Software for Doubled-Precision Floating-Point Computations

SEPPO LINNAINMAA
Lappeenranta University of Technology

A method for exact multiplication of two maximum-precision numbers, so that the result is represented using two such numbers, was proposed by T.J Dekker a decade ago. A modification of Dekker's method is presented and is proved to be valid in most existing arithmetics, while the original method is valid only in a quite restricted class of arithmetics. The new method is computer independent, can be written using a high-level language, and still makes it possible to double quite economically the "maximum available" precision of a computer during critical phases of computations. Corresponding methods for doubling the precision are presented and also analyzed for addition, subtraction, and division

Key Words and Phrases floating-point arithmetic, exact multiplication, rounding errors, software portability
CR Categories 4.22, 5.11, 6.32

1. INTRODUCTION

Artifices to extend the maximum available precision have been widely analyzed. In such methods, if they are computer independent, the fact that two numbers can always be subtracted from each other exactly, with no rounding errors, if they are almost equal, is used. It has been shown that the sum and difference of two numbers of maximum available precision can easily be computed in a very wide variety of arithmetics [2–10]. The result is represented as a double number, that is, two floating-point numbers whose sum is the actual result. In fact, if the limitations of the exponent range are neglected, arbitrary-precision addition and subtraction can be implemented computer independently, using a high-level language [10]. As many floating-point numbers as required are then used to represent each multiple-precision number.

The only well-known computer-independent methods to double the available precision in multiplication and division as well have been presented by T.J. Dekker [2]. Obviously doubling the precision of multiplication means that the result is always exact. Dekker's multiplication method is restricted to arithmetics

This research was supported by a Herman Rosenberg grant from the University of Helsinki, Finland, and was done while the author was a visiting scholar at the University of California, Berkeley.
Author's address: Lappeenranta University of Technology, Box 20, SF-53851 Lappeenranta 85, Finland

where addition and subtraction are correctly rounding, that is, where the roundoff error is never more than half unit in the last remaining position. Another restriction is that the precision of the arithmetic must be even in other than binary arithmetics. In the present paper a modification of Dekker's method is proposed whereby both restrictions have been removed.

Dekker also presented algorithms for operations between two double numbers, when the result is also wanted as a double number. These algorithms utilize the exact multiplication method. This paper presents modifications of these algorithms which differ from those described by Dekker, if one or more extra digits beyond the normal maximum precision are available in restricted use, as proposed in [1].

2 DEFINITIONS AND NOTATIONS

Below, the base of the arithmetic is denoted by β and the normal maximum precision by p. A number is said to be an *i-digit number*, if it can be represented in the form $f \times \beta^k$, where f and k are integers and $|f| < \beta^i$. If x is an expression, then $\mathrm{fl}_i(x)$ means that the result of each operation included in x is rounded to an i-digit number using the current rounding rule. Especially if $i = p$, shorter notation $\mathrm{fl}(x)$ will be used instead of $\mathrm{fl}_p(x)$.

Let a be an arbitrary nonzero real number and x and y be two consecutive i-digit numbers with the same sign as a, such that $|x| \leq |a| < |y|$. The arithmetic having precision i is called *correctly rounding* if $\mathrm{fl}_i(a) = x$ when $|x - a| < |y - a|$, $\mathrm{fl}_i(a) = y$ when $|x - a| > |y - a|$, and $\mathrm{fl}_i(a)$ is equal to either x or y when $|x - a| = |y - a|$. The arithmetic is called, respectively, *correctly chopping* if $\mathrm{fl}_i(a) = x$, and *faithful* if $\mathrm{fl}_i(a)$ is equal to x when $a = x$ and equal to either x or y when $a \neq x$. The exponent range is assumed to be unlimited, that is, no underflows or overflows are considered. However, some discussion of their effects is given in Section 6.

For an arbitrary nonzero real number a, notations $F(a)$, $L(a)$, and $W(a)$ are used for the unit of leading (that is, first nonzero) digit of a, for the unit of the last nonzero digit of a and for the width of a, respectively. Then $\beta^{F(a)} \leq |a| < \beta^{F(a)+1}$, $a \bmod \beta^{L(a)+1} \neq 0$, but $a \bmod \beta^{L(a)} = 0$ and $W(a) = F(a) - L(a) + 1$. For completeness, define $F(0) = -\infty$, $L(0) = \infty$, and $W(0) = 0$. Obviously a is an i-digit number if and only if $W(a) \leq i$.

