Improved Table Lookup Algorithms for Postscaled Division

David W. Matula
Department of Computer Science and Engineering
Southern Methodist University
Dallas, Texas, 75275-0122
matula@seas.smu.edu

Abstract

Postscaled division is a non-iterative algorithm delivering a quotient of single precision accuracy by the three term product $(x\hat{y})c$ and of double precision accuracy by the formula $[(x\hat{y})c][2-(y\hat{y})c]$. Here x is the dividend, $\hat{y}$ is a low order part complemented form of the divisor y, and c is a table lookup value approximating a "reciprocal function" $1/(y\hat{y})$ to a precision of over 27 bits. Table lookup latency is hidden by performing the lookup in parallel with the first multiplication $(x\hat{y})$, with the second multiplication the "postscaling" by the lookup function value.

Our contribution herein is the description of two new lookup algorithms for approximating the reciprocal function $\frac{1}{y\hat{y}}$ to high accuracy in fewer cycles than a typical floating point multiply latency. Our indirect bipartite lookup procedure has a latency of two successive lookups followed by a small integer addition. This first algorithm generates a 27 bit approximation of $\frac{1}{y\hat{y}}$ with total table size about 5 Kbytes. Our second lookup algorithm generates a 34 bit approximation with latency determined by 11 and 12 bit table lookups and a 4-1 addition. This second approximation employs some 20 Kbytes of tables to allow for a double extended precision division result in the same number of cycles as a double precision result.

1 Introduction and Summary

Division is the hardest to implement of the standard arithmetic operations due to the inherent dependent operations in all of the popular known algorithms [Go64, Fl70, FS89, WF91, EL94]. For high precision division the iterative methods based on Newton Raphson reciprocal refinement seem the most efficient. Goldschmidt [Go64] provided a major improvement to the reciprocal refinement method for pipelined multipliers by cutting in half the number of required dependent multiplications to achieve a desired accuracy.

Many researchers have dealt with improvements to Goldschmidt's method either by employing a more accurate initial table lookup procedure or by accelerating the iteration accuracy [WF91, DM95, IT97, Ob99, EI00].

In a recent paper [IM99] Iordache and this author presented a new variation of the Newton Raphson refinement procedure that effectively allowed a formula based non-iterative computation for IEEE standard [IEEE85] single and double precision division. Single precision is provided by the three term product $(x\hat{y})c$ and double precision by the formula $[(x\hat{y})c][2-(y\hat{y})c]$. Here x is the dividend, $\hat{y}$ is a low order part complemented form of the divisor y, and c is a table lookup value approximating a "reciprocal function" $1/(y\hat{y})$ to a precision of over 27 bits.

The impetus for our method comes from the recent result of Ito, Takagi and Yajima [IT97] that the equivalent of linear interpolation for a reciprocal approximation can be obtained by a single multiplication operation $c\hat{y}$ with a single table lookup of c. The reformulation of one Newton Raphson refinement to obtain a high precision quotient approximation from the formula $[(x\hat{y})c][2-(y\hat{y})c]$ offered a novel approach to speeding up division. In particular, the operations may be rescheduled to start with the pipelined computations of $y\hat{y}$ and $x\hat{y}$ while the table lookup value c is determined in parallel and off the critical path. Since the constant $c \approx \frac{1}{y\hat{y}}$ need only be obtained in the time of the pipelined multiplier latency, new table lookup algorithms utilizing up to 3 or 4 cycles can be employed without increasing the overall division operation latency.

In Section 2 we summarize the "postscaled" division algorithm and results from [IM99] sufficient to make this paper self contained.In Section 3 we describe a table lookup procedure using successive dependent table lookups. The resulting "indirect" bipartite lookup algorithm provides that $c \approx \frac{1}{y\hat{y}}$ can be determined to some 28 bits of accuracy employing only 5 Kbytes of tables with a latency of two successive 10 bit indexed table lookups and a low precision addition.

In Section 4 we describe a table lookup procedure hav-

0-7695-1150-3/01 $10.00 © 2001 IEEE

ing the latency of a table lookup and a 4-1 addition. This improved multipartite lookup procedure provides some 27 bits of accuracy for $c \approx \frac{1}{y\hat{y}}$ with less then 4 Kbytes of tables, and nearly 34 bits of accuracy with some 20 Kbytes of tables. The latter result allows for an aggressive design achieving double extended division in the same number of cycles as double precision.

2 Postscaled Division

Let $y = 1.b_1 b_2 b_3 \cdots b_{p-1}$ be a normalized *p-bit divisor* and $y < x \leq 2y$ be a *p-bit dividend* normalized relative to y so that the infinitely precise quotient falls in the standard binade $1 \leq q = \frac{x}{y} < 2$. Let $y_i = 1.b_1 b_2 b_3 \cdots b_i$, with the i bit string $b_1 b_2 b_3 \cdots b_i$ providing the *primary index* for a reciprocal function table lookup. Let $f = .b_{i+1} b_{i+2} \cdots b_{p-1}$ be the low order part of y normalized as a fraction so that $y = y_i + f2^{-i}$. We shall also be interested in a partially complemented version of the divisor $\hat{y} = y_i + (1-f)2^{-i}$, termed the *low order part complemented* divisor.

