FAITHFUL BIPARTITE ROM RECIPROCAL TABLES*

Debjit Das Sarma and David W. Matula
Department of Computer Science & Engineering
Southern Methodist University
Dallas, Texas 75275

Abstract

We describe bipartite reciprocal tables that employ separate table lookup of the positive and negative portions of a borrow-save reciprocal value. The fusion of the parts includes a rounding so the output reciprocals are guaranteed correct to a unit in the last place, and typically provide a round-to-nearest reciprocal for over 90% of input arguments. The output rounding can be accomplished in conjunction with multiplier recoding yielding practically no cost in logic complexity or time in employing bipartite tables. We demonstrate these tables to be 2 to 4 times smaller than conventional 4-9 bit reciprocal tables. For 10-16 bit reciprocal table lookup the compression grows from a factor of 4 to over 16, making possible the use of larger seed reciprocals than previously considered cost effective.

1 Introduction and Summary

Current day designs of IEEE standard floating point units for the platforms of workstations and PC's generally have substantial design effort and chip area devoted to providing a multiplier with at most a couple of cycle latency. Newton Raphson, convergence, prescaled, and short reciprocal division are multiplier based iterative division algorithms that can be fine tuned to benefit from selected fast multiplier designs. These algorithms each provide speedups by factors of 2 to 4 over more traditional shift-and-subtract iterative division algorithms. All of these multiplier based division algorithms initially employ a seed reciprocal of the divisor typically provided by a ROM reciprocal table or equivalent PLA [BM 93, DM 94, EL 94, Fe 67, FS 89, Na 87, WF 91]. More bits of accuracy in the seed reciprocals reduce the necessary number of dependent multiply cycles, significantly reducing division time, albiet at the cost of exponential growth in the reciprocal table size.

An important question is then "How can one obtain more accurate seed reciprocals at acceptable costs in time and area ?" Clearly table compression can be effectively obtained by applying interpolation in a small reciprocal table, but this entails the added cost of the multiplication and/or addition to effect the interpolation [Fa 81, Fe 67, Na 87]. For overall efficiency there is a compelling need for compressing reciprocal tables in a manner where table look up remains a very simple direct operation and where predictability of table accuracy is maintained. Our contribution here is to define and show how to construct "bipartite ROM reciprocal tables". These tables implicitly utilize a transparent form of interpolation avoiding any multiplications or additions.

An exacting standard for our bipartite reciprocal table output is obtained in that the reciprocal provided will be guaranteed to be correct to a unit in the last place, *i.e.* less than one ulp deviation from the infinitely precise reciprocal of the infinitely precise input argument. For both the prescale and short reciprocal division algorithms the size of the reciprocal effects the size of the circuitry employing the reciprocal [BM 93, EL 94, Na 87, WF 91]. The one ulp bound provides that the reciprocals are both accurate and short. There are many compelling arguments in the literature for providing that function approximation should satisfy a 1 ulp bound and attempt to maximize the percentage of input arguments that are rounded to nearest [AC 86, BM 93, FB 91].

Our methodology gains practical value from the following observation. The aforementioned division algorithms each apply multiplier recoding to the output reciprocal from the ROM table to provide for subsequent multiplication of the divisor and/or dividend. For this purpose it is sufficient that the reciprocal table provide output in redundant form that may be directly and efficiently subjected to multiplier recoding.

The bipartite ROM reciprocal table provides separate table look up for the positive bit part and negative bit part of a redundant binary reciprocal value. The fusion of positive and negative bits includes a recoding to round off a couple of low order bits to obtain the one ulp precision guarantee. With little extra logic complexity this recoding can convert the redundant binary values -1,0,1 to the Booth recoded radix four digits -2,-1,0,1,2 or radix eight digits -4,-3,-2,-1,0,1,2,3,4. The Booth recoded reciprocal output from this process can be obtained in time only negligibly greater than for the conventional Booth recoding of the output of a traditional ROM reciprocal table [LM 95, TY 87]. Our table compression is thus obtained at almost no cost in logic circuit complexity or cycle time.

To provide standards for measuring compression, we first investigate optimal results in table size minimization and portion round to nearest maximization pertaining to conventional ROM reciprocal tables.

*This work was supported in part by a grant from Cyrix Corporation and by the Texas Advanced Technology Program Grant 003613013.

ROM reciprocal tables are generally constructed by assuming that the argument is normalized $1 \leq x < 2$ and truncated to i bits to the right of the radix point, $trunc(x) = 1.b_1b_2...b_i$. These i bits are used to index a table providing j output bits which are taken as the j bits after the leading bit in the j+1 bit fraction reciprocal approximation $recip(x) = 0.1b'_1b'_2...b'_j$. Such a table is termed an i-bits-in j-bits-out reciprocal table of size $2^i j$ bits. The accuracy of ROM reciprocal tables from the point of view of minimizing relative error was thoroughly investigated by us in [DM 94].

In Section 2 we define as faithful those i-bits-in j-bits-out reciprocal tables for which the output always satisfies a one ulp bound, and as max RN those tables which maximize the portion of input $1 \leq x < 2$ which yields an output reciprocal that is a round-to-nearest value of $\frac{1}{x}$. Our principal result in Section 2 is that for $g \geq 1$ indicating a number of input guard bits, we determine that the $j+g$-bits-in, j-bits-out ROM reciprocal tables generated by the midpoint reciprocal algorithm are both faithful and max RN for any $g \geq 1$ and all $j \geq 3$. The percentage round-to-nearest is further shown to be of the order 88% even with the smallest number $g = 1$ of input guard bits for which one ulp guaranteed output is obtained.

A $j+g$-bits-in, j-bits-out bipartite reciprocal table is constructed by assuming the argument is normalized $1 \leq x < 2$ and truncated to $j+g$ places to the right of the radix point, $trunc(x) = 1.b_1b_2...b_{j+g}$. The bipartite reciprocal table lookup procedure is described by reference to the logic circuitry illustrated in Figure 1. The figure shows $j+2 = 3k+1$ bits of input and a faithful reciprocal of $j = 3k-1$ bits as output represented in redundant binary (or multiplier recoded) format. The lookup steps are:

1. the $j+2 = 3k+1$ input bits are partitioned into high, middle, and low fields of sizes $k+1, k, k$,
2. the leading $2k+1$ ($\sim \frac{2}{3}j$) input bits index a table P giving a $j = 3k+1$ bit positive part of a borrow-save reciprocal value,
3. the $2k+1$ ($\sim \frac{2}{3}j$) bits of the high and low fields index a table N giving a $k+1$ ($\sim \frac{1}{3}j$) bit negative part of a borrow-save reciprocal value,
4. the borrow-save fraction is fused rounding off the two low order bits with multiplier recoding as desired to obtain a $j = 3k-1$ bit reciprocal value guaranteed correct to 1 ulp.

A $j+g$-bits-in, j-bits-out reciprocal table can have alternative partitioning of the input $j+g$ bits and an alternative number of guard bits internally in the table as desired to effect the goal of an output reciprocal accurate to a unit in the last place. The size of the P and N tables ($2^{2k+1} \times (3k+1) + 2^{2k+1} \times (k+1) = 2^{2k+2} \times (2k+1)$ in Figure 1) is the size of the bipartite reciprocal table and is effected by the input part parameterization. Our partitioning attempts to reach the goal of total table size of order $2^{\frac{2}{3}j+O(1)}$ for a faithful output, which produces substantial compression from conventional "non redundant" reciprocal tables of size $2^{j+1}j$ for faithful output.

In Section 3 we note the existance of $j+1$-bits-in, j-bits-out bipartite reciprocal tables which yield output identical to conventional reciprocal tables for the useful values $j = 5,6,7$ with total table size reduced by a factor of about 2. For $j = 8,9$ we obtain bipartite reciprocal tables preserving faithfulness and compressed by factors over 4 compared to the smallest faithful conventional reciprocal tables of Section 2.

Our principal contribution in Section 4 is the description of an algorithmic process for constructing $j+2$-bits-in, j-bits-out faithful bipartite reciprocal tables whose total size grows only as $2^{\frac{2}{3}j+O(1)}$. We present extensive statistics on the exhaustively evaluated bipartite reciprocal tables in comparison to conventional reciprocal tables for the now feasible output sizes $10 \leq j \leq 16$. In particular these bipartite tables are confirmed to be faithful, are shown to be some 4 to 16 times smaller than conventional tables, and demonstrate that over 90% of all input obtains a round-to-nearest output value.

