Improved Normalization Results for Digit On-line Arithmetic

Richard J. Zaccone[1] and Jesse L. Barlow[2]

Department of Computer Science
The Pennsylvania State University
University Park, PA 16802

In digit on-line arithmetic, operands are introduced a digit at a time. After the first few operand digits have been introduced, the result begins to appear a digit at a time. This feature of digit on-line arithmetic allows a significant amount of overlapping of arithmetic operations.

Digit on-line arithmetic can sometimes produce unnormalized results. This can present a problem for the divide and square root algorithms. If the divisor and radicand are highly unnormalized, these algorithms will not produce the correct results. Two advances in overcoming this problem are presented. First, several techniques for producing results that are closer to being normalized are developed. Second, it is shown that normalized results are not necessary for divide and square root to work properly. Combining these results yields algorithms that will always give the correct results.

1. Introduction

Digit on-line algorithms for performing computer arithmetic are a recent development [6, 8, 13, 14]. These algorithms are a radical departure from conventional techniques for performing computer arithmetic.

Digit on-line algorithms are iterative algorithms based on the continued sums/products algorithms of DeLugish [4]. What distinguishes these algorithms from conventional algorithms is that the operands are introduced a digit at a time, and the result is generated a digit at a time. At each iteration new digits of the operands are introduced. After a brief delay, result digits are generated, one per iteration.

With digit on-line arithmetic, one can construct pipeline architecures for solving recurrence relations. Digit on-line arithmetic allows these pipelines to be constructed with a high degree of modularity, and simple interconnections [9]. Also, a speedup factor between 2 and 16 over conventional arithmetic can be expected [7].

The algorithms for digit on-line arithmetic can sometimes produce unnormalized results. This has seriously limited their usefulness, since the divide and square root algorithms, as originally presented [9], require a normalized divisor and radicand. In this paper, it will be shown that a normalized divisor and radicand are not necessary for divide and square root to work properly. A condition which is much easier to satisfy will be shown to be sufficient. In the cases where this new condition is not met, methods for conditioning the operands so that it will be met are presented. The result is that the normalization problem in digit on-line arithmetic can always be avoided.

[1]Work supported by the Office of Naval Research under contract No. N0014-80-0517.

[2]Work supported by the National Science Foundation under contract No. DCR-8402363.

2. Notation

All of the digit on-line algorithms will use a redundant base 8 number system.[3] Thus all digits used to represent a number are from the digit set D, where

$$D = \{\bar{7}, \bar{6}, \bar{5}, \bar{4}, \bar{3}, \bar{2}, \bar{1}, 0, 1, 2, 3, 4, 5, 6, 7\},$$

and $\bar{7} = -7$, $\bar{6} = -6$, etc. (X_C, X_M) and (Y_C, Y_M) will be used to represent the floating point operands of an operation. X_C and Y_C are the characteristic portions of the floating point number, and X_M and Y_M are the mantissas. (Z_C, Z_M) will represent the floating point result. The following symbols will also be used.

Symbols Used

symbol	meaning
m_C	number of characteristic digits
m_M	number of mantissa digits
$\hat{x}_i$	i^{th} digit of X_C
x_i	i^{th} digit of X_M
$\hat{y}_i$	i^{th} digit of Y_C
y_i	i^{th} digit of Y_M

Then

$$X_C = \hat{x}_1\hat{x}_2\cdots\hat{x}_{m_C}$$
$$X_M = .x_1x_2\cdots x_{m_M}$$
$$Y_C = \hat{y}_1\hat{y}_2\cdots\hat{y}_{m_C}$$
$$Y_M = .y_1y_2\cdots y_{m_M}$$

where $\hat{x}_i, x_i, \hat{y}_i, y_i \in D$. Thus,

$$\text{value of } X_C = \sum_{i=0}^{m_C-1} \hat{x}_{m_C-i}8^i,$$

and

$$\text{value of } X_M = \sum_{i=1}^{m_M} x_i 8^{-i}.$$

Also, $X_M^{\downarrow i}$ will be used to represent the first i digits of X_M. I.e.

$$X_M^{\downarrow i} = .x_1\cdots x_i.$$

A floating point number (X_C, X_M) is *normalized* if

$$.1_8 \le |X_M| < 1.$$

[3]Other bases such as 2, 4 and 16 are possible, and the results presented here can easily be extended to these bases.

Note that because of the signed digits, it is not sufficient to check the first digit of the mantissa to determine if it is normalized. For example .12 is not normalized. In the worst case, it is not possible to determine if a number is unnormalized until the last mantissa digit has been examined.

Digit on-line algorithms are iterative, and at each iteration, they expect to be provided with the next digit of the operands. Thus, if an algorithm requires two operands, then $\hat{x}_j$ and $\hat{y}_j$ must be provided at the j^{th} characteristic iteration. Likewise, x_j and y_j must be provided during the j^{th} mantissa iteration.

3. Floating Point Division - RDIV

Let (Y_C, Y_M) be the floating point dividend, (X_C, X_M) the floating point divisor, and (Z_C, Z_M) the floating point quotient. Algorithm RDIV computes successive approximations to the floating point quotient.

Following is the algorithm RDIV for computing the floating point quotient.

