Semi-Logarithmic Number Systems

Jean-Michel Muller, *Member*, *IEEE*, Alexandre Scherbyna, and Arnaud Tisserand

Abstract—We present a new class of number systems, called *Semi-Logarithmic Number Systems*, that constitute a family of various compromises between floating-point and logarithmic number systems. This allows trade between the speed of the arithmetic operations and the size of the required tables. We give arithmetic algorithms (addition/subtraction, multiplication, division) for the Semi-Logarithmic Number Systems, and we compare these number systems to the classical floating-point or logarithmic number systems.

Index Terms—Logarithmic number systems, floating-point arithmetic.

1 Introduction

The floating-point number system [6] is widely used for representing real numbers in computers, but many other number systems have been proposed. Among them, one can cite: the logarithmic and sign-logarithm number systems [9], [15], [14], [17], [8], [3], [10], the level-index number system [13], [20], [21], some rational number systems [11], and some modifications of the floating-point number system [22], [12]. Those systems have been designed to achieve various goals, e.g., to avoid overflows and underflows, to improve the accuracy, or to accelerate some computations. For instance, the sign-logarithm number system, introduced by Swartzlander and Alexpoulos [15], was designed in order to accelerate the multiplications. As pointed out by the authors, "*it cannot replace conventional arithmetic units in general purpose computers; rather it is intended to enhance the implementation of special-purpose processors for specialized applications*." That number system is interesting for problems where the required precision is relatively low, and where the ratio of multiplies (or divides, or square roots) to adds is relatively high. Roughly speaking, in such systems, the numbers are represented by their radix-2 logarithms written in fixed-point representation. The multiplications and divisions are performed by adding or subtracting the logarithms, and the additions and subtractions are performed using tables for the functions $\log_2(1 + 2^x)$ and $\log_2(1 - 2^x)$, since:

$$\begin{cases}\log_2(A+B) = \log_2(A) + \log_2\left(1 + 2^{\log_2(B)-\log_2(A)}\right)\\ \log_2(A-B) = \log_2(A) + \log_2\left(1 - 2^{\log_2(B)-\log_2(A)}\right)\end{cases}$$

The major drawback of the Logarithmic Number System arises when a high level of accuracy is required. If the computations are performed with n-bit numbers, then a straightforward implementation requires a table containing 2^n elements. Interpolation techniques allow the use of smaller tables (see [16], [2], [8]), so that 32-bit logarithmic number systems become feasible with current VLSI technologies. Our purpose in this paper is to present a new number system that allows the use of even smaller tables. That number system will be a sort of compromise between the logarithmic and the floating-point number systems. More exactly, we show a *family* of number systems, parameterized by a number k, and the systems obtained for the two extremal values of k are the floating-point and the logarithmic number systems. With some of these number systems, multiplication and division will be almost as easy to perform as in the logarithmic number system, whereas addition and subtraction will require smaller tables.

2 The Semi-Logarithmic Number Systems

Let k be an integer, let x be a real number different from 0, and define $e_{k,x}$ as the multiple of 2^{-k} satisfying

$$2^{e_{k,x}} \le |x| < 2^{e_{k,x}+2^{-k}}. \quad (1)$$

We immediately find

$$e_{k,x} = \frac{\lfloor 2^k \log_2|x| \rfloor}{2^k}. \quad (2)$$

Define $m_{k,x}$ as:

$$m_{k,x} = \frac{|x|}{2^{e_{k,x}}}.$$

If s_x is the sign of x, we obviously have

$$x = s_x \times m_{k,x} \times 2^{e_{k,x}},$$

where $e_{k,x}$ is a k-bit approximation of $\log_2|x|$ and $m_{k,x}$ is a multiplicative correction factor.

From (1) we deduce:

$$1 \le m_{k,x} = \frac{|x|}{2^{e_{k,x}}} < 2^{\frac{1}{2^k}}.$$

Now, let us bound the value $2^{\frac{1}{2^k}}$. This value is equal to $e^{\frac{\ln 2}{2^k}}$. One can easily show that, for $\alpha \in (0, 1)$,

$$1 + \alpha > 2^{\alpha}.$$

• J.-M. Muller and A. Tisserand are with CNRS, Laboratoire LIP, École Normale Supérieure de Lyon, 46 Allée d'Italie, 69364 Lyon Cedex 07, France. E-mail: jmmuller@lip.ens-lyon.fr.
• A. Scherbyna is with State Technical University, Cathedral SAPU, 31 Povitroflotski prospekt, Kiev, 252037, Ukraine.

Manuscript received 16 Mar. 1996; revised 7 July 1997.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number 106065.

Using this result with $\alpha = 1/2^k$, we get

$$2^{\frac{1}{2^k}} \leq 1 + \frac{1}{2^k} \tag{3}$$

for $k \geq 0$. As a consequence, $1 \leq m_{k,x} < 1 + \frac{1}{2^k}$. This leads to the following two definitions. The difference between the two is the "normalization" of $m_{k,x}$: The bound on $m_{k,x}$ required by the "general form" is easier to check.

DEFINITION 1 (Canonical Form). *Let k be a positive integer. Every nonzero real number x is represented in the* **Canonical form** *of the* Semi-Logarithmic Number System (SLNS for short) of Parameter *k by three values s_x, $m_{k,x}$, and $e_{k,x}$ satisfying:*

- $s_x = \pm 1$
- *$e_{k,x}$ is a multiple of* 2^{-k}
- $1 \leq m_{k,x} < 2^{\frac{1}{2^k}}$
- $x = s_x \times m_{k,x} \times 2^{e_{k,x}}$

DEFINITION 2 (General Form). *Let k be a positive integer. Every nonzero real number x is represented in the* **General form** *of the SLNS of Parameter k by three values s_x, $m_{k,x}$, and $e_{k,x}$ satisfying:*

- $s_x = \pm 1$
- *$e_{k,x}$ is a multiple of* 2^{-k}
- $1 \leq m_{k,x} < 1 + 2^{-k}$
- $x = s_x \times m_{k,x} \times 2^{e_{k,x}}$

The canonical form is a kind of floating-point representation, with exponents that are multiples of 2^{-k}, and a corresponding "normalized" mantissa.

