Rounding for Quadratically Converging Algorithms for Division and Square Root

Eric M. Schwarz

S/390 Microprocessor Development
IBM Systems Technology and Architecture Division
522 South Rd., MS:P312
Poughkeepsie, NY 12601

Abstract

Exactly rounded results are necessary for many architectures such as IEEE 754 standard [5]. For division and square root, rounding is easy to perform if a remainder is available. But for quadratically converging algorithms, the remainder is not typically calculated. Past implementations have required the additional delay to calculate the remainder, or calculate the approximate solution to twice the accuracy, or have resulted in a close but not exact solution. This paper shows how the additional delay of calculating the remainder can be reduced if extra precision is available.

1 Introduction

Division and square root is defined by many common architectures to have an "exact" result. The result is calculated as though an infinitely precise intermediate result were rounded to the desired precision. ESA/390 architecture defines the result for division to be truncated (similar to round to zero) and for square root to be round to nearest. For IEEE 754 the rounding mode is selectable between four possible modes: 1) round to nearest even (default mode), 2) round to zero, 3) round to positive infinity, and 4) round to negative infinity. It is not acceptable in these architectures to calculate an answer to plus or minus the least significant digit ($\pm ulp$). Some algorithms are easier to implement with this "exactness" constraint. Restoring methods easily produce an exactly truncated result and provide a remainder. Non-restoring methods are easy to correct the least significant bit by noting the sign and magnitude of the remainder. But, quadratically converging methods are difficult to exactly round because no remainder is calculated in intermediate iterations. Typically these methods need additional cycles to explicitly calculate a remainder and then to exactly round the result appropriately.

For the past 30 years, designers have tried to solve the problem of determining the exact result for quadratically converging division and square root without calculating the remainder. To this end, this paper shows a technique for avoiding the remainder calculation for certain cases. But first, well-known techniques are presented.

1.1 Previous Inaccurate Machines

The two most popular quadratically converging algorithms for division and square root are the Newton-Raphson method and the Goldschmidt algorithm. The Goldschmidt algorithm is particularly difficult to converge on the "exact" result due to having iteration dependent error terms. This algorithm is not self-correcting and thus errors in converging the divisor to one are observable as additional errors in converging the dividend to the exact quotient. This algorithm does have the benefit of having independent operations which can be executed in parallel to reduce the execution time. To take advantage of this speed, some implementations were known to have violated their architectural requirements and only produced a close but not necessarily exact result.

The first implementation of the Goldschmidt algorithm [1, 2] produced 10 extra bits of precision and rounded up if all guard bits were equal to one. This did not produce the exactly rounded quotient in all cases.

This is a fast way to produce a somewhat accurate result but in most cases this is an unacceptable result.

1.2 Calculating Direction of Remainder

The typical method of rounding quadratic algorithms is to create a preliminary result and use it to create a remainder and compare the remainder with zero. In some implementations the remainder is explicitly calculated. In other implementations, the equivalence is evaluated without an explicit remainder calculation.

$$\begin{aligned} Q &\approx \frac{A}{B} \\ R &= A - Q * B \\ R &\ ?\ 0 \\ Q * B &\ ?\ A \end{aligned}$$

Rather than calculating the remainder, the product of the quotient and the divisor is compared with the dividend. If the calculated quotient is very accurate which should be the case, then only a few bits of the product need to be compared to the dividend [3, 6].

1.3 Calculating Twice the Accuracy

One recent technique actually avoids the remainder calculation. Markstein [4] has demonstrated that an exactly rounded quotient can be produced if an approximate result is available to twice the precision. This method also is discussed in [7]. The main disadvantage of this technique is that an additional iteration is necessary to calculate the approximate quotient to this accuracy.

1.4 Proposed Method of Using Guard Bits

The proposed method calculates one extra bit of precision for the approximate result which in most cases does not require any additional latency. Most implementations use a 7 bit initial approximation which increases to 14 bits after 1 iteration, 28 bits after 2 iterations, and to 56 bits after 3 iterations. Even with truncation errors it is typical to produce at least $N+1$ bits of precision in the preliminary result where N is the desired precision. For IEEE 754 standard, 53 bits is the required precision for long operands and this would require a 54 bit approximation. After an $N+1$ bit result is produced with the desired error tolerance of less than plus or minus the $N+1$th bit, then the least significant bit, called the guard bit, is examined. In half the cases the exactly rounded result can be determined from just inspecting this bit. In the other half of the cases, a remainder is calculated, and the rounding is based on its comparison with zero. The advantage of the proposed method as compared to other methods is that no additional delay is required in half the cases which results in a significant savings on average in terms of latency.

