22

INTERVAL ARITHMETIC FOR THE BURROUGHS B5500: FOUR ALGOL PROCEDURES AND PROOFS OF THEIR CORRECTNESS

by

Donald I. Good*
&
Ralph L. London*

Computer Sciences Technical Report # 26
June 1968

* Computer Sciences Department and Computing Center.

Reprinted December, 1968.

ACKNOWLEDGMENTS

We gratefully acknowledge the support of this work as follows:

Good:	UW Computing Center and Gas Supply Research Department, Northern Natural Gas Company, Omaha, Nebraska.
London:	UW Computing Center and NSF Grant GP-7069.
Computer Time:	University Research Committee.

We are both indebted to Ramon E. Moore for his encouragement and advice.

TABLE OF CONTENTS

ABSTRACT

Four Algol procedures for interval add, subtract, multiply and divide are presented. Also included are proofs that each is correct according to theory developed in the first part of the paper.

INTRODUCTION

This report presents a set of four Algol procedures for performing interval addition, subtraction, multiplication, and division on the Burroughs B5500. The algorithms are presented in the context of Algol on the B5500. In the conclusion we indicate the wider applicability of the algorithms to other computers.

Also included is a set of proofs that show that these procedures are correct. These proofs are given in order to show, first, that the property of interval arithmetic of computing guaranteed bounds is preserved by these procedures, and, second, that these procedures produce the best possible bounds that can be represented on the B5500.

Only by proving the code have we uncovered (and corrected) several errors in an earlier version which had escaped detection by the usual debugging methods. Thus the code clearly has benefited by proof.

But let there be no misunderstanding about our claims that we have proved the correctness of the procedures. In this paper we define what we demand of an implementation of interval arithmetic, and then prove in a mathematical sense that the actual Algol code accomplishes our demands. In other words, we give a set of mathematical-type arguments that the code does what we require. Hopefully the arguments are sufficiently convincing and rigorous.

Yet we should add a note of caution. Certainly we do this not to cast doubts upon our work. Rather, it is included simply because we are not warranting the code in any legal sense. Nor could we. Even with our proofs, we cannot guarantee that every user of the code will always obtain "correct" answers. This is so for two reasons.

First, our proofs are at the level of the Algol code. The proofs say nothing about the accuracy and reliability of either the software or hardware of the B5500. Second, even though we have applied the rules of inference carefully, and we think accurately, there is still no guarantee in an absolute sense that our proofs are error-free. This same hazard is, of course, common to nearly all mathematical proofs. Naturally, we confidently doubt the existence of such an error.

Nevertheless, our proofs exceed the usual attempts to assert the correctness of code. In summary then, the reader should understand both our claims and our cautions.

It should be noted that the present paper is <u>not</u> intended as a user's manual. Instead, it is intended to give a complete development of an interval arithmetic suitable for implementation on a computer. Proofs are given of the properties of this interval arithmetic as well as proofs of the correctness of the actual implementation. The attempt to be as complete as possible and to record all the details in one place accounts for the length of this technical report. A condensed version is in preparation.

Although we both take responsibility for the entire paper, the division of work is basically as follows:

Good: The mathematical presentation of interval arithmetic, statements of machine environment, most of the statements and proofs regarding machine operands; and the actual coding and program organization.

London: Nearly all of the proofs concerned with Good's code.

This paper is directed to two classes of readers, and this situation is a reflection of the interests of its two authors. One class of readers will mainly be interested in the fact that there now exists an implementation of interval arithmetic on the Burroughs B5500. These readers certainly would have used the procedures with or without the attached proofs. For them the proofs are much like the just-discovered proof of convergence of an algorithm that they had been using "successfully" for some time--confirmation of what was "known" all along. Nevertheless, the existence of the proofs is reassuring. This class of readers will probably want to concentrate on the sections dealing with the theory of interval arithmetic and its implementation on the B5500 (Sections I, II and III).