The representation of x with n mantissa bits in the *semi-logarithmic number system of parameter k* will be constituted by s_x, $e_{k,x}$, and an n-fractional bit rounding of $m_{k,x}$. In practice, since $1 \leq m_{k,x} < 1 + 2^{-k}$, $m_{k,x}$ has a binary representation of the form:

$$1.\overbrace{\underbrace{0000\ldots000}_{k\text{ zeros}}\text{xxxx}\ldots\text{xx}}^{n\text{ bits}}$$

Since the first $k + 1$ bits of $m_{k,x}$ are known in advance, there is no need to store them (this is similar to the *hidden bit* convention of some radix-2 floating point systems [6]). Exactly as for normalized floating point representations, a special representation must be chosen for zero. In the following, k is considered implicit, and we write "m_x" and "e_x" instead of "$m_{k,x}$" and "$e_{k,x}$." Some points need to be emphasized:

- If $k = 0$, then the semi-logarithmic system of order k is reduced to a n-bit mantissa floating-point system.
- If $k \geq n$, then the semi-logarithmic system of order k is reduced to a logarithmic number system.
- The canonical form is a *nonredundant* representation. In that form, comparisons are easily performed: If the format of the representation is, from left to right, constituted by the sign, the exponent—which is a multiple of 2^{-k}—and then the mantissa, then comparisons are performed exactly as if we were comparing integers.
- The general form is a *redundant* representation. For instance, if $k = 1$, then $\sqrt{2}$ has two possible representations, namely $1.0000000\ldots \times 2^{0.1}$—the exponent and mantissa are written in radix-2—and $1.011010100000100111\ldots \times 2^{0.0}$. Although the comparisons are slightly more difficult to perform with the general form—this is due to the redundancy—we will prefer that form, because the condition "$1 \leq m_{k,x} < 1 + 2^{-k}$" is easier to check than the condition "$1 \leq m_{k,x} < 2^{\frac{1}{2^k}}$," and because the general form leads to simpler arithmetic algorithms. Anyway, the conversion from the general form to the canonical form is easily performed: Assume $s_x \times m_x \times 2^{e_x}$ is in general form. Compare m_x with $\rho_k = 2^{2^{-k}}$. If $m_x < \rho_k$, then the number is already represented in canonical form. If $m_x \geq \rho_k$, then add 2^{-k} to e_x and divide m_x by ρ_k. The obtained result will be the representation of x in canonical form.

So, the parameter k makes it possible to choose various compromises between the floating-point number system and the logarithmic number system.

Exactly as in floating-point arithmetic, there are various possible rounding modes. For instance, if we define $Z(x)$ as the number obtained by rounding m_x (in canonical form) to zero, then we get:

$$Z(x) = s_x \times \frac{\left\lfloor 2^n \times \frac{|x|}{2^{\left\lfloor 2^k \log_2 |x| \right\rfloor / 2^k}} \right\rfloor}{2^n} \times 2^{\left\lfloor 2^k \log_2 |x| \right\rfloor / 2^k}.$$

Similarly, we can define rounding towards $\pm\infty$ and rounding to the nearest.

3 The SLNS Viewed as a "Mixed-Base Logarithmic System"

When implementing a logarithmic number system, one has to choose the *base* (or *radix*) of the system. In the introduction, we assumed base two (i.e., a number is represented by its base-2 logarithm). Another "natural" choice is base e. Both systems have pros and cons. The main advantage of base two is that multiplying a fixed-point number by an *integer* power of two reduces to a shift (assuming that this number is represented in radix-2). This may save memory and time when performing additions or conversions. The main advantage of base e lies in the fact that, if ϵ is small,

$$\exp(\epsilon) \approx 1 + \epsilon.$$

To sum up, if x is an integer 2^x is easily computed, and if x is very small, e^x is easily computed.

As pointed out by one of the referees, the semi-logarithmic number systems can be viewed as a "mixed-base" logarithmic number system that uses both bases: 2 and e. Let

```
$s \leftarrow s_x \times s_y$                          The sign of the final result
$e \leftarrow e_x + e_y$                               1st approximation of the exponent
$m \leftarrow m_x \times m_y$                          If $k > n/2$ then $m_x \times m_y$ can be reduced to
                                                       an addition ($m_x = 1+\epsilon_1, m_y = 1+\epsilon_2$, with
                                                       $\epsilon_1, \epsilon_2 < 2^{-n/2}, m_x \times m_y = 1+\epsilon_1+\epsilon_2+\epsilon_1\epsilon_2$,
                                                       and the product $\epsilon_1\epsilon_2$ can be ignored, since it
                                                       is less than $2^{-n}$)
$m = 1.000\ldots 0m_{k-1}m_km_{k+1}\ldots m_n$          $m$ is between 1 and $1+2^{-k+1}+2^{-2k}$, there-
                                                       fore the bits of weight $2^{-k+1}$ and $2^{-k}$ of $m$
                                                       ($m_{k-1}$ and $m_k$) may be different from zero
$(\alpha, 2^\alpha) \leftarrow \mathsf{Table}_\times(m_{k-1}, m_k, m_{k+1})$   Look up $\alpha$ and $2^\alpha$ in a small (8-entry) ta-
                                                       ble (with $m_{k-1}$, $m_k$ and $m_{k+1}$ as address
                                                       bits) with $\alpha = \frac{\lfloor -\log_2(m^*)\times 2^k \rceil}{2^k}$, where $\lfloor u \rceil$
                                                       is the integer which is closest to $u$ and $m^* =$
                                                       $1.000\ldots 0m_{k-1}m_km_{k+1}$
$\hat{m} \leftarrow m \times 2^\alpha$                  Can be reduced to an addition if $k > n/2+2$
$\hat{e} \leftarrow e - \alpha$
If $\hat{m} \geq 1$ Then
    Return $(s, \hat{m}, \hat{e})$
Else
    Return $(s, \hat{m} \times 2^{2^{-k}}, \hat{e} - 2^{-k})$   If $k > n/2+2$ the product $\hat{m} \times 2^{2^{-k}}$ can be
                                                       reduced to an addition
```

