REDUNDANT LOGARITHMIC NUMBER SYSTEMS

M.G. Arnold T.A. Bailey J.R. Cowles J.J. Cupal

University of Wyoming
Laramie, Wyoming

Abstract

A new number system is described which offers advantages over conventional floating point and sign/logarithm number systems. Redundant logarithmic arithmetic, like conventional logarithmic arithmetic, relies on table lookups to make the arithmetic unit simpler than an equivalent floating point unit. The cost of 32 bit subtraction in a redundant logarithmic number system is lower than previously published logarithmic subtraction methods. Another advantage of a redundant logarithmic number system is that a single arithmetic unit can use the same hardware to add, subtract, or multiply in similar times.

1 INTRODUCTION

Several computers and circuits have been built using logarithmic number systems, which offer fast, accurate, and inexpensive multiplication and division regardless of word size. Earlier studies have shown that interpolation makes it economical to implement logarithmic *addition* with precision and range equivalent to 32 bit IEEE 754 floating point.

This paper discusses a problem which has previously been unrecognized: fast and accurate logarithmic *subtraction* for wordsizes near 32 bits. In order to overcome the speed and accuracy problems of logarithmic subtraction, we propose a new number system, referred to as *redundant logarithmic*, which usually avoids the difficulties of logarithmic subtraction.

Outline

Section 2 provides a review of pertinent literature on number systems, and gives the conventional algorithms for logarithmic arithmetic. Section 3 introduces a new model for the acceptable error in logarithmic subtraction which shows that even under the most optimistic assumptions, there is a subtraction problem for wordsizes that approach 32 bits. Section 4 describes the new dual redundant logarithmic number system and its arithmetic algorithms, which solve this subtraction problem. Section 5 shows some simple examples that raise questions about the accuracy of redundant logarithmic multiplication. Section 6 presents conclusions.

2 REVIEW OF RELATED WORK

The term *redundant number system* indicates that if w is the number of bits in the redundant word, then 2^w is greater than the number of unique values that can be represented in the system. An example is the signed digit number system [3]. There are two non-redundant number systems for representing reals described in the literature that can be referred to as "logarithmic":

1. **Sign Logarithm Number System.** Although there are several minor variations [19,13,22,17,4, 25] on this system (such as FOCUS and CRD), most implementations use X (composed of X_S and X_L) to represent the real value x given by

$$x = \begin{cases} -b^{X_L} & \text{if } X_S = 1 \\ b^{X_L} & \text{if } X_S = 0 \\ 0 & \text{if } X_L = -\infty \end{cases} \quad (1)$$

   where $-\infty$ is a special value for representing zero (outside the fixed point range used to represent logarithms of non-zero numbers), $b \neq 1$ is positive, and X_S is the sign of the number being represented. If $|x| < 1$, X_L is negative, regardless of the sign of x.

   Use of the signed digit number system has been suggested for representing X_L [21]. The new number system described in Section 4 is redundant in a different way.

2. **Level Index Number System.** The level index (li) system [9] is a generalized logarithmic number system that allows gradual loss of precision as values increase in magnitude. Between 1 and e, the li representation is the same as the sign logarithm representation.

Sign Logarithm Arithmetic

The algorithms for multiplication and division of sign logarithm quantities are quite simple. For example, to multiply two numbers, x and y, represented with X and Y and obtain the result, r, represented by R:

$$R_L = X_L + Y_L, \quad R_S = X_S \oplus Y_S, \tag{2}$$

where $+$ indicates fixed point addition and $\oplus$ indicates exclusive OR.

The algorithms for logarithmic addition and subtraction are more involved. The first known description of these algorithms was given by Leonelli in 1803 [18]. To add x and y, represented with X and Y:

$$R_L = \begin{cases} Y_L & \text{if } X_L = -\infty \\ X_L + s_b(Y_L - X_L) & \text{if } X_S = Y_S \\ X_L + d_b(Y_L - X_L) & \text{if } X_S \neq Y_S \end{cases} \tag{3}$$

$$R_S = \begin{cases} X_S & \text{if } X_L > Y_L \\ Y_S & \text{otherwise} \end{cases}$$

$$s_b(z) = \log_b(1 + b^z) \tag{4}$$

$$d_b(z) = \log_b|1 - b^z|. \tag{5}$$

The function s_b is known as the addition logarithm or Gaussian logarithm (since Gauss published the first table of this function and helped popularize the manual use of s_b among European astronomers and surveyors). The function d_b is known as the subtraction logarithm.

