A Bit-Serial Arithmetic Unit for Rational Arithmetic

Peter Kornerup
Aarhus University
Aarhus, Denmark

David W. Matula*
Southern Methodist University
Dallas, Texas

Abstract

We describe a binary implementation of an algorithm of Gosper to compute the sum, difference, product, quotient and certain rational functions of two rational operands applicable to integrated approximate and exact rational computation. The arithmetic unit we propose is an eight register computation cell with bit serial input and output employing the binary lexicographic continued fraction (LCF) representation of the rational operands. The operands and results are processed in a most-significant-bit first on-line fashion with bit level logic leading to less delay in the computation cell when compared to operation on the full partial quotients of the standard continued fraction representation. Minimization of delay is investigated with the aim of supporting greater throughput in cascaded parallel computation with such computation cells.

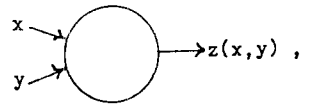
I. Introduction

This paper presents an arithmetic unit, and in particular its binary implementation, which can take as input two bitstreams and produce as output a bitstream representing the result. The bitstreams are consumed and produced on-line most-significant-bit first, as representations of rational numbers utilizing their continued fraction expansions. The arithmetic unit can realize a variety of dyadic operations supplementing the standard add, subtract, multiply and divide operations. The general form of the permissible primitive operations is given by

$$z(x,y) = \frac{axy + bx + cy + d}{exy + fx + gy + h}$$

where a,b,c,d,e,f,g,h are arbitrary prespecified integers. This allows as a more general example computation of Axy + B as one primitive dyadic operation, given the prespecified rational constants A and B.

The arithmetic unit can be envisioned as a bit-serial, precision driven computation cell:



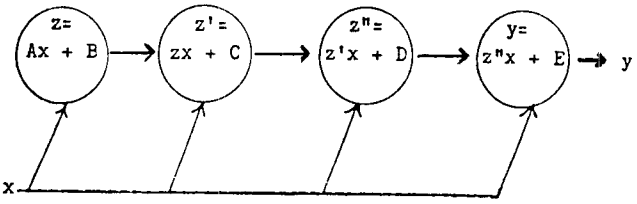
where z(x,y) can be computed, most-significant-bit first, by pulling in information about x and y most-significant-bit first. The unit can request information from x and y as needed to support the computation of z(x,y) to the precision desired. Such a unit synergistically supports both exact rational and finite precision approximate real computation, of importance to computational environments requiring integration of symbolic and numeric procedures.

Multiple units may be cascaded to support more general computational needs, in general forming a binary tree structure analogous to the parsing tree of an expression in a programming language. In such a tree the result appears at the root, with input data supplied bit-serially (as requested and only as needed) at the leaves.

As an example of such a cascading of units in pipeline form, consider the evaluation of a polynomial with rational coefficients using the Horner scheme:

$$y = (((Ax + B)\,x + C)x + D)\,x + E$$

which can be evaluated by the structure:



A considerable literature exists for on-line arithmetic (e.g. see [TE77,Er84]) in the context of fixed-radix digital representation. Our methods effectively extend the features of on-line arithmetic to computation with rational values, affording the advantages of convenient short exact representation of simple rationals within a variable precision approximate system. For many applications a finite precision representation based on rational numbers is an attractive alternative to floating point numbers, as data and

*This research was supported by the National Science Foundation under grant DCR-8315289.

results may be known to be rational. Symbolic and algebraic systems currently support exact rational arithmetic in the form of unlimited precision integer representation of the numerators and denominators of rational fractions, with only limited and inconvenient inclusion of numerically approximate real computation. Finite precision arithmetic on rational operands, also in integer numerator/denominator form, has been proposed and analysed in [MK80,KM83,MK85].

This paper demonstrates the possibility of supporting arithmetic on rational numbers represented as continued fractions, considered as sequences of integers and appropriately encoded as bitstrings, and applicable concurrently to both finite precision and exact rational computation.

The theory of continued fractions has been extensively developed by mathematicians [HW79] yielding some tools of application to computational practitioners, e.g. the Pade approximations. However, direct computation with continued fraction representation of numbers was not considered tractable until quite recently in "mathematical research time." In a classic number theory monograph written as recently as 1935, A. Y. Khinchin [Kh63] states:

> "On the other hand, for continued fractions there are no practically applicable rules for arithmetical operations: even the problem of finding the continued fraction for a sum from the continued fractions representing the addends is exceedingly complicated, and unworkable in computational practice."

