J. E. Meggitt

Pseudo Division and Pseudo Multiplication Processes

Abstract: Some digit-by-digit methods for the evaluation of the elementary functions are described. The methods involve processes that resemble repeated-addition multiplication and repeated-subtraction division. Consequently, the methods are easy to implement and the resultant execution times are short.

Introduction

It is customary in computers to build an arithmetic unit which is capable of adding, subtracting, multiplying and possibly dividing. To perform more complicated operations it is usual to write routines that use these basic operations, operating on words at a time. However, in certain cases, digit-by-digit methods exist for the evaluation of certain functions. Some of these are attractive because they can be made faster than the corresponding subroutine methods, and also because they do not need so much storage space; sophisticated fast subroutines for evaluating the elementary functions require the storage of many constants. Hence, it is worthwhile considering whether more powerful arithmetic units could be provided which would be capable of performing these digit-by-digit methods.

The advent of microprogramming as a method of computer control has made it very easy to construct relatively complicated arithmetic units. The methods described here are ideal for a machine with such control. However, these methods can also easily be implemented in a conventional manner.

The first part of this paper will show flow diagrams of four routines. These will form, respectively, y/x, $\log[1 + (y/x)]$, $\tan^{-1}(y/x)$, and $\sqrt{y/x}$ for given y and x. A striking feature of these four routines is their similarity. This means that a common microprogram subroutine or common hardware may be used, in one of four different modes of operation, and this, of course, represents an economy in the number of microinstructions or in the amount of hardware required. These ideas are particularly helpful to the designer of small but powerful machines, but they may also have applications for larger machines, where it is required to have the elementary functions "built in."

The second part of this paper shows how essentially, by reversing the above routines, $\tan y$, xe^y and xy^2 may be generated.

In the paper it will be supposed that base 10 arithmetic is being used. This is by no means a restriction. However, the fact that operations can be performed with base 10 is an advantage in the area of small machines where it may be inconvenient to provide decimal-to-binary conversion.

Section 1: Division

The basic repeated subtraction process is, of course, very well known indeed. For completeness the flow diagram for it is shown in Fig. 1.

Initially register A contains an n digit word y, and register B an n digit word x. x is subtracted from y as many times as is possible until A becomes as small as it can without going negative. The number of subtractions is recorded in a counter whose contents are transferred to a shifting register Q. One too many subtractions is performed and this requires a subsequent addition. It is possible of course to omit this addition and alternately subtract and add, and this possibility remains in the cases of the routines to be described. However, for the sake of simplicity, this complication will be omitted. (A) are now multiplied by 10 and the process is repeated. When it has been repeated n times as shown by the counter j, the (Q) are the n digits of the quotient y/x.

To keep the digits of the answer less than 10, there is a restriction $y/x < 10$. A must be a register of length $n + 1$ to allow for the shift left. The answer is, of course, exact, except for the remainder.

• *Modified division*

The modification to this basic routine which is proposed here is to provide a modifier register M. This is loaded during each subtraction cycle immediately prior to subtraction. After each subtraction the pseudo divisor which is in B is updated. This is done by adding to the divisor, the contents of the modifier M shifted j places to the right, where the $j + 1^{th}$ quotient

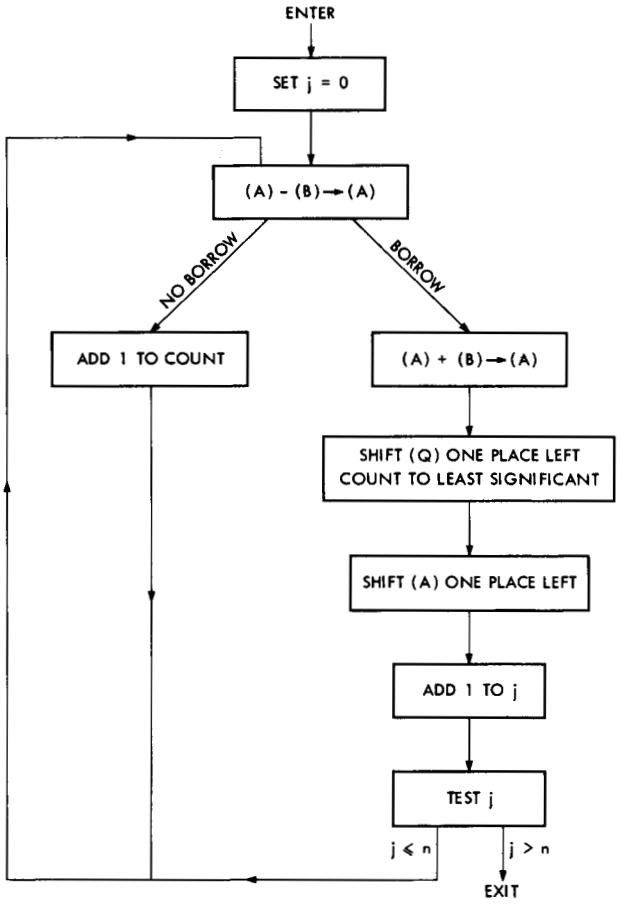


Figure 1 **Flow diagram for elementary divider.** *Initially A contains y and B contains x. The quotient is obtained in Q.*

digit q_j, is being formed. A flow diagram for this process is shown in Fig. 2.

It will transpire roughly that the various routines differ only in that the contents of different registers are used to load the modifier. It will be found indeed that in one case the digits q_j are such that

$$\log[1 + (y/x)] = \sum_j q_j \log(1 + 10^{-j}) ;$$

in another that

$$\tan^{-1}(y/x) = \sum_j q_j \tan^{-1} 10^{-j}$$

while in a third

$$\sqrt{y/x} = \sum_j q_j 10^{-j} .$$

The precise ways in which the various routines work will next be described. Flow diagrams for all of them are shown together in Fig. 3.

• *To form* $\log[1 + (y/x)]$

The method used is essentially Brigg's method modified so that the main part of the calculation is by a pseudo division process, and handled in such a way that accuracy is retained.

The method consists of choosing digits q_j so that

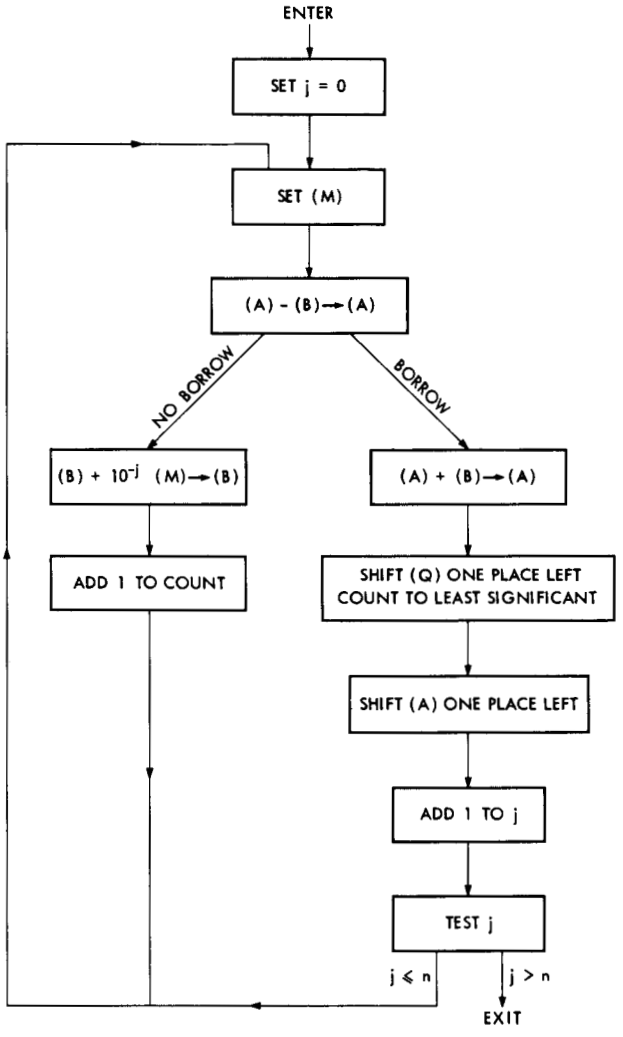
$$y + x = x \prod_{j=0} (1 + 10^{-j})^{q_j} , \qquad (1)$$

where y and x are given n digit positive numbers. The calculation of q_j is made to resemble a division. When this has been done,

$$\log[1 + (y/x)] = \sum_j q_j \log(1 + 10^{-j}) . \qquad (2)$$

The calculation is split into two parts. In the first part, digits q_j are calculated. In the second, the logarithm is calculated from (2) using stored values for $\log(1 + 10^{-j})$. This latter calculation turns out to be a pseudo multiplication.

