be explained by the fact that the schedule length approaches its lower limit as determined by the chain of tasks on the critical path. Adding additional processors after this point is reached will not reduce the schedule length. For the problems generated here, the schedule lengths for four or more processors remain about the same as for three processors. In most cases the branch-and-bound algorithm verified the optimality of the critical path solution with relatively little additional computation. $ASMAX$ is the maximum storage requested by the active node set and $TOTFEAS$ is the total number of feasible nodes generated. In many cases $TOTFEAS = 0$, indicating that the initial lower bound was equal to the critical path heuristic, and hence, no additional enumeration was necessary to prove that the critical path solution was optimal. These preliminary experiments confirm the independent conclusion of Adam, Chandy, and Dickson [1] that the critical path heuristic is near optimal and indicate that optimal solutions can be efficiently found in most cases by using the critical path solution to initialize a branch-and-bound search algorithm.

V. SUMMARY

This correspondence reviewed the known work on deterministic scheduling of multiprocessor systems and presented a preliminary evaluation of a simple critical path heuristic. While progress has been made on the problem, it is clear that much more remains to be learned even about this restricted model of a multiprocessor system considered here.

REFERENCES

[1] T. L. Adam, K. M. Chandy, and J. R. Dickson, "A comparison of list schedules for parallel processing systems," *Comm. Ass. Comput. Mach.*, vol. 17, no. 12, pp. 685–690, Dec. 1974.

[2] J. L. Baer, "A survey of some theoretical aspects of multiprocessing," *Comput. Surveys*, vol. 15, pp. 31–80, Mar. 1973.

[3] E. K. Bowdon, "Priority assignment in a network of computers," *IEEE Trans. Comput.*, vol. C-18, pp. 1021–1026, Nov. 1969.

[4] E. G. Coffman, Jr., and P. J. Denning, *Operating Systems Theory*. Englewood Cliffs, N. J.: Prentice-Hall, 1973.

[5] E. G. Coffman, Jr., and R. L. Graham, "Optimal scheduling for two processor systems," *Acta Informatica*, vol. 1, no. 3, pp. 200–213, 1972.

[6] P. J. Denning and G. S. Graham, "A note on subexpression ordering in the execution of arithmetic expressions," *Comm. Ass. Comput. Mach.*, vol. 16, no. 11, pp. 700–702, Nov. 1973.

[7] E. B. Fernandez and B. Bussell, "Bounds on the number of processors and time for multiprocessor optimal schedules," *IEEE Trans. Comput.*, vol. C-22, pp. 745–751, Aug. 1973.

[8] M. R. Garey and R. L. Graham, "Bounds on scheduling with limited resources," *ACM—Oper. Syst. Rev.*, vol. 7, no. 4, pp. 96–101, Oct. 1973.

[9] R. L. Graham, "Bounds for certain multiprocessing anomalies," *Bell Syst. Tech. J.*, vol. 45, no. 9, pp. 1563–1581, Nov. 1966.

[10] ——, "Bounds on multiprocessing timing anomalies," *SIAM J. Appl. Math*, vol. 17, no. 2, pp. 416–429, Mar. 1969.

[11] T. C. Hu, "Parallel sequencing and assembly line problems," *Oper. Res.*, vol. 9, no. 6, pp. 841–848, Nov. 1961.

[12] M. T. Kaufman, "An almost-optimal algorithm for the assembly line scheduling problem," *IEEE Trans. Comput.*, vol. C-23, pp. 1169–1174, Nov. 1974.

[13] W. H. Kohler and K. Steiglitz, "Characterization and theoretical comparison of branch-and-bound algorithms for permutation problems," *J. Comput. Mach.*, vol. 21, pp. 140–156, Jan. 1974.

[14] ——, "Exact, approximate, and guaranteed accuracy algorithms for the flow-shop problem $n/2/F/\bar{F}$," *J. Ass. Comput. Mach.*, vol. 22, pp. 106–114, Jan. 1975.

[15] ——, "Enumerative and iterative computational approaches," in *Comput. and Job-Shop Scheduling Theory*, E. G. Coffman, Jr., Ed. New York: Wiley, to be published.