Algorithm 4.1. SLNS multiplication.

$$x = 1.\overbrace{\underbrace{0000\ldots000}_{k \text{ zeros}}\text{xxxx}\ldots\text{xx}}^{n \text{ bits}} \times 2^{e_x} = \left(1+\epsilon_x\right) \times 2^{e_x},$$

where ϵ_x is very small (less than 2^{-k}) and e_x is a multiple of 2^{-k}. We have

$$x = \left(1+\epsilon_x\right) \times 2^{e_x} \approx e^{\epsilon_x} 2^{e_x} = 2^{e_x+\epsilon_x/\ln(2)} = 2^{e_x+\epsilon'_x},$$

where $\epsilon'_x = \epsilon_x/\ln(2)$. Therefore, the SLNS can be viewed as a mix-up of two logarithmic number systems, a base-2 system for e_x and a base-e system for ϵ_x. We will see later that this makes it possible to take benefit from the presented above advantages of both bases.

4 Basic Arithmetic Algorithms

Now, let us present basic algorithms for multiplication, division, addition, subtraction, and comparison. We must notice that, as soon as k is larger than $\frac{n}{2}+2$, these algorithms—and, especially, the multiplication and division algorithms—become very simple.

4.1 Multiplication

Assume we want to multiply $s_x \times m_x \times 2^{e_x}$ by $s_y \times m_y \times 2^{e_y}$, where these values are represented in the Semi-Logarithmic Number System of parameter k (general form). Algorithm 4.1 describes the multiplication method.

Proof of the Algorithm.

From

$$\alpha = \frac{\lfloor -\log_2(m^*) \times 2^k \rceil}{2^k},$$

we easily deduce

$$-\log_2 m^* - 2^{-k-1} \leq \alpha \leq \log_2 m^* + 2^{-k-1},$$

therefore,

```
$s \leftarrow s_x \times s_y$                          The sign of the final result
$e \leftarrow e_x - e_y$                               1st approximation of the exponent
$m \leftarrow \frac{m_x}{m_y}$                          If $k > n/2$ then $m_x/m_y$ can be reduced to a
                                                       subtraction ($m_x = 1+\epsilon_1$, $m_y = 1+\epsilon_2$,
                                                       with $\epsilon_1, \epsilon_2 < 2^{-n/2}$, therefore $m_x/m_y =$
                                                       $1+\epsilon_1-\epsilon_2-\epsilon_1\epsilon_2+\epsilon_2^2+\ldots$, and all the
                                                       terms but $1+\epsilon_1-\epsilon_2$ can be neglected). Even
                                                       if $k \leq n/2$, the fact that $m_y$ is very close to
                                                       1 can be used to accelerate the division pro-
                                                       cess using an iterative division method such
                                                       as Goldschmidt's algorithm
$m = \begin{cases} 1.000\ldots 00m_{k+1}m_{k+2}\ldots \\ \text{or} \\ 0.111\ldots 1m_{k+1}m_{k+2}\ldots \end{cases}$
                                                       From $1 \leq m_x < 1+2^{-k}$ and $1 \leq m_y <$
                                                       $1+2^{-k}$, we deduce $\frac{1}{1+2^{-k}} < m = \frac{m_x}{m_y} < 1+$
                                                       $2^{-k}$, which implies $1-2^{-k} < m < 1+2^{-k}$
If $m_k = 0$ Then
    Return $(s, m, e)$                                  The final result
Else
    $(\alpha, 2^\alpha) \leftarrow \mathsf{Table}_\div(m_{k+1})$   Look up $\alpha$ and $2^\alpha$ in a small (2-entry) ta-
                                                       ble with $\alpha = \frac{\lfloor -\log_2(m^*)\times 2^k \rceil}{2^k}$, where $\lfloor u \rceil$ is
                                                       the integer which is closest to $u$ and $m^* =$
                                                       $0.111\ldots 1m_{k+1}$
    $\hat{m} \leftarrow m \times 2^\alpha$              Can be reduced to an addition if $k > n/2+2$
    $\hat{e} \leftarrow e - \alpha$
    If $\hat{m} \geq 1$ Then
        Return $(s, \hat{m}, \hat{e})$
    Else
        Return $(s, \hat{m} \times 2^{2^{-k}}, \hat{e} - 2^{-k})$   If $k > n/2+2$ the product $\hat{m} \times 2^{2^{-k}}$ can be
                                                       reduced to an addition
```

Algorithm 4.2. SLNS division.