Figure 2 **Flow diagram for elementary modified divider.** *Initially A contains y and B contains x. The pseudo quotient is obtained in Q.*



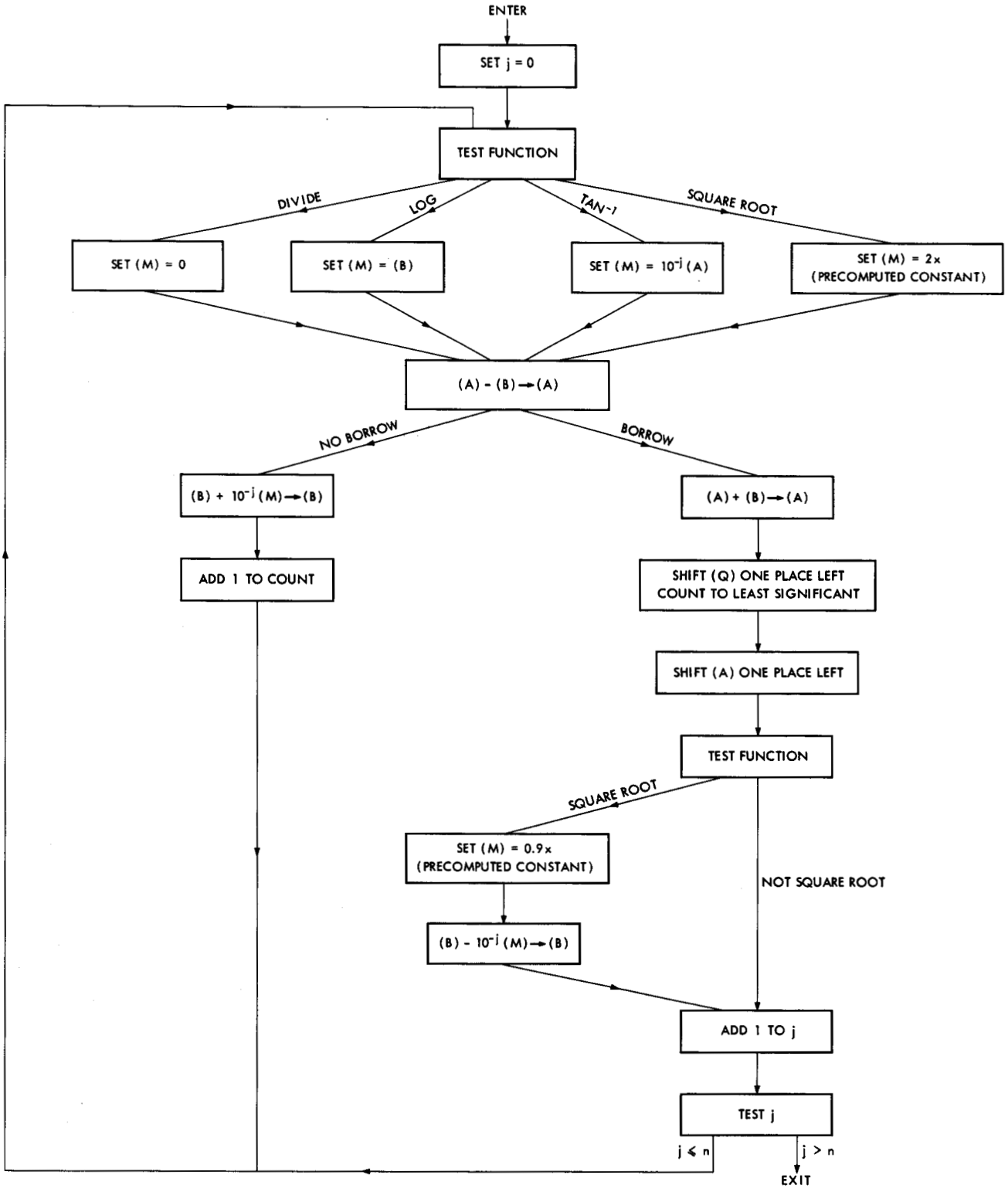


Figure 3 **Flow diagram for modified divider.**
Initially A contains y and B contains x. The pseudo quotient is obtained in Q. This routine is used for 1) division, 2) Part 1 of $\log[1 + (y/x)]$ calculation, 3) Part 1 of $\tan^{-1}(y/x)$ calculation and 4) $\sqrt{y/x}$ calculation.

• *Part 1 of calculation*

The basic idea is to suppose that

$$y - x\left[\prod_{k=0}^{j-1}(1+10^{-k})^{q_k} - 1\right]$$

has been calculated, where $q_0 \cdots q_{j-1}$ are digits that have already been chosen, and that they have been chosen in such a way as to make this expression as small as possible. It is now required to find the next digit q_j. For this, successive calculations of

$$y_a^{(j)} = y - x\left\{\left[\prod_{k=0}^{j-1}(1+10^{-k})^{q_k}\right](1+10^{-j})^a - 1\right\} \quad (3)$$

for $a = 0, 1, 2 \cdots q_j$ are made. q_j is defined by

$$y_{q_j}^{(j)} \geqq 0 > y_{q_j+1}^{(j)}.$$

The idea is to try the effect of including further factors $(1 + 10^{-j})$, with the object of keeping $y_a^{(j)}$ positive but making it small.

It is convenient to define

$$x_a^{(j)} = x\left[\prod_{k=0}^{j-1}(1+10^{-k})^{q_k}\right](1+10^{-j})^a. \quad (4)$$

The successive y's and x's may be calculated from (3) and (4) by the recurrence relations

$$\begin{aligned} y_{a+1}^{(j)} &= y_a^{(j)} - 10^{-j}x_a^{(j)} \\ x_{a+1}^{(j)} &= x_a^{(j)} + 10^{-j}x_a^{(j)}. \end{aligned} \quad (5)$$

By design the y's get smaller and smaller. To keep accuracy, it is convenient to do what is done in the case of a true division and put

$$z_a^{(j)} = 10^j y_a^{(j)}. \quad (6)$$

Then for each j the recurrence relations become

$$\begin{aligned} z_{a+1}^{(j)} &= z_a^{(j)} - x_a^{(j)} \\ x_{a+1}^{(j)} &= x_a^{(j)} + 10^{-j}x_a^{(j)}, \end{aligned} \quad (7)$$

and these are the equations for the evaluation of q_j. q_j is defined by

$$z_{q_j}^{(j)} \geqq 0 > z_{q_j+1}^{(j)}. \quad (8)$$

This is clearly a pseudo division process. $z_a^{(j)}$ is the pseudo remainder and $x_a^{(j)}$ is the pseudo divisor. It only differs from a true division in that the pseudo divisor is being constantly updated by the addition of $10^{-j}x_a^{(j)}$ to it instead of being held constant.

When q_j has been found it is clear that the initial conditions for the evaluation of q_{j+1} are

$$\begin{aligned} z_0^{(j+1)} &= 10^{j+1}y_0^{(j+1)} = 10^{j+1}y_{q_j}^{(j)} = 10z_{q_j} \\ x_0^{(j+1)} &= x_{q_j}^{(j)}. \end{aligned} \quad (9)$$

Hence, when q_j has been found, the process continues for the extraction of q_{j+1} with the pseudo remainder $z_{q_j}^{(j)}$ multiplied by 10. This exactly resembles what happens in the case of a true division.

When the process is first started it is also clear that

$$\begin{aligned} z_0^{(0)} &= y \\ x_0^{(0)} &= x. \end{aligned} \quad (10)$$

Hence, the following algorithm is established:

If y is divided by x using a long division repeated subtraction process wherein the divisor x is continually updated by having $10^{-j}x$ added to it during the formation of the quotient digit, q_j, then the pseudo quotient $q_0, q_1, q_2 \cdots$ is such that

$$\log[1 + (y/x)] = \sum_{j=0} q_j \log(1 + 10^{-j}). \quad (11)$$

The flow chart in Fig. 3 contains this process and shows it explicitly. It is identical for this case, with that shown in Fig. 1 for a true division except for the provision of the modifier register M as shown in Fig. 2, whose contents update the divisor. Each time a subtraction is performed the modifier is set with the divisor itself, so that the recurrence relations (7) are satisfied.

• *Magnitudes of x and y*

It is desirable that all the quotient digits q_j should be less than 10. As in the case of a true division, this implies an upper restriction on y/x. Indeed, for $q_0 < 10$, we must have

$$y/x < 2^{10} - 1.$$

For subsequent digits the condition is automatically met because it is certainly met for a true division, and in the pseudo division the divisor is being continually increased. When y/x is small, the calculation approaches a true division and not surprisingly

$$\log[1 + (y/x)] \sim y/x.$$

• *Register lengths*

x and y are each n digit numbers with, it is supposed, their decimal points aligned. However, since the number contained in B grows during the calculation, B may have to be a register of length greater than n. The contents of B at the end of the calculation are, in fact,

$$x\prod_{k=0}(1+10^{-k})^{q_k}$$

$$= y + x \quad \text{by construction,}$$

so that it is obvious that a register of length $n + 1$ for B will suffice. The remainder register A, will also never have to contain a number greater than ten times that in B. Hence, the length of register A is made $n + 2$. Q is given a length of n, and the pseudo quotient is calculated to n digits. Considerations of accuracy show that it is not worthwhile calculating more digits. Table 1 shows a typical calculation, with the successive contents of registers shown explicitly.