Whether Khinchin knew of any means of doing arithmetic on continued fractions and considered such means computationally intractable, or whether he did not know of any such methods is unknown to these authors. In 1972, in a memo from MIT's AI lab, Gosper [Go72, see also p. 360 of Kn81] published an algorithm for performing arithmetic on continued fraction represented operands. Although appearing too complex for hand calculations the method is quite tractable for computer implementation, in particular becoming surprisingly competitive to traditional methods when implemented using VLSI. A more recent presentation of Gosper's idea is given in [Se83].

The purpose of this paper is to extend Gosper's algorithm, which is based on manipulating the integer valued partial quotients of a continued fraction expansion, to a bit level algorithm based on binary LCF-representation [MK83,KM85] which is ideally suited to this application.

In Section II we survey Gosper's algorithm, and explain the selection procedure used to determine the next partial quotient of $z(x,y)$. Section III then investigates a binary implementation of the unit as an eight register computation cell, utilizing the LCF-representation of continued fractions as a binary encoding for input and output. A suitable matrix notation is introduced to describe the primitive shift, add and subtract operations to be performed on the internal register contents, and to describe the details of the selection procedure for determining the individual bits to be output. The delay between input and output is discussed, and in most cases this is seen to be reasonably small. We describe certain "worst case" circumstances where the delay could still become quite large. We note that such situations could then be handled by incorporating redundancy into the representation, but this subject is not pursued in this paper.

Section IV discusses various time and space considerations utilizing previous analyses of rational computation and using heuristic arguments supported by simulations.

II. The Basic Algorithm

Following an idea of Gosper [Go72] we are interested in evaluating a general expression of the form:

$$z(x,y) = \frac{axy + bx + cy + d}{exy + fx + gy + h} \qquad (1)$$

where a,b,c,d,e,f,g,h are integers. By appropriate choices of these constants, the form can be used to compute $x+y$, $x-y$, $x*y$, x/y, and other expressions in x and y.

As pointed out by Gosper, it is possible to enter x and y into the computation of $z(x,y)$, by utilizing the continued fraction expansions of x and y. Let the c.f. expansion of x be given by

$$x = [a_0/a_1/.../a_n]$$

where $a_i \geq 1$ for $i=1,2,...,n$ whenever $a_0 \neq 0$, and $a_i \geq 1$ for $i=2,3,...n$ when $a_0=0$ and $a_1 \neq 0$, so then

$$x = a_0 + \cfrac{1}{a_1 + \cfrac{1}{a_2 + \cfrac{1}{\ddots + \cfrac{1}{a_n}}}} .$$

We thus consider $[a_0/a_1/.../a_n]$ as the operable representation of x, and enter the value of x into the computation by successively entering the partial quotients $a_0, a_1, ..., a_n$, in that order. Notice that we may associate the sign of x with the first non-zero partial quotient, a_0 or a_1.

Entering a partial quotient p corresponds to the substitution $x = p + 1/x'$ which transforms the expression (1) into

$$z(x',y) = \frac{(pa+c)x'y + (pb+d)x' + ay + b}{(pe+g)x'y + (pf+h)x' + ex + f} . \qquad (2)$$

Notice that if p is chosen as the first partial quotient of x, then $x' = [a_1/a_2/.../a_n] \geq a_1$, i.e. x' contains the remaining partial quotients of x.

Equivalently if we substitute $y = q + 1/y'$ into (1) we obtain:

$$z(x,y') = \frac{(qa+b)xy' + ax + (qc+d)y' + c}{(qe+f)xy' + ex + (qg+h)y' + g} . \qquad (3)$$

Now notice that (2) and (3) have the same form as (1), where the eight (integral) constants have been transformed by simple linear transformations using the partial quotient of x and/or y as input.

Since we want to perform arithmetic on continued fractions it is necessary to transform z(x,y) into its continued fraction expansion for output. Let us thus rewrite (1) into

$$z(x,y) = r + \frac{1}{z'(x,y)} \qquad (4)$$

where

$$z'(x,y) = \frac{exy + fx + gy + h}{(a-re)xy + (b-rf)x + (c-rg)y + (d-rh)} \qquad (5)$$

which again has the same form as (1).

To be able to accept r as the first partial quotient of z(x,y), it is necessary that (4) is satisfied with $|z'(x,y)| \geq 1$, which is the selection procedure to be discussed.

We want to realize an on-line algorithm for the computation of the c.f. expansion of z(x,y), given the c.f. expansions of x and y. Hence we want to output partial quotients of z as soon as possible, before we have read all the partial quotients of the initial x and y. At any stage of the on-line algorithm, x and y in (1) are taken to denote the values of the remaining continued fractions of the two input arguments, and z(x,y) is the value of the remaining part of the original expression, possibly after some leading partial quotients have been output.

