A Simulative Study of Correlated Error Propagation in Various Finite-Precision Arithmetics

JOHN D. MARASA AND DAVID W. MATULA

Abstract—The accumulated roundoff error incurred in long arithmetic computations involving a randomized mixture of addition, subtraction, multiplication, and division operations applied to an initial randomly generated data base is studied via simulation. Truncated and rounded floating-point arithmetic and truncated and rounded logarithmic arithmetic are simultaneously utilized for each of the computation sequences and the resulting roundoff error accumulations for these four systems are compared. The nature of the correlated errors incurred under various arithmetic operator mixes are discussed.

Index Terms—Accumulated roundoff error, correlated error, floating-point arithmetic, logarithmic arithmetic, simulation study.

I. Introduction and Summary

The accumulation of roundoff error in long computerized calculations is a phenomenon that can destroy an efficient and sound computational procedure predicated on arithmetic in the real number field. The additional time consuming steps of computing rigorous error bounds for the computation can yield overly pessimistic assessments of the accuracy of the results for two reasons. First, the progressive computation of error bounds via techniques such as interval analysis do not adequately compensate for the correlation of individual errors introduced when the same variable reappears in the computation, so the generated error bounds can be far from the tightest error bounds possible. Second, the general statistical nature of the errors introduced by roundoff will usually cause the results to satisfy much tighter error bounds than the worst possible cases with probability $1 - \epsilon$, where ϵ may be much smaller than the error probability for erroneous results due to undetected machine failure.

It is the purpose of this study to investigate and summarize the effects of accumulation of roundoff error in long randomly chosen computerized computations via simulation. More details relevant to the computerized implementation of the procedures described in this study are given in [1].

With regard to the methods of roundoff, four different "arithmetics" have been utilized simultaneously for the computation sequences. Rounded and truncated floating-point six significant hexadecimal digit arithmetic are two of the modes investigated. An overall assessment of the effect on error growth of using a large base such as 16 for floating-point computations is difficult to determine. It is well known [2] that the relative spacing between neighbors in a base β floating-point system has a log-periodic variability of a factor of β. For comparison, to avoid this variability of a factor of 16, we have also introduced a logarithmic number system where the relative spacing between neighbors is uniform. Two arithmetics characterized as rounded logarithmic and truncated logarithmic are developed. In Section II the rigorous specification for these four finite-precision arithmetics is presented.

The nature of our computational model for the assessment of correlated error growth for long "randomized" arithmetic computations is presented in Section III. A computerized computation may well involve 10^5 to 10^9 arithmetic operations executed on only 10^3 to 10^5 independent input data elements. Thus the phenomenon of correlated error due to multiple use of the same variable can be very pronounced in actual long computations.

In Section IV the results of numerous simulated computations are described and interpreted. The overall error-growth rates have a distinctive and somewhat predictable pattern depending on the operator mixes (i.e., add, subtract, multiply, divide frequencies). The results show that only a modest perturbation of the error-growth pattern is effected by the particular finite-precision arithmetic utilized for any operator mix.

The simulations of various multiply-divide operator mixes yield log-linear relative error accumulation curves when plotted against the number of operations performed, with higher multiply frequencies yielding steeper error accumulation slopes. This behavior for higher multiplication frequencies is attributed to a phenomenon that we have termed the principle of correlated multiplication: a continuing sequence of multiplications applied to a bounded set of approximate data elements will almost always eventually yield uniformly signed error biases and no further benefit from error cancellation.

Repetitive additions of positive elements exhibit only a linear growth of relative error accumulation with the number of operations. Combined multiply-divide-add operator mixes have error-growth rates essentially determined by the multiply and divide frequencies, as the multiply and divide error propagation swamps the errors attributable to addition operations. Introduction of the subtraction operation into the operator

Manuscript received January 22, 1973. This work was supported in part by NSF Grant GJ-446 and in part by the Advanced Research Projects Agency of the Department of Defense under Contract SD-302.
J. D. Marasa is with the School of Medicine at the Missouri Institute of Psychiatry, University of Missouri, St. Louis, Mo. 63139.
D. W. Matula is with the Department of Applied Mathematics and Computer Science, Washington University, St. Louis, Mo. 63130.

mix causes the steepest error-growth rates as can be expected from the explosive nature of error accumulation under subtraction (i.e., occasionally computing the difference of nearly equal approximate values).

The total accumulated error may be logically divided into two parts: 1) the initial conversion errors in the starting data will be propagated by (exact) arithmetic operations; and 2) operations on exact initial data will be subjected to the repeated roundoff errors of finite-precision arithmetic operations which error will also be propagated. Several simulations are performed that allow a separation and comparison of the effects of propagated initial error and overall accumulated error.

II. Finite Arithmetic Systems

Six significant hexadecimal digit-truncated arithmetic and six significant hexadecimal digit-rounded arithmetic describe two finite-precision arithmetic systems that are fairly well understood. In this section we wish to formalize these notions and also to define finite-precision arithmetics derived from truncated and rounded logarithmic arithmetic.

Following Matula [2], for the integers $\beta \geqslant 2$, called the *base*, and $n \geqslant 1$, called the *significance*, the *significance space* S_β^n is the following set of real numbers:

$$S_\beta^n = \{ b \mid b = i\beta^j \quad \text{where } i, j \text{ are integers, } |i| < \beta^n \}.$$

The set S_β^n may be interpreted as the space of β-ary numbers of n significant β-ary digits, or simply as the n-digit base β floating-point numbers.

The *truncation conversion mapping* T_β^n and the *rounding conversion mapping* R_β^n of the reals into S_β^n are defined, for $n \geqslant 1, \beta \geqslant 2$, by

$$T_\beta^n(x) = \begin{cases} \max \{ b \mid b \leqslant x, b \in S_\beta^n \}, & \text{for } x > 0 \\ \min \{ b \mid b \geqslant x, b \in S_\beta^n \}, & \text{for } x < 0 \\ 0, & \text{for } x = 0 \end{cases}$$

and

$$R_\beta^n(x) = \begin{cases} \min \left\{ b \mid \left(\dfrac{b + b'}{2} \right) > x, b \in S_\beta^n \right\}, & \text{for } x > 0 \\ \min \left\{ b \mid \left(\dfrac{b + b'}{2} \right) \geqslant x, b \in S_\beta^n \right\}, & \text{for } x < 0 \\ 0, & \text{for } x = 0 \end{cases}$$

where b' is the successor of b and is defined for $b \neq 0$ by $b' = \min \{ a \mid a \in S_\beta^n, a > b \}$.

By applying T_β^n *arithmetic* to an expression we shall mean the performance of the implied sequence of arithmetic operations on the specified real numbers where all real numbers utilized and all intermediate results obtained are mapped by T_β^n to S_β^n before each subsequent operation. Thus applying T_β^n arithmetic to the expression $(a + b) * c$ yields $T_\beta^n(T_\beta^n(T_\beta^n(a) + T_\beta^n(b)) * T_\beta^n(c))$. The relative difference between $T_\beta^n(T_\beta^n(T_\beta^n(a) + T_\beta^n(b)) * T_\beta^n(c))$ and $(a + b) * c$ is then the accumulated relative error effected by applying T_β^n arithmetic to $(a + b) * c$. Six significant hexadecimal digit-truncated arithmetic is then characterized as T_{16}^6 arithmetic.

