A RATIONAL ARITHMETIC PROCESSOR

Mary Jane Irwin
Dwight R. Smith

Department of Computer Science
The Pennsylvania State University

Abstract

An arithmetic processor based upon a rational representation scheme is examined. The key feature of this rational processor is its ability to efficiently reduce a result ratio to its irreducible form (the greatest common divisor of the numerator and denominator is unity). The reduction algorithm presented generates the reduced ratio in parallel with the evaluation of the ratio's greatest common divisor. Hardware designs for the reduction algorithm and the basic arithmetic operations are given.

Introduction

The storage of information in computers usually becomes an attempt to pack the most meaning into the least amount of space. Consequently, numeric data is normally restricted to two forms--integer (or fixed point) and floating point. An alternative number representation scheme examined in this paper is the rational number system[1,2,4]. One instance of rational representation allows for the storage of two integer values which represent a ratio; that is, the numerator and denominator are kept as distinguishable quantities. A comparison between a floating point representation and a rational representation for the fraction 1/3 is given below. Assume a 24 bit floating point system with an implied binary base, a fractional mantissa, and an excess-64 exponent. As can be seen, the floating point representations are inexact as depicted by the continuing string in parentheses, whereas the rational representation has several numerically equivalent, exact representations for 1/3.

```
Floating Point Example:
    0 0111111 1010101010101010 (1010 . . . )
    0 1000000 0101010101010101 (0101 . . . )
    0 1000001 0010101010101010 (1010 . . . )
Rational Example:
    0 00000000001 0 00000000011
    0 00000000010 0 00000000110
    0 00000000011 0 00000001001
```

Consider the several numerical representations for 1/3. It is generally desirable to specify one of these as the unique normal form for storage in memory. In the floating point system that normal form is the representation which is <u>normalized</u>; that is, the representation in which the most significant digit of the mantissa is non-zero. In the rational system, a number is defined to be normalized when either the numerator or the denominator is odd. If both are even then a normalization or "partial reduction" procedure must be employed which involves shifting both the numerator and denominator to the right until either the numerator or the denominator has a one in the right most bit position. This simultaneous shifting does not change the value of the rational number since the original ratio has merely been divided by unity in the form of 2/2. Note that this rational normalization process results in fewer equivalent representations, but does not reduce all representations to one unique one.

In order to achieve one unique representation, rational numbers can further be defined as <u>irreducible</u>. For a ratio to be irreducible, the <u>g</u>reatest <u>c</u>ommon <u>d</u>ivisor (gcd) of the numerator and the denominator must be unity. Reducing rational numbers to this form both eliminates redundant representations and helps to prevent overflow[5,7] of the numerator and/or denominator contents. Unfortunately, producing these irreducible forms does not appear to be a straightforward process. Since, by definition the gcd must be unity, the determination of both the gcd and the consequent reduced ratio requires some evaluation process other than just the simple shifting process required for normalization. Reducible ratios may be introduced during processing by a simple arithmetic operation on two irreducible ratios as follows:

```
 00011      00100      01100
------- * ------- = -------   n
 00010      01001      10010   o  p
( 3/2   *   4/9    =  12/18    r  | r
                        |      m  | o
                        |      a  | c
                        v      l  | e
( 2/3      <----       6/9     i  v s
 00010   reduction    00110    z    s
-------   <----     -------    e
 00011    process     01001
```

The next section of this paper presents a reduction algorithm which produces a reduced ratio in parallel with the evaluation of the greatest common divisor. Then the hardware requirements for this proposed reduction algorithm and for the basic arithmetic operations (addition, subtraction, multiplication, and division) when using a rational system are examined.