[16] G. K. Manacher, "Production and stabilization of real-time task schedules," *J. Ass. Comput. Mach.*, vol. 14, pp. 439–465, July 1967.

[17] R. R. Muntz and E. G. Coffman, Jr., "Optimal preemptive scheduling on two-processor systems," *IEEE Trans. Comput.*, vol. C-18, pp. 1014–1020, Nov. 1969.

[18] ——, "Preemptive scheduling of real-time tasks on multiprocessor systems," *J. Ass. Comput. Mach.*, vol. 17, no. 2, pp. 324–338, Apr. 1970.

[19] C. V. Ramamoorthy, K. M. Chandy, and M. J. Gonzalez, Jr., "Optimal scheduling strategies in a multiprocessor system," *IEEE Trans. Comput.*, vol. C-21, pp. 137–146, Feb. 1972.

[20] L. Schrage, "Solving resource-constrained network problems by implicit enumeration-nonpreemptive case," *Oper. Res.*, vol. 18, pp. 263–278, Mar.–Apr. 1970.

[21] ——, "Solving resource-constrained network problems by implicit enumeration-preemptive case," *Oper. Res.*, vol. 20, no. 3, pp. 668–677, May–June 1972.

[22] J. D. Ullman, "Polynomial complete scheduling problems," *ACM—Oper. Syst. Rev.*, vol. 7, pp. 96–101, Oct. 1973.

The Sign/Logarithm Number System

EARL E. SWARTZLANDER, JR., AND ARISTIDES G. ALEXOPOULOS

***Abstract*—A signed logarithmic number system, which is capable of representing negative as well as positive numbers is described. A number is represented in the sign/logarithm number system by a sign bit and the logarithm of the absolute value of the number (scaled to avoid negative logarithms).**

Algorithms have been developed to perform all four of the basic arithmetic operations (i.e., addition, subtraction, multiplication, and division). It appears that such an arithmetic unit can be somewhat faster than a conventional arithmetic unit of comparable complexity. This system is intended for use in implementing special purpose computers, where constant relative accuracy is not objectionable.

***Index Terms*—Computer arithmetic units, logarithmic addition, logarithmic arithmetic, logarithmic subtraction, number systems.**

INTRODUCTION

Over the years, many number systems have been proposed and used to implement computer arithmetic units [1]. Most of the practical implementations have used variations of the familiar weighted binary number system (e.g., binary fractions, binary integers, or binary floating point numbers). The main problem with any of these systems is the relative slowness or high circuit complexity of multiplication and division. As a result the design of arithmetic units for weighted binary numbers involves a speed complexity compromise.

The residue number system has received much attention because carries do not propagate between the moduli thus resulting in a great speed increase for multiplication. Severe problems arise however, concerning division, overflow detection, and in magnitude comparison (e.g., to detect which of two numbers is larger). These problems have effectively prevented the widespread use of residue number systems in the realization of general purpose computers, although some special purpose processors have been built which take advantage of the relative ease and speed of addition, subtraction, and multiplication with a residue system [2].

This correspondence describes the sign/logarithm number system, a new approach which avoids the slowness of multiplication and division with conventional weighted numbers, without incurring the problems inherent in residue number systems. It must be emphasized that this system can not replace conventional arithmetic units in general purpose computers; rather it is intended to enhance the

Manuscript received March 4, 1974; revised June 3, 1975. This work is based on research performed on the Logarithmic Data Processor Development Program, Hughes Aircraft Company, with assistance from U. von der Embse, M. Noda, and R. Quinn.