R_β^n *arithmetic* is defined similarly, and R_{16}^6 arithmetic then characterizes six significant hexadecimal digit-rounded arithmetic.

As an alternative to floating-point arithmetic a logarithmically based arithmetic system will now be introduced.

For the arbitrary real number $\mu > 1$, called the *base*, the *logarithmic space* L_μ is the following set of real numbers:

$$L_\mu = \{ x \mid |x| = \mu^i, \quad i \text{ an integer} \} \cup \{ 0 \}.$$

The truncation mapping T_μ and the rounding mapping R_μ into the logarithmic space L_μ are defined as follows:

$$T_\mu(x) = \begin{cases} \mu^i \text{ such that } i = \left\lfloor \dfrac{\log |x|}{\log \mu} \right\rfloor, & x > 0 \\ -\mu^i \text{ such that } i = \left\lfloor \dfrac{\log |x|}{\log \mu} \right\rfloor, & x < 0 \\ 0, & x = 0 \end{cases}$$

$$R_\mu(x) = \begin{cases} \mu^i \text{ such that } i = \left\lfloor \dfrac{\log |x|}{\log \mu} + \frac{1}{2} \right\rfloor, & x > 0 \\ -\mu^i \text{ such that } i = \left\lfloor \dfrac{\log |x|}{\log \mu} + \frac{1}{2} \right\rfloor, & x < 0 \\ 0, & x = 0. \end{cases}$$

Note that mapping by R_μ employs geometric rounding rather than arithmetic rounding since the operation of rounding is carried out on the exponents.

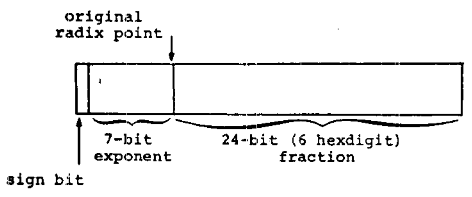
The notion of applying T_μ (or R_μ) arithmetic to an expression is then defined similarly to that of applying T_β^n arithmetic. The details of realizing T_μ arithmetic and R_μ arithmetic via integer arithmetic and log–antilog subroutines are described in [1].

In order to compare T_β^n arithmetic with T_μ arithmetic a choice of μ must be made so that the relative spacing and density of points in L_μ is comparable to that in S_β^n. Consider the members of S_β^n between β^{i-1} and β^i. Each such member may be thought of as the product of β^i and an n-digit base β fractional value between $1/\beta$ and β. Floating-point representation involves the storage of the integer exponent and the n-digit fraction. If instead we were to approximate a real number between β^{i-1} and β^i by a fixed-point approximation to its base β logarithm, this approximate log would have the characteristic $i - 1$ and could have a mantissa correct to n-base β digits stored in place of the n-digit fraction of the floating-point representation. This means that for a comparison with equitable storage constraints in a machine word we should have $\beta = \mu^{(\beta^n)}$. Thus

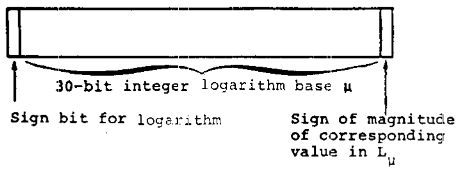
$$\mu = \beta^{(\beta^{-n})}.$$

To have L_μ comparable to S_{16}^6, we then have $\mu = 16^{(16^{-6})} = 2^{(2^{-22})}$, or about 1.000000165.

Thus the four finite-precision arithmetics denoted by T_{16}^6, R_{16}^6, T_μ, and R_μ for $\mu = 2^{(2^{-22})}$ are chosen as comparable on the basis of storage space needed in their implementation. The complexity of circuitry and total cost of implementation of these arithmetics will certainly vary, and constitutes an interesting study that we shall not pursue. Our concern here is



(a)


30-bit integer logarithm base μ
Sign bit for logarithm
Sign of magnitude of corresponding value in L_μ

(b)

Fig. 1. IBM/360 32-bit word usage for (a) floating-point and (b) logarithmic representations.

solely with the accuracy preservation properties of these comparable precision finite-precision arithmetics in heavy computing environments.

III. Random Computation Model

Our computational model for analysis of error accumulation is based essentially on studying the effect of a random sequence of arithmetic operations applied to randomly generated initial data. The total number of data values (i.e., variables) under consideration remains constant as each computed result replaces the value of one of the operand variables. Thus effects of correlated error are pertinent to our model just as they must be in any practical computational procedure where the number of arithmetic operations performed far exceeds the maximum number of data values ever stored at any time during the computation.

Specifically the simulation model developed consists of a 60-element vector $\boldsymbol{a}$ of pseudorandom IBM 360 double-precision floating-point numbers (i.e., in S_{16}^{14}) with the first 30 numbers chosen uniformly in [+0.25, +1.00) and the remaining 30 chosen uniformly in [+1.00, +4.00). Thus the first 30 hexadecimal numbers have exponent 0 and the latter 30 have exponent 1. From the double-precision vector $\boldsymbol{a}$ four more vectors are formed giving the initial data sets for the four different arithmetics considered. Letting $\mu = 2^{(2^{-22})}$,

$$\begin{aligned} r_i &= R_{16}^6\,(a_i), & i &= 1, \cdots, 60\\ t_i &= T_{16}^6\,(a_i), & i &= 1, \cdots, 60\\ m_i &= 2 \times \log_\mu |\,T_\mu\,(a_i)\,| + p\,(T_\mu\,(a_i)), & i &= 1, \cdots, 60\\ n_i &= 2 \times \log_\mu |\,R_\mu\,(a_i)\,| + p\,(R_\mu\,(a_i)), & i &= 1, \cdots, 60 \end{aligned}$$

where $p(x) = 1$ for $x < 0$, $p(x) = 0$ for $x > 0$.

The elements of the vectors $\boldsymbol{r}$ and $\boldsymbol{t}$ are IBM 360 single-precision floating-point numbers. The logarithmic encoding must contain sign information both for the logarithm and the original value being represented. It was found convenient to represent twice the base μ logarithm of $|\,T_\mu(a_i)\,|$ as an IBM 360 integer m_i with the final (parity) bit of m_i giving sign information on $T_\mu(a_i)$. Similarly n_i is an integer representing $R_\mu(a_i)$. The floating-point and logarithmic formats are shown in Fig. 1 (see Marasa [1]).

For each arithmetic operation two indices i and j are chosen at random uniformly over 1 to 60 (with $i \neq j$). An arithmetic operation $\odot$ is also randomly chosen according to some specified distribution over the operations add, subtract, multiply, and divide. Then in the vectors $\boldsymbol{a}, \boldsymbol{r}, \boldsymbol{t}$

$$\begin{aligned} a_i &\leftarrow T_{16}^{14}\,(a_i \odot a_j)\\ r_i &\leftarrow R_{16}^6\,(r_i \odot r_j)\\ t_i &\leftarrow T_{16}^6\,(t_i \odot t_j). \end{aligned}$$

Similarly m_i and n_i are replaced with the appropriately coded data resulting from the same operation using T_μ and R_μ arithmetic on the ith and jth elements of $\boldsymbol{m}$ and $\boldsymbol{n}$, respectively [1]. Notice that the jth element of each vector is unchanged in this process.