The Reduction Algorithm

As has been discussed, all operations involving rational numbers should ultimately yield results that can be stored in reduced form. In addition to diminishing the chances of overflow, this simplifies the use of stored results in future

CH1630-3/81/0000/0241$00.75 © 1981 IEEE

arithmetic operations such as comparisons. An obvious way to reduce the ratio would be to calculate the greatest common divisor of the numerator and denominator and then divide the numerator and denominator by that gcd. However, the hardware and time requirements of this scheme are usually unacceptable. Another reduction method which does not involve division will now be proposed.

In reviewing Euclid's scheme[1,4] for finding the gcd of two numbers, a suitable method for generating the reduced form of a rational number as the gcd is being computed which does not require any explicit divisions can be derived. Euclid's method involves continually subtracting the smaller of two terms (in this instance, the numerator and the denominator) from the larger and then saving the smaller of the two along with the difference for the next iteration. When the difference and the smaller term are the same, the process terminates with that resultant number being the gcd. Only the basic operations of subtraction and comparison need be used. The method proposed in this paper determines the reduced form <u>in parallel</u> with the computation of the gcd according to Euclid's algorithm. The algorithm requires just four additional variables along with the arithmetic operations of addition, subtraction, and shifting. A formal definition of this reduction algorithm is shown at the right.

A correctness proof of the reduction algorithm will now be given. The proof consists of two parts: 1) the reduction algorithm correctly generates the gcd, and 2) the terms a, b, c, and d are, in actuality, the results of division of the numerator and the denominator by that gcd.

<u>Theorem 1</u>

The reduction algorithm correctly generates the gcd in a fashion similar to that of Euclid's gcd algorithm.

<u>Proof:</u>

The correctness of Euclid's algorithm depends upon the fact that at all iterations, the value of the Y and X terms are just multiples of the gcd. This is true at all iterations, for if it were not, then at some time the subtraction of a multiple from another multiple would yield a non-multiple. This is an obvious contradiction. If, at all times, $X = F * gcd$ and $Y = H * gcd$ then the difference $Y - X$ is $H * gcd - F * gcd = (H - F) * gcd$. Thus, it can be seen that the difference of Y and X must also be a multiple of the gcd. The same is true for the reduction algorithm; i.e., at each iteration the values of Y and X are multiples of the gcd. The major difference between the reduction algorithm and Euclid's algorithm in computing the gcd is the fact that Y and X are kept as odd values in the reduction algorithm. The reduction algorithm first partially reduces the ratio to a normalized form. The gcd of two numbers, of which at least one is odd, must also be odd. Thus, forcing the remaining even term odd will not affect the odd gcd. Now, as before, if $X = F * gcd$ and $Y = H * gcd$ where the gcd, X, and Y are odd, then $Y - X = (H - F) * gcd$. Since the gcd is odd and the difference of two odd numbers, $Y - X$, must be even, then $H - F$ must also be even. Therefore, forcing this even difference odd once again does not affect the odd gcd. Thus, the values of Y and X are always multiples of the gcd.

```
Reduction Algorithm
              PROC ( NUM , DEN );
/* NUM and DEN are non-zero positive integers */
          INTEGER NUM, DEN, X, Y, a, b, c, d
/* Normalize the rational number; i.e., force */
/* one of the pair (NUM, DEN) odd */
          WHILE NUM mod 2 = 0 & DEN mod 2 = 0 DO
             NUM = NUM/2; DEN = DEN/2;
          ENDWHILE;
/* Initialize */
          X = NUM; Y = DEN; a, d = 1; b, c = 0;
/* Force the remaining even term odd if necessary */
          WHILE X mod 2 = 0 DO
             X = X/2; a = a * 2;
          ENDWHILE;
          WHILE Y mod 2 = 0 DO
             Y = Y/2; d = d * 2;
          ENDWHILE;
/* GCD iteration */
          WHILE X .NE. Y DO
      SWAP:        IF  Y < X  THEN
                      SWAP  ( X , Y );
                      SWAP  ( a , c );
                      SWAP  ( b , d );
                   ENDIF;
      SUBT:        Y = Y - X;
                   a = a + c;
                   b = b + d;
      FORCE_ODD:   WHILE  Y mod 2 = 0  DO
                      Y = Y/2;
                      c = c * 2;
                      d = d * 2;
                   ENDWHILE;
          ENDWHILE;
/* GCD is the common value of Y and X and the */
/* reduced NUM and DEN are computed from a, b, */
/* c, and d */
          GCD = X;
          NUM = a + c;
          DEN = b + d;
          RETURN;
       ENDPROC;
```

