Exploiting Redundancy in Bit-Pipelined Rational Arithmetic *

Peter Kornerup
Odense Universitet
DK 5230 Odense, Denmark

David W. Matula
Southern Methodist University
Dallas, TX 75275-0122

Abstract

We develop and analyse the use of a redundant continued fraction representation of the rationals in the implementation of an arithmetic unit for computing the sum, difference, product, quotient and other useful functions of two rational operands. Our representation of operands and results allows the computations of the unit to be performed in a signed bit serial, on-line fashion. Several such units can then be interconnected for the computation of more complicated expressions in a pipelined manner. Redundancy is used towards the goal of achieving a small bounded on-line delay and uniform throughput.

1 Introduction

This paper exploits the possibilities of using redundancy in the representation of operands, to be used as digit-serial input and produced as output of an on-line arithmetic unit for rational arithmetic. In [7] and [8] such an arithmetic unit was described, but based on operands in non-redundant representations [11] and [9]. To reduce, and possibly bound the delay between input and output of such a unit, it is essential to use redundancy.

The novel features of these types of architectures is that they operate on bit or digit stream representations of the rational numbers, which are derived from the continued fraction expansions of the operand values. The basic idea was suggested in a 1972 memo from MIT's AI lab by Gosper [2], [5] p 360, [12] as an algorithm operating on the partial quotients of a continued fraction expansion.

The arithmetic unit supports the standard operations of addition, subtraction, multiplication, division and many other usefull 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, h are arbitrary prespecified integers. The unit thus operates as a bit- or digit serial, precision demand driven cell, several of which can be interconnected to compute more complicated arithmetic expressions, in general in a tree structured pipeline computation.

A considerable literature exists on on-line arithmetic (see e.g. [14], [1]) where operands are in redundant radix representations, e.g. floating point where the exponent is treated separately. Such arithmetic units achieve constant input-output delay through redundant representation of operands.

It is the purpose of this paper to introduce redundancy in the representation of rationals discussed in [11] and [9], and demonstrate that an architecture for an arithmetic unit can be constructed. For this purpose, in Section II we will describe the basic operation of the unit as operating on the continued fraction expansions, but using signed partial quotients to introduce redundancy at this level. Such redundancy is needed (as in signed digit radix representations), to allow the output of a partial quotient which can then later effectively be corrected by plus or minus a unit.

Section III then carries redundancy into the representation of the individual partial quotients, in essence using a signed digit representation of these. A binary level algorithm is presented for the implementation of the cell wanted.

Section IV concludes this paper with some considerations concerning future work to be done.

[0]This work has been supported by the Danish Natural Science Research Council, grant no.11-7319, and by a grant from the APT Corporation, Dallas.

2 Arithmetic on Redundant Continued Fractions

As in [8] we will investigate the design of an algorithm for evaluating the general expression

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

with integer coefficients a, b, c, d, e, f, g, h, where the variables x, y and the result z are to be represented by the sequence of partial quotients of their continued fraction expansions. Note that by appropriate choice of coefficients in (1), the expression can be employed to compute the standard arithmetic operations $x + y$, $x - y$, $x \cdot y$, x/y, as well as other expressions in x and y.

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

$$x = a_0 + \cfrac{1}{a_1 + \cfrac{1}{a_2 + \cfrac{1}{\ddots + \cfrac{1}{a_n}}}} \qquad (2)$$

with $|[a_i/a_{i+1}/\cdots/a_n]| > 1$ for $i \geq 1$.

The continued fraction expansions of x in (2) may be variously referred to as **signed quotient** continued fractions to emphasize the fact the a_i may be negative as well as positive and/or **redundant** continued fractions to emphasize that the choice of a_i is not unique for non integral x.