Interpolation of s_b

Direct lookup of s_b is expensive for $w = 32$ since memories having 2^{32} words are not economical. Interpolation of s_b [7,1,24,2,12] can greatly reduce the size of tables required for 32 bit accuracy. Assuming the use of the non-redundant binary number system in (4), z can be broken into two parts, z_H and z_L:

$$\begin{gathered} z = z_H + z_L, \\ z_H = \mathrm{iz}_H \cdot \Delta z_H, \quad z_L = \mathrm{iz}_L \cdot \Delta z_L, \\ \Delta z_H = 2^{k-n}, \qquad \Delta z_L = 2^{k-n-j}, \end{gathered} \tag{6}$$

where k is the number of bits in the integer portion of z, iz_H is an n bit integer, and iz_L is an j bit integer. The total word size, including the sign bit of the real number being represented, is $w = 1+m+n+j$, where m is the number of additional bits on the left of the word including the sign bit of z_H. These m bits can be ignored during addition because of the essential zero method [11] and the fact that $s_b(z) = s_b(-z) + z$. One method of interpolation approximates

$$s_b(z) \approx s_b(z_H) + \frac{s_b(z_H + \Delta z_H) - s_b(z_H)}{\Delta z_H} \cdot z_L \tag{7}$$

in the range $z_H \leq z < z_H + \Delta z_H$. It has been shown [1] that the number of valid bits in the fractional part of (7) is $2 \cdot (n-k) + 5$ for $b = 2$. The number of bits in the fractional part is by definition $n+j-k$, and so $j \leq n$, assuming $k = 5$. Interpolation can approximate s_b with accuracy and range roughly equivalent to IEEE 32 bit floating point *addition* when $b = 2$, $w = 32$, $m = 3$, $n = 14$, $j = 14$, and $k = 5$. This requires a memory having 2^{14} words.

3 THE SUBTRACTION PROBLEM

Slow Subtraction Algorithms

This section presents new results that show that logarithmic *subtraction* is more difficult than logarithmic addition for a wordsize of approximately 32 bits. Logarithmic subtraction is harder because d_b cannot be directly interpolated for all z with as much accuracy as s_b can. There are algorithms that allow accurate computation of d_b using multiple s_b interpolation (using, for example $d_b(z) \approx -[s_b(z) + s_b(z \cdot 2^1) + s_b(z \cdot 2^2) + s_b(z \cdot 2^3) + \cdots]$) or binary search techniques (a modern implementation of the subtraction technique originally used by Leonelli and Gauss). However these algorithms either require a significant amount of iteration or a large number of parallel s_b hardware units. It can be shown that iterative algorithms have the effect of making 32 bit subtraction between 14 to 54 times slower than addition.

Required Accuracy

There is a fundamental limitation in the accuracy of subtraction, sometimes refered to as *catastrophic cancellation* [20], that is shared by both floating point and logarithm number systems. No one previously has used this limitation to analyze the required accuracy for d_b.

When two numbers are subtracted that are very close to the same magnitude, the quantization error in the original numbers leads to an inherent error of about the same magnitude in the result. On the

other hand, the value of the result may be several orders of magnitude smaller than the original numbers. This makes the relative error in the result much larger than the relative error in the original numbers. Any algorithm for calculating d_b should be accurate enough to avoid introducing errors which are significantly greater than this limitation. However, under the above assumptions, there is no need to provide d_b values which are much more accurate than this inherent error. The required accuracy for d_b is roughly equal to the inherent error in the result of the subtraction.

Given x and y having sign logarithm representations X and Y with $X_L > Y_L$ and $X_L \approx Y_L$, then the result $r = x - y$ with representation R has inherent error

$$\begin{aligned} \mathrm{err}(R_L) &= \frac{1}{\ln b} \cdot \frac{\mathrm{err}(r)}{r} \\ &= \frac{1}{\ln b} \cdot \frac{\frac{\mathrm{err}(r)}{r}}{\frac{\mathrm{err}(x)}{x}} \cdot \frac{\mathrm{err}(x)}{x} \\ &= \frac{1}{\ln b} \cdot \frac{\mathrm{err}(r)}{\mathrm{err}(x)} \cdot \frac{x}{r} \cdot \ln b \cdot \mathrm{err}(X_L) \end{aligned} \quad (8)$$

since the relative errors in x and r are proportional to the absolute errors in X_L and R_L. Using the maximum deviation of the distribution of represented values (x) about their actual values as a measure of the error,

