An Algorithm for Redundant Binary Bit-Pipelined Rational Arithmetic

PETER KORNERUP AND DAVID W. MATULA

***Abstract*—We introduce a redundant binary representation of the rationals and an associated algorithm for computing the sum, difference, product, quotient, and other useful functions of two rational operands, employing our representation. Our algorithm extends Gosper's partial quotient arithmetic algorithm and allows the design of an on-line arithmetic unit with computations granularized at the signed bit level. Each input or output port can independently be set to receive/produce operands/result in either binary radix or our binary rational representation. We investigate by simulation the interconnection of several such units for the parallel computation of more complicated expressions in a tree-pipelined manner, with particular regards to measuring individual and compounded on-line delays.**

***Index Terms*—Arithmetic unit, continued fraction, fine grained parallelism, on-line, radix, redundancy, signed bit.**

I. Introduction

THIS paper introduces redundancy into the representation of operands to be used as digit-serial input and produced as output of an on-line arithmetic unit for rational arithmetic. Previous investigations of an on-line rational arithmetic unit by several authors ([2], [7], [8], [12]) have considered only operands in nonredundant representations. As with well-known results for on-line units employing radix representation [1], [13], it is essential to introduce redundancy to bound the delay between input and output of such a unit.

The basic idea of this type of on-line rational arithmetic unit was suggested in a 1972 memo from MIT's AI lab by Gosper [2], [5, p. 360], [12] as an algorithm operating serially on the partial quotients of a continued fraction expansion [3], [4].

The arithmetic unit envisioned supports the standard operations of addition, subtraction, multiplication, division, and many other useful functions of two variables, expressed as

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

where a, b, c, d, e, f, g, and h are arbitrary prespecified integers. The unit thus operates as a digit serial, precision demand-driven cell, several of which can be interconnected to compute more complicated arithmetic expressions, in general in a tree structured pipelined computation.

A unified arithmetic algorithm providing the foundation for this unit may be described by just three steps related to the coefficient 8-tuple $Q = (a, b, c, d, e, f, g, h)$:

1) Initiation: Q is initiated corresponding to the desired arithmetic operation (e.g., $a = h = 1$ and $b = c = d = e = f = g = 0$ will make the unit a multiplier).

2) Transition Determination: A function $Zrange(Q)$ determines whether an element (bit or digit) is available for appending to the output stream, otherwise dictating input from the input stream of x or y. □

3) Execution: Each element of output to the stream for z, or input from the stream for x or y, is accompanied by a simple transformation of the 8-tuple Q.

Such an algorithm gives precedence to output over further input to speed up computational throughout in a pipelined sequence or interconnected network of units, each employing the algorithm. To bound the delays between input and output of each unit, it is necessary that the operands and results for the algorithm use a redundant representation.

The principal result of this paper is the design of a redundant binary version of this unified arithmetic algorithm. Input and output to our algorithm are given by signed bit string representations of x, y, and z which are processed left-to-right. Any rational is shown to have a finite length bit string representation, so that the computation cell will always terminate without cycling. The evaluation of $Zrange(Q)$ is efficient and all transformations of Q in our algorithm reduce to elementary shift and add/subtract operations on the elements of Q. This provides that given sufficient register lengths, each unit can operate in constant time at each step of processing arbitrary long input/output bit streams.

The generality of our algorithm is further demonstrated by showing that with only a small change in the transition rules for transformations of Q, input streams for x and y, in an appropriately self-normalized form of binary radix or redundant binary radix representation, may be directly input to the unit with implicit conversion. Output in redundant binary radix representation is similarly realizable, however, with the caveat that cycling must ensue for nonfinitely representable rational output.

In Section II, we derive a course grained version of our unified arithmetic algorithm. The input and output streams are here composed of integers, interpreted as signed partial quotients of redundant continued fraction expansions of the operands and result. The principal purpose of establishing this

Manuscript received November 2, 1989; revised March 1, 1990. This work was supported by the Danish Natural Science Research Council under Grant 11-7319 and by a grant from the Cyrix Corporation, Dallas, TX. Some of the results of this paper were presented at the 9th Symposium on Computer Arithmetic, September 1989, Santa Monica, CA.

P. Kornerup is with the Department of Mathematics and Computer Science, Odense Universitet, Odense, Denmark.

D. W. Matula is with the Department of Computer Science and Engineering, Southern Methodist University, Dallas, TX 75275.

IEEE Log Number 9036145.

algorithm is to provide a rigorous foundation for computation of the *Zrange*(Q) function. The resulting transformations to be applied to Q are shown to be elementary linear transformations conveniently illustrated by 2×2 matrix multiplications.

In Section III, a redundant binary continued fraction representation is introduced. The signed bits are shown to be in one-to-one correspondence with a refined factoring of the 2×2 linear transformations of the course grained algorithm. Resultant factors are limited to a set of eight particular 2×2 matrices, each containing only entries from $\{0, 1, \pm\frac{1}{2}, \pm 2\}$ and corresponding to a shift and add/subtract operation on the 8-tuple Q. The redundant binary algorithm is then described, and the variations for incorporating redundant binary radix input and/or output summarized.

Regularity of input/output delay for a unit employing this algorithm is investigated by simulation in Section IV. The compounded on-line delay of a tree structured pipelined computation is illustrated. Section V concludes with some open questions and directions for future research.

II. Serial Rational Arithmetic

The serial rational arithmetic algorithm developed in this section in extension of [2], [8] provides the essential foundation for our subsequent binary algorithm, and requires a suitable notion of redundant continued fraction representation.

Herein we allow a *redundant continued fraction expansion* of a rational number x to be given by any sequence of integer valued *signed partial quotients* $a_0, a_1, \cdots, a_k$, denoted by $x = [a_0/a_1/\cdots/a_k]$, for which x has the value

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

subject to the constraint that the *fractional part* f_i at level i satisfies

$$f_i = \cfrac{1}{a_{i+1} + \cfrac{1}{a_{i+2} + \cfrac{1}{\ddots + \cfrac{1}{a_k}}}} \quad \text{where } |f_i| < 1 \text{ for } 0 \le i \le k. \tag{1}$$

The requirement that the fractional parts f_i have magnitude less than unity assures that the truncated values $x_i = [a_0/a_1/\cdots/a_i]$ are good approximations, corresponding to rounding up or down by less than one ULP (unit in the last place). The fractional parts f_i may be more tightly bounded, resulting in fewer allowed redundant continued fraction representations as summarized here.