The restriction $|[a_i/a_{i+1}/\ldots/a_n]| > 1$ for $i \geq 1$ in (2) allows that we may write $x = [a_1/a_2/\ldots/a_{i-2}/(a_{i-1} + f)]$ with the fractional part $|f| < 1$. This assures that any truncated value $x' = [a_1/a_2/\ldots/a_{i-1}]$ is a good approximation to x in the manner of rounding up or down in the last place (as with a radix representation). It further follows from this restriction that

1. only a_0 may have the value zero,
2. $|a_i| = 1$ implies a_i and a_{i+1} have the same sign, and
3. there is only a finite number of such redundant continued fraction representations for any rational x.

The set of all continued fraction expansions of $\frac{11}{4}$ allowed by (2) is

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

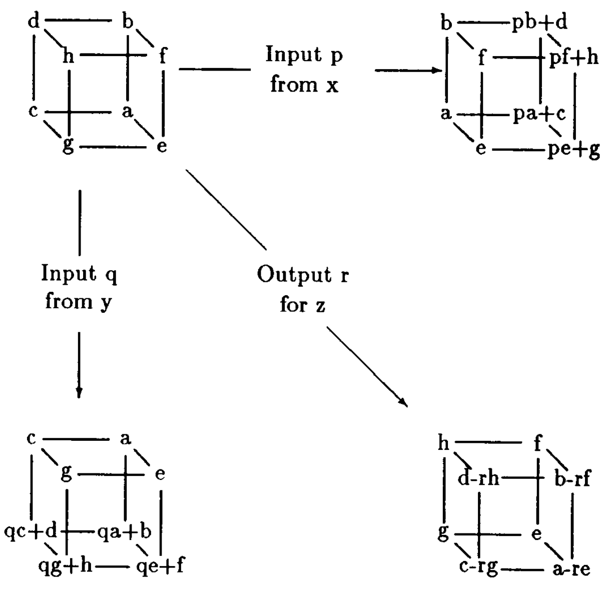


Figure 1: The coefficient cube and its transformations by partial quotient input and output.

Note that the set of continued fraction expansions available for x may be interpreted as 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 then unique and of magnitude ≥ 2 for $n \geq 1$.

In [8] we described the partial quotient algorithm of Gosper [2] by the apparatus of the transformations on the **coefficient cube** illustrated in Figure 1. We here simply note that the input transformations for x and/or y employing signed partial quotients proceed identically to the description in [8]. Our need here is to formulate a new selection criteria allowing earlier determination of a partial quotient to output, exploiting the flexibility of the redundant continued fraction representation.

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]. Since the expansions of x and y are assumed redundant, we assume that $|x| > 1$ and $|y| > 1$, except possibly initially (a_0 may take any integer value, and is the only partial quotient when x and/or y is integral). We avoid any initialization problem by delaying any attempt to output until the first nonzero partial quotient of each argument has been input or the argument has terminated. 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 $|x| > 1$ and $|y| > 1$, then certainly the next partial quotient (say a_k) is 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 (2) on the yet to be determined tail $z'(x,y) = [a_{k+1}/a_{k+2}/\dots/a_n]$.

Assume for the moment that $z(x,y)$ is well defined, i.e. the denominator is nonzero. 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 extreme values of $z(x,y)$ at the four limits of x and y over their ranges:

$$\begin{array}{ll} z(-1,-1) = & z(1,-1) = \\ \quad \dfrac{a-b-c+d}{e-f-g+h} & \quad \dfrac{-a+b-c+d}{-e+f-g+h} \\[2ex] z(-1,1) = & z(1,1) = \\ \quad \dfrac{-a-b+c+d}{-e-f+g+h} & \quad \dfrac{a+b+c+d}{e+f+g+h} \end{array} \tag{3}$$

If there exists an r such that

$$r-1 < z(1,1), z(1,-1), z(-1,1), z(-1,-1) < r+1.$$

then r is the next partial quotient of $z(x,y)$, and the output transformation can be performed updating $z(x,y)$ to $z'(x,y)$, and updating the coefficient 8-tuple as indicated in Figure 1.