The selection procedure we want will attempt to determine a next partial quotient of z, utilizing only the 8-tuple of integer constants in (1), i.e. not using any further information about the actual values of x and y. Since the value of a c.f. is greater than its first partial quotient, we know that $x \geq 1$ and $y \geq 1$, except possibly initially (a_0 may take any value, and if $a_0 = 0$ then a_1 may take any non zero value). We avoid any initialization problem by delaying any attempt to output until the first non zero partial quotients of each argument have been input. Note that if the range of the function z(x,y) is within the interval $r \leq z(x,y) < r + 1$ over the domain: $x \geq 1$ and $y \geq 1$, then certainly the next partial quotient is r.

Assume for the moment that z(x,y) is well-defined, i.e. the denominator is non-zero. Then after having input sufficiently many partial quotients of x and y, z(x,y) will be a monotonic function of x and y over the domain of the remaining "tails" of x and y. For the selection of the next partial quotient r of z(x,y) it is then sufficient to consider the values of z(x,y) at the four extremes of x and y over their ranges:

$$z(1,1) = \frac{a+b+c+d}{e+f+g+h}, \quad z(1,\infty) = \frac{a+c}{e+g},$$
$$z(\infty,1) = \frac{a+b}{e+f}, \quad z(\infty,\infty) = \frac{a}{e}, \qquad (6)$$

where for simplicity we have used the symbol ∞ for the limiting values of x and y. Thus if there exists an r such that

$$r \leq \min(z(1,1), z(1,\infty), z(\infty,1), z(\infty,\infty)),$$
$$r + 1 > \max(z(1,1), z(1,\infty), z(\infty,1), z(\infty,\infty)),$$

then r is the next partial quotient of z(x,y), and z(x,y) can be reduced by (4) and (5) into z'(x,y).

Recalling the definition of a finite continued fraction, we may add an "end-marker" in the form of the symbol ∞ as an extra last partial quotient. Thus an empty "tail" of x or y may be interpreted as the value "∞". In the selection procedure, if x or y have been exhausted, then only two of the extreme values have to be considered. Finally, if both x and y have been exhausted, the c.f. expansion of $z(\infty,\infty) = a/e$ provides the remaining partial quotients.

Before returning to the problem of when we can assume that z(x,y) is a monotonic function, let us consider an example. Here we will record the eight integer coefficients of z(x,y) in a 2×2×2 array, which will be written in the following notation:

$$\begin{matrix} d & b \\ h & f \end{matrix}$$
$$\begin{matrix} c & a \\ g & e \end{matrix}$$

Example: Let us compute the value of z(x,y) = x - y where x = 25/9 = [2/1/3/2] and y = 8/3 = [2/1/2]. The computation will be recorded in a table, initialized in the upper left-hand corner with the 2×2×2 array which corresponds to the computation of z(x,y) = x - y. The table below just records the input of x and y, no partial quotients of z are removed as it is difficult to display the process in the third dimension.

x-y	x->		2	1	3	2	-
	0	1	2	3			
y	1	0	1	-			
	-1	0	-1	-1			
	0	0	0	0			
2			0	1	3		
			1	1	4		
1			-1	0	-1	-2	
			1	1	4	9	
				1	1	3	
2				3	12	27	
-							

At the stage of the computation indicated by the dotted box, we may observe that:

$$z(1,1) = \frac{0}{4}; \quad z(1,\infty) = -\frac{1}{2}; \quad z(\infty,1) = \frac{1}{2}; \quad z(\infty,\infty) = \frac{0}{1}.$$

Hence it is possible to conclude that $-1/2 \leq z(x,y) \leq 1/2$ since $z(x,y)$ does not have any singularities. It is then possible to determine that the first partial quotient of $z(x,y)$ is 0, and the second of absolute value of at least 2, however it is not yet possible to determine the sign of z.

After input of one more quotient in the x-direction (of value 3) and the last of y (of value 2), utilizing that y now has been exhausted it is sufficient to consider $z(1,\infty) = 2/15$ and $z(\infty,\infty) = 1/12$. Thus $1/12 \leq z(x,y) \leq 2/15$ and it is now known that the first partial quotient is 0, the next is positive and between 7 and 12 (recall that removing the partial quotient zero just interchanges the roles of numerator and denominator, and this has not been done).

