Note on Triple-Precision Floating-Point Arithmetic with 132-Bit Numbers

Yasuhiko Ikebe
The University of Texas, Austin, Texas

In a recent paper, Gregory and Raney described a technique for double-precision floating-point arithmetic. A similar technique can be developed for triple-precision floating-point arithmetic and it is the purpose of this note to describe this technique. Only the multiplication and the division algorithms are described, since the addition-subtraction algorithm can be obtained by a trivial modification of the algorithm in Gregory's and Raney's paper.

1. Introduction

It is assumed, as Gregory and Raney [1] did, that we have a machine with a word length of 48 bits and that each triple-precision floating-point number is in the "standard" format, namely, the 12-bit sign-and-exponent (of which the leftmost bit represents the sign of the number, the second bit the sign of the exponent and the next 10 bits the magnitude of the exponent) plus a normalized 132-bit (=36 + 48 + 48) mantissa. We further assume that we are using one's complement arithmetic. Hence, the negative of each number is obtained by a bit-by-bit complementing of the binary representation.

2. Multiplication

Let A and B be triple-precision floating-point numbers. It is desired to compute the product AB in the same format. To do this the exponent and the mantissa are computed separately. Since the computation of the exponent is trivial, we describe here the computation of the mantissa. Assume $A > 0$ and $B > 0$. Let the mantissa of A and B be denoted by a and b respectively. We first extract each of a and b in the following bit pattern.

0	0 ↑	46 most significant bits	(P1)

0	0 ↑	46 less significant bits	(P2)

0	0 ↑	40 least significant bits	6 zeros	(P3)

(Here the ↑ signifies the location of the binary point.)

Now a and b have the following representations:

$$a = a_1 + a_2 \cdot 2^{-46} + a_3 \cdot 2^{-92} \tag{1}$$

$$b = b_1 + b_2 \cdot 2^{-46} + b_3 \cdot 2^{-92} \tag{2}$$

Work performed in part under the auspices of the U.S. Atomic Energy Commission.

where in the bit patterns (P1), (P2), (P3) respectively:

$$2^{-1} \leqq \begin{Bmatrix} a_1 \\ b_1 \end{Bmatrix} \leqq 1 - 2^{-46}, \tag{3}$$

$$0 \leqq \begin{Bmatrix} a_2 \\ b_2 \end{Bmatrix} \leqq 1 - 2^{-46}, \tag{4}$$

$$0 \leqq \begin{Bmatrix} a_3 \\ b_3 \end{Bmatrix} \leqq 1 - 2^{-40}. \tag{5}$$

Thus

$$ab = a_1b_1 + (a_1b_2 + a_2b_1)2^{-46} + (a_1b_3 + a_2b_2 + a_3b_1)2^{-92} \tag{6}$$

(approximately), where, using the above inequalities (3), (4) and (5),

$$2^{-2} \leqq a_1b_1 \leqq 1 - 2^{-45} + 2^{-92}, \tag{7}$$

$$0 \leqq a_1b_2 + a_2b_1 \leqq 2 - 2^{-44} + 2^{-91}, \tag{8}$$

$$0 \leqq a_1b_3 + a_2b_2 + a_3b_1 \leqq 3 - 2^{-39} - 2^{-44} + 2^{-85} + 2^{-92}. \tag{9}$$

The relation (6) shows the method for carrying out the computation. In fact, we propose the scheme which immediately follows.

Step 1. Compute $(a_1b_3 + a_2b_2 + a_3b_1) \cdot 2^{-92}$.

Compute first a_1b_3, a_2b_2 and a_3b_1 in double length using the fixed-point operations. Retain only the upper 46 significant bits of each product and arrange them in the pattern:

a_1b_3, a_2b_2, a_3b_1 :	0	0 ↑	46 significant bits	(P4)

Add them bit-by-bit using fixed-point addition. The inequality (9) and the fact that our fixed-point addition is modulo $2^{48} - 1$ shows that overflow never occurs in the last addition. We obtain the sum $a_1b_3 + a_2b_2 + a_3b_1$ in the form

$a_1b_3 + a_2b_2 + a_3b_1$:	c_1	c_2 ↑	46 significant bits

where

c_1	c_2 ↑

are possible carry bits to the left of the binary point. Separate the carry bits from the fraction part of the sum in the following form and store them in the memory.