A number of comments are due here. We want to determine the range of $z(x,y)$ over the domain $|x| > 1$ and $|y| > 1$. These domains are both infinite, open intervals, which have to be interpreted in the affine sense, i.e. through infinity. The "value" infinity here can be taken as a value that signals the termination of the continued fraction, i.e. as an "end-marker" of x or y (c.f. the definition of the value of a continued fraction). Since the intervals are open, to obtain bounds on $z(x,y)$ by evaluating the four values in (3), we must assume that $z(x,y)$ is well defined and monotone on the **closed** intervals $|x| \geq 1$ and $|y| \geq 1$.

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 non-zero over the domain $|x| \geq 1$ and $|y| \geq 1$. 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 four points $(-1,-1),(1,1),(-1,1)$ and $(1,-1)$. Since the values at these points are needed anyway, it is possible to construct a function Zrange(Q). Given as input the 8-tuple of integers of the coefficient cube Q, it will determine the range of values of $z(x,y)$ over the domain $|x| > 1$ and $|y| > 1$. It will compute the values of the numerator and denominator at the four points, and if their signs indicate that $z(x,y)$ or $1/z(x,y)$ is well defined and hence monotone on $|x| \geq 1$ or $|y| \geq 1$, it will return their minimum and maximum ratio as the interval:

$$\text{Zrange(Q)} = \{z(x,y) \mid |x| > 1, |y| > 1\}$$

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 function will request more input from x or 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 $z(x,y)$. A terminated continued fraction may just be considered "stuck at infinity", so if say x is terminated, the values of $z(\infty,-1)$ and $z(\infty,1)$ determines 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.

We can now sketch the algorithm for computing the value of

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

where x and y are given as redundant continued fraction expansions, and the resulting value $z(x,y)$ is produced in the same representation. For this it is convenient to recall a matrix notation from [8].

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:

$$\left\{\left\{\begin{matrix} d & & b & \\ & h & & f \\ c & & a & \\ & g & & e \end{matrix}\right\}\right\}\left\{\begin{matrix} 0 & 1 \\ 1 & p \end{matrix}\right\} = \left\{\left\{\begin{matrix} b & & pb+d & \\ & f & & pf+h \\ a & & pa+c & \\ & e & & pe+g \end{matrix}\right\}\right\}. \tag{4}$$

The transformation corresponding to the substitution $y = q + 1/y'$ may similarly be expressed as a multiplication by the matrix

$$\left\{\begin{matrix} 0 & 1 \\ 1 & q \end{matrix}\right\}$$

on a properly transposed version of the coefficient cube array (see Figure 1).

Whenever the quotient selection algorithm assures that a leading partial quotient r of $z(x, y)$ can be determined, i.e. Zrange(Q) $\subseteq (r-1, r+1)$, then the transformation can be effected by multiplication with the matrix

$$\left\{\begin{matrix} 0 & 1 \\ 1 & -r \end{matrix}\right\},$$

again on a suitably transposed version of the coefficient cube Q.

As in [8] it is possible to perform equivalent matrix transformations on a $2 \times 2 \times 2$ array, containing the numerators and denominators of the four values $z(-1,-1), z(-1,1), z(1,-1)$ and $z(1,1)$ (the **decision cube**), so that these are directly and recursively computed. We will however not pursue this optimization here, referring to [8] for further details.

For a numeric example of the algorithm we refer the reader to the example in Figure 2, where the computation of $\frac{18}{11} + \frac{14}{11}$ with $\frac{18}{11} = [2/\bar{3}/4]$ and $\frac{14}{11} = [1/4/\bar{3}]$ is displayed in terms of cube transformations.

We may now conclude this section with the observation that it is possible to construct an algorithm, which takes as input x and y, given as a redundant continued fraction, and produce the value of the function $z(x, y)$ in the same representation. The algorithm is on-line at the partial quotient level, most significant partial quotient first. As the output of a very large partial quotient may require 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, so we now develope a signed bit binary version of this algorithm.