Ito, Takaji and Yajima introduced and demonstrated a fundamental property of the low order part complemented divisor that can be expressed as follows:

Lemma 1 [IT97]: Let a linear interpolated approximation of the reciprocal of the divisor $y = y_i + f2^{-i}$, be given by the multiply-add operation $approx(\frac{1}{y}) = c_1 - c_2 f 2^{-i}$, with $c_1 = \frac{1}{y_i}, c_2 = \frac{1}{y_i(y_i+2^{-i})}$. Employing the low order part complemented divisor $\hat{y} = y_i + (1-f)2^{-i}$, we can compute $approx(\frac{1}{y})$ by a single multiply operation

$$approx(\frac{1}{y}) = c_2 \hat{y}. \tag{1}$$

Employing (1) for linear interpolation, only one table lookup is required. More important for our purposes is the fact that the operation simplifies to a single multiplication, albeit of necessarily higher precision operands.

Lemma 1 allows that the i bit indexed table lookup value c_2 provides an approximation to $\frac{1}{y\hat{y}}$ of accuracy somewhat over $2i$ bits. Note that $y = y_i + f2^{-i}$ and $\hat{y} = y_i + (1-f)2^{-i}$ are symmetric about the midpoint within the $i+1$ bit ulp interval determined by y_i and $y_i + 2^{-i}$. Thus the product $y\hat{y}$ has the first order effect from f cancel out, leaving only a small second order dependence on f. Thus the reciprocal function $\frac{1}{y\hat{y}}$ considered as a function of i and f has a large dependence on the i bit string $b_1 b_2 \cdots b_i$ and only a second order dependence on $f = .b_{i+1} b_{i+2} \cdots$,

$$\frac{1}{y\hat{y}} = \frac{1}{y_i^2 + y_i 2^{-i} + f(1-f)2^{-2i}}. \tag{2}$$

The importance of the multiplicative nature of (1) and the facility to obtain a very accurate approximation of $\frac{1}{y\hat{y}}$ with a relatively small sized table lookup led Iordache and this author [IM99] to propose a simplified non-iterative division algorithm where the equivalent of just one Newton Raphson reciprocal refinement provides a quotient approximation of double precision accuracy. The following provides the fundamental operations sufficient to obtain IEEE single ($p = 24$) and double ($p = 53$) precision results.

Observation 2 [IM99]: Let $c = \frac{1}{y\hat{y}}(1+\delta)$ with $|\delta| < 2^{-27}$. Then

$$\begin{aligned} q_1 &= (x\hat{y})c = \frac{x}{y}(1+\delta), \quad |\delta| < 2^{-27}, & (3) \\ q_2 &= [(x\hat{y})c][2 - (y\hat{y}c)] & \\ &= \frac{x}{y}(1-\delta^2), \quad \delta^2 < 2^{-54}. & (4) \end{aligned}$$

For (3) note that q_1 may be rounded to 25 bits with less than one ulp error in the last place, and this is sufficient to determine the correct round and sticky bits from the sign of the remainder associated with q_1. For a double precision result, q_2 in (4) can be rounded up to 54 bits determining a rounded quotient with less than one ulp error in the last place, allowing the correct round and sticky bits to be computed from the sign of the associated remainder.

The equations for q_1 and q_2 in Observation 2 facilitate scheduling of concurrent computations leading to an improved cycle time compared to division algorithms employing multiple dependent operations. A description of the algorithm [IM99] sufficient to make this paper self contained is presented here.

Postscaled Division Algorithm

For a *single precision quotient* ($p = 24$):

- Step 1: The input dividend x is multiplied by the low order complemented input divisor $\hat{y}$, with table lookup of c initiated in parallel.
- Step 2: The product $(x\hat{y})$ is multiplied by the lookup value c and the result rounded to 25 places determining q'.
- Step 3: The positive, negative or 0 status of the associated remainder $rem = x - yq'$ is determined by a multiply-add operation or a separate logic computation, and the result used to select the correctly rounded 24 bit result.

For a *double precision quotient* ($p = 53$):

- Step 1: The input dividend x and the input divisor y are each multiplied by the low order complemented divisor $\hat{y}$, with table lookup of c initiated in parallel.
- Step 2: The products $(x\hat{y})$ and $(y\hat{y})$ are each multiplied by the lookup value c.

- Step 3: The product $[(x\hat{y})c]$ is multiplied by the 2's complement product $[2-(y\hat{y})c]$ and the result rounded to 54 bits determining q'.

- Step 4: The positive, negative or 0 status of the associated remainder $rem = x - yq'$ is determined by a multiply-add operation or a separate logic computation, and the result used to select the correctly rounded p=53 bit result.

Properties of Postscaled Division

- The latency for a single precision rounded result is two multiply latencies plus the rounding latency.