$C_1 \equiv$ carry :	46 zeros	c_1	c_2 ↑	(P5)

$F_1 \equiv$ fraction:	0	0 ↑	46 significant bits	(P6)

Step 2. Compute $(a_1b_2 + a_2b_1)2^{-46} + (a_1b_3 + a_2b_2 + a_3b_1)2^{-92}$.

The second term (i.e., C_1 and F_1 in (P5) and (P6) above) has been computed and stored in the memory. Compute a_1b_2 and a_2b_1 in double length using fixed-point multiplication and arrange them in the following form.

	upper			*lower*			
a_1b_2, a_2b_1 :	0	0 ↑	46 most significant bits	0	0 ↑	46 least significant bits	(P7)

Add to F_1 the lower halves of a_1b_2 and a_2b_1. Here again we have no overflow for the same reason given in step 1. Just as in step 1, separate the two carry bits from the fraction part of the above sum in the form (P5) and (P6). Add to these carry bits the upper halves of a_1b_2 and a_2b_1, and the carry C_1 from the last step. We have now $(a_1b_2 + a_2b_1)2^{-46} + (a_1b_3 + a_2b_2 + a_3b_1)2^{-92}$ in the form

d_1	d_2	46 most significant bits	0	0	46 least significant bits	(P8)

where

d_1	d_2

are the carry bits. As before the two carry bits are stored separately and we introduce two zeros at the location of the carry bits. Consequently, we store the above quantity in the form which follows in (P9).

46 zeros	d_1	d_2	0	0	46 less significant bits	0	0	46 least significant bits	(P9)
($\equiv C_2$)			($\equiv F_2$)			($\equiv X_3$)			

Step 3. Compute

$$a_1b_1 + (a_1b_2 + a_2b_1)2^{-46} + (a_1b_3 + a_2b_2 + a_3b_1)2^{-92} = ab.$$

The sum of the second and the third terms has been computed and stored in the memory. Repeat step 2 with C_2, F_2 in place of C_1, F_1 and with a_1b_1 in place of a_1b_2 and a_2b_1. Then we shall have the desired product (ab) from step 2 in the form

ab:	e_1	e_2	46 most significant bits	0	0	46 less significant bits	0	0	46 least significant bits
	$\equiv X_1$			$\equiv X_2$			$\equiv X_3$		

(P10)

where the carry bits e_1 and e_2 are actually both 0. (This is known a priori from the expression (6) and from the fact that $0 < a < 1$ and $0 < b < 1$.) The three 46-bit significant parts in the above constitute the unnormalized fraction part of the product AB. If this is less than $\frac{1}{2}$, we need a normalization (left shift) to remove a zero immediately following the binary point. (If this is the case, the corresponding adjustment in the exponent is necessary.) Now we retain only the first 132 significant bits as the true mantissa of the product AB.

3. Division

Let $A = 2^{\alpha}\cdot a$ and $B = 2^{\beta}\cdot b$ be given triple-precision floating-point numbers. It is desired to compute A/B in the same form. We assume $A > 0$ and $B > 0$. Write A and B in the form

$$A = 2^{\alpha+2}\cdot 2^{-2}\cdot a = 2^{\alpha+2}\cdot(a_1 + 2^{-47}\cdot a_2 + 2^{-94}\cdot a_3) \quad (10)$$

$$B = 2^{\beta}\cdot b = 2^{\beta}\cdot(b_1 + 2^{-47}\cdot b_2 + 2^{-94}\cdot b_3) \quad (11)$$

where

$$\begin{aligned} \tfrac{1}{8} &\leqq a_1 \leqq \tfrac{1}{4} - 2^{-47}, & \tfrac{1}{2} &\leqq b_1 \leqq 1 - 2^{-47}, \\ 0 &\leqq a_2 \leqq 1 - 2^{-47}, & 0 &\leqq b_2 \leqq 1 - 2^{-47}, \\ 0 &\leqq a_3 \leqq 1 - 2^{-40}, & 0 &\leqq b_3 \leqq 1 - 2^{-38}, \end{aligned} \quad (12)$$

	a_1					a_2		a_3			
$2^{-2}a$:	0	0	0	1	44 bits	0	47 bits	0	40 bits	7 zeros	(P11)

	b_1			b_2		b_3			
b:	0	1	46 bits	0	47 bits	0	38 bits	9 zeros	(P12)