Observation 1: Imposing alternative fractional part bounds in (1) results in allowed redundant/nonredundant continued fraction expansions for rational values x as follows.

a) $|f_i| < p$ for $0 < p \le \frac{1}{2}$: Some values x would have no allowed expansions, e.g., $x = i + \frac{1}{2}$ for any integer i.

b) $|f_i| \le \frac{1}{2}$: All values x would have either one or two allowed continued fraction expansions, the latter corresponding only to a last partial quotient being ± 2.

c) $|f_i| < p$ for $\frac{1}{2} < p \le 1$: Progressively more redundant continued fraction representations would be allowed parametrically with increasing p, so that for $p = 1$ every nonintegral rational x would have at least two, but no more than a finite number of allowed redundant continued fraction expansions.

d) $|f_i| \le p$ for $p \ge 1$: Undesirable expansions would be allowed, such as nonunique zero $0 = [0] = [1/\bar{1}]$, and embedded strings carrying no information as shown by the identity $[a_0/1/\bar{1}/1/1/\bar{1}/1/a_1] = [a_0/a_1]$. □

Our requirement $|f_i| < 1$ in (1) thus allows maximal redundancy in continued fraction representation without the anomalies noted in Observation 1 d). Recall that anomalies in redundant radix representation are similarly avoided by requiring that truncation after any digit position yields a remaining fractional part of less than one ULP (this condition leads to the derived constraint that all digit values have magnitude less than b in a base b system). Redundant continued fractions still may contain arbitrarily large individual partial quotient values. Certain important restrictions on allowed subsequences of signed partial quotient values derive from the fractional part constraint (1), and should be noted for the purpose of recognizing a redundant continued fraction from a presumed sequence of partial quotients.

Observation 2: A sequence of integers $a_0, a_1, \cdots, a_k$, $k \ge 0$, form the partial quotients of a redundant continued fraction $[a_0/a_1/\cdots/a_k]$ if and only if

a) only a_0 may have the value zero, and

b) $|a_i| = 1$ for $1 \le i \le k-1$ implies a_i and a_{i+1} have the same sign, and

c) $|a_k| \ge 2$ whenever $k \ge 2$. □

The set of all redundant continued fraction expansions of $\frac{11}{4}$ is then

$$\frac{11}{4} = \begin{cases} [2/1/3] \\ [2/2/\bar{1}/\bar{2}] \\ [2/2/\bar{2}/2] \\ [3/\bar{4}] \end{cases}.$$

Note that a_{i+1} is always given by $\lceil 1/f_i \rceil$ or $\lfloor 1/f_i \rfloor$, so the set of continued fraction expansions available for x is the result of a process allowing the choice of one of two successive integers for the next partial quotient (given the previous quotients) except for the last partial quotient, which is unique and of magnitude ≥ 2 for $k \ge 1$.

We now investigate the design of an algorithm for the evaluation of the expression

$$z(x, y) = \frac{axy + bx + cy + d}{exy + fx + gy + h} \tag{2}$$

under the assumption that x and y are given by redundant continued fraction expansions, and that the value of $z(x, y)$ will be serially computed as such an expansion. The desired algorithm extends to signed partial quotients the procedure introduced by Gosper [2] as formalized in [8] for nonredundant continued fraction input and output. We here review the salient features of the algorithm developed in [8] with principal concern for the modifications necessary regarding signed partial quotients. We then illustrate the mechanisms of the modified algorithm by an example computation.

Any redundant continued fraction $x = [a_0/a_1/a_2/\cdots/a_k]$ has a *leading signed partial quotient* a_0 which is an integer, and a *tail* $x' = [a_1/a_2/\cdots/a_k]$ which is the continued fraction composed of the remaining signed partial quotients, where then

$$x = a_0 + \frac{1}{x'}. \tag{3}$$

The algorithmic step of entering the leading signed partial quotient $p = a_0$ of x into $z(x, y)$ corresponds to a substitution of the variable x by the expression (3) into (2),

$$\begin{aligned} z(x, y) &= z\left(p + \frac{1}{x'}, y\right) \\ &= \frac{(pa+c)x'y + (pb+d)x' + ay + b}{(pe+g)x'y + (pf+h)x' + ey + f} = z_1(x', y). \end{aligned} \tag{4}$$

Similarly entering the leading signed partial quotient q of y corresponds to substituting $y = q + 1/y'$ into (2) obtaining

$$z_2(x, y') = \frac{(qa+b)xy' + ax + (qc+d)y' + c}{(qe+f)xy' + ex + (qg+h)y' + g}. \tag{5}$$

Note that the resulting expressions in (4) and (5) both have the same form as (2), where the eight (integral) coefficients have been updated by simple linear transformations using only the leading signed partial quotient of x and/or y as input. The *coefficient 8-tuple* (a, b, c, d, e, f, g, h) is now observed to provide the basis for our algorithm design, with the dynamics of the algorithm specified in the updating of this 8-tuple. We then term the transformations dictated by (4) and (5) *input transformations* on this coefficient 8-tuple.

To complete the description of serial rational arithmetic, we now describe the process of determining and extracting the leading signed partial quotient of the output value $z(x, y) = [a_0/a_1/a_2/\cdots/a_k]$ also by its transformation of the coefficient 8-tuple. Letting $r = a_0$ denote an allowed leading signed partial quotient, we rewrite z in the form

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

and from (2) solve for the tail $z'(x, y)$ obtaining

$$z'(x, y) = \frac{exy + fx + gy + h}{(a-re)xy + (b-rf)x + (c-rg)y + (d-rh)} \tag{6}$$

which again is seen to have the same form as (2). Expression (6) describes the *output transformation* on the coefficient 8-tuple, and may be recognized as a step of the Euclidean algorithm applied concurrently to the four pairs (a, e), (b, f), (c, g), and (d, h).

In contrast to the expressions $z_1(x', y)$ in (4) and $z_2(x, y')$ in (5) which have maintained the same value as $z(x, y)$ by compensatory changes in arguments and coefficients, the value of the tail $z'(x, y)$ in (6) becomes the inverse of the "fractional part" $z(x, y) - r$. To confirm for output the value r as a leading partial quotient of $z(x, y)$, it is necessary that $|z'(x, y)| > 1$ to satisfy (1). Before analyzing this condition to ascertain a suitable recursive algorithm design for the step of extracting the next signed partial quotient for output, let us introduce a notation to depict the 8-tuple of coefficients of $z(x, y)$.

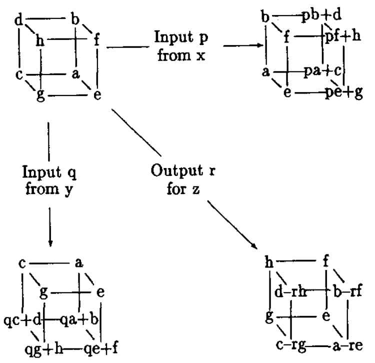


Fig. 1. The coefficient cube and its transformations by partial quotient input and output.

An appropriate apparatus to describe the mechanics of the transformations is the *coefficient cube* illustrated in Fig. 1, where the eight coefficients of $z(x, y)$ are placed in a $2 \times 2 \times 2$ array. The 4-tuple b, a, e, f of the coefficient cube identifies the *leading x-face*, c, a, e, g the *leading y-face*, and h, f, e, g the *leading z-face*. The transformation (4) which inputs the leading partial quotient p of x effects a linear transformation of coefficients in the *x-direction* creating a new leading x-face $pb+d, pa+c, pe+g, pf+h$. Equation (5) similarly yields a new leading y-face and (6) a new leading z-face.

As we shall be describing a recursive process, in the following, x and y will be taken to denote the continued fractions given by the remaining tails of the initial x and y, and z is the yet to be determined tail of the original z having deleted the leading partial quotients of z already extracted as output. Similarly, a, b, c, d, e, f, g, h will denote the coefficients of the updated coefficient cube.