The value $|\,a_i\,|$ is then examined and if it falls outside the range [+0.25, +4.00), multiplication by the appropriate power of 16 to scale the value of $|\,a_i\,|$ to this range is then performed. The same scale factor is applied to r_i, t_i and the values encoded by m_i and n_i.

Relative errors for the arithmetics effected by R_{16}^6, T_{16}^6, T_μ, and R_μ are calculated with respect to arithmetic on the standard double-precision vector $\boldsymbol{a}$ as the "true value." Thus for example we compute $(r_i - a_i)/a_i$ for each $i = 1, \cdots, 60$ to find the accumulated relative errors under the rounded arithmetic R_{16}^6, and similarly for the other vectors. Although the magnitude of the values represented by r_i, t_i, m_i, and n_i can drift outside the range [0.25, 4.00), no anomalous behavior for these values was observed until the median errors were close to unity.

The scaling process does not alter any current relative error nor does it affect the further accumulation of relative error under multiply and divide operations. The scaling will tend to accentuate the error growth under subtraction, as prospective arguments are brought closer together in value.

The simulated random computation sequences we performed involved between 200 and 2500 arithmetic operations simultaneously performed by five different arithmetics. These were the four finite-precision arithmetics under study and the standard IBM 360 double-precision (T_{16}^{14}) computation that was taken as "exact," that is, the errors of the double-precision computation were taken as negligible compared to those of the single-precision level computation.

IV. Accumulated Error Growth in Random Computation Sequences

A data base for our model consists of a randomly generated double-precision vector $a = (a_1, \cdots, a_{60})$ with $a_i \in S_{16}^{14}$ for $1 \leqslant i \leqslant 60$, and the converted values of a in r, t, m, and n. Two data bases, K_1 and K_2, were generated. The process of converting each of the double-precision "true" values of the vector a to the initial data for $T_{16}^6, R_{16}^6, T_\mu$, and R_μ arithmetic ($\mu = 2^{(2^{-22})}$) introduces initial conversion error. These initial errors will be propagated through the succeeding arithmetic operations, so they will have a marked effect on the overall accumulated error.

Fig. 2 shows the median relative error introduced in the data bases K_1 and K_2 by the necessary conversions. As expected, the median relative error under truncation is close to twice the median relative error introduced by rounding.

The specification of a probability distribution for the four arithmetic operations add, subtract, multiply, and divide constitutes an *operator mix*. Numerous random computation sequences were applied to members of the data bases K_1 and K_2 by the procedure described in the previous section utilizing different operator mixes. The specific operator mixes studied were as follows.

1) 100 percent multiply.
2) 50 percent multiply–50 percent divide.
3) 75 percent multiply–25 percent divide.
4) 100 percent addition.
5) 100 percent subtraction.
6) 75 percent addition–25 percent subtraction.
7) 50 percent addition–50 percent subtraction.
8) 25 percent each operation.
9) $33\frac{1}{3}$ percent each for addition, multiplication, and division.

Some additional examples involving random "double-precision" T_{16}^{14} computations applied to the vector t of data base K_1 were generated to allow a separate assessment of the effects of just the propagated error.

The growth of relative error as a function of the number of arithmetic operations executed was plotted on a semilog scale for 16 different random computation sequences. The median relative error for the 60 elements of each data vector was computed by averaging the 30th and 31st ranked (in absolute value) relative errors and this quantity exhibited sufficiently smooth behavior for plotting.

Considerable information is immediately gleaned from the graphs. By far the determining characteristic of error growth is the operator mix. Although rounding is preferable to truncation in general, the improved performance is essentially limited to a relatively small factor (2 to 70) which does not significantly grow with the number of operations performed. The logarithmic arithmetics show a modest advantage for

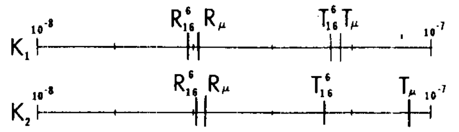


Fig. 2. Median of the absolute values of the relative errors introduced to the data bases K_1 and K_2 by the initial conversions R_{16}^6, R_μ, T_{16}^6, and T_μ.

multiply-divide operator mixes, but otherwise no trend is discernible. The operations on data bases K_1 and K_2 are sufficiently similar so that the results should be considered reproducible. A detailed analysis of the various cases follows.

A. Multiplication and Division Operator Mixes

For multiplication (division) operations the propagated error is essentially the sum (difference) of the relative errors of the operands. Note that for the truncated arithmetics T_{16}^6 and T_μ all initial relative errors are negative or zero. Thus under 100 percent multiply these errors will be compounded without cancellation. For rounded arithmetic the situation is more subtle.

The initial relative errors from conversion utilizing rounding are about one-half that of those imposed by truncation in average magnitude (absolute value). If the signs of the average relative errors are taken into account the initial average relative error is then essentially zero. The nature of the 100 percent multiply computation would then make plausible the conclusion that the average expected relative error should remain zero. Actually the expected relative error of zero can be justified, but only as an average over many such simulated computations with different random data bases. For any single data base it should be noted that with probability one the signs of all 60 relative errors in the data of R_{16}^6 and R_μ arithmetic will eventually become either all positive (50 percent chance) or all negative (50 percent chance). From that point on the propagated errors will show no more cancellation, and the growth rate of the magnitude of the relative error should pattern that of the truncated arithmetics. This fading out of the beneficial effects of error cancellation under correlated multiplication is a noteworthy phenomenon that we term the principle of correlated multiplication.

The data of Figs. 3 and 4 clearly show this behavior. Over the first several hundred multiply operations the rounded arithmetics exhibit slower growth rates in Figs. 3 and 4, but after 300–400 operations the error-growth rates then increase slope and proceed essentially parallel to the truncated arithmetics' error-growth rates. Applying a curve-smoothing function over the range of near linear behavior on this semilog scale one derives the equations of Table I.

From Table I it is evident that the error-growth rates given by the slopes in the linear equations vary only by a few percent for the different finite-precision arithmetics and the different random data bases. The rounded arithmetics are uniformly better than the corresponding truncated arithmetics in controlling overall accumulated error by factors of from about 7 to 75, but these factors do not significantly change over the

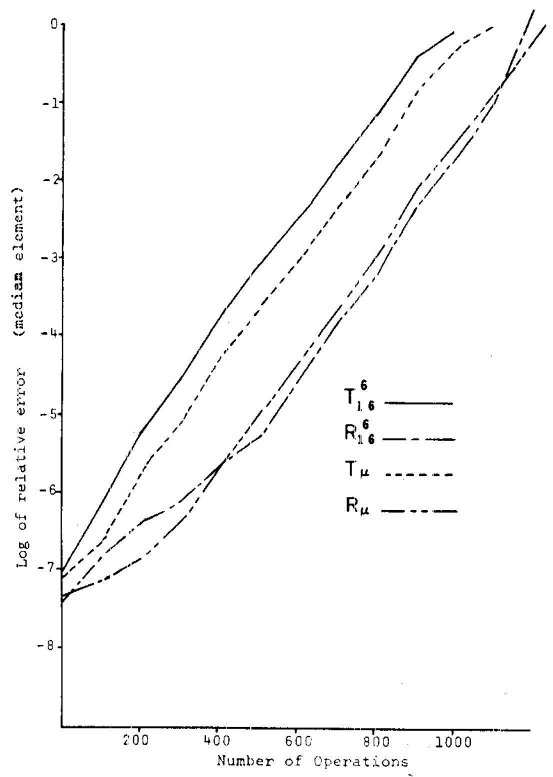


Fig. 3. 100 percent multiply using data base K_1.