E. E. Swartzlander, Jr. is with TRW Systems, Redondo Beach, Calif. 90278.

A. G. Alexopoulos is with the Hughes Aircraft Company, Los Angeles, Calif. 90009.

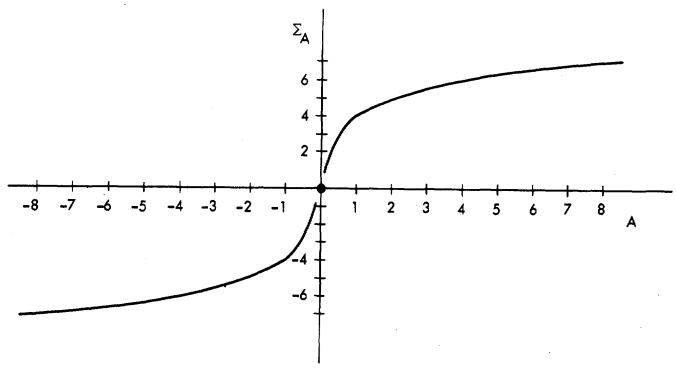


Fig. 1. Relationship between Σ_A and A.

implementation of special purpose processors for specialized applications (e.g., pattern recognition, digital image enhancement, radar processing, speech filtering, etc.).

The sign/logarithm representation of a number consists of the sign of the number appended to the logarithm of the absolute value of the number (scaled to avoid negative logarithms). This system thus avoids the classic problem with logarithmic number systems; the inability to represent negative numbers.

Algorithms for all four of the basic arithmetic operations, addition, subtraction, multiplication, and division, will be described in order to illustrate the speed and simplicity of implementation which this system affords to the designer of relatively low (finite) precision digital systems.

THE SIGN/LOGARITHM NUMBER SYSTEM

For the sign/logarithm number system any number, e.g., A, is represented by its sign S_A, and the binary logarithm of its magnitude L_A:

$$\begin{aligned} S_A &= 1 \quad \text{if} \quad A \leq 0 \\ S_A &= 0 \quad \text{if} \quad A \geq 0 \end{aligned} \tag{1}$$

Note: $S_A = 0$ or 1 if $A = 0$.

$$\begin{aligned} L_A &= \log (|\tau A|), \qquad \text{if } |A| > 1/\tau \\ L_A &= 0, \qquad\qquad\ \ \text{if } |A| \leq 1/\tau. \end{aligned} \tag{2}$$

Note that the logarithm of A is scaled by a constant factor, τ, to insure that $L_A \geq 0$. τ is determined by the designer of the arithmetic unit; it can be used to create an all fractional number system, an all integer system, or can be varied dynamically to effect block scaling [3]. All computations in the number system are performed on the "signed logarithm," $\sum_A$, which represents A:

$$\sum_A \equiv (1 - 2S_A)L_A. \tag{3}$$

The relationship between $\sum_A$ and A is sketched on Fig. 1 for $-8 \leq A \leq 8$, with $\tau = 16$. It is apparent from the graph that comparison of the magnitude of various numbers is implemented as easily with signed logarithms as it is in the original number system, since the logarithm is a monotonic function. It also may be seen that the scaling by τ and the use of the algebraic sign of the original number solves the classic inability to represent negative numbers in logarithmic number systems. There is, however, a deadband of width $2/\tau$ around zero.

The number A may be found from $\sum_A$ by separating it into S_A and L_A and applying the relationship:

$$A = (1 - 2S_A)(1/\tau)2^{L_A}. \tag{4}$$

Practical utilization of the sign/logarithm number system requires that the logarithm be represented as a finite precision number. Since L_A is nonnegative, a simple n bit weighted binary number will suffice for its representation. The finite precision form of L_A will be denoted K_A represented by n bits, $k_{nA}, k_{n-1A}, \cdots, k_{1A}$):

$$K_A = \sum_{i=1}^{n} k_{iA} 2^{i-\eta} \tag{5}$$

where the $\eta - 1$ least significant bits represent the fractional part of K_A. Since K_A is of finite precision, it is formed by rounding L_A from (2)

$$K_A = [\tfrac{1}{2} + L_A 2^{\eta-1}]2^{1-\eta} \tag{6}$$

where $[X]$ denotes the largest integer that is not larger than X (i.e., the ENTIER or FLOOR function). The additive constant of $\frac{1}{2}$ causes roundoff to occur in the formation of K_A instead of simple truncation, thus unbiasing the error and reducing error accumulation when numerous arithmetic operations are to be performed. In the following section algorithms for the four basic arithmetic operations will be presented. It is appropriate to summarize the procedure for converting a number, e.g., X, to the signed logarithm form.