$$\mathrm{err}(r) = 2 \cdot \mathrm{err}(x) \quad (9)$$

$$\mathrm{err}(R_L) = 2 \cdot \frac{x}{r} \cdot \mathrm{err}(X_L). \quad (10)$$

Assuming the representation X_L has an error which is uniformly distributed with a width equal to the quantisation,Δz_L, and again using the maximal deviation as a measure of the error,

$$\mathrm{err}(X_L) = \frac{1}{2} \cdot \Delta z_L. \quad (11)$$

The factor $\frac{x}{r}$ reduces to

$$\frac{x}{r} = \frac{x}{x-y} = \frac{b^{X_L}}{b^{X_L} - b^{Y_L}} = b^{-d_b(z)} = d_b'(-z) \quad (12)$$

where $d_b'(z)$ is the derivative of $d_b(z)$ with respect to z, and $z = Y_L - X_L$ is negative. For small z, we can use

$$\lim_{z \to 0} z \cdot d_b'(z) = \log_b(e) \quad (13)$$

to approximate $d_b'(-z)$, so

$$d_b'(-z) \approx \frac{\log_b(e)}{-z}. \quad (14)$$

Thus, the required accuracy for small z is estimated as

$$\mathrm{ReqAcc}(z, \Delta z_L) \approx \mathrm{err}(R_L) \approx \frac{\log_b(e)}{-z} \cdot \Delta z_L. \quad (15)$$

The following table compares the required accuracy with the error from interpolating $d_b(z)$ in the region near $z = -0.015$ using $b = 2$, $m = 3$, $k = 5$, and $n = j$ for various n.

n	z	Error	$\mathrm{ReqAcc}(z, \Delta z_L)$
14	-0.01463	$3.2 \cdot 10^{-3}$	$1.2 \cdot 10^{-5}$
15	-0.01513	$7.5 \cdot 10^{-4}$	$2.9 \cdot 10^{-6}$
16	-0.01538	$1.8 \cdot 10^{-4}$	$7.0 \cdot 10^{-7}$
17	-0.01550	$4.5 \cdot 10^{-5}$	$1.7 \cdot 10^{-7}$
18	-0.01556	$1.1 \cdot 10^{-5}$	$4.3 \cdot 10^{-8}$

In each instance the required accuracy is several orders of magnitude smaller than the error in the interpolated value of $d_b(z)$. As a consequence, errors are introduced during subtraction. Straightforward linear interpolation fails to produce the required accuracy. More complicated methods of interpolation could be considered (for example $n = 18, j = 10$ would produce the required accuracy near $z = -0.015$), however such methods are too expensive to be practical. (For $n = 18$, there is a factor of 16 increase in memory requirements and yet the subtraction error problem is still unsolved for $z > -0.015$.) There is no obvious way to keep the economy and simplicity of (7) for d_b interpolation and yet maintain the required accuracy. The next section discusses a new number system that eliminates the need to approximate d_b.

4 DUAL REDUNDANT LOGARITHM NUMBER SYSTEM

Definition

In contrast to the sign logarithm techniques described in the literature, the dual redundant logarithm number system (DRLNS) represents the real value x as two parts, $\hat{X}_P$ and $\hat{X}_N$. The wordsize is $w = 2(n + j + m)$. There is no sign bit for x in a DRLNS number, although both $\hat{X}_P$ and $\hat{X}_N$ are signed fixed point numbers. The value x represented by $\hat{X}$ is

$$x = b^{\hat{X}_P} - b^{\hat{X}_N}. \quad (16)$$

Since this is a redundant system, the representation $\hat{X}$ for a given x is not unique.

DRLNS Algorithms

The following briefly describes the fundamental algorithms for operating on DRLNS numbers.

Converting Sign Logarithm to DRLNS Form

DRLNS can be used in conjunction with the sign logarithm number system. In such an application, the hardware may need to convert from the sign logarithm number system to DRLNS. Since this implements a one to many mapping, there are several possible conversion methods.

Method 1. This method requires the DRLNS hardware to recognize the special representation of $-\infty$:

$$\hat{X}_P = \begin{cases} -\infty & \text{if } X_L = -\infty \\ X_L & \text{if } X_L \neq -\infty \text{ and } X_S = 0 \\ -\infty & \text{otherwise} \end{cases} \tag{17}$$

$$\hat{X}_N = \begin{cases} -\infty & \text{if } X_L = -\infty \\ -\infty & \text{if } X_L \neq -\infty \text{ and } X_S = 0 \\ X_L & \text{otherwise} \end{cases}$$

Method 2. Rather than require the DRLNS hardware to process the special representation of $-\infty$, use an additional bit in the representation of both $\hat{X}_P$ and $\hat{X}_N$ to insure that a value may always be chosen for every representable real number which is significantly smaller than the logarithm that represents the real number. In effect this value acts like $-\infty$.