- The latency for a double precision rounded result is three multiply latencies plus the rounding latency.

- For a pipelined multiplier with the remainder sign computed in alternative circuitry, the pipeline stall determining the throughput is just one multiply latency for single precision division, and two multiply latencies plus one cycle for double precision division.

- The facility to schedule multiplication by the table lookup value c as the second operation in evaluating the three term product $x\hat{y}c$ (alias postscaling), allows the time of a full multiply latency for table assisted computation of c.

Figure 1 illustrates a pipeline schedule for a double precision implementation of postscaled division. In this example we have assumed a 4 cycle latency multiply operation for a dependent multiply which allows up to four cycles for lookup of c. We have also assumed that rounding can be performed in two cycles by producing two alternative rounded values with final selection dictated by the parallel computation of the remainder sign(+, -, or zero) in special circuitry.

The postscaled division example of Figure 1 suggests the feasibility of implementing IEEE standard double precision division with a throughput of 9 cycles and a latency of 15 cycles. A necessary feature to achieve such an implementation is a cost effective table lookup procedure for determining $c \approx \frac{1}{y\hat{y}}$ to some 27 bits of accuracy. The case for *double extended precision (p = 64)* division is considerably more demanding. To achieve a 64 bit rounded quotient, and possibly also to support an internal division of say 68 to 70 bits for microcoded transcendentals, some 33 to 35 bits of accuracy in the approximate c would be required.

The following identity from [IM 99] is the basis for modeling our table lookup algorithms.

Cycle	Operations
1	$y \times \hat{y}$, initiate lookup of c
2	$x \times \hat{y}$
3	-
4	-
5	$(y\hat{y}) \times c$
6	$(x\hat{y}) \times c$
7	-
8	-
9	2's comp $(y\hat{y}c)$
10	$(x\hat{y}c) \times (2 - y\hat{y}c)$
11	-
12	-
13	-
14	initiate rounding
15	rounded output available

Figure 1. Pipeline schedule for an implementation of postscaled division.

Observation 3: The reciprocal function $\frac{1}{y\hat{y}}$ for a given i and f satisfies

$$\frac{1}{y\hat{y}} = C_i + \frac{1-8f(1-f)}{y_i^2(y_i+2^{-i})^2}2^{-(2i+3)} + \delta 2^{-(4i+4)}, \quad (5)$$

with $0 \le \delta < 1$, where

$$C_i = \frac{1}{y_i(y_i+2^{-i})} - \frac{2^{-(2i+3)}}{y_i^2(y_i+2^{-i})^2}. \quad (6)$$

Note that C_i is a function of the i bit primary index $b_1b_2b_3\cdots b_i$ and may be looked up to any desired accuracy in a table of sufficient width, say here between $3i$ and $4i+4$ bits. For $1 \le \alpha \le 2i$, let

$$C' = C'(y_i, f) = \frac{1-8f(1-f)}{y_i^2(y_i+2^{-i})^2}(1+\epsilon 2^{-\alpha}), \ |\epsilon| < 1, \quad (7)$$

be an approximation of the second term in (5) accurate to α bits.

Employing $c = C_i + C'2^{-(2i+3)}$ then yields a quotient approximation $x\hat{y}c$ of accuracy some $2i+\alpha+3$ bits. We note the following tradeoffs in obtaining $2i+\alpha+3 = 27$.

Employing only $C_i \approx \frac{1}{y\hat{y}}$ requires an i = 12 bit primary table with width 39 bits and table size 19.5 Kbytes [IM99] to obtain 27 bit accuracy. Using an i = 11 bit primary table and deriving α = 2 bits of accuracy from a table for the C' term, the total table size for a 27 bit accurate approximation can be reduced to a manageable 7.5 Kbytes by bipartite evaluation [IM99]. To further reduce table size and lookup time for the primary table for C_i, it is desirable to find better approximation schemes for $C' = \frac{1-8f(1-f)}{y_i^2(y_i+2^{-i})^2}$ yielding accuracies of say one part in 2^{α} for $4 \le \alpha \le 9$.

3 Indirect Bipartite Table Lookup

Direct bipartite evaluation of $c = C_i + C'$ involves table lookup of $C' = \frac{1-8f(1-f)}{y_i^2(y_i+2^{-i})^2}$ using some $\frac{i}{2}$ leading fraction bits each from y_i and f. Only some $\alpha \approx \frac{i}{2} - 2$ bits of improvement come from the C' term due to the excessive non linearity in both the numerator *f*-factor $1 - 8f(1-f)$ and reciprocal denominator *y*-factor $\frac{1}{y_i^2(y_i-2^{-i})^2}$. Note in particular that for $1 \le y \le 1\frac{1}{8}$, the *y*-factor covers the interval $[.625, 1]$, which is most of a binade. Similarly, for $.933 \le f \le 1$, the *f*-factor covers the binade $[.5, 1]$. Thus some four bits (two for *y* and two for *f*) of the C' table index is lost to this non linear "overhead".