Using (10) and (11),

$$\frac{A}{B} = 2^{\alpha+2-\beta}\cdot\frac{a_1 + 2^{-47}a_2 + 2^{-94}a_3}{b_1 + 2^{-47}b_2 + 2^{-94}b_3}$$

$$= 2^{\alpha+2-\beta}\cdot\frac{1}{b_1}\left(\frac{a_1 + 2^{-47}a_2 + 2^{-94}a_3}{1 + 2^{-47}\dfrac{b_2}{b_1} + 2^{-94}\dfrac{b_3}{b_1}}\right)$$

$$= 2^{\alpha+2-\beta}\frac{1}{b_1}(a_1 + 2^{-47}a_2 + 2^{-94}a_3)\left[1 - \left(2^{-47}\frac{b_2}{b_1} + 2^{-94}\frac{b_3}{b_1}\right) + \left(2^{-47}\frac{b_2}{b_1} + 2^{-94}\frac{b_3}{b_1}\right)^2 - \cdots\right] \quad (13)$$

$$= 2^{\alpha+2-\beta}\cdot\frac{1}{b_1}\left[a_1 + 2^{-47}a_2 + 2^{-94}\left(\frac{a_1b_2^2}{b_1^2} + a_3\right) - 2^{-47}\cdot\frac{a_1b_2 + 2^{-47}(a_1b_3 + a_2b_2)}{b_1}\right]$$

$$= 2^{\alpha+2-\beta}\cdot\frac{1}{b_1}(U - V)$$

where

$$U = a_1 + 2^{-47}a_2 + 2^{-94}\left(\frac{a_1b_2^2}{b_1^2} + a_3\right), \quad (14)$$

$$V = 2^{-47}\cdot\frac{a_1b_2 + 2^{-47}(a_1b_3 + a_2b_2)}{b_1}. \quad (15)$$

We have chosen $U \geqq 0$ and $V \geqq 0$ to avoid subtractions in their formation. Since the computation of the exponent $\alpha + 2 - \beta$ is trivial, we describe here a method for computing $(1/b_1)(U - V)$. The relations (14) and (15) show that we need to compute U in triple-precision and $2^{47}\cdot V$ in double-precision.

Step 1. Compute U.

First compute $(a_1b_2^2/b_1^2) + a_3$ in single length. The inequalities (12) show that $0 \leqq a_1b_2^2/b_1^2 < 1$ and $0 \leqq a_3 < 1$. Thus, we can proceed as follows.

$$a_1 \to a_1b_2 \to a_1b_2/b_1 \to a_1b_2^2/b_1 \to a_1b_2^2/b_1^2 \to a_1b_2^2/b_1^2 + a_3,$$

where each multiplication or division is done using fixed-point operations. (We assume that fixed-point (fractional) multiplication forms a double-length product from single-length operands and that fixed-point (fractional) division forms a single-length quotient and a remainder from a double-length dividend and a single-length divisor.) The last quantity has the form:

$(a_1b_2^2/b_1^2) + a_3$:	c	47 significant bits	(P13)

where c is the carry bit. Now the quantity U is obtained by straightforward fixed-point additions. (In fact, we need only to add the carry c to the last bit of a_2 and then add the carry yielded in this addition to the last bit of a_1.) Finally we have U in the form

U:	0	0	f	f	....	f	0	f	f	....	f	0	f	f	....	f	(P14)

where f signifies any significant bit (0 or 1) and the leftmost f in the leftmost cell could be 1. Store U in this form.

Step 2. Compute V.

First we need to compute the dividend, $a_1b_2 + 2^{-47}(a_1b_3 + a_2b_2)$, in double length. To do this we secure a_1b_2 in double length and a_1b_3, a_2b_2, both in single length, using fixed-point operations.

Then simple addition yields the above-mentioned dividend in the form

$a_1b_2 + 2^{-47}(a_1b_3 + a_2b_2)$: | 0 | 0 | g | g | .... | g | | 0 | g | g | .... | g | (P15)

where g signifies any significant bit and the leftmost g in the left cell could be 1. Note that the inequalities (12) show

$$a_1b_2 + 2^{-47}(a_1b_3 + a_2b_2) < 2^{-1}.$$

Now we divide this result by b_1. Since we must have a double-length quotient and since each fixed-point division which we are to use now gives a single-length quotient and a single-length remainder, we first divide the double-length dividend by the single-length divisor b_1, and obtain the single-length quotient q_1 and a single-length remainder. We then augment this remainder with a single-length zero (thus obtaining the double-length number) and divide this double-length number by b_1 to obtain the single-length quotient q_2. The q_1 and q_2 constitute the desired double-length quotient $(a_1b_2 + 2^{-47}(a_1b_3 + a_2b_2))/b_1$. Thus we now have V in the form

V: | 0 | 47 zeros | $\overbrace{| 0 | g | g | .... | g |}^{q_1}$ $\overbrace{| 0 | g | g | .... | g |}^{q_2}$ (P16)

Step 3. Compute $U - V$.