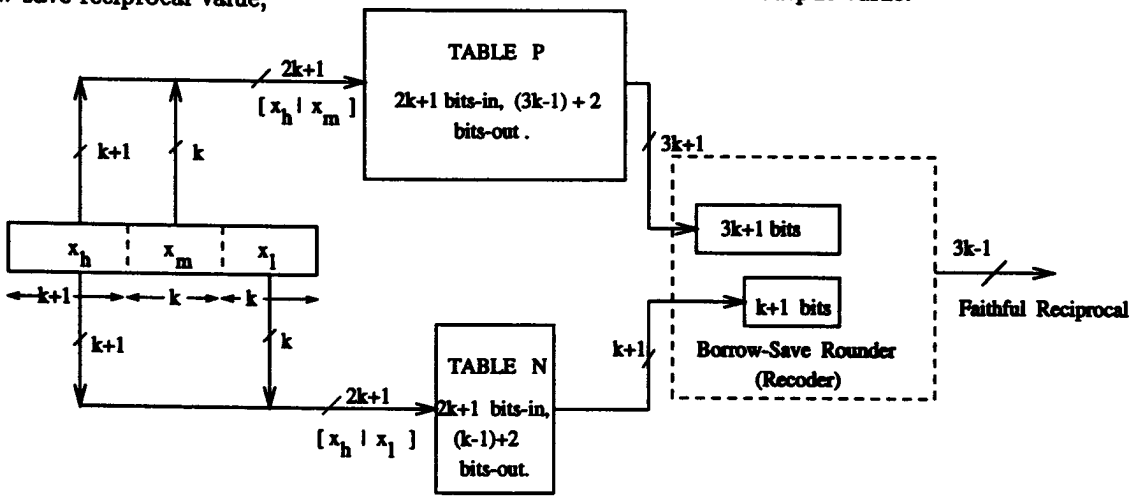

Figure 1: A j+2=3k+1 bits-in j=3k-1 bits out Faithful Bipartite Reciprocal table

Our approach here is both to provide foundations for the methodology and to exhibit reduction to practice. Thus selected reciprocal and bipartite reciprocal tables are included in an appendix. We believe the simplicity of structure of bipartite reciprocal tables and the efficiency of lookup (particularly in combination with multiplier recoding) augers well for their inclusion in the next generation of faster multiplicative based division designs.

2 Faithful Reciprocal Tables

A *faithful* reciprocal table denotes an i-bits-in, j-bits-out reciprocal table such that for any input argument $1 \leq x < 2$, the output differs from $\frac{1}{x}$ by less than 1 ulp. For such tables we allow the value unity as a table output value as well as the 2^j values of the form $0.1b_1b_2...b_j$. Note that for any j-bits-in, j-bits-out reciprocal table with $j \geq 3$, the second smallest input interval $[\frac{(2^j+1)}{2^j}, \frac{(2^j+2)}{2^j})$ has an interval of reciprocal values extending from less than $\frac{(2^{j+1}-3)}{2^{j+1}}$ to greater than $\frac{(2^{j+1}-2)}{2^{j+1}}$. So the table must have a maximum ulp error strictly greater than one ulp.

Observation 1: Any i-bits-in, j-bits-out reciprocal table for $i \leq j$ with $j \geq 3$ has a maximum error in ulps strictly greater than one ulp and therefore is not faithful. □

Thus our search for faithful reciprocal tables must be limited to $j+g$-bits-in, j-bits-out reciprocal tables where $g \geq 1$ is termed the *number of input guard bits*. We are particularly interested in the reciprocal tables generated by the following midpoint reciprocal round-to-nearest procedure [DM 94, Fe 67, FS 89] which we herein term the *midpoint reciprocal algorithm*.

Algorithm 1 [Midpoint Reciprocal Algorithm]

Stimulus: Integers $i \geq 1$ and $j \geq 1$

Response: An i-bits-in j-bits-out reciprocal table

Method: **for** $n = 2^i$ to $2^{i+1} - 1$ **step** 1
<for each input interval $[\frac{n}{2^i}, \frac{n+1}{2^i})$>

begin

L1: $table(n) := RN(\frac{2^{i+j+1}}{n+\frac{1}{2}})$
<rounding is to the nearest integer in ulps>

end

In [DM 94] we proved the midpoint reciprocal algorithm always generates an *optimal (i,j) reciprocal table*, that is, the maximum relative error for each entry in the table is the minimum possible for that entry. Herein an optimal (i, j) reciprocal table is synonomous with the output of the midpoint reciprocal algorithm. We note another significant property (proved in [DM 95]) pertaining to optimal $(j+g, j)$ reciprocal tables.

Theorem 1: For $j \geq 1$, $g \geq 0$, an optimal $(j+g, j)$ reciprocal table is faithful for any $g \geq 1$, and in general has a maximum error for any output strictly less than $\frac{1}{2} + 2^{-g}$ ulps for any $g \geq 0$. □

Corollary: The minimum size of any faithful reciprocal table is $2^{j+1}j$ bits, as occurs for the optimal $(j+1, j)$ reciprocal table. □

Appendix Table A-1 provides considerable information relevant to the optimal 5-bits-in, 4-bits-out reciprocal table that generalizes to observations about $(j+g, j)$ optimal tables. For each input interval in Table A-1 the output error interval is given in ulps, confirming that the (5,4) optimal table is faithful with maximum error 0.970 ulps occuring for input approaching $\frac{33}{32}$ in input interval $[\frac{32}{32}, \frac{33}{32})$. This conforms with the upper bound of 1.00 ulp from Theorem 1.

Maximizing the portion of input rounded to nearest is a useful "tiebreaking" refinement of faithfulness. We herein cite selected results referring the reader to [DM 95] for details and proofs. A *max RN* reciprocal table is an i-bits-in, j-bits-out reciprocal table for which the portion of input values $1 \leq x < 2$ obtaining round-to-nearest output is as large as possible over all i-bits-in, j-bits-out reciprocal tables.

Theorem 2: For any $g \geq 1, j \geq 1$, an optimal $(j+g, j)$ reciprocal table is a max RN table, where the portion not rounded-to-nearest is at most $\frac{1}{2^{g+1}}$. □

Observation 2: To the extent that the non zero round-to-nearest percentages of the max RN table input intervals are uniformly distributed between 0 and 50%, the portion of the input $1 \leq x < 2$ not rounded to nearest for a max RN $(j+g, j)$ reciprocal table should be $\frac{1}{2^{g+2}}$. □

We now investigate bipartite reciprocal tables, with the minimum size and maximum round-to-nearest percentages of faithful $(j+g, j)$ conventional reciprocal tables providing the standards against which to measure the size reduction achieved and overall accuracy sacrificed by compression.

3 Bipartite Reciprocal Tables

An i-bits-in, j-bits-out bipartite reciprocal table is termed *optimal* if it provides identical results to that given by an i-bits-in, j-bits-out optimal ROM reciprocal table. The ratio of the size of the optimal ROM reciprocal table to the size of the optimal bipartite reciprocal table will be termed the *compression factor* for the optimal bipartite reciprocal table. Note that although the size of the bipartite reciprocal table is not fixed by the numbers of input and output bits, the size of the optimal ROM reciprocal table is always $2^i j$ bits, so that the compression factor is simply determined from the parameters giving the component table sizes of the bipartite reciprocal table.

We require that the output of our bipartite reciprocal tables be faithful, that is, guaranteed accurate to a unit in the last place, achieving what many take to be a virtual standard for floating point function accuracy [AC 86, FB 91]. The smallest faithful ROM reciprocal tables as shown in Section 2 are the optimal $(j+1)$-bits-in, j-bits-out tables of size $2^{j+1}j$ bits. We have constructed optimal (hence also faithful) $(j+1)$-bits-in, j-bits-out bipartite reciprocal tables of about half the optimal reciprocal table size for the small but non trivial values j = 5,6,7 [DM 95].

Output Bits j	Input bit Partition of $j+1$	Positive Part Table Dimension	Negative Part Table Dimension	Total Table Size in Bytes	Compression Factor
5	2,2,2	4 in,7 out	4 in,4 out	22	1.82
6	3,2,2	5 in,8 out	5 in,4 out	48	2.00
7	3,3,2	6 in,9 out	5 in,4 out	88	2.55

Table 1: Input bit partition, positive and negative part table dimensions,table sizes, and compression factors for some optimal bipartite reciprocal tables

Table 1 tabulates the input bit partition, positive and negative part table dimensions, size and compression factor for each of these known bipartite reciprocal tables.