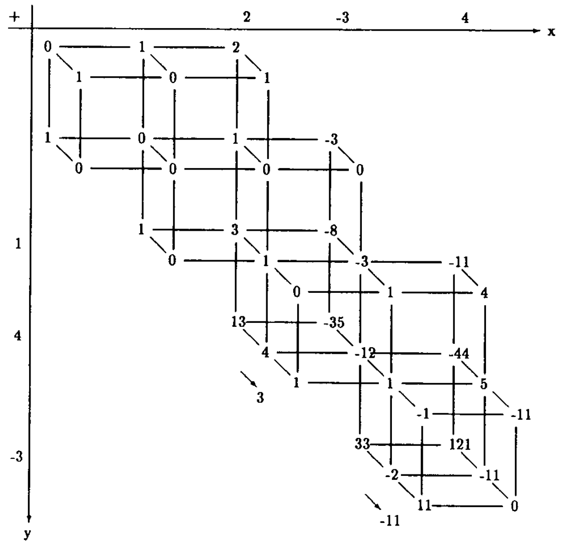


Figure 2: The cube transformations for the computation $\frac{32}{11} = \frac{18}{11} + \frac{14}{11}$, with $\frac{32}{11} = [3, \bar{1}\bar{1}]$

3 The Binary-Level Algorithm

To reduce the on-line delay, it is necessary to be able to consume input and produce output in small "bounded units", e.g. to operate at a bit level. The LCF representation of rationals [7], [9] provides such a facility albeit by a non-redundant representation. To allow the possibility of bounding the on-line delay we now introduce a signed bit redundant representation of continued fractions modeled after the LCF representation [8].

A signed bit representation

$$[p]_2 = b_n b_{n-1} \dots b_1 b_0,$$

$b_i \in \{\bar{1}, 0, 1\}$ is termed **normalized** whenever $|b_n| = 1$, $b_{n-1} \neq -b_n$. Thus we obtain the range $2^{n-1} + 1 \leq p \leq 2^{n+1} - 1$ for any normalized $(n+1)$-signed bit representation. Extending to a 4-letter alphabet $\{u, \bar{1}, 0, 1\}$ we term the even length strings

$$R(p) = u^{n-1} b_n b_{n-1} \dots b_1 b_0 \ for \ n \geq 1$$

admissible self delimiting signed bit representations of value p, whenever $b_n b_{n-1} \dots b_1 b_0$ is a normalized signed bit representation of p, for $n \geq 2$, whereas for $n = 1$ the strings $b_1 b_0$ of length two need not be normalized signed bit strings, affording $R(0) = 00$ as a convenient device to represent the possible initial zero of a continued fraction representation.

For the redundant continued fraction $x = [a_0/a_1/\dots/a_n]$ we then obtain by concatenation the **admissible** string

$$R(x) = R(a_0) \circ R(a_1) \circ \dots \circ R(a_n).$$

The representation $R(p) = u^{n-1}b_n b_{n-1}\dots b_1 b_0$ will allow us to perform all input transformations $\left\{\begin{matrix} 0 & 1 \\ 1 & p \end{matrix}\right\}$ in terms of just the following seven primitive "shift-and-add" transformations itemized in the following tree classes:

Unary transform:

$$\left\{\begin{matrix} 0 & 1 \\ 1 & 2 \end{matrix}\right\}$$

Switch bit transforms:

$$\left\{\begin{matrix} 0 & 1 \\ 2 & -2 \end{matrix}\right\}, \left\{\begin{matrix} 0 & 1 \\ 2 & 0 \end{matrix}\right\}, \left\{\begin{matrix} 0 & 1 \\ 2 & 2 \end{matrix}\right\}$$