Finally, in the lower right-hand corner $z(\infty,\infty) = 3/27$ represents the result of the computation $x - y = 25/9 - 8/3 = 3/27 = [0/9]$. It is easy to confirm that any entry $\binom{u}{v}$ in the table represents the numerator u and denominator v of $u/v = z(x,y)$, where x and y are the values of the continued fractions input so far. □

Returning to the question of the well-definedness of $z(x,y)$ it is obvious, because of the numerator-denominator symmetry, that this problem is equivalent to the problem of zero values of $z(x,y)$, and hence to the problem of determining the sign of the first non-zero partial quotient of $z(x,y)$. If a root-curve of either the numerator or the denominator (or both) extends into the domain of (x,y), corresponding to the "tails" of x and y, then obviously the sign of $z(x,y)$ cannot be determined. Reading the next partial quotient, say p of x, restricts the domain of the tail of x to the interval $(p,p+1]$, which is by the substitution $x = p + 1/x'$ mapped into the interval $[1,\infty)$. Thus looking at the four values $z(1,1)$, $z(1,\infty)$, $z(\infty,1)$ and $z(\infty,\infty)$ can only safely determine the sign and possibly the value of the next partial quotient of $z(x,y)$, if no root-curve (neither of the numerator nor the denominator) extends into the domain of the remaining tails of x and y. If one root curve passes through the domain, it may be possible to determine a leading partial quotient of value zero, as in the example above. If root-curves of both the numerator and the denominator extend into the domain, nothing can be concluded from the four extreme values, as the sign-changes may cancel out.

For our selection procedure to work, it is thus necessary to assume that either the numerator or the denominator is non zero over the domain determined by those partial quotients of x and y that have been read before applying the selection procedure. It should be emphasized that employing the unit to realize any of the standard arithmetic operations $\{+,-,\times,\div\}$ can yield no such intermediate problem. It is only in formulating more complex functions with the cell that this issue must be resolved.

Of concern for standard arithmetic $\{+,-,\times,\div\}$ is the observation that we should anticipate a large delay in the output of any partial quotient of $z(x,y)$ of large absolute value. As indicated by the example, it might be possible to make conclusions on the order of magnitude of the next partial quotient, without even being able to determine the sign. But if our objective is to output the correct partial quotient, we must occasionally experience large delays which will be compounded in any cascaded sequence or tree of computation cells. In the next section we show that the anticipated delay can be substantially reduced if we read and emit partial quotients "bitwise" in a suitable binary representation.

III. A Binary Algorithm

The previous section described the computation of the expression $z(x,y)$ in terms of integer partial quotients, as introduced by Gosper [Go72]. We now introduce an equivalent algorithm utilizing a binary encoding of the partial quotients, in particular the LCF-representation introduced in [MK83] and [KM85].

For the purpose of the algorithm described in Section II it is initially sufficient to notice that the LCF-representation of a continued fraction utilizes a self-delimiting binary representation of the integer partial quotients. This representation is composed of a unary encoding of the length of the standard binary representation, followed by the binary representation of the integer partial quotient with the leading bit inverted to indicate the switch.

More precisely, if $[p]_2 = 1\, b_{n-1}\ldots b_1 b_0$ is the standard binary representation of $p \geq 1$, then

$$\ell(p) = 1^n\, 0\, b_{n-1}\ldots b_1 b_0 \quad (\text{so } \ell(1) = 0)$$

is the encoding to be used. This representation turns out to be particularly well suited to the purpose of realizing the arithmetic unit as a computation cell. In particular, the bits of $\ell(p)$ may be individually interpreted as encodings of primitive shift, shift-and-add or shift-and-subtract operations, performed in a binary implementation of the algorithm.

In the following it will be necessary to employ a variety of variable substitutions in the expression (1) of $z(x,y)$ to correspond to these bit level transformations. A more general substitution of the variable x is given by

$$x = \frac{\gamma + \alpha x'}{\delta + \beta x'} \tag{7}$$

which when substituted into $z(x,y)$ given by (1) yields the following expression:

$$z(x',y) = \frac{(\alpha a+\beta c)x'y+(\alpha b+\beta d)x'+(\gamma a+\delta c)y+(\gamma b+\delta d)}{(\alpha e+\beta g)x'y+(\alpha f+\beta h)x'+(\gamma e+\delta g)y+(\gamma f+\delta h)} \tag{8}$$

which again has the same form as (1).

To describe the more complicated substitution as a sequence of simple bit level transformations, it is convenient to introduce the following matrix notation. With a straight-forward interpretation of matrix multiplication, where one operand is our 2×2×2 array and the other is a standard 2×2 matrix, we denote the substitution of (7) into (1) yielding (8) as follows:

$$\left\{\begin{pmatrix} d & b \\ h & f \end{pmatrix}\begin{pmatrix} c & a \\ g & e \end{pmatrix}\right\}\begin{pmatrix} \delta & \beta \\ \gamma & \alpha \end{pmatrix} = \left\{\begin{pmatrix} \delta d+\gamma b & \beta d+\alpha b \\ \delta h+\gamma f & \beta h+\alpha f \end{pmatrix}\begin{pmatrix} \delta c+\gamma a & \beta c+\alpha a \\ \delta g+\gamma e & \beta g+\alpha e \end{pmatrix}\right\}$$

The standard substitution from Section I, $x = p + 1/x'$ may thus be described by the matrix

$$\begin{pmatrix} 0 & 1 \\ 1 & p \end{pmatrix}$$

and the substitution $y = q + 1/y'$ may be described by a similar matrix. For y either the multiplication has to be performed "from below," or "the usual way" on a properly transposed version of the 2×2×2 array.