<u>Theorem 2</u>

The additional terms (a, b, c, and d) maintained in the reduction algorithm during the gcd evaluation are the effective results of division of the normalized NUM and DEN by the gcd.

<u>Proof:</u>

The gcd iterative cycle in the reduction algorithm maintains the sums

$$NUM = a * X + c * Y$$
$$DEN = b * X + d * Y$$

under the conditions

1) X and Y are both odd and
2) Y is greater than X.

The following equivalence relations based on these sums can be used to prove the validity of the SUBT portion of the reduction algorithm.

$$NUM = a * X + c * Y - c * X + c * X = (a + c)\ X + c\ (Y - X)$$

and

$$DEN = b * X + d * Y - d * X + d * X = (b + d)\ X + d\ (Y - X)$$

These equations reveal the following iterative equivalences as used in the reduction algorithm

$a = a + c$,
$b = b + d$, and
$Y = Y - X$.

By continued application of the above equations, while guaranteeing that conditions 1) and 2) are maintained, the algorithm will terminate when X and Y are equal. After the last iteration, a, b, c, and d can be used to determine the reduced forms of NUM and DEN since when

$X = Y = GCD$

then

$NUM = a * X + c * Y = (a + c) * X$

and

$DEN = b * X + d * Y = (b + d) * X$

Because X is the gcd of NUM and DEN, the ratio NUM/DEN can be converted to its irreducible form as follows

$$\frac{NUM}{DEN} = \frac{NUM/X}{DEN/X} = \frac{(a + c)}{(b + d)} .$$

Hardware Requirements for a Rational Processor

The hardware requirements of a rational processor will now be presented. Such a processor should be able to perform both the reduction algorithm and the basic operations of add, subtract, multiply and divide with a minimum of complexity and a maximum of speed. It should also be configured to return the truth values of comparisons between pairs of operands.

The four basic arithmetic operations on rational numbers require only hardware multiply, add and subtract units as follows:

$$\frac{J}{L} + \frac{K}{M} = \frac{J*M + K*L}{L*M} \qquad \frac{J}{L} - \frac{K}{M} = \frac{J*M - K*L}{L*M}$$

$$\frac{J}{L} * \frac{K}{M} = \frac{J*K}{L*M} \qquad \frac{J}{L} / \frac{K}{M} = \frac{J*M}{L*K}$$

The necessary operations for the common comparisons are shown below. The equality test assumes operands are irreducible. The other comparisons are of the form (op1 # op2) where # is any of the conditions >, <, >=, or <=. Not equals would be the inverse of the condition for equality.

$$\frac{J}{L} = \frac{K}{M} \Rightarrow J{=}L \ \&\ L{=}M \qquad \frac{J}{L} \# \frac{K}{M} \Rightarrow J*M \# L*K$$

By examining the above relations, it appears that three hardware multiply units are necessary to accomplish the most basic of operations - addition. This number can be reduced to two multiply units if the numerator logic is changed from that of adding the results of two multiplies to that of shifting the sum of two single digit multiplies. Mathematically the problem can be expressed as follows:

$$J*M + K*L \Rightarrow \sum_{i=0}^{n-1} (2^i * j_i * M) + \sum_{i=0}^{n-1} (2^i * k_i * L)$$

$$\Rightarrow \sum_{i=0}^{n-1} 2^i * (j_i * M + k_i * L)$$

In the binary case, the multiplies can be processed through a register file consisting of four registers containing the values

Register 0)	0
Register 1)	M
Register 2)	L
Register 3)	(M + L)

This register file would be addressed by a concatenation of the low order bits of K and J, respectively. Then each multiply could be performed either bit-wise with the output of the register file as the quantity iteratively added to the partial product or, in a fast multiply scheme, with the output of the register file feeding a carry save adder tree. A sample hardware configuration for the numerator logic is given in Figure 1.