The new selection procedure we now describe will determine a next partial quotient of z, again utilizing only the updated coefficient cube 8-tuple of integer constants as in [8]. To select the next signed partial quotient of $z(x, y)$ it is necessary to determine the range of $z(x, y)$ over appropriate domains $x \in D_x$ and $y \in D_y$. Initially $D_x = D_y = \mathbb{R}$, the set of real numbers, but whenever the first partial quotients of x and y have been read, $|x| > 1$ and $|y| > 1$ since their continued fraction expansions are redundant, hence $D_x \subset (1, -1)$ and $D_y \subset (1, -1)$. Here $(1, -1)$ denotes the open affine interval $\{x | x > 1 \text{ or } x < -1\}$, which includes the single point at infinity, $\infty = +\infty = -\infty$. Since $|a_i| = 1$ implies that a_i and a_{i+1} have the same sign, then if say the last partial quotient input from x was $a_i = 1$, then $D_x = (1, \infty)$. Referring to the definition of the continued fractions, it may be seen that for a terminated expansion the corresponding variable x or y

may just be considered "stuck at infinity," i.e., $D_x = \{\infty\}$ or $D_y = \{\infty\}$.

Note that if the range of the function $z(x, y)$ is within the interval $r - 1 < z(x, y) < r + 1$ over the domain $x \in D_x$ and $y \in D_y$, then certainly the next partial quotient (say a_i) may be chosen to be r. After the output transformation, the "tail" $z'(x, y)$ then satisfies $|z'(x, y)| > 1$ so we shall always obtain the necessary condition of (1) on the remaining fractional part $z(x, y) - r$ corresponding to the yet to be determined tail $z'(x, y) = [a_{i+1}/a_{i+2}/\cdots/a_k]$.

Thus, if there exists an r such that

$$\{z(x, y) | x \in D_x \text{ and } y \in D_y\} \subset (r - 1, r + 1)$$

then a next partial quotient for $z(x, y)$ of r can be output, otherwise more input will have to be taken from the x-direction and/or y-direction to reduce the range of z. Notice that the sets D_x and D_y are "state-dependent," but normally they will be of the form $D_x = (1, -1)$ and $D_y = (1, -1)$. Most such domains are infinite, open domains which have to be interpreted in the affine sense, i.e., through infinity. Since the intervals are open at finite endpoints we must assure that $z(x, y)$ is well defined on the intervals closed at such endpoints, e.g., $[1, -1]$. Then if say $D_x = D_y = (1, -1)$, the range of $z(x, y)$ can be determined from the four values:

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

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

As numerators and denominators are treated separately, it is sufficient for these considerations if either $z(x, y)$ or $1/z(x, y)$ is well defined and monotone. This is the case if *either* the numerator $axy + bx + cy + d$ *or* the denominator $exy + fx + gy + h$ is nonzero over the domain $x \in D_x$ and $y \in D_y$. An analysis of the root curves of the numerator or the denominator shows that it is possible to determine the well definedness of $z(x, y)$ or $1/z(x, y)$ by the signs of their denominators at the endpoints of the intervals D_x and D_y, e.g., at the four points $(-1, -1)$, $(1, 1)$, $(-1, 1)$, and $(1, -1)$. From the values at these points it is possible to construct a function *Zrange*(Q). Given as input the 8-tuple of integers of the coefficient cube Q and the state of the input x and y (i.e., D_x and D_y), *Zrange*(Q) determines the range of values of $z(x, y)$ over the domain $D_x \times D_y$. To determine *Zrange*(Q), first compute the values of the numerator and denominator at the endpoints, and if their signs indicate that $z(x, y)$ or $1/z(x, y)$ is well defined and hence monotone on D_x and D_y, return their minimum and maximum ratio as the interval:

$$Zrange(Q) = \{z(x, y) | x \in D_x \text{ and } y \in D_y\}$$

where the interval is given in the affine sense. The value of *Zrange*(Q) will normally be an open interval; however, if $z(x, y)$ is constant, the interval reduces to a single point. If necessary, the process of computing *Zrange*(Q) will request more input from x or y thus restricting the domains D_x and D_y to assure the well definedness of $z(x, y)$, and perform the appropriate transformation on Q as a side effect.

If either x or y but not both become exhausted, only two ratios determine the range of $z(x, y)$. So if say x is terminated, and $D_y = (1, -1)$, the values of $z(\infty, -1)$ and $z(\infty, 1)$ determine the range of $z(x, y)$. And if both x and y terminate, the range of $z(x, y)$ reduces to the value $z(\infty, \infty) = a/e$. In this case, *Zrange*(Q) reduces to the single point a/e.

The procedures we have just described for computing input and output transformations on the coefficient cube (a, b, c, d, e, f, g, h) utilizing the *Zrange*(Q) function can be summarized as follows.

Algorithm SRA (Serial Rational Arithmetic):

Given input queues of signed partial quotients for the variables x and y and the initial values of the coefficients (a, b, c, d, e, f, g, h) forming the coefficient cube Q, this algorithm serially computes an output queue of signed partial quotients for

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

Input from x or y:

Whenever *Zrange*(Q) is not the point at infinity and there is no r such that *Zrange*(Q) $\subset (r - 1, r + 1)$, input a partial quotient from either the x or y queue, update the coefficient cube Q, and compute the new *Zrange*(Q).

{Corresponding to the variable substitution $x = p + 1/x'$ to be performed when a leading partial quotient p from x is consumed, the input transformation for x may be described as

$$\left\{\left\{\begin{array}{cccc} d & & b & \\ & h & & f \\ c & & a & \\ & g & & e \end{array}\right\}\right\} \left\{\begin{array}{cc} 0 & 1 \\ 1 & p \end{array}\right\} = \left\{\left\{\begin{array}{cccc} b & & pb + d & \\ & f & & pf + h \\ a & & pa + c & \\ & e & & pe + g \end{array}\right\}\right\}$$

where a generalized notion of matrix multiplication has been applied to $2 \times 2 \times 2$ arrays. The transformation corresponding to the substitution $y = q + 1/y'$ may similarly be expressed as a multiplication by the array $\left\{\begin{array}{cc} 0 & 1 \\ 1 & q \end{array}\right\}$ on a properly transposed version of the coefficient cube array (see Fig. 1).}

Output for z:

1) Whenever *Zrange*(Q) is the point at infinity, output the endmarker to the output queue and terminate.

2) Whenever *Zrange*(Q) $\subset (r - 1, r + 1)$ for some r, output one such r to the output queue, update the coefficient cube Q, and compute the new *Zrange*(Q). {The transformation corresponding to the output of r can be effected by multiplication with the matrix $\left\{\begin{array}{cc} 0 & 1 \\ 1 & -r \end{array}\right\}$, again on a suitably transposed version of the coefficient cube Q.} □

The mechanics of Algorithm SRA are illustrated in Fig. 2, where a computation of $\frac{18}{11} + \frac{14}{11}$ with $\frac{18}{11} = [2/\bar{3}/\bar{4}]$ and $\frac{14}{11} = [1/4/\bar{3}]$ is displayed in terms of coefficient cube transformations. Each transformation creates a new face in the x, y, or z direction. The new face along with the preceding face in that direction constitute the coefficient cube values at that point in the serial computation process.