On the other hand, a second class of readers will be mainly interested in our answer to the question, "How would one prove that the present implementation is correct?" These readers should view interval arithmetic (defined in Section I) as an interesting domain in

which to prove the correctness of computer programs. These readers will probably want to concentrate on the proofs of the code (Section IV) and, to a lesser extent, on the environment sections (Sections II and III).

I. INTERVAL ARITHMETIC

For sake of completeness, the relevant properties of interval arithmetic are presented here. For a complete discussion of interval arithmetic and interval analysis see (3).

Real Interval Arithmetic.

Let R denote the set of all real numbers. Define the set of real interval numbers, $\mathfrak{I}$, as

$$\mathfrak{I} = \{I \mid I = (s,t) \text{ where } s,t \in R \text{ and } s \leq t\}. \tag{1}$$

The interval number $I = (s,t)$ represents the set of real numbers x such that $x \in [s,t]$. In view of this we write $I = [s,t]$.

Let o be any binary operation defined on pairs of elements (s,t) where $s \in R$ and $t \in R$. Then the operation o can be extended from elements in R to the operation $\odot$ defined in $\mathfrak{I}$ as follows. Let $A, B \in \mathfrak{I}$, then

$$A \odot B = \{z \mid z = s \, o \, t \text{ for some } s \in A \text{ and } t \in B\}. \tag{2}$$

For example, the operation of addition on the real numbers can be extended to addition on the interval numbers as follows. Let $A = [v,w]$ and $B = [x,y]$ then

$$A + B = \{z \mid z = s + t \text{ for some } s \in [v,w] \text{ and } t \in [x,y]\}.$$

The same kind of extension can be made for unary operations, and indeed for any kind of operation.

A degenerate interval is an interval of the form [s,s]. Hence there is a one-to-one correspondence between the degenerate intervals and the real numbers. Note that in the preceding discussion the extended operation $\circledcirc$ is exactly the operation o if A and B are both degenerate intervals.

In this report, we are concerned only with the four binary operations +, -, $\boxtimes$, /. Let A = [v,w] and B = [x,y] be elements of $\mathfrak{I}$. By applying Eq. 2 to these operators, it can be shown that (3, pp. 8-9)

$$[v,w] + [x,y] = [v + x,\ w + y] \tag{3.1}$$

$$[v,w] - [x,y] = [v - y,\ w - x] \tag{3.2}$$

$$[v,w] \boxtimes [x,y] = [\min(vx, vy, wx, wy),\ \max(vx, vy, wx, wy)] \tag{3.3}$$

$$[v,w] / [x,y] = [\min(v/x, v/y, w/x, w/y),\ \max(v/x, v/y, w/x, w/y)] \tag{3.4}$$

Thus operations on interval numbers involve only endpoints. The operations +, -, $\boxtimes$ can be defined for all interval numbers; however, the operation / must remain undefined if the interval [x,y] contains zero.

If the intervals A and B are regarded as bounds for the real variables s and t, respectively, then it can be shown (3, p. 11) that the intervals given by Eqs. 3 are guaranteed bounds on s + t, s - t, s $\boxtimes$ t, and s / t. For example, if s may lie anywhere in the interval [v,w], and t anywhere in the interval [x,y], then s + t must lie in the interval [v + x, w + y]. Thus [v + x, w + y] is a guaranteed bound

for the expression $s + t$. By guaranteed we mean that given any $s \in [v,w]$ and any $t \in [x,y]$, the sum $s + t$ cannot possibly be outside the interval $[v + x, w + y]$. The interval $[v + x, w + y]$ is also the best bound in the sense that there exist specific values of s and t such that the lower bound $v + x$ is attained; the same is true of the upper bound. Thus the bound $[v + x, w + y]$ is as "tight" as possible. The same is true for the other operations in Eqs. 3.