For the substitution $x = p + 1/x'$, with $\ell(p) = 1^n\ 0\ b_{n-1}\ b_{n-2}\cdots b_1 b_0$, $\ell(1) = 0$, it is easy to see that:

$$\begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}\begin{pmatrix} 2 & 1 \\ 0 & 1 \end{pmatrix}^n \begin{pmatrix} \frac{1}{2} & \frac{b_{n-1}}{2} \\ 0 & 1 \end{pmatrix} \cdots \begin{pmatrix} \frac{1}{2} & \frac{b_0}{2} \\ 0 & 1 \end{pmatrix}$$

$$= \begin{pmatrix} 0 & 1 \\ 1 & (2^n + b_{n-1}2^{n-1} + \cdots + b_0) \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & p \end{pmatrix}. \qquad (9)$$

Notice the correspondence between the terms of (9) to the bits of $\ell(p)$. The initial (leftmost) matrix corresponds to the information "p is at least one", the following n matrices each tell us "I am at least twice as big," and add the denominator into the numerator, leftshifting the former. The zero in the middle of $\ell(p)$ has no counterpart matrix, but indicates a switch of operation. The n rightmost matrices each rightshift the denominator and (depending on the bit values) add it to the numerator, gradually building up num + p × den, where num and den are any of the four numerator-denominator pairs in the 2×2×2 array.

Suitably interpreted we may similarly input a partial quotient q of y, and perform the appropriate substitutions described by the matrices derived from $\ell(q)$. Note that the order in which bits from $\ell(p)$ or $\ell(q)$ (from x or y) are input is immaterial, since the corresponding variable substitutions can be performed in any order. The essential observation is, of course, that we are free to input bits from $\ell(p)$ when we need more information on x, and bits from $\ell(q)$ when we need more information on y, and, as we shall see below, to output bits of $\ell(r)$ where r is the next partial quotient of z(x,y).

To output bits of $\ell(r)$ the selection procedure of the previous section has to be modified, so as to allow us to determine bits of $\ell(r)$ from the four values $z(1,1)$, $z(1,\infty)$, $z(\infty,1)$ and $z(\infty,\infty)$.

Assume first that $r \geq 1$. If $r = 1$ can be determined, then output $\ell(r) = \ell(1) = 0$. If $z(x,y) \geq 2$ can be determined, we would like to eject one of the leading (unary) bits of $\ell(r)$ and divide z(x,y) by 2, and continue doing so while we can assure that $z(x,y) \geq 2$. This corresponds to repeatedly performing the transformation $z(x,y) = 2\,z'(x,y)$ which can be performed by multiplying by the matrix:

$$\begin{pmatrix} 1 & 0 \\ 0 & 2 \end{pmatrix}$$

in the appropriate direction (corresponding to multiplying all denominators by 2, or leftshifting these).

When, however, we can determine that $1 \leq z(x,y) < 2$, we are ready to switch, as we have determined the leading bit of r. Ejecting a zero bit indicating the switch in $\ell(r)$, we want to perform the transformation $z(x,y) = 1 + z'(x,y)$, which can be achieved by multiplication with the matrix:

$$\begin{pmatrix} 1 & 0 \\ -1 & \frac{1}{2} \end{pmatrix}$$

This subtracts the denominator from the numerator, and rightshifts the former one position. Thus we have actually performed the transformation $z(x,y) = 1 + z'(x,y)/2$, readying our unit for the next step.

Whenever it is possible to determine that one of the following cases occurs, the unit can emit an extra bit of $\ell(r)$, rightshifting the denominator back towards its original position:

$\underline{1 \leq z(x,y) < 2}$: perform the transformation:

$$z(x,y) = 1 + \frac{1}{2}\,z'(x,y) \sim \begin{pmatrix} 1 & 0 \\ -1 & \frac{1}{2} \end{pmatrix}$$

and emit a one.

$\underline{z(x,y) < 1}$: perform the transformation:

$$z(x,y) = \frac{1}{2}\,z'(x,y) \sim \begin{pmatrix} 1 & 0 \\ 0 & \frac{1}{2} \end{pmatrix}$$

and emit a zero.

When the denominator has been shifted back into its original position, r has been completely determined. Then an interchange of numerators and denominators has to be performed, corresponding to the transformation $z(x,y) = 1/z'(x,y)$.