• *Accuracy*

Approximations only occur when the shifted modifier is added to the divisor. However, it is easy to see the effect of the figures dropped in this shift. It will be supposed that the dropped figures are used for rounding in the addition. The first round-off errors occur when q_1 is being calculated. Let δx_k be the rounding error introduced at the k^{th} rounding. Then, when q_1 has been formed, (A) will be in error by

$$\delta y_1 = (q_1 - 1)\delta x_1 + (q_1 - 2)\delta x_2 + \cdots \tag{12}$$

while (B) will be in error by

$$\delta x_1 + \delta x_2 + \cdots + \delta x_{q_1} . \tag{13}$$

Before q_2 is formed,(A) are shifted left. Hence, if we consider only errors that are caused while q_1 is being formed, there will be an error of

$$\delta y_2 = 10[(q_1 - 1)\delta x_1 + (q_1 - 2)\delta x_2 + \cdots] + q_2(\delta x_1 + \cdots + \delta x_{q_1})$$

in A when q_2 has been found and so on.

When q_{n-1} has been formed, there will be an error in A of

$$\delta y_{n-1} = 10^{n-2}[(q_1 - 1)\delta x_1 + (q_1 - 2)\delta x_2 + \cdots] + (10^{n-3}q_2 + 10^{n-4}q_3 + \cdots + q_{n-1}) \times (\delta x_1 + \delta x_2 + \cdots + \delta x_{q_1}) . \tag{14}$$

In the worst case $q_1 = q_2 = q_3 = \cdots = 9$

$$\delta x_1 = \delta x_2 = \cdots = \delta .$$

It is then found that

$$\delta y_{n-1} = 45 \times 10^{n-2}\delta . \tag{15}$$

The effect of round-off errors that occur when q_2 is formed is identical except that it is ten times as small, and so on. Thus, in the worst case, the effect of all rounding errors is to produce a total error in A, when q_{n-1} has been formed,

$$\delta y_{n-1} = 45 \times 10^{n-2}\delta(1 + 10^{-1} + 10^{-2} + \cdots) = 50 \times 10^{n-2}\delta . \tag{16}$$

When q_{n-1} has been formed,

$$(B) \sim y + x .$$

q_{n-1} is found effectively by the true division of (A) by (B) since, at this stage the effect of the modifier is negligible. Clearly, the error in (A) predominates, and due to this cause there is an error in q_{n-1} of

$$50 \times 10^{n-2}[\delta/(y + x)] . \tag{17}$$

Because of the rounding in the addition and because it is supposed that at least one of y or x contains n significant digits,

$$[\delta/(y + x)] < 5 \times 10^{-n} . \tag{18}$$

The worst case error in q_{n-1} is, therefore, less than 2.5. Thus, the last digit of the quotient will never be in error by more than 2.5. Due to the fact, however, that it is likely for round-off errors to compensate, it is usual for q_{n-1} to be exact in typical calculations. (The probability of this happening may, in fact, be

Table 1 **Example of formation of log[1 + (y/x)].** $y = 67719$ and $x = 21608$.

j	B	A	Count	Q
0	21608	67719		
	21608	21608	1	
	43216	46111		
	43216	43216		
	86432	2895	2	
1	86432	28950		00002
2	86432	289500		00020
	864	86432		
	87296	203068	1	
	873	87296		
	88169	115772	2	
	882	88169		
	89051	27603	3	
3	89051	276030		00203
	89	89051		
	89140	186979	1	
	89	89140		
	89229	97839	2	
	89	89229		
	89318	8610	3	
4	89318	86100		02033
				20330

A pseudo multiplication then causes the evaluation of $\log_e[1 + (y/x)] = 2 \log_e 2 + 3 \log_e 1.01 + 3 \log_e 1.001$ *giving* $\log_e[1 + (y/x)] = 1.4192$. *If more digits of the pseudo quotient are calculated a value for* $\log_e[1 + (y/x)] = 1.419240$ *is obtained which is in fact exact.*

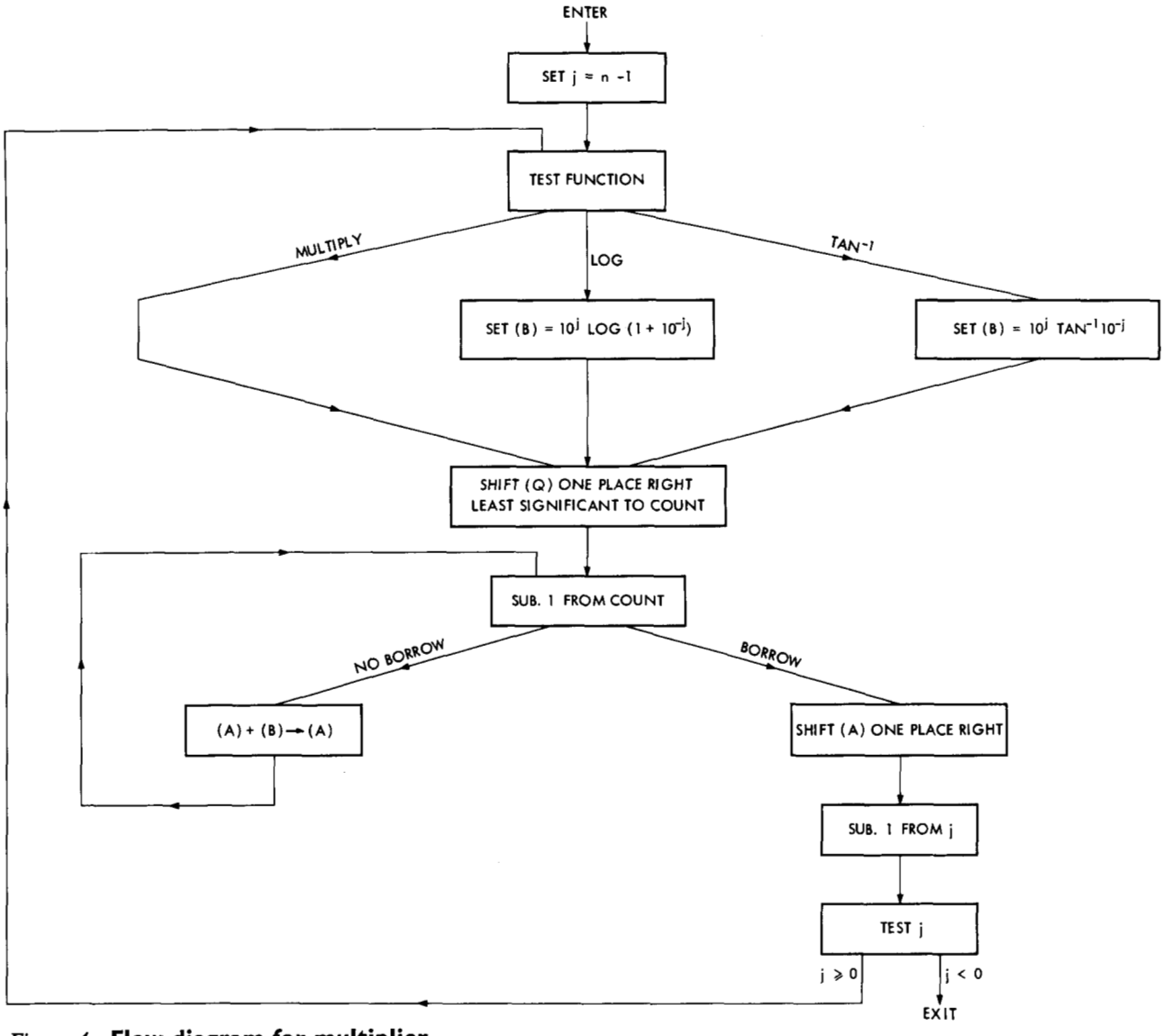


Figure 4 **Flow diagram for multiplier.**
Initially multiplier is in Q, and B contains multiplicand. The product is obtained in A. This routine is used for 1) multiplication, 2) Part 2 of log[1 + (y/x)] calculation and 3) Part 2 of $\tan^{-1}(y/x)$ calculation.

calculated.) Therefore, the method is inherently an accurate one.

• *Part 2 of calculation*

The second part of the calculation consists of finding $\log[1 + (y/x)]$ from (11). The base to which the logarithm is calculated is determined by the base to which the stored constants $\log(1+10^{-j})$ are calculated. It should be observed that $\log_e(1 + 10^{-j}) \sim 10^{-j}$.