$$\frac{m}{m^*} \times 2^{-2^{-k-1}} \leq m \times 2^\alpha \leq \frac{m}{m^*} \times 2^{+2^{-k-1}}$$

$$\frac{m}{m^*} = 1 + \frac{m-m^*}{m^*} < 1 + \frac{2^{-k-1}}{m^*} < 1 + \frac{2^{-k-1}}{1+2^{-k}}.$$

This gives

$$\begin{aligned} 2^{2^{-k-1}} \frac{m}{m^*} &\leq \left(1+2^{-k-1}\right)\left(1+\frac{2^{-k-1}}{1+2^{-k}}\right) \\ &= \frac{1}{1+2^{-k}}\left(1+2\times 2^{-k}+2^{-2k-1}+2^{-2k-2}\right) \\ &< \frac{1}{1+2^{-k}}\left(1+2\times 2^{-k}+2^{-2k}\right) \\ &= \frac{\left(1+2^{-k}\right)^2}{1+2^{-k}} = 1+2^{-k}. \end{aligned}$$

If $m \times 2^\alpha < 1$, then (since $m/m^* \geq 1$):

$$2^{-2^{-k-1}} \leq m \times 2^\alpha < 1,$$

therefore,

$$1 < m \times 2^\alpha \times 2^{2^{-k}} \leq 1 + 2^{-k}.$$

4.2 Division

Assume we want to divide $s_x \times m_x \times 2^{e_x}$ by $s_y \times m_y \times 2^{e_y}$, where these values are represented in the Semi-Logarithmic Number System of parameter k (general form). This can be done as described in Algorithm 4.2. The proof of this algorithm is very similar to the proof of the multiplication algorithm.

```
If $e_x < e_y$ Then
    Exchange$(x, y)$
$u \leftarrow \lfloor e_x - e_y \rceil, v \leftarrow e_x - e_y - u$        $u$ is an integer, $v$ satisfies $0 \le |v| \le 1/2$
$m_y \leftarrow$ Rshift$(m_y, u)$                                      Perform a $u$-bit right shift of $m_y$
$\beta \leftarrow$ Table$_{\pm,\beta}(v_1, v_2, \ldots, v_{k-1})$      Look up $\beta = 2^{-v}$ in a table with $(k-1)$ address bits
$m_y^* \leftarrow m_y \times \beta$
$p \leftarrow s_x \times m_x \pm s_y \times m_y^*$                     Addition $\Rightarrow +$, subtraction $\Rightarrow -$
$s \leftarrow \mathrm{sign}(p)$                                        The sign of the final result
$p \leftarrow |p|$
If $p \ge 2$ Then
    $m \leftarrow$ Rshift$(p, 1)$                                      $m$ is the 1-bit right shift of $p$
    $e \leftarrow e_x + 1$
Else
    If $p = 0$ Then
        Return $(s_0, m_0, e_0)$                                       $(s_0, m_0, e_0)$ is the special code chosen for the number zero
    $j \leftarrow \#_0(p)$                                             $\#_0(p)$ gives the number of zeros between the bit of weight $2^0$ and the most significant "1" in $p$
    $m \leftarrow$ Lshift$(p, j)$                                      $m$ is the $j$-bit left shift of $p$
    $e \leftarrow e_x - j$
$(\alpha, 2^\alpha) \leftarrow$ Table$_{\pm,\alpha}(m_1, \ldots, m_{k+1})$   Look up $\alpha$ and $2^\alpha$ in a table with $(k+1)$ address bits (with $m_1, m_2, \ldots, m_{k+1}$ as address bits) with $\alpha = \frac{\lfloor -\log_2(m^*) \times 2^k \rceil}{2^k}$
$\hat{m} \leftarrow m \times 2^\alpha$
$\hat{e} \leftarrow e - \alpha$
If $\hat{m} \ge 1$ Then
    Return $(s, \hat{m}, \hat{e})$                                     The final result
Else
    Return $(s, \hat{m} \times 2^{2^{-k}}, \hat{e} - 2^{-k})$          If $k > n/2 + 2$ the product $\hat{m} \times 2^{2^{-k}}$ can be reduced to an addition
```

Algorithm 4.3. SLNS addition/subtraction.

4.3 Addition and Subtraction

Assume we want to compute $(s_x \times m_x \times 2^{e_x}) \pm (s_y \times m_y \times 2^{e_y})$, where these values are represented in the Semi-Logarithmic Number System of parameter k (general form). Exactly as in floating-point arithmetic, the basic method consists of "aligning" the mantissas (i.e., rewriting both numbers with the same exponent), adding the aligned mantissas and renormalizing the result. Algorithm 4.3 describes the addition/subtraction method.

Provided that $k > n/2 + 2$, the only "large multiplication" that appears in the arithmetic algorithms is the calculation of $\hat{m} = m \times 2^\alpha$ of the addition/subtraction algorithm (this is a multiplication of two n-bit integers). It is possible to avoid this $n \times n$ multiplication by slightly modifying the algorithm: If, instead of only returning α and 2^α, the table used also returns $2^{-\alpha}$, then one can compute $\hat{m}$ as $(m - 2^{-\alpha}) \times 2^\alpha + 1$. It is easy to show that $m - 2^{-\alpha} < 2^{-(k-1)}$, therefore, the multiplication $(m - 2^{-\alpha}) \times 2^\alpha$ is the multiplication of an $n - k + 1$-bit number by an n-bit number. If $k > n/2$, this leads to a significant reduction in the size of the required multiplier and the time of computation. Moreover, this method does not increase the required amount of memory: We only need $n - k + 1$ bits of $2^{-\alpha}$ (since its $k - 1$ most significant bits are zeroed when they are added to m), and we only need, at most, $n - k + 1$ bits of 2^α, since the influence of its less significant bits is negligible.

