Analysis of Reciprocal and Square Root Reciprocal Instructions in the AMD K6-2 Implementation of 3D Now *

Cristina Iordache and David W. Matula
Dept. of Computer Science and Engineering
Southern Methodist University
Dallas, Texas 75275
Email: cristina, matula@seas.smu.edu

Abstract

Reciprocal and root reciprocal functions at "half" and IEEE single precision formats are specified in the AMD 3D Now instruction set. Implementations in the recently released AMD K6-2 microprocessor are analyzed herein by exhaustive computation and timing loops to ascertain the accuracy and monotonicity properties of the output and throughput/latency cycle counts. Periodicities in stepwise function output were observed and employed to construct an underlying bipartite table that can serve as the core of the respective reciprocal function outputs. The recommended RISC instruction macros generated single precision reciprocals and root reciprocals accurate to a unit in the last place. However, the root reciprocal functions failed to satisfy the desirable monotonicity property typically implemented as an industry standard for elementary functions on x86 floating point units. Reasons for the failure are provided and an adjusted table is shown to satisfy the monotonicity standard. Results are summarized in Table 1 and described in the body of this report.

		reciprocal	reciprocal square root
"half" precision table lookup reciprocal instruction	Table size	2.87 KBytes (bipartite-2 @ 10 bit index)	5.5 KBytes (bipartite-4 @ 10 bit index)
	Accuracy	14.9 bits	15.5 bits
	Unit-in-last-place guaranteed at	14 bits	15 bits
	Monotonic Output	yes	no
	Throughput/latency	1/2	1/2
IEEE single precision reciprocal (3 to 4 instruction macro)	Unit-in-last-place guaranteed at	24 bits	24 bits
	% round-to-nearest results	99.8	87.4
	Monotonic Output	yes	no
	Latency (one reciprocal)	6	8
	Latency (two reciprocals)	7	9

Table 1: Table size, accuracy, and cycle count for 3DNow reciprocal instructions on the AMD K6-2

*This work was supported by grant No. 5-27921 from Cyrix Corporation

1 Introduction and Summary

The new AMD K6-2 processor supports floating-point intensive applications, in particular 3D graphics, with an extension of 21 instructions [1], [2], [3] for single precision floating point operations.

Division can be performed by a four data-dependent instruction macro: the first three form the single precision reciprocal of the divisor, and the fourth one is a multiplication. Similarly, the single precision square root reciprocal can be formed by a four instruction macro, and then the square root can be obtained with one multiplication.

Our study is concerned with the accuracy and performance of the reciprocal and square root reciprocal operations. Accuracy measures we used included the *maximum round-up and round-down residuals: max. RU, max. RD* and the percentage of outputs correctly rounded to nearest. Time performance was evaluated by measuring the latency and the throughput of each instruction involved, as well as the average latency when instruction sequences operating on independent data are interleaved.

2 The AMD K6-2 3D Now Single Precision Reciprocal

To compute the single-precision reciprocal $1/y$, an initial approximation is first produced with the **PFRCP** instruction. The next two instructions **PFRCPIT1, PFRCPIT2** apply one Newton-Raphson iteration to improve its accuracy:

PFRCP X_0, X_1
PFRCPIT1 X_1, X_0
PFRCPIT2 X_1, X_0

PFRCPIT1 and PFRCPIT2 are vector instructions, meaning that the 64-bit MMX registers used as operands pack two 32-bit floating point values each and two results are computed in parallel. However, the PFRCP instruction is scalar and since each PFRCPIT1 must be preceded by a PFRCP, the performance gain introduced by a vectorized PFRCPIT1 is questionable.

By inspecting the output values, we verified that PFRCPIT1 computes and stores the factor $1 - y * X_0$ rather than $2 - y * X_0$ typical for the Newton-Raphson iteration $X_1 = X_0 * (2 - y * X_0)$. $(1 - y * X_0)$ is very small in magnitude ($\approx 2^{-15}$) and, as we found, it is stored shifted 8 positions to the left in order to preserve a total of 32 significant bits in the factor $(2 - y * X_0)$ while using a single precision format. PFRCPIT2 performs a simple multiplication of the reconstituted factor. We studied the PFRCP instruction more extensively in order to determine the likely table construction method it uses; our conclusions are presented later in this section.