$$\hat{X}_P = \begin{cases} -2^{m+k} & \text{if } X_L = -\infty \\ X_L & \text{if } X_L \neq -\infty \\ & \text{and } X_S = 0 \\ X_L - (n+j-k) & \text{otherwise} \end{cases} \tag{18}$$

$$\hat{X}_N = \begin{cases} -2^{m+k} & \text{if } X_L = -\infty \\ X_L - (n+j-k) & \\ & \text{if } X_L \neq -\infty \\ & \text{and } X_S = 0 \\ X_L & \text{otherwise} \end{cases}$$

Method 3. This method uses the fact that $x - 2x = -x$.

$$\hat{X}_P = \begin{cases} -\infty & \text{if } X_L = -\infty \\ X_L + \log_b(2) & \text{if } X_L \neq -\infty \\ & \text{and } X_S = 0 \\ X_L & \text{otherwise} \end{cases} \tag{19}$$

$$\hat{X}_N = \begin{cases} -\infty & \text{if } X_L = -\infty \\ X_L & \text{if } X_L \neq -\infty \\ & \text{and } X_S = 0 \\ X_L + \log_b(2) & \text{otherwise} \end{cases}$$

As will be shown, this approach is less desirable because under many circumstances it will introduce extra error in a computation.

Converting DRLNS to Sign Logarithm Form

Converting a redundant representation, $\hat{X}$, to non-redundant sign logarithm form, X, is expensive. It requires computing the subtraction logarithm, d_b, which costs from 14 to 54 times what computing s_b costs when using one of the techniques described in section 3.

$$X_S = \begin{cases} 0 & \text{if } \hat{X}_P \geq \hat{X}_N \\ 1 & \text{if } \hat{X}_P < \hat{X}_N \end{cases} \tag{20}$$

$$X_L = \begin{cases} \hat{X}_P + d_b(\hat{X}_N - \hat{X}_P) & \\ \qquad \text{if } \hat{X}_P \neq \hat{X}_N & \\ -\infty \quad \text{otherwise} & \end{cases}$$

Sign Detection

As illustrated in (20), the sign of a DRLNS number can be detected by comparing the fixed point $\hat{X}_P$ against $\hat{X}_N$. When $\hat{X}_P$ equals $\hat{X}_N$, $\hat{X}$ represents zero. When $\hat{X}_P$ is greater than $\hat{X}_N$, $\hat{X}$ represents a positive number. When $\hat{X}_P$ is less than $\hat{X}_N$, $\hat{X}$ represents a negative number.

Change of Sign

Interchanging $\hat{X}_P$ with $\hat{X}_N$ changes the sign of $\hat{X}$.

Addition and Subtraction

DRLNS allows both addition and subtraction without the need to compute the subtraction logarithm, d_b. Given the representations of x and y in DRLNS form, $\hat{X}$ and $\hat{Y}$, the following produces their sum, $\hat{R}$:

$$\hat{R}_P = \begin{cases} \hat{Y}_P & \text{if } \hat{X}_P = -\infty \\ \hat{X}_P + s_b(\hat{Y}_P - \hat{X}_P) & \text{otherwise} \end{cases} \tag{21}$$

$$\hat{R}_N = \begin{cases} \hat{Y}_N & \text{if } \hat{X}_N = -\infty \\ \hat{X}_N + s_b(\hat{Y}_N - \hat{X}_N) & \text{otherwise} \end{cases}$$

Unlike conventional sign logarithm arithmetic, there is no need to have two cases depending on the signs of x and y. Subtraction simply requires a change of sign of $\hat{Y}$ followed by the addition algorithm.