The addition/subtraction algorithm is the only algorithm that requires the use of a large table (containing 2^{k+1} values). This should be compared to the 2^n values that are required when implementing a Logarithmic Number System without interpolations. Of course, the use of interpolation techniques can result in substantially smaller tables, but the tables required by the SLNS can be interpolated as well. If a table with 2^{k+1} elements cannot be implemented, one can use two tables with $2^{\frac{k+1}{2}+1}$ elements, and decompose the computation of $\hat{m}$ in two steps:

- Define $j = \frac{k+1}{2}$. In the first step, look up, in a table with $(j + 1)$ address bits (with $m_1, m_2, \ldots, m_{j+1}$ as address bits), the values α_1 and 2^{α_1} satisfying:

$$\alpha_1 = \frac{\left\lceil -\log_2\left(1.m_1 m_2 \ldots m_{j+1}\right) \times 2^k \right\rceil}{2^k}$$

and compute $m^{(1)} = m \times 2^{\alpha_1}$. One can show that $m^{(1)}$ is between 1 and $1 + 2^{-j+1}$.

- Look up, in a table with $(j + 1)$ address bits (with $m_j^{(1)}, m_{j+1}^{(1)}, \ldots, m_{k+1}$ as address bits), the values α_2 and 2^{α_2} satisfying:

$$\alpha_2 = \frac{\left\lfloor -\log_2\left(1.000\ldots0 m_j^{(1)} m_{j+1}^{(1)} m_{k+1}^{(1)}\right) \times 2^k \right\rfloor}{2^k}$$

and compute $\hat{m} = M^{(1)} \times 2^{\alpha_2}$ and $\hat{e} = e - \alpha_1 - \alpha_2$. If $\hat{m} \ge 1$, then $\hat{m}$ is the mantissa of the result, while $\hat{e}$ is its exponent. If $\hat{m} < 1$, then multiply $\hat{m}$ by $2^{2^{-k}}$ and subtract 2^{-k} from the new computed value $\hat{e}$: This gives the mantissa and the exponent of the result.

If tables of size $2^{\frac{k+1}{2}+1}$ are still too large, then both previous steps can be decomposed again.

If we view the Semi-Logarithmic Number System as a mixed-base logarithmic number system (see Section 3), that is, if we write:

$$x = s_x \times 2^{e_x} \times e^{\epsilon_x}$$
$$y = s_y \times 2^{e_y} \times e^{\epsilon_y},$$

with $\epsilon_x, \epsilon_y < 2^{-k}$, then the algorithm uses (we assume $e_x \ge e_y$):

$$x + y = 2^{e_x} e^{\epsilon_x}\left(1 + e^{\epsilon_y - \epsilon_x} 2^{e_y - e_x}\right)$$
$$\approx 2^{e_x} e^{\epsilon_x}\left(1 + \left(1 + \epsilon_y - \epsilon_x\right) \times 2^{e_y - e_x}\right)$$
$$= 2^{e_x} e^{\epsilon_x}\left(1 + \left(1 + \epsilon_y - \epsilon_x\right) \times 2^{\text{integer part}\left(e_y - e_x\right)} \times 2^{\text{fractional part}\left(e_y - e_x\right)}\right).$$

In this last approximation, the fact that we use two different bases is essential: Using radix-2 allows us to reduce the multiplication by $2^{\text{integer part}(e_y - e_x)}$ to a mere shift. Only the multiplication by $2^{\text{fractional part}(e_y - e_x)}$ requires a table. Using radix e allows us to approximate $e^{\epsilon_y - \epsilon_x}$ by $1 + \epsilon_y - \epsilon_x$.

4.4 Comparisons

Assume we want to compare $x = s_x \times m_x \times 2^{e_x}$ and $y = s_y \times m_y \times 2^{e_y}$, where these values are represented in the Semi-Logarithmic Number System of parameter k (general form). We assume that x and y are positive (if their signs are different, then the comparison is straightforward, and if both numbers are negative, the required modification of the algorithm is obvious). We also assume that $e_x \geq e_y$ (if this is not true, exchange x and y). The comparison can be done as follows:

- If $e_x - e_y > 2^{-k}$, then $x > y$.
- If $e_x = e_y$, then $x \geq y$ if and only if $m_x \geq m_y$.
- If $e_x - e_y = 2^{-k}$, then multiply m_y by the precomputed value 2^{-2^k}—if $k > n/2$, then this multiplication can be reduced to an addition—this gives a value m_y^*. Then, $x \geq y$ if and only if $m_x \geq m_y^*$.

4.5 Conversions

4.5.1 From the Floating-Point Representation to the SLNS Representation

Let x be a positive number (dealing with the signs is obvious), represented in binary floating-point as $M_x \times 2^{E_x}$, that is,

- $x = M_x \times 2^{E_x}$,
- $1 \leq M_x < 2$,
- E_x is an integer.

We want to convert x to the SLNS system of parameter k (general form), i.e., to find m_x and e_x satisfying:

- $x = m_x \times 2^{e_x}$,
- $1 \leq m_x < 1 + 2^{-k}$,
- e_x has k fractional bits.