If $|X| > 1/\tau$:

$$K_X = 2^{1-\eta}[\tfrac{1}{2} + 2^{\eta-1} \log_2 |\tau X|]. \tag{7}$$

If $|X| \leq 1/\tau$:

$$K_X = 0.$$

ARITHMETIC ALGORITHMS

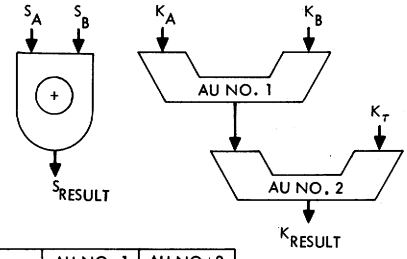
In this section, it will be assumed that it is desired to generate a signed logarithm result by performing each of the basic arithmetic operations on a pair of signed logarithm operands S_A, K_A and S_B, K_B. Since the algorithms for addition and subtraction are based on a multiplicative procedure, the multiplication and division algorithms will be described first. To illustrate these procedures examples are given in the Appendix.

Multiplication and Division

With conventional logarithmic number systems the logarithm of the product of two factors is given by the sum or their logarithms. Considering the system described by (2) with positive numbers A and B, simple summation of the logarithms of the factors yields erroneous results:

$$\begin{aligned} L_P &= L_A + L_B \\ &= \log_2 (\tau A) + \log_2 (\tau B) = \log_2 (\tau\tau AB) \end{aligned}$$

where τ is the scale factor. The result of this simple scheme is actually the product of the scale factor for the number system and the desired product. This problem can be alleviated by subtracting $\log_2 \tau$ from the sum of the two "factors." Converting to finite precision yields:



OPERATION	AU NO. 1	AU NO. 2
MULTIPLICATION	ADD	SUBTRACT
DIVISION	SUBTRACT	ADD

Fig. 2. Implementation of a multiplier/divider for the sign/logarithm number system.

$$K_P = K_A + K_B - K_\tau \tag{8}$$

where

$$K_\tau = 2^{1-\eta}[\tfrac{1}{2} + 2^{\eta-1}\log_2(\tau)]. \tag{9}$$

As with conventional sign/magnitude number systems, the sign of the product is determined by modulo 2 addition of the signs of the factors:

$$S_P = S_A \oplus S_B.$$

In a similar manner the quotient S_Q,K_Q of S_A,K_A divided by S_B,K_B is given by

$$S_Q = S_A \oplus S_B \tag{10a}$$

$$K_Q = K_A - K_B + K_\tau. \tag{10b}$$

Since both (8) and (10b) involve an addition and a subtraction it is quite possible for overflow or underflow to occur. In the case of overflow, the magnitude of the result exceeds the largest number which can be represented with an n bit ($\eta - 1$ of which are fractional) binary logarithm. Thus, if e.g., $|K_P| \geq 2^{n-\eta}$ overflow will result. Such overflow is directly analogous to overflow in a weighted binary number system, and corrective action can be initiated on its detection if desired. If block sealing is used all numbers can be shifted right and τ doubled. Underflow occurs when a negative number is formed in the computation of K_P or K_Q. For example, suppose that $K_B > K_A + K_\tau$ and division were attempted. Then by (10b) $K_Q < 0$, which is in direct violation of (2). This indicates that the logarithm of the magnitude of the quotient is so small that:

$$\log_2 Q_\tau \leq 2^{-\eta}. \tag{11}$$

In many cases it will be useful to detect underflow and assign a value of 0 to the result. In other cases it may be desirable to rescale the operands and repeat the calculation.

This illustrates one advantage of the sign/logarithm number system; unlike the residue system, overflow and underflow conditions are easily detected (and corrected if desired). It should also be noted that one adder and one subtractor are used for both multiplication and division; only the order of the two operations is interchanged. A direct implementation of a multiplier/divider is shown in Fig. 2. Although a fixed adder and subtractor may be used to effect this implementation, it may be advantageous to use two adder/subtractors which perform either function depending on their control signals.