The exponent is always easy to obtain for the reciprocal. The value of the significand is usually computed from the operand's significand only, thus accuracy measurements can be limited to the fundamental binade range $[1, 2)$. Since there are only 8 million (2^{23}) distinct single precision values in this range, exhaustive testing and verification is possible. In fact tables may be perturbed and exhaustive tests employed in each case to improve the computed single precision reciprocals overall. We also checked the $[2, 4)$ binade to verify that the results are consistent.

2.1 Accuracy Measurements

For the single precision reciprocal obtained by refining the initial approximation given by PFRCP with a Newton-Raphson iteration (PFRCPIT1,PFRCPIT2), we measured results exhaustively for all 8 million (2^{23}) cases over the entire range $[1, 2)$. The instruction macro was configured to provide a one ulp rounding, meaning that each result was either a round up value or a round down value of the infinitely precise reciprocal. We measured the maximum residual magnitude in ulps for all values effectively rounded up, and separately the maximum magnitude residual corresponding to effective round down results.

max. $RU = 0.503793$ ulp at $y = 1.967107$

max. $RD = 0.514782$ ulp at $y = 1.063600$ (1 ulp of output $= 2^{-24}$),

and a percentage round to nearest of 99.819648%. The 15129 cases (out of 2^{23}) that are not round to nearest correspond to the outputs with residuals above 0.5 ulp in absolute value. Stepping through all 2^{23} such cases occur about twice as frequently at the beginning of the $[1, 2)$ binade as they do towards the end. In one case near 2 ($y = 1.997711$), a sequence of three equal outputs is encountered. This cannot happen for infinitely precise reciprocals rounded to nearest, which for inputs near 2 alternate in differences of 0 and 1 ulp. The 2^{23} outputs in sequence were monotonic.

For PFRCP (short reciprocal only), we also determined the accuracy in bits [5] as defined by the negative base 2 log of the maximum magnitude of the relative error, obtaining:

max. $RU = 532.261236$ ulp at $y = 1.030304$

max. $RD = 532.227777$ ulp at $y = 1.001099$ (1 ulp of output $= 2^{-24}$).

with the accuracy being about 14.9 bits. For any single precision (24 bit) input, the result belongs to a run of 256 or 512 equal outputs, meaning that only the first 15 bits after the leading 1 are used to determine the reciprocal. Also, the outputs are sequentially monotonic and given rounded to the 17th place. The PFRCP instruction may be described as consistent with the format of a 15 bits in, 16 bits out lookup table. Respecting the 17th place as the effective last place, the maximum residuals are then slightly larger than 4 ulps.

2.2 Latency and Throughput

Each instruction was tested separately; we found a latency of 2 cycles and throughput of 1 for all instructions that take both operands from registers. When one operand was taken from memory, the latency appeared to be somewhat larger probably due to memory stalls.

The methodology used was the following: The clock rate was estimated by timing empty loops and loops containing NOPs. For the system tested, we obtained

$\frac{2^{31}}{clock\_rate} = 2^{31} \cdot clock\_cycle\_time \approx 7$ sec.

for $i = 1$ **to** 2^{31} **do**

PFRCP $mm0, mm7$

is executed in 14 sec., which verifies the latency of PFRCP is 2 cycles. (The instructions in this loop are considered data dependent because the destination register is the same; for instructions such as PFRCPIT1 or PFRCPIT2 the destination register also carries input data).

for $i = 1$ **to** 2^{31} **do**

PFRCP $mm0, mm7$

PFRCP $mm1, mm7$

takes the same amount of time, verifying that the throughput is 1.

In a similar way, we found the same results for PFRCPIT1 and PFRCPIT2.

When one operand is taken from memory, it was found that

for $i = 1$ **to** 2^{31} **do**

PFRCP $mm0, y$

takes 16 to 17 sec., while

for $i = 1$ **to** 2^{31} **do**

PFRCP $mm0, y$

PFRCP $mm1, y$

takes 19 sec., suggesting that on average a stall of about $\frac{2.5}{7} \approx 0.357$ cycles occurred each time the memory operand was accessed (for the system tested).

