) \leftarrow C(T) + C(U)$	$C(T) \leftarrow C(X)$ $C(V) \leftarrow m$ computed from $C(T)$ $C(U) \leftarrow 2^{-m} \times C(T)$ $C(X) \leftarrow C(T) + C(U)$	$C(T) \leftarrow C(X)$ $C(V) \leftarrow m$ computed from $C(T)$ $C(U) \leftarrow 2^{-m} \times C(T)$ $C(X) \leftarrow C(T) + C(U)$ $C(T) \leftarrow C(X)$ $C(U) \leftarrow 2^{-m} \times C(T)$ $C(X) \leftarrow C(T) + C(U)$
YNOW	$C(T) \leftarrow C(Y)$ $C(U) \leftarrow 2^{-m} \times C(T)$ $C(Y) \leftarrow C(T) + C(U)$	$C(T) \leftarrow C(Y)$ $C(Y) \leftarrow C(T) + C(m)$	$C(T) \leftarrow C(Y)$ $C(U) \leftarrow 2^{-m} \times C(T)$ $C(Y) \leftarrow C(T) + C(U)$	$C(T) \leftarrow C(Y)$ $C(U) \leftarrow 2^{-m} \times C(T)$ $C(Y) \leftarrow C(T) + C(U)$
	if $m \leq \hat{m}$ go to XNOW Answer $\leftarrow C(Y) + C(Y) \times C(X)$	if $m \leq \hat{m}$ go to XNOW Answer $\leftarrow C(Y) + 1 - C(X)$	if $m \leq \hat{m}$ go to XNOW Answer $\leftarrow C(Y) + C(Y) \times [1 - C(X)]$	if $m \leq \hat{m}$ go to XNOW Answer $\leftarrow C(Y) + C(Y) \times 2^{-1} \times [1 - C(X)]$

Table 3 Iteration counts for test cases (quantities in parentheses are errors).

x_0	we^{x_0}	$w + \ln x_0$	w/x_0	$w/x_0^{\frac{1}{2}}$
0.0555 5555	$5(-2 \times 10^{-8})$	(out of range)	(out of range)	(out of range)
0.1555 5555	$5(-2 \times 10^{-9})$	(out of range)	(out of range)	(out of range)
0.2555 5555	$4(-1 \times 10^{-9})$	(out of range)	(out of range)	$10(5 \times 10^{-8})$
0.3555 5555	$6(-2 \times 10^{-8})$	(out of range)	(out of range)	$8(-3 \times 10^{-9})$
0.4555 5555	$10(-3 \times 10^{-9})$	(out of range)	(out of range)	$6(-2 \times 10^{-8})$
0.5555 5555	$5(-1 \times 10^{-8})$	$6(-3 \times 10^{-8})$	$6(4 \times 10^{-9})$	$7(-6 \times 10^{-9})$
0.6555 5555	$8(-2 \times 10^{-8})$	$7(-2 \times 10^{-8})$	$7(-2 \times 10^{-8})$	$5(-3 \times 10^{-8})$
0.7555 5555	(out of range)	$7(-2 \times 10^{-8})$	$7(-2 \times 10^{-8})$	$6(5 \times 10^{-8})$
0.8555 5555	(out of range)	$6(-2 \times 10^{-8})$	$6(-1 \times 10^{-8})$	$5(1 \times 10^{-8})$
0.9555 5555	(out of range)	$5(-2 \times 10^{-8})$	$5(-2 \times 10^{-8})$	$5(8 \times 10^{-9})$
Average count:	6.2	6.2	6.2	6.5

The test values of x are N-bit binary representations of $(0.0555\ 5555 + 0.1n)$, where n assumes all integer values permissible under the range restrictions given in the third section. The run assumed $N = 24$ (S/360–370 floating-point fraction size), 6 guard bits and unrounded arithmetic; also $\tilde{N} = N$, hence $\hat{m} = 12$. The left-ones count is used to find m to avoid complementation. Iteration continued until the computed m exceeded $\hat{m}$, then the terminal algorithm was invoked with the error-halving measure. The correctness of the algorithms and the accuracy of the result are both amply verified by the tests. No failure due to the algorithm has ever been found. The average Δm was indeed close to 2 after one or two iterations.