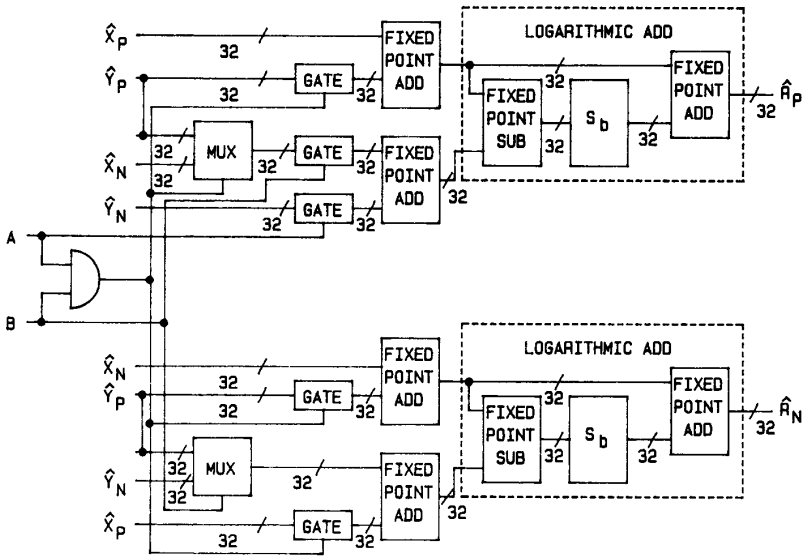

Figure 1: DRLNS Arithmetic Unit

Multiplication

Multiplication of DRLNS numbers is quite analogous to multiplication of complex numbers since

$$\begin{aligned} r &= x \cdot y \\ &= \left(b^{\hat{X}_P} - b^{\hat{X}_N}\right) \cdot \left(b^{\hat{Y}_P} - b^{\hat{Y}_N}\right) \\ &= \left(b^{\hat{X}_P+\hat{Y}_P} + b^{\hat{X}_N+\hat{Y}_N}\right) \\ &\quad - \left(b^{\hat{X}_P+\hat{Y}_N} + b^{\hat{X}_N+\hat{Y}_P}\right) \\ &\approx b^{\hat{R}_P} - b^{\hat{R}_N}. \end{aligned} \tag{22}$$

In DRLNS, -1 is treated the way one treats $\sqrt{-1}$ in the complex number system. Given two DRLNS numbers, $\hat{X}$ and $\hat{Y}$, the following produces their product, $\hat{R}$:

$$\hat{R}_P = \begin{cases} \hat{X}_N + \hat{Y}_N & \text{if } \hat{X}_P + \hat{Y}_P = -\infty \\ \hat{X}_P + \hat{Y}_P + & \\ s_b(\hat{X}_N + \hat{Y}_N - \hat{X}_P - \hat{Y}_P) & \\ & \text{otherwise} \end{cases}$$

$$\hat{R}_N = \begin{cases} \hat{X}_P + \hat{Y}_N & \text{if } \hat{X}_N + \hat{Y}_P = -\infty \\ \hat{X}_N + \hat{Y}_P + & \\ s_b(\hat{X}_P + \hat{Y}_N - \hat{X}_N - \hat{Y}_P) & \\ & \text{otherwise} \end{cases} \tag{23}$$

This computation requires only two additional fixed point adders beyond that required for addition and subtraction. Figure 1 shows how the same hardware that is used for multiplication can be used to implement addition and subtraction. The following shows what operation the control inputs, A and B, select:

A	B	Operation
0	1	Addition
1	0	Subtraction
1	1	Multiplication

When multiplication is selected, the AND gates force the adders to pass $\hat{X}_P$, $\hat{Y}_P$, $\hat{X}_N$, and $\hat{Y}_N$ respectively. The multiplexers choose $\hat{Y}_P$ for addition and multiplication, and $\hat{X}_N$ and $\hat{Y}_N$ for subtraction. Depending on the timing requirements in the implementation of Figure 1, it might be possible for the two s_b approximation units to share a common table and/or fixed point multiplier. Also, if there is space remaining, the same memory that holds the s_b table could store commonly used constants.

Mixed Multiplication

In some applications, it may be desirable to keep constants or frequently used data of unchanging sign in sign logarithm form. By doing this, the storage requirements are cut in half and the accuracy and cost of the computation may be improved. To make optimal use of such numbers, rather than converting them to DRLNS form, it is better to operate directly

on the number in sign logarithm form in a mixed computation. For example, the following multiplies a sign logarithm number X by a DRLNS number $\hat{Y}$:

$$\hat{R}_P = \begin{cases} -\infty & \text{if } X_L = -\infty \\ \hat{Y}_P + X_L & \text{if } X_L \neq -\infty \text{ and } X_S = 0 \\ \hat{Y}_N + X_L & \text{otherwise} \end{cases} \quad (24)$$

$$\hat{R}_N = \begin{cases} -\infty & \text{if } X_L = -\infty \\ \hat{Y}_N + X_L & \text{if } X_L \neq -\infty \text{ and } X_S = 0 \\ \hat{Y}_P + X_L & \text{otherwise} \end{cases}$$

Note that this can be accomplished with only two fixed point adders and two multiplexers that operate in parallel to each other. If DRLNS is used to implement the FFT, the butterfly operation can be rewritten to use (24) because the $e^{-2\cdot\pi\cdot\sqrt{-1}\cdot t}$ terms are constant, and can be stored in sign logarithm form.