Guaranteed bounds for the range of any real arithmetic expression can be computed by changing the real variables to interval numbers which are guaranteed to contain the domain of the variables, and replacing the real arithmetic operations by interval operations. For example, suppose $s \in [-1, 2]$ and we wish to compute bounds for the expression $s \boxtimes s$. Then using Eq. 3.3 we compute $[-1, 2] \boxtimes [-1, 2]$ to be the interval $[-2, 4]$. The true bound for the range of $s \boxtimes s = s^2$ for $s \in [-1, 2]$ is the interval $[0, 4]$. In this case the computed bound is guaranteed, but is not optimal because there is no value of s between -1 and 2 such that $s^2 < 0$. Now suppose we wish to bound the range of the expression $s \boxtimes t$ for $s \in [-1, 2]$ and $t \in [-1, 2]$. We get the same bound as before, namely $[-2, 4]$; but this time the bound is optimal. It can be shown that this technique of bounding real arithmetic expressions has the following properties (3, p. 11):

(i) A guaranteed bound always results.

(ii) The bound is also optimal if each variable in the expression occurs only once. E.g., the expression $(x + y) \boxtimes z + u/v$

would guarantee an optimal bound, whereas the bound for $(x + y) \boxtimes y + x/y$ might or might not be optimal.

Let $A = [v,w]$ and $B = [x,y]$. The operations $\boxtimes$ and $/$ can be simplified computationally by examining the signs of v, w, x, y. With regard to sign, there are nine cases to be considered. These results for $\boxtimes$ are given in Table 1. The notation $[-,+]$ in the A column means that $v < 0$ and $w \geq 0$, and similarly for x and y in the B column.

Table 1. Sign analysis of multiplication.

Case No.	A [v,w]	B [x,y]	A $\boxtimes$ B =
1	[+,+]	[+,+]	[vx,wy]
2	[+,+]	[-,+]	[wx,wy]
3	[+,+]	[-,-]	[wx,vy]
4	[-,+]	[+,+]	[vy,wy]
5	[-,+]	[-,+]	[min (vy,wx), max (vx,wy)]
6	[-,+]	[-,-]	[wx,vx]
7	[-,-]	[+,+]	[vy,wx]
8	[-,-]	[-,+]	[vy,vx]
9	[-,-]	[-,-]	[wy,vx]

Similarly, in Table 2 are the results for $/$.

Table 2. Sign analysis of division.

Case No.	A [v,w]	B [x,y]	A/B =
1	[+,+]	[+,+]	[v/y,w/x] *
2	[+,+]	[-,+]	undefined
3	[+,+]	[-,-]	[w/y,v/x]
4	[-,+]	[+,+]	[v/x,w/x] **
5	[-,+]	[-,+]	undefined
6	[-,+]	[-,-]	[w/y,v/y]
7	[-,-]	[+,+]	[v/x,w/y] *
8	[-,-]	[-,+]	undefined
9	[-,-]	[-,-]	[w/x,v/y]

* Undefined if either $x = 0$ or $y = 0$.

** Undefined if $x = 0$.

Before concluding this discussion let us note some of the properties of real interval arithmetic.

Lemma 1. Let $A = [v,w]$ and $B = [x,y]$ be in $\mathfrak{I}$. Then

$$A - B = A + (-B)$$

where $-B$ is defined to be $[-y,-x]$.

Proof. $A + (-B) = [v,w] + [-y,-x] = [v-y,w-x] = A - B$.

Interval addition and multiplication are both commutative and associative (3, p. 9), i.e. if $A, B, C \in \mathfrak{I}$, then

$$A + B = B + A$$

$$A\,B = B\,A$$

$$A + (B + C) = (A + B) + C$$

$$A\,(B\,C) = (A\,B)\,C$$

Also interval arithmetic is inclusion monotonic (3, p. 10), i.e. if $A, B, C, D \in \mathfrak{I}$ and $A \subseteq C$ and $B \subseteq D$, then

$$A + B \subseteq C + D$$

$$A - B \subseteq C - D$$

$$A\,B \subseteq C\,D$$

$$A / B \subseteq C / D \qquad \text{provided } 0 \notin D.$$

Also it can be shown that strict containment occurs if both $A \subset C$ and $B \subset D$. Because of this fact, it would be appropriate to say that real interval arithmetic is strictly inclusion monotonic.