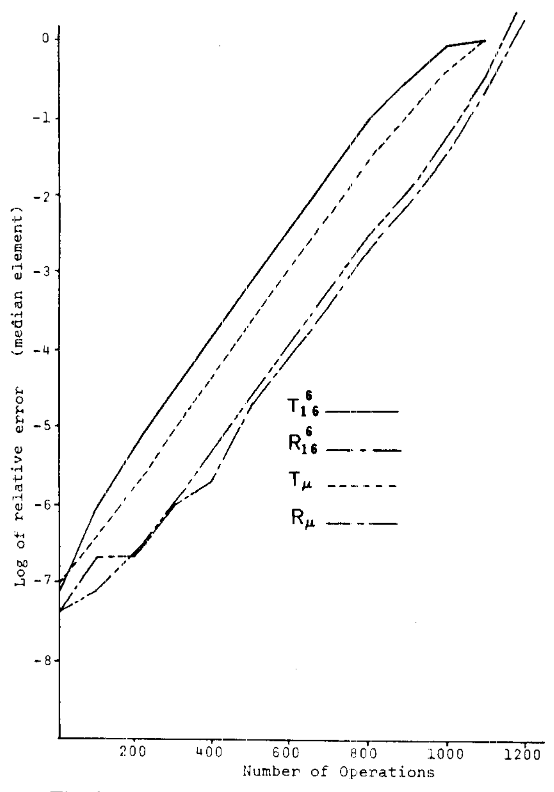


Fig. 4. 100 percent multiply using data base K_2.

TABLE I
EQUATIONS DERIVED FROM SMOOTHED CURVES FOR 100 PERCENT MULTIPLICATION OPERATIONS ON RANGE FROM 200 TO 900 OPERATIONS

Mode of Arithmetic	Data Base K_1	Data Base K_2
T^6_{16}	$Y = 0.00688X - 6.90$	$Y = 0.00671X - 6.47$
R^6_{16}	$Y = 0.00710X - 8.77$	$Y = 0.00660X - 8.04$
T_μ	$Y = 0.00690X - 7.12$	$Y = 0.00694X - 7.16$
R_μ	$Y = 0.00678X - 8.13$	$Y = 0.00687X - 8.03$

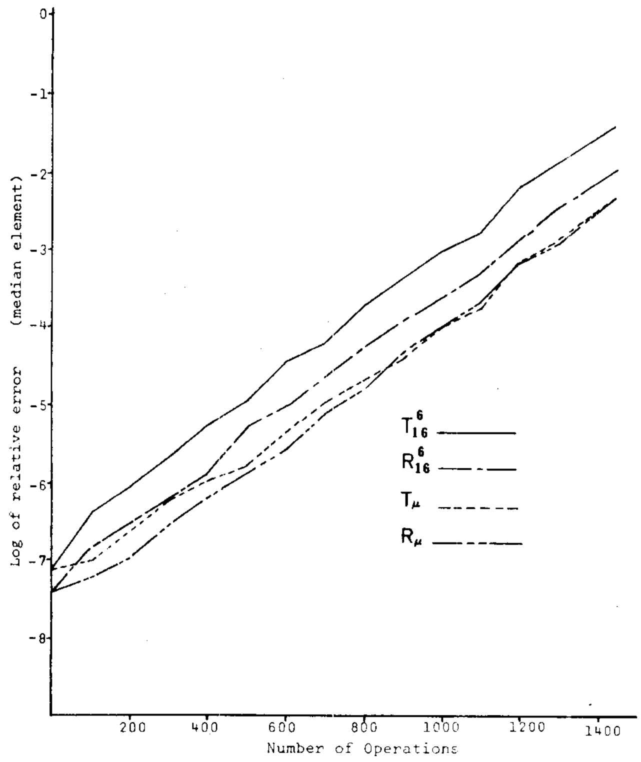


Fig. 5. 50 percent multiply-50 percent divide using data base K_1.

TABLE II
EQUATIONS DERIVED FROM SMOOTHED CURVES FOR 50 PERCENT MULTIPLICATION AND 50 PERCENT DIVISION OPERATIONS USING DATA BASE K_1

T^6_{16}	$Y = 0.00388X - 6.88$
R^6_{16}	$Y = 0.00379X - 8.24$
T_μ	$Y = 0.00321X - 7.36$
R_μ	$Y = 0.00366X - 8.27$

range of operations considered. Thus rounding does more to delay the eventual growth rate rather than lessen its slope.

The error-growth rate corresponding to an operator mix of 50 percent multiply and 50 percent divide is seen in Fig. 5 to have a less severe slope. Fitting linear equations to the data of Fig. 5 yields Table II, and it can be seen that the growth rate for 50 percent multiply and 50 percent divide has a slope essentially one-half that of the 100 percent multiply operator mix.

This is a significant improvement in the control of error accumulation. The reason for this improvement is that division yields a relative error equal to the difference of the relative errors of the operands. Thus repeated error cancellations will continue to occur in our random computation model in contrast to the 100 percent multiply case. Figs. 6 and 7 show results for the operator mix, 75 percent multiply-25 percent divide, applied to both data sets K_1 and K_2, and the growth rate of error is seen to fall in between the rates for 100 percent multiply and 50 percent multiply-50 percent divide.

The conclusions from these computations with various multiply-divide operator mixes is that the actual operator mix essentially determines the extent of error cancellation and therefore the growth rate of accumulated error. The purpose of rounded arithmetic has been thought of as an effort to effect error cancellations, but these computations show that when the number of arithmetic operations far exceeds the number of variables utilized in the operations, the accumulated correlated error inherent in the finite-precision computations eventually swamps any further benefits from rounding after some definite initial advantages. It is noted that the T_{16}^6 standard floating-point arithmetic as implemented on the IBM System/360 gives uniformly the worst relative accuracy for all multiply-divide operator mixes examined.