Two numbers a and b are called nonoverlapping if either $L(a) > F(b)$ or $F(a) < L(b)$. A *double number* $a + b$ is a combination of two nonoverlapping p-digit numbers a and b, for which $a = \mathrm{fl}(a + b)$.

3. SPLITTING OF FLOATING-POINT NUMBERS

The exact multiplication methods are based on splitting p-digit operands into smaller units. This splitting is done by adding a properly chosen mask to the number to be split, causing the trailing part of the number to be rounded off. Then the mask is subtracted in hope that the remaining number is the leading part of the original number. Theorem 1 below shows that a proper mask can easily be found for any floating-point number in a fairly general class of arithmetics, using a computer-independent high-level language. The mask used in

Theorem 1 is equal to the mask used by Dekker [2], but the splitting is shown to be successful essentially more generally than he stated.

THEOREM 1 (DEKKER'S SPLITTING METHOD). *If A is a p-digit number and k is such integer that $1 \le k < p$, then the numbers a and α produced by operations*

$$t \leftarrow fl((\beta^{p-k} + 1) \times A),$$

$$a \leftarrow fl(t - (t - A)),$$

$$\alpha \leftarrow fl(A - a)$$

are such that $a + \alpha = A$, a is a k-digit number, α is a $(p - k)$-digit number, and a and α are nonoverlapping, provided that the arithmetic is faithful and that for any integers ι and d, $0 < d < \beta$, either of the following is true.

(i) the product of the numbers $(\beta^{p-k} + 1)$ and $(\beta^{\iota} - d\beta^{\iota-p})$ is chopped, or

(ii) the difference of the numbers $(\beta^{\iota+p-k} + \beta^{\iota})$ and $(\beta^{\iota} - d\beta^{\iota-p})$ is correctly rounded (i.e., chopped) to $\beta^{\iota+p-k}$

PROOF. The assertion is trivial for $A = 0$. To prove other cases, for simplicity assume that $\beta^{k-1} \le A < \beta^k$, that is, that A is a positive number whose integer part is k digits wide. Obviously this is no actual restriction. It is easy to see that if $\beta^{p-1} \le t - A < \beta^p$, then a is the absolute value of A rounded to k digits faithfully, the magnitude of α being equal to the magnitude of the roundoff error of this rounding. So the validity of the theorem is obvious in this case. Other possible cases are considered below where $(\beta^{p-k} + 1) \times A$ is denoted by T, so that t is the value of T rounded to p digits.

If $t - A < \beta^{p-1}$, certainly $T < \beta^p$, and thus the absolute value of the roundoff error $t - T$ is less than 1. So $\beta^{p-1} > t - A \ge (T - 1) - A = \beta^{p-k}A - 1$, and hence $A < \beta^{k-1} + \beta^{k-p}$. Then, because A is a floating-point number, it follows that $A = \beta^{k-1}$. But $t - A = \beta^{p-1}$; so $t - A < \beta^{p-1}$ is impossible.

If $t - A \ge \beta^p$, the roundoff error of the multiplication is less than β, since obviously T is never $\ge \beta^{p+1}$. So $\beta^p \le t - A < T + \beta - A = \beta^{p-k}A + \beta$, and thus $A > \beta^k - \beta^{k-p+1}$. Because A is a floating-point number, $A = \beta^k - d\beta^{k-p}$, where d is an integer, $0 < d < \beta$. Then $T = \beta^p(1 + \beta^{k-p} - d\beta^{-p} - d\beta^{k-2p})$. If $t \le T$, then $t - A \le T - A = \beta^{p-k}A < \beta^p$. But we assumed $t - A \ge \beta^p$, so necessarily $t > T$ in the present case. It follows that if assumption (i) is true, then $t - A \ge \beta^p$ is impossible and the proof is completed. Otherwise assumption (ii) must be true. Then, if $t > T$, it follows that $t = \beta^p(1 + \beta^{k-p}) = \beta^p + \beta^k$, and hence $t - A$ is, according to assumption (ii), rounded to β^p. Therefore $a = t - \beta^p = \beta^k$ and $\alpha = A - \beta^k = -d\beta^{k-p}$, both being one-digit numbers. □

Obviously condition (i) or (ii) is valid when correct rounding or correct chopping arithmetic is used. To check the validity in other faithful arithmetics, note that the exact value of the product mentioned in assumption (i) is $100 \cdots 0mm \cdots md''mm \cdots md' \times \beta^{\iota-p}$, where $m = \beta - 1$, $d' = \beta - d$, $d'' = d' - 1$, and the lengths of the 0, m, and m patterns are $p - k$, $k - 1$, and $p - k - 1$, respectively. So the first digit to be rounded off is d'', preceded by 0 when $k = 1$, and by m otherwise. Correspondingly, the exact value of the difference mentioned in assumption (ii) is $100 \cdots 0d \times \beta^{\iota-p}$, with $2p - k - 1$ zeros.