As expected, the latency for computing one single-precision reciprocal is 6 cycles (three data dependent instructions), but two reciprocals interleaved can be computed in 7 cycles:

for $i = 1$ **to** 2^{31} **do**

PFRCP $mm1, mm0$

PFRCPIT1 $mm0, mm1$

PFRCPIT2 $mm0, mm1$

and
for $i = 1$ **to** 2^{31} **do**
PFRCP $mm1, mm0$
PFRCP $mm2, mm7$
PFRCPIT1 $mm0, mm1$
PFRCPIT1 $mm7, mm2$
PFRCPIT2 $mm0, mm1$
PFRCPIT2 $mm7, mm2$
both take about 43 sec. (i.e. 6 cycles per iteration).

2.3 PFRCP Implementation

As also mentioned in [1], [2], the short reciprocal instruction PFRCP is based on table lookups. We measured its accuracy ($-\log_2 max\,|rel.\,error|$) to be about 14.9 bits and determined that it uses only the first 16 bits of the input (including the leading 1); the output is 17 bits long (including the leading 1). A direct 15 bits in, 16 bits out lookup table would be 64 KBytes in size. To avoid such a large direct table lookup, the table output values were analyzed for patterns that would reveal composite table lookup implementation from compressed tables.

A repeating pattern in the output differences indicated that the PFRCP implementation is not a simple table lookup: by splitting the outputs into 32 groups of length 1024, we noticed that within each group the differences between successive outputs are periodic with a period of 32 (in positions divisible by 32, values may differ). Patterns change with each group of 1024 values. Such periodicity always occurs for bipartite table lookup [4], but not in multiplicative interpolation. Consider the following property of additive bipartite table interpolation.

Observation
Let $z(1.b_1 b_2 \dots b_p) = T_1(b_1 b_2 \dots b_k \dots b_m) + T_2(b_1 \dots b_k, b_{m+1} \dots b_p)$ *be a bipartite scheme for computing an approximation of* $f(1.b_1 b_2 \dots b_p)$*. Then within a group of* 2^{p-k} *inputs (for which* $b_1 \dots b_k$ *has the same value), output differences are periodic with a period of* 2^{p-m}*, with the exception of differences at points where* $b_{k+1} \dots b_m$ *changes.*
Proof
For $b_{m+1} \dots b_p > 0$, the output difference is
$z(1.b_1 \dots b_k b_{k+1} \dots b_m b_{m+1} \dots b_p) - z(1.b_1 \dots b_k b_{k+1} \dots b_m b_{m+1} \dots b_p - 2^{-p}) =$
$T_2(b_1 \dots b_k, b_{m+1} \dots b_p) - T_2(b_1 \dots b_k, b_{m+1} \dots b_p - 1)$.
The value of the output difference does not depend on bits $b_{k+1} \dots b_m$, and thus is repeated 2^{m-k} times within a group of 2^{p-k} inputs for which $b_1 \dots b_k$ is the same.
For $b_{m+1} \dots b_p = 0$, there can be different values every time:
$z(1.b_1 \dots b_k b_{k+1} \dots b_m b_{m+1} \dots b_p) - z(1.b_1 \dots b_k b_{k+1} \dots b_m b_{m+1} \dots b_p - 2^{-p}) = T_2(b_1 \dots b_k, 0) - T_2(b_1 \dots b_k, 1) + T_1(b_1 \dots b_m) - T_1(b_1 \dots b_m - 1)$ (the difference depends on all bits $b_1 \dots b_m$). $\diamondsuit$

No deeper nested periodicity (which in a similar manner would indicate a tri- or multi-partite table implementation) was found within groups of 32 inputs, thus according to the observations above, PFRCP suggests a bipartite table implementation: the high order table based on the first ten bits, and the low order table based on the ten bits $b_1 \dots b_5$ and $b_{11} \dots b_{15}$.

Differences that are symmetric [6] about the middle of each 31 input group (period) would indicate that the low order table was centered around $b_{11} \dots b_{15} = 01111, b_{11} \dots b_{15} = 10000$, meaning that only $b_{12} \dots b_{15}$ are used to index the table, while b_{11} is used as a sign bit to halve the size of the second table.