In summary, we conclude that it is possible to construct an

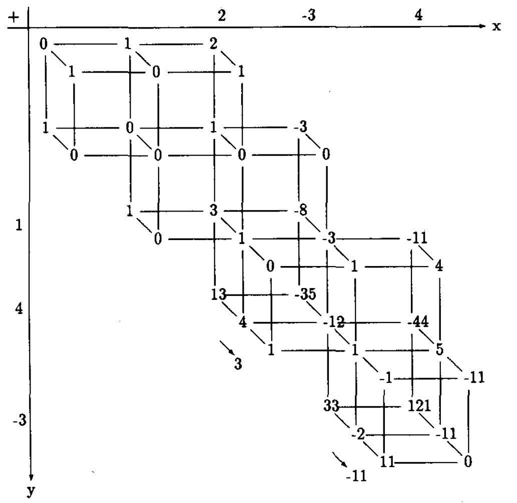


Fig. 2. Coefficient cube transformations for the computation $\frac{32}{11} = \frac{18}{11} + \frac{14}{11}$, yielding for output the redundant continued fraction $[3, \bar{1}\bar{1}] = \frac{32}{11}$.

algorithm, which takes two rational operands x and y, given as redundant continued fractions, and produces as output the value of the function $z(x, y)$ in the same representation. This algorithm allows for performing the standard arithmetic operations $+$, $-$, $\times$, $/$ in an integrated manner by a common arithmetic unit. The algorithm is on-line at the partial quotient level, most significant partial quotient first. As the output of a very large partial quotient potentially requires an unbounded number of (small) partial quotients to be input, we must measure the granularity for on-line delay at a level other than the number of partial quotients. For this purpose a signed bit binary version of this algorithm is now developed.

III. The Binary-Level Algorithm

To provide more uniform throughput and bound the on-line delay of the serial rational arithmetic algorithm of Section II, it is necessary to be able to consume input and produce output in small "granularized units," e.g., to operate with redundancy at a bit level. The LCF representation [7]–[9], [11] and the "continued logarithm" [2] provide useful models of serial binary rational representation without, however, the essential feature of redundancy. We now introduce a signed bit redundant binary representation of the rationals following the construction of the LCF representation [8]. Our representation demonstrates the viability and usefulness of including implicit self-normalization in the signed bit string interpretation.

Let $[p]_2 = b_n b_{n-1} \cdots b_1 b_0$ where $b_i \in \{\bar{1}, 0, 1\}$ denote any redundant binary radix representation of the integer p, allowing arbitrary many leading zeros. Corresponding to $[p]_2$ and using the extended four-letter "signed bit" alphabet $\{u, \bar{1}, 0, 1\}$ we term the even length string $u^{n-1} b_n b_{n-1} \cdots b_1 b_0$ for $n \geq 1$ a *self-delimiting* signed bit string of value p.

An *admissible* redundant binary signed bit string $R(p)$ is then given for any integer $p \neq 0$ by any self-delimiting string

$$R(p) = u^{n-1} b_n b_{n-1} \cdots b_1 b_0 \qquad \text{with } |b_n| = 1, \qquad (7)$$

satisfying the *range constraint*

$$2^{n-1} + 1 \leq |p| \leq 2^{n+1} - 1 \qquad \text{for } n \geq 2,$$

and for $p = 0$ by the single bit $R(0) = b_0 = 0$.

We term u^{n-1} the *unary* part of $R(p) = u^{n-1} b_n b_{n-1} \cdots b_1 b_0$, and $b_n b_{n-1} \cdots b_1 b_0$ the *binary* part of $R(p)$. The unary part serves as a counter to determine n, hence the range within a factor of four for any $p \neq 0$. The condition (7) admits a minimal notion of redundancy in the range determining part u^{n-1} while avoiding the inefficiency attendant to processing the equivalent of an extended string of leading zeros, e.g., $1\bar{1}\bar{1} \cdots \bar{1} = 00 \cdots 01$. The admissibility condition (7) affects restrictions on the leading bits of the binary portion of $R(p)$ that are relevant to the problem of recognizing admissible signed bit strings.

Observation 3: For $n \geq 2$, a self-delimiting signed bit string $u^{n-1} b_n b_{n-1} \cdots b_1 b_0$ is admissible if and only if when $b_n b_{n-1} = 1\bar{1}$ or $\bar{1}1$, the sign of $b_{n-2} b_{n-3} \cdots b_0$ agrees with that of b_n. □

For any redundant continued fraction $x = [a_0/a_1/\cdots/a_k]$ we then obtain by concatenation an *admissible* string for x,

$$R(x) = R(a_0) \circ R(a_1) \circ \cdots \circ R(a_k)$$

whenever $R(a_i)$ is an admissible string for a_i for $0 \leq i \leq k$.

For the redundant continued fraction $\frac{18}{11} = [2/\bar{3}/4]$ we obtain as one admissible string

$$R(\tfrac{18}{11}) = R([2/\bar{3}/4]) = 10u\bar{1}1\bar{1}u100.$$

Note that $10\bar{1}\bar{1}u100$ is a shorter admissible string also determining the redundant continued fraction $[2/\bar{3}/4]$. The string $1\bar{1}1\bar{1}10u\bar{1}00$ determines the sequence of admissible partial quotient values 1, 1, 2, -4, which constitutes an alternative redundant continued fraction of value $\frac{18}{11}$. In choosing an admissible string representation for $\frac{18}{11}$ there is redundancy in both choosing the partial quotients, and in the representation of the partial quotients. The reverse process is, however, unique, reading any admissible string from left to right allows for determination of a unique sequence of partial quotients, and then determination of a unique real value. If a bit string is not known to be admissible, Observations 3 and 2 may be employed during a left-to-right scan to confirm admissibility.

As noted in Algorithm SRA, the following matrix product expresses the transformations for $x = [a_0/a_1/\cdots/a_k]$,

$$\begin{Bmatrix} 0 & 1 \\ 1 & a_0 \end{Bmatrix} \begin{Bmatrix} 0 & 1 \\ 1 & a_1 \end{Bmatrix} \cdots \begin{Bmatrix} 0 & 1 \\ 1 & a_k \end{Bmatrix},$$

corresponding to the input of x into the computation cube in terms of partial quotients. We shall show that this factorization may be further refined into a product of primitive "shift-and-add" transformations.

Observation 4: The matrix $\begin{Bmatrix} 0 & 1 \\ 1 & p \end{Bmatrix}$ has the following factorization in one-to-one correspondence with the elements of the representation $R(p) = u^{n-1}b_nb_{n-1}\cdots b_1b_0$ for $p \neq 0$:

$$\begin{Bmatrix} 0 & 1 \\ 1 & p \end{Bmatrix} = \begin{Bmatrix} 1 & 0 \\ 0 & 2 \end{Bmatrix}^{n-1} \begin{Bmatrix} 0 & 1 \\ 2 & 2b_n \end{Bmatrix} \cdot \begin{Bmatrix} \frac{1}{2} & \frac{b_{n-1}}{2} \\ 0 & 1 \end{Bmatrix} \cdots \begin{Bmatrix} \frac{1}{2} & \frac{b_0}{2} \\ 0 & 1 \end{Bmatrix}. \quad (8)$$

Note that the factorization in (8) involves just seven primitive matrices which may be categorized into three groups, depending on whether the signed bit of $R(p)$ for $p \neq 0$ is 1) in the unary part, 2) in the leading position of the binary part denoting the switch from unary to binary, or 3) in a trailing (nonleading) position of the binary part.