Characteristic Initialization

$$A_{-3} = 0$$

Characteristic Iterations $j = 1, 2, \ldots, m_C$

$$A_{j-3} = A_{j-4} + 8^{m_C - j}(\hat{y}_j - \hat{x}_j)$$

Characteristic Trailing Iteration $j = m_C + 1$

$$A_{j-3} = A_{j-4} + 1$$

Characteristic Trailing Iterations $j = m_C + 2, m_C + 3$

$$A_{j-3} = A_{j-4}$$

Mantissa Initialization

$$U_0 = 0$$

$$Q_{-3} = 0$$

$$S_0 = 1$$

Mantissa Iteration $j = 1$

$$U_j = 8^1 U_{j-1} + 8^{-2} S_{j-1} x_j$$

$$Q_{j-3} = Q_{j-4} + 8^{-1} y_j$$

$$S_1 = S_0$$

Mantissa Iteration $j = 2$

$$t_j = 8^1 U_{j-1} + 8^{-2} S_{j-1} x_j$$

$$s_j = \mathrm{DIV}_j(t_j)$$

$$U_j = t_j s_j - 1$$

$$Q_{j-3} = (Q_{j-4} + 8^{-j} y_j) s_j$$

$$S_j = s_j S_{j-1}$$

Mantissa Iterations $j = 3, 4, \ldots, m_M$

$$t_j = 8^1 U_{j-1} + 8^{-2} S_{j-1} x_j$$

$$s_j = \mathrm{DIV}_j(t_j)$$

$$U_j = t_j(1 + 8^{2-j} s_j) + s_j$$

$$Q_{j-3} = (Q_{j-4} + 8^{-j} S_{j-1} y_j)(1 + 8^{2-j} s_j)$$

$$S_j = S_{j-1}(1 + 8^{2-j} s_j)$$

Mantissa Trailing Iterations $j = m_M + 1, m_M + 2, m_M + 3$

$$t_j = 8^1 U_{j-1}$$

$$s_j = \mathrm{DIV}_j(t_j)$$

$$U_j = t_j(1 + 8^{2-j} s_j) + s_j$$

$$Q_{j-3} = Q_{j-4}(1 + 8^{2-j} s_j)$$

$$S_j = S_{j-1}(1 + 8^{2-j} s_j)$$

The definitions for DIV_j, $j = 1, \ldots, m_M + 3$ can be found in [16].

4. One Step Normalization

Owens [9] describes an algorithm called "one step normalization" that can be incorporated into each floating point operation to bring the result one step closer to being normalized. It does this by waiting until the first approximation to the result mantissa has been generated, before producing the last characteristic digit. If the mantissa can be shifted left one place (i.e. if the first mantissa digit produced would have been a zero), the last characteristic digit is decreased by one, and the mantissa is shifted.

For example, in the multiplication algorithm, let P_1 be the first mantissa approximation. P_1 is an approximation to the product which has been produced after examining only the first two digits of the operands. One step normalization says that the mantissa may be shifted left and the characteristic adjusted if $|P_1| < .04_8$. If $|P_1| \geq .04_8$, the result is produced as usual.

The are many instances where $|P_1| \geq .04_8$ but it is possible to shift the mantissa anyway. One step normalization will not detect these cases, and the following sections suggest some improvements that allow many more shifts to occur.

5. Multiplication

Let (Z_C, Z_M) be the result of a floating point multiplication, where (X_C, X_M) and (Y_C, Y_M) are the operands. By definition,

$$Z_M = X_M Y_M. \tag{1}$$

If one step normalization is used to determine if a shift is possible, the on-line delay increases by one, and a shift left is performed if

$$|X_M^2 Y_M^2| < .04_8. \tag{2}$$

This test will miss many of the instances when a shift can be performed. The following discussion will show how it can be improved.

From (1) it follows that

$$\begin{aligned} |Z_M| &< (|X_M^1| + 8^{-1})(|Y_M^1| + 8^{-1}) \\ &= |X_M^1 Y_M^1| + 8^{-1}(|X_M^1| + |Y_M^1|) + 8^{-2}. \end{aligned}$$

Therefore, it is possible to shift Z_M left one place after the first digit of each operand has been introduced if

$$|X_M^1 Y_M^1| + 8^{-1}(|X_M^1| + |y_M^1|) + 8^{-2} \leq .1_8, \tag{3}$$

or equivalently if

$$|X_M^1 Y_M^1| + 8^{-1}(|X_M^1| + |Y_M^1|) \leq .07_8. \tag{4}$$

This test is easy to compute, and since only the first digit of each operand is needed, it does not increase the on-line delay.

Another benefit is that this method can also allow a shift of two places. If the right hand side of (3) is replaced with $.01_8$, then this gives the test necessary to check if a shift of two is possible. A shift of two is possible if

$$X_M^1 = Y_M^1 = 0. \tag{5}$$

For purposes of comparison, consider the best that can be done after examining two digits of each operand. From (1),

$$|Z_M| < (|X_M^2| + 8^{-2})(|Y_M^2| + 8^{-2}),$$

and therefore, a shift left is possible if

$$(|X_M^2| + 8^{-2})(|Y_M^2| + 8^{-2}) \leq .1_8,$$

or equivalently if

$$|X_M^2 Y_M^2| + 8^{-2}(|X_M^2| + |Y_M^2|) \leq .0777_8. \tag{6}$$

This test will increase the on-line delay by one, and gives optimal results if only two digits are being examined. However, the number of bits involved in the comparison (6) make this test prohibitively expensive and beyond current technology. Therefore, this test will serve to gauge the effectiveness of the other two methods presented in this section.