To compute $U - V$ by adding the complemented V to U, namely, $U - V = U + (-V)$, we proceed as follows. First, subtract 1 from the rightmost bit of the leftmost word of U (borrow) and add 1 to the rightmost bit of the rightmost word of U (end-around carry). Call the result U'.

Next, complement each bit g of V in (P16). Call the result V'.

V': | 0 | 47 zeros | | 0 | $\bar{g}$ | $\bar{g}$ | .... | $\bar{g}$ | | 0 | $\bar{g}$ | $\bar{g}$ | .... | $\bar{g}$ | (P17)

where $\bar{g}$ means the complemented g. Add U' ((P14)) and V' bit-by-bit using fixed-point operations, where any carry bit that might be produced from the addition of the lower two words must be properly added to the next upper word. We again arrange the result $(=U - V)$ in the form

$U - V$: | 0 | 0 | f | f | .... | f | | 0 | f | f | .... | f | | 0 | f | f | .... | f | (P18)

Step 4. Compute $U - V/b_1 =$ Ans.

We must now divide the triple-length number $U - V$ by the single-length number b_1 to obtain the triple-length quotient. We accomplish this by three successive fixed-point divisions, each time obtaining a single-length quotient and a single-length remainder. Hence it is vital to obtain the correct remainder at least from the first two divisions. To obtain this we regard the dividend $U - V$ and the divisor b_1 as integers and use the integer divide operation. (The fractional divide operation may not retain the last bit of the remainder.) It is then necessary to rearrange $U - V$ of (P18) in the form

$U - V$: | 0 | 0 | 0 | f | f | .... | f | | f | f | .... | f | | f | f | .... | f | 0 | (P19)

an extra zero

(We do not have to rearrange b_1.) Now the proposed division $(U - V)/b_1$ can be easily performed by three successive single-precision fixed-point divisions, thus obtaining the triple-length quotient. The interpretation of the result and the adjustment of the binary point are easy.

4. Remarks

Four triple-precision (132-bit) floating-point arithmetic subroutines (addition, subtraction, multiplication and division subroutines) have actually been coded for the Control Data 1604 computer using the described algorithm and they are now working at the University of Texas Computation Center. The Control Data 3600 computer version of the above subroutines are also working at Argonne National Laboratory.

Acknowledgments. This work was supported in part by the National Science Foundation under Grant GP-217 and by the U.S. Army Research Office (Durham) under Grant DA-ARO(D)-31-124-G388. The technique explained in this note was first developed at the Applied Mathematics Division, Argonne National Laboratory, Argonne, Illinois, during the summer of 1963 while I was working there as a Resident Student Associate. Many thanks are due to Dr. Robert T. Gregory of the University of Texas for reading the rough draft of this note. I am very much indebted to Dr. Richard F. King of Argonne National Laboratory for his many constructive suggestions.

Received September, 1964

REFERENCE

1. Gregory, R. T., and Raney, J. L. Floating-point arithmetic with 84-bit numbers. *Comm. ACM 7*, 1 (Jan. 1964), 10–13.

Method in Randomness

Martin Greenberger
Massachusetts Institute of Technology, Cambridge, Mass.

Certain nonrandom properties of a commonly used random number generator are described and analyzed.

Introduction

Almost all prescribers of random number generators label their concoctions with a virtual skull and crossbones that warns the user against indiscriminate application. The label usually consists of a qualifier for random, like *pseudo* or *quasi*, plus a few general words of caution.

There is more to this than a simple hedging. Most generators in use are of the congruential type, wherein each number of the generated series is calculated solely from its predecessor and some fixed parameters. A typical generator, for example, has the form

$$x_{i+1} \equiv \lambda x_i + c \qquad (\text{mod } 2^P) \quad (1)$$

where x_{i+1} is the number being generated, x_i is its predecessor, λ and c are fixed parameters of the generator, and 2^P is the modulus.

On first glance, it is not clear how a simple rule like (1) can produce numbers that are random even in pretense; that is, pseudo-random. Obviously, it cannot produce true randomness. There is blatant linkage between x_i and x_{i+1}.