Thus, the decimal number $q_0.q_1q_2q_3 \cdots$ is already a fairly close approximation to $\log_e[1 + (y/x)]$. Indeed, if working is carried to n figures, only about the first half of the q_j cause corrections to be made.

The formation of $\log[1 + (y/x)]$ from (11) clearly resembles a multiplication. The number in Q is the multiplier, while the multiplicand is given the value $\log(1 + 10^{-j})$ while multiplication by the digit q_j is taking place. It is, therefore, convenient to use a pseudo multiplier for this operation. The operation is identical to an ordinary multiplier save that the multiplicand is set from some read-only store to the required value, as each digit of the multiplier is processed. Figure 4 shows the process explicitly and also shows how it is combined with a true multiplication. It is good enough, of course, to set

$$10^j \log(1 + 10^{-j}) = 1 \tag{19}$$

while the least significant half of the multiplier digits are processed. This economizes on the number of stored constants required.

The intention is that this routine should be no more than a particular mode of the multiplication routine. It is, of course, true that the pseudo multiplication and division could be combined into a single process. They have, however, been split so that they may be combined with true multiplication and division.

• *Execution times*

The time to form $\log[1 + (y/x)]$ using this method will be about three multiply times, and, of course, this includes the division of y by x which conventionally would have to be done before the calculation of the logarithm. This assertion may be slightly unfair since it assumes that multiplication would be performed by a repeated addition process, without the use of any tricks for speeding it up. However, the pseudo processes described are also performed without any tricks and it may be that similar tricks are applicable in all cases.

• *Log z*

If it is wished to evaluate $\log z$, where z is an n digit number with the decimal point to the left, then the complement of z should be put in A and z itself into B before the process is begun.

Then $y = 1 - z$ so that

$$\log[1 + (y/x)] = \log[1 + (1 - z)/z] = -\log z \tag{20}$$

y/x will not be too large if z is not too small. This makes an excellent method of finding $\log z$ if $0.1 \leqq z < 1$, and so this method can be used for finding the logarithm of the fractional part of a floating point number.

• *To form* $\tan^{-1}(y/x)$

The method used is one that resembles Brigg's method, but it is applied to complex logarithms. The central idea is to find integers q_j such that

$$(x + iy) \prod_{j=0} (1 - i10^{-j})^{q_j} = R\,, \tag{21}$$

where x and y are n digit positive numbers and R is real. When this is done

$$\log(x + iy) = \log R - \sum_{j=0} q_j \log(1 - i10^{-j})\,. \tag{22}$$

The imaginary part of this equation gives

$$\tan^{-1}(y/x) = \sum_{j=0} q_j \tan^{-1} 10^{-j}\,. \tag{23}$$

Calculation is again split into two parts. In the first, integers q_j are calculated by a pseudo division process. In the second $\tan^{-1}(y/x)$ is found by a pseudo multiplication, using stored values for $\tan^{-1} 10^{-j}$.

• *Part 1 of calculation*

The idea is to suppose that $(x + iy) \prod_{k=0}^{j-1} (1 - i10^{-k})^{q_k}$ has been calculated, where $q_0 \cdots q_{j-1}$ are digits that have already been chosen in such a way as to make the imaginary part of this expression as small as possible, and to consider what is required to find q_j. For this, successive calculations of

$$x_a^{(j)} + iy_a^{(j)} = (x + iy) \prod_{k=0}^{j-1} (1 - i10^{-k})^{q_k} (1 - i10^{-j})^a \tag{24}$$

are made for $a = 0, 1, 2 \cdots q_j$. q_j is defined by

$$y_{q_j}^{(j)} \geqq 0 > y_{q_j}^{(j)} y_{q_j+1}^{(j)}\,.$$

That is, $y^{(j)}$ must be made as small as possible, while keeping it positive.

$$\begin{aligned} y_{a+1}^{(j)} &= y_a^{(j)} - 10^{-j} x_a^{(j)} \\ x_{a+1}^{(j)} &= x_a^{(j)} + 10^{-j} y_a^{(j)}\,. \end{aligned} \tag{25}$$

By design, the y's get smaller as the process is continued. Hence, to keep accuracy, it is convenient to write

$$z_a^{(j)} = 10^j y_a^{(j)}\,. \tag{26}$$

The recurrence relations then become

$$\begin{aligned} z_{a+1}^{(j)} &= z_a^{(j)} - x_a^{(j)} \\ x_{a+1}^{(j)} &= x_a^{(j)} + 10^{-2j} z_a^{(j)}\,. \end{aligned} \tag{27}$$

These are obeyed repeatedly until

$$z_{q_j}^{(j)} \geqq 0 > z_{q_j+1}^{(j)}\,. \tag{28}$$

This is again a pseudo division process with $z_a^{(j)}$ the pseudo remainder and $x_a^{(j)}$ the pseudo divisor. In this case, however, the pseudo divisor is repeatedly updated by adding to it $10^{-2j} z_a^{(j)}$. As in the case of logarithms, it is clear that when the iteration for q_{j+1} is started, initial conditions are

$$\begin{aligned} z_0^{(j+1)} &= 10 z_{q_j}^{(j)} \\ x_0^{(j+1)} &= x_{q_j}^{(j)} \end{aligned} \tag{29}$$

so that the pseudo remainder has to be shifted one place to the left. Also, at the beginning of the entire process

$$\begin{aligned} z_0^{(0)} &= y \\ x_0^{(0)} &= x\,. \end{aligned} \tag{30}$$

Hence, the following algorithm is established. If y is divided by x using a long division repeated subtraction process wherein the divisor is continually updated by having $10^{-2j} z$ added to it (z the remainder) during the formation of the quotient digit q_j, then the pseudo quotient $q_0 q_1 \cdots$ is such that

$$\tan^{-1}(y/x) = \sum_{j=0} q_j \tan^{-1} 10^{-j}\,. \tag{31}$$

The flow chart in Fig. 3 shows the process explicitly and it only differs from that for the formation of

$\log[1 + (y/x)]$ in that (M) is set equal to $10^{-j}(A)$ instead of to (B).

• *Magnitudes of x and y*

Since $0 \leqq \tan^{-1}(y/x) \leqq \pi/2$ it is obvious $q_0 \leqq 2$.

Also, since the divisor is continually being increased, it is clear that all the other pseudo quotient digits are less than 10. Hence, all the quotient digits are less than 10 and there are no restrictions on the ratio y/x.

Table 2 **Example of formation of $\tan^{-1}(y/x)$.** $y = 30912$ and $x = 59438$.

j	B	A	Count	Q
0	59438	30912		
1	59438	309120		00000
	3091	59438		
	62529	249682	1	
	2497	62529		
	65026	187153	2	
	1872	65026		
	66898	122127	3	
	1221	66898		
	68119	55229	4	
2	68119	552290		00004
	55	68119		
	68174	484171	1	
	48	68174		
	68222	415997	2	
	42	68222		
	68264	347775	3	
	35	68264		
	68299	279511	4	
	28	68299		
	68327	211212	5	
	21	68327		
	68348	142885	6	
	14	68348		
	68362	74537	7	
	7	68362		
	68369	6175	8	
3	68369	61750		00048
4	68369	617500		00480
		68369		
		549131	1	

At this stage it becomes a straight division process giving

68369	2179	9	
			04809

A pseudo multiplication then causes the evaluation of $\tan^{-1}(y/x) = 4 \tan^{-1} 0.1 + 8 \tan^{-1} 0.01 + 9 \tan^{-1} 0.0001$ *giving* $\tan^{-1}(y/x) = 0.4796$. *If more digits of the pseudo quotient are calculated* $\tan^{-1}(y/x) = 0.479578$. *The exact answer is 0.479574.*

• *Register lengths*

x and y are each n digit numbers with their decimal points aligned. At the end of the process, B will contain approximately the number R from (21), and

$$R = |x + iy| \prod_{j=0} |1 - i10^{-j}|^{q_j} \tag{32}$$

from which it may be shown

$$R \leqq 2|x + iy| \leqq 2\sqrt{2} \max(x, y) . \tag{33}$$

Hence, a register of length $n + 1$ will certainly accommodate R. Therefore, as for the logarithm routine, it is sufficient to make B a register of length $n + 1$ and A a register of length $n + 2$.

Table 2 shows a typical calculation set out in detail.

• *Accuracy*

The discussion of accuracy is virtually the same as the discussion for logarithms, since, again the only errors that occur, occur when the contents of the modifier are shifted and added to update the divisor. Errors occur because of the figures dropped. The previous discussion applies to all pseudo division processes of this type.

The result, therefore, is that the worst possible error in the last digit of the quotient q_{n-1} is

$$50 \times 10^{n-2}\delta/R . \tag{34}$$

In this case too, therefore, the worst possible error is of 2.5 in the last digit of the pseudo quotient, and, of course, owing to the cancellation of errors, it is usual for q_{n-1} to be exact.