Using an 11-*bits-in* 29-*bits-out* table for C_{11} and an 8-*bits-in* 8-*bits-out* table for C', the approximation $c = C_{11} + C'$ was confirmed to provide 27 bit accuracy for approximation of $\frac{1}{y\hat{y}}$ in [IM99]. The tables total a somewhat modest size of 7.5 Kbytes, but further reduction is desirable. Note that a 10-*bits-in* table for $c = C_{10} + C'$ requires a table index of some 12 bits for C' to produce a 27 bit accurate c, making the secondary table too large.

We now present an indirect bipartite lookup algorithm that substantially reduces table size at the cost of an additional dependent table lookup step in the lookup value latency. We introduce an *index mapping table* that briefly operates as follows. The range $[\frac{1}{16}, 1]$ of the factor $\frac{1}{y_i^2(y_i+2^{-i})^2}$ is partitioned into $2^{n'}$ intervals with widths of equal or decreasing size as the interval boundaries approach unity. The y_i *mapping table* is *i-bits-in, n′-bits-out* giving an n' bit *y-key* denoting the interval containing the factor $\frac{1}{y_i^2(y_i+2^{-i})^2}$. For the factor $1 - 8f(1-f)$ the unit interval is partitioned into $2^{n''}$ intervals with widths of non increasing size. The f_j *mapping table* is then a *j-bits-in*, $(n''+1)$-*bits-out* table giving a sign bit and an n'' *bit f-key* denoting the interval containing the magnitude of the factor $1 - 8f_j(1-f_j)$, where $f_j = .b_{i+1}b_{i+2}\cdots b_{i+j}1$ is determined by the *j-bit secondary index* $b_{i+1}b_{i+2}\cdots b_{i+j}$. The y_i and f_j mapping table output keys are concatenated to form an $(n'+n'')$ bit index to a *product table* that produces an approximation of C'.

Indirect Bipartite Lookup Algorithm

This algorithm uses three table lookups and an addition to form the lookup value $c = C_i + C' \approx \frac{1}{y\hat{y}}$.

- Step 1: The primary index $b_1b_2\cdots b_i$ is utilized to lookup both the term C_i and an $n' \approx \frac{i}{2}$ bit *y-key* for the factor $\frac{1}{y_i^2(y_i+2^{-i})^2}$. Separably and in parallel a $j \approx i$ bit index $b_{i+1}b_{i+2}\cdots b_{i+j}$ is utilized to lookup an $n'' \approx \frac{i}{2}$ bit *f*-key and a sign bit s in an f_j mapping table.
- Step 2 [Second level lookup]: The *y*- and *f*-keys are concatenated to form an $(n'+n'') \approx i$ bit index into a product table providing the magnitude of C'.
- Step 3 [Fusion]: The value $c = C_i + (-1)^s|C'|$ is formed.

Performance of Indirect Bipartite Lookup

Latency: The latency is the time of two successive i bit table lookups plus a small precision integer addition.

Accuracy: The accuracy is $\approx (\frac{5}{2}i + 3)$ bits.

Table Size: The total table size is about $2^{(i-10)}(\frac{i+1}{2})$ Kbytes, split about $\frac{2}{3}$ for C_i and $\frac{1}{3}$ for the C' tables.

The symmetry of the product $f(1-f)$ allows the f_j mapping table to be halved in size by employing the conditionally complemented $j-1$ bit index determined by

$$f_j' = \begin{cases} .0b_{i+2}b_{i+3}\cdots b_{i+j}1 & \text{if } b_{i+1} = 0\,, \\ .0\bar{b}_{i+2}\bar{b}_{i+3}\cdots \bar{b}_{i+j}1 & \text{if } b_{i+1} = 1\,. \end{cases} \tag{8}$$

Figure 2 illustrates a circuit for indirect bipartite lookup. Let us consider the case i=10 in Figure 2. Table 1 employs a 10-*bits-in* 34-*bits-out* table producing both a 29 bit value for C_{10} and a 5-bit key. Table 2 uses the conditionally complemented 9 bit index for f_{10}' from (8) to output a sign bit and a 5 bit key. Table 3 employs a 10 bit index formed from the two keys to output a 6 bit value for the magnitude of C'. The adder then yields a 29 bit value for $\frac{1}{y\hat{y}} \approx c = C_{10} + (-1)^s|C'|$. Total table size is slightly less than $5\frac{1}{2}$ Kbytes with accuracy for $\frac{1}{y\hat{y}}$ of about 28 bits.

The indirect bipartite lookup algorithm can be refined to allow for smaller table sizes by having the mapping tables produce keys denoting floating point factors. Specifically, the y_i mapping table can provide a 2 bit exponent and an $n' \approx \frac{1}{2}i$ bit index denoting one of $2^{n'}$ intervals covering the range $[\frac{1}{2}, 1]$ that contains the significand value for the *y*-factor $\frac{1}{y_i^2(y_i+2^{-i})^2}$. Similarly, the f_j mapping table can provide a sign bit, an exponent, and an n'' bit index for the significand value of the factor $1 - 8f_j(1-f_j)$. This allows the product table to output a floating point product having one additional bit of accuracy since each of the input significands is in an interval of about one half the size of the inputs previously described.