Assume that the binary representation of M_x is $1.M_1 M_2 \ldots M_n$, and define

$$M_x^* = 1.M_1 M_2 \ldots M_{k+1}.$$

The conversion is similar to the last two steps of the addition algorithm:

1) Look up the values α and 2^α defined below (in the table with $(k + 1)$ address bits already required by the addition algorithm, with $M_1, M_2, \ldots, M_{k+1}$ as address bits):

$$\alpha = \frac{\left\lfloor -\log_2\left(M_x^*\right) \times 2^k \right\rfloor}{2^k}.$$

2) Compute $\hat{m} = M_x \times 2^\alpha$ and $\hat{e} = E_x - \alpha$. If $\hat{m} \geq 1$, then $\hat{m}$ is the mantissa of the result, while $\hat{e}$ is its exponent. If $\hat{m} < 1$, then multiply $\hat{m}$ by $2^{2^{-k}}$ and subtract 2^{-k} from the new computed value $\hat{e}$: This gives the mantissa and the exponent of the result—if $k > n/2 + 2$, this last multiplication can be reduced to an addition.

4.5.2 From the SLNS Representation to the Floating-Point Representation

Let x be represented in the SLNS system of parameter k by m_x and e_x. We want to find the mantissa M_x ($1 \leq M_x < 2$) and the (integer) exponent E_x of the binary floating-point representation of x. This can be done as follows:

1) Define $e_f = \text{frac}(e_x)$, look up for 2^{e_f} in a table with k address bits (or compute it) ;
2) Multiply m_x by 2^{e_f} (this is the multiplication of an $n - k$-bit number by a k-bit number), this gives a number M^*. Define $E^* = \lfloor e_x \rfloor$;
3) M^* is between one and $2^{1-2^{-k}}(1 + 2^{-k})$. If $M^* < 2$, then $M_x = M^*$ and $E_x = E^*$. If $M^* \geq 2$ (this case is very unlikely to occur, since the upper bound on M^* is very close to two), then M_x is obtained by shifting M^* by one position to the right, and E_x is equal to $E^* + 1$.

5 Static Accuracy of the Semi-Logarithmic Number System

In this section, we evaluate the Maximum Relative Representation Error (MRRE) and the Average Relative Representation Error (ARRE) [5] of the semi-logarithmic number systems. We assume that numbers are represented in the **canonical** form (since the general form is redundant, it is much more difficult to define the representation errors of that form). We perform the computations for the case of the "rounding-to-zero" mode. In the other cases, the computations are very similar. For the evaluation of the average errors, we assume Hamming's *logarithmic distribution* of numbers [7], [4], [18], [19]. That is, we assume the density function

$$P(x) = \frac{1}{x \ln 2}, \quad \text{where } 1 \leq x < 2.$$

We will compare the SLNS of parameters k and n with

- a floating-point system with n mantissa bits (the first "1" is not included);
- an LNS with n bits in the fractional part of the fixed-point representation.

5.1 Maximum Relative Representation Error (MRRE)

Assume x is between one and two. We have:

$$\left|\frac{x - Z(x)}{x}\right| = \frac{x - \dfrac{\left\lfloor 2^n \times \dfrac{x}{2^{\left\lfloor 2^k \log_2 x \right\rfloor / 2^k}} \right\rfloor}{2^n} \times 2^{\left\lfloor 2^k \log_2 x \right\rfloor / 2^k}}{x}.$$

Let us define Δ_c as the domain where $\lfloor 2^k \log_2 x \rfloor / 2^k$ equals c. That is, $\Delta_c = [2^c, 2^{c+2^{-k}})$. In that domain, $\left|\frac{x - Z(x)}{x}\right|$ is equal to

TABLE 1
ARRE AND MRRE OF THE SEMI-LOGARITHMIC NUMBER SYSTEMS FOR DIFFERENT VALUES OF k

	Rounding to zero		Rounding to nearest	
	MRRE	ARRE	MRRE	ARRE
Floating Point	2^{-n}	0.36×2^{-n}	2^{-n-1}	$0.36 \times 2^{-n-1}$
SLNS ($k \geq 2$)	2^{-n}	0.5×2^{-n}	2^{-n-1}	$0.5 \times 2^{-n-1}$
Logarithmic	0.69×2^{-n}	0.35×2^{-n}	$0.69 \times 2^{-n-1}$	$0.35 \times 2^{-n-1}$

$$\left(\frac{2^n x}{2^c} - \left\lfloor \frac{2^n x}{2^c} \right\rfloor\right) \times \frac{2^c}{2^n x}. \tag{4}$$

From this, we deduce:

$$MRRE = \max_{x \in [1,2]} \left| \frac{x - Z(x)}{x} \right|$$
$$\approx \max_{c=0,\ldots,1} \max_{x \in \left[2^c, c^{c+1/2^k}\right)} \left(\frac{2^n x}{2^c} - \left\lfloor \frac{2^n x}{2^c} \right\rfloor\right) \times \frac{2^c}{2^n x}.$$

Let us estimate $\mu = \max_{x \in [2^c, 2^{c+1/2^k})} \left(\frac{2^n x}{2^c} - \left\lfloor \frac{2^n x}{2^c} \right\rfloor\right)$. From $x \in [2^c, 2^{c+1/2^k})$, we easily deduce $2^n x / 2^c \in [2^n, 2^{n+2^{-k}})$. The upper bound $2^{n+2^{-k}}$ is approximately equal to $2^n + 2^{n-k} \ln 2$. Therefore:

- If $n > k$, then the interval $[2^n, 2^{n+2^{-k}})$ contains at least two integers, hence, $\mu = 1$. This gives
$$MRRE \approx \max_{c=0,1/2^k,\ldots,1} \max_{x \in \left[2^c, 2^{c+1/2^k}\right)} \frac{2^c}{2^n x} = 2^{-n}.$$
- If $n = k$ (i.e., if we actually use the logarithmic number system), then $\mu \approx \ln 2$. This gives $MRRE \approx 2^{-n} \ln(2)$.