Combining the transformations described above actually yields one extra rightshift, which we may correct in conjunction with the final switch

obtaining

$$\begin{Bmatrix} 1 & 0 \\ 0 & 2 \end{Bmatrix}^n \begin{Bmatrix} 1 & 0 \\ -1 & \frac{1}{2} \end{Bmatrix} \begin{Bmatrix} 1 & 0 \\ -b_{n-1} & \frac{1}{2} \end{Bmatrix} \cdots \begin{Bmatrix} 1 & 0 \\ -b_0 & \frac{1}{2} \end{Bmatrix} \begin{Bmatrix} 0 & 1 \\ 2 & 0 \end{Bmatrix}$$

$$= \begin{Bmatrix} 0 & 1 \\ 1 & -(2^n + b_{n-1}2^{n-1} + \ldots + b_0) \end{Bmatrix} = \begin{Bmatrix} 0 & 1 \\ 1 & -r \end{Bmatrix},$$

corresponding to a combined transformation $z(x,y) = r + 1/z'(x,y)$.

Notice that all of the transformations performed during the selection procedure above are invariant under the variable substitutions performed when bits of the binary continued fraction representations of x and y are input by the unit. Whenever the selection procedure cannot determine the next bit to be emitted, input from x or y can be requested and fed into the unit, until the selection procedure is satisfied and ready to emit the next bit.

We have now seen how to handle positive partial quotients in the representation $\ell(\cdot)$, from the input of x and y, as well as the output of a partial quotient of z. A partial quotient of value zero can only occur as the first partial quotient, and it is easy to see that the matrix:

$$\begin{Bmatrix} 0 & 1 \\ 1 & 0 \end{Bmatrix}$$

will perform the necessary transformation corresponding to a zero quotient on input as well as on output. However, zero cannot be represented in the $\ell(\cdot)$ encoding, but has to be handled separately in the LCF-representation of continued fractions.

The LCF-representation is defined by [MK83,KM85]:

$$LCF\left(\frac{p}{q}\right) = \begin{cases} 1 \circ \ell(a_0) \circ \overline{\ell(a_1)} \circ \ldots \overline{\ell(a_{2n-1})} \circ \ell(a_{2n}) \circ \overline{\ell(\infty)} & \text{for } \frac{p}{q} \geq 1 \\ 0 \circ \overline{\ell(a_1)} \circ \ell(a_2) \circ \ldots \overline{\ell(a_{2n-1})} \circ \ell(a_{2n}) \circ \overline{\ell(\infty)} & \text{for } \frac{p}{q} < 1 \end{cases}$$

where $p/q = [a_0/a_1/a_2/\ldots/a_{2n-1}/a_{2n}]$ is in "terminal index even form." Notice that a leading zero partial quotient in the case $0 \leq p/q < 1$ is encoded as a zero, and the terminating $\overline{\ell(\infty)}$ is just an infinite string of zeroes. Only the leading non-zero part of LCF(p/q) need be represented for any rational number, implicitly assuming an infinite string of trailing zeroes. The complementation of the representation of the odd numbered partial quotients suffices to achieve lexicographic ordering, since the value of a continued fraction is an increasing function of the partial quotients in the even positions, and a decreasing function of the quotients in odd positions.

A signed LCF-representation was also introduced [MK83,KM85] by prepending a signbit together with complementation as follows:

$$SLCF\left(\frac{p}{q}\right) = \begin{cases} 1 \circ LCF\left(\frac{p}{q}\right) & \text{for } \frac{p}{q} \geq 0, \\ 0 \circ \widetilde{LCF\left(-\frac{p}{q}\right)} & \text{for } \frac{p}{q} < 0, \end{cases}$$

where $\tilde{\cdot}$ denotes 2's complementation. SLCF is again lexicographic order preserving since:

$$\widetilde{LCF\left(\frac{p}{q}\right)} = LCF\left(\frac{q}{p}\right) \quad \text{for } \frac{p}{q} > 0$$

due to the fact that 2's complementation, besides complementation, also changes a "terminal index even" into a "terminal index odd" continued fraction, and vice versa.

Recalling the example in Section II we notice that we cannot start emitting bits of SLCF(z) before the sign of z has been determined. In general we can emit bits of the LCF or SLCF representation of a partial quotient before the complete partial quotient has been determined. However, there are situations where there is still an inherent delay which cannot be avoided without a redundant representation.

Successive approximations to a rational number u/v, obtained by including the partial quotients successively, will oscillate around u/v. Hence the computed approximations of z(x,y), based on the leading partial quotients of x and y, will oscillate around the final value. Thus there might be situations where more input is continuously needed to decide whether the next partial quotient is going to be say r or r+1 because z(x,y) continuously oscillates around r+1, as more and more partial quotients are being read. Hence it is difficult to decide whether to output r+1 followed by a large partial quotient, or to output r, then 1 and then some equivalently large partial quotient.