Table A-2 of the appendix shows the positive and negative parts of a 6-bits-in, 5-bits-out optimal bipartite reciprocal table. Consider a 6 bit input operand $1.b_5b_4b_3b_2b_1b_0$. The input operand is partitioned into three parts, each having two bits, namely $x_h = b_5b_4$ called the high-order part, $x_m = b_3b_2$ called the middle-order part, and $x_l = b_1b_0$ called the low-order part. The high order part in conjunction with the middle order part $[x_h|x_m]$ index to a 4 bits-in, 7 bits-out positive part table (Table P) and the high order part in conjunction with the low order part $[x_h|x_l]$ index to a 4 bits-in, 4 bits-out negative part table (Table N). Given any 6 bit input operand, the P and N Tables are looked up for the positive and negative parts of the redundant binary reciprocal given in borrow-save form. Two guard bits in each of the P and N outputs are rounded off during fusion (in the recoder) with the result that the output is identical to the optimal ROM reciprocal table output. For convenience of illustration, the outputs of the P and N Tables are each shown with 10 bits, a leading bit before the radix point followed by 9 bits beyond the radix point. However only 7 bits and 4 bits respectively need to be stored in the P and N tables. Observe for the Table P positive part shown, the leading bits denote that the reciprocal must lie between $\frac{1}{2}$ and 1, and the trailing bit shown is added implicitly when P and N values are fused and is not a part of the bits stored in Table P. Similarly for the Table N negative part, the first 4 zero bits beyond the radix point and the trailing 0 bit are shown only for visualizing alignment in fusing the table N value with the corresponding table P value.

It is useful to provide a verification walkthrough to further illustrate the workings of a bipartite reciprocal table. This is provided for the 6-bits-in, 5-bits-out optimal bipartite table shown in Table A-2 by the expansion given in Table A-3. For each 6 bit input, the corresponding infinite precision middle point reciprocal and the optimal reciprocal (rounded middle point reciprocal) are provided. To achieve optimality the P and N table entries are constructed such that the computed reciprocal output for any 6 bit input obtained from the corresponding P and N values is not only close but on the same side of the output value mid point in ulps as the corresponding infinite precision middle point reciprocal. Observe that the full (6,5) bit table can be divided into 4 blocks, the number of blocks determined by the number of bits in the high order part of the input, and each block can be divided into 4 segments, the number of segments determined by the number of bits in the middle order part of the input. Each segment has 4 entries, determined by the number of low order bits in the input. Notice that for each segment there is one Table P entry since the leading 4 bits of the 6 bit input in the segment are the same. Also there are four different Table N entries in each segment corresponding to the four inputs in the segment having different low order bits. Observe that in each block, the same set of four Table N entries occur in each of the four segments. Given a 6 bit input operand, the two high order bits in conjunction with the two middle order bits identify the block and the segment it belongs to, and the corresponding Table P entry is read. Two trailing bits of the input determine the appropriate Table N entry in the segment.

Consider for example line seven of Table A-3. Line seven corresponds to the input 1.000110 shown in Table A-3 as 70 ulps where an ulp here is $\frac{1}{64}$. This input is the third entry of the second segment in the first block. The corresponding Table P is indexed by 1.0001xx and the output is shown in Table A-3 line five to be 59.875 ulps. Table N is indexed by 1.00xx10 and the output is shown in Table A-3 line seven as 1.75 ulps. The computed value of the reciprocal is obtained as (59.875-1.75) = 58.125 ulps and then finally rounded to nearest to yield 58 ulps as the 5 bit reciprocal of the 6 bit input $\frac{70}{64}$. Observe from Table A-3 that for any input argument the rounded computed reciprocal is identical to the rounded midpoint reciprocal from the conventional 64 entry table. Thus this 6-bit-in, 5-bits-out bipartite reciprocal table with (4,7) and (4,4) tables for positive and negative parts is optimal (hence faithful).

Tables A-3 incorporates the simplification that the positive and negative parts need not be subjected to a carry completion addition, but rather can go directly to a multiplier (Booth) recoder accepting borrow-save form with the recoding including the rounding [LM 95, TY 87]. This is significant since we essentially obtain compression paying very little penalty in hardware complexity or cycle time when the reciprocal is to be employed as a multiplier, as in the fast division algorithms previously cited.

In general for higher values of j it is not possible to

construct $(j+1)$-bits-in, j-bits-out optimal bipartite reciprocal tables where the index size of both tables is only about $\frac{2}{3}j$. For larger values of j where a greater compression factor is sought we shall shift our focus to construction of bipartite reciprocal tables that are still guaranteed correct to a unit in the last place even though they might not match identically the optimal $(j+1, j)$ reciprocal tables.

Note that if an input interval of arguments $[\frac{n}{2^j}, \frac{(n+1)}{2^j})$ is such that the interval of reciprocals falls strictly between the two output values of m ulps and $m+1$ ulps, then either choice is an acceptable faithful reciprocal. The optimal choice is always the unique one that maximizes the portion of the interval that rounds to nearest, even though the split may be near half and half. When generating a bipartite reciprocal table accurate to a unit in the last place, the extent to which it differs from the corresponding optimal ROM reciprocal table may be usefully measured by comparing either or both the maximum ulp error of each table and the portions of input that realize round-to-nearest lookup in each table. These comparative metrics will be emphasized in the balance of this paper.

An i-bits-in, j-bits-out bipartite reciprocal table is termed *faithful* when the i bit input corrresponding to any input argument $1 \leq x < 2$ provides a j bit output that differs by less than one ulp from the infinitely precise value $\frac{1}{x}$. The *compression factor* for any faithful bipartite reciprocal table is given by $2^{j+1}j$ divided by the size of the i-bits-in, j-bits-out faithful bipartite reciprocal table. Note that this latter compression factor is not for exclusively lossless compression, but is particular for compression preserving the 1 ulp bound. Observe also that the compression factor is defined in comparison to the size of the $(j+1)$-bits-in, j-bits-out optimal reciprocal table which was shown in Section 2 to be the smallest ROM reciprocal table satisfying the 1 ulp bound. It is important to note that the notion of faithful compression allows for greater possibilities in reducing table size than would pertain to the requirement of lossless compression. We have constructed faithful bipartite reciprocal tables for the practically useful sizes 9-bits-in, 8-bits-out and 10-bits-in, 9-bits-out which both attain compression factors of better than 4 to 1. Table 2 tabulates their input bit partition, positive and negative part table dimension, size and compression factor for faithful compression. The 9-bits-in, 8-bits-out table is Appendix Table A-4.

Table A-4 may be exhaustively developed to verify faithfulness in a manner comparable to the enumeration in Table A-3 . It is instructive to consider a typical line of such an enumeration where the faithful bipartite table differs from the optimal reciprocal table, as for example for the input $1.000100001 = \frac{545}{512}$ shown in Table A-5. Note that in Table A-5 the successive low end reciprocals $\frac{480.998}{512}$ and $\frac{480.117}{512}$ indicate that any point in the input interval having input index 1.000100001 may have its reciprocal represented with less than one ulp error by either $\frac{480}{512}$ or $\frac{481}{512}$. The optimal table choice is $\frac{481}{512}$ with maximum ulp error for this interval of 0.883 ulp where 56.5% of input values from this interval will obtain a round-to-nearest table lookup reciprocal value. Alternately our faithful bipartite table choice is $\frac{480}{512}$ with maximum ulp error for this interval of 0.998 ulp where the other complimentary 43.5% of the input values from this input interval will obtain a round-to-nearest table lookup reciprocal value.

During the exhaustive enumeration these statistics can be accumulated, here leading to the following results. Both faithful bipartite and conventional optimal tables realize the same worst case ulp difference of 0.99805 realized for the input 1.0000000001. The more useful metric here is that the faithful bipartite table realizes round-to-nearest output for some 82% of all input over $1 \leq x < 2$, as compared to some 88% for the optimal reciprocal table. It can be argued that the bipartite table's slightly poorer average case performance with equal worst case rounding error in ulps is an acceptable price to pay for a more than 4 to 1 compression in table size.

A perhaps more telling comparison is found between the enhanced accuracy documented in the 120 byte 9-bits-in,8-bits-out faithful bipartite reciprocal table in comparison to the nearly equivalent size 128 byte 7-bits-in,8-bits-out optimal reciprocal table. While the latter achieves a maximum relative error bound of 7.775 bits [DM 94], it provides a faithful 8 bit result for only 85.82% of all input compared to 100% for the bipartite table, and it provides a round-to-nearest result for only 52.33% of all input compared to some 82% for the faithful bipartite table.

The $(j+1, j)$ faithful bipartite reciprocal tables we constructed relied on avoiding compounded worst case features which was tractable for less than 10 output bits. The construction of further such $(j+1)$-bits-in, j-bits-out faithful bipartite reciprocal tables with near $\frac{2}{3}j$ table index sizes becomes unmanageable and in certain cases impossible for $j \geq 10$. For faithful bipartite tables of 10 or more output bits we proceed instead to the algorithmic generation of $(j+2)$-bits-in, j-bits-out faithful bipartite tables.