Example

Given NUM = 420 and DEN = 231 find the reduced rational representation.

```
  INITIALIZATION:
     NUM/DEN = 420/231 (normalized)
     NUM = 1 * 420 + 0 * 231 = 4 * 105 + 0 * 231
     DEN = 0 * 420 + 1 * 231 = 0 * 105 + 1 * 231
  GCD ITERATION:

                 SWAP          SUBT       FORCE ODD

  1) NUM=4*105+0*231= 4*105+0*126= 4*105+0*63
     DEN=0*105+1*231= 1*105+1*126= 1*105+2*63
  2) NUM=0* 63+4*105= 4* 63+4* 42= 4* 63+8*21
     DEN=2* 63+1*105= 3* 63+1* 42= 3* 63+2*21
  3) NUM=8* 21+4* 63=12* 21+4* 42=12* 21+8*21
     DEN=2* 21+3* 63= 5* 21+3* 42= 5* 21+6*21

  GCD RESULTS:
     GCD = 21

  REDUCED RATIO:
     NUM = (12 + 8) = 20
     DEN = (5 + 6) = 11
```

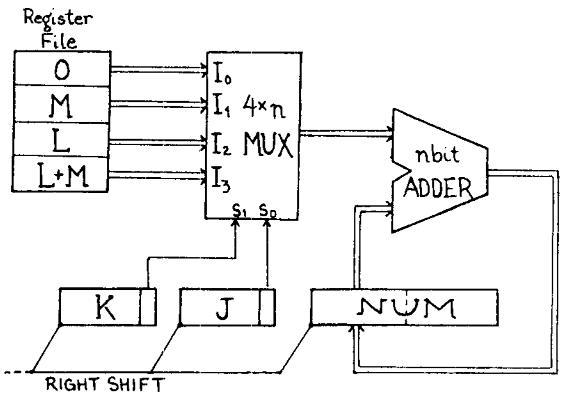

Figure 1. The Numerator Hardware

The values loaded into the register file initially would determine the operation to be performed by the numerator hardware. Loaded as given in Figure 1, the final output would be the numerator portion for rational addition. If instead, registers 2) and 3) are also set to zero, the numerator for the multiply or divide (and compare) could be obtained by placing either K or M, respectively, into register 1). The execution would then continue as before to obtain the product of either K or M by J. For subtraction, -L and M - L should be loaded into registers 2) and 3), respectively.

The denominator hardware could be a standard n bit multiplier. The denominator hardware would multiply either K or M by L. The denominator multiply hardware should be of the same basic structure and require a similar execution time as that of the numerator hardware to enable parallel computation of the two values. The output of the numerator and denominator hardware could be used at this point to set condition codes or could serve as inputs to the reduction hardware shown in Figure 2.