The following table summarizes the performance of one-step normalization (2), test (4), and test (6). Each of these tests was applied to all positive combinations for X_M^2 and Y_M^2 (i.e. all combinations of X_M^2 and Y_M^2 were assumed to be equally likely). "Percent Allowed" shows how often a particular method allowed a shift.

Improvements to One Step Normalization for Multiply

Test	Percent Allowed	On-line Delay Increases?
One step normalization	25%	Yes
Test (4)	31%	No
Test (6)	37%	Yes

Test (4) is clearly superior to one step normalization, since it can detect when a shift is possible more often, and it does not increase the on-line delay. It has the added benefit of allowing two shifts in a special case (5). Although test (6) is not feasible to implement, it shows that by looking at only one operand digit, test (4) can perform almost as well as the best possible case when looking at two operand digits.

6. Division

Let (Z_C, Z_M) be the result of a floating point division, where (Y_C, Y_M) is the numerator and (X_C, X_M) is the denominator.

The divide algorithm generates its first approximation to the quotient after the fourth operand digits have been introduced. Call this approximation Q_1. According to one step normalization, a shift is possible if

$$|Q_1| < .4_8. \tag{7}$$

The rest of this section will describe how the test (7) can be improved.

By definition,

$$|Z_M| = \frac{|Y_M|}{|X_M|} 8^{-1}, \tag{8}$$

and therefore

$$|Z_M| < \frac{|Y_M^2| + 8^{-2}}{|X_M^2| - 8^{-2}} 8^{-1}$$

Thus, after the first two operand digits have been introduced, Z_M can be shifted left one place if

$$\frac{|Y_M^2| + 8^{-2}}{|X_M^2| - 8^{-2}} 8^{-1} \leq .1_8,$$

or equivalently if

$$|X_M^2| - |Y_M^2| \geq .02_8. \tag{9}$$

Also, Z_M can be shifted left two places if

$$\frac{|Y_M^2| + 8^{-2}}{|X_M^2| - 8^{-2}} 8^{-1} \leq .01_8,$$

or equivalently if

$$8^{-1}|X_M^2| - |Y_M^2| \geq .011_8. \tag{10}$$

Combining (9) and (10) yields the following shift algorithm for divisions.

New Shift Algorithm for Division

if $8^{-1}|X_M^2| - |Y_M^2| \geq .011_8$ **then**
    shift = 2
else if $|X_M^2| - |Y_M^2| \geq .02_8$ **then**
    shift = 1
else
    shift = 0

For comparison, consider using a similar technique after the first four operand digits have been introduced. Using (8),

$$|Z_M| < \frac{|Y_M^4| + 8^{-4}}{|X_M^4| - 8^{-4}} 8^{-1},$$

and so a shift can be performed if

$$|X_M^4| - |Y_M^4| \geq .0002_8. \tag{11}$$

Similarly, a shift of two is possible if

$$8^{-1}|X_M^4| - |Y_M^4| \geq .00011_8. \tag{12}$$

This method is too expensive to practically compute, but serves as a good benchmark since no better than this can be done after the first four operand digits have been introduced.

The following table summarizes the performance of one step normalization (7), new shift algorithm for division, and the tests given by (11) and (12). "Percent Allowed" indicates how often a shift of one or more was allowed "Two Shifts" indicates how often two shifts were allowed. The observations were made by considering all positive possibilities for X_M^4 and Y_M^4. The decisions for the new shift algorithm for division were made by considering only the first 2 digits of X_M^4 and Y_M^4.

Improvements to One Step Normalization for Divide

Test	Percent Allowed	Two Shifts	On-line Delay Increases?
One step normalization	28%	0%	Yes
New shift algorithm for division	54%	6%	No
Tests (11) and (12)	56%	6%	Yes

The new shift algorithm for division clearly performs better than one step normalization. Also, by only looking at the first two operand digits, it performs almost as well as the optimal algorithm which considers four operand digits, and it does not increase the on-line delay.

7. Addition

Let (Z_C,Z_M) be the result of a floating point addition, where (X_C,X_M) and (Y_C,Y_M) are the operands. $\tilde{X}_M$ and $\tilde{Y}_M$ are equal to X_M and Y_M after they have been shifted so that X_C and Y_C are equal. If one step normalization is used, the on-line delay increases by one, and a shift is possible when

$$|\tilde{X}_M^2 + \tilde{Y}_M^2| < .4_8. \tag{13}$$

The following analysis will show how (13) can be improved.

By definition,

$$Z_M = 8^{-1}(\tilde{X}_M + \tilde{Y}_M),$$

and therefore,

$$|Z_M| < 8^{-1}(|\tilde{X}_M^1 + \tilde{Y}_M^1| + 2{\cdot}8^{-1}).$$

This suggests that Z_M can be shifted left if

$$8^{-1}(|\tilde{X}_M^1 + \tilde{Y}_M^1| + 2{\cdot}8^{-1}) \leq .1_8,$$

or equivalently if

$$|\tilde{X}_M^1 + \tilde{Y}_M^1| \leq .6_8. \tag{14}$$

Since (14) requires only the first digit of each operand to decide if a shift is possible, the on-line delay does not increase.