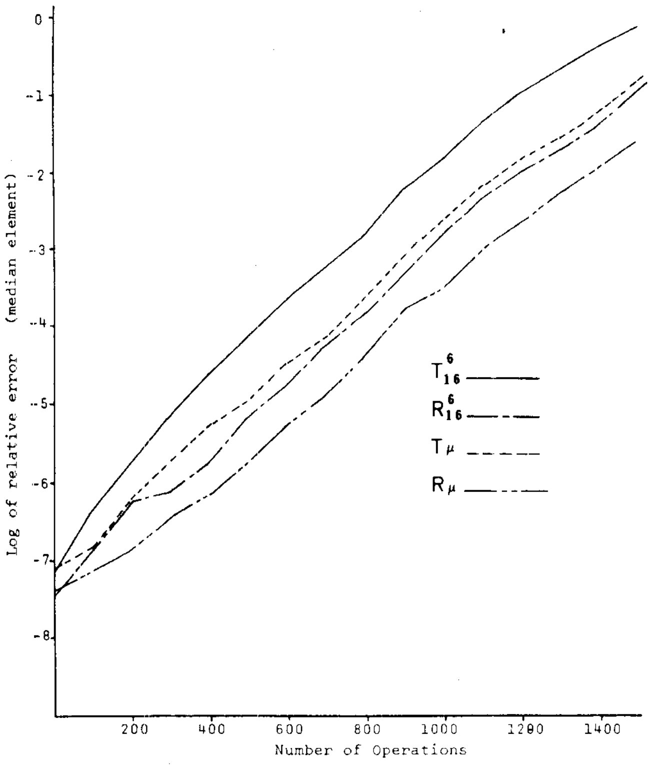


Fig. 6. 75 percent multiply-25 percent divide using data base K_1.

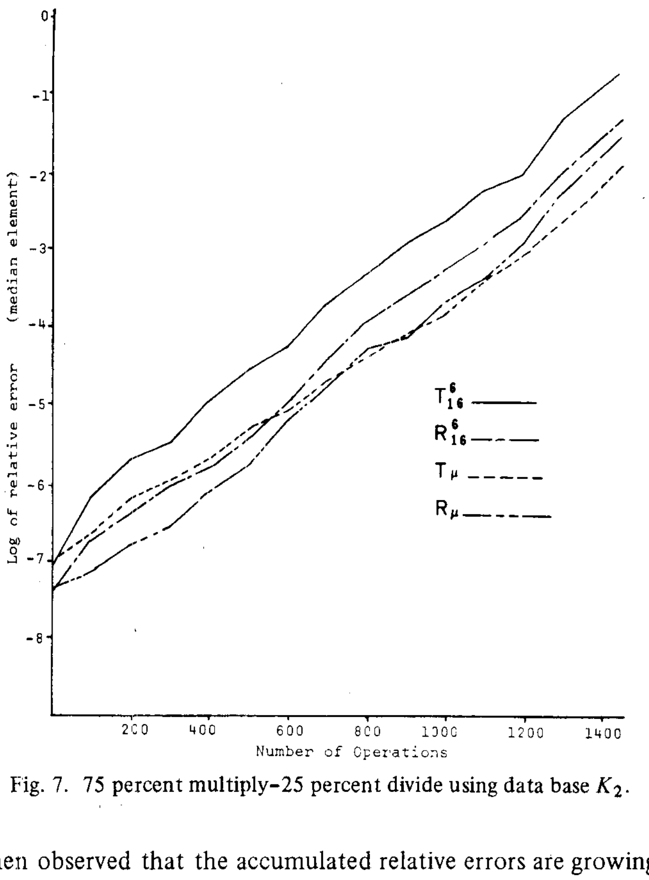


Fig. 7. 75 percent multiply-25 percent divide using data base K_2.

B. Addition and Subtraction Operator Mixes

An analysis of the growth of accumulated relative error for addition and subtraction is not nearly so straightforward as that for multiplication and division. The error in the result of an addition of positive numbers will tend to be dominated by the error of the operand that is the larger operand in magnitude of the two.

Consider two positive operands a_1 and a_2 and their respective accumulated relative errors r_1 and r_2. Let

$$a_1 = A_1 + \Delta a_1 \quad \text{and} \quad a_2 = A_2 + \Delta a_2$$

where A_1 and A_2 are the "true values" of a_1 and a_2 and Δa_1 and Δa_2 are the absolute errors in these values. The relative error of the sum of the two values a_1 and a_2 can be calculated by $r_+ = \Delta(a_1 + \Delta a_2)/(A_1 + A_2)$ or

$$r_+ = \frac{A_1}{A_1 + A_2}\left(\frac{\Delta a_1}{A_1}\right) + \frac{A_2}{A_1 + A_2}\left(\frac{\Delta a_2}{A_2}\right)$$
$$= \frac{A_1}{A_1 + A_2} r_1 + \frac{A_2}{A_1 + A_2} r_2.$$

If we let $\theta = A_1/(A_1 + A_2)$, then $(1 - \theta) = A_2/(A_1 + A_2)$ and

$$r_+ = \theta r_1 + (1 - \theta) r_2.$$

Thus, the relative error in addition is a weighted function of the relative errors of the operands [3].

The results of 100 percent addition applied to data base K_1 (all positive values) are plotted in Fig. 8. The median values of the relative errors are seen to grow very slowly on the semilog scale. The data from Fig. 8 with abscissa beyond 1000 operations is replotted in Fig. 9 using a linear-ordinate axis. It is then observed that the accumulated relative errors are growing linearly with the number of observations.

Using the same notation as above, Pennington [3] also describes the accumulated relative error r_- for the difference $a_1 - a_2$. This expression is

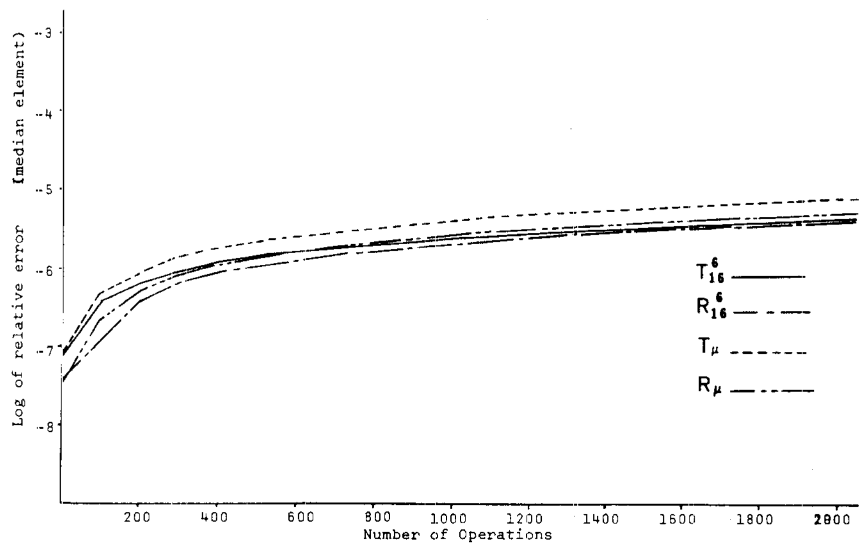


Fig. 8. 100 percent addition using data base K_1.