Observation *If the low order table of a bipartite table implementation is such that* $T_2(b_1 \dots b_k, b_{m+1} \dots b_p) = -T_2(b_1 \dots b_k, \overline{b_{m+1} \dots b_p})$*, then within a group of* $2^{p-m} - 1$ *outputs (such that the input bits* $b_1 \dots b_m$ *are the same and* $b_{m+1} \dots b_p > 0$*), the output differences are symmetric around the middle one (for which* $b_{m+1} \dots b_p = 10 \dots 0$*).*

Proof As shown earlier, for $b_{m+1} \ldots b_p < 2^{p-m} - 1$ two successive outputs are within the same $b_1 \ldots b_m$ group and their difference is
$D(b_1 \ldots b_m b_{m+1} \ldots b_p + 2^{-p}) = z(1.b_1 \ldots b_m b_{m+1} \ldots b_p + 2^{-p}) - z(1.b_1 \ldots b_m b_{m+1} \ldots b_p) = T_2(b_1 \ldots b_k, b_{m+1} \ldots b_p + 1) - T_2(b_1 \ldots b_k, b_{m+1} \ldots b_p)$.

Now consider
$D(b_1 \ldots b_m \overline{b_{m+1} \ldots b_p}) = z(1.b_1 \ldots b_m \overline{b_{m+1} \ldots b_p}) - z(1.b_1 \ldots b_m \overline{b_{m+1} \ldots b_p} - 2^{-p}) =$
$= T_2(b_1 \ldots b_k, \overline{b_{m+1} \ldots b_p}) - T_2(b_1 \ldots b_k, \overline{b_{m+1} \ldots b_p + 1}) =$
$-T_2(b_1 \ldots b_k, b_{m+1} \ldots b_p) + T_2(b_1 \ldots b_k, b_{m+1} \ldots b_p + 1)$.

Thus $D(b_1 \ldots b_m \overline{b_{m+1} \ldots b_p}) = D(b_1 \ldots b_m b_{m+1} \ldots b_p + 2^{-p})$ (differences within the same $b_1 \ldots b_m$ group are symmetric). $\diamondsuit$

Such symmetry was not found within groups of 31 output differences for the PFRCP instruction. All of the information obtained by output pattern analysis allows us to infer that the PFRCP uses a high order table with 10 bits in, 16 bits out (the most significant bit is not stored), and a low order table with 10 bits in, 7 bits out for a total table size of $(16+7) * 2^{10}/8$ bytes = 2.875 Kbytes.

A high order table T_1' and a low order table T_2' that yield the same outputs can be easily separated by setting to zero one entry of T_2' for each combination of the first $k = 5$ bits.

We chose to set $T_2'(b_1 \ldots b_5, 16) = 0$ (the middle entry) because it yields a relatively centered low order table; it was also noticed, by searching over the whole range $[1, 2)$ for all combinations of the last 5 bits that the minimum maximum absolute error is at $b_{11} \ldots b_{15} = 10000$. (The precise reciprocal was taken to be $\frac{1}{1.b_1 \ldots b_{15} + 2^{-16}}$, which centers the residuals).

Then $T_1'(b_1 \ldots b_{10}) = T_1(b_1 \ldots b_{10}) + T_2(b_1 \ldots b_5, 16)$ (the output at $1.b_1 b_2 \ldots b_{10} 10000xxx \ldots$), and
$T_2'(b_1 \ldots b_5, b_{11} \ldots b_{15}) = T_2(b_1 \ldots b_5, b_{11} \ldots b_{15}) - T_2(b_1 \ldots b_5, 16)$.

The table values can be found in the appendices of this report. All of the PFRCP outputs in the K6-2 3D implementation can be generated by combining them in a bipartite scheme as shown.

We also investigated whether a simple formula could be determined to algebraically construct the bipartite tables of the appendix that yield the PFRCP results but were not able to find a simple formula, perhaps implying some fine tuning of final table values was employed.

3 The AMD K6-2 Single Precision Square Root Reciprocal

The single precision square root reciprocal in the K6-2 3D implementation can be obtained with a sequence of five instructions, as follows:

PFRSQRT X_1, y
MOV X_0, X_1
PFMUL X_1, X_1
PFRSQIT1 X_1, y
PFRCPIT2 X_1, X_0

Most observations made for the reciprocal also hold for the square root reciprocal, as will be seen. While the instructions used for the Newton-Raphson refining step (PFMUL, PFRSQIT1 and PFRCPIT2) are vector instructions, the initial approximation instruction PFRSQRT is scalar. Note that the same instruction, PFRCPIT2, is used in both the Newton-Raphson step for the reciprocal and the Newton-Raphson step for the square root reciprocal.