Figure 3 illustrates a circuit for this floating point version using a 9 bit primary index to produce 21 bits of accuracy

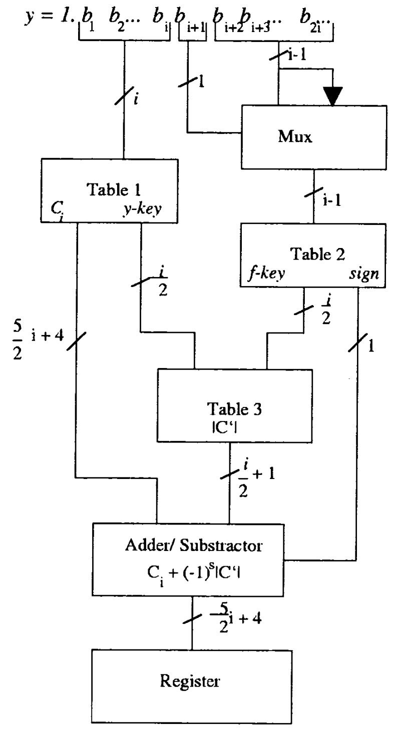

Figure 2. Table lookup circuit for indirect bipartite lookup.

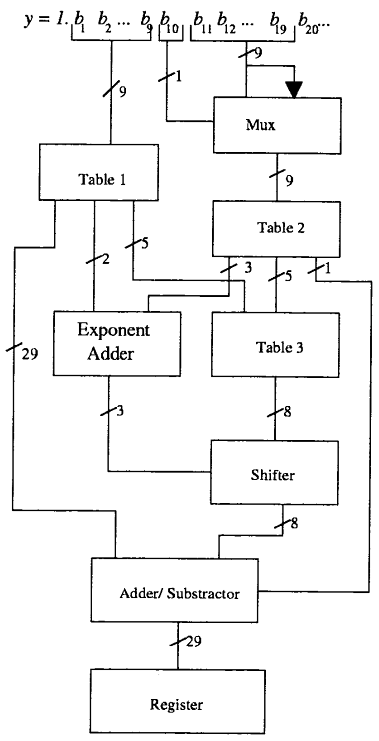

Figure 3. Table lookup circuit for floating point indirect bipartite lookup of $c \approx \frac{1}{y\hat{y}}$ to 27 bit accuracy.

in C_9. The secondary table for C' provides an additional 6 bits of accuracy to achieve $\frac{1}{y\hat{y}} \approx c = C_9 + C'$.

In Figure 3, Table 1 produces a *y-key* with 2 exponent bits and 5 significand bits determining an interval of width about $\frac{1}{2^8}$ for the factor $\frac{1}{y_i^2(y_i+2^{-i})^2}$. Table 2 produces an *f-key* with a sign bit, 3 exponent bits, and 5 significand bits determining an interval of width about $\frac{1}{2^8}$ for the factor $1 - 8f(1 - f)$. Three exponent bits are sufficient since otherwise $|C'| < 2^{-29}$. The product is then determined by Table 3 to be in an interval of size less than $\frac{1}{2^8}$, with a midpoint approximation giving accuracy to $\pm\frac{1}{2^9}$. It is possible to systematically reduce the intervals covering [.5, 1] from somewhat over $\frac{1}{2^8}$ at .5 to somewhat less than $\frac{1}{2^8}$ at unity, allowing the product table to produce about $\frac{1}{2}$ bit extra accuracy in the limit. The details are somewhat tedious and will not be given here. The extra accuracy allows for relatively few output guard bits to provide the overall 27 bit accuracy.

The architecture of the table lookup scheme in Figure 3 provides for 9 bit indices for Tables 1 and 2, which should be implementable in a single cycle. The 10 bit index lookup of Table 3 and shift of the output should be possible in at most 2 more cycles, with a 4th cycle for the low precision addition. This lookup scheme would match the critical path for a four cycle latency multiplier producing the first product $x\hat{y}$ in the postscaled algorithm. Note that the total table size here is under 4 Kbytes to achieve the 27 bit accurate result at a latency of two successive table lookups followed by a shift and a 2-1 add. Furthermore, note that the adder-subtractor could be replaced by a redundant binary recoder [Ni82, DM95, LM95] providing the output c in Booth recoded form for use in the subsequent multiply operation.

The floating point indirect bipartite lookup procedure provides a candidate for double extended precision accuracy using $i = 12$. In this case the six bit significand keys for the y- and f- factors yield a 12 bit index product table with output producing C' to some 7 bits of accuracy. The term c is then available to some 34 bits of accuracy sufficient for a double precision quotient of some 69 bits of accuracy.

In this case the total table size would be some 28 Kbytes, with the successive 12 bit table lookups possibly extending the table lookup computation beyond the pipelined multiplier feedback latency. For double extended precision postscaled division an alternative multipartite table procedure is described in the next section.