Output Bits j	Input bit Partition of $j+1$	Positive Part Table Dimension	Negative Part Table Dimension	Total Table Size in Bytes	Compression Factor
8	3,3,3	6 in,10 out	6 in,5 out	120	4.27
9	4,3,3	7 in,11 out	7 in,5 out	256	4.50

Table 2: Parameters for certain faithful bipartite reciprocal tables

4 Algorithmic Construction of Faithful Bipartite Reciprocal Tables

In this section we present a general algorithm for constructing $j+2$-bits-in, j-bits-out faithful bipartite reciprocal tables. The parameterization of the positive and negative part tables separates into three cases based on the value of j modulo 3. It is convenient to introduce the parameter $k = \lceil \frac{j}{3} \rceil$ and view the cases cyclicly for the three values $j = 3k-2, 3k-1, 3k$.

The reasons for the parameterization will become evident from the proof of the one ulp table bound pertaining to each of these cases. Suffice it to notice here that the positive part tables increase in size with j periodically by factors of approximately 2, 2, and 1, while the negative part tables remain the same size for three consecutive values of j and then jump by a factor somewhat over four. The combined effect is that asymptotically in j the total table size grows periodically by factors $\frac{8}{5} = 1.60$, $\frac{7}{4} = 1.75$, and $\frac{10}{7} = 1.43$ for a rate asymptotically averaging $2^{\frac{2}{3}} = 1.587$. Thus generating 3 more bits with 1 ulp accuracy is obtained at a cost of a factor 4 growth in bipartite reciprocal table size as compared to the larger factor 8 growth in size of a conventional optimal reciprocal table. Figure 1 showed the input bit partitions, indexing schemes, and dimensions of P and N tables to construct a $3k+1$-bits-in, $3k-1$-bits-out bipartite reciprocal table.

To motivate our method for constructing the Table P positive part and Table N negative part of a bipartite ROM reciprocal table, we inspect selected portions of the 10-bits-in, 8-bits-out low and middle point reciprocal table shown in Table A-6. For our bipartite table the 10 input bits are split into 4, 3, and 3 bits as per the parameterization corresponding to $k = \lceil \frac{8}{3} \rceil = 3$ and $j = 8 = 3\times3\text{-}1 = 3k-1$ in Table 3. The leading 7 bits index to the desired Table P, and the high order 4 bits in conjunction with low order 3 bits index into Table N. To reflect this structure the full 10 bits in, 8 bits out reciprocal table will be partitioned into $2^{k+1} = 16$ blocks, each block containing $2^k = 8$ segments, and each segment containing $2^k = 8$ entries. In Table A-6, selected inputs relevant to constructions for the first block are shown, including all 8 entries in the first and last segment and the first and last entries of each of the 6 other segments. For each of the 10 bit inputs shown, the end point reciprocal and middle point reciprocal are each given to three decimal fraction digits of ulps, implicitly indicating the analytic infinitely precise values for this part of the computation. Notice that for each segment in Table A-6 we need to determine one Table P entry, and for each block set eight Table N entries which are repeated for each of the segments in the block. Clearly for any method of computing the Table P and Table N entries, the construction of these entries for different blocks are independent of each other since the high order four input bits are different for each block. It is representative to show the computation of P and N entries in the first block.

We will first show analytically how to compute the infinite precision values providing the basis for the P and N entries. From hereon the middle point reciprocal, and the P and N table entries are considered to be infinite precision values unless mentioned otherwise. Selected middle point reciprocals of the (10,8) reciprocal table are used to construct the bipartite reciprocal table. The entries in the bipartite table are selected such that the computed reciprocal of each input (the difference of presumed infinitely precise P and N entries) differs by no more than $\frac{1}{8}$ ulp from the corresponding middle point reciprocal, such resulting differences being enumerated for reference in Table A-6. Observe further from Table A-6 that each middle point reciprocal differs from each of the corresponding interval end point reciprocals by at most $\frac{1}{4}$ ulps, this amount decreasing monotonically down the table from the initial value 0.250 ulps to 0.222 ulps at the end of the first block. This yields that the middle point reciprocal in the table differs from any input argument reciprocal $\frac{1}{x}$ by at most $\frac{1}{4}$ ulp, so that the computed reciprocal as determined from the P and N entries illustrated in Table A-6 differs from $\frac{1}{x}$ by at most $\frac{1}{4}+\frac{1}{8} = \frac{3}{8}$ ulps. This guarantee of at most $\frac{3}{8}$ ulp error in the analytic computed reciprocal is instrumental in insuring that the subsequent compound roundings producing the P and N table values and output look up value achieve a total error bound strictly less than 1 ulp.

We define the *spread* of a segment to be the difference between the middle point reciprocals of the first and last inputs in the segment. For the first segment in Table A-6 the spread is 511.750 - 508.277 = 3.473 ulps, and the spreads of the next seven segments are 3.419 ulps, 3.367 ulps, 3.316 ulps, 3.266 ulps, 3.217 ulps, 3.170 ulps, and 3.123 ulps. Figure 2 shows the reciprocal curve for a block partitioned into the four segments. Notice that the slope of the curve decreases monotonically from the lower segments to the higher segments. The reciprocal curves for the different segments are then overlayed to show the relative difference of the slopes and the spreads of the segments.

Output Bits j	Input Bit Partition of $j+2$	Positive Part Table Dimension	Negative Part Table Dimension	Total Table Size in Bits
$3k-2$	$k+1, k-1, k$	$2k$ in,$3k$ out	$2k+1$ in,$k+1$ out	$2^{2k}(5k+2)$
$3k-1$	$k+1, k, k$	$2k+1$ in,$3k+1$ out	$2k+1$ in,$k+1$ out	$2^{2k}(8k+4)$
$3k$	$k+1, k+1, k$	$2k+2$ in,$3k+2$ out	$2k+1$ in,$k+1$ out	$2^{2k}(14k+10)$

Table 3: Parameterization for the faithful bipartite reciprocal tables to be constructed by our general algorithm

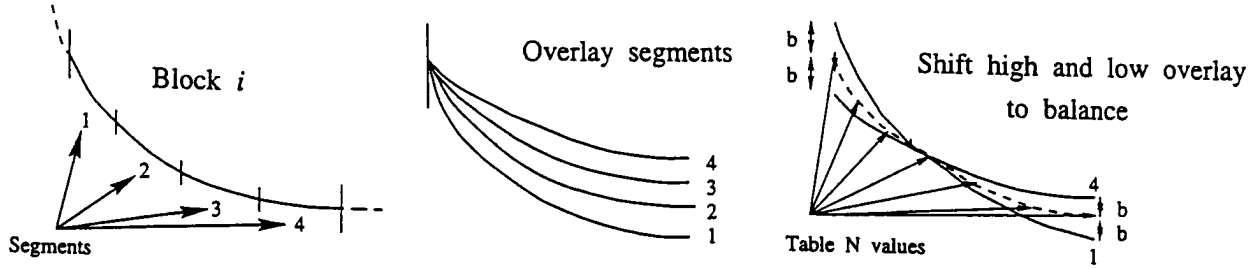

Figure 2: Algorithmic generation of table P and N entries of a bipartite table

To obtain a tight fit of the curve resulting from the Table P and N values, the spreads and the slopes of the first segment and the last segment of a block are then averaged to generate the dotted curve shown in the figure. The low (first segment) and the high (last segment) overlays of the block are shifted down and up respectively by an amount b such that the spread of the dotted curve is the average of the spreads of the first and the last segments. The other overlays (not shown in the figure for clarity) are shifted accordingly. Thus the first entry of each segment is adjusted accordingly to compute the Table P entries to cover the spread. Note that in Table A-6, the computed spread is 3.298 ulps which is the average of the spreads of the first and last segments. Also note that the middle point reciprocal of the first entry of the first segment is pushed down by $\frac{(3.473-3.123)}{2}$ ulps *i.e.* 0.175 ulps to obtain the Table P entry of the first segment. Similarly the middle point reciprocal of the first entry of the last segment is pushed up by 0.175 ulp to obtain the Table P entry of the last segment. To compute the remaining Table P entries as given in Algorithm 3, we note that corresponding middle point reciprocals need to be adjusted by 0.087 ulps, 0.061 ulps, 0.035 ulps, 0.009 ulps, 0.016 ulps, 0.040 ulps, 0.064 ulps, and 0.087 ulps and the Table P entries of the eight segments are computed as $\frac{511.663}{512}$, $\frac{507.724}{512}$, $\frac{503.846}{512}$, $\frac{500.027}{512}$, $\frac{496.266}{512}$, $\frac{492.561}{512}$, $\frac{488.911}{512}$, and $\frac{485.315}{512}$.