We verified that PFRSQIT1 computes the first factor used in the Newton-Raphson step: $1/2 * (3 - y * X_0 * X_0) = 1/2 * (3 - y * X_1) = 1 + 1/2 * (1 - y * X_1)$. The result is stored in single precision format, but since the term $(1 - y * X_1)$ with the implied unit deleted is small, it can be shifted a few positions to preserve more bits of accuracy. We found that indeed,

$1/2*(1-y*X_1)$ is shifted 8 positions to the left. PFRCPIT2, as already mentioned, simply performs a multiplication of the reconstituted factor.

3.1 Accuracy Measurements

For the square root reciprocal, measurements can be limited to the two binade range $[1,2),[2,4)$.
In the [1,2) range:
max. RU= 0.846738 ulp at $y = 1.997219$
max. RD= 0.850265 ulp at $y = 1.989491$,
Accuracy: 23.738 bits,
Percentage of outputs correctly rounded to nearest: 85.22%.
In the $[2,4)$ range:
max. RU= 0.752070 ulp at $y = 3.970857$
max. RD= 0.745616 ulp at $y = 3.953556$,
Accuracy: 23.414 bits,
Percentage of outputs correctly rounded to nearest: 89.544%.

Corresponding to sequential one ulp steps of input, some output gap sequences that cannot occur for the infinitely precise root reciprocals rounded to nearest do occur for these approximations. For example, sequences of three equal outputs occur as early as $y = 1.248443$, while they should not occur for inputs below 1.5. Other examples of undesirable output gap sequences are groups of 4 equal outputs as early as $y = 1.589142$ and groups of 5 equal outputs as early as $y = 1.892303$ (while neither of these should occur for inputs below 2).

A significant deficiency in the implementation is that the recommended macro produced non-monotonic root reciprocals: while the infinitely precise function is strictly decreasing, the AMD single precision root reciprocal is strictly increasing at some points in the intervals $(1.5, 2)$ and $(3, 4)$. This is particularly undesirable for multimedia applications where human perceptions are more sensitive to visual and audio perturbation artifacts than to uniform error levels. We know that PFRSQIT1 rounds at the 32nd bit, which should provide sufficient accuracy to prevent this from happening, as we will show shortly. However, PFRSQIT1 takes the square of the short approximation as an operand, and this square is computed with a rounding error to single precision by a PFMUL instruction, creating the anomaly.

The distance between consecutive root reciprocals obtained by refining the short approximations X_0 provided by PFRSQRT with a Newton-Raphson iteration computed with infinite precision is:

$\Delta X_1 = \frac{3}{2}\Delta X_0 \cdot (1 - yX_0^2) - \frac{1}{2}X_0^3\Delta y.$

We measured the precision of PFRSQRT to be about 15.5 bits, and thus we can estimate $|1 - yX_0^2| < 2^{-14.5}$. We also know that $|\Delta X_0| < 2^{-15}$, and thus the first term in the equality above is very small, of the order of $2^{-29.5}$. For single precision inputs we have $\Delta y = 2^{-23}$, and for inputs above 1.588 we have $X_0^3 < 0.5$. Thus the second term which provides a bound to insure monotonically decreasing behavior is of the order of 2^{-25}.

Now considering the roundoff error ϵ from the preceding PFMUL operation as the only source of error, we get:

$\Delta X_1 = \Delta X_0 \cdot (1 + \frac{1-yX_0^2}{2}) - \frac{1}{2}X_0^3\Delta y - \frac{yX_0}{2}\Delta X_0^2 = \frac{3}{2}\Delta X_0 \cdot (1 - yX_0^2) - \frac{1}{2}X_0^3\Delta y + (yX_0)\cdot\epsilon.$

For inputs between 1.5 and 2, $|\epsilon|$ can be as large as 2^{-25} , and for $y > 2$, it can be as large as 2^{-26}. $yX_0 \sim \sqrt{y} > 1.2$. Clearly, the error term introduced by computing X_0^2 in single precision can reverse the decreasing order of consecutive root reciprocals and explain the lack of monotonicity in the output of the macro instruction sequence recommended for a single precision root reciprocal. It also explains the fact that only 87% of the single precision results agreed with round to nearest starting from a more accurate table than that of the reciprocal function, where refinement gave over 99% round to nearest results.