Addition and Subtraction

The procedure for addition and subtraction with sign/logarithm numbers is based on an extension of multiplication. Consider the sum, S, of two numbers, A and B:

$$S = A + B$$

$$S = A(1 + B/A). \tag{12}$$

Note that the sum is now a product of the first number with a factor that is a function of the ratio of the two numbers. This can be restated in the form:

$$S = A\psi(B/A) \tag{13}$$

where $\psi(X) = 1 + X$. Three operations are required to compute the sum of A and B: first, the ratio B/A is computed, then the function ψ is evaluated, and finally the multiplication of A by the value of the function ψ is performed.

The procedure for adding (or subtracting) numbers of arbitrary sign is identical, except that different tables of the function ψ [of (13)] are used depending on whether the signs are alike or different.

If $K_A \geq K_B$:

$$S_S = S_D = S_A$$

$$K_S = K_A + \beta(K_B - K_A)$$

$$K_D = K_A + \gamma(K_B - K_A) \tag{14}$$

where $\beta(X) = \log_2(1 + 2^X)$ and $\gamma(X) = \log_2(1 - 2^X)$.

If $K_B > K_A$:

$$S_S = S_D = S_B$$

$$K_S = K_B + \beta(K_A - K_B)$$

$$K_D = K_B + \gamma(K_A - K_B). \tag{15}$$

For both (14) and (14) K_S denotes the logarithm of the sum, K_D the logarithm of the difference, etc. Note that (15) is identical to (14) except for interchanging the subscripts, A and B.

Realization of (14) and (15) with a comparator, an adder, a subtractor, a Read Only Memory, and a small amount of ancillary logic is shown in Fig. 3. The Read Only Memories for β and γ must each contain 2^n words of n bits each, for a total a total of $n2^{n+1}$ bits of storage. For $n = 8$ this is a total of 4096 bits which is easily attained with a single 512 × 8 Read Only Memory.

Complete Arithmetic Unit

Comparison of Figs. 2 and 3 reveals considerable commonality in the processing required to realize the four arithmetic operations with sign/logarithm numbers. Fig. 4 shows such an arithmetic unit. It is essentially similar to the sign/logarithm implementation of addition with some added logic to permit multiplication and division. This system will be used as a baseline for comparison of performance with conventional arithmetic units of comparable complexity.

Following a serial path through Fig. 4, it is evident that the delay in performing any of the four arithmetic operations, T_{OP}, is simply

$$T_{OP} = T_{COMP} + 2T_{ADD} + T_{ROM} \tag{16}$$

where T_{COMP} is the delay of the n bit comparator, T_{ADD} is the delay of the n bit adder/subtractor, and T_{ROM} is the delay of a $2^n \times n$ bit Read Only Memory. For convenience, the delay of the ancillary logic has been assumed to be negligible. As a matter of practicality, the

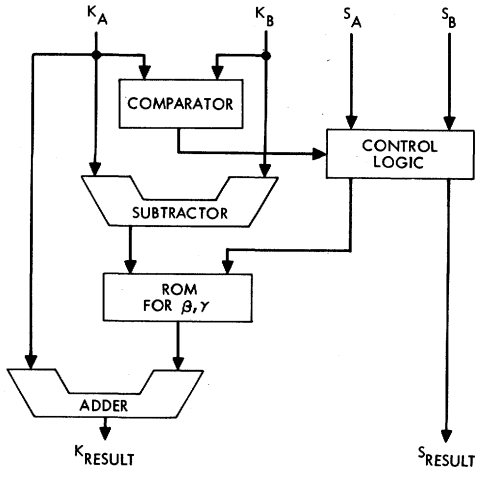


Fig. 3. Implementation of an adder/subtractor for the sign/logarithm number system.