2 Error Tolerance

Assume the actual (infinitely precise) result is called Q and that an exactly rounded machine representable number is to be calculated. Assume $Q^\star$ is the needed answer which has N bits of accuracy and that the approximate solution has been normalized, to less than 1.0; and for hex normalization is greater than or equal to 1/16, and for binary normalization is greater than or equal to 1/2. Assume this approximate solution is called Q' and has $N+k$ bits with an error of plus or minus $2^{-(N+j)}$ where $k \geq j > 0$.

$$-2^{-(N+j)} \leq Q - Q' \leq +2^{-(N+j)}$$

Q' needs to be transformed into an $N+1$ bit number with an accuracy of plus or minus the last bit, which is weighted $2^{-(N+1)}$. To do this, first Q' is incremented by more than $2^{-(N+j)}$ but less than $2^{-(N+1)}$. Assume Q' is incremented by $2^{-(N+2)}$ and called Q'':

$$\begin{array}{rcl}
-2^{-(N+2)} - 2^{-(N+j)} \leq & Q - (Q' + 2^{-(N+2)}) & \leq -2^{-(N+2)} + 2^{-(N+j)} \\
-2^{-(N+2)} - 2^{-(N+j)} \leq & Q - Q'' & \leq -2^{-(N+2)} + 2^{-(N+j)}
\end{array}$$

Q'' is an overestimate of Q. The next step is to truncate this overestimate to N +1 bits which reduces the estimate by 0 or up to $2^{-(N+1)}$ and is called Q'''.

$$\begin{array}{rccl}
-2^{-(N+2)} - 2^{-(N+j)} + e \leq & Q - (Q'' - e) & & AND \\
-2^{-(N+2)} + 2^{-(N+j)} + e \geq & Q - (Q'' - e) & & \\
where: \quad 0 \leq & e & < 2^{-(N+1)} & \\
-2^{-(N+2)} - 2^{-(N+j)} \leq & Q - Q''' & & AND \\
+2^{-(N+1)} - 2^{-(N+2)} + 2^{-(N+j)} > & Q - Q''' & & \\
& & & \\
-2^{-(N+2)} - 2^{-(N+j)} \leq & Q - Q''' & & AND \\
+2^{-(N+2)} + 2^{-(N+j)} > & Q - Q''' & & \\
& & & \\
if \quad j = 2 \quad then: & & & \\
-2^{-(N+1)} \leq & Q - Q''' & < +2^{-(N+1)} & \\
if \quad j > 2 \quad then: & & & \\
-2^{-(N+1)} < & Q - Q''' & < +2^{-(N+1)} &
\end{array}$$

Assume $j > 2$ in our implementation. Then

$$Q''' - 0.5ulp < Q < Q''' + 0.5ulp$$

where 1 ulp is defined to be a unit in the last place of $Q^\star$ and since this is an N bit fraction, 1 ulp = 2^{-N}. The estimate Q''' is N+1 bits and has error of less than plus or minus the least significant bit, in this case the guard bit. This result easily is rounded to N bits which is shown in the next section, but first an example is given of producing Q'''.

Example: A 53 bit result is needed (N=53) and an approximate solution has 64 bits (k=11) with an accuracy of plus or minus bit 56 (j=3). 2^{-55} is added to Q' possibly in parallel with other calculations.

2.1 Collapsing the Delay

For the Goldschmidt algorithm it is typical that the last iterations multiplication needs to be executed in hardware as a multiply and add to compute the required precision. For instance, let A_i represent the i-th iteration's dividend, C_i represent the i-th convergence factor, B_i the i-th divisor.

$$\begin{array}{rcl}
A_i & = & C_{i-1} * A_{i-1} \\
B_i & = & C_{i-1} * B_{i-1} \\
C_i & = & 2 - B_i
\end{array}$$

The last calculation of the dividend or approximate quotient requires a multiplication of more than N by N bits to get desired accuracy in the quotient. Since it is known that the convergence factor is close to 1.0, the multiplication is performed as a multiply and add. Let,

$$\begin{array}{rcl}
X_{i-1} & = & 1 - C_{i-1} \\
A_i & = & A_{i-1} \quad + \quad X_{i-1} * A_{i-1}
\end{array}$$

Then,

$$\begin{array}{rcl}
Q' & = & A_{i-1} \quad + \quad X_{i-1} * A_{i-1} \\
Q'' & = & (A_{i-1} + 2^{-55}) \quad + \quad X_{i-1} * A_{i-1}
\end{array}$$

The incrementation of A_{i-1} can be computed in parallel with the multiplication and thus result in no additional delay. Then the result is truncated to 54 bits with the least significant being the guard bit. Truncation takes no additional time either. So, Q''' can be calculated with no additional delay in some implementations.