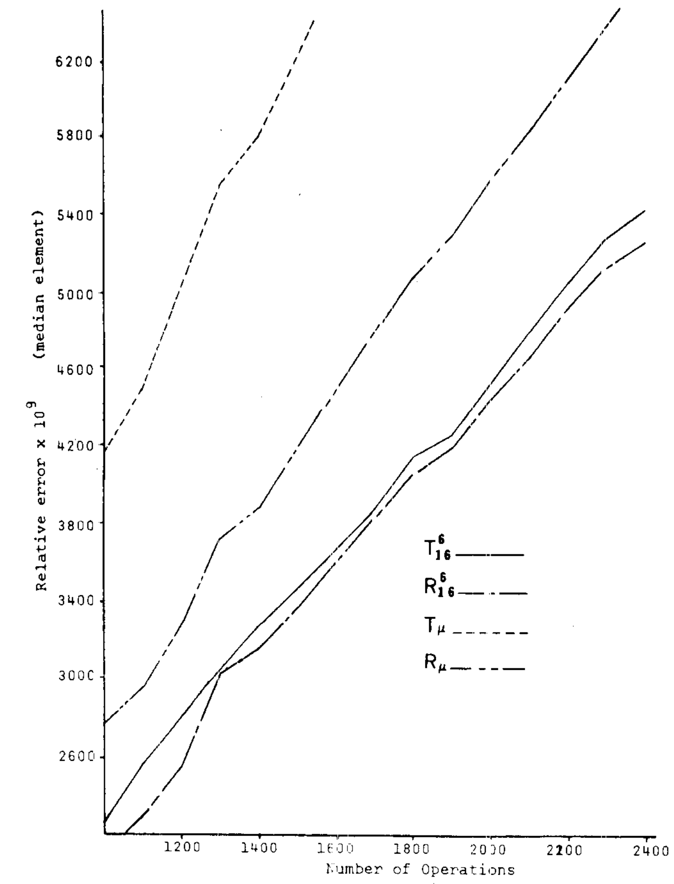


Fig. 9. 100 percent addition-actual relative error.

$$r_- = \frac{A_1 + A_2}{A_1 - A_2} \left[\theta r_1 + (1 - \theta) r_2 \right].$$

If A_1 and A_2 are two numbers that are near each other in value, the accumulated relative error could "explode" into quite a large value. To quote from Pennington [3, p. 93]:

The loss of the leading significant figures in the subtraction of two nearly equal numbers is the greatest source of inaccuracy in most calculations and forms the weakest link in a chain computation where it occurs. Floating-point arithmetic offers little or no protection against this form of accuracy loss · · · .

Fig. 10 is a graph of 100 percent subtraction operations on data base K_1 and the operations do indeed "explode" after relatively few operations. This effect when combined with addition is illustrated graphically in Fig. 11 which gives the curves for the accumulated relative error growth for the oper-

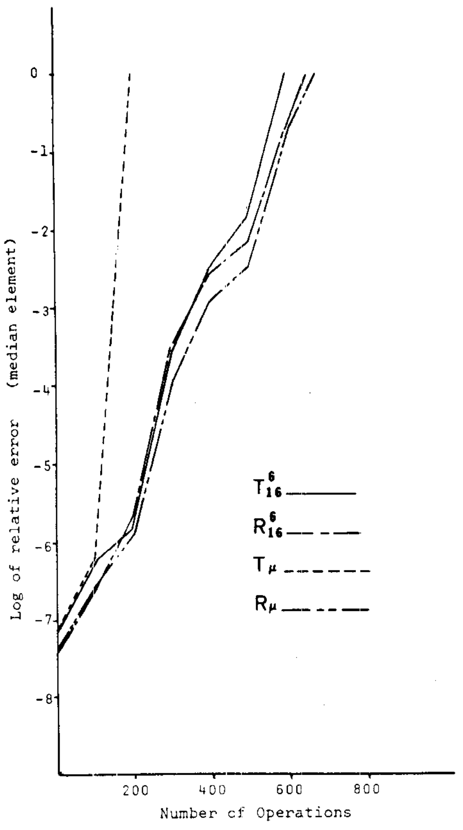


Fig. 10. 100 percent subtraction using data base K_1.

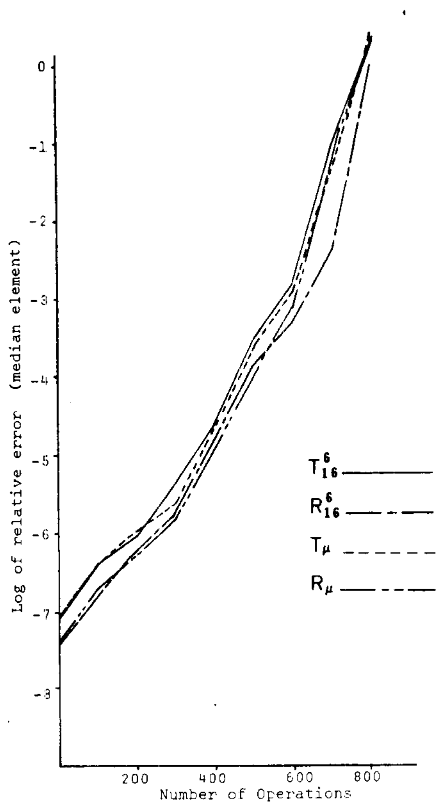


Fig. 11. 75 percent addition–25 percent subtraction using data base K_1.

ator mix consisting of 75 percent addition and 25 percent subtraction. The growth rate is somewhat steeper than linear on the log scale. Similar behavior is noted for the operator mix 50 percent addition and 50 percent subtraction in Fig. 12. Figs. 13 and 14 involve data base K_2 and show results comparable to Figs. 8 and 11.

For addition and subtraction operator mixes no one of the four arithmetics T_{16}^6, R_{16}^6, T_μ, or R_μ clearly dominates in controlling error growth in all cases. The error growth is almost completely a function of the add-subtract operator mix with practically no dependence on the mode of arithmetic utilized.

C. Operator Mixes for All Operations

We have previously seen that multiplication and division operator mixes tend to propagate and accumulate roundoff error at a linear rate on a log scale and that the accumulation of relative error of strict addition grows at a true linear rate. The introduction of subtraction, however, caused a great difference in the patterns of growth in the addition tests. This is true in this third class of tests also. Fig. 15 shows that the growth of accumulated relative error for a sequence involving equal amounts of all four arithmetic operations is very similar to the tests involving only addition and subtraction as shown in Figs. 11, 12, and 14. The slope of the error growth is increasing even on the log scale. Once again, the subtraction operation tends to dominate the growth of the error, although it occurs only 25 percent of the time.

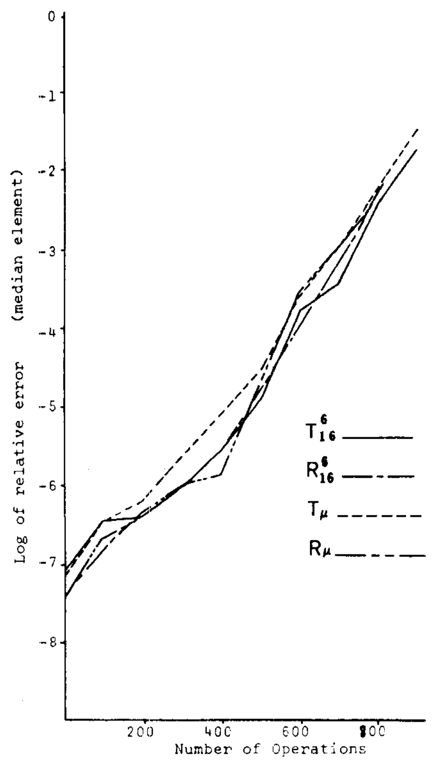


Fig. 12. 50 percent addition–50 percent subtraction using data base K_1.