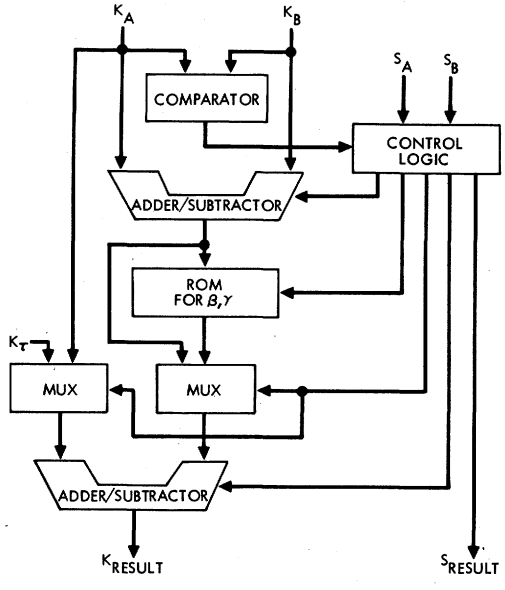


Fig. 4. Complete sign/logarithm arithmetic unit.

comparator may be implemented with a subtractor (a borrow output indicating that $K_A < K_B$) so that $T_{\text{COMP}} = T_{\text{ADD}}$ and

$$T_{\text{OP}} = 3T_{\text{ADD}} + T_{\text{ROM}}. \tag{17}$$

Note that multiplication and division can be performed in $T_{\text{OP}} = 2T_{\text{ADD}}$, at the expense of added circuit complexity since the comparison and function evaluation are not required. Thus, (17) represents a safe upper bound on the delay of the sign/logarithm arithmetic unit.

The hardware which is necessary to implement the unit of Fig. 4 is determined by inspection. Essentially three n bit adder/subtractors and two $n \times 2^n$ bit Read Only Memories are required.

COMPARISON WITH CONVENTIONAL ARITHMETIC UNITS

In order to make a realistic evaluation of the advantage in speed/simplicity achieved with the sign/logarithm arithmetic unit, it is necessary to determine the characteristics of a conventional arithmetic unit of equivalent complexity. It is assumed that two n bit adder/subtractors and two $n \times 2^n$ bit Read Only Memories are available. Addition or subtraction is trivial: either of the adder/subtractors is used with a delay of T_{ADD}. Multiplication is performed by dividing the bit matrix into quarters and using a $2^n \times n$ bit Read Only Memory to look up the corresponding partial product. Since the Read Only Memory permits the sum of each quadrant to be determined in a single step only four memory accession delays are required. A total of three additional adder delays (i.e., T_{ADD}) are needed to sum the partial products, yielding the complete product. Thus, the total delay is

$$T_{\text{MULT}} = 3T_{\text{ADD}} + 4T_{\text{ROM}}. \tag{18}$$

Division can be simply implemented with an iterative procedure using the remaining $n \times 2^n$ bit Read Only Memory to generate an n bit approximation to the quotient. In order to achieve $2n$ bit precision a delay of approximately $2T_{\text{MULT}}$ from (18) results [4]. The total delay, T_{DIV}, is thus

$$T_{\text{DIV}} = 6T_{\text{ADD}} + 8T_{\text{ROM}}. \tag{19}$$

TABLE I
DELAY COMPARISON

Arithmetic Unit Operation	Sign/Logarithm	Conventional
Add	$4\,T_{\text{ADD}}$	T_{ADD}
Subtract	$4\,T_{\text{ADD}}$	T_{ADD}
Multiply	$2\,T_{\text{ADD}}$	$7\,T_{\text{ADD}}$
Divide	$2\,T_{\text{ADD}}$	$14\,T_{\text{ADD}}$

Table I compares the speed of the sign/logarithm arithmetic unit with a conventional arithmetic unit. (Based on the assumption that $T_{\text{ADD}} \approx T_{\text{ROM}}$.) A significant feature of the sign/logarithm unit is that it takes within a factor of 2 of the same amount of time to perform any of the four basic operations. By comparison the conventional unit exhibits a 7–1 disparity between multiply and add and a 14–1 disparity between divide and add.