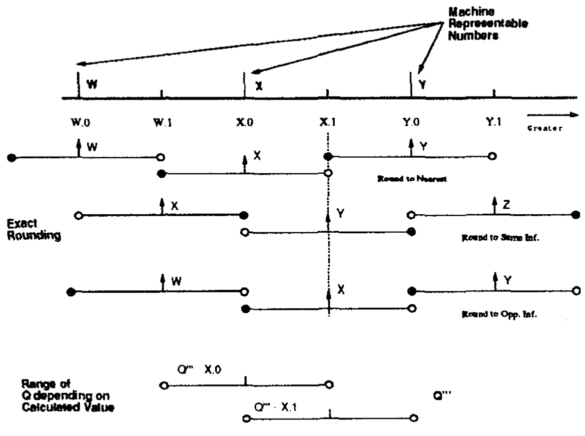

Figure 1: Rounding Diagram for Round to Nearest

3 Rounding

The action for rounding is described for the five common rounding types which can be separated into three distinct rounding modes:

1. **Round to Nearest** which includes ESA/390 round to nearest (used for square root) and IEEE 754 round to nearest even. The action is the same for either due to there being no halfway case for division or square root [4].

2. **Round to Infinity Same Sign** which indicates either round to positive infinity with a positive result or round to negative infinity with a negative result.

3. **Round to Infinity Opposite Sign** which indicates either round to positive infinity with a negative result or round to negative infinity with a positive result. And also this mode is used for round to zero or ESA/390 truncation (used for division).

In Figure 1 a number line is shown where W, X, and Y are machine representable numbers. Their relationship is $W < X < Y$. Underneath the number line a guard bit is shown after a radix point with W, X, or Y. The exact rounding actions are shown for each of the three rounding modes. Exact rounding dictates that an infinitely precise quotient must be rounded as shown by the lines below the number line. A closed end point represents the inclusion of a point and an open end represents the exclusion of the end point.

Assume that the N most significant bits of Q''' are represented by X, then the two lowest lines represent the range of the exact quotient given a guard bit of 0 (X.0) and a guard bit of 1 (X.1). From this diagram the rounding action is easily determined.

3.1 Round to Nearest

For round to nearest or round to nearest even, the lines below the number line in Figure 1 dictate the action that needs to be taken. If the guard bit equals 0 it is easily seen that result should be X which corresponds to truncating the calculated value. If the guard bit equals 1 the action depends on the relationship of the remainder to zero. If the remainder is greater than zero, then Y needs to be selected which is incrementing the calculated value. If the remainder is less than zero, then X is selected which corresponds to truncating the calculated value. The remainder can not be equal to zero, but round to nearest would dictate rounding up for this case, and round to nearest even would dictate rounding to the closest even number.

The following summarizes the actions required:

1. Guard bit equal 0: truncate
2. Guard bit equal 1: calculate remainder
   (a) Remainder > 0: increment.
   (b) Remainder < 0: truncate.
   (c) Remainder $= 0$: can't occur [4].

3.2 Round to Infinity Same Sign

For round to positive infinity and quotient positive or round to negative infinity and quotient negative:

1. Guard bit equal 1: increment
2. Guard bit equal 0: calculate remainder
   (a) Remainder > 0: increment.
   (b) Remainder ≤ 0: truncate.

3.3 Round to Infinity Opposite Sign

For round to positive infinity and quotient negative or round to negative infinity and quotient positive; also round to zero and truncate:

1. Guard bit equal 1: truncate
2. Guard bit equal 0: calculate remainder
   (a) Remainder ≥ 0: truncate.
   (b) Remainder < 0: decrement.

4 Relaxing Error Tolerance

An error tolerance of the approximate result, Q''', for avoiding the remainder calculation has been shown, but it is possible in some implementations this may not be the optimal for reducing the latency. This is because some implementations may require additional delay to achieve the suggested error tolerance. In this case, it may be advantageous to consider preliminary results with other error tolerances.