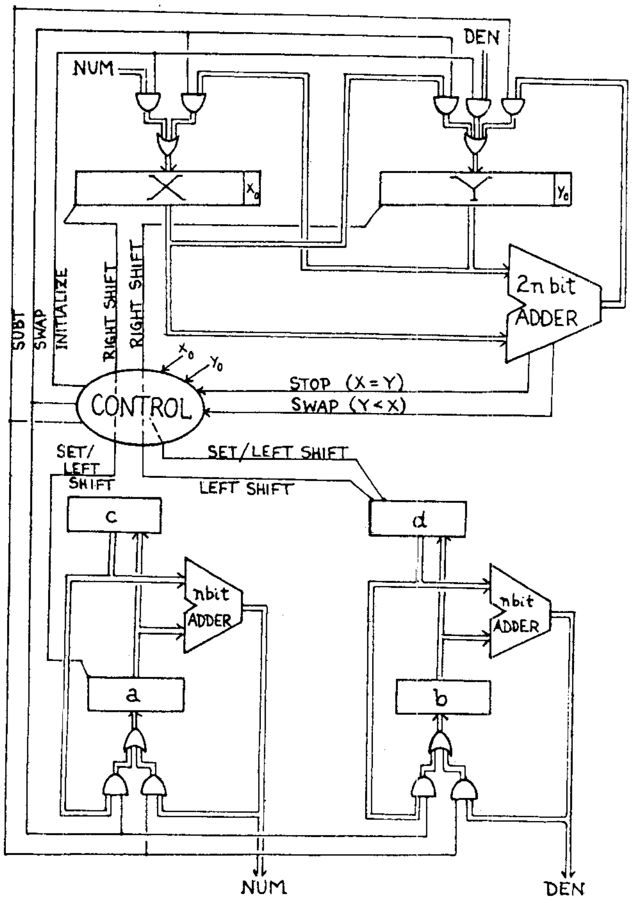

Figure 2. The Reduction Hardware

The very nature of the reduction algorithm as an iterative method yields a variable time operation whose worst case processing time is O(max(log(NUM), log(DEN))). This may or may not be of importance. If the rational processor represents one segment of a pipeline, then the worst case performance must be allowed for. On the other hand, the reduction can be done at the end of the pipeline immediately before the values are to be stored. In this case the register widths would have to be expanded in the pipeline to prevent loss due to overflow.

Summary

A rational representation scheme has been shown to be a viable, implementable scheme. The key feature of the proposed rational processor is the ability to efficiently reduce ratios according to the reduction algorithm presented. Several other important questions remain unanswered by this paper. What are the possible expansions of the rational representation to extend the representable range of numbers? Does using the rational format provide a "better" or "worse" representation of numbers than that of standard floating point formats? How does reduction affect the possibility of overflow? The reader is referred to the literature[2,3,5,6,7,8] for further insight into these issues.

References

[1] Hardy, G. H. and E. M. Wright, "An Introduction to the Theory of Numbers," Clarendon Press, Oxford, 1954.

[2] Hehner, E. C. and R. N. Horspool, "A New Representation of the Rational Numbers for Fast Easy Arithmetic," Siam Journal on Computing, Vol. 8, No. 2, May 1979.

[3] Hwang, Kai and T. P. Chang, "An Interleaved Rational/Radix Arithmetic System for High-Precision Computations," Proc. of the Fourth Symposium on Computer Arithmetic, Santa Monica, CA, 1978.

[4] Knuth, D. E., "The Art of Computer Programming: Vol. 1, Fundamental Algorithms" and "The Art of Comp. Prog.: Vol. 2, Seminumerical Algorithms," Addison-Wesley, 1969.

[5] Kornerup, Peter and David Matula, "A Feasibility Analysis of Fixed-Slash Rational Arithmetic," Proc. of the Fourth Symposium on Computer Arithmetic, Santa Monica, CA, 1978.

[6] Matula, David W., "Fixed-Slash and Floating-Slash Rational Arithmetic" Proc. of the Third Symposium on Computer Arithmetic, Dallas, TX, 1975.

[7] Matula, David W. and Peter Kornerup, "A Feasibility Analysis of Binary Fixed-Slash and Floating-Slash Number Systems," Proc. the Fourth Symposium on Computer Arithmetic, Santa Monica, CA, 1978.

[8] Smith, Dwight R., "A Study of Fixed-Slash Numbers in the Context of a Comparison to Floating Point Numbers," M.S. Paper, Department of Computer Science, The Pennsylvania State University, University Park, PA, 1980.