If however, an increase of one in the on-line delay is tolerable, the following technique can be used. Note that

$$|Z_M| < 8^{-1}(|\tilde{X}_M^2 + \tilde{Y}_M^2| + 2{\cdot}8^{-2}),$$

and a shift is therefore possible if

$$8^{-1}(|\tilde{X}_M^2 + \tilde{Y}_M^2| + 2{\cdot}8^{-2}) \leq .1_8,$$

which reduces to

$$|\tilde{X}_M^2 + \tilde{Y}_M^2| \leq .76_8. \tag{15}$$

The following table summarizes the performance of one step normalization (13), test (14), and test (15). To obtain these results, all possible values for $\tilde{X}_M^2$ and $\tilde{Y}_M^2$ were tested to see when a shift was possible. The "Percent Allowed" is the percent of the total number of observations where a shift was permitted.

Improvements to One Step Normalization for Add

Test	Percent Allowed	On-line Delay Increases?
One step normalization (13)	47%	Yes
Test (14)	68%	No
Test (15)	77%	Yes

The performance of test (14) and test (15) exceeds that of one step normalization. Note that although test (14) performs slightly worse than test (15), it has the advantage of not increasing the on-line delay.

8. Post Normalization

This section describes an algorithm which can be appended to any one of the digit on-line algorithms, and which will bring the result one step closer to being normalized. The advantage of this algorithm is that it can be appended more than once to a digit on-line algorithm and thus bring a result even closer to being completely normalized.

Let (Z_C,Z_M) be the result of a floating point operation, and let $\hat{z}_i$ be the i^{th} characteristic digit, and z_i be the i^{th} mantissa digit. If the post normalization algorithm is supplied with the digits of (Z_C,Z_M) in an on-line fashion, it will produce (Z_C',Z_M'), where (Z_C',Z_M') is one step closer to being normalized.

The post normalization algorithm examines z_1 to decide if it is possible to decrease Z_C by one and then shift Z_M left one place. If $z_1 = 0$, then the shift and decrement takes place. If $z_1 \neq 0$, then no shift or decrement takes place. Note that because of the redundant arithmetic being used, there are instances where $|Z_M| < .1_8$ and it is not possible to shift Z_M. For instance, suppose that $Z_M^8 = .17_8$. The post normalization algorithm will not shift Z_M left since $z_1 = 1$. However, since $Z_M < .1_8$, it is not normalized. The post normalization algorithm handles this case by changing Z_M^8 to $.01_8$. Then, if it is applied again, it will perform the shift. Note that each application of the post normalization algorithm increases the on-line delay by one.

Post Normalization

Characteristic Iteration $j = 1$

$E_j = 8^{-1}\hat{z}_j$

Characteristic Iterations $j = 2, \ldots, m_C$

$t_j = 8E_{j-1} + 8^{-1}\hat{z}_j$

$s_j = \text{DIS}(t_j)$

$E_j = t_j - s_j$

$\hat{z}'_{j-1} = s_j$

Mantissa Iteration $j = 1$

$$e = \begin{cases} 1 & \text{if } z_j = 0 \\ 0 & \text{otherwise} \end{cases}$$

$t_j = 8E_{m_C} - e$

$s_j = \text{DIS}(t_j)$

$\hat{z}'_{m_C} = s_j$

$F_j = 8^{-1}z_j$

Mantissa Iterations $j = 2, \ldots, m_M$

$$t_j = 8^e (8 F_{j-1} + 8^{-1} z_j)$$

$$s_j = \mathrm{NORM}_j(t_j)$$

$$z'_{j-1} = s_j$$

$$F_j = t_j - s_j$$

Mantissa Trailing Iteration $j = m_M + 1$

$$t_j = 8^{e+1} F_{j-1}$$

$$s_j = \mathrm{NORM}_j(t_j)$$

$$z'_{j-1} = s_j$$

$$\mathrm{NORM}_2(x) = \begin{cases} 7 & 7 \le x \\ 6 & 6 \le x < 7 \\ 5 & 5 \le x < 6 \\ 4 & 4 \le x < 5 \\ 3 & 3 \le x < 4 \\ 2 & 2 \le x < 3 \\ 1 & 1 \le x < 2 \\ 0 & -1 < x < 1 \\ \bar{1} & -2 < x \le -1 \\ \bar{2} & -3 < x \le -2 \\ \bar{3} & -4 < x \le -3 \\ \bar{4} & -5 < x \le -4 \\ \bar{5} & -6 < x \le -5 \\ \bar{6} & -7 < x \le -6 \\ \bar{7} & -10_8 < x \end{cases}$$

$$\mathrm{NORM}_j(x) \quad j = 3, \ldots, m_M+1 = \begin{cases} 7 & 7 \le x \\ 6 & 6 \le x < 7 \\ 5 & 5 \le x < 6 \\ 4 & 4 \le x < 5 \\ 3 & 3 \le x < 4 \\ 2 & 2 \le x < 3 \\ 1 & 0 < x < 2 \\ 0 & x = 0 \\ \bar{1} & -2 < x < 0 \\ \bar{2} & -3 < x \le -2 \\ \bar{3} & -4 < x \le -3 \\ \bar{4} & -5 < x \le -4 \\ \bar{5} & -6 < x \le -5 \\ \bar{6} & -7 < x \le -6 \\ \bar{7} & -10_8 < x \end{cases}$$

Since the only function of the post normalization algorithm is to try and normalize its input, it is necessary to verify that it has no other unexpected effects. In particular, one would like to verify that (Z_C, Z_M) and (Z'_C, Z'_M) will always have the same value. It is fairly easy to see that the characteristic iterations of the post normalization algorithm work correctly. The following lemma and theorem will show that the post normalization algorithm does in fact preserve the value of (Z_C, Z_M).