• *Part 2 of calculation*

$\mathrm{Tan}^{-1}(y/x)$ is now calculated from (31). This is done by means of a pseudo multiplication and is exactly as described for the logarithm and is shown in Fig. 4. However, the multiplicand is set to $10^j \tan^{-1} 10^{-j}$ for successive values of j instead of to $10^j \log(1 + 10^{-j})$. For 2/3 of the values of j it is accurate enough to set

$$10^j \tan^{-1} 10^{-j} = 1 \tag{35}$$

and this, therefore, saves stored constants.

• *Execution times*

The time to form $\tan^{-1}(y/x)$ using this method is about three multiply times and, of course, this one also gains a division.

To form $\tan^{-1} z$, it is a very simple matter to put

$$\begin{aligned} y &= 10^a z \\ x &= 10^a \end{aligned} \tag{36}$$

for a suitable scale factor 10^a, though then, of course, the extra division facility is wasted.

• *Assessment*

The two routines that have been described give accurate values of $\log[1 + (y/x)]$ and $\tan^{-1}(y/x)$ in three multiply times which is quicker than any of the current subroutine methods. They have the advantage of simplicity since they merely employ pseudo dividers and pseudo multipliers. Moreover, they are not unduly expensive with stored constants.

• *Square roots* $\sqrt{y/x}$

It is also tempting to use the modified divider that has been described for the extraction of square roots. This can be done at the expense of slightly more complication. The method is the standard digit by digit one. The flow diagram is shown also in Fig. 3. It is seen that the only difference from the previous routines is that the modifier is set to a constant, while the pseudo divisor is altered between the extraction of successive digits.

Integers q_j are found such that

$$y = x\left[\sum_{j=0} q_j 10^{-j}\right]^2 \tag{37}$$

so that

$$\sqrt{y/x} = \sum_{j=0} q_j 10^{-j}. \tag{38}$$

Here, it is supposed that $q_0 \cdots q_{j-1}$ have been found and it is required to find q_j. This is done by calculating successively

$$y_a^{(j)} = y - x\left[\sum_{k=0}^{j-1} q_k 10^{-k} + a10^{-j}\right]^2 \tag{39}$$

for $a = 0, 1, 2 \cdots$. The object again is to make $y_a^{(j)}$ as small as possible but to keep it positive. Define

$$x_a^{(j)} = 2x\left[\sum_{k=0}^{j-1} q_k 10^{-k} + a10^{-j}\right] + x10^{-j}. \tag{40}$$

This can be calculated from the recurrence relation

$$x_{a+1}^{(j)} = x_a^{(j)} + 2x10^{-j}. \tag{41}$$

In terms of $x_a^{(j)}$, it may be seen that

$$y_{a+1}^{(j)} = y_a^{(j)} - 10^{-j} x_a^{(j)}. \tag{42}$$

As before, for accuracy's sake, it is convenient to put $z_a^{(j)} = 10^j y_a^{(j)}$ and so get the recurrence relations

$$\begin{aligned} z_{a+1}^{(j)} &= z_a^{(j)} - x_a^{(j)} \\ x_{a+1}^{(j)} &= x_a^{(j)} + 2x10^{-j}. \end{aligned} \tag{43}$$

These equations are iterated until $z_a^{(j)}$ goes negative. That is, q_j is defined by

$$z_{q_j}^{(j)} \geqq 0 > z_{q_j+1}^{(j)}. \tag{44}$$

These equations are again like those that arise in a division process. $z_a^{(j)}$ is the remainder, and $x_a^{(j)}$ is the pseudo divisor, which is continually being updated by the addition of the constant $2x$, shifted j places.

The initial conditions for the extraction of q_{j+1} are, however, more complicated. Clearly

$$z_0^{(j+1)} = 10 z_{q_j}^{(j)}.$$

However

$$x_0^{(j+1)} = x_{q_j}^{(j)} - x10^{-j} + x10^{-j-1}. \tag{45}$$

Thus, before the calculation of q_{j+1} the pseudo divisor must be updated by subtracting from it $0.9 \times 10^{-j}x$, and this constitutes the extra complication.

At the beginning of the process

$$\begin{aligned} y_0^{(0)} &= y \\ x_0^{(0)} &= x. \end{aligned} \tag{46}$$

Hence, the following algorithm is obtained. If a pseudo division of y by x is performed, where the modifier is held constant at $2x$, and if between the calculation of successive digits $0.9 \times 10^{-j}x$ is subtracted from the divisor, then the quotient is $\sqrt{y/x}$. Of course the numbers $2x$ and $0.9x$ can be calculated before the process is begun.

This routine probably looks more familiar if it is performed with $x = 1$. It should be observed that it is very easy to set the modifier at $2x$ in a binary machine and also that in a binary machine the factor 0.9 that occurs in the subtraction is replaced by a factor 0.1.

• *Magnitudes of numbers*

It is convenient to suppose that y and x are given as numbers with n significant digits and that either their decimal points are aligned or else misaligned by one digit. If they are aligned this implies that $10 > y/x > 1/10$. If they are misaligned, y is shifted one place left to align them before calculation is begun. In the latter

event $100 > y/x > 1$. In either case all the quotient digits will be less than 10.

• *Sizes of registers*

At the end of the process B will contain approximately

$$2\sqrt{xy}\,. \tag{47}$$

$x \leqq C$, the largest n digit number $y \leqq 10C$.

Hence,

$$2\sqrt{xy} \leqq 2\sqrt{10}\,C\,. \tag{48}$$

Hence, as before, a register of length $n + 1$ will suffice for B and a register of $n + 2$ for A. A typical calculation is shown in Table 3.

• *Accuracy*

Errors occur when the shifted modifier updates the divisor and figures are dropped, and also, when the divisor is updated between the extraction of successive digits. The first errors are those that have been encountered previously. The others are of the same type but occur in the worst case one tenth as often. Hence, from (16) there is a worst case error of about

$$55 \times 10^{n-2}\delta \tag{49}$$

in A when q_{n-1} has been formed.

q_{n-1} was obtained essentially by a division of (A) by (B). B contains at this time $2\sqrt{yx}$. Hence, the worst case error in q_{n-1}

$$\sim (55 \times 10^{n-2}/2)(\delta/\sqrt{xy})\,. \tag{50}$$

Since, by design x and y each contain, at least, n significant digits

$$\delta/\sqrt{xy} < 5 \times 10^{-n}\,. \tag{51}$$

This leads to a worst case error in q_{n-1} of 1.5 and

Table 3 **Example of formation of square root $\sqrt{y/x}$.** $y = 77208, x = 16804, 2x = 33608, 0.9x = 15124$.

j	B	A	Count	Q
0	16804	77208		00000
	33608	16804		
	50412	60404	1	
	33608	50412		
	84020	9992	2	
	15124−			
1	68896	99920		00002
	3361	68896		
	72257	31024	1	
	1512−			
2	70745	310240		00021
	336	70745		
	71081	239495	1	
	336	71081		
	71417	168414	2	
	336	71417		
	71753	96997	3	
	336	71753		
	72089	25244	4	
	151−			
3	71938	252440		00214
	34	71938		
	71972	180502	1	
	34	71972		
	72006	108530	2	
	34	72006		
	72040	36524	3	
	15−			
4	72025	365240		02143
	3	72025		
	72028	293215	1	
	3	72028		
	72031	221187	2	
	3	72031		
	72034	149156	3	
	3	72034		
	72037	77122	4	
	3	72037		
	72040	5085	5	
				21435

The answer is therefore 2.1435.

If the process is continued it gives 2.143507, *which happens to be exact. Note that it is assumed y and x have their decimal points aligned.*

so this routine has the same kind of accuracy as the others.

• *Conclusion*

It is, therefore, possible to make a simple pseudo divider and a simple pseudo multiplier which between them, operating in different modes, can compute y/x, $\log[1 + (y/x)]$, $\tan^{-1}(y/x)$, $\sqrt{y/x}$.

Section 2: Multiplication

The processes that have been described in Section 1 may be reversed. This reversal leads to methods for forming exponentials, tangents and squares. In the reverse processes, multiplications become divisions and divisions multiplications. In the multiplications that result, it is naturally expected that the least significant digit of the multiplier will be processed first and that the answer will be obtained as the ratio of the contents of registers A and B. This implies that an ultimate extra division is necessary.

It transpires, however, that accurate methods for forming exponentials and squares can be devised that avoid this final division, but in the multiplications that they contain, the order of multiplication is changed and the most significant digit of the multiplier is processed first. In the case of the tangent, however, the method must be the exact reverse of the method for the inverse tangent. This makes the methods slightly different from each other.