CONCLUSIONS

The sign/logarithm number system has been described and fast algorithms for addition, subtraction, multiplication, and division have been presented. Because of the current implementation of the addition and subtraction algorithms with Read Only Memories, the complexity of sign/logarithm arithmetic units increases rather drastically with increasing word size. It should be possible to reduce the complexity as more research is devoted to optimizing these algorithms. A specific area which appears to warrant more investigation is a more efficient method for computing $\beta(X)$ and $\gamma(X)$ for (14) and (15). The sparseness of these tables suggests that use of programmed logic arrays might result in a very efficient implementation. Note, for example, that although X takes on 64 values in Table II there are only 9 values of $\beta(X)$ and 17 values of $\gamma(X)$. No claim is made that this system is suitable for implementing general purpose computer arithmetic; indeed it should be noted that the constant relative precision of any logarithmic number system is inherently unsuitable for most integer operations, e.g., financial record keeping, computer address calculation, etc.

The current state of the art in Read Only Memories is such that a sign/logarithm arithmetic unit is feasible for words up to only 12 bits. Precision of this level should be quite adequate for many applications in the specialized areas of digital image enhancement [5], pattern recognition, and digital filtering [6]. For many such applications input transducers (i.e., photodetectors for image processing applications) with logarithmic response may be used to generate the logarithm directly; other applications will require the use of a logarithmic analog to digital converter [7] or a Read Only Memory to generate the logarithm. For more information on efficient methods for converting between linear and logarithmic number systems see [6] and [8]–[10].

TABLE II
$\beta(X)$ AND $\gamma(X)$ AS A FUNCTION OF X FOR $\tau = 16$ AND $\eta = 4$

X	$\beta(X)$	X	$\gamma(X)$
0 and −0.125	1.000	0	Difference = 0
−0.250 and −0.375	0.875	−0.125	−3.625
−0.500 and −0.625	0.750	−0.250	−2.625
−0.750 to −1.000	0.625	−0.375	−2.125
−1.125 to −1.375	0.500	−0.500	−1.750
−1.500 to −2.000	0.375	−0.625	−1.500
−2.125 to −2.750	0.250	−0.750	−1.250
−2.875 to −4.375	0.125	−0.875	−1.125
≤ −4.500	0	−1.000	−1.000
		−1.125	−0.875
		−1.250 and −1.375	−0.750
		−1.500 and −1.625	−0.625
		−1.750 and −1.875	−0.500
		−2.000 to −2.250	−0.375
		−2.375 to −3.000	−0.250
		−3.125 to −4.500	−0.125
		≤ −4.625	0

TABLE III
K_X AS A FUNCTION OF X FOR $\tau = 16$ AND $\eta = 4$

X	K_X	X	K_X
0	0	3.000 and 3.125	5.625
0.125	1.000	3.250 to 3.500	5.750
0.250	2.000	3.625 and 3.750	5.875
0.375	2.625	3.875 to 4.125	6.000
0.500	3.000	4.250 to 4.500	6.125
0.625	3.375	4.625 to 4.875	6.250
0.750	3.625	5.000 to 5.375	6.375
0.875	3.750	5.500 to 5.875	6.500
1.000	4.000	6.000 to 6.375	6.625
1.125	4.125	6.500 to 7.000	6.750
1.250	4.375	7.125 to 7.625	6.875
1.375	4.500	7.750 to 8.250	7.000
1.500	4.625	8.375 to 9.000	7.125
1.625 and 1.750	4.750	9.125 to 9.875	7.250
1.875	4.875	10.000 to 10.750	7.375
2.000	5.000	10.875 to 11.750	7.500
2.125 and 2.250	5.125	11.875 to 12.875	7.625
2.375	5.250	13.000 to 14.000	7.750
2.500 and 2.625	5.375	14.125 to 15.250	7.875
2.750 and 2.875	5.500	15.375 to 16.625	8.000