If arithmetic is faithful but both assumptions fail, splitting fails for numbers with fraction parts $mm \cdots md$ (and only for them). As an example, let $\beta = 10$, $p = 6$, $k = 3$, and $A = 999993$. Then $t = \mathrm{fl}(1000992993) = 1001000000$ (because (i) fails), so $\mathrm{fl}(t - A) = \mathrm{fl}(1000000007) = 1000010000$ (because (ii) fails), $\alpha = 990000$, and $\alpha = 9993$. Thus α has $4 > 3 = p - k$ digits.

The value of the base number β is assumed to be known in the method considered in Theorem 1. Well-known algorithms that determine the base number of the computer exist and can be written computer independently, so this assumption does not destroy the computer independency of the splitting method. One very effective such algorithm, which requires no loops and is due to W. Kahan [5], is stated in Theorem 2 below.

THEOREM 2 (BASE AND PRECISION DETERMINATION). *If the base number is even and not divisible by* 3, *and the arithmetic is faithful, then the following operations produce the values of base number* β *and precision* p:

$$
\begin{aligned}
&U \leftarrow \left| fl\left(3 \times \left(\frac{4}{3} - 1\right) - 1\right) \right|,\\
&R \leftarrow fl\left(\left(\frac{U}{2} + 1\right) - 1\right),\\
&\textit{if} \quad R \neq 0 \quad \textit{then} \quad U \leftarrow R,\\
&u \leftarrow \left| fl\left(3 \times \left(\frac{2}{3} - 0.5\right) - 0.5\right) \right|,\\
&r \leftarrow fl\left(\left(\frac{u}{2} + 0.5\right) - 0.5\right),\\
&\textit{if} \quad r \neq 0 \quad \textit{then} \quad u \leftarrow r,\\
&\beta \leftarrow fl\left(\frac{U}{u}\right),\\
&p \leftarrow \textit{entier}\left(fl\left(\frac{-\log(u)}{\log(\beta)} + 0.5\right)\right).
\end{aligned}
$$

PROOF. Let $t_1 = \mathrm{fl}(4/3)$. Apparently then $t_1 = 4/3 + e_1\beta^{1-p}/3$, where e_1 is either -2, -1, $+1$, or $+2$. After that, while computing the value of U, each intermediate result x is such that $F(x) \leq 0$ and $L(x) \geq 1 - p$, so $W(x) \leq p$. Thus no more rounding errors occur, and so $U = |3 \times (t_1 - 1) - 1| = |3t_1 - 4| = |e_1|\beta^{1-p}$.

If $|e_1| = 2$, then R is necessarily computed exactly, and thus $R = U/2 = \beta^{1-p}$. If $|e_1| = 1$, then $U/2 + 1 = 1 + \frac{1}{2}\beta^{1-p}$ is rounded to either 1 or $1 + \beta^{1-p}$. In all these cases, the final value of U is obviously β^{1-p}.

Let $t_2 = \mathrm{fl}(2/3)$. Apparently then $t_2 = 2/3 + e_2\beta^{-p}/3$, where again $|e_2|$ is either 1 or 2. After that, while computing u, each intermediate result x is such that $F(x) \leq -1$ and $L(x) \geq -p$; therefore $W(x) \leq p$. Thus no more rounding errors occur, and so $u = |3 \times (t_2 - 0.5) - 0.5| = |3t_2 - 2| = |e_2|\beta^{-p}$. When the value of τ is computed and tested, the final value of u is necessarily β^{-p}, which can be seen using similar reasoning as above for U.