• *To form exponentials*

The method to be described enables $x(e^p - 1)$ to be calculated for given positive p and x. p is expressed as

$$p = \sum_{j=0} q_j \log(1 + 10^{-j}), \tag{52}$$

for integers q_j. Then, clearly

$$x(e^p - 1) = x\left[\prod_{j=0} (1 + 10^{-j})^{q_j} - 1\right]. \tag{53}$$

If p were negative, it would be possible to make an expansion of p in terms of $\log(1 - 10^{-j})$. This would be the reverse process of what would be done in finding $\log[1 - (y/x)]$ for positive x and y. This was not treated explicitly in the first section of the paper, but the reader should observe that the only difference for that case is that the pseudo divisor should be updated by a subtraction rather than by an addition.

In floating point applications, it is probably sufficient, however, for p to be positive and so only this case will be dealt with now.

The calculation is split into two parts. In the first, integers q_j are found by a pseudo division, and in the second the exponential is evaluated by a pseudo multiplication.

• *Part 1*

To find integers q_j a division of p is made. The divisor is set to $\log(1 + 10^{-j})$ from a read only store while the digit q_j is being formed. Figure 5 shows the process explicitly, and it is clearly the reverse of what is shown in Fig. 4. Of course, constants $10^j \log(1 + 10^{-j})$ are taken from the same store.

• *Magnitude of p*

It is necessary to have all digits q_j less than 10. This means that $p < 10 \log 2$ for $q_0 < 10$, while this condition is met automatically for other digits since

$$\log(1 + 10^{-j}) < 10 \log(1 + 10^{-j-1}) . \tag{54}$$

p and the constants $10^j \log(1 + 10^{-j})$ must have their decimal points aligned. The constants are stored as n digit numbers, the decimal point being to the left. Hence, the number p must be shifted if necessary before the process is begun. n pseudo quotient digits are calculated. The accuracy to which p is known will not warrant the calculation of further digits.

• *Part 2*

To calculate

$$x\left[\prod_{j=0} (1 + 10^{-j})^{q_j} - 1\right] \tag{55}$$

a pseudo multiplication is performed using $q_0 \cdots q_{n-1}$ as the pseudo multiplier, and starting with the most significant digit q_0.

For this, suppose that

$$x\left[\prod_{k=0}^{j-1} (1 + 10^{-k})^{q_k} - 1\right] \tag{56}$$

has already been calculated and that it is now required to introduce a further factor $(1 + 10^{-j})^{q_j}$. Define

$$\begin{aligned} y_a^{(j)} &= x\left[\prod_{k=0}^{j-1} (1 + 10^{-k})^{q_k}(1 + 10^{-j})^a - 1\right] \\ x_a^{(j)} &= x\left[\prod_{k=0}^{j-1} (1 + 10^{-k})^{q_k}(1 + 10^{-j})^a\right]. \end{aligned} \tag{57}$$

Then, for successive a's, the recurrence relations

$$\begin{aligned} y_{a+1}^{(j)} &= y_a^{(j)} + 10^{-j} x_a^{(j)} \\ x_{a+1}^{(j)} &= x_a^{(j)} + 10^{-j} x_a^{(j)} \end{aligned} \tag{58}$$

are obtained and these are iterated for $a = 0 \cdots q_j - 1$.

It is convenient to make

$$z_a^{(j)} = 10^j y_a^{(j)} . \tag{59}$$

Then the recurrence relations become

$$\begin{aligned} z_{a+1}^{(j)} &= z_a^{(j)} + x_a^{(j)} \\ x_{a+1}^{(j)} &= x_a^{(j)} + 10^{-j} x_a^{(j)} . \end{aligned} \tag{60}$$

These equations now resemble a multiplication; $z_a^{(j)}$ is the partial sum, while $x_a^{(j)}$ is the multiplicand which is being continually updated by the addition of itself shifted j places. $q_0 \cdots q_{n-1}$ is the multiplier.

When q_j has been processed, initial conditions for the processing of q_{j+1} are

$$z_0^{(j+1)} = 10z_{q_j}^{(j)}$$
$$x_0^{(j+1)} = x_{q_j}^{(j)}. \tag{61}$$

There is, therefore, a shift left of the partial product between the processing of successive digits and this is exactly what happens in the case of a true multiplication where the most significant digit is processed first.

When the process is started

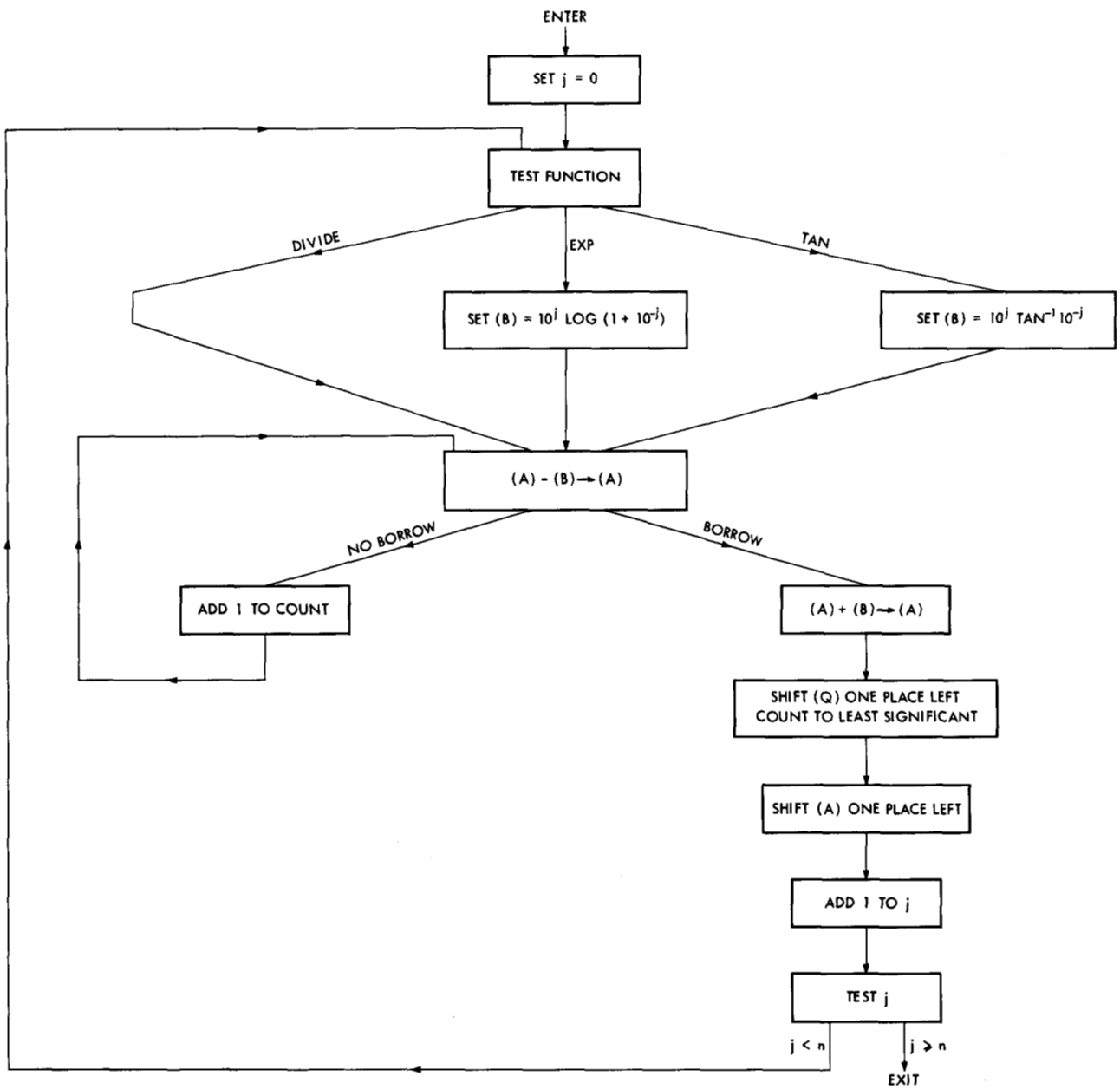
$$z_0^{(0)} = 0$$
$$x_0^{(0)} = x. \tag{62}$$

Hence, a pseudo multiplication is being made, where x is the pseudo multiplicand that is repeatedly updated, and $q_0 \cdots q_{n-1}$ is the multiplier.

The flow chart in Fig. 6 shows this process. It is identical with that for a true multiplication (multiplication from the left) except for the provision of the modifier register which updates the pseudo multiplicand after each successive addition.

If it is required to calculate xe^p rather than $x(e^p - 1)$, then the partial sum register A should initially be set at x. Of course, x may be set at one for the calculation of e^p but use of the method's full power is attractive.

Figure 5 **Flow diagram for divider.**
A contains dividend and B contains divisor. The quotient appears in Q. This routine is used for 1) division, 2) Part 1 of $x(e^p - 1)$ calculation and 3) Part 1 of tan x calculation.