Square Root, Reciprocal, and Division

In the sign logarithm number system, division, reciprocal, and square root operations are trivial. These operations are difficult to implement in DRLNS. It is possible to convert the DRLNS number to sign logarithm form, take the reciprocal or square root, and convert it back to DRLNS form. (In the case of division, the sign logarithm representation of the reciprocal could be multiplied directly using (24) to form the DRLNS quotient).

For algorithms that require large amounts of division, DRLNS does not offer the advantages that it does for algorithms involving only addition, subtraction and multiplication. If it is possible to factor division into an outer loop (eg. Gaussian elimination), DRLNS may be economical.

5 ERROR ANALYSIS

Several studies have shown that sign logarithm arithmetic is as or more accurate than floating point arithmetic [20,14,5,15,6,16,21]. If an algorithm is restricted to DRLNS addition, subtraction, and mixed multiplication these earlier studies will be applicable to DRLNS error analysis. For example, the error analysis of the FFT [23] is similar because (24) can implement the FFT. The accuracy of unrestricted DRLNS arithmetic requires careful study beyond the scope of this paper. However, some simple observations on DRLNS accuracy will illustrate why care must be taken during input conversion and DRLNS by DRLNS multiplication.

Significant Bits

Consider a positive DRLNS number, $\hat{X}$. (The roles of $\hat{X}_N$ and $\hat{X}_P$ are reversed for a negative value). The number of bits that are significant is

$$\begin{aligned} \text{sig}(\hat{X}) &\approx n + j - k + \log_2\left(\frac{b^{\hat{X}_P} - b^{\hat{X}_N}}{b^{\hat{X}_P}}\right) \quad (25) \\ &= n + j - k + \log_2(b) \cdot d_b(\hat{X}_N - \hat{X}_P). \end{aligned}$$

Therefore, the difference of $\hat{X}_P$ and $\hat{X}_N$ determines the accuracy of the answer. For $-1 < z < 0$, $d_b(z) \approx \lceil \log_b |z| - (\log_b \log_b e) \rceil$, and so

$$\begin{aligned} \text{sig}(\hat{X}) &\approx n + j - k - \lceil \log_2(b) \cdot (\log_b \log_b e) \rceil \\ &\quad + \lceil \log_2 |\hat{X}_N - \hat{X}_P| \rceil \quad (26) \end{aligned}$$

which can be computed easily by counting the number of leading ones in the binary representation of $\hat{X}_N - \hat{X}_P$. Let

$$a_x = b^{\hat{X}_P - \hat{X}_N}. \quad (27)$$

Since $\hat{X}$ represents a positive value, $a_x > 1$. The number of significant bits in $\hat{X}$ can be expressed in terms of a_x:

$$\text{sig}(\hat{X}) \approx n + j - k + \log_2(b) \cdot d_b(-\log_b(a_x)). \quad (28)$$

The following table shows this relationship for a few values of a_x when $b = 2$, $n = 14$, $j = 14$, and $k = 5$.

a_x	$\text{sig}(\hat{X})$	a_x	$\text{sig}(\hat{X})$
1.00001	6.39	2.50000	22.26
1.00010	9.71	3.00000	22.42
1.00100	13.03	4.00000	22.58
1.01000	16.34	5.00000	22.68
1.10000	19.54	6.00000	22.74
1.50000	21.42	7.00000	22.78
1.75000	21.78	8.00000	22.81
2.00000	22.00	9.00000	22.83

We use a_x as a convienent measure of how well conditioned the representation $\hat{X}$ is. Regardless of the choice of b, n, j, and k, when $a_x > 2$, $\hat{X}$ can be considered well conditioned because the difference between the theoretical maximum precision $(n + j - k)$ and $\text{sig}(\hat{X})$ is less than one. When $1 \leq a_x \leq 2$, $\hat{X}$ is ill conditioned since $\text{sig}(\hat{X})$ is considerably less than $n + j - k$.