We checked exhaustively and indeed, in all cases when the root reciprocal is increasing, the difference between roundoff errors introduced by the single precision multiplication is between $(0.54 \cdot 2^{-24}, 0.96 \cdot 2^{-24})$ for inputs between 1.5 and 2, and between $(0.54 \cdot 2^{-25}, 0.94 \cdot 2^{-25})$

for inputs between 3 and 4. We verified that for these cases $-\frac{1}{2}X_0^3\Delta y + (yX_0)\cdot\epsilon$ is always positive and of the order of at least 2^{-29}, which explains this phenomenon. Obviously, all of these cases occur at points where the leading 16 bit portion changes (because this is when the PFRSQRT output changes) and in all of them, the result differs by 1 ulp from the one obtained by applying an exact Newton-Raphson iteration on the PFRSQRT output.

We should also mention that we found one case where PFRSQRT itself is not monotonic, which should have been detected and corrected. This happens at $y = 3.865234$: PFRSQRT goes up by 2^{-17}.

The nonmonotonicity problem can NOW! be fixed by slightly altering the lookup tables. A solution we found is presented in Subsection 3.3.

For the short reciprocal (PFRSQRT) only, we measured
max. RU= 282.171898 ulp at $y = 1.092895$
max. RD= 285.153030 ulp at $y = 1.034302$
and a precision of about 15.795 bits in the $[1,2)$ range.
In the $[2,4)$ range:
max. RU= 251.344890 ulp at $y = 2.056762$
max. RD= 248.465250 ulp at $y = 2.050659$
(precision about 15.506 bits).

3.2 Latency and Throughput

As for the reciprocal instruction, the latency and throughput were measured for each instruction independently and also verified for the back-to-back macro sequence that computes the root reciprocal. All tested individual instructions were found to have a latency of 2 cycles and throughput of one cycle.

for $i = 1$ **to** 2^{31} **do**
  PFRSQRT $mm0, mm7$
and
for $i = 1$ **to** 2^{31} **do**
  PFRSQRT $mm0, mm7$
  PFRSQRT $mm1, mm7$

both take 14 sec., which verifies the latency of 2 cycles and throughput of 1 for PFRSQRT. (The clock cycle time was measured to be about $7/2^{31}$ sec.) PFMUL and PFRSQIT1 were verified in the same way, with the same results.

The latency of the 5 instruction sequence needed to compute the reciprocal root is 8 cycles, since three of the five instructions are dependent. Indeed,

for $i = 1$ **to** 2^{31} **do**
  PFRSQRT $mm1, mm7$
  MOV $tmp, mm1$
  MOV $mm0, tmp$
  PFMUL $mm1, mm1$
  PFRSQIT1 $mm1, mm7$
  PFRCPIT2 $mm1, mm0$
takes $57 \approx 8\cdot 7$ sec.

However, it is possible to compute two square root reciprocals with 9 cycle latency. For example,

for $i = 1$ **to** 2^{31} **do**
  PFRSQRT $mm1, mm7$
  PFRSQRT $mm3, mm6$
  MOV $tmp, mm1$
  MOV $mm0, tmp$
  MOV $tmp, mm3$
  MOV $mm2, tmp$

PFMUL $mm1, mm1$
PFMUL $mm3, mm3$
PFRSQIT1 $mm1, mm7$
PFRSQIT1 $mm3, mm6$
PFRCPIT2 $mm1, mm0$
PFRCPIT2 $mm3, mm2$
takes the same amount of time, 57 sec.

The same instructions appear to have a longer latency when memory operands are used (due to stalls).

3.3 PFRSQRT Implementation

The implementation of the short root reciprocal instruction appears to be very similar to that of the short reciprocal (PFRCP) instruction. Only the first 15 bits (after the leading 1) from the input are used, and all outputs are rounded to the 17th place.

The same output patterns we observed for the PFRCP instruction are present in the PFRSQRT outputs in both binade ranges, $[1,2)$ and $[2,4)$. Within each group of 1024 outputs, output differences repeat with a period of 32 (except for the cases where bits $b_{11} \dots b_{15}$ of the input are divisible by 32). No deeper nested periodicity can be found, and symmetry is not present within each group of 31 repeating differences.

The arguments presented in Subsection 2.3 allow us to infer that a bipartite table lookup scheme was also used for the PFSQRT instruction. Since the same patterns and differences rounded to the 17th place are found in both binade ranges, a separate set of tables is used for each binade range (and thus the total table size is doubled compared to that used for PFRCP). A low order table with 10 bits in, 6 bits out (as compared to 7 for PFRCP) and a 10 bits in, 16 bits out high order table would be necessary for each range, for a total of $2*(16+6)*2^{10}/8 = 5.5$ KBytes.