All three bipartite algorithms for approximating $\frac{1}{y\hat{y}}$ by $c = C_i + C'$ employ a primary table for C_i that contributes accuracy quadratically to c at the rate of 2 bits of accuracy per table index bit. Similarly, the three algorithms contribute just $\frac{1}{2}$ bit of accuracy per table index bit in the secondary table system for C'. The substantial improvement in accuracy as a function of table size for the indirect bipartite schemes derives from the design overhead changing from a penalty of two bits to a bonus of one bit in contributions from the secondary table system. Table 1 summarizes the contributions of the table system components in terms of bits of accuracy added per table index bit. Employing bipartite lookup with i bit primary and secondary table indices yields accuracy at the asymptotic rate of $2\frac{1}{2}$ bits per table index bit in all three cases in Table 1. Alternative table assisted schemes are needed to obtain a higher asymptotic rate.

Table Part	Direct Bipartite	Indirect Bipartite	Floating Point Indirect Bipartite
primary	$2i + 3$	$2i + 3$	$2i + 3$
secondary	$\frac{1}{2}i - 2$	$\frac{1}{2}i$	$\frac{1}{2}i + 1$

Table 1. Bits of accuracy in $\frac{1}{y\hat{y}}$ approximation per table index bit of the primary and secondary tables for various bipartite lookup algorithms.

4 A Multipartite Lookup Algorithm

An asymptotic rate of three bits of accuracy per table index bit is seen to be theoretically possible by employing an underlying log/antilog table system. Consider that a first pair of tables can deliver the logs of the y-factor and f-factor of C'. The logs can be added and input to an antilog table determining C', which is then added to C_i to determine c. With i bit indexed base 2 log and antilog tables for the mantissas, essentially i bits of accuracy can be obtained for C' providing a $3i$ bit accurate c. The latency is two successive i bit table lookups alternating with two 2-1 low precision additions. For table size indices i = 11 or 12 as may be needed for double extended precision, the latency of this log/antilog system could exceed the pipelined multiplier feedback latency.

In the following we present an alternative lookup system achieving a $3i$ bit accurate $c \approx \frac{1}{y\hat{y}}$ over the practical index size range $8 \leq i \leq 12$ with latency a more acceptable i bit table lookup and a 4-1 add. This final table lookup algorithm generates an approximation to the product C' of the y-factor $\frac{1}{y_i^2(y_i+2^{-i})^2}$ and the f-factor $1 - 8f_j(1 - f_j)$ as the sum of three partial products. The algorithm and its performance are outlined followed by more details on two proposed implementations.

Table Fed Partial Product Generation Algorithm

This algorithm uses two table lookups and a 4-1 addition to form the lookup value $c = C_i + C' \approx \frac{1}{y\hat{y}}$.

- Step 1: The primary index $b_1b_2\cdots b_i$ is utilized to lookup the term C_i and one or two $n \approx i$ bit scaled partial products of the y-factor $\frac{1}{y_i^2(y_i+2^{-i})^2}$. Separably and in parallel a $j \approx i$ bit secondary index $b_{i+1}b_{i+2}\cdots b_{i+j}$ is utilized to lookup a 3-digit Booth encoded radix 4 or 8 value $d_2\beta^2 + d_1\beta + d_0$ denoting the interval containing the f-factor $1 - 8f_j(1 - f_j)$.
- Step 2: Each of the three Booth encoded digits is used to select an appropriate final partial product of the y-factor.
- Step 3: The three partial products and the C_i term are accumulated in a 4-1 adder providing c.

Performance of the Table Fed PPG Algorithm

The following performance characteristics pertains to the primary index size range $8 \leq i \leq 12$ with the secondary index size $j \approx i$.

Latency: The latency is an i-bit table lookup followed by a 4-1 addition.

Accuracy: The accuracy is $\approx 3i$ bits.

Table Size: The total table size is about $2^{(i-10)}(\frac{3}{4}i + \frac{1}{2})$ Kbytes.

The table size refers to radix 8 partial product accumulation. For $i \leq 9$, radix 4 PP-accumulation saves about 15% in table size. The Table Fed PPG implementation can be extended to $i \geq 13$ with the same accuracy and table size performance, but the latency must grow to be an i bit table lookup followed by an $\lceil\frac{i}{3}\rceil$ to 1 addition.

Figure 4 illustrates a Table Fed PPG circuit with 9 bit tables employing radix 4 Booth encoding.

Figure 5 illustrates a similar circuit with an 11 bit indexed primary table and a 12 bit indexed secondary table employing radix 8 Booth encoding. The latency and table sizes are evident from the figures. Some details on the table value construction will provide the basis for the accuracy of the resulting approximation in each case.