As a consequence, the floating-point system and the semi-logarithmic system (with $k < n$) lead to the same value of the MRRE, while the radix-2 logarithmic number system has a slightly better MRRE, that is $2^{-n} \ln(2)$. The lack of continuity between the cases $k < n$ and $k = n$ (LNS) may seem strange. It is due to the difference in the value of μ.

5.2 Average Relative Representation Error (ARRE)

We want to evaluate

$$ARRE = \int_1^2 \frac{1}{x \ln 2} \times \left| \frac{x - Z(x)}{x} \right| dx. \tag{5}$$

Using the domain Δ_c defined in the previous section, we find[1]:

$$\int_{\Delta_c} \frac{1}{x \ln 2} \times \left| \frac{x - Z(x)}{x} \right| dx \approx \int_{\Delta_c} \frac{1}{x \ln 2} \times \frac{1}{2} \times \frac{2^c}{2^n x} dx$$
$$\approx \frac{2^{c-n-1}}{\ln 2} \left(\frac{1}{2^c} - \frac{1}{2^{c+1/2^k}} \right).$$

The extremal possible values for c are 0 (for $x = 1$) and 1 (for $x = 2$). This gives (by defining i as $c \times 2^k$):

$$ARRE \approx \sum_{i=0}^{2^k} \frac{2^{i \times 2^{-k} - n - 1}}{\ln 2} \times \frac{2^{2^{-k}} - 1}{2^{(i+1) \times 2^{-k}}}.$$

Therefore,

$$ARRE \approx \sum_{i=0}^{2^k} \frac{2^{-2^{-k} - n - 1}}{\ln 2} \times \left(2^{2^{-k}} - 1\right)$$
$$\approx 2^k \times \frac{2^{-2^{-k} - n - 1}}{\ln 2} \times \left(2^{2^{-k}} - 1\right)$$
$$\approx 2^{-n-1},$$

using $2^{2^{-k}} \approx 1 + 2^{-k} \ln 2$. This approximation is not valid for small values of k (say, for $k \leq 1$). For $k = 0$ (i.e., for the floating-point representation), the ARRE is equal to

$$\frac{2^{-n-2}}{\ln 2} \approx 0.36 \times 2^{-n}.$$

For the radix-2 logarithmic number system, the ARRE is equal to $2^{-n-1} \ln(2)$. Table 1 sums up the different values of the maximum and average relative representation error for various cases. An immediate conclusion from this table is that, although the floating-point and the logarithmic number systems are slightly better than the semi-logarithmic number systems, all these systems lead to approximately the same accuracy: The ratio of the ARRE of the SLNS system to the ARRE of the logarithmic system is $1/\ln(2) \approx 1.4$. This corresponds to $\log_2(1/\ln(2)) \approx 1/2$ bit of accuracy.

6 CONCLUSION

We have proposed a new class of number systems, called *semi-logarithmic number systems*. They constitute a compromise between the floating point and the logarithmic number systems: If the parameter k is larger than $n/2 + 2$, multiplication and division are almost as easily performed as in the logarithmic number systems, whereas addition and subtraction require much smaller tables. For instance, with $n = 23$, $k = 15$, and $j = 9$, we would require a small 24 kb

1. To get this, we replace $\left(\frac{2^n x}{2^c} - \left\lfloor \frac{2^n x}{2^c} \right\rfloor\right)$ by its average value. This approximation is valid if 2^k is small compared to 2^n. In practice, this holds as soon as k is less than $n - 3$.

ROM. The best value for k must result from a compromise: If k is large, the tables required for addition may become huge and, if k is small, the algorithms become complicated. Values of k slightly larger than $n/2$ are probably the best choice. Although the semi-logarithmic number systems are slightly less accurate than the floating-point and the logarithmic number systems, the difference is very small (roughly speaking, 1/2 bit of accuracy). The domain of application of the semi-logarithmic number systems is the same as that of the logarithmic number systems: Special purpose processors for solving problems where the ratio of multiplies (or divides, or square roots) to adds is relatively high.

Acknowledgments

This is an expanded version of a paper that was presented at the 12th Symposium on Computer Arithmetic (ARITH12), Bath, U.K., July 1995.

References

[1] S.F. Anderson, J.G. Earle, R.E. Goldschmidt, and D.M. Powers, "The IBM 360/370 Model 91: Floating-Point Execution Unit," *IBM J. Research and Development*, Jan. 1967. Reprinted in E.E. Swartzlander, *Computer Arithmetic*, vol. 1. IEEE CS Press Tutorial, 1990.

[2] M.G. Arnold, T.A. Bailey, J.R. Cowles, and J.J. Cupal, "Redundant Logarithmic Number Systems," *Proc. Ninth Symp. Computer Arithmetic*, M.D. Ercegovac and E.E. Swartzlander, eds., pp. 144–151, Santa Monica, Calif., Sept. 1989.

[3] M.G. Arnold, T.A. Bailey, J.R. Cowles, and M.D. Winkel, "Applying Features of IEEE 754 to Sign/Logarithm Arithmetic," *IEEE Trans. Computers*, vol. 41, no. 8, pp. 1,040–1,050, Aug.1992.

[4] J.L. Barlow and E.H. Bareiss, "On Roundoff Error Distributions in Floating-Point and Logarithmic Arithmetic," *Computing*, vol. 34, pp. 324–347, 1985.

[5] W.J. Cody, "Static and Dynamic Numerical Characteristics of Floating-Point Arithmetic," *IEEE Trans. Computers*, vol. 22, no. 6, pp. 598–601, June 1973.

[6] D. Goldberg, "What Every Computer Scientist Should Know About Floating-Point Arithmetic," *ACM Computing Surveys*, vol. 23, no. 1, pp. 5–47, Mar. 1991.