The results are summarized in Table 3; the number of iterations varies from three to ten, the mean being rather close to 6.3.

For two values of x_0, the x-transformations are displayed in Fig. 3. The automatic sequential bit-counting mechanism is seen to be extremely worthwhile; it allows both the skipping over of unimportant bits and, no less important, the use of the same m for more than one interation. In Fig. 3(c), for example, we encounter a case with $\Delta m = 0$ in one iteration, then $\Delta m = 8$ in the next. Without the bit counting technique one would have to employ costly trial and error comparison schemes, and/or severe limitations on the arguments to guarantee $\Delta m \geq 1$.

Conclusion

We have shown how the four functions of x can be evaluated by using shifts, adds, limited table look-ups, and sequential bit counting. The timing cost is low, being of the order of a conventional multiply time and sometimes less. The implementation by a unified piece of hardware is entirely within the state of the art. The possible drawbacks are a) the word-length incompatibility with the rest of the system because of the special guard bits required; and b) the variable timing.

Short of actual implementation, the hardware can be emulated by a nest of interrelated microsubroutines for

```
ITERATION 0   X= 0.1000111000011100011100011000000   M= 1
ITERATION 1   X= 0.0010011100110110001010011110100   M= 3  ΔM=2  R=0.27
ITERATION 2   X= 0.0000100001000101101001011111001   M= 5  ΔM=2  R=0.22
ITERATION 3   X= 0.0000000001100100101001010001001   M=10  ΔM=5  R=0.048
ITERATION 4   X= 0.0000000000100100101011001001   M=11  ΔM=1  R=0.36
ITERATION 5   X= 0.0000000000000100101011110001001   M=14  ΔM=3  R=0.13


ITERATION 0   X= 0.1010011111010001001111101000000   M= 1
ITERATION 1   X= 0.0100000000000101111001010100   M= 2  ΔM=1  R=0.38
ITERATION 2   X= 0.0000011011100101111111101110001   M= 6  ΔM=4  R=0.11
ITERATION 3   X= 0.0000000101110110111171000101100   M= 7  ΔM=1  R=0.42
ITERATION 4   X= 0.0000000011101111111001110000011   M= 9  ΔM=2  R=0.32
ITERATION 5   X= 0.0000000001111000000001110000001   M=10  ΔM=1  R=0.47
ITERATION 6   X= 0.0000000000111000000001110000001   M=11  ΔM=1  R=0.43
ITERATION 7   X= 0.0000000000010000000100000001   M=12  ΔM=1  R=0.33
ITERATION 8   X= 0.0000000000000000000100000001   M=20  ΔM=8  R=0.004
```

(a) we^{x_0}

```
ITERATION 0   X= 0.1000111000111000111000110000000   M= 2
ITERATION 1   X= 0.1011000111000111000110111110000   M= 2  ΔM=0  R=0.69
ITERATION 2   X= 0.1101111000111000111000101011000   M= 3  ΔM=1  R=0.43
ITERATION 3   X= 0.1111100111111111111111111000001   M= 6  ΔM=3  R=0.18
ITERATION 4   X= 0.1111110011110001111111111000000   M= 7  ΔM=1  R=0.35
ITERATION 5   X= 0.1111111111100001110001110111111   M=12  ΔM=5  R=0.063
ITERATION 6   X= 0.1111111111110011110011001111   M=13  ΔM=1  R=0.43


ITERATION 0   X= 0.1010011111010001111110100000   M= 2
ITERATION 1   X= 0.1101000111000111000110000010000   M= 3  ΔM=1  R=0.62
ITERATION 2   X= 0.1110101111111111111111111100010   M= 8  ΔM=1  R=0.40
ITERATION 3   X= 0.1111101011111111111111111110001   M= 6  ΔM=2  R=0.25
ITERATION 4   X= 0.1111111010101010111111111000   M= 8  ΔM=2  R=0.25
ITERATION 5   X= 0.1111111110101001010101010101111   M=10  ΔM=2  R=0.25
ITERATION 6   X= 0.1111111111101001100101001001   M=12  ΔM=2  R=0.26
ITERATION 7   X= 0.1111111111111001100100111101111   M=14  ΔM=2  R=0.29
```