Now $U/u = \beta^{1-p}/\beta^{-p} = \beta$ and $\log(u)/\log(\beta) = \log_\beta(u) = -p$; therefore, the validity of the theorem is obvious with any reasonable implementation of the log function. □

4. EXACT MULTIPLICATION

After successful splitting, the exact multiplication, producing a double number, is rather straightforward. Dekker [2] described two quite similar methods for this purpose, one being his own and the other due to G. W. Veltkamp. Dekker stated that both work correctly provided that addition and subtraction are correctly rounding and multiplication is faithful. Dekker proved that his own version works in binary arithmetic, and stated that the results can be generalized to other bases. This is true with the restriction that precision p must be an even number in the methods that he describes, if base $\beta > 2$. In Theorem 3 below a proof is given that a modification of the version due to Veltkamp requires only faithfulness of the arithmetic, provided that the splitting can be made successfully. No practical limitations to the base number or precision are needed.

Both base β and precision p are implicitly assumed to be known before the method described in Theorem 3 can be implemented. Both of them can be computed computer independently, as shown in Theorem 2; so the method as a whole can be treated as computer independent.

THEOREM 3 (EXACT MULTIPLICATION). *Let A and C be two p-digit numbers and let k be such integer that $k \geq 2$ and $p/3 \leq k \leq p/2$. Further let*

$$A = a + \alpha, \qquad C = c + \gamma, \qquad \text{and} \qquad \gamma = \kappa + \epsilon$$

where a, c, and κ are k-digit approximations of A, C, and γ, respectively, each being nonoverlapping with the corresponding $(p - k)$-digit remaining part. Then the double number $X + x$ produced by the operations

$$X \leftarrow fl(AC),$$

$$x \leftarrow fl(((((ac - X) + a\gamma) + c\alpha) + \kappa\alpha) + \epsilon\alpha)$$

is such that $X + x = AC$, provided that the arithmetic is faithful.

PROOF. If A or C or both are equal to zero, the assertion is trivial. To prove the assertion in other cases, assume for simplicity that $\beta^{p-1} \leq A < \beta^p$ and $\beta^{p-1} \leq C < \beta^p$, that is, that they both are positive p-digit integers, which is no actual restriction. Obviously $W(ac) \leq k + k \leq p$, $W(a\gamma) \leq k + (p - k) = p$, $W(c\alpha) < k + (p - k) = p$, $W(\kappa\alpha) \leq k + (p - k) = p$, and $W(\epsilon\alpha) \leq (p - 2k) + (p - k) \leq p$; so all these multiplications can be done without rounding errors. If $AC - X$ is denoted by E, then obviously $|E| < \beta^p$, $W(E) \leq p$, and X and E are nonoverlapping numbers.

Let $t_1 = ac - X$. Then $|t_1| = |(AC - \alpha C - a\gamma) - (AC - E)| = |\alpha C + a\gamma - E|$. Suppose first that $AC \geq \beta^{2p-1}$. Then $|t_1| < \beta^{2p-k} + \beta^{2p-k} + \beta^p < \beta^{2p-k+2}$, and thus $F(t_1) \leq 2p - k + 1$. On the other hand, $L(ac) \geq p$ and $L(X) \geq p$ in this case; so $L(t_1) \geq p$ and thus $W(t_1) \leq (2p - k + 1) - p + 1 = p - k + 2 \leq p$. If $AC < \beta^{2p-1}$, then, due to the assumed limitations for A and C, necessarily $A + C < \beta^p + \beta^{p-1}$. Thus $|t_1| = |ac - X| \leq |(A + \beta^{p-k})(C + \beta^{p-k}) - AC| + |E| < (A + C)\,\beta^{p-k} +$

$\beta^{2p-2k} + \beta^{p-1} < \beta^{2p-k} + \beta^{2p-k-1} + \beta^{2p-2k} + \beta^{p-1} < \beta^{2p-k+1}$; so $F(t_1) \leq 2p - k$. Since in this case $L(X) \geq p - 1$, $W(t_1) \leq (2p - k) - (p - 1) + 1 = p - k + 2 \leq p$. Hence, in any case, t_1 can be computed exactly, with no roundoff errors.

Let $t_2 = t_1 + a\gamma$. Then $|t_2| = |\alpha C - E| \leq (\beta^{p-k} - 1)(\beta^p - 1) + \beta^p = \beta^{2p-k} - \beta^{p-k} + 1 < \beta^{2p-k}$; so $F(t_2) \leq 2p - k - 1$. On the other hand, $L(t_1) \geq p$ and $L(a\gamma) \geq p - k$; so $L(t_2) \geq p - k$. But then $W(t_2) \leq (2p - k - 1) - (p - k) + 1 = p$; so t_2 can be computed exactly, too.