Lemma 16. Let $z_1, z_2, \ldots, z_{m_M}$ be the mantissa digits which are input to the post normalization algorithm, and $z'_1, z'_2, \ldots, z'_{m_M}$ be the mantissa digits output by the post normalization algorithm. Let $a_j = 8 F_j$. For $j = 2, \ldots, m_M - 1$, where F_j is as defined in the mantissa iterations of the post normalization algorithm. Then $|a_j| \le 7$, and $.z_1 z_2 \cdots z_j = .z'_1 z'_2 \cdots z'_{j-1} a_j$.

Proof. By induction on j. Assume that $z_1 \neq 0$ and hence $e = 0$. The proof for when $z_1 = 0$ is similar.

Base $j = 2$.

By definition, $t_2 = z_1.z_2$. If z_1 and z_2 are of the same sign, then from the definition of NORM_2, $z'_1 = z_1$. Hence $F_2 = .z_2$, $a_2 = z_2$, and $|a_2| \le 7$. So, $.z_1 z_2 = .z'_1 a_2$.

Suppose that z_1 and z_2 have different signs. Without loss of generality, assume $z_1 \ge 1$ and $z_2 \le -1$. By definition, $t_2 = z_1.z_2$, and from the definition of NORM_2, $z'_1 = z_1 - 1$. Therefore,

$$\begin{aligned} F_2 &= z_1.z_2 - (z_1 - 1) \\ &= .z_2 + 1. \end{aligned}$$

Thus $a_2 = 8 + z_2$, which implies $|a_2| \le 7$. Since

$$z_1 8^{-1} + z_2 8^{-2} = (z_1 - 1) 8^{-1} + (8 + z_2) 8^{-2},$$

then $.z_1 z_2 = z'_1 a_2$.

Induction Hypothesis

$$.z_1 z_2 \cdots z_j = .z'_1 z'_2 \cdots z'_{j-1} a_j \qquad \text{where } |a_j| \le 7.$$

Induction Step $j = 3, \ldots, m_M+1$

Suppose that either $a_j \ge 1$ and $z_{j+1} \ge 1$, or $a_j \le -1$ and $z_{j+1} \le -1$ By definition $t_{j+1} = a_j.z_{j+1}$. Therefore, from the definition of NORM_j $z'_j = a_j$, and thus by the induction hypothesis

$$.z_1 \cdots z_j = .z'_1 \cdots z'_j. \tag{17}$$

Also, from the definition of F_{j+1}, $F_{j+1} = .z_{j+1}$, and therefore,

$$a_{j+1} = z_{j+1} \qquad \text{where } |a_{j+1}| \le 7. \tag{18}$$

Thus, using (17) and (18)

$$.z_1 \cdots z_{j+1} = .z'_1 \cdots z'_j a_{j+1}.$$

Suppose $a_j = 0$ and $z_{j+1} \ge 1$. Then $t_{j+1} = 0.z_{j+1}$. Then from the definition of NORM_j, $z'_j = 1$ and from the induction hypothesis

$$.z_1 \cdots z_j = .z'_1 \cdots z'_j - 8^{-j}. \tag{19}$$

From the definition of F_{j+1} $F_{j+1} = 0.z_{j+1} - 1$, and therefore

$$a_{j+1} = z_{j+1} - 8 \qquad \text{where } |a_{j+1}| \le 7. \tag{20}$$

Using (19) and (20) yields

$$.z_1 z_2 \cdots z_{j+1} - 8^{-j} = .z'_1 \cdots z'_j - 8^{-j} + 8^{-j-1} a_{j+1},$$

and hence

$$.z_1 z_2 \cdots z_{j+1} = .z'_1 \cdots z'_j a_{j+1}.$$

The cases for $a_j = 0$ and $z_{j+1} \le -1$, $a_j = 0$ and $z_{j+1} = 0$, $a_j \le -1$ and $z_{j+1} = 0$, $a_j \le -1$ and $z_{j+1} \ge 1$, $a_j \ge 1$ and $z_{j+1} \le -1$, and $a_j \ge 1$ and $z_{j+1} = 0$ are similar. □

Let $z_1, z_2, \ldots, z_{m_M}$ be the mantissa digits which are input to the post normalization algorithm, and $z'_1, z'_2, \ldots, z'_{m_M}$ be the mantissa digits output by the post normalization algorithm. The following theorem shows that the value of the mantissa is not altered by the post normalization algorithm.

Theorem 21.

$$.z_1 \cdots z_{m_M} = .z'_1 \cdots z'_{m_M}.$$

Proof. By lemma 16,

$$.z_1 z_2 \cdots z_{m_M} = .z'_1 z'_2 \cdots z'_{m_M - 1} a_{m_M}.$$

But, by definition

$$\begin{aligned} z'_{m_M} &= \mathrm{NORM}_j(t_{m_M}) \\ &= \mathrm{NORM}_j(8 F_{m_M}) \\ &= \mathrm{NORM}_j(a_{m_M}) \\ &= a_{m_M}, \end{aligned}$$

where $|a_{m_M}| \le 7$. □

The improvements suggested for multiply, add, and divide work significantly better than one step normalization. Also, they are easy to implement and they do not increase the on-line delay. Therefore, they should probably be incorporated into the formulation of the digit on-line algorithms.