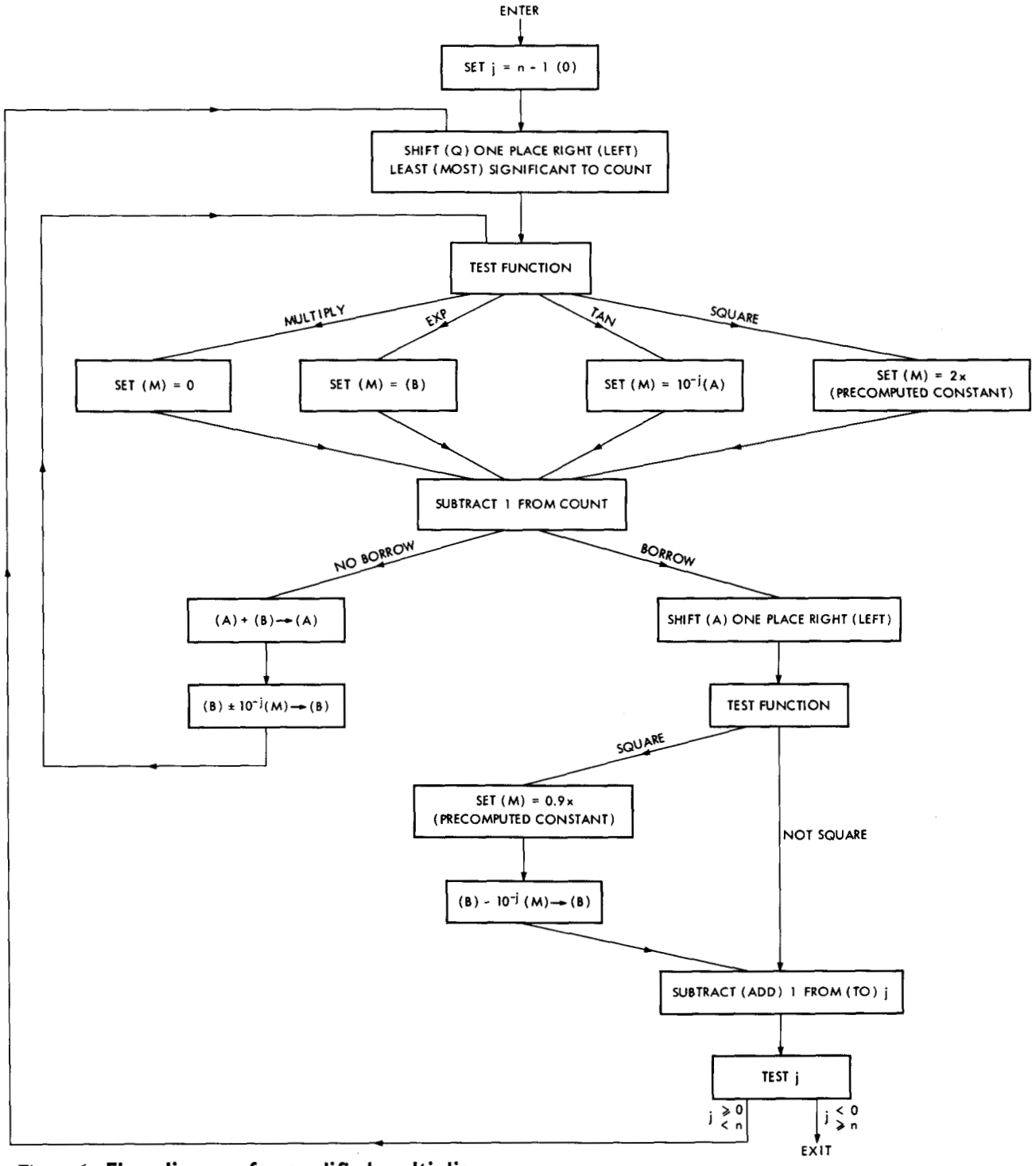


Figure 6 **Flow diagram for modified multiplier.**
B contains pseudo multiplicand, Q contains pseudo multiplier, and A contains pseudo product. This routine is used for 1) multiplication, 2) Part 2 of $x(e^p - 1)$ calculation, 3) Part 2 of tan p calculation and 4) xq^2 calculation. Where choice is indicated, multiply and tangent routines employ the first possibility, while exp and square routines employ the second.

• *Register sizes*

As the calculation progresses, the size of the pseudo multiplicand increases. Eventually it will approximately equal xe^p, and so care must be taken to see that register B does not overflow. In an application where floating point arithmetic is used, it is likely that x will be a number with n significant digits and that $1 \leqq e^p < 10$. In this case B should be a register of length $n + 1$.

Table 4 **Example of formation of $x(e^p - 1)$.** $x = 21608$ and $p = 1.4192$. *An initial pseudo division of p gives (Q) = 20330. This pseudo division expresses p as* $p = 2 \log_e 2 + 3 \log_e 1.01 + 3 \log_e 1.001$.

j	B	A	Count	Q
			2	03300
0	21608	00000		
	21608	21608		
	43216	21608	1	
	43216	43216		
	86432	64824	0	
1	86432	648240	0	33000
2	86432	6482400	3	30000
	864	86432		
	87296	6568832	2	
	873	87296		
	88169	6656128	1	
	882	88169		
	89051	6744297	0	
3	89051	67442970	3	00000
	89	89051		
	89140	67532021	2	
	89	89140		
	89229	67621161	1	
	89	89229		
	89318	67710390	0	

The method, therefore, gives the answer 67710. *The correct answer is* 677154. *The decimal point is aligned with that in the number x. There appears to be a large error. However, this disappears if (Q) are calculated to 6 figures rather than 5 and the process is continued one stage further. If p is known, however, only to the number of figures shown, this further accuracy is spurious.*

At the end of the process, A will contain a number of similar size, except that it has been shifted left $n - 1$ times. Hence, A must be a register of length $2n$. It is unfortunate that A has to be such a long register. However, it can be joined up with register Q as is often done, since the number of digits in Q decreases as the number in A increases.

• *Accuracy*

The number that is in A at the end of the process is the required answer and it is necessary to find how many digits of it are accurate. Inaccuracies occur in the pseudo multiplication process, due to the dropping of figures when the pseudo multiplicand is updated. The way in which these affect the contents of A is exactly the same way in which the contents of A were affected in the corresponding pseudo division process. Hence, from (16), the error in (A) when q_{n-1} has been processed is at worst

$$50 \times 10^{n-2}\delta. \tag{63}$$

Because of the rounding in the modification, $\delta = 0.5$. Thus, there is an error of about 2.5 in the $n - 1^{th}$ digit from the right in A. (A) are, therefore, shifted $n - 1$ places to the right. A now contains a number of length $n + 1$ with its decimal point aligned with that in the number x. Its least significant digit is in error by at most 3.

Inaccuracies also occur because in the preliminary division only n digits of the pseudo quotient have been calculated. However, as has been explained, this is only the accuracy that is warranted and this inaccuracy is inherent in the number system used.

The method, therefore, is inherently an accurate one. Table 4 shows a typical calculation with the successive contents of registers shown.

• *Execution times*

xe^p or alternatively $x(e^p - 1)$ is formed in approximately three multiply times, so this is certainly a fast method.

• *To form tangents*

The method to be described enables tan p to be calculated. The answer is obtained as a ratio and as has been explained, a final division is necessary.

It is supposed that $0 \leqq p \leqq \pi/2$. p is expressed as

$$p = \sum_{j=0} q_j \tan^{-1} 10^{-j} \tag{64}$$

for integers q_j. An evaluation of

$$x + iy = R \prod_{j=0} (1 + i10^{-j})^{q_j} \tag{65}$$

is then made for some real R. Clearly then

$$\tan p = y/x. \tag{66}$$

The integers q_j are obtained by means of a pseudo division, and x and y are then calculated by means of a pseudo multiplication.

• *Part 1*

The pseudo division is shown in Fig. 5. It is identical with the corresponding process used in forming the

exponential, except constants $10^j \tan^{-1} 10^{-j}$ are used instead of $10^j \log(1 + 10^{-j})$.

• *Magnitudes of p*

Since $p \leqq \pi/2$, $q_0 \leqq 2$; therefore since

$$\tan^{-1} 10^{-j} < 10 \tan^{-1} 10^{-j-1} \tag{67}$$

all other digits q_j are less than ten.

As in the case of the exponential, p and the constants $10^j \tan^{-1} 10^{-j}$ must have their decimal points aligned. The number p is, therefore, shifted if necessary. n pseudo quotient digits are calculated.

• *Part 2*

To calculate

$$R \prod_{j=0} (1 + i10^{-j})^{q_j} \tag{68}$$

a pseudo multiplication is performed using $q_0 \cdots q_{n-1}$ as the pseudo multiplier, and starting with the least significant digit q_{n-1}. Define

$$x_a^{(j)} + iy_a^{(j)} = R \prod_{k=j+1}^{n-1} (1 + i10^{-k})^{q_k}(1 + i10^{-j})^a \tag{69}$$

and put $z_a^{(j)} = 10^j y_a^{(j)}$. (70)

Then for successive values of a, the recurrence relations

$$\begin{aligned} z_{a+1}^{(j)} &= z_a^{(j)} + x_a^{(j)} \\ x_{a+1}^{(j)} &= x_a^{(j)} - 10^{-2j} z_a^{(j)} \end{aligned} \tag{71}$$

are obtained, and these are iterated for $a = 0 \cdots q_j - 1$.