Let $t_3 = t_2 + c\alpha$. Then $|t_3| = |\alpha\gamma - E| < \beta^{2p-2k} + \beta^p \leq 2\beta^{2p-2k}$; so $F(t_3) \leq 2p - 2k$. $L(t_3) \geq p - k$; so $W(t_3) \leq (2p - 2k) - (p - k) + 1 = p - k + 1 < p$, and thus t_3 can also be computed exactly.

Let $t_4 = t_3 + \kappa\alpha$. Then $|t_4| = |\epsilon\alpha - E| < \beta^{p-k}\beta^{p-2k} + \beta^p \leq 2\beta^p$, and hence $F(t_4) \leq p$. Since $L(t_4) \geq \min(L(t_3), L(\kappa) + L(\alpha)) \geq p - 2k$, $W(t_4) \leq p - (p - 2k) + 1 = 2k + 1$. If p is odd, $k \leq p/2$ implies $k \leq (p - 1)/2$ and thus $W(t_4) \leq p$. If p is even and $k < p/2$, $W(t_4) \leq p - 1$. If p is even and $k = p/2$, then $\epsilon = 0$ and thus $|t_4| = |E|$. So in all cases, t_4 can be computed exactly.

Now $x = t_4 + \epsilon\alpha = E$, which completes the proof. □

If $k = p/2$, then $\epsilon = 0$; so there is no need to split γ, and the addition of the term $\epsilon\alpha$ can be omitted. κ is obviously replaced by γ in the remaining formula for x. Then the original method due to Veltkamp [2] results.

It can also be easily found that $\epsilon = 0$ when the arithmetic is binary and correctly rounding, p is odd, $k = (p - 1)/2$, and the splitting is done using Dekker's splitting method. So the simplified formula can be used in this situation, too.

The simplified formula is obviously also applicable for any p, if $k = entier(p/2)$ and the product $\gamma\alpha$ is not rounded to a p-digit number, but rather to a $(p + h)$-digit number for some integer $h > 0$. The availability of such arithmetic is proposed, for example, in [1].

The corollary below summarizes the possibilities to use the simplified formula.

COROLLARY (EXACT MULTIPLICATION). *Let A and C be two p-digit numbers, $p \geq 4$, $k = entier(p/2)$ and $h \geq 0$. Further let*

$$A = a + \alpha \qquad \text{and} \qquad C = c + \gamma$$

where a and c are k-digit approximations of A and C, respectively, both being nonoverlapping with the corresponding $(p - k)$-digit remaining part. Then the double number $X + x$ produced by the operations

$$X \leftarrow fl(AC);$$

$$x \leftarrow fl(((((ac - X) + a\gamma) + c\alpha) + fl_{p+h}(\gamma\alpha))$$

is such that $X + x = AC$, provided that the arithmetic is faithful and either

(i) p is an even integer,
(ii) binary correctly rounding arithmetic is used, or
(iii) $h > 0$. □

When the above corollary is applied in the case in which $h > 0$, obviously, if preferred, other intermediate results may also be rounded to $p + h$ instead of p digits, and not only $\gamma\alpha$, when the value of x is computed.

5. ALGORITHMS FOR DOUBLED-PRECISION OPERATIONS

Effective algorithms for doubled-precision operations that utilize the exact multiplication method can be constructed. Such algorithms for multiplication, division, and addition are given below as portable PASCAL procedures. The first procedure implements the exact multiplication method of Theorem 3. It is valid when either of the conditions of Theorem 1 is valid. It is especially valid in correctly rounding and correctly chopping arithmetics with no further restrictions.

```
procedure exactmul (a, c:real:var x, xx:real);
  {this procedure implements x + xx ← a*c.
  constant is assumed to be a global variable whose
  value is β^(p−entier(p/2)) + 1 where β is the base and
  p is the precision of real numbers}
  var a1, a2, c1, c2, c21, c22, t:real;
begin
  t := a*constant; a1 := (a − t) + t; a2 := a − a1;
  t := c*constant; c1 := (c − t) + t; c2 := c − c1;
  t := c2*constant; c21 := (c2 − t) + t; c22 := c2 − c21;
  x := a*c;
  xx := ((((a1*c1 − x) + a1*c2) + c1*a2) + c21*a2) + c22*a2
end;
```