(b) $w+\ln x_0$ or w/x_0

```
ITERATION 0   X= 0.1000111000111000111000111000000   M= 3
ITERATION 1   X= 0.1011001111111111111111111010011   M= 3  ΔM=0  R=0.67
ITERATION 2   X= 0.1110001111001111111111111000110   M= 5  ΔM=2  R=0.37
ITERATION 3   X= 0.1111001001000101111100011000010   M= 6  ΔM=1  R=0.49
ITERATION 4   X= 0.1111110011100111010000110111111   M= 7  ΔM=1  R=0.44
ITERATION 5   X= 0.1111110111010001011001011101100   M= 9  ΔM=1  R=0.36
ITERATION 6   X= 0.1111111111001111011011111000101   M=12  ΔM=4  R=0.087
ITERATION 7   X= 0.1111111111101111011011001111111   M=13  ΔM=1  R=0.34


ITERATION 0   X= 0.1010011111010010011111010000000   M= 3
ITERATION 1   X= 0.1110101000110011001100011000101   M= 4  ΔM=1  R=0.49
ITERATION 2   X= 0.1110111113000111100110010101111   M= 5  ΔM=1  R=0.37
ITERATION 3   X= 0.1111111100000000000000100110101   M=10  ΔM=5  R=0.062
ITERATION 4   X= 0.1111111101111111100101001100001   M=10  ΔM=0  R=0.5
ITERATION 5   X= 0.1111111111111111111001001100001   M=18  ΔM=8  R=0.0047
```

(c) $w/x_0^{1/2}$

Figure 3 Typical runs using $x_0 = 0.5555\ 5555$ and $0.6555\ 5555$.

microprogrammable machines. The algorithms can also be written as conventional subroutines; but then the compactness of the table will not be a major advantage, and the manipulations may be difficult using conventional instructions.

Acknowledgments

The writer is indebted to John C. McPherson for suggesting the shift-add approach and for sharing his experience and insight on the problem. The author also benefited from discussions with colleagues, particularly Ralph A. Willoughby, Irving T. Ho, Shmuel Winograd, Gerald S. Shedler, and Andrew R. Heller.

Appendix. Convergence of the iterations

We shall study Δm, the average advance of m, and give some detailed estimates for r.

• *Average advance of m*

We shall denote by $Z(\alpha)$ as the number of leading zero bits in a binary fraction α. Then

$$Z(2\alpha) = Z(\alpha) - 1 \qquad \text{for } \alpha < 1/2 . \tag{A1}$$

In terms of this notation we can write for all four algorithms

$$m = Z(|x_k - x_\omega|) + q , \qquad q = 1 \text{ or } 2 . \tag{A2}$$

The average advance of m between two iterations is

$$\begin{aligned}\langle \Delta m \rangle &= \langle Z(|x_{k+1} - x_\omega|) - Z(|x_k - x_\omega|) \rangle \\ &\sim \langle Z(qp) \rangle - Z[q(2^{-m} + p)] \qquad 0 \le p < 2^{-m} \\ &\sim \langle Z(p) \rangle - Z(2^{-m} + p) \\ &\sim \langle Z(p) \rangle - (m - 1) . \end{aligned} \tag{A3}$$