Trailing bit transforms:

$$\left\{\begin{matrix} \frac{1}{2} & -\frac{1}{2} \\ 0 & 1 \end{matrix}\right\}, \left\{\begin{matrix} \frac{1}{2} & 0 \\ 0 & 1 \end{matrix}\right\}, \left\{\begin{matrix} \frac{1}{2} & \frac{1}{2} \\ 0 & 1 \end{matrix}\right\}. \quad (5)$$

Observation 1: The matrix $\left\{\begin{matrix} 0 & 1 \\ 1 & p \end{matrix}\right\}$ has the following factorization into the seven primitive transformations of (5) in one-to-one correspondence with the elements of the representation $R(p) = u^{n-1}b_n b_{n-1}\dots b_1 b_0$

$$\left\{\begin{matrix} 0 & 1 \\ 1 & p \end{matrix}\right\} = \left\{\begin{matrix} 1 & 0 \\ 0 & 2 \end{matrix}\right\}^{n-1} \left\{\begin{matrix} 0 & 1 \\ 2 & 2b_n \end{matrix}\right\} \times \left\{\begin{matrix} \frac{1}{2} & \frac{b_{n-1}}{2} \\ 0 & 1 \end{matrix}\right\} \dots \left\{\begin{matrix} \frac{1}{2} & \frac{b_0}{2} \\ 0 & 1 \end{matrix}\right\}. \quad (6)$$

The factorization (6) provides an algorithm for the digitwise input of information from a partial quotient of x or y. Each matrix of (6) corresponds to a variable substitution, in general the substitution

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

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

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

The three types of matrices of (5) hence corresponds to the following variable transformations.

$$\left\{\begin{matrix} 1 & 0 \\ 0 & 2 \end{matrix}\right\} \sim x' = \frac{1}{2}x$$

$$\left\{\begin{matrix} 0 & 1 \\ 2 & 2b \end{matrix}\right\} \sim x' = \frac{1}{\frac{x}{2} - b}$$

$$\left\{\begin{matrix} \frac{1}{2} & \frac{b}{2} \\ 0 & 1 \end{matrix}\right\} \sim x' = \frac{x}{2 - bx}$$

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

It is here essential to observe that the leading digits of x and y may be read in any order, and the corresponding matrix multiplications in the x- and y-direction may be interleaved in any combination. 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 operations, which even may be performed in parallel on four register pairs. The matrices just represent a convenient notation for some simple register level operations, which allows 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 at the digit level, corresponding to the multiplication in the z-direction by the appropriate matrix, which can also be factored as:

$$\left\{\begin{matrix} 0 & 1 \\ 1 & -r \end{matrix}\right\} = \left\{\begin{matrix} 1 & 0 \\ 0 & 2 \end{matrix}\right\}^{n-1} \left\{\begin{matrix} 0 & 1 \\ 1 & -b_n \end{matrix}\right\} \times \left\{\begin{matrix} \frac{1}{2} & -\frac{b_{n-1}}{2} \\ 0 & 1 \end{matrix}\right\} \dots \left\{\begin{matrix} \frac{1}{2} & \frac{b_0}{2} \\ 0 & 1 \end{matrix}\right\} \quad (7)$$

where $R(r) = u^{n-1}b_n b_{n-1}\dots b_0$. Notice that it is now possible to emit leading digits of the $R(\cdot)$ representation of r, before r is completely determined. For each digit emitted, the appropriate transformation on the coefficient cube is performed, by multiplication with the corresponding matrix in the z-direction. Then it may be possible to determine another digit of r, and the cycle is repeated. If it is not possible to determine the next digit, more input from x or y will have to be taken before another attempt to output is made.