[7] R.W. Hamming, "On the Distribution of Numbers," *Bell Systems Technical J.*, vol. 49, pp. 1,609–1,625, 1970. Reprinted in E.E. Swartzlander, *Computer Arithmetic*, vol. 1. IEEE CS Press Tutorial, 1990.

[8] H. Henkel, "Improved Addition for the Logarithmic Number Systems," *IEEE Trans. Acoustics, Speech, and Signal Processing*, vol. 37, pp. 301–303, 1989.

[9] N.G. Kingsbury and P.J.W. Rayner, "Digital Filtering Using Logarithmic Arithmetic," *Electronic Letters*, vol. 7, pp. 56–58, 1971. Reprinted in E.E. Swartzlander, *Computer Arithmetic*, vol. 1. IEEE CS Press Tutorial, 1990.

[10] D.M. Lewis, "An Accurate LNS Arithmetic Using Interleaved Memory Function Interpolator," *Proc. 11th Symp. Computer Arithmetic*, M.J. Irwin, E.E. Swartzlander, and G. Jullien, eds., pp. 2–9, June 1993.

[11] D.W. Matula and P. Kornerup, "Finite Precision Rational Arithmetic: Slash Number Systems," *IEEE Trans. Computers*, vol. 34, no. 1, pp. 3–18, Jan. 1985.

[12] S. Matsui and M. Iri, "An Overflow/Underflow Free Floating-Point Representation of Numbers," *J. Information Processing*, vol. 4, no. 3, pp. 123–133, 1981. Reprinted in E.E. Swartzlander, *Computer Arithmetic*, vol. 2. IEEE CS Press Tutorial, 1990.

[13] F.W.J. Olver, "A Closed Computer Arithmetic," *Proc. Eighth IEEE Symp. Computer Arithmetic*, May 1987. Reprinted in E.E. Swartzlander, *Computer Arithmetic*, vol. 2. IEEE CS Press Tutorial, 1990.

[14] T. Stouraitis and F.J. Taylor, "Floating-Point to Logarithmic Encoder Error Analysis," *IEEE Trans. Computers*, vol. 37, pp. 858–863, 1988.

[15] E.E. Swartzlander and A.G. Alexpoulos, "The Sign-Logarithm Number System," *IEEE Trans. Computers*, Dec. 1975. Reprinted in E.E. Swartzlander, *Computer Arithmetic*, vol. 1, IEEE CS Press Tutorial, 1990.

[16] F.J. Taylor, "An Extended Precision Logarithmic number System," *IEEE Trans. Acoustics, Speech, Signal Processing*, vol. 31, p. 231, 1983.

[17] F.J. Taylor, R. Gill, J. Joseph, and J. Radke, "A 20 bit Logarithmic Number System Processor," *IEEE Trans. Computers*, vol. 37, no. 2, pp. 190–200, Feb. 1988.

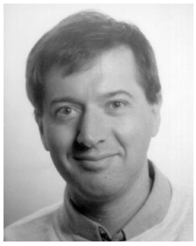
[18] P.R. Turner, "The Distribution of Leading Significant Digits," *IMA J. Numerical Analysis*, vol. 2, pp. 407–412, 1982.

[19] P.R. Turner, "Further Revelations on LSD," *IMA J. Numerical Analysis*, vol. 4, pp. 225–231, 1984.

[20] P.R. Turner, "Implementation and Analysis of Extended SLI Operations," *Proc. 10th IEEE Symp. Computer Arithmetic*, P. Kornerup and D. Matula, eds., pp. 118–126, June 1991.

[21] P.R. Turner, "Complex SLI Arithmetic: Representation, Algorithms, and Analysis," *Proc. 11th Symp. Computer Arithmetic*, M.J. Irwin, E.E. Swartzlander, and G. Jullien, eds., pp. 18–25 , June 1993.

[22] H. Yokoo, "Overflow/Underflow-Free Floating-Point Number Representations with Self-Delimiting Variable-Length Exponent Fields," *IEEE Trans. Computers*, vol. 41, no. 8, pp. 1,033–1,039, Aug. 1992.



Jean-Michel Muller received the Engineer degree in applied mathematics and computer science in 1983 and the PhD in computer science in 1985, both from the Institut National Polytechnique de Grenoble, France. In 1986, he joined the CNRS (French National Center for Scientific Research). From 1986 to 1989, he was posted to the Tim3-Imag Laboratory, Grenoble, and then to the LIP Laboratory, Lyon. He teaches computer arithmetic at the Ecole Normale Supérieure de Lyon. His research interests include computer arithmetic and computer architecture. Dr. Muller served as general chairman of the 10th Symposium on Computer Arithmetic (Grenoble, France, June 1991) and as co-program chair of the 13th Symposium on Computer Arithmetic (Asilomar, California, July 1997). He has been an associate editor of *IEEE Transactions on Computers* since 1996. He is a member of the IEEE.

Alexandre Scherbyna received the Engineer degree and the PhD degree in computer science from the Kiev Institute of Technology in 1977 and 1983, respectively. He has been with the Kiev Technical University since 1980, where he teaches CAD. His research interests include parallelism and computer arithmetic.

Arnaud Tisserand received the MSc degree and the PhD degree in computer science from the École Normale Supérieure de Lyon, France, in 1994 and 1997, respectively. He is with the Laboratoire de l'Informatique du Parallélisme (LIP) in Lyon, France. He teaches computer architecture and VLSI design at the École Normale Supérieure de Lyon, France. His research interests include computer arithmetic, computer architecture, and VLSI design.