Unary transform:

$$\begin{Bmatrix} 1 & 0 \\ 0 & 2 \end{Bmatrix},$$

Switch bit transforms:

$$\begin{Bmatrix} 0 & 1 \\ 2 & -2 \end{Bmatrix}, \begin{Bmatrix} 0 & 1 \\ 2 & 0 \end{Bmatrix}, \begin{Bmatrix} 0 & 1 \\ 2 & 2 \end{Bmatrix},$$

Trailing bit transforms:

$$\begin{Bmatrix} \frac{1}{2} & -\frac{1}{2} \\ 0 & 1 \end{Bmatrix}, \begin{Bmatrix} \frac{1}{2} & 0 \\ 0 & 1 \end{Bmatrix}, \begin{Bmatrix} \frac{1}{2} & \frac{1}{2} \\ 0 & 1 \end{Bmatrix}. \quad (9)$$

Observation 5: Given an admissible string for a rational $|x| \geq 1$,

$$R(x) = u^{n_1-1}b^1_{n_1}b^1_{n_1-1}\cdots b^1_0 u^{n_2-1}b^2_{n_2}b^2_{n_2-1} \cdots b^2_0 \cdots u^{n_k-1}b^k_{n_k}b^k_{n_k-1}\cdots b^k_0,$$

the matrix product $\prod_{i=1}^{k} \begin{Bmatrix} 0 & 1 \\ 1 & a_i \end{Bmatrix}$ has a factorization into the seven primitive transformation matrices (9) in one-to-one correspondence with the signed bits of $R(x)$,

$$\prod_{i=1}^{k} \begin{Bmatrix} 1 & 0 \\ 0 & 2 \end{Bmatrix}^{n_i-1} \begin{Bmatrix} 0 & 1 \\ 2 & 2b^i_{n_i} \end{Bmatrix} \cdot \begin{Bmatrix} \frac{1}{2} & \frac{b^i_{n_i-1}}{2} \\ 0 & 1 \end{Bmatrix} \cdots \begin{Bmatrix} \frac{1}{2} & \frac{b^i_0}{2} \\ 0 & 1 \end{Bmatrix}. \quad (10)$$

For $|x| < 1$, an admissible string for x is given by prepending a 0 to an admissible string for $1/x$, i.e., $R(x) = 0 \circ R(1/x)$. The factorization for x then has an initial "reciprocating" factor $\begin{Bmatrix} 0 & 1 \\ 1 & 0 \end{Bmatrix}$ corresponding to the leading 0 bit, followed by the factorization corresponding to $R(1/x)$. □

The factorization (10) provides an algorithm for the (signed) bitwise input of information from the bit streams for x and/or y. Each matrix of (10) corresponds to a variable substitution, in general the substitution

$$x = \frac{\gamma + \alpha x'}{\delta + \beta x'} \text{ yielding } x' = \frac{\delta x - \gamma}{\alpha - \beta x}$$

can be performed by multiplying the coefficient cube Q in the x-direction by an appropriate matrix

$$Q' = Q \times_x \begin{Bmatrix} \delta & \beta \\ \gamma & \alpha \end{Bmatrix}.$$

Similarly input from y may be performed bitwise, by multiplication in the y-direction with similar matrices, corresponding to a factorization of the matrix representing the variable transformation $y = q + 1/y'$.

It is here essential to observe that successive leading signed bits from x or y may be read in any order between x and y, and thus the corresponding matrix multiplications in the x- and y-direction may similarly be interleaved. This is due to the fact that the variable transformations of x and y may be interleaved in any way.

It is also important to notice that each matrix multiplication is no more than some simple shift-and-add/subtract operation, which internal to the unit may be performed in parallel on four register pairs in constant time. The matrices just represent a convenient notation for some simple register level operations, which allow us to express their individual and combined effect in a rigorous way.

The output of a partial quotient r of $z(x, y)$ may also be performed serially at the signed bit level, corresponding to a multiplication in the z-direction employing the factorization

$$\begin{Bmatrix} 0 & 1 \\ 1 & -r \end{Bmatrix} = \begin{Bmatrix} 1 & 0 \\ 0 & 2 \end{Bmatrix}^{n-1} \begin{Bmatrix} 0 & 1 \\ 2 & -2b_n \end{Bmatrix} \cdot \begin{Bmatrix} \frac{1}{2} & -\frac{b_{n-1}}{2} \\ 0 & 1 \end{Bmatrix} \cdots \begin{Bmatrix} \frac{1}{2} & -\frac{b_0}{2} \\ 0 & 1 \end{Bmatrix} \quad (11)$$

where $R(r) = u^{n-1}b_nb_{n-1}\cdots b_0$. Notice that it is now possible to emit leading signed bits of the $R(\cdot)$ representation of r, before r is completely determined. For each signed bit emitted, the appropriate transformation on the coefficient cube is performed by multiplication with the corresponding primitive matrix in the z-direction, yielding a new value for z as follows.

A transformation by multiplication in the z-direction by a matrix

$$\begin{Bmatrix} \delta & \beta \\ \gamma & \alpha \end{Bmatrix}$$

corresponds to a rewriting

$$z(x, y) = \frac{-\gamma + \alpha z'(x, y)}{\delta - \beta z'(x, y)} \text{ yielding } z'(x, y) = \frac{\gamma + \delta z(x, y)}{\alpha + \beta z(x, y)}$$

i.e., $z(x, y)$ is substituted by $z'(x, y)$. In particular, we have transformations

$$\begin{Bmatrix} 1 & 0 \\ 0 & 2 \end{Bmatrix} \text{ yields } z'(x, y) = \frac{z(x, y)}{2},$$

$$\begin{Bmatrix} 0 & 1 \\ 2 & -2b \end{Bmatrix} \text{ yields } z'(x, y) = \frac{2}{z(x, y) - 2b},$$

$$\begin{Bmatrix} \frac{1}{2} & -\frac{b}{2} \\ 0 & 1 \end{Bmatrix} \text{ yields } z'(x, y) = \frac{z(x, y)}{2 - bz(x, y)},$$

where the possibly necessary initial reciprocal transformation

$$\left\{\begin{matrix} 0 & 1 \\ 1 & 0 \end{matrix}\right\} \text{ yields } z'(x, y) = \frac{1}{z(x, y)}.$$

After emitting a signed bit of r, it may be possible to determine another signed bit of r, and the cycle is repeated, otherwise more input from x or y will have to be taken. From the transformations of $z(x, y)$ above it follows that it is permissible to interleave output transformations with input transformations in any order between x, y, and z.