4.1 One Inclusive Point

Referring to section 2, the variable j could be relaxed to be equal to 2 rather than strictly greater than. This would cause the lower end point to the range of Q''' to be an inclusive point and the other end point to be still exclusive.

$$Q''' - 0.5ulp \leq Q < Q''' + 0.5ulp$$

In Figure 1 the left end points of the range would be inclusive. Since the halfway case can be ignored, the left end point for a guard bit equal to zero is a don't care point but the left end point for a guard bit equal to one is important. For round to nearest it can be seen that for a guard bit equal to zero, the correct result would still be truncate Q'''. And for the guard bit equal to one, the remainder would still need to be calculated. Thus, for round to nearest the actions would remain the same.

This is also true for round to infinity opposite sign. But for round to infinity same sign, the actions would change. For guard bit equal to zero, the remainder would still need to be calculated. For guard bit equal to one, the case of the actual quotient equal to exactly X would need to be examined. To do this the remainder of Q''' truncated is needed rather than the remainder without truncation. If the remainder is zero then the result is X, if it is non-zero then the result is Y.

Thus, if j is equal to 2, the actions remain the same except for round to infinity same sign with the guard bit equal to one. For round to infinity same sign, a remainder calculation is needed for either value of the guard bit.

4.2 Two Inclusive Points

Another possibility is that the error tolerance equation has both end points inclusive, as shown by the following:

$$Q''' - 0.5ulp \leq Q \leq Q''' + 0.5ulp$$

This would cause both end points to be inclusive. The left and right end points for a guard bit equal to zero can be ignored due to the halfway case not existing.

The rounding actions for round to nearest remain the same for this case but the other two rounding modes change. For round to infinity same sign and a guard bit of one, the case of exactly equal to X needs to be determined. If equal to X then the answer is X, else Y. For round to infinity opposite sign there is a similar case. For guard bit equal to one, the case of exactly equal to Y needs to be determined. If equal to Y then the answer is Y, else the answer is X.

For this error tolerance two rounding modes are penalized, round to infinity same sign and round to infinity opposite sign (which includes truncation). But the most common mode round to nearest is not penalized with extra cases of calculating a remainder comparison.

Thus, various error tolerances are possible depending on the algorithm and implementation chosen. The optimal one for avoiding the remainder calculation has been shown with an error of less than plus or minus a half ulp.

5 Conclusion

This rounding technique can be applied to both square root and division. The proposed rounding method is useful in a system where variable latency is acceptable. In almost half the cases, truncation of the preliminary result can be guaranteed to be the exactly rounded result. In a few other cases, incrementing the preliminary result is the expected result. And, in the remaining cases, the relation of remainder with respect to zero needs to be calculated to conditionally select truncate, increment, or decrement the preliminary result. For half the cases, the operation can be completed early after inspecting the guard bit of the approximate solution. It is suggested that this inspection be performed in parallel with the equivalent to a remainder comparison.

This paper has shown a simple technique for eliminating the remainder calculation in some cases if the additional accuracy is available.

References

[1] S. F. Anderson, J. G. Earle, R. E. Goldschmidt, and D. M. Powers. "The IBM system/360 model 91: floating-point execution unit,". *IBM Journal of Research and Development*, 11(1):34–53, Jan. 1967.

[2] R. E. Goldschmidt. "Applications of division by convergence,". Master's thesis, M.I.T., June 1964.

[3] H.M. Darley et. al. "Floating Point / Integer Processor with Divide and Square Root Functions,". *U.S. Patent No. 4,878,190*, Oct. 31, 1989.

[4] P. Markstein. "Computation of elementary functions on the IBM RISC system/6000 processor,". *IBM Journal of Research and Development*, 34(1):111–119, Jan. 1990.

[5] "IEEE standard for binary floating-point arithmetic, ANSI/IEEE Std 754-1985,". The Institute of Electrical and Electronic Engineers, Inc., New York, Aug. 1985.

[6] T. Taniguchi. "Apparatus for Performing Floating Point Arithmetic Operation and Rounding the Result thereof,". *U.S. Patent No. 5,212,661*, May 18, 1993.

[7] S. Waser and M. J. Flynn. *Introduction to Arithmetic for Digital Systems Designers.* CBS College Publishing, New York, 1982.