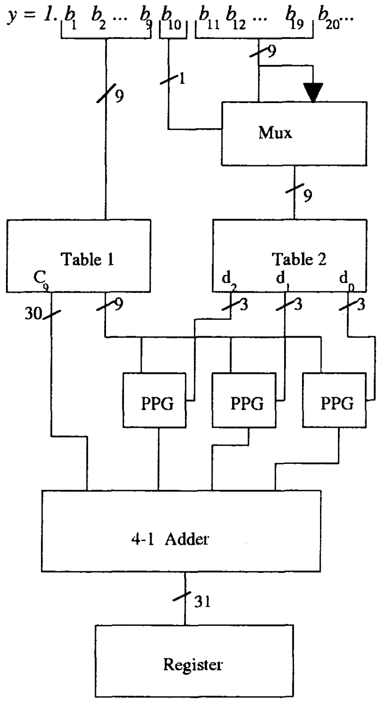

Figure 4. Table Fed PPG circuit with 9 bit tables and Booth radix 4 PPG's.

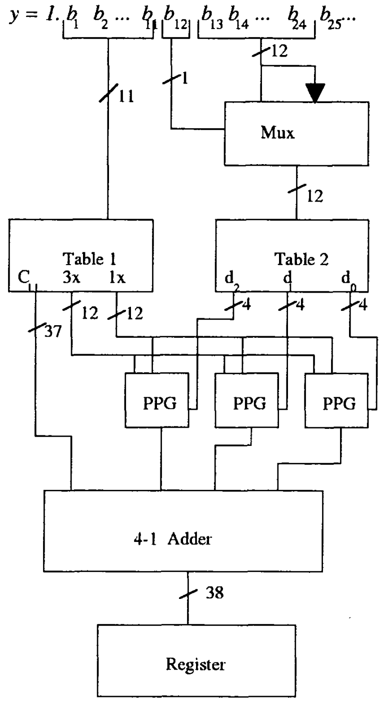

Figure 5. Table Fed PPG circuit with 11/12 bit tables and Booth radix 8 PPG's.

Note that any fixed multiple of the y-factor $\frac{1}{y_i^2(y_i+2^{-i})^2}$ can be looked up to arbitrary accuracy using the i-bit primary index, so the error analysis rests heavily on the features of the f-factor approximation in Table 2.

Consider the radix 4 case with 9 bit tables in Figure 4. For radix 4 with digit set $\{-2,-1,0,1,2\}$, the three digit number $N = d_2 4^2 + d_1 4 + d_0$ can represent any integer in the range $[-42, 42]$. Thus the interval $(-1, 1]$ can be partitioned into 85 intervals $(\frac{N-\frac{1}{2}}{42\frac{1}{2}}, \frac{N+\frac{1}{2}}{42\frac{1}{2}}]$ with N chosen in Table 2 such that the f-factor with $j = 10$ satisfies $\frac{N-\frac{1}{2}}{42\frac{1}{2}} < 1 - 8f_{10}(1-f_{10}) \leq \frac{N+\frac{1}{2}}{42\frac{1}{2}}$. The maximum error in approximating $1 - 8f_{10}(1-f_{10})$ is then $\frac{1}{85}$ or about 6.41 bits. The maximum error in approximating $1-8f(1-f)$ by $1-8f_{10}(1-f_{10})$ is $\frac{1}{256}$, so the resulting accuracy is 6-bits. With Table 2 encoding only the factor $N = d_2 4^2 + d_1 4 + d_0$, the output from Table 1 is the scaled y-factor $\frac{1}{(42\frac{1}{2})y^2(y_i+2^{-i})^2}$.

Overall the Table Fed PPG circuit of Figure 4 provides an approximation to $\frac{1}{y\hat{y}}$ of accuracy 27 bits with total table size just 3Kbytes. The circuit should be implementable in just 2 or 3 cycles. Note that the second table may be increased to a 10 bit index to improve the approximation of $1 - 8f(1-f)$ to about 6.2 bits with only a modest increase of about $\frac{1}{2}$ Kbyte in table size. Alternatively a radix 8 version could provide over 7 bits of accuracy from the secondary table system. Thus design alternatives exist for reaching the goal of 27^+ bits of accuracy in the approximation of $\frac{1}{y\hat{y}}$ with table size under 4 Kbytes and a table lookup latency of three cycles or less.

Now consider the radix 8 case with an 11 bit indexed primary table and a 12 bit indexed secondary table as illustrated in Figure 5. The three digit number $N = d_2 8^2 + d_1 8 + d_0$ with $-4 \leq d_0, d_1, d_2 \leq 4$, has the integer range $[-292, 292]$. We thus choose a digit encoding for N in Table 2 such that $\frac{N-\frac{1}{2}}{292\frac{1}{2}} < 1 - 8f_{13}(1-f_{13}) \leq \frac{N+\frac{1}{2}}{292\frac{1}{2}}$. The maximum error in approximating $1 - 8f_{13}(1-f_{13})$ is then $\frac{1}{585}$ or about 9.19 bits. The maximum error in approximating $1 - 8f(1-f)$ by $1 - 8f_{13}(1-f_{13})$ is $\frac{1}{2048}$, so the result accuracy is over 8.8 bits for the f-factor. We design the output from the primary Table 1 to include both the 1x and 3x values needed for partial product selection in a Booth radix 8 PPG. These values here are $\frac{1}{(146\frac{1}{2})y^2(y_i+2^{-i})^2}$ and $\frac{3}{(146\frac{1}{2})y^2(y_i+2^{-i})^2}$.