9. A Closer Look at Division

Owens [9] says that a normalized divisor is a sufficient condition for RDIV to give correct results. This condition can be relaxed considerably. It will be shown that X_M^2 (the first two digits of the mantissa of the divisor) must only satisfy

$$.10_8 \le |X_M^2| \le .77_8$$

so that RDIV gives correct results. Since the divisor does not need to be normalized, it is no longer necessary to wait until the last mantissa digit has been seen to determine if a divide will be successful. This can now be determined after the second mantissa digit.

The characteristic iterations are fairly straightforward since they compute $Y_C - X_C + 1$. However, the mantissa iterations are a little bit more complicated. Here, the object is to compute $Z_M = Y_M / X_M$. (To make the discussion simpler, the 8^{-1} factor will be ignored). If d_j, $j = 2, \ldots, m_M + 3$ are chosen such that

$$X_M \prod_{j=2}^{m_M+3} d_j \approx 1,$$

then

$$\frac{Y_M}{X_M} = \frac{Y_M \prod_{j=2}^{m_M+3} d_j}{X_M \prod_{j=2}^{m_M+3} d_j} \approx Y_M \prod_{j=2}^{m_M+3} d_j.$$

The following lemma shows that $S_M + 3$ is the same as $\prod_{j=2}^{m_M+3} d_j$ in the above discussion.

Lemma 22. Let s_j and S_j be as in algorithm RDIV. By definition, $s_j = \mathrm{DIV}_j(t_j)$, $j = 2, \ldots, m_M + 3$. Let $d_j = (1 + 8^{2-j} s_j)$, $j = 3, \ldots, m_M + 3$ and $d_2 = s_2$. Then

$$\begin{aligned} S_j &= \prod_{i=2}^{j} d_i \\ &= S_{j-1} d_j, \end{aligned}$$

where $S_1 = 1$.

Proof. $S_2 = d_2$ since by hypothesis $s_2 = d_2$ and by definition $S_1 = 1$. Also by definition $S_j = S_{j-1}(1 + 8^{2-j} s_j)$. □

Lemma 23 establishes the definition of U_j. A direct result of this lemma is that if U_j is "small" (this will be defined later), $S_j \approx 1/X_M^j$.

Thus, the value of U_j can be used as a measure of whether S_j is a good approximation to $1/X_M^j$. Note also that t_j is an updated version of $8^1 U_{j-1}$ when $2 \le j \le m_M$.

Lemma 23. For $2 \le j \le m_M + 3$, $U_j = 8^{j-2}(S_j X_M^j - 1)$, where $X_M^j = X_M$ if $j \ge m_M$. Also, for $3 \le j \le m_M$, $t_j = 8^{j-2}(S_{j-1} X_M^j - 1)$, where $X_M^j = X_M$ if $j \ge m_M$.

Proof. By induction on j.
Base. $j = 2$. By definition $U_2 = t_2 s_2 - 1$. It can easily be seen that $t_2 = X_M^2$ and $s_2 = S_2$. Thus $U_2 = S_2 X_M^2 - 1$.

Induction Step. By definition

$$\begin{aligned} t_j &= 8^1 U_{j-1} + 8^{-2} S_{j-1} x_j \\ &= 8^{j-2}(S_{j-1} X_M^{j-1} - 1) + 8^{-2} S_{j-1} x_j \\ &= 8^{j-2} S_{j-1}(X_M^{j-1} + 8^{-j} x_j) - 8^{j-2} \\ &= 8^{j-2}(S_{j-1} X_M^j - 1). \end{aligned}$$

The proof for U_j is similar.

$$\begin{aligned} U_j &= t_j(1 + 8^{2-j} s_j) + s_j \\ &= 8^{j-2}(S_{j-1} X_M^j - 1)(1 + 8^{2-j} s_j) + s_j \\ &= 8^{j-2} X_M^j S_{j-1}(1 + 8^{2-j} s_j) - 8^{j-2} \\ &= 8^{j-2}(X_M^j S_j - 1). \end{aligned}$$

□

Lemma 24. For $2 \le j \le m_M + 3$ $Q_{j-3} = Y_M^j S_j$, where $Y_M^j = Y_M$ for $j \ge m_M$.

Proof. By induction on j.
Base. $j = 2$. By definition,

$$Q_{-1} = (Q_{-2} + 8^{-2} y_2) s_2.$$

But $Q_{-2} = Y_M^2$ and $s_2 = S_2$. Therefore, $Q_{-1} = Y_M^2 S_2$.

Induction Step. By definition,

$$\begin{aligned} Q_{j-3} &= (Q_{j-4} + 8^{-j} S_{j-1} y_j)(1 + 8^{2-j} s_j) \\ &= (Y_M^{j-1} S_{j-1} + 8^{-j} S_{j-1} y_j)(1 + 8^{2-j} s_j) \\ &= (Y_M^{j-1} + 8^{-j} y_j) S_{j-1}(1 + 8^{2-j} s_j) \\ &= Y_M^j S_j. \end{aligned}$$

□

By Lemma 23, if U_j is "small", $S_j \approx 1/X_M^j$, and therefore by Lemma 24 $Q_{m_M} \approx Y_M/X_M$. This will be formalized later. The following three lemmas can easily be verified by writing a small program. These lemmas help to establish bounds on U_j (Remember that U_j can be used to estimate how close S_j is to the reciprocal of X_M^j). Lemma 28 uses these results to establish a bound for U_j when X_M^2 (the first two digits of the mantissa of the divisor) satisfies (29). t_j, S_j, and s_j are as defined in algorithm RDIV.