Also the value of the dotted curve in Figure 2 at each point is the average of the slopes of the high and low overlays at that point. The value of the dotted curve at each point is used to compute the Table N entries of the block. Thus the successive differences of the entries of the first and last segments of the first block shown in Table A-6 are averaged to compute the Table N entries. This centers the errors across each segment and the errors of the segments across each block. The first Table N entry of each segment is zero. For $i \geq 1$, the i^{th} Table N entry of the block is computed as the average difference between the middle point reciprocals of the first input and the i^{th} input of the first and the last segments. In Table A-6 for the first block, the first Table N entry is 0, the second Table N entry is $\frac{(511.75-511.25)+(485.227-484.779)}{2}$ ulps which equals 0.474 ulps, the third Table N entry is $\frac{(511.75-510.753)+(485.227-484.331)}{2}$ ulps which equals 0.974 ulps, and likewise the remaining five Table N entries are 1.419 ulps, 1.890 ulps, 2.360 ulps, 2.830 ulps, and 3.298 ulps.

With these eight Table P entries, and eight Table N entries, the reciprocals of the first sixty-four 10 bit inputs are computed and shown in the sixth column of Table A-6. The last column of Table A-6 shows the signed differences in ulps between the computed reciprocals and the corresponding middle point reciprocals of each 10 bit input in the first block. Notice that these differences in each segment are symmetric and centered around zero, and their magnitudes are maximum for the first and last entries of the segment. Also note that the maximum difference in each block is well centered around zero and is realized in the first and last segments of the block. It is important however to note that these Table P and Table N values shown in Table A-6 are infinite precision values and are each subjected to rounding before being stored in the bipartite table. Table P entries are rounded down to 10 bits, the two trailing bits being the guard bits. The Table N entry is rounded to nearest to 4 bits. Notice that the value of a spread of any segment in Table A-6 is strictly less than 4 ulps, so only 2 bits are needed to the left of the radix point in ulps to cover any spread, and so only 4 bits need to be stored in Table N, the two trailing bits being the guard bits.

For the first block of the (10,8) bipartite table in Table A-6 the table P entries are obtained by rounding the infinitely precise values down to the closest quarter and adding $\frac{1}{8}$, effectively rounding to the nearest odd eighth. The resulting values with the $\frac{1}{8}$ included are then 511.625, 507.625, 503.875, 500.125, 496.375, 492.625, 488.875, and 485.375. The table N entries are obtained by rounding to the nearest quarter yielding 0, 0.50, 1.00, 1.50, 2.00, 2.25, 2.75, and 3.25. Note that since the computed prerounded reciprocal obtained by fusing the P and N parts will have an odd number of eighth's of ulps, the final rounding adds at most $\frac{3}{8}$ ulp additional error. The compounded worst case additional error due to these discretizing roundings is then $\frac{5}{8}$ ulps. In Table A-6, the rounded P and N table values for the input $\frac{1027}{1024}$ in line four are $\frac{511.50}{1024}$, and $\frac{1.5}{1024}$ which constitute the positive part and the negative part of the redundant reciprocal in borrow-save form. The final non-redundant 8 bit reciprocal of the 10 bit input $\frac{1027}{1024}$ is computed by fusing the positive part ($\frac{511.625}{1024}$) with the negative part $\frac{1.5}{1024}$ and rounding to nearest to the nearest ulp yielding 8 bits which yields $\frac{510}{1024}$ matching the optimal reciprocal for this input operand. For any 10 bit input operand in

the (10,8) bipartite table, the computed reciprocal is guaranteed to be faithful.

We now describe the algorithm formally to compute the P and N tables for the positive and negative parts of the bipartite ROM table to generate $3k-1+u$ bit faithful reciprocals of $3k+1+u$ bit input operands for $u = 0$, 1, and -1. The $3k+1+u$ bit input operand $1.b_{3k+u}b_{3k+u-1}...b_3b_2b_1b_0$ is partitioned into three parts, $x_h = b_{3k+u}b_{3k-1+u}...b_{2k+u}$ is the $k+1$ bit high order part, $x_m = b_{2k+u-1}b_{2k+u-2}...b_k$ is the $k+u$ bit middle order part, and $x_l = b_{k-1}b_{k-2}...b_0$ is the k bit low order part. The three parts, x_h, x_m, and x_l can be conveniently represented by a radix $k+1$ digit, a radix $k+u$ digit, and a radix k digit respectively. Thus any input operand x is encoded as $[x_h|x_m|x_l]$ whose value is $1 + 2^{-(k+1)}x_h + 2^{-(k+u)}x_m + 2^{-k}x_l$, and clearly an input operand $[l|m|n]$ is the $n-1^{th}$ entry of the $m-1^{th}$ segment in the $l-1^{th}$ block of the $3k+1+u$-bits-in, $3k-1+u$-bits-out ROM reciprocal table.

The high order part in conjunction with the middle order part, $[x_h|x_m]$ index to a $2k+1+u$ bits-in, $3k+1+u$ bits-out table called Table P and the high order part in conjunction with the low order part, $[x_h|x_l]$ index to a $2k+1$ bits-in, $k+1$ bits-out table called Table N. We define $recip_{mid}([x_h|x_m|x_l])$ as the infinite precision middle point reciprocal of the $3k+1+u$ bit input x.

Algorithm 2 [Bipartite Reciprocal Table Construction]

Stimulus: Integers $k \geq 2$, and $u = 0, 1, -1$

Response: $2k+1+u$-bits-in $3k+1+u$-bits-out table P, and $2k+1$-bits-in $k+1$-bits-out table N

Method:

Step 1 [Construction of Table P]
for $x_h = 0$ to $2^{k+1}-1$ step 1
for $x_m = 0$ to $2^{k+u}-1$ step 1
<for each segment in each block>
begin

L1: firstspread(x_h) = $recip_{mid}([x_h|0|0]) - recip_{mid}([x_h|0|2^k-1])$

L2: lastspread(x_h) = $recip_{mid}([x_h|2^{k+u}-1|0]) - recip_{mid}([x_h|2^{k+u}-1|2^k-1])$

L3: averagespread(x_h) = $\frac{(firstspread(x_h)+lastspread(x_h))}{2}$

L4: spread(x_h, x_m) = $recip_{mid}([x_h|x_m|0]) - recip_{mid}([x_h|x_m|2^k-1])$
<compute the spread of the segment>

L5: adjust(x_h, x_m) = $\frac{averagespread(x_h)-spread(x_h,x_m)}{2}$
<compute the adjustment>

L6: P(x_h, x_m) = $recip_{mid}([x_h|x_m|0]) + adjust(x_h, x_m)$

L7: Round down P(x_h, x_m) to $3k+1+u$ bits
end

Step 2 [Construction of Table N]
for $x_h = 0$ to $2^{k+1}-1$ step 1
for $x_l = 0$ to 2^k-1 step 1
<for each block construct 2^k Table N entries>
begin

L8: firstdiff(x_h, x_l) = $recip_{mid}([x_h|0|0]) - recip_{mid}([x_h|0|x_l])$

L9: lastdiff(x_h, x_l) = $recip_{mid}([x_h|2^{k+u}-1|0]) - recip_{mid}([x_h|2^{k+u}-1|x_l])$

L10: N(x_h, x_l) = $\frac{firstdiff(x_h,x_l)+lastdiff(x_h,x_l)}{2}$

L11: Round to nearest N(x_h, x_l) to $k+1$ bits
end

Theorem 3: For any $j \geq 6$, Algorithm 2 generates a faithful $(j+2,j)$ bipartite reciprocal table.

Proof: (see DM 95]) □

We apply Algorithm 2 to construct $j+2$-bits-in, j-bits-out bipartite reciprocal tables for $10 \leq j \leq 16$. Table 4 shows the ROM table size in Kbytes $= 2^{13}$ bits, percentage of inputs not RN, and maximum absolute error (in ulps) of our $(j+2,j)$ bipartite tables and that of $(j+1,j)$, and $(j+2,j)$ optimal tables for $10 \leq j \leq 16$. Observe that the $(j+1,j)$ optimal table is the minimum sized table to guarantee faithful reciprocals. So the $(j+2,j)$ biparite table size is compared to the $(j+1,j)$ optimal table size to obtain the proper measure of faithful compression. Table 4 shows that while the compression factor ranges from about four to sixteen, the $(j+2,j)$ biparite table consistently outperforms the $(j+1,j)$ optimal table in percentage of inputs round-to-nearest, and maximum absolute error incurred. The percentage of inputs round-to-nearest in the $(j+2,j)$ biparite table is about 91.5% compared to about 87.5% in the $(j+1,j)$ optimal table. This suggests that even though a $(j+2,j)$ bipartite table is not guaranteed optimal, the percentage of the input arguments for which the output of the bipartite table is the round to nearest value is nearly as high as possible as given by the optimal $(j+2,j)$ table. Table 4 also shows that while the maximum absolute error (in ulps) incurred by the $(j+2,j)$ biparite table grows from 0.826 ulps to 0.919 ulps approaching the upper bound of 1 ulp for large j, it is still better than the worst absolute error suffered in the $(j+1,j)$ optimal table which is very close to 1 ulp for any j, realized at the very first entry of the table.