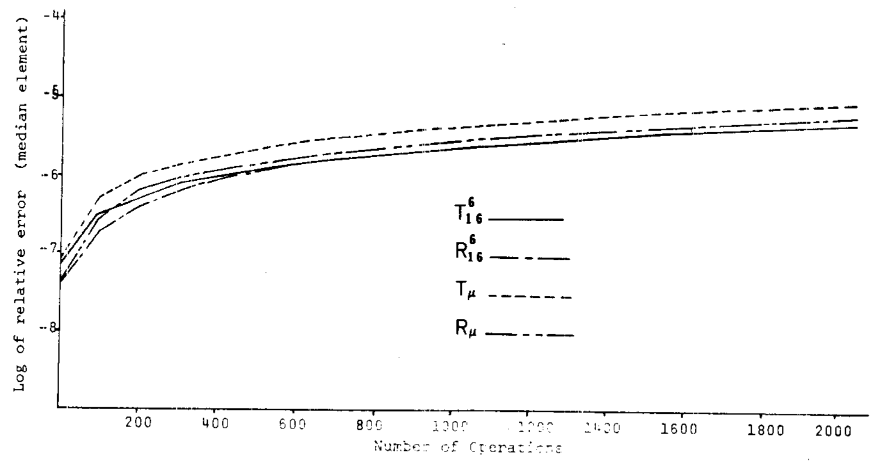


Fig. 13. 100 percent addition using data base K_2.

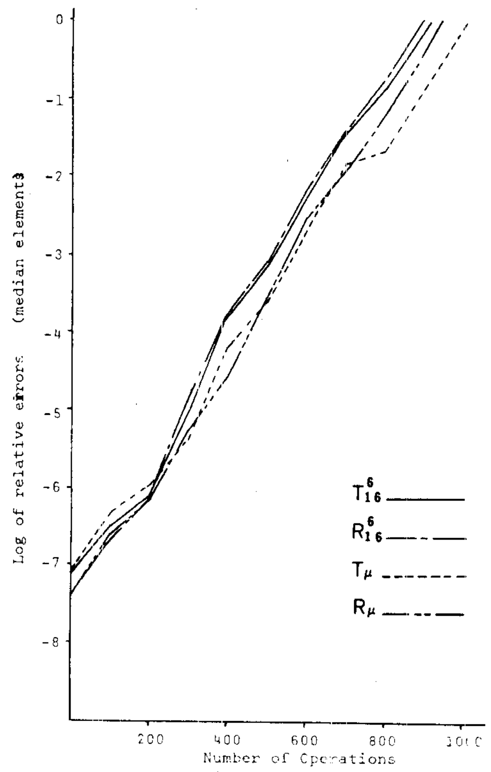


Fig. 14. 75 percent addition–25 percent subtraction using data base K_2.

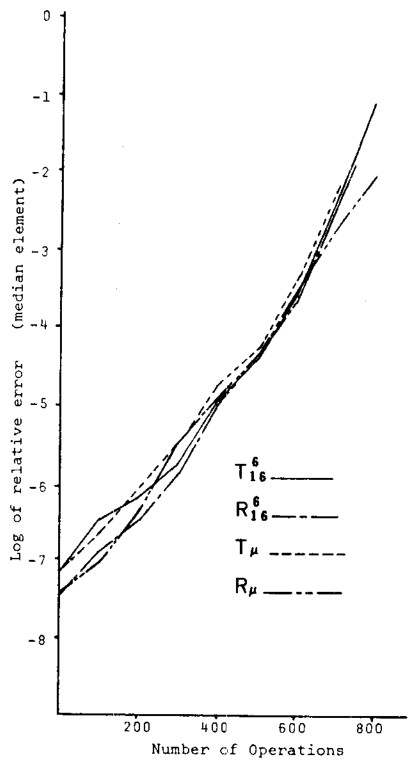


Fig. 15. 25 percent-all operations using data base K_1.

Fig. 16 shows the accumulated relative error curves for equal ratios of addition, multiplication, and division. After a full 2500 operations, the total accumulated relative error has still not reached 0.1 in any of the four finite-precision arithmetics. In this case, the addition operations are smoothing and slowing the growth rate of the relative error from the multiplication and division operations. Note that compressing this curve by a factor of 2/3 on the horizontal axis yields a curve similar in general growth to the 50-50 multiplication and division operator mix results of Fig. 5. No one of the four particular finite-precision arithmetics shows an improvement factor greater than 10 over any other in Figs. 15 and 16.

D. Propagation of Initial Error

We now want to examine the effects of propagated conversion error only, unaffected by the accumulated roundoff error inherent in the arithmetic operations. To implement this, the data base vector $\boldsymbol{a}$ is simply mapped by T^6_{16} arithmetic to $\boldsymbol{t}$, and then further arithmetic on $\boldsymbol{t}$ is executed in standard double-precision just as are operations on the "true values" of $\boldsymbol{a}$.

In Fig. 17 it is shown that the propagated conversion error generated for all operations using data base K_1 grows at a rate equally erratic and comparable to the curve of Fig. 15 which includes accumulated roundoff error.

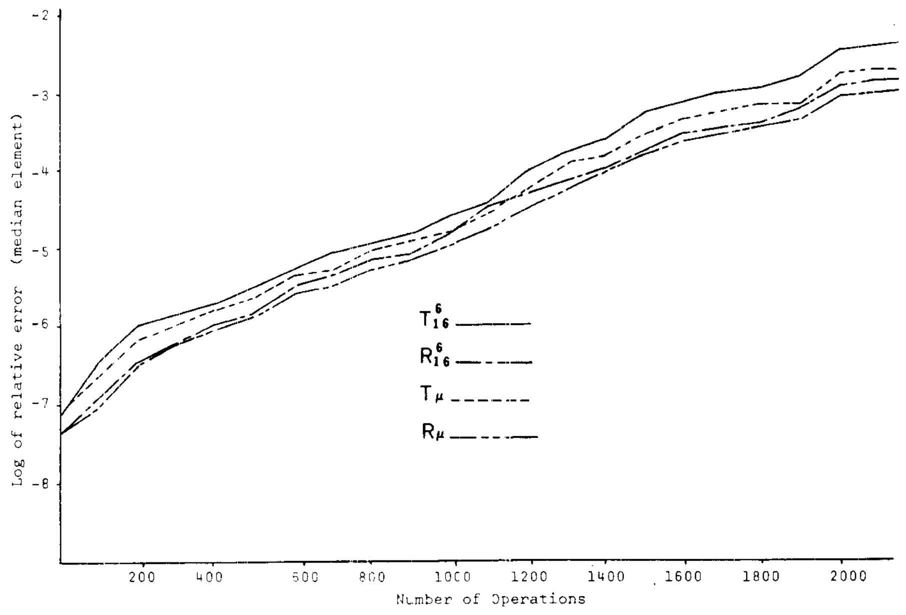


Fig. 16. $33\frac{1}{3}$ percent additon-$33\frac{1}{3}$ percent multiply-$33\frac{1}{3}$ percent divide using data base K_1.