We are now ready to formulate an algorithm which will determine the signed bits of $R(r)$, where r is a leading partial quotient of $z(x, y)$, and where repeated application of the algorithm will determine $R(z(x, y))$. The process is similar to the quotient selection process of SRT type division, however, tied to the process of reading operands in such a way that input is only requested when necessary to determine the next signed bit of output. It will utilize the function *Zrange*(Q), which given the coefficient cube Q will return an interval such that $z(x, y) \subset$ *Zrange*(Q) for all permissible values of the tails of x and y. If necessary, the function *Zrange* will as a side effect request more input from x and y, to assure that such an interval can be returned. The following algorithm uses a loop-construct, where, however, the guards must be tested in the order listed. If true, the following statement will be executed and the loop repeated, testing from the top again.

```
Algorithm RPQ
{This algorithm determines a string $u^{n-1}b_nb_{n-1}\dots b_0$ which is
an admissible representation $R(r)$ of the next partial quotient
$r$ of $z(x,y)$, as specified by the given coefficient cube $Q$, and
admissible input strings $R(x)$ and $R(y)$.}
n:=1;
loop
    Zrange(Q) ⊂ (−1,1):   {output 0; n:=0;
                           perform transf. $\left\{\begin{matrix} 0 & 1 \\ 1 & 0 \end{matrix}\right\}$;
                           exit loop};
    Zrange(Q) ⊂ (−4,0):   {output $\bar{1}$;
                           perform transf. $\left\{\begin{matrix} 0 & 1 \\ 2 & 2 \end{matrix}\right\}$;
                           exit loop};
    Zrange(Q) ⊂ (0,4):    {output 1;
                           perform transf. $\left\{\begin{matrix} 0 & 1 \\ 2 & -2 \end{matrix}\right\}$;
                           exit loop};
    Zrange(Q) ⊂ (2,−2):   {output u; n:=n+1;
                           perform transf. $\left\{\begin{matrix} 1 & 0 \\ 0 & 2 \end{matrix}\right\}$};
              true :      {take more input from x or y};
end loop;
Assert{Zrange(Q) ⊂ (1,−1) }
loop
                n=0 :     {exit loop};
    Zrange(Q)⊂ (−∞,−1):   {output $\bar{1}$; n:=n-1;
                           perform transf. $\left\{\begin{matrix} \frac{1}{2} & \frac{1}{2} \\ 0 & 1 \end{matrix}\right\}$};
    Zrange(Q)⊂ (1,∞):     {output 1; n:=n-1;
                           perform transf. $\left\{\begin{matrix} \frac{1}{2} & -\frac{1}{2} \\ 0 & 1 \end{matrix}\right\}$};
    Zrange(Q) ⊂ (2,−2):   {output 0; n:=n-1;
                           perform transf. $\left\{\begin{matrix} \frac{1}{2} & 0 \\ 0 & 1 \end{matrix}\right\}$};
              true :      {take more input from x or y};
end loop;
Assert{Zrange(Q) ⊂ (1,−1) }
```

After completion of the algorithm, Q will have been transformed into a new cube Q' representing $z'(x, y) = 1/(z(x, y) - r)$ in agreement with the factorization (11).

Lemma 1: The output $R(r) = u^{n-1}b_nb_{n-1}\cdots b_0$ of Algorithm RPQ is an admissible string for the integer r. Repeated application of Algorithm RPQ yields $R(a_0) \circ R(a_1) \circ \cdots \circ R(a_k)$, which is an admissible string for $z(x, y)$.

Proof: Consider first a single application of Algorithm RPQ. If the initial loop outputs first a 0 then the algorithm just switches the role of numerators and denominators (reciprocates), having emitted the admissible string for a_0.

If either a $\bar{1}$ or 1 is emitted then the first loop exits with $n = 1$, and the computation will proceed to the second loop where one more signed bit will be output. Any of the resulting strings $\bar{1}\bar{1}$, $\bar{1}0$, $\bar{1}1$, $1\bar{1}$, 10, 11 are then admissible.

If the initial loop emits at least one u, then $n \geq 2$ and either $b_n = 1$ or $b_n = \bar{1}$ upon exit from the first loop. In case $b_n = 1$, then the new *Zrange*(Q) $\subset (1, -2)$. If the first output of loop two is either $b_{n-1} = 0$ or 1, the final result is readily seen to be admissible. If the first output of loop two is $b_{n-1} = \bar{1}$, then the next *Zrange*(Q) $\subset (1, \infty)$. From this condition it follows that $b_{n-2} = 1$ must subsequently be emitted, thence yielding an admissible string for r. The result is symmetric for the case $b_n = \bar{1}$, completing the proof that $R(r)$ is admissible.

Repeated application of Algorithm RPQ yields the sequence of admissible strings $R(a_0)$, $R(a_1)$, $\cdots$. Both x and y input to the cell must terminate (as indicated by the end marker) and the unit will itself terminate in a finite number of steps. Recall that a/e of the coefficient cube determines the output when input has terminated; hence, $e = 0$ indicates that the complete continued fraction has been extracted. At that point, a string of completed partial quotients $R(a_0) \circ R(a_1) \circ \cdots \circ R(a_k)$ has been output. As the assertion *Zrange*(Q) $\subset (1, -1)$ holds after emitting $R(a_i)$ for all i, $[a_0/a_1/\cdots/a_k]$ is a redundant continued fraction for $z(x, y)$ and the output an admissible string for $z(x, y)$. □

A computational unit employing this algorithm can utilize a transition lookup table to assist the computation of *Zrange*(Q). There are several states to consider when computing *Zrange*(Q), corresponding to different D_x and D_y domains for the tails of x and y. Initially $D_x \times D_y = \mathbb{R} \times \mathbb{R}$, and after the endmarker (symbol e) on both strings has been processed finishing all input we shall have $D_x \times D_y = \{\infty\} \times \{\infty\}$. Inputs from x and y are each independently in one out of two major states corresponding to reading the unary/switch, respectively, the balance of the binary part. These major states are each refined into five recognized substates corresponding to intervals D of possible values of the tail of each of x and y. Table I describes the state transitions

TABLE I
Evaluation-State Lookup Table

Sub-state	D_x or D_y domains	Unary/Switch					Binary		
		u	1	$\bar{1}$	0	e	1	$\bar{1}$	0
Start	$\mathbb{R}$	A	A	A	A	-	-	-	-
A	(1,-1)	A	B	D	-	F	A	A	A
B	(1,-2)	-	-	-	-	-	A	C	A
C	(1,∞)	C	B	-	-	-	A	-	C
D	(2,-1)	-	-	-	-	-	E	A	A
E	(−∞,-1)	E	-	D	-	-	-	A	E
F	{∞}	-	-	-	-	-	-	-	-

and the associated intervals pertaining separately to each of D_x and D_y.

A computational unit (cell) employing Algorithm RPQ as described above will thus repeatedly apply the algorithm to its coefficient cube as long as output is requested, implicitly ingesting input when needed and available. Hence, it will run in a "precision demand-driven" mode.

We conclude this section with some observations generalizing the integer representation $R(p)$ and the factorization (8) of Observation 4 to nonintegral finite precision binary numbers, i.e., standard binary radix represented numbers.

Observation 6 (Radix Input): If $[x]_2 = b_n b_{n-1} \cdots b_0 \cdot b_{-1} \cdots b_{-k}$ is a (possibly redundant) binary radix representation of $x = p + f$, where p is integral and $|f| < 1$, then the integer $R(p)$ representation may be generalized into $R'(p+f) = u^{n-1} b_n b_{n-1} \cdots b_0 b_{-1} \cdots b_{-k}$. The unary part here serves as a scale factor, i.e., "exponent part," used to position the radix point. Correspondingly the input of x in (redundant) radix form may be performed by the following sequence of matrix multiplications:

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

The product of these matrices is $\begin{Bmatrix} 0 & 1 \\ 2^{-k} & x \end{Bmatrix}$ which contains nonintegral entries. A compensatory scaling by 2^k does not affect the value of $z(x, y)$ of (2).