For iterative refinement division methods such as Newton Raphson or convergence division, reducing the maximum relative error in the initial approximation to the reciprocal can be a bigger concern than the 1 ulp accuracy guaranteed after the final rounding. For those applications, we can measure the maximum relative error in a biparite table using the prerounded borrow-save value and compare it with that of the optimal table where the maximum relative error is known to be minimized. The precision in bits of a table is the negative base two logarithm of the maximum relative

j	$j+1$-bits-in,j-bits-out Optimal ROM table			$j+2$-bits-in,j-bits- out Optimal ROM table			$j+2$-bits-in,j-bits-out Bipartite table		
	Table size (Kbytes)	Percent not RN	Max error (ulps)	Table size (Kbytes)	Percent not RN	Max error (ulps)	Table size (Kbytes)	Percent not RN	Max error (ulps)
10	2.5	12.453	0.999	5	6.259	0.722	0.6875	8.628	0.826
11	5.5	12.710	≈ 1	11	6.126	0.736	1.125	8.514	0.857
12	12	12.694	≈ 1	24	6.103	0.743	2.0625	8.438	0.853
13	26	12.511	≈ 1	52	6.217	0.746	3.375	8.638	0.865
14	56	12.501	≈ 1	112	6.248	0.748	5.5	8.616	0.901
15	120	12.455	≈ 1	240	6.228	0.747	10	8.578	0.904
16	256	12.522	≈ 1	512	6.259	0.748	16	8.677	0.919

Table 4: Table sizes, percent not RN, and maximum absolute error in ulps of optimal tables and bipartite tables of different sizes

error realized in the table. A table precision of α bits (with α not necessarily an integer) then denotes that the approximation of $\frac{1}{x}$ by the table value yields a relative error of at most $\frac{1}{2^\alpha}$. The precision of the $j+2$ bit unrounded borrow-save values of the $(j+2, j)$ bipartite table, and the precisions of optimal $(j+2, j+2)$ and optimal $(j+1, j+2)$ tables for j = 10,12,14,16 are shown in Table 5. Observe that the minimum precision of the $j+2$ bit unrounded borrow-save values of $(j+2, j)$ bipartite table and that of the optimal $(j+1, j+2)$ table are about the same, with the precision of the optimal $(j+2, j+2)$ table larger by about 0.7 bits. So the bipartite table compares most reasonably with the optimal $(j+1, j+2)$ table in terms of precision, for which the slightly larger compression factors ranging from 4 to 18 as j goes from 10 to 16 are obtained.

j	10	12	14	16
$(j+2, j+2)$ optimal table	12.428	14.423	16.418	18.417
$(j+1, j+2)$ optimal table	11.701	13.687	15.683	17.680
$j+2$ bit unrounded	11.744	13.678	15.678	17.634

Table 5: Precision (in bits) of optimal tables and unrounded borrow-save values of bipartite table values of different sizes

References

[AC 86] R. C. Agarwal, J. W. Cooley et al, "New Scaler and Vector Elementary Functions for the IBM/370", in *IBM J. Res. and Develop.* , Vol. 30, No. 2, 1986, pp 126-144.

[BM 93] W. B. Briggs and D. W. Matula, "A 17×69 Bit Multiply and Add Unit with Redundant Binary Feedback and Single Cycle Latency" in *Proc.* 11^{th} *IEEE Symp. Comput. Arithmetic* , 1993, pp 163-170.

[DM 94] D. Das Sarma and D. W. Matula, "Measuring the Accuracy of ROM Reciprocal Tables" in *IEEE Trans. Comput.*, Vol. 43, No. 8, 1994, pp 932-940.

[DM 95] D. Das Sarma and D. W. Matula, "Faithful Bipartite ROM Reciprocal Tables", Technical Report, Computer Science Department, Southern Methodist University, May 1995.

[EL 94] M.D. Ercegovac, T. Lang and P. Montuschi, "Very High Radix Division with Selection by Rounding and Prescaling", in *IEEE Trans. Comput.* , Vol. 43, No. 8, 1994, pp 909-918.

[Fa 81] P. M. Farmwald, "On the Design of High Performance Digital Arithmetic Units," in *Ph. D. thesis, Stanford University*, 1981.

[FB 91] W. E. Ferguson and T. Brightman, "Accurate and Monotone Approximations of Some Transcendental Functions", in *Proc.* 10^{th} *IEEE Symp. Comput. Arithmetic*, 1991, pp 237-244.

[Fe 67] D. Ferrari, "A Division Method Using a Parallel Multiplier", in *IEEE Trans. Electron. Comput.* , 1967, EC-16, pp 224-226.

[FS 89] D. L. Fowler and J. E. Smith, "An Accurate High Speed Implementation of Division by Reciprocal Approximation", in *Proc.* 9^{th} *IEEE Symp. Comput. Arithmetic*, 1989, pp 60-67.

[LM 95] A. Lyu and D. W. Matula, "Efficient Multiplier Recoding of Redundant Binary Values", in *Proc.* 12^{th} *IEEE Symp. Comput. Arithmetic*, 1995.

[Na 87] H. Nakano, "Method And Apparatus For Division Using Interpolation Approximation", in *United States Patent*, No. 4,707,798, 1987.

[TY 87] N. Tagaki and S. Yajima, "On a Fast Iterative Multiplication Method by Recoding Intermediate Product," in *Proc.* 36^{th} *National Convention of Information Science*, Kyoto University, Aug. 1987.

[WF 91] D. C. Wong and M. J. Flynn, "Fast Division Using Accurate Quotient Approximations to Reduce the Number of Iterations", in *Proc.* 10^{th} *IEEE Symp. Comput. Arithmetic*, 1991, pp 191-201.

Appendix

Chopped Input	Input Interval	Middle point Reciprocal ×(1/32)	Rounded Reciprocal	Output Interval ×(1/32)	Error Interval in ulps	Percentage Not RN
1.00000	[32/32,33/32)	31.508	32	(31.030,32.000]	(-0.970,0.000]	49.21
1.00001	[33/32,34/32)	30.567	31	(30.118,31.030]	(-0.882,0.030]	42.62
1.00010	[34/32,35/32)	29.681	30	(29.257,30.118]	(-0.743,0.118]	28.81
1.00011	[35/32,36/32)	28.845	29	(28.444,29.257]	(-0.556,0.257]	7.02
1.00100	[36/32,37/32)	28.055	28	(27.676,28.444]	(-0.324,0.444]	0.00
1.00101	[37/32,38/32)	27.307	27	(26.947,27.676]	(-0.053,0.676]	23.64
1.00110	[38/32,39/32)	26.597	27	(26.256,26.947]	(-0.744,-0.053]	35.85
1.00111	[39/32,40/32)	25.924	26	(25.600,26.256]	(-0.400,0.256]	0.00
1.01000	[40/32,41/32)	25.284	25	(24.976,25.600]	(-0.024,0.600]	15.69
1.01001	[41/32,42/32)	24.675	25	(24.381,24.976]	(-0.619,-0.024]	20.41
1.01010	[42/32,43/32)	24.094	24	(23.814,24.381]	(-0.186,0.381]	0.00
1.01011	[43/32,44/32)	23.540	24	(23.273,23.814]	(-0.727,-0.186]	42.55
1.01100	[44/32,45/32)	23.011	23	(22.756,23.273]	(-0.244,0.273]	0.00
1.01101	[45/32,46/32)	22.505	23	(22.261,22.756]	(-0.739,-0.244]	48.89
1.01110	[46/32,47/32)	22.022	22	(21.787,22.261]	(-0.213,0.261]	0.00
1.01111	[47/32,48/32)	21.558	22	(21.333,21.787]	(-0.667,-0.213]	37.21
1.10000	[48/32,49/32)	21.113	21	(20.898,21.333]	(-0.102,0.333]	0.00
1.10001	[49/32,50/32)	20.687	21	(20.480,20.898]	(-0.520,-0.102]	4.88
1.10010	[50/32,51/32)	20.277	20	(20.078,20.480]	(0.078,0.480]	0.00
1.10011	[51/32,52/32)	19.883	20	(19.692,20.078]	(-0.308,0.078]	0.00
1.10100	[52/32,53/32)	19.505	20	(19.321,19.692]	(-0.679,-0.308]	48.72
1.10101	[53/32,54/32)	19.140	19	(18.963,19.321]	(-0.037,0.321]	0.00
1.10110	[54/32,55/32)	8.789	19	(18.618,18.963]	(-0.382,-0.037]	0.00
1.10111	[55/32,56/32)	18.450	18	(18.286,18.618]	(0.286,0.618]	35.14
1.11000	[56/32,57/32)	18.124	18	(17.965,18.286]	(-0.035,0.286]	0.00
1.11001	[57/32,58/32)	17.809	18	(17.655,17.965]	(-0.345,-0.035]	0.00
1.11010	[58/32,59/32)	17.504	18	(17.356,17.655]	(-0.644,-0.345]	48.57
1.11011	[59/32,60/32)	17.210	17	(17.067,17.356]	(0.067,0.356]	0.00
1.11100	[60/32,61/32)	16.926	17	(16.787,17.067]	(-0.213,0.067]	0.00
1.11101	[61/32,62/32)	16.650	17	(16.516,16.787]	(-0.484,-0.213]	0.00
1.11110	[62/32,63/32)	16.384	16	(16.254,16.516]	(0.254,0.516]	6.06
1.11111	[63/32,64/32)	16.126	16	(16.000,16.254]	(0.000,0.254]	0.00