As stated in the Corollary of Theorem 3, procedure *exactmul* can be replaced by the following simpler procedure *exactmul* 2 when, in addition to the conditions for the validity of *exactmul*, one of the following conditions is fulfilled:

(i) the number of digits in each *real*-type floating-point fraction is even,
(ii) binary correctly rounded arithmetic is used, or
(iii) at least the product $c2*a2$ is rounded to an extended number (having at least one more digit than real numbers) instead of a real number when computing xx.

```
procedure exactmul2 (a, c:real; var x, xx:real);
  var a1, a2, c1, c2, t:real,
begin
  t := a*constant; a1 := (a − t) + t; a2 := a − a1;
  t := c*constant; c1 := (c − t) + t; c2 := c − c1,
  x := a*c;
  xx := (((a1*c1 − x) + a1*c2) + c1*a2) + c2*a2
end;
```

The alogrithms for doubled-precision operations given below are modifications of the algorithms presented by Dekker [2]. They serve an obvious improvement in accuracy, if such extended precision is available as suggested in [1]. A few extra bits guarantee full precision for doubled-precision operations, as can be concluded from the results presented below. This fact is a strong support for constructing hardware containing such extended precision. The procedures will reduce to the original procedures of Dekker if no such extended precision is available.

In the procedures below variable zz is declared as *extended*, meaning that it has $h \geq 0$ extra digits in its fraction part, compared with *real* numbers. It is assumed that all the operations in the expression whose value will be assigned to zz are computed using this kind of extended arithmetic. It shall be noted that in

the theoretical situation $h = \infty$ all the algorithms for doubled-precision operations below produce numbers z and zz such that $z + zz$ is the exact result of the corresponding operation. The original algorithms of Dekker do not have this property. Finally, $z + zz$ is rounded to double number $x + xx$. In computing xx, the difference $z - x$ can be rounded to extended, as well as to usual real, precision.

Note that extended arithmetic must not be used when computing $a1$ or $c1$ in the above procedures. If one wants to use extended arithmetic there also, that is, in each arithmetic operation excluding the ones whose result will be assigned to a *real* variable immediately, then the value of *constant* must be changed to $\beta^{p+h-entier(p/2)} + 1$.

If especially $h = 0$, that is, no extended precision is available, then in procedure *longmul* below $\mathrm{fl}(a + aa) = a$ and thus obviously $(a + aa)$ can be replaced by simple a. Correspondingly $(c + cc)$ can be replaced by c in procedure *longdiv* when $h = 0$. These reduced forms of the procedures are identical to the original procedures of Dekker.

```
procedure longmul(a, aa, c, cc:real; var x, xx:real);
  {this procedure implements x + xx ← (a + aa)*(c + cc)}
  var z, qq:real; zz:extended;
begin
  exactmul(a, c, z, qq);
  zz:=((a + aa)*cc + aa*c) + qq;
  x:= z + zz;
  xx:=(z - x) + zz
end;
procedure longdiv(a, aa, c, cc:real; var x, xx:real);
  {this procedure implements x + xx ← (a + aa)/(c + cc)}
  var z, q, qq:real, zz:extended;
begin
  z:=a/c;
  exactmul(c, z, q, qq);
  zz:=((((a - q) - qq) + aa) - z*cc)/(c + cc);
  x:=z + zz;
  xx:=(z - x) + zz
end;
procedure longadd(a, aa, c, cc:real; var x, xx:real);
  {this procedure implements x + xx ← (a + aa) + (c + cc)}
  var z, q:real; zz:extended;
begin
  z:=a + c;
  q:=a - z;
  zz:=(((q + c) + (a - (q + z))) + aa) + cc;
  x:=z + zz;
  xx:=(z - x) + zz
end;
```

Obviously procedure *longadd* can be used as well for subtraction by changing the signs of c and cc. If preferred, procedure *longadd2* below can be used instead of procedure *longadd*, if at least one of the following conditions is fulfilled [7]:

(i) binary correctly rounding arithmetic is used,
(ii) correctly chopping arithmetic is used, or
(iii) a correctly rounding arithmetic with $h > 0$ is used.