Just as it is possible to receive input in either rational or radix form, it is also possible to produce the result in both forms. □

Observation 7 (Radix Output): If in the first (and only) execution of Algorithm RPQ the output is not terminated when $n = 0$, but is allowed to continue until $n = k$ for some $k < 0$, the algorithm will continue to emit signed bits of the $R'(\cdot)$ representation of $z(x, y)$ as defined in Observation 6. It is necessary that output be terminated (and possibly with a rounding) since the $R'(\cdot)$ representation of rationals in general is infinite (cyclic). □

Summarizing the results of this section we have the following.

Observation 8 (Rational and/or Radix On-Line Computations): Each computation cell can compute an expression

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

where the input arcs to the cell can be initialized independently to receive and accept x and y in either rational $R(\cdot)$ or radix $R'(\cdot)$ representation. Similarly the result can be produced in either one of the two representations.

Several cells can be combined into an expression tree where operations in the cells may proceed in parallel, in a precision demand-driven on-line computation. Operands and results can be represented in the $R(\cdot)$ and/or $R'(\cdot)$ redundant form in any combination. □

IV. On-Line Delay and Bit-Pipelining

An on-line arithmetic unit is intended to operate by inputting digits of the arguments and outputting digits of the result with little delay and considerable regularity. One characterization of the on-line property [1], [13] is that to generate the kth digit of the result, it is necessary and sufficient to input up to $k + \delta$ digits of the operands where δ is some small positive integer. Typical values for δ are of the order 1 to 4, depending on the arithmetic operation (+, −, ×, /) and base of the redundant representation employed. Units based on this property will impose an initial delay of δ digits and then produce output on a regular basis, with one output digit per input from each operand.

An on-line unit modeled on our Algorithm RPQ will always favor output. A bit pipelined sequence of operations will then have units downstream initiated earlier in the computation process. Some loss of regularity occurs in this model. The availability of output given additional input can vary, even though the average number of output bits per input bit is close to unity. Regularity can be quite simply measured by investigating the distribution of local delay, that is, the frequency of the size 0, 1, 2, ⋯ of the number of additional bits of input between successive output bits.

Delays due specifically to the nature of our rational representation can be analyzed by considering the conversion of a binary radix string $R'(x)$ to an admissible string $R(x)$. Employing Observation 8, our unit is implicitly a binary radix-to-rational converter when adding $R'(0) = 0e$ to $R'(x)$ with rational string output $R(x)$. A simulation of Algorithm RPQ for the conversion of $55005/65536 = 0.1101011011011101$, where signed bits are from the alphabet $\{u, 0, 1, m, e\}$ with m denoting $\bar{1}$ and e denoting termination, yielded:

```
a>0e
b>..0.1..1.0.1.0.1.10...1..101..1..10.1...e
c>   0 1m u 1 1 m u  u10 mm   m0 uu  u 1m1 1m1mu11m1m
```

Note that the gaps between the 21 output bits of $R(x)$ before input termination showed one 3-bit delay, two 2-bit delays, nine 1-bit delays, and eight 0-bit delays (no further input before output). The percentage distribution of such radix-to-rational on-line delays for 1000 conversions of 16-bit numbers (selected randomly over $[\frac{1}{2}, 1 - 2^{-16}]$) is shown in Table II.

For comparison we have also included delay distributions for conversion to two nonredundant binary rational representa-

TABLE II
ON-LINE DELAY DISTRIBUTION (IN %) FOR BINARY RADIX-TO-RATIONAL CONVERSION

	Redundant	Non Redundant	
Delay	Admissible	LCF [7]	Cont.Log[2]
0	34.1	41.8	41.8
1	49.8	36.8	37.1
2	13.2	11.6	11.3
3	2.5	5.3	5.4
4	0.4	2.5	2.4
5	0	1.1	1.1
≥ 6	0	0.9	0.9

tions. Note that the LCF [7] and continued log [2] binary representations both show k-bit delays occurring with frequency near $1/2^{k+1}$ for $k \geq 2$. This distribution is characteristic of nonredundant binary delays as illustrated by the equivalent of the wait to determine (parametrically in k) which of the two k-bit strings, $011\cdots1$ or $100\cdots0$ is next to be output. Our redundant admissible strings show quite reasonable regularity, with no delays over 4 bits, and delays over 2 bits in less than 3% of all output. This performance approaches that of redundant binary radix representation, where delays over 2 bits can be shown never to occur.

The delay will become somewhat more irregular when rational representation $R(\cdot)$ is used both for input and output with varying arithmetic operations, particularly for units downline in a tree-pipelined computation. This is best shown by some examples, the first being a simple multiplication $\frac{328}{145} * \frac{27}{101} = \frac{8856}{14645}$. Here the operands employ the following representations

$$R\left(\frac{328}{145}\right) = 11\bar{1}1\bar{1}\bar{1}u11\bar{1}11\bar{1}1\bar{1}0$$

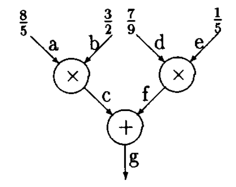
$$R\left(\frac{27}{101}\right) = 0111\bar{1}11u\bar{1}\bar{1}\bar{1}.$$

A simulation of Algorithm RPQ then gave the following output:

```
a>.1..1..m...1...m.m..u...1.1....m.1...1.m.1m.0.e
b>0.1..1...1...m..1.1...u..m.m....m...e
c>   1  m u m 0 1    1 0 u    u10   00  u u  m 0 m1uummm111e
  123456789 123456789 123456789 123456789 123456789 123456789
```

Each line here represents the flow along the arcs into or from the cell, where the horizontal position represents the time step at which the signed bit is on the arc (i.e., consumed). E.g., at step 4 the first bit of the result is produced, based on the 1-bit consumed from operand b. The periods indicate that a signed bit is available, but has not been consumed. As may be noticed, if both operands are available the individual cells are servicing their input arcs in a round-robin manner. A simple strategy for the optimal selection of input is an interesting open problem.

As the second example, we consider a tree-pipelined computation of the expression $\frac{8}{5} \times \frac{3}{2} + \frac{7}{9} \times \frac{1}{5} = \frac{23}{9}$, as pictured in the tree below:



The redundant signed bit representations of the operands used are

$$R\left(\frac{8}{5}\right) = 1\bar{1}1\bar{1}1\bar{1}10 \quad R\left(\frac{3}{2}\right) = 1\bar{1}10$$

$$R\left(\frac{7}{9}\right) = 01\bar{1}1110 \quad R\left(\frac{1}{5}\right) = 0u11\bar{1}.$$

The computation in the cells performing multiplications may proceed in parallel, producing the input operands for the cell doing the addition. In each cell, the eight registers of the coefficient cube are initialized to realize the appropriate operation for that cell. The result of our simulation illustrates the flow of information in and out of the cells.

```
a>.1.m..1.m..1....m........10e
b>1.m..1.0..e
c>    1    1  ...m .1..m..0   1.0.e
d>.0....1.m...1.....1....1...0e
e>0..u...1.1...m.....e
f>  0 .u    u1  ...0  m.1 .m0  u.1.......00..e
g>:: :  :::   u10 m  m   0   :     um00uu  uu e
  123456789 123456789 123456789 123456789 123456789
```

Note that at step 37 the last signed bit of the result has actually been produced, but since not all information produced on arc f has been consumed the add-node continues to emit u's until all information on arc f has been input. This is due to the algorithm preferring to do output when possible, rather than reading more input. The colons indicate that the cell is idle, waiting for input and not being able to output. Note that the final cell was idle only for 7 of the 45 cycles, and only once after initiating its own output.