A transformation by multiplication in the z-direction by a matrix $\left\{\begin{matrix} \delta & \beta \\ \gamma & \alpha \end{matrix}\right\}$ corresponds to a rewriting:

$$z(x,y) = \frac{-\gamma + \alpha z'(x,y)}{\delta - \beta z'(x,y)} \qquad (8)$$

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

$$\begin{aligned}
\left\{\begin{matrix} 1 & 0 \\ 0 & 2 \end{matrix}\right\} \sim z'(x,y) &= \frac{1}{2} z(x,y) \\
\left\{\begin{matrix} 0 & 1 \\ 2 & -2b \end{matrix}\right\} \sim z'(x,y) &= \frac{1}{\frac{z(x,y)}{2} - b} \\
\left\{\begin{matrix} \frac{1}{2} & \frac{b}{2} \\ 0 & 1 \end{matrix}\right\} \sim z'(x,y) &= \frac{z(x,y)}{2 - bz(x,y)}
\end{aligned}$$

Whereas the input transformations do not change the value of $z(x,y)$, output transformations do so. But it may now also be seen that it is permissible to interleave output transformations with input transformations in any order, as assumed above.

We are now ready to formulate an algorithm which will determine the digits of $R(r)$, where r is the leading partial quotient of $z(x,y)$. It will utilize the function Zrange(Q), which given the coefficient cube Q will return an interval such that $z(x,y) \subseteq$ 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 algorithm will use a loop-construct, where the guards will 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** {determines $R(r)$ from cube $Q$ }
n:=1;
**loop**
  Zrange(Q) $\subseteq (-\frac{3}{2}, \frac{3}{2})$: {output b = 0;
                        perform transf. $\left\{\begin{matrix} 0 & 1 \\ 2 & 0 \end{matrix}\right\}$
                        exit loop };
  Zrange(Q) $\subseteq (-4, -1)$: {output $\bar{1}$;
                        perform transf. $\left\{\begin{matrix} 0 & 1 \\ 2 & 2 \end{matrix}\right\}$
                        exit loop};
  Zrange(Q) $\subseteq (1, 4)$: {output 1;
                        perform transf. $\left\{\begin{matrix} 0 & 1 \\ 2 & -2 \end{matrix}\right\}$
                        exit loop};
  Zrange(Q) $\subseteq (3, -3)$: {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) $\subseteq (1, -1)$ }
**loop**
                 n=0 : {exit loop};
  Zrange(Q) $\subseteq (3, -3)$: {output 0; n:=n-1;
                        perform transf. $\left\{\begin{matrix} \frac{1}{2} & 0 \\ 0 & 1 \end{matrix}\right\}$};
  Zrange(Q) $\subseteq (-4, -1)$: {output $\bar{1}$; n:=n-1;
                        perform transf. $\left\{\begin{matrix} \frac{1}{2} & \frac{1}{2} \\ 0 & 1 \end{matrix}\right\}$};
  Zrange(Q) $\subseteq (1, 4)$: {output 1; n:=n-1;
                        perform transf. $\left\{\begin{matrix} \frac{1}{2} & -\frac{1}{2} \\ 0 & 1 \end{matrix}\right\}$};
                 true : {take more input from x or y};
**end loop;**
Assert{Zrange(Q) $\subseteq (1, -1)$ }
```

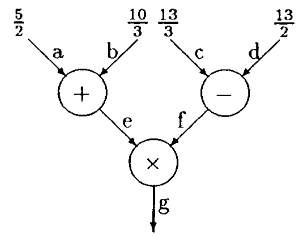
Output of Algorithm RPQ is thus a string $u^{n-1} b_n b_{n-1} \ldots b_0$ which is the representation $R(r)$ of the next partial quotient r of $z(x,y)$, as described by the initial coefficient cube Q. After completion of the algorithm, Q has been transformed into a new cube Q' representing $z'(x,y)$, where

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

in agreement with the factorization (7).

Note that if at least one u has been output in the first loop, then $b_n \neq 0$ and Zrange(Q) $\subseteq (1,4) \cup (-4,-1)$ holds before the second loop is entered. Which subinterval depends on whether a 1 or $\bar{1}$ was generated, and it then follows that the $R(\cdot)$ representation generated is admissible.

We will conclude this section with an example of a pipelined computation of the expression $(\frac{5}{2} + \frac{10}{3})(\frac{13}{3} - \frac{13}{2}) = \frac{455}{36}$, as pictured in the tree below:



The redundant signed bit representations of the operands are:

$$R(\tfrac{5}{2}) = 1010 \qquad R(\tfrac{10}{3}) = 1111$$
$$R(\tfrac{13}{3}) = u10011 \quad R(\tfrac{13}{2}) = u11010$$

The computation in the cells performing addition and subtraction may proceed in parallel, producing the input operands for the cell doing the divide operation. In each cell the eight registers of the coefficient cube are initialized to realize the appropriate operation for that cell. We will however not show the actual operations performed on the coefficient cubes, but illustrate the flow of information in and out of the cells. From a simulation of the RPQ Algorithm in the cells the following was obtained:

```
a: 1 0  1 0    e
b:1 1 1  1    e
c:     u   110          u m m 0  e
d: u 1 0 0 1 1    e
e:u 1 1 0 1 0  e
f:            m 0 u mm   0 e
g:               u u  mm     0 m1 1um00mme
  1234567890123456789012345678901234567890
  0        1         2         3         4
```

Each line here represent the flow along the arcs of the computation tree above, where the horizontal position represents the time step at which the signed bit is on the arc (i.e. consumed). E.g. at step 16 the first bit of the result is produced, based of the availability of the 0 bit produced by node f. The signed bits are represented in the alphabet {u,0,1,m,e}, where m represents $\bar{1}$ and e represents termination. As may be noticed, if both operands are available the individual cells are servicing their arcs in a round-robin manner, which is not the optimal strategy. These results are based on a tool under development, the optimal choice of input selection is still under investigation.

4 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 produce the result $z(x, y)$ signed bit serial in an on-line fashion. The representation of operands and result is redundant over a four letter alphabet, thus allowing a smooth flow of information through the unit. Although we have not been able here to demonstrate that there will be a constant or bounded delay between input and output, this is expected to be the case.

The RPQ Algorithm is fairly straightforward to implement in hardware, 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 Zrange function, which require further development.

An implementation of the Zrange function might use a PLA look-up, based on leading bits of the contents of the eight registers. However a straightforward look-up 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.

We have developed the Euclidean Quotion Bit Engine (EQUBE) simulation system [13] for the purpose of experimenting with alternative decision rules for input/output to our eight register coefficient cube. The facility to deal with signed digits has been incorporated in EQUBE to provide an experimental computing environment to complement the theoretical work on the foundations of arithmetic on redundant continued fractions. A tool for the simulation of interconnected cells is under development by Søren Johansen. This tool implements the RPQ Algorithm, and is intended to be used for experimenting with decision rules, and with composite computations, including possible feed-back in computations.

In summary, our work so far has demonstrated that redundant continued fraction representations can be utilized for an on-line arithmetic over rational operands, but the details require more investigation.

References

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

[2] R. W. Gosper, *Item 101 in Hakmem,* AIM239, MIT, Feb. 1972, pp 37-44.

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

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

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

[6] 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.

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

[8] P. Kornerup and D. W. Matula, *An On-Line Arithmetic Unit for Bit-Pipelined Rational Arithmetic,* Journal of Parallel and Distributed Computing, 5, 1988, pp 310-330.

[9] P. Kornerup and D. W. Matula, *LCF: A Lexicographic Binary Representation of the Rationals,* submitted for publication.

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

[11] 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.

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

[13] K. L. Townsend, P. Bartholomew, M.M. Tanik and D. W. Matula, *EQUBE: Euclidean Quotient Bit Engine Simulator,* Tech. Rep. 88-CSE-32 Southern Methodist University, Dallas, TX 1988.

[14] 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.