Accuracy of DRLNS Multiplication

Although sign logarithm multiplication is exact, DRLNS multiplication can produce errors. Assume

that two positive numbers x (represented by $\hat{X}$) and y (represented by $\hat{Y}$) are to be multiplied. Analogously to (27) let

$$a_y = b^{\hat{Y}_P - \hat{Y}_N}. \tag{29}$$

The product of x and y is

$$\begin{aligned} p &= x \cdot y \\ &= (a_x b^{\hat{X}_N} - b^{\hat{X}_N})(a_y b^{\hat{Y}_N} - b^{\hat{Y}_N}) \\ &= (a_x a_y + 1) b^{\hat{X}_N} \cdot b^{\hat{Y}_N} - (a_x + a_y) b^{\hat{X}_N} \cdot b^{\hat{Y}_N}. \end{aligned} \tag{30}$$

From (30), the representation of p, $\hat{P}$, is given by

$$\begin{aligned} \hat{P}_P &= \log_b(a_x a_y + 1) + \hat{X}_N + \hat{Y}_N \\ \hat{P}_N &= \log_b(a_x + a_y) + \hat{X}_N + \hat{Y}_N \end{aligned} \tag{31}$$

Analogously to (27) and (29), define

$$a_p = b^{\hat{P}_P - \hat{P}_N} \tag{32}$$

which simplifies to

$$a_p = \frac{a_x a_y + 1}{a_x + a_y}. \tag{33}$$

Consider an iterated computation (of the form $p := x * y;\ x := p$) where y remains in the form in which it was initially converted. The following shows the value of a_p and $\text{sig}(\hat{P})$ during the first ten iterations:

iter.	a_p	$\text{sig}(\hat{P})$	a_p	$\text{sig}(\hat{P})$
1	$4.19430 \cdot 10^6$	23.00	1.25000	20.68
2	$2.79620 \cdot 10^6$	23.00	1.07692	19.19
3	$2.09715 \cdot 10^6$	23.00	1.02500	17.64
4	$1.67772 \cdot 10^6$	23.00	1.00826	16.07
5	$1.39810 \cdot 10^6$	23.00	1.00275	14.49
6	$1.19837 \cdot 10^6$	23.00	1.00091	12.90
7	$1.04857 \cdot 10^6$	23.00	1.00030	11.32
8	$9.32067 \cdot 10^5$	23.00	1.00010	9.74
9	$8.38860 \cdot 10^5$	23.00	1.00003	8.15
10	$7.62600 \cdot 10^5$	23.00	1.00001	6.57

The table on the left shows what happens with the second method of input conversion ($a_x = a_y \approx 2^{23}$). The table on the right shows what happens with the third method ($a_x = a_y = 2$). The third method loses more than one bit per iteration, but the second method maintains nearly constant relative precision.

Beyond avoiding the third method of input conversion, there is a possible solution to the accuracy problem: When the number of significant bits (as estimated by (26)) falls below an application specific criteria, conversion from redundant to sign logarithm form could be triggered. This might stabilize the accuracy, but would require additional time for the conversion.

The problem described here is related to the problems of other redundant (non-logarithmic) number systems, such as maximally redundant signed digit arithmetic [8]. To obtain a given precision in such a system requires an unbounded number of digits. Given a fixed number of bits, such redundant systems, including DRLNS, cannot guarantee a given precision without some (perhaps costly) stabilizing procedure (such as converting DRLNS to sign logarithm form).

6 CONCLUSIONS

A new logarithmic number system (named dual redundant logarithm number system or DRLNS) was described as a solution to the difficulties of logarithmic subtraction. DRLNS has the advantage of using the same hardware and cycle time for addition, subtraction and multiplication.

The disadvantages of DRLNS arithmetic are that division and square root are difficult. Like all redundant number systems, DRLNS requires extra bits. Unlike the sign logarithm number system, multiplication (other than DRLNS by sign logarithm) is not exact, and so the final precision may be reduced if care is not taken during the computations. Two techniques given here to improve accuracy are keeping frequently used data in sign logarithm form (converting to this form when needed) and proper input conversion.

DRLNS holds promise for algorithms that meet these critera:

1. The algorithm should have an inner loop that uses primarily addition, subtraction, and sign logarithm by DRLNS multiplication.

2. Division and square root should occur infrequently.

Many important algorithms, such as matrix multiply (by a constant matrix), Gaussian elimination, FIR and IIR filters, and the FFT satisfy these requirements.

The authors wish to thank A. Bechtolsheim, R. Jacquot, M. Magee, J. Richardson,R. Robertson, J. Rowland, P. Schlump, T. Stouraitis, D. Winkel, M. Winkel, and the referee who pointed out the importance of the relationship of (28) to well conditioned values.

References

[1] M.G. Arnold, "Extending the Precision of the Sign Logarithm Number System," M. S. Thesis, University of Wyoming, Laramie, 1982.

[2] M.G. Arnold, T.A. Bailey, and J.R. Cowles, "Improved Accuracy for Logarithmic Addition in DSP Applications," *Proc. IEEE Int. Conf. on Acoust., Speech, Signal Proc.*, p. 1714, 1988.