Table I Theoretical Upper Bounds of the Relative Errors of Doubled-Precision Operations[a]

Procedure	Correctly rounding[b]	Any faithful
longmul	$\frac{1}{2}\beta^{-b} + 2\beta^{1-h}$	$1 + 8\beta^{1-h}$
longdiv	$\frac{1}{2}\beta^{-b} + 3\beta^{1-h}$	$1 + 12\beta^{1-h}$
longadd[c]	$\frac{1}{2}\beta^{-b} + \beta^{1-h}$	$1 + 5\beta^{1-h}$

[a] All entries are multiplied by β^{2p-1}.
[b] The value of b is 1 for binary and 0 for other arithmetics Especially if $h = 0$, then the term $\frac{1}{2}\beta^{-b}$ can be omitted.
[c] The summands are assumed to have equal signs.

The accuracy of the result can be expected to remain similar, when *longadd* is replaced by *longadd*2.

```
procedure longadd2(a, aa, c, cc:real; var x, xx:real);
var z, q:real; zz:extended;
begin
  z:=a + c;
  zz:=if abs(a) ≥ abs(c) then(((a − z) + c) + aa) + cc
    else (((c − z) + a) + cc) + aa;
  x:=z + zz;
  xx:=(z − x) + zz
end;
```

In Table I the approximate upper bounds for the rounding errors are given for the results of the above procedures. They are calculated using the first-order approximation of the accumulated error in terms of relative local errors. It is noticed that $X + x$ is a rounded $(2p + b)$-digit approximation of $Z + z$, where $b = 1$ for binary correctly rounding arithmetic and 0 for other arithmetics. Especially if $h = 0$, then $X + x = Z + z$ in correctly rounding arithmetic [7]. It is noted also that the expression $(a - q) - qq$ can always be calculated exactly in procedure *longdiv* provided that the arithmetic is faithful. Recall for comparison that in "conventional" double precision all the table entries would be $\frac{1}{2}$ for correctly rounding and 1 for any faithful arithmetic.

Also Dekker gave bounds for correctly rounding binary arithmetic [2]. His bounds agree with the corresponding bounds above. Only the bound for a product is slightly better above because Dekker was not able to assume $z = \mathrm{fl}(z + qq)$.

If the summands have different signs, then cancellation may occur, and this yields worse bounds for relative errors than were found in connection with multiplication and division. But in these cases, efforts for obtaining as good bounds would be of no use, since the summands cannot be assumed to have more than about $2p$ correct digits. One way to compare addition with other operations is to divide the absolute error by the larger summand instead of the result, if the two numbers to be added have different signs. Using this approach the entries for *longadd* in Table I would be $\frac{1}{2}\beta^{-b} + 5\beta^{1-h}/4$ for correctly rounding and $1 + 6\beta^{1-h}$ for any faithful arithmetic.

To test the accuracy of procedures *longmul* and *longdiv*, an experiment was run on a VAX 11/780 computer where 1000 pairs of random numbers were multiplied and divided using these procedures. Binary and octal correctly round-

Table II. Maximum Observed and Theoretical Relative Errors of Product and Quotient of Two Numbers, Produced, Respectively, By *longmul* and *longdiv*[a]

		Rounding				Chopping			
		Binary		Octal		Binary		Octal	
Procedure	h	Observed	Theo-retical	Observed	Theo-retical	Observed	Theo-retical	Observed	Theo-retical
longmul	0	1 11	4.00	2.14	16.00	5.32	17 00	10.80	65 00
	1	0.49	2 25	0.49	2.50	2 36	9 00	1.68	9.00
	2	0.38	1.25	0 46	0 75	1.46	5.00	0.97	2.00
	3	0 32	0.75	0.47	0.53	1.07	3.00	0 93	1.13
	4	0 27	0.50	0.46	0 50	0 97	2.00	0.90	1.02
	5	0 25	0.38	0.47	0.50	0.93	1 50	0.93	1.00
	6	0 24	0 31	0.46	0.50	0 92	1.25	0.90	1.00
longdiv	0	1 53	6.00	4.61	24.00	3.31	25 00	13.65	97.00
	1	0.67	3.25	0.66	3 50	1.86	13.00	1.02	13.00
	2	0 46	1 75	0 44	0.88	1 02	7.00	0.90	2.50
	3	0 38	1.00	0.46	0.55	0.95	4.00	0 86	1 19
	4	0 28	0 63	0 44	0 51	0.97	2.50	0.98	1.02
	5	0 24	0 44	0.45	0.50	0 91	1.75	0 87	1 00
	6	0 26	0 34	0 46	0 50	0.95	1.38	0.86	1.00

[a] All values are multiplied by β^{2p-1}

ing and correctly chopping arithmetics were simulated. The maximum errors that were observed, scaled by β^{2p-1}, are given in Table II, confirming the theoretical results.