Again these equations resemble a multiplication. $z_a^{(j)}$ is the partial product. $x_a^{(j)}$ is the pseudo multiplicand which is continually being updated by the subtraction of $z_a^{(j)}$ shifted $2j$ places. $q_0 \cdots q_{n-1}$ is the multiplier.

After q_j has been processed, q_{j-1} is processed and the initial conditions are

$$\begin{aligned} z_0^{(j-1)} &= 10^{-1} z_{q_j}^{(j)} \\ x_0^{(j-1)} &= x_{q_j}^{(j)} . \end{aligned} \tag{72}$$

At the beginning of the process

$$\begin{aligned} z_0^{(n-1)} &= 0 \\ x_0^{(n-1)} &= R . \end{aligned} \tag{73}$$

Thus, a pseudo multiplication of R by $q_0 \cdots q_{n-1}$ is made, starting with the least significant digit, q_{n-1}. This is exactly like a true multiplaction except for the updating by subtraction, of the multiplicand.

The flow chart for this process is shown in Fig. 6 also.

• *Register sizes*

The pseudo multiplicand decreases as the process continues. Initially, it contains R, which is arbitrary. For accuracy's sake, it is set to the largest convenient $n + 1$ digit number, and register B is made of length $n + 1$. A never has to contain a number more than ten times that in B, so A is made of length $n + 2$. An example is shown in Table 5.

• *Accuracy*

The contents of A and B at the end of the process are in error due to the dropping of figures when the shifted contents of M update the pseudo multiplicand.

The effect of this cause is easily seen. The final ratio of (B) to (A) will be $\tan(p + \delta_p)$ for some small δ_p, instead of $\tan p$. It is convenient to discuss errors in terms of δ_p. If the method for forming an inverse tangent is applied to (B) and (A), it will be found that step by step their contents approximate their con-

Table 5 **Example of formation of tan p.** $p = 0.4796$. *An initial pseudo division gives* $(Q) = 04809$. *This pseudo division expresses p as* $p = 4 \tan^{-1} 0.1 + 8 \tan^{-1} 0.01 + 9 \tan^{-1} 0.0001$. *(B) are initially set for convenience at* 100000.

j	B	A	Count	Q
4	100000	00000	9	00480
	------	-----	-	
	------	-----	-	
	100000	900000	0	
3	100000	90000	0	00048
2	100000	9000	8	00004
	1	100000		
	99999	109000	7	
	11	99999		
	99988	208999	6	
	21	99988		
	99967	308987	5	
	31	99967		
	99936	408954	4	
	41	99936		
	99895	508890	3	
	51	99895		
	99844	608785	2	
	61	99844		
	99783	708629	1	
	71	99783		
	99712	808412	0	

j	B	A	Count	Q
1	99712	80841	4	00000
	808	99712		
	98904	180553	3	
	1806	98904		
	97098	279457	2	
	2795	97098		
	94303	376555	1	
	3766	94303		
	90537	470858	0	
0	90537	47085	0	00000

A division of (A) by (B) leads to a value of tan p = 0.5201. The correct answer is 0.52010.

Values of sin p = 0.4614 and cos p = 0.8872 can also be obtained. The correct values are 0.46142 and 0.88725.

tents during the formation of the tangent. Actually, after digits $q_0 \cdots q_j$ have been formed in the inverse tangent process, the contents of B and A will exceed the corresponding contents in the tangent process by a factor $\prod_{k=0}^{j}(1 + 10^{-2k})^{q_k}$. However, rounding errors occur similarly in both processes. If compensating errors occur in the inverse process at exactly those places where errors occurred in the forward process, then $\tan^{-1}\tan(p + \delta_p)$ will be calculated as p, there being an error δ_p. From the discussion of errors for the inverse tangent process it is, therefore, clear that δ_p is at worst 2.5 in the least significant figure of p, and so this gives a measure of the error in the tangent process. In short, the dropping of figures when the modification is performed gives the same errors in the tangent as in the inverse tangent process.

• *Trigonometrical functions*

This method produces two numbers, x and y whose ratio is tan p. To obtain tan p a further division must be performed and obviously care has to be taken to see that x is not too small.

It is also possible to form sin p and cos p from

$$\sin p = \sqrt{y^2/(x^2 + y^2)}\cos p = \sqrt{x^2/(x^2 + y^2)}\,. \qquad (74)$$

These may be calculated by first squaring x and y and then using the square root process previously described. In a microprogram machine in particular, the necessary control for this is easy to provide.

If the sines and cosines are held in n digit stores with the decimal point one place from the left-hand end, errors due to the trigonometrical part of the method will never exceed 2.5 in the least significant digit place.

• *Execution times*

x and y are obtained in three multiply times. Tan p may be calculated in four times, while sin p and cos p will take approximately seven multiply times.

• *To form squares*

The square root method may also be reversed, to give a method for finding squares. This, of course, is not of much value but is mentioned here for interest's sake. It enables xq^2 to be calculated for given x and q.

A pseudo multiplication is performed. x is the initial pseudo multiplicand and q is the multiplier and multiplication is performed starting with the most significant digit of q. At each stage, the pseudo multiplicand is updated in exactly the way the pseudo divisor was updated in the square root process. The pseudo product is then xq^2. The proof of this follows almost exactly the proof for the square root method.

• *Register sizes*

It is convenient to suppose that x and q are numbers with n significant digits. It may then be shown that register B has to have length $n + 2$ to allow for the growth of the pseudo multiplicand. Register A has to have length $2n + 1$ to allow for the succession of $n - 1$ shifts left.

• *Accuracy*

The way in which errors occur is exactly the way errors occur in the square root routine. Hence, there is an error in (A) at worst of $55 \times 10^{n-2}\delta$ when q_{n-1} has been processed. Accordingly, at the end of the process the contents of A are shifted right $n - 1$ places, giving an error of not more than 3 in the least significant digit of (A), (A) being now a number with n, $n + 1$ or $n + 2$ significant digits.

• *Decimal points*

If it is assumed that the n digit numbers x and q have their decimal point one place from the left-hand end, that is $1 \leqq x, q < 10$, then the decimal point of xq^2 is aligned with them; that is $1 \leqq xq^2 < 1000$.

This routine is also shown in Fig. 6, and a worked example is included in Table 6.

• *Assessment*

This method enables xq^2 to be formed in two multiply times. It seems it may have some value in certain applications.

• *Conclusion*

The second section has shown how $x(e^p - 1)$, tan p, sin p, cos p, xq^2 may be calculated using pseudo multipliers and dividers.

All the elementary functions may, therefore, be

Table 6 **Example of formation of xq^2.** $x = 1.6804$, $2x = 3.3608$, $0.9x = 1.5124$, $q = 2.1435$.

j	B	A	Count	Q
			2	14350
0	16804	00000		
	33608	16804		
	50412	16804	1	
	33608	50412		
	84020	67216	0	
	15124−			
1	68896	672160	1	43500
	3361	68896		
	72257	741056	0	
	1512−			
2	70745	7410560	4	35000
	336	70745		
	71081	7481305	3	
	336	71081		
	71417	7552386	2	
	336	71417		
	71753	7623803	1	
	336	71753		
	72089	7695556	0	
	151−			
3	71938	76955560	3	50000
	34	71938		
	71972	77027498	2	
	34	71972		
	72006	77099470	1	
	34	72006		
	72040	77171476	0	
	15−			
4	72025	771714760	5	00000
	3	72025		
	72028	771786785	4	
	3	72028		
	72031	771858813	3	
	3	72031		
	72034	771930844	2	
	3	72034		
	72037	772002878	1	
	3	72037		
	72040	772074915		

The answer given is therefore 7.7207. *The correct answer is* 7.72075.

generated using the methods described. In a machine with microprogram or conventional control, it is likely that one general pseudo multiplier/divider routine would be constructed. This would have many branches in it, and the particular flow path required would be set up by the function decoder. This would enable multiplication, division and the calculation of the elementary functions to be performed very economically in terms of the number of microinstructions or of the conventional hardware required.

The methods described are as fast or faster than any conventional subroutine methods. These methods are extremely attractive for small machines where there is not, perhaps, space to provide conventional subroutines.

References

1. A. Ralston and H. S. Wilf, *Mathematical Methods for Digital Computers*, John Wiley and Sons, Inc., 1960.
2. J. H. Wensley, "A Class of Non-Analytical Iterative Processes," *The Computer Journal*, **1**, No. 4, 163 (Jan. 1959).

Received August 29, 1961