As an example of this situation consider the product xy, where both x and y are rational approximations of $\sqrt{2}$, obtained by truncating the (infinite) continued fraction expansion:

$$\sqrt{2} = [1/2/2/2\ \ldots].$$

Until either x or y terminate it is not possible to determine whether the result is [2/k] or [1/1/m], where k and m are some large integers.

The corresponding LCF-representations are:

$$LCF([2/k]) = 1 \circ 10 \circ 0 \circ \overline{11\ldots10}\ldots$$

$$= 110000\ldots01\ldots$$

$$LCF([1/1/m]) = 1 \circ 0 \circ 0 \circ \overline{0} \circ 11\ldots10\ldots$$

$$= 10111\ldots10\ldots$$

which are approximations from above and below of LCF(2) = 1100... . Notice that these LCF representations are very close in the lexicographic ordering. This property of ordering is identical to the problem in standard radix representation to any base, e.g. with numbers like 9.9999... and 10.000... . As in radix representation the solution is to introduce redundant representation, e.g. signed digits. An analogous solution for redundancy in LCF-representation is being pursued, and will be presented in a later paper.

IV. Considerations Concerning Time and Space

For the implementation of an arithmetic unit as a "computation cell" along the lines described in the previous sections, it should be noticed that parallelism can be utilized to perform the linear transformations on four $\binom{u}{v}$ pairs concurrently. We will thus assume the existence of four such register pairs with a shift capability, and four add/subtract units which we assume can be connected appropriately. The capacity (bit-width) of the registers and add-subtract units affect the overall complexity of the cell and requires careful analysis.

The time-complexity of the partial quotient driven unit described in Section II, can be measured in terms of major cycles, given either by the input of a partial quotient of x, a partial quotient of y, or the output of a partial quotient of z(x,y). Each major cycle then again consists of a number of minor cycles, which will consist of a shift, or an add-subtract operation, used to implement the multiplications or division steps of the linear transformations.

The number of major cycles is thus the sum of the number of partial quotients of x and y and the number of partial quotients of z that are requested, which will depend on circumstances. If x and y are exact numbers then z(x,y) can be computed exactly, and all partial quotients of z(x,y) are significant. On the other hand, if x and y are imprecise numbers, it does not make sense to input more quotients of x and y than those that can be considered significant. This implies that it is sufficient only to pull out as many partial quotients of z(x,y) as can be extracted considering the domain of the "tails" of x and y to be indeterminate, i.e. only using the terminating symbol ∞, whenever x and/or y can be considered exact.

For finite precision arithmetic we may restrict our consideration to some subset of continued fraction represented rational numbers. For this purpose we will choose the set:

$$H_k = \left\{ \frac{p}{q} \middle| |p \times q| < k \right\},$$

which has certain desirable properties [MK80] in support of approximate real arithmetic. This then limits the set of representable continued fractions by restricting the numerators and denominators of their rational fraction equivalent, rather than by defining restrictions on the continued fraction expansions themselves. Notice that k determines a maximum bound on the length of the c.f. expansion, as well as the size of its partial quotients.

Using some heuristic arguments along the lines of those presented in [Kn81] and [KM78], it is possible to show that the average number of partial quotients of p/q grows asymptotically as

$$\frac{3(\ln 2)^2}{\pi^2} \log_2 k \sim 0.1460 \log_2 k$$

when p/q is chosen according to a log-uniform distribution from H_k.

Since the relative accuracy of rational numbers in H_k is of the order of one part in k, we may for finite precision arithmetic assume that x, y and z(x,y) all are restricted to H_k. We then conclude that the number of major cycles in computing the approximate answer z(x,y) is $3 \times 0.1460 \log_2 k \sim 0.44 \log_2 k$.

It is also easy to see that the average number of shifts (minor cycles) in total is $3 \times 1.5 \log_2 k$, however the number of add/subtract operations can be expected to be smaller. Following [KM78] it can be shown that the average number of add-subtract cycles in computing z(x,y) is

$$3 \times \left\{ \frac{3}{4} + \frac{3(\ln 2)^2}{2} \right\} \log_2 k \approx 2.688 \log_2 k$$

assuming that the operations on the four register pairs takes place in parallel in one of these cycles.

The same analysis applies to the LCF-represented operands; there is one "shift-matrix" for each shift to be performed, some of which also contain an add-subtract operation. Recalling that on input of x and y the number of matrix applications corresponds exactly to the number of shifts, and on output of z(x,y) there is one extra corresponding to the switch of numerators and denominators, we get a total of

$$(0.15 + 4.5)\log_2 k = 4.65 \log_2 k$$

matrix applications for a computation in the H_k system.