Table A-1: Optimal 5 bits in,4 bits out Reciprocal Table

Input Index Bits	Positive Part
1.00 00 xx	0.1 1111110 1
1.00 01 xx	0.1 1101111 1
1.00 10 xx	0.1 1100001 1
1.00 11 xx	0.1 1011000 1
1.01 00 xx	0.1 1001010 1
1.01 01 xx	0.1 1000000 1
1.01 10 xx	0.1 0111000 1
1.01 11 xx	0.1 0110000 1
1.10 00 xx	0.1 0101001 1
1.10 01 xx	0.1 0100011 1
1.10 10 xx	0.1 0011100 1
1.10 11 xx	0.1 0010111 1
1.11 00 xx	0.1 0010001 1
1.11 01 xx	0.1 0001101 1
1.11 10 xx	0.1 0001000 1
1.11 11 xx	0.1 0000100 1

Input Index Bits	Negative Part
1.00 xx 00	0.0000 0000 0
1.00 xx 01	0.0000 0011 0
1.00 xx 10	0.0000 0111 0
1.00 xx 11	0.0000 1010 0
1.01 xx 00	0.0000 0000 0
1.01 xx 01	0.0000 0010 0
1.01 xx 10	0.0000 0101 0
1.01 xx 11	0.0000 0111 0
1.10 xx 00	0.0000 0000 0
1.10 xx 01	0.0000 0010 0
1.10 xx 10	0.0000 0011 0
1.10 xx 11	0.0000 0101 0
1.11 xx 00	0.0000 0000 0
1.11 xx 01	0.0000 0010 0
1.11 xx 10	0.0000 0011 0
1.11 xx 11	0.0000 0100 0

Table A-2: (4,7) positive part and (4,4) negative part ROM tables composing the borrow-save result of a 6-bits-in,5-bits-out optimal bipartite reciprocal table

Chopped Input ×(1/64)	Low end Reciprocal ×(1/64)	Middle point Reciprocal ×(1/64)	Rounded Reciprocal ×(1/64)	Table P Value+$\frac{1}{8}$ ×(1/64)	Table N Value ×(1/64)	Computed Reciprocal ×(1/64)	Rounded Comp. Recip. ×(1/64)
64	64.000	63.504	64	**63.625**	0.00	63.625	64
65	63.015	62.534	63		0.75	62.875	63
66	62.061	61.594	62		1.75	61.875	62
67	61.134	60.681	61		2.50	61.125	61
68	60.235	59.796	60	**59.875**	0.00	59.875	60
69	59.362	58.935	59		0.75	59.125	59
70	58.514	58.099	58		1.75	58.125	58
71	57.690	57.287	57		2.50	57.375	57
72	56.889	56.497	56	**56.375**	0.00	56.375	56
73	56.110	55.728	56		0.75	55.625	56
74	55.351	54.980	55		1.75	54.625	55
75	54.613	54.252	54		2.50	53.875	54
76	53.895	53.542	54	**54.125**	0.00	54.125	54
77	53.195	52.852	53		0.75	53.375	53
78	52.513	52.178	52		1.75	52.375	52
79	51.848	51.522	52		2.50	51.625	52
80	51.200	50.882	51	**50.625**	0.00	50.625	51
81	50.568	50.258	50		0.50	50.125	50
82	49.951	49.648	50		1.25	49.625	50
83	49.349	49.054	49		1.75	49.125	49
84	48.762	48.473	48	**48.125**	0.00	48.125	48
85	48.188	47.906	48		0.50	47.625	48
86	47.628	47.353	47		1.25	47.125	47
87	47.080	46.811	47		1.75	46.625	47
88	46.545	46.282	46	**46.125**	0.00	46.125	46
89	46.022	45.765	46		0.50	45.625	46
90	45.511	45.260	45		1.25	45.125	45
91	45.011	44.765	45		1.75	44.625	45
92	44.522	44.281	44	**44.125**	0.00	44.125	44
93	44.043	43.807	44		0.50	43.625	44
94	43.574	43.344	43		1.25	43.125	43
95	43.116	42.890	43		1.75	42.625	43
96	42.667	42.446	42	**42.375**	0.00	42.375	42
97	42.227	42.010	42		0.50	41.875	42
98	41.796	41.584	42		0.75	41.625	42
99	41.374	41.166	41		1.25	41.125	41
100	40.960	40.756	41	**40.875**	0.00	40.875	41
101	40.554	40.355	40		0.50	40.375	40
102	40.157	39.961	40		0.75	39.125	40
103	39.767	39.575	40		1.25	39.625	40
104	39.385	39.196	39	**39.125**	0.00	39.125	39
105	39.010	38.825	39		0.50	38.625	39
106	38.642	38.460	38		0.75	38.375	38
107	38.280	38.102	38		1.25	37.875	38
108	37.926	37.751	38	**37.875**	0.00	37.875	38
109	37.578	37.406	37		0.50	37.375	37
110	37.236	37.068	37		0.75	37.125	37
111	36.901	36.735	37		1.25	36.625	37
112	36.571	36.409	36	**36.375**	0.00	36.375	36
113	36.248	36.088	36		0.50	35.875	36
114	35.930	35.773	36		0.75	35.625	36
115	35.617	35.463	35		1.00	35.375	35
116	35.310	35.159	35	**35.375**	0.00	35.375	35
117	35.009	34.860	35		0.50	34.875	35
118	34.712	34.565	35		0.75	34.625	35
119	34.420	34.276	34		1.00	34.375	34
120	34.133	33.992	34	**34.125**	0.00	34.125	34
121	33.851	33.712	34		0.50	33.625	34
122	33.574	33.437	33		0.75	33.375	33
123	33.301	33.166	33		1.00	33.125	33
124	33.032	32.900	33	**33.125**	0.00	33.125	33
125	32.768	32.637	33		0.50	32.625	33
126	32.508	32.379	32		0.75	32.375	32
127	32.252	32.125	32		1.00	32.125	32

Table A-3: Optimal 5 bit reciprocals generated from the (6,5) bipartite reciprocal table

Input Index Bits	Positive Part	Input Index Bits	Positive Part
1.000 000 xxx	0.1 1111111110 1	1.100 000 xxx	0.1 0101010100 1
1.000 001 xxx	0.1 1111011110 1	1.100 001 xxx	0.1 0101000110 1
1.000 010 xxx	0.1 1110111111 1	1.100 010 xxx	0.1 0100111000 1
1.000 011 xxx	0.1 1110100010 1	1.100 011 xxx	0.1 0100101100 1
1.000 100 xxx	0.1 1110000101 1	1.100 100 xxx	0.1 0100011110 1
1.000 101 xxx	0.1 1101101010 1	1.100 101 xxx	0.1 0100010010 1
1.000 110 xxx	0.1 1101010000 1	1.100 110 xxx	0.1 0100000100 1
1.000 111 xxx	0.1 1100110101 1	1.100 111 xxx	0.1 0011111000 1
1.001 000 xxx	0.1 1100011010 1	1.101 000 xxx	0.1 0011101100 1
1.001 001 xxx	0.1 1101000010 1	1.101 001 xxx	0.1 0011100000 1
1.001 010 xxx	0.1 1011101010 1	1.101 010 xxx	0.1 0011010100 1
1.001 011 xxx	0.1 1011010011 1	1.101 011 xxx	0.1 0011001000 1
1.001 100 xxx	0.1 1010111100 1	1.101 100 xxx	0.1 0010111100 1
1.001 101 xxx	0.1 1010100101 1	1.101 101 xxx	0.1 0010110101 1
1.001 110 xxx	0.1 1010010000 1	1.101 110 xxx	0.1 0010100111 1
1.001 111 xxx	0.1 1001111011 1	1.101 111 xxx	0.1 0010011101 1
1.010 000 xxx	0.1 1001100101 1	1.110 000 xxx	0.1 0010010001 1
1.010 001 xxx	0.1 1001010001 1	1.110 001 xxx	0.1 0010001001 1
1.010 010 xxx	0.1 1000111101 1	1.110 010 xxx	0.1 0001111101 1
1.010 011 xxx	0.1 1000101011 1	1.110 011 xxx	0.1 0001110101 1
1.010 100 xxx	0.1 1000010111 1	1.110 100 xxx	0.1 0001101001 1
1.010 101 xxx	0.1 1000000101 1	1.110 101 xxx	0.1 0001100001 1
1.010 110 xxx	0.1 0111110011 1	1.110 110 xxx	0.1 0001011000 1
1.010 111 xxx	0.1 0111100011 1	1.110 111 xxx	0.1 0001001101 1
1.011 000 xxx	0.1 0111010000 1	1.111 000 xxx	0.1 0001000101 1
1.011 001 xxx	0.1 0111000000 1	1.111 001 xxx	0.1 0000111100 1
1.011 010 xxx	0.1 0110110000 1	1.111 010 xxx	0.1 0000110001 1
1.011 011 xxx	0.1 0110100000 1	1.111 011 xxx	0.1 0000101001 1
1.011 100 xxx	0.1 0110010000 1	1.111 100 xxx	0.1 0000100001 1
1.011 101 xxx	0.1 0110000000 1	1.111 101 xxx	0.1 0000011001 1
1.011 110 xxx	0.1 0101110000 1	1.111 110 xxx	0.1 0000010001 1
1.011 111 xxx	0.1 0101100100 1	1.111 111 xxx	0.1 0000001001 1