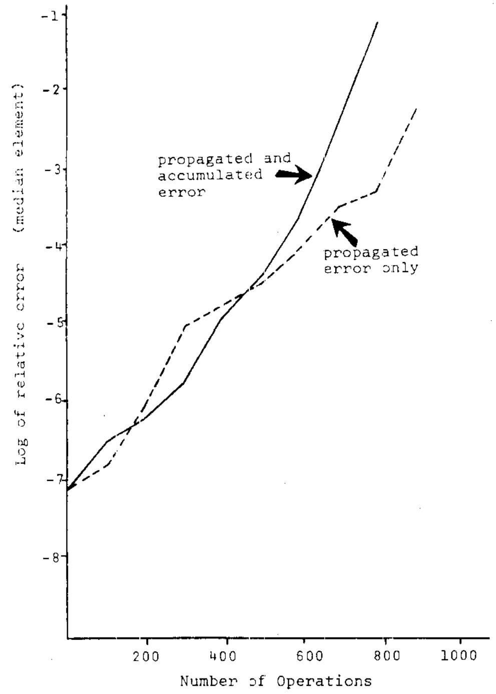


Fig. 17. Propagated initial conversion error versus accumulated roundoff error for 25 percent–all operations mapped by T^6_{16} using data base K_1.

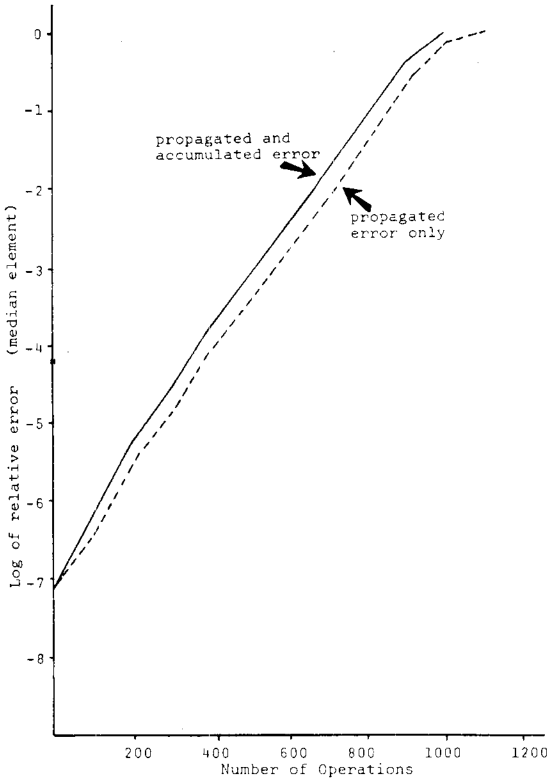


Fig. 18. Propagated initial conversion error versus accumulated roundoff error for 100 percent multiplication mapped by T^6_{16} using data base K_1.

Fig. 18 shows that propagated conversion error for 100 percent multiplication mapped initially by T^6_{16} (data base K_1) does compare favorably to the case also admitting further truncation errors. It is less by an approximate factor of 0.3 on the vertical log scale (a factor of two). The curves are virtually parallel implying the same growth rate once the model has settled down.

The propagation of initial conversion error for 100 percent addition mapped initially by T^6_{16} (data base K_1) is compared to the same arithmetic including accumulated roundoff error in Fig. 19. After 500 operations the error stops growing completely. This is to be expected since the relative error of the result of an addition operation was seen to be a linear combination of the relative errors of the two operands. Thus the accumulated relative error can never exceed the largest initial relative error.

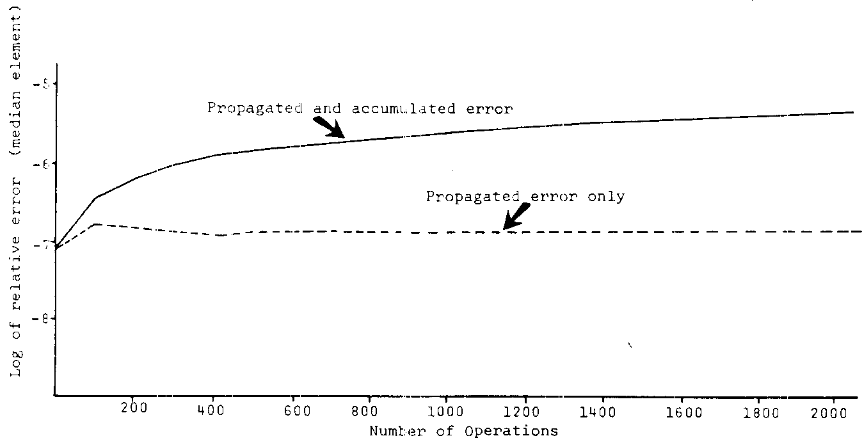


Fig. 19. Propagated initial conversion error versus accumulated roundoff error for 100 percent addition mapped by T_{16}^{6} using data base K_1.

References

[1] J. D. Marasa, "Accumulated arithmetic error in floating-point and alternative logarithmic number systems," M.S. thesis, Sever Inst. Technol., Washington Univ., St. Louis, Mo., June 1970.

[2] D. W. Matula, "A formalization of floating-point numeric base conversion," *IEEE Trans. Comput.*, vol. C-19, pp. 681-692, Aug. 1970.

[3] R. H. Pennington, *Introductory Computer Methods and Numerical Analysis*, 2nd ed. Toronto, Canada: Macmillan, 1970.

John D. Marasa was born in St. Louis, Mo., on April 5, 1946. He received the B.S. degree in applied mathematics and computer science from Washington University, St. Louis, Mo., in 1968 and the M.S. degree in computer science from the Sever Institute of Technology of Washington University in 1970.

From 1968 to 1969 he taught introductory Fortran programming at Washington University and in 1970 taught at the University of Missouri, St. Louis. Since June 1970 he has been employed as Senior Scientific Programmer/Analyst for the University of Missouri School of Medicine at the Missouri Institute of Psychiatry in the Department of Clinical Neurophysiology and Psychopharmacology. He is engaged in research in the quantitative real-time analysis of human electroencephalograms (EEG) and the effects of drug therapy on the EEG and other physiological measures during sleep and wakefulness.

Mr. Marasa is a member of the Association for Computing Machinery.

David W. Matula was born in St. Louis, Mo., on November 6, 1937. He received the B.S. degree in engineering physics from Washington University, St. Louis, Mo., in 1959 and the Ph.D. degree in engineering science (operations research) from the University of California, Berkeley, in 1966.

From 1959 to 1960 he was a Woodrow Wilson Fellow at the University of California, Berkeley. During the period from 1957 to 1968 he was associated both part time and as a consultant with the Monsanto Chemical Company, St. Louis, doing research into the structure of inorganic polymers. From 1960 to 1966, while at the Computer Center, University of California, Berkeley, he did programming and research in conjunction with the center's consultant to the physical sciences, and co-developed a 20-h videocourse on Fortran programming. He has been a consultant to both education and research divisions of International Business Machines. Since 1966 he has been at Washington University, where he is currently Associate Professor of applied mathematics and computer science. For the period February-June 1973 he has been a Visiting Research Scientist at the Center for Numerical Analysis, University of Texas, Austin. His research interests are computer arithmetic, finite-precision arithmetic, graph theory, and combinatorial mathematics.

Dr. Matula is currently a National Lecturer for the Association for Computing Machinery and Secretary-Treasurer of their special interest group on computer science education. He is also a member of Tau Beta Pi, Sigma Xi, the Operations Research Society, the Mathematical Association of America, and the Society for Industrial and Applied Mathematics.