In overall performance the Table Fed PPG circuit of Figure 5 provides an approximation to $\frac{1}{y\hat{y}}$ of accuracy nearly 34 bits with a total table size of some 21 Kbytes. The circuit should be implementable in three cycles, allowing for it to remain off the critical path even if the feedback multiplier latency can be reduced to three cycles.

All four of the table lookup procedures illustrated in Figures 2-5 suffer some accumulated error which has not been analyzed in detail herein. Fortunately a clear way to exhaustively check these effects exists with only a modest size computation. The procedure is to compare the lookup schemes with a single table lookup that uses all the bits $b_1 b_2 \cdots b_{i+j}$ as a single index. Referring to Figure 5, this would mean a single table for C_{24}. Such a table provides an approximation to $\frac{1}{y\hat{y}}$ of accuracy 51 bits by Observation 3. Thus the output of the lookup algorithm in all $2^{24} \approx 16$ million cases need simply be compared with the value that would be employed for the 2^{24} outputs of the single table for C_{24} to determine a worst case accumulated error bound. The procedure is simple and efficient enough that the table widths and other design parameters can be fine tuned relying on the modest 16 million or less exhaustive test case computations to confirm the overall desired accuracy for an optimized table lookup circuit.

References

[IEEE 85] *IEEE Standard 754 for Binary Floating Point Arithmetic*, ANSI/IEEE Standard No. 704, American National Standards Institute, Washington DC, 1988.

[CG99] M.A. Cornea-Hasegan, R.A. Golliver, P. Markstein, *Correctness Proofs Outline for Newton-Raphson Based Floating-Point Divide and Square Root Algorithms*, Proc. 14th IEEE Symp. on Comput. Arithmetic, 1999, pp. 96-105.

[DM95] D. DasSarma, D.W. Matula, *Faithful Bipartite ROM Reciprocal Tables*, Proc. 12th IEEE Symp. Comput. Arithmetic, 1995, pp. 17-28.

[DM97] D. DasSarma, D.W. Matula, *Faithful Interpolation in Reciprocal Tables*, Proc. 13th IEEE Symp. Comput. Arithmetic, 1997.

[FS89] D.L. Fowler and J.E. Smith, *An Accurate High Speed Implementation of Division by Reciprocal Approximation*, Proc. 9th IEEE Symp. Comput. Arithmetic, 1989, pp. 60-67.

[IT97] M. Ito, N. Takagi, S. Yajima, *Efficient Initial Approximation for Multiplicative Division and Square Root by a Multiplication with Operand Modification*, IEEE Transactions on Computers, vol. 46, No. 4, April 1997, pp. 495-498.

[Ob99] S.F. Oberman, *Floating Point Division and Square Root Algorithms and Implementation in the AMD-K7 Microprocessor*, Proc. 14th IEEE Symp. on Comput. Arithmetic, 1999, pp. 106-115.

[Pa99] M. Parks, *Number-theoretic Test Generation for Directed Rounding*, Proc. 14th IEEE Symp. on Comput. Arithmetic, 1999, pp. 241- 248.

[WF91] D.C. Wong, M.J. Flynn, *Fast Division Using Accurate Quotient Approximations to Reduce the Number of Iterations*, Proc. 10th IEEE Symp. Comput. Arithmetic, 1991, pp. 191-201.

[EL94] M.D. Ercegovac and T. Lang, *Division and Square Root: Digit-Recurrence Algorithms and Implementations*. Boston: Kluwer Academic, 1994.

[Fl70] M.J. Flynn, "On Division by Functional Iteration", *IEEE Trans. Computers*, vol. 19, no.8, pp. 702-706, Aug 1970. Reprinted in *Computer Arithmetic*, E.E. Swartzlander, vol. 1. Los Alamitos, Calif.: IEEE Press, 1990.

[Go64] R.E. Goldschmidt, "Applications of Division by Convergence", MSc dissertation, MIT, June 1964.

[LM95] C. N. Lyn and D. W. Matula: *Redundant Binary Booth Recoding*, Proc. 12th IEEE Symp. Comput Arithmetic, pp. 50-57, July 1995.

[EI00] M.D, Ercegovac, L. Ilmbert, D.W. Matula, J.-M. Muller, G. Wei, *Improving Goldschmidt Division, Square Root, and Square Root Reciprocal*, IEEE Transactions on Computers, vol. 49, No. 7, July 2000, pp. 759-762.

[IM99] C. Iordache, and D.W. Matula, *Hiding Table Lookup Latency in Convergence Division and Square Root,* submitted for publication.

[Se99] P. Seidel, *High-Speed Redundant Reciprocal Approximation*, INTEGRATION, the VLSI Journal, 28(1999), pp. 1-12.

[Ni81] T. Nishimoto: 'Multiple/Divide Unit,' U.S. Patent 4337519, June 1982.