Input Index Bits	Negative Part	Input Index Bits	Negative Part
1.000 xxx 000	0.000000 00000 0	1.100 xxx 000	0.000000 00000 0
1.000 xxx 001	0.000000 00100 0	1.100 xxx 001	0.000000 00010 0
1.000 xxx 010	0.000000 00111 0	1.100 xxx 010	0.000000 00100 0
1.000 xxx 011	0.000000 01011 0	1.100 xxx 011	0.000000 00110 0
1.000 xxx 100	0.000000 01111 0	1.100 xxx 100	0.000000 01000 0
1.000 xxx 101	0.000000 10010 0	1.100 xxx 101	0.000000 01010 0
1.000 xxx 110	0.000000 10110 0	1.100 xxx 110	0.000000 01100 0
1.000 xxx 111	0.000000 11001 0	1.100 xxx 111	0.000000 01110 0
1.001 xxx 000	0.000000 00000 0	1.101 xxx 000	0.000000 00000 0
1.001 xxx 001	0.000000 00011 0	1.101 xxx 001	0.000000 00001 0
1.001 xxx 010	0.000000 00110 0	1.101 xxx 010	0.000000 00011 0
1.001 xxx 011	0.000000 01001 0	1.101 xxx 011	0.000000 00100 0
1.001 xxx 100	0.000000 01100 0	1.101 xxx 100	0.000000 00110 0
1.001 xxx 101	0.000000 01111 0	1.101 xxx 101	0.000000 00111 0
1.001 xxx 110	0.000000 10010 0	1.101 xxx 110	0.000000 01001 0
1.001 xxx 111	0.000000 10101 0	1.101 xxx 111	0.000000 01011 0
1.010 xxx 000	0.000000 00000 0	1.110 xxx 000	0.000000 00000 0
1.010 xxx 001	0.000000 00011 0	1.110 xxx 001	0.000000 00001 0
1.010 xxx 010	0.000000 00110 0	1.110 xxx 010	0.000000 00011 0
1.010 xxx 011	0.000000 01001 0	1.110 xxx 011	0.000000 00100 0
1.010 xxx 100	0.000000 01011 0	1.110 xxx 100	0.000000 00110 0
1.010 xxx 101	0.000000 01101 0	1.110 xxx 101	0.000000 00111 0
1.010 xxx 110	0.000000 01111 0	1.110 xxx 110	0.000000 01001 0
1.010 xxx 111	0.000000 10001 0	1.110 xxx 111	0.000000 01010 0
1.011 xxx 000	0.000000 00000 0	1.111 xxx 000	0.000000 00000 0
1.011 xxx 001	0.000000 00010 0	1.111 xxx 001	0.000000 00001 0
1.011 xxx 010	0.000000 00100 0	1.111 xxx 010	0.000000 00011 0
1.011 xxx 011	0.000000 00110 0	1.111 xxx 011	0.000000 00100 0
1.011 xxx 100	0.000000 01000 0	1.111 xxx 100	0.000000 00110 0
1.011 xxx 101	0.000000 01010 0	1.111 xxx 101	0.000000 00111 0
1.011 xxx 110	0.000000 01100 0	1.111 xxx 110	0.000000 01001 0
1.011 xxx 111	0.000000 01110 0	1.111 xxx 111	0.000000 01010 0

Table A-4: (6,10) positive part and (6,5) negative part ROM tables composing the borrow-save result of a 9-bits-in,8-bits-out faithful bipartite reciprocal table

Chopped Input ×(1/512)	Low end Reciprocal ×(1/512)	Middle point Reciprocal ×(1/512)	Rounded Reciprocal ×(1/512)	Table P Value+$\frac{1}{8}$ ×(1/512)	Table N Value ×(1/512)	Prerounded Reciprocal ×(1/512)	Rounded Reciprocal ×(1/512)
545	480.998	480.557	481	481.375	1.00	480.375	480
546	480.117	479.678	480	481.375	1.75	479.625	480

Table A-5: Two lines from the 256 line enumeration of reciprocals generated by the (9,8) faithful bipartite reciprocal table

Chopped Input ×(1/1024)	Low end Reciprocal ×(1/512)	Middle point Reciprocal ×(1/512)	Table P Value ×(1/512)	Table N Value ×(1/512)	Pre-rounded Reciprocal ×(1/512)	Difference in ulps
1024	512.000	511.750	511.663	0.000	511.663	0.087
1025	511.500	511.251		0.474	511.189	0.062
1026	511.002	510.753		0.947	510.716	0.037
1027	510.504	510.256		1.419	510.244	0.012
1028	510.008	509.760		1.890	509.773	-0.013
1029	509.512	509.265		2.360	509.302	-0.038
1030	509.017	508.770		2.830	508.833	-0.063
1031	508.524	508.277		3.298	508.365	-0.087
1032	508.031	507.785	507.724	0.000	507.724	0.061
1033	507.539	507.294		0.474	507.250	0.043
1034	507.048	506.803		0.947	506.778	0.026
1035	506.558	506.314		1.419	506.305	0.008
1036	506.069	505.825		1.890	505.834	-0.009
1037	505.581	505.338		2.360	505.364	-0.026
1038	505.094	504.851		2.830	504.895	-0.043
1039	504.608	504.366		3.298	504.426	-0.061
1040	504.123	503.881	503.846	0.000	503.846	0.035
..	..	..	..	..	..	
1047	500.753	500.514		3.298	500.548	-0.035
1048	500.275	500.036	500.027	0.000	500.027	0.009
..	..	..	..	..	..	
1055	496.955	496.720		3.298	496.729	-0.009

Chopped Input ×(1/1024)	Low end Reciprocal ×(1/512)	Middle point Reciprocal ×(1/512)	Table P Value ×(1/512)	Table N Value ×(1/512)	Pre-rounded Reciprocal ×(1/512)	Difference in ulps
1056	496.485	496.250	496.266	0.000	496.266	-0.016
..	..	..	..	..	..	
1063	493.215	492.984		3.298	492.968	0.016
1064	492.752	492.520	492.561	0.000	492.561	-0.040
..	..	..	..	..	..	
1071	489.531	489.303		3.298	489.263	0.040
1072	489.075	488.847	488.911	0.000	488.911	-0.064
1073	488.619	488.391		0.474	488.437	-0.04
1074	488.164	487.937		0.947	487.964	-0.027
1075	487.710	487.483		1.419	487.492	-0.009
1076	487.257	487.030		1.890	487.021	0.009
1077	486.804	486.578		2.360	486.550	0.028
1078	486.353	486.127		2.830	486.081	0.046
1079	485.902	485.677		3.298	485.613	0.064
1080	485.452	485.227	485.315	0.000	485.315	-0.087
1081	485.003	484.779		0.474	484.841	-0.062
1082	484.555	484.331		0.947	484.368	-0.037
1083	484.107	483.884		1.419	483.896	-0.012
1084	483.661	483.438		1.890	483.425	0.013
1085	483.215	482.992		2.360	482.954	0.038
1086	482.770	482.548		2.830	482.485	0.063
1087	482.326	482.104		3.298	482.017	0.087

Table A-6: The pre-rounded reciprocals and the errors incurred in the first block of the 10-bits-in 8-bits-out bipartite reciprocal table