In the previous analysis we have implicitly assumed that it is possible to pull out partial quotients (or bits thereof) at about the same rate as they are being read from that operand (x or y) that has so far provided the fewest. This is the maximal rate we might expect from an information theoretic heuristic argument, and is confirmed by simple experiments, except for some of the special

cases discussed previously. The same heuristics also tell us that we might expect a growth of the size of the numbers in the registers at the same rate. This can again be found as the normal case in simple experiments. These heuristics thus indicate that it seems possible to support finite precision arithmetic over H_k, if the internal registers have a width of the order $1/2 \log_2 k$ bits. If x and y can be represented exactly in H_k, then using the terminating symbol ∞, the exact result z(x,y) could be obtained from the unit, even if z(x,y) does not belong to H_k. In these considerations we have implicitly assumed that the constants a,b,...,h are small (0's or $\pm$1's). If larger values are used then the registers and ALU's must have suitably larger capacity.

The final question to address is the delay between input of partial quotients from x and y, and the output of quotients of z(x,y). We must expect a varying delay when complete partial quotients are to be output, and a more smooth and hence smaller delay when output is produced bit-by-bit, except for the special case of 1000... or 0111... previously mentioned. This again can be seen from simple experiments.

Simulations to obtain empirical evidence on the delay were performed at both the partial quotient (or major cycle) level corresponding to the basic algorithm of Section II, and at the bitwise level corresponding to the binary LCF representation (or minor cycle) level of Section III. $\text{Log}_2(p \times q)$ for $p/q = [a_0/a_1/.../a_i]$ was used as a measure of the bit length for determining delays at the partial quotient input/output level, and the lengths of the LCF bit strings were used for measuring delays at the binary algorithm LCF-bit input/output level. We measured average delay in both cases to more fairly gauge the general improvement obtained by the binary algorithm, and the results are summarized below. For application we are of course more concerned with the maximum, as opposed to the average, delay. We believe a redundant form of LCF representation can be formed and employed to achieve approximately the same 4-bit delay in the worst case, comparable and possibly better than the average case for non-redundant form, and investigations in this area are continuing.

	Partial Quotient I/O (Basic Algorithm of Section II)	LCF Bitstream I/O (Binary Algorithm of Section III)
Bit Delay (average)	4.5 - 5.0	3.8 - 4.2

Acknowledgement

We wish to thank Steve Hickman and Sam Chen for developing the simulation programs and results.

References

[Er84] M. D. Ercegovac, "On-Line Arithmetic: An Overview," SPIE Vol 495 Real Time Signal Processing VII, 1984, pp 86-93.

[Go72] R. W. Gosper, Item 101 in HAKMEN, AIM239, MIT, Feb. 1972, pp 37-44.

[HW79] C. H. Hardy and E. M. Wright, "An Introduction to the Theory of Numbers," 5th ed., Oxford University Press, London, 1979.

[Kh63] A. Y. Khinchin, "Continued Fractions," 1935, Translated from Russian by P. Wynn, P. Noordhoff Ltd, Grooningen, 1963.

[Kn81] D. E. Knuth, "The Art of Computer Programming, Vol. 2, Seminumerical Algorithms," 2nd ed., Addison Wesley, 1981.

[KM78] P. Kornerup and D. W. Matula, "A Feasibility Analysis of Fixed-Slash and Floating-Slash Rational Arithmetic," Proc. 4th IEEE Symp. Comp. Arith., 1978, pp 39-47.

[KM83] P. Kornerup and D. W. Matula, "Finite Precision Rational Arithmetc: An Arithmetic Unit," IEEE-TC, Vol. C-32, No. 4, April 1983, pp 378-387.

[KM85] P. Kornerup and D. W. Matula, "Finite Precision Lexicographic Continued Fraction Number Systems," Proc. 7th IEEE Symp. Comp. Arith., 1985, pp 207-214.

[MK80] D. W. Matula and P. Kornerup, "Foundations of Finite Precision Arithmetic," Computing, Suppl. 2, 1980, pp 88-111.

[MK83] D. W. Matula and P. Kornerup, "An Order Preserving Finite Binary Encoding of the Rationals," Proc. 6th IEEE Symp. Comp. Arith., 1983, pp 201-209.

[MK85] D. W. Matula and P. Kornerup, "Finite Precision Rational Arithmetic: Slash Number Systems," IEEE-TC, Vol. C-34, No. 1, Jan. 1985, pp 3-18.

[Se83] R. B. Seidensticker, "Continued Fractions for High-Speed and High-Accuracy Computer Arithmetic," Proc. 6th IEEE Symp. Comp. Arith., 1983.

[TE77] K. S. Trivedi and M. D. Ercegovac, "On-line Algorithms for Division and Multiplication," IEEE-TC, Vol C-26, No. 7, July 1977, pp 681-687.