In fact, with an extension of at least five digits, binary correctly rounding arithmetic produced in both operations an even better approximation for the true result than the corresponding double-precision approximation in more than 60 percent of the sample cases. These results reflect situations where x alone differs by less than $1/\beta$ units in the last place from the true result, which allows xx to catch one extra digit compared with conventional double precision. A worse approximation was never produced by *longmul*, and only in 2 percent of the sample quotients by *longdiv*. In binary chopping and both octal arithmetics, an improved result was obtained in about 20 percent of the sample cases, while worse approximations were never produced by *longmul*, and in only a few percent of the sample cases by *longdiv*.

6. DISCUSSION

Above, rather simple methods were presented that allow doubling the longest available precision of any computer with a reasonable arithmetic. So a quadruple precision is available using this software, if double precision or extended double precision, as suggested in [1], is implemented in the hardware and fulfills the conditions for faithfulness. In correctly rounding arithmetic an extension of at least three digits guarantees that the upper bound of the rounding error is never worse than the corresponding double- (or quadruple, if the basic arithmetic is double) precision upper bound for a faithful arithmetic in general. A very

interesting fact is that in binary rounding arithmetic with an extension of at least five bits the theoretical upper bound of the result is less than the corresponding double-precision upper bound.

The execution time of each operation will increase one order of magnitude. So the proposed methods are considerably slower than the use of a corresponding precision built into hardware, but the need for such extreme precision is so rare that the existence of software like this probably is a sufficient reason to avoid ever building such hardware. The full portability of this software allows its easy use in any connection.

The possibility of underflows and overflows was neglected above, but is now discussed briefly to further establish the usability of the methods. It is easy to see that if procedure *exactmul* is able to produce the exact product of its input parameters, no further problems arise, provided the corresponding single-precision operation neither overflows nor produces a result adjacent to the overflow threshold. But the splitting done by *exactmul* uses numbers requiring exponents that are about $p/2$ units larger than the exponents of the numbers to be split, where p is the number of digits in the fraction part of numbers. Thus the available exponent range is slightly decreased, unless extended numbers with extended exponent fields are used in splitting.

If the conventional single-precision result underflows, essentially the same result is obtained using the above procedures. If the trailing part of the resulting double number does not underflow, the procedures work as well as usual. Some critical situations occur if the leading part does not underflow but the trailing part does. If underflowed numbers are replaced by zero, then the result produced by *longmul* is at worst the same as the product produced by multiplying the leading parts of the operands only, that is, "half precision". A similar effect is possible in division, if the dividend is very near underflow threshold. These inconvenient effects are avoided if *gradual underflow* (see, for example, [1]) is used instead of flushing to zero. When gradual underflow is used, the result can never be more inaccurate than the corresponding conventional single-precision result.

ACKNOWLEDGMENTS

I am grateful to Professor William Kahan for suggesting that I seek generalizations of Dekker's results.

REFERENCES

1 COONEN, J T An implementation guide to a proposed standard for floating point arithmetic *Computer 13* (1980), 68–79
2. DEKKER, T J. A floating-point technique for extending the available precision. *Numer. Math 18* (1971), 224–242.
3. KAHAN, W. Further remarks on reducing truncation errors. *Commun. ACM 8*, (1965), 40.
4 KAHAN, W A survey of error analysis. In *Proc IFIP Cong 1971*, vol 2, C V Freeman (Ed), North-Holland, Amsterdam, 1972, pp. 1214–1239.
5. KAHAN, W Personal communication, 1980
6. KNUTH, D.E. *The Art of Computer Programming: Seminumerical Algorithms*, vol. 2 Addison-Wesley, Reading, Mass., 1969.

7 LINNAINMAA, S. Analysis of some known methods of improving the accuracy of floating-point sums. *BIT 14* (1974), 167–202.
8. MØLLER, O. Quasi double precision in floating-point addition. *BIT 5* (1965), 37–50.
9. MØLLER, O Note on quasi double precision *BIT 5* (1965), 251–255.
10 VIRKKUNEN, J A unified approach to floating-point rounding with applications to multiple-precision summation. Rep A-1980-1. Dep of Comput Sci, Univ of Helsinki, Helsinki, Finland, 1980.

Received July 1980; revised January 1981, accepted February 1981