Lemma 25. By definition, $t_2 = X_M^2$, $s_2 = \mathrm{DIV}_2(t_2)$ and $U_2 = t_2 s_2 - 1$. If $.10_8 \le |t_2| \le .77_8$, then $|U_2| \le .25_8$.

Lemma 26. By definition, $s_3 = \mathrm{DIV}_3(t_3)$ and $U_3 = t_3(1 + 8^{-1}s_3) + s_3$. If $|t_3| \le 3.31_8$, then $|U_3| \le .5_8$.

Lemma 27. By definition, if $4 \le j \le m_M+3$, then $s_j = \mathrm{DIV}_j(t_j)$ and $U_j = t_j(1 + 8^{2-j}s_j) + s_j$. If $|t_j| \le 5.7_8$, then $|U_j| \le .441_8$.

Lemma 28. If $.10_8 \le |X_M^2| \le .77_8$, ($X_M^2$ is the first two digits of the mantissa of the divisor) then

$$|U_j| < 0.5_8 \quad j = 2, \ldots, m_M+3. \tag{29}$$

Thus, if X_M^2 satisfies (29), S_j will be a good approximation to $1/X_M^j$.

Proof. By induction on j.
Base. $j = 2$. If $.10_8 \le |X_M^2| \le .77_8$, then $.10_8 \le |t_2| \le .77_8$, since by definition $t_2 = X_M^2$. By lemma 25, $|U_2| \le .25_8$.
For $j = 3$, $t_3 = 8^1 U_2 + 8^{-2} S_2 x_3$, which implies that $|t_3| \le 3.4_8$ since $U_2 \le .25_8$, $|S_2| \le 7$, and $|x_3| \le 7$. By lemma 26, $|U_3| \le .5_8$.

Induction Step. By definition,

$$t_j = 8^1 U_{j-1} + 8^{-2} S_{j-1} x_j,$$

and hence

$$\begin{aligned} |t_j| &\le |8^1 U_{j-1}| + 8^{-2} |S_{j-1} x_j| \\ &\le 5 + 8^{-2} |S_{j-1} x_j| \\ &\le 5.7_8. \end{aligned}$$

Therefore, by lemma 27, $|U_j| \le .441_8$. □

The following theorem establishes a condition on X_M (the mantissa of the divisor) which is sufficient to cause the divide algorithm to produce the correct quotient. Note that this is a much stronger result than just requiring that X_M be normalized. Condition (31) requires that the first two digits of the mantissa must fall within a certain range. This is stronger than previous results [9] since normalized divisors are a small subset of the values allowed by (31). Previously there were many instances when every mantissa digit of the divisor had to be examined so it could be determined if it was normalized.

Theorem 30. If

$$.10_8 \le |X_M^2| \le .77_8, \tag{31}$$

then

$$\left| \frac{Y_M}{X_M} - Q_{m_M} \right| < 6.2_8 \times 8^{-(m_M+1)},$$

where Q_{m_M} is the final approximation to $\frac{Y_M}{X_M}$ produced by algorithm RDIV. Therefore, if the first two digits of the mantissa of the divisor satisfy (31), then RDIV produces an accurate approximation the the true quotient $\frac{Y_M}{X_M}$.

Proof. Using lemma 24,

$$\begin{aligned} \left| \frac{Y_M}{X_M} - Q_{m_M} \right| &= \left| \frac{Y_M}{X_M} - Y_M S_{m_M+3} \right| \\ &\le |Y_M| \left| \frac{1}{X_M} - S_{m_M+3} \right|. \end{aligned}$$

Then by lemma 23,

$$\left| \frac{Y_M}{X_M} - Q_{m_M} \right| \le |Y_M| \left| \frac{1}{X_M} U_{m_M+3} 8^{-(m_M+1)} \right|.$$

By definition, $|Y_M| < 1$, $\frac{1}{X_M} < 12_8$, and by lemma 28, $|U_{m_M+3}| < .5_8$. Thus

$$\left| \frac{Y_M}{X_M} - Q_{m_M} \right| < 6.2_8 \times 8^{-(m_M+1)}.$$ □

The following result can also be proven for the square root algorithm. The proof is precisely analogous to that for the division algorithm. It is given in [17].

Theorem 32. If

$$.10_8 \le X_M^2 \le .77_8, \tag{33}$$

then

$$|\sqrt{X_M} - R_{m_M}| < 5.62_8 \times 8^{-(m_M+1)}$$

where R_{m_M} is the final approximation to $\sqrt{X_M}$ produced by algorithm RSQR from [9]. Therefore, if the first two digits of the mantissa of the radicand satisfy (33), then RSQR produces an accurate approximation the the true square root $\sqrt{X_M}$.

Note that this is a much stronger result than that given by Owens [9], since theorem 32 does not require that X_M be normalized, but only that (33) be satisfied.

10. Conclusion

The improvements suggested for multiply, add, and divide work significantly better than one step normalization. Also, they are easy to implement and they do not increase the on-line delay. Therefore, they should probably be incorporated into the formulation of the digit on-line algorithms.