We separated the high and low order tables by setting the low order table to zero at entries with $b_{11} \dots b_{15} = 10000_2$, by using the same procedure described in Subsection 2.3. The low order tables obtained are relatively, but not perfectly centered (as we argued, when centered low order tables are used, the output differences are symmetric, which is not the case here).

Since the AMD K6-2 single precision root reciprocal implementation is not monotonic, we tried a solution that extends the high order table to 23 bits out. The low order bits in positions 19 through 24 were chosen such that the new root reciprocal of $y = 2^{e_y} \cdot 1.b_1 b_2 \dots b_{15} 0 \dots 00$, $e_y \in \{0,1\}$ would not be above that of $y - 2^{-23+e_y}$. (Remember that such anomalies occured only when bits $b_1 \dots b_{15}$ change). Our search gave preference to positive values and small absolute values. We found that in most cases, these anomalies could be eliminated by adding small positive amounts to the high order table (i.e. only the last two or three bits, 22-24, needed changing). However, in just a few cases, negative amounts in the range $(-64, 0)$ were needed to eliminate the problem (meaning that bits 1-17 of the table entry would have to be changed leading to a 23 bit out table). Thus a monotonic single precision root reciprocal can be obtained with the same algorithm and a total table size of $2*(23+6)*2^{10}/8 = 7.25$ KBytes. The other properties we measured for the root reciprocal (see Subsection 3.1) remain practically unaffected by this change.

4 Conclusions and Summary

Our study of the AMD K6-2 single precision reciprocal and square root reciprocal operations found the following:

All instructions involved in the computation of these operations have latency of 2 and throughput of one cycle. The total latency for computing one single precision reciprocal is 6 cycles and 8 cycles for one single precision root reciprocal. Two reciprocals can be computed in 7 cycles and two root reciprocals in 9 cycles. The initial approximation instructions PFRCP and PFRSQRT are scalar, while the others are vectorized and can compute two results in parallel. However, they must be always preceded by either PFRCP or PFRSQRT, which limits the potential speedup for these two operations.

99.82% of the single precision reciprocals and 87.38% of the single precision root reciprocals as provided by the AMD implementation are correctly rounded to nearest. The most notable deficiency in the implementation regarding desired properties of the results were that the root reciprocals are not monotonic.

Both PFRCP and PFRSQRT use only the most significant 16 bits of the input; their output is always rounded to 17 places. The short reciprocal computed by PFRCP is accurate to about 14.9 bits, while PFRCP yields about 15.5 bits of accuracy. Both instructions are consistent with use of bipartite table lookup (10 bits in for each table). The total table size is 2.875 KB for PFRCP and 5.5 KB for PFRSQRT.

References

[1] *AMD-3D Technology Manual*, February 1998.

[2] Brian Case, *3DNOW Boosts Non-Intel 3D Performance*, Microprocessor Report, June 1998, pp.18-21.

[3] Linley Gwennap, *AMD Deploys K6-2 With 3DNow*, Microprocessor Report, June 1998, pp.16-17.

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

[5] D. Das Sarma, D.W. Matula, *Measuring the Accuracy of ROM Reciprocal Tables*, IEEE Trans. Comput., vol. 43, No. 8, August 1994, pp. 932-940. Also see Proc. 11th IEEE Symp. Comput. Arithmetic, 1993, pp. 95-102.

[6] M.J. Schulte and J.E. Stine, *Improved Bipartite Tables for Accurate Function Approximation*, Proc. 13th Symposium on Computer Arithmetic, 1997.

APPENDIX

A. SHORT RECIPROCAL (PFRCP) LOW ORDER TABLE

B. SHORT RECIPROCAL (PFRCP) HIGH ORDER TABLE

C. SHORT ROOT RECIPROCAL (PFRSQRT) LOW ORDER TABLE, RANGE $[1, 2)$

D. SHORT ROOT RECIPROCAL (PFRSQRT) HIGH ORDER TABLE, RANGE $[1, 2)$

E. SHORT ROOT RECIPROCAL (PFRSQRT) LOW ORDER TABLE, RANGE $[2, 4)$

F. SHORT ROOT RECIPROCAL (PFRSQRT) HIGH ORDER TABLE, RANGE $[2, 4)$