APPENDIX

In order to clarity the use of the sign/logarithm number system for arithmetic; examples of addition, subtraction, multiplication, and division are given below. For each algorithm, K_X is determined from Table III which was generated from (2) and (6) with $\tau = 16$ and $\eta = 4$. Table II, which is used for addition and subtraction, was generated from the relations:

$$\left.\begin{aligned}\beta(X) &= 2^{1-\eta}\left[\tfrac{1}{2} + 2^{\eta-1}\log_2(1 + 2^X)\right]\\ \gamma(X) &= 2^{1-\eta}\left[\tfrac{1}{2} + 2^{\eta-1}\log_2(1 - 2^X)\right]\end{aligned}\right\}, \quad \text{for } X \leq 0.$$

Again the quantization parameter of $\eta = 4$ was used.

Addition

Find:	$1 + 3$.
From Table III:	$K_A = 4.000$ and $K_B = 5.625$.
Equation (15):	$K_S = 5.625 + (4.00 - 5.625)$.
Table II:	$\beta(-1.625) = 0.375$.
Thus:	$K_S = 5.625 + 0.375 = 6.000$.

$1 + 3 = 4$.

Substraction

Find:	$2 - 5$.
From Table III:	$K_A = 5.000$ and $K_B = 6.375$.
Equation (14):	$K_D = 6.375 + (5.000 - 6.375)S_D = S_B = -$.
Table II:	$\gamma(-1.375) = -0.750$.
Thus:	$K_D = 6.375 - 0.750 = 5.625$.

$2 - 5 = -3$.

Multiplication

Find:	1.25×8.00.
from Table III:	$K_A = 4.375$ and $K_B = 7.000$.
Equation (9b):	$K_P = 4.375 = 7.000 - 4.000$.
Thus:	$K_P = 7.375$.

$1.250 \times 8.000 = 10.000$.

Division

Find:	$9 \div 2.125$.
from Table III:	$K_A = 7.125$ and $K_B = 5.125$.
Equation (10b):	$K_Q = 7.125 - 5.125 + 4.000$.
Thus	$K_Q = 6.000$.

$9 \div 2.125 = 4.000$.

ACKNOWLEDGMENT

We wish to thank C. G. Ugianskis and F. R. Schmid for their support.

REFERENCES

[1] H. L. Garner, "Number systems and arithmetic," in *Advances in Computers*, vol. 6, F. L. Alt and M. Rubinoff, Ed. New York: Academic, 1965.
[2] P. W. Cheney, "A digital correlator based on the residue number system," *IRE Trans. Electron. Comput.*, vol. EC-10, pp. 63–70, Mar. 1961.
[3] P. D. Welch, "A fixed-point fast Fourier transform error analysis," *IEEE Trans. Audio Electroacoust.*, vol. AU-17, pp. 151–157, June 1969.
[4] M. J. Flynn, "On division by functional iteration," *IEEE Trans. Comput.*, vol. C-19, pp. 702–706, Aug. 1970.
[5] T. S. Huang, W. F. Schreiber, and O. J. Tretiak, "Image processing," *Proc. IEEE*, vol. 59, pp. 1586–1609, Nov. 1971.
[6] E. L. Hall, D. D. Lynch, and S. J. Dwyer, III, "Generation of products and quotients using approximate binary logarithms for digital filtering applications," *IEEE Trans. Comput.*, vol. C-19, pp. 97–105, Feb. 1970.
[7] E. J. Duke, "*RC* logarithmic analog-to-digital (LAD) conversion," *IEEE Trans. Instrum. Meas.* (Corresp.), vol. IM-20, pp. 74–76, Feb. 1971.
[8] T. C. Chen, "Automatic computation of exponentials, logarithms, ratios, and square roots," *IBM J. Res. Develop.*, vol. 16, pp. 380–388, July 1972.
[9] D. Marino, "New algorithms for the approximate evaluation in hardware of binary logarithms and elementary functions," *IEEE Trans. Comput.* (Short Notes), vol. C-21, pp. 1416–1421, Dec. 1972.
[10] J. C. Majithia and D. Levan, "A note on base-2 logarithm computations," *Proc. IEEE* (Lett.), vol. 61, pp. 1519–1520, Oct. 1973.