$Z(p)$ is a step function of p:

$$\begin{aligned} Z(p) &= m \qquad \text{for } p \text{ in } (2^{-m} - 2^{-N}, 2^{-m-1}) , \\ &\qquad\qquad (2^{m-1} \text{ cases}) ; \\ &= m + 1 \text{ for } p \text{ in } (2^{-m-1} - 2^{-N}, 2^{-m-2}) , \\ &\qquad\qquad (2^{m-2} \text{ cases}) ; \\ &= \cdots \\ &= m + i \text{ for } p \text{ in } (2^{-m-i} - 2^{-N}, 2^{-m-i-1}) , \\ &\qquad\qquad (2^{m-i-1} \text{ cases}) ; \\ &= N - 1 \text{ for } p = 2^{-N} , \qquad \text{(one case)} ; \\ &= N \qquad \text{for } p = 0 , \qquad \text{(one case)} . \end{aligned}$$

Hence we have for all 2^m cases, assuming each to be equally probable:

$$\begin{aligned}\langle Z(p) \rangle &= \frac{1}{2^m} [m \cdot 2^{m-1} + (m+1)\ 2^{m-2} + \cdots] \\ &= m + [0 \cdot 2^{-1} + 1 \cdot 2^{-2} + 2 \cdot 2^{-3} + \cdots] \\ &\sim m + \frac{1}{2} \sum_0^\infty i 2^{-i} = m + \frac{1}{2} \cdot 2 = m + 1 . \end{aligned}$$

Hence $\langle \Delta m \rangle \sim \langle Z(p) \rangle - (m - 1) = 2$.

• *Asymptotic estimate of r*

In the fifth section on convergence and accuracy an intuitive estimate of $r \sim 1/3$ was obtained. A better estimate is possible using $\hat{p} \ge p > 0$, with $\hat{p} = 2^{-m} + 2^{-Q}$. Q equals $(N + J)$ usually, but is $(N + J + 1)$ for the inverse square root. We have

$$\begin{aligned} s &\equiv 1 - r \\ &= (x_k - x_{k+1}) / (x_k - x_\omega) \\ &= 2^{-m} / (2^{-m} + p) + O(2^{-m}) \end{aligned}$$

then

$$\begin{aligned}\langle s \rangle &= (2^{-m}/\hat{p}) \int_0^{\hat{p}} dp / (2^{-m} + p) + O(2^{-m}) \\ &= \ln 2 + 2^{m-Q}(\ln 2 - 1/2) + O(2^{-m}) + O(2^{2m-2Q}) \\ &\sim \ln 2 . \end{aligned}$$

Hence

$$\langle r \rangle \sim 1 - \ln 2 = 0.30685 \cdots .$$

• *Detailed analysis*

We shall develop, for the four algorithms, bounds for r as well as average values.

1) For the exponential we have

$$s = [\ln(1 + 2^{-m})]/x_k = (2^{-m} - 2^{-2m}/2 + 2^{-3m}/3 - \cdots) \div (2^{-m} + p) > 1/2 - 2^{-m-2} < 1 - 2^{-m}/2 + 2^{-2m}/3 ,$$

and

$$\langle s \rangle \sim [\ln(1 + 2^{-m})] \cdot 2^m \ln 2 \sim \ln 2 - 2^{-m-1} \sim 0.693 - 2^{-m-1} ;$$

hence

$$1 > 1/2 + 2^{-m-2} > r > 2^{-m}(1/2 - 2^{-m}/3) > 0 ,$$

with

$$\langle r \rangle \sim 0.307 + 2^{-m-1} ;$$

further, $r < 5/8$ for the worst case value of $m = 1$.

2) For both $(w + \ln x)$ and (w/x),

$$s = 2^{-m} x_k/(1 - x_k) = 2^{-m}[1/(2^{-m} + p) - 1] > 1/2 - 2^{-m} \leq 1 - 2^{-m} ,$$

and

$$\langle s \rangle \sim \ln 2 - 2^{-m} ;$$

hence

$$1 > 1/2 + 2^{-m} > r \geq 2^{-m} > 0 ,$$

with

$$\langle r \rangle \sim 0.307 + 2^{-m} .$$

Second-order convergence is achieved whenever $x_k = 1 - 2^{-m}$, where $x_{k+1} = 1 - 2^{-2m}$ and $r = 2^{-m}$. Elsewhere for the worst case of $m = 2$, we have $r < 3/4$.