According to Owens [9], a normalized divisor is a sufficient condition for RDIV to give correct results. Therefore, it was necessary to make sure that the divisor was normalized to insure a correct result. In the worst case, this means that if the mantissa of the divisor is $.10\cdots01$, where the 1 occurs in the last mantissa digit, the divide algorithm won't know that it has an unnormalized divisor until it has seen every digit. Thus a number that was only slightly unnormalized has completely disrupted a divide operation. The only way that the problem could be avoided with certainty was to increase the on-line delay to m_M (the number of mantissa digits). However, all advantages over conventional arithmetic have then been lost.

With the results presented in the preceding sections, this problem can be eliminated. In order to do so, it is necessary to use the post normalization algorithm that was presented in an earlier section of this paper. This

algorithm, which has an on-line delay of one, takes the result of a floating point operation, (Z_C,Z_M), as its input. It produces (Z'_C,Z'_M) where (Z'_C,Z'_M) is one step closer to being normalized.

The post normalization algorithm and the results presented in the previous two sections can now be combined to produce a new approach to the normalization problem.

This approach involves a pre-conditioning for all divisors and radicands. As shown in the previous two sections, it is only necessary to look at the first two digits of the mantissas of these values. If conditions (31) for divide and (33) for square root are satisfied, then the operation can proceed without any pre-conditioning, since it has been shown that these are sufficient conditions for divide and square root to work properly. If these conditions are not satisfied, then the post normalization algorithm can be used as many times as necessary.

The fact that the post normalization algorithm is digit on-line can be used to make this easier. The result digits of one application of the algorithm can be fed back into the algorithm as they are produced. The net effect is that a new version of the operand is available at each time step. That is, at each time step the operand is brought closer to satisfying either (31) or (33). As soon as the proper condition is satisfied, the divide or square root can continue.

Since the divide and square root algorithms will now have variable on-line delays, this will affect other operations in the pipeline. Results will need to be buffered and some timing problems may result. However, these are implementation details that shouldn't be difficult to solve.

11. References

1. R. L. Ashenhurst and N. Metropolis, Unnormalized Floating Point Arithmetic, *J. ACM 6*, (1959), pp. 415-428.
2. R. L. Ashenhurst, Experimental Invesigation of Unnormalized Arithmetic, in *Error in Digital Computation*, vol. 2, L. B. Rall (ed.), John Wiley & Sons, New York, NY, 1965, pp. 3-37.
3. D. E. Atkins, Introduction to the Role of Redundancy in Computer Arithmetic, *Computer 8*, 6 (June 1975), pp. 76-84.
4. B. G. DeLugish, A Class of Algorithms for Automatic Evaluation of Certain Elementary Functions in a Binary Computer, Ph.D. Diss., Report 399, Dept. of Computer Science, University of Illinois, Urbana, IL, 1970.
5. M. D. Ercegovac, Radix-16 Evaluation of Certain Elementary Functions, *IEEE Trans. on Computers C-22*, 6 (June 1973), pp. 561-566.
6. M. D. Ercegovac, An On-line Square Rooting Algorithm, *Proc. of fourth Symp. on Computer Arithmetic* Santa Monica, CA, Oct. 1978.
7. M. D. Ercegovac and A. L. Grnarov, On the Performance of On-line Arithmetic, *Proc. of the 1980 Inter. Conf. on Parallel Processing*, , 1980, pp. 55-61.
8. A. L. Grnarov and M. D. Ercegovac, An Algorithm for On-Line Normalization, Quarterly Report, UCLA Computer Science Dept., July 1979.
9. R. M. Owens, *Digit On-Line Algorithms for Pipeline Architectures*, Ph.D. Diss., Dept. of Computer Science, The Pennsylvania State Univ., University Park, PA, 1980.
10. R. M. Owens, Compound Algorithms for Digit On-Line Arithmetic, *Proc. 5th Symp. on Computer Arithmetic*, Ann Arbor, MI, May 1981, pp. 64-71.
11. P. H. Sterbenz, *Floating-Point Computation*, Prentice Hall, Englewood Cliffs, NJ, 1974.
12. K. S. Trivedi and J. G. Rusnak, Higher Radix On-line Division, *Proc. of the Fourth Symp. on Computer Arithmetic*, Santa Monica, CA, Oct. 1978.
13. O. Watanuki and M. D. Ercegovac, Floating-Point On-Line Arithmetic: Error Analysis, *Proc. 5th Symp. on Computer Arithmetic*, Ann Arbor, MI, May 1981, pp. 87-91.
14. O. Watanuki and M. D. Ercegovac, Floating-Point On-Line Arithmetic: Algorithms, *Proc. 5th Symp. on Computer Arithmetic*, Ann Arbor, MI, May 1981, pp. 81-86.
15. J. H. Wilkinson, *Rounding Errors in Algebraic Processes*, Prentice Hall, Englewood Cliffs, NJ, 1963.
16. R. J. Zaccone and J. L. Barlow, Improved Normalization Results for Digit Online Arithmetic, CS-84-16, Dept. of Computer Science, The Pennsylvania State Univ., University Park, PA, Oct. 1984.
17. R. J. Zaccone, *Numerical Properties of Digit On-line Arithmetic*, Ph.D. Diss., Dept. of Computer Science, The Pennsylvania State Univ., University Park, PA, 1984.