As a final example, we will repeat the first example $\frac{328}{145} * \frac{27}{101} = \frac{8856}{14645}$; however, this time producing the result in radix representation:

```
a>.1..1..m...1..m.m..u..1...1..m..1...1..m1.m.0e
b>0.1..1...1..m..1..1..u..m..m...m..e
c>   1  m 0 m  m   0  m  1 0  m 0  m 0 10  m 1  1mm1m111111mm
  123456789 123456789 123456789 123456789 123456789 123456789
```

V. CONCLUSIONS

In the preceding sections, it has been demonstrated that it is possible to implement an arithmetic unit, in the form of a cell which will take two operands x, y signed bit serial, and product the result $z(x, y)$ signed bit serial in an on-line fashion. The representation of rational or radix operands and result is redundant over a four-letter signed bit alphabet, and our algorithm allows for a smooth flow of information through a network of such units.

Regarding hardware realizations, the procedures for updating register contents for input/output in the RPQ Algorithm is fairly straightforward to implement, utilizing parallel operations on the appropriate four pairs of registers when performing the simple shift-and-add type operations. The integer contents of each register can be kept in redundant form allowing for true parallel addition (i.e., no carry propagation) with resulting constant input/output processing time. The problem areas lie in the design of the Zrange function. An implementation of the Zrange function might use a PLA lookup, based on leading bits of the contents of the eight registers. However, a straightforward lookup would require much too large a PLA, so a "factoring" is necessary. One way of factoring would be two parallel PLA's dealing separately with numerator and denominator followed by one determining the range. A number of other architectural issues have been raised in [8] which also need further investigation.

Regarding applicability, further testing is now needed to demonstrate that larger networks of cells can run in a highly parallel mode, to provide final results efficiently. Tests of rational versus radix based computations in the cell are needed to measure the overall efficiency implications of finite termination of all rational subcomputations. Further theoretical work to minimize local and compounded delay rates is also needed here.

Acknowledgment

The authors express their gratitude toward S. Johansen for developing the simulation tool, performing numerous experiments with the algorithm, and for working out details of the Zrange computation as illustrated in the Evaluation-State Lookup Table. This tool implements the RPQ Algorithm with flexibility for experimenting with a variety of decision rules, such as those in the extension of the unit to allow radix or rational representations. The simulator allows for extensive composite computations, including computations of standard functions and the use of feedback in computations, and thus provides a vehicle for the next step of measuring applicability and aiding development of a hardware realization of this type of arithmetic unit.

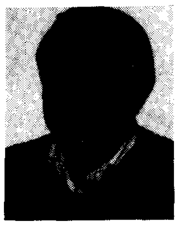
References

[1] M. D. Ercegovac, "On-line arithmetic: An overview," *SPIE Vol. 495, Real Time Signal Processing VII*, pp. 86-93, 1984.
[2] R. W. Gosper, "Item 101 in Hakmem," AIM239, MIT, Feb. 1972, pp. 37-44, further developed in an unpublished manuscript.
[3] C. H. Hardy and E. M. Wright, *An Introduction to the Theory of Numbers*, 5th ed. London, England: Oxford University Press, 1979.
[4] A. Y. Khinchin, *Continued Fractions*, 1935, Translated from Russian by P. Wynn and P. Noordhoff Ltd., Grooningen, 1963.
[5] D. E. Knuth, *The Art of Computer Programming, Vol. 2, Seminumerical Algorithms*, 2nd ed. Reading, MA: Addison-Wesley, 1981.
[6] P. Kornerup and D. W. Matula, "Finite precision rational arithmetic: An arithmetic unit," *IEEE Trans. Comput.*, vol. C-32, no. 4, pp. 378-387, Apr. 1983.
[7] —, "Finite precision lexicographic continued fraction number systems," in *Proc. 7th IEEE Symp. Comput. Arithmetic*, 1985, pp. 207-214.
[8] —, "An on-line arithmetic unit for bit-pipelined rational arithmetic," *J. Parallel Distributed Comput.*, vol. 5, pp. 310-330, 1988.
[9] —, "LCF: A lexicographic binary representation of the rationals," submitted for publication.
[10] D. W. Matula and P. Kornerup, "Foundations of finite precision rational arithmetic," *Computing*, Suppl., vol. 2, pp. 88-111, 1980.
[11] —, "An order preserving finite binary encoding of the rationals," in *Proc. 6th IEEE Symp. Comput. Arithmetic*, 1983, pp. 201-209.
[12] R. B. Seidensticker, "Continued fractions for high-speed and high-accuracy computer arithmetic," in *Proc. 6th IEEE Symp. Comput. Arithmetic*, 1983.
[13] K. S. Trivedi and M. D. Ercegovac, "On-line algorithms for division and multiplication," *IEEE Trans. Comput.*, vol. C-26, no. 7, pp. 681-687, July 1977.

Peter Kornerup was born in Aarhus, Denmark, on April 29, 1939. He received the mag.scient. degree in mathematics (numerical analysis) from Aarhus University in 1967.

After a period employed as a consultant for the university computing center, from 1969 he was involved in establishing the computer science curriculum at Aarhus University, and the founding of the Computer Science Department in 1971. Through most of the 1970's and 1980's he served as Chairman of the Department. In 1975-1976 he spent a leave at the University of Southwestern Louisiana, Lafayette, and the Spring of 1979 another leave with Southern Methodist University, Dallas, TX. Since 1988 he has been Professor of Computer Science, Odense University, Odense, Denmark, in charge of building up a new graduate program. His research interests include computer architecture, in particular computer arithmetic, and number representations.

Mr. Kornerup has served on the program committies for a number of IEEE and ACM sponsored meetings and is Coprogram Chairman for the upcoming 10th IEEE Symposium on Computer Arithmetic.



David W. Matula was born in St. Louis, MO on November 6, 1937. He received the B.S. degree in engineering physics from Washington University, St. Louis, MO, in 1959 and the Ph.D. degree in engineering from the University of California, Berkeley, in 1966.

He has been Professor of Computer Science and Engineering at Southern Methodist University, Dallas, TX, since 1974, where he served as Department Head from 1974-1979 and again in 1988-1989. He has held visiting positions at IBM Research Center, Yorktown Heights, NY, Stanford University, Aarhus University, Frankfurt University, and at Odense University. He currently serves as a consultant to Cyrix Corporation on arithmetic unit design. His research interests include computer arithmetic, algorithm design, graph algorithms, random structures, and cluster analysis.

Dr. Matula is on the editorial boards of the *Journal of Classification* and the journal *Random Structures and Algorithms*. He has served on the program committees for a number of IEEE sponsored meetings and is Coprogram Chairman for the upcoming 10th IEEE Symposium on Computer Arithmetic.