[3] A. Avizienis, "Signed-Digit Representations for Fast Parallel Arithmetic," *IRE Trans. Electr. Comput.*, vol. EC-10, p. 389, 1961.

[4] E.H. Bareiss and A.A. Grau, "Basics of the CRD Computer," Northwestern University ERDA Report COO-2280-25, August 1977.

[5] J.L. Barlow, "Probabalistic Error Analysis of Floating Point and CRD Arithmetic," PhD Dissertation, Northwestern University, Evanston, 1981.

[6] J.L. Barlow and E.H. Bareiss, "On Roundoff Distribuition in Floating Point and Logarithmic Arithmetic," *Computing*, vol. 34, p. 325, 1985.

[7] A. Bechtolsheim and T. Gross, "The Implementation of Addition in Logarithmic Arithmetic," Unpublished Manuscript, Computer Systems Laboratory, Stanford University, draft of March 1, 1980.

[8] T. Chen, "Maximal Redundancy Signed Digit Systems," *Proceedings of the 7th Symposium on Computer Arithmetic*, p. 296-300, 1985.

[9] C.W. Clenshaw and F.W.J. Olver, "Beyond Floating Point," *J. ACM*, vol. 31, p. 319, April, 1984.

[10] A.D. Edgar and S.C. Lee, "FOCUS Microcomputer Number System," *Commun. ACM*, vol. 22, p. 166, 1979.

[11] M.L. Frey and F.J. Taylor, "A Table Reduction Technique for Logarithmically Architected Digital Filters," *IEEE Trans. on Acoust., Speech, and Signal Proc.*, vol. ASSP-33, p. 718, 1985.

[12] H. Henkel, "Improved Accuracy for the Logarithmic Number System," *IEEE Trans. on Acoust., Speech, and Signal Proc.*, vol. ASSP-37, p. 301, 1989.

[13] N.G. Kingsbury and P.J.W. Rayner, "Digital Filtering Using Logarithmic Arithmetic," *Electron. Lett.*, vol. 7, p. 56, 1971.

[14] T. Kurokawa, J.A. Payne, and S.C. Lee, "Error Analysis of Recursive Digital Filters Implemented with Logarithmic Number Systems," *IEEE Trans. Acoust., Speech, Signal Proc.*, vol. ASSP-28, p. 706, 1980.

[15] R.O. LaMaire and J.H. Lang, "Performance of Digital Linear Regulators Which Use Logarithmic Arithmetic," *IEEE Trans. Automat. Contr.*, vol. AC-31, p. 394, 1986.

[16] J.H. Lang, C.A. Zukowski, R.O. LaMaire, and C.H. An, "Integrated Circuit Logarithmic Arithmetic Units," *IEEE Trans. Comput.*, vol. C-34, p. 475, 1985.

[17] S.C. Lee and A.D. Edgar, "The FOCUS Number System," *IEEE Trans. Comput.*, vol. C-26, p. 1167, 1977.

[18] Z. Leonelli, *Supplémente Logarithmique*, a reprint of Leonelli's 1803 manuscript with a biography by J. Hoüel, Gauthier-Villars, Paris, 1875.

[19] J.D. Marasa, "Accumulated Arithmetic Error in Floating-Point and Alternative Logarithmic Number Systems," M. S. Thesis, Sever Institute of Technology, Washington University, St. Louis, 1970.

[20] J.D. Marasa and D.W. Matula, "A Simulative Study of Correlated Error in Various Finite-Precision Arithmetics," *IEEE Trans. Comput.*, vol. C-22, p. 587, 1973.

[21] T. Stouraitis, "Logarithmic Number System Theory, Analysis, and Design," Ph. D. Dissertation, University of Florida, Gainesville, 1986.

[22] E.E. Swartzlander and A.G. Alexopoulos, "The Sign/Logarithm Number System," *IEEE Trans. Comput.*, vol. C-24, p. 1238, 1975.

[23] E.E. Swartzlander, D. Chandra, T. Nagle, and S.A. Starks, "Sign/Logarithm Arithmetic for FFT Implementation," *IEEE Trans. Comput.*, vol. C-32, p. 526, 1983.

[24] F.J. Taylor, "An Extended Precision Logarithmic Number System," *IEEE Trans. Acoust., Speech, Signal Proc.*, vol. ASSP-31, p. 231. 1983.

[25] F.J. Taylor, R. Gill, J. Joseph, and J. Radke, "A 20 Bit Logarithmic Number System Processor," *IEEE Trans. on Computers*, vol. C-37, p. 190, 1988.