When simple left-ones counting is used (see the sixth section), we may have $x_k = 1 - 2^{-m}$ but choose $\tilde{m} = m + 1$, hence $a_k = 1 + 2^{-m-1}$. For this unlikely occurrence we also have $r = 1/2 + 2^{-m-1} < 3/4$.

3) For the inverse square root, $x_k = 1 - 2(2^{-m} + p)$, and $m \geq 2$,

$$s = 2^{-m}(2 + 2^{-m})x_k/(1 - x_k) = 2^{-m}(2 + 2^{-m})[1/2(2^{-m} + p) - 1] > 1/2 - 7 \cdot 2^{-m-2} - 2^{-2m} \geq 0$$

$$\leq 1 - 3 \cdot 2^{-m-1} - 2^{-2m} < 1$$

$$\langle s \rangle \sim \ln 2 - 2^{-m}(2 - (1/2)\ln 2 + 2^{-m}) = \ln 2 - 2^{-m}(1.654 + 2^{-m})$$

thus

$$1 \geq 1/2 + 7 \cdot 2^{-m-2} + 2^{-2m} > r \geq 3 \cdot 2^{-m-1} + 2^{-2m} > 0$$

with

$$\langle r \rangle \sim 0.307 + 2^{-m}(1.654 + 2^{-m}) .$$

The minimum convergence speed cannot be readily deduced from the upper bound of r, but $x_k/(1 - x_k)$ monotonically increases with x_k, and

$$s \geq 2^{-m}(2 + 2^{-m})(1/4)/(3/4) \geq 3/16 \text{ (when } m = 2)$$

and $r \leq 13/16 = 0.813$. As commented in the sixth section, "Nonstandard choices for m," the range $(1/4, 4/9)$ can be mapped into $(1/2, 1)$ in one iteration, using $m = 1$. Using this device, the smallest x_k becomes $1/2$, henceforward $r \leq 7/16$.

References

1. Tien Chi Chen, "Efficient Arithmetic Apparatus and Method," U.S. Patent No. 3,631,230, issued Dec. 28, 1971.
2. S. F. Anderson, J. G. Earle, R. E. Goldschmidt, and D. M. Powers, "The IBM System/360 Model 91: Floating-Point Execution Unit," *IBM J. Res. Develop.* **11,** 34–53 (1967). See especially pp. 48–52.
3. For a recent discussion see H. E. Salzer, "Radix Tables for Finding the Logarithm of Any Number of 25 Decimal Places," in *Tables of Functions and of Zeros of Functions*, National Bureau of Standards Applied Mathematics Series No. 37, 1954, pp. 143–144.
4. J. E. Meggitt, "Pseudo Division and Pseudo Multiplication Processes," *IBM J. Res. Develop.* **6,** 210–226 (1962).
5. B. P. Sarkar and E. V. Krishnamurthy, "Economic Pseudo-division Processes for Obtaining Square Roots, Logarithm, and Arctan," *IEEE Trans. Computers*, **C20,** 1589–1593 (December 1971).
6. J. S. Walther, "A Unified Algorithm for Elementary Functions," *Proc. AFIPS 1971 Spring Joint Computer Conference*, pp. 379–385.
7. D. Cantor, G. Estrin, and R. Turn, "Logarithmic and Exponential Function Evaluation in a Variable Structure Digital Computer," *IRE Trans. Electronic Computers*, **EC-11,** 155–164 (1962).
8. W. H. Specker, "A Class of Algorithms for $\ln x$, $\exp x$, $\sin x$, $\cos x$, $\tan^{-1} x$, and $\cot^{-1} x$," *IEEE Trans. Electronic Computers*, **EC-14,** 85–86 (1965).
9. B. G. de Lugish, "A Class of Algorithms for Automatic Evaluation of Certain Elementary Functions in a Binary Computer," University of Illinois, Department of Computer Science Report No. 399, June 1, 1970, 182 pages.

Received November 17, 1971

The author is located at the IBM San Jose Research Laboratory, San Jose, California 95114.