Rational Interval Arithmetic.

Just as real arithmetic cannot be performed on any actual computer, neither can real interval arithmetic. In order to develop an interval arithmetic that can be implemented on a computer, we are led to consider

interval arithmetic on a finite set of rational numbers. Let Q denote the set of rational numbers, and let M be any finite subset of Q whose elements are distinct. Then M can be written as

$$M = \{x_0, x_1, \ldots, x_p\}$$

where, because the x_i are distinct, we may assume $x_0 < x_1 < \ldots < x_{p-1} < x_p$. We also associate with this set the special elements $-K$ and $+K$ such that $-K = x_0$ and $+K = x_p$. (This is simply a notational convenience and does not necessarily imply that $-K = -(+K)$.) Let x be any element x_i of M. Then we will use the following notation: $x^- = x_{i-1}$ provided $x \neq -K$, and $x^+ = x_{i+1}$ provided $x_i \neq +K$. Thus x^- and x^+ are, respectively, the next numbers smaller and larger than x.

Now we define the set, $\mathfrak{I}_M$, of <u>rational interval numbers</u> on M as

$$\mathfrak{I}_M = \{ I \mid I = (s,t);\ s,t \in M \text{ and } s \leq t \}.$$

We also want to define the operations +, -, ⊠, / on $\mathfrak{I}_M$ so as to preserve the guaranteed bounding properties of real interval arithmetic. The definitions in Eqs. 3 cannot be used because $\mathfrak{I}_M$ is not closed with respect to these operations. In particular, there are two kinds of violations of closure which we shall call violation of magnitude, and violation of "betweenness". Let us denote by $[\alpha,\beta]$ the result of applying one of the operations of Eqs. 3 to a pair of intervals in $\mathfrak{I}_M$ and allowing $[\alpha,\beta]$ to be any element of $\mathfrak{I}$. A magnitude violation occurs if either $\alpha < -K$ or $\beta > +K$. For example, $[+K,+K] + [+K,+K] = [\alpha,\beta] = [2(+K), 2(+K)]$ and both α and β are greater than $+K$. Hence β, but not α, commits a violation of magnitude.

A violation of betweenness occurs in the following way. The endpoint, say α, exhibits the violation of betweenness if it satisfies the condition $-K \leq x_i < \alpha < x_{i+1} \leq +K$ for some i between 0 and p-1. Thus α lies strictly between two successive elements of M. Now let us consider the following definition of $+$ on $\mathcal{S}_M$. Let $A = [v,w]$ and $B = [x,y]$ be in $\mathcal{S}_M$. Then

$$[v,w] + [x,y] = \Big[\max\{z \mid z \leq v + x,\ z \in M\},\ \min\{z \mid z \geq w + y,\ z \in M\}\Big] = [a,b] \qquad (4.1)$$

provided $-K \leq v + x$ and $w + y \leq +K$. Thus the interval $[a,b]$ is the interval of smallest width in $\mathcal{S}_M$ which contains the interval $[v+x,w+y]$. The property of guaranteed bounding is preserved for addition because by Eq. 4.1 we have $[v + x,\ w + y] \subseteq [a,b]$. Because it may be that $[v + x,\ w + y] \subset [a,b]$, optimality in the sense of Eqs. 3 is not necessarily preserved. However, the interval $C = [a,b]$ is optimal with respect to M in the sense that there exists no other interval $D \in \mathcal{S}_M$, different from C, such that $[v + x,\ w + y] \subset D \subset C$. Said another way $[a,b]$ is the best possible interval that can be computed, or represented, from the rational numbers in M.

Thus violations of betweenness are overcome by the max and min operations over M of Eq. 4.1. In order to note violations of magnitude, we associate with the addition operation an integer F. If no violation occurs, $F = 0$ and $[a,b]$ is computed according to Eq. 4.1. However, three kinds of violation may occur: in the left endpoint, the right

endpoint, or both. In these cases the following actions are taken by the addition operation:

i) If $v + x < -K$, set $F = 1$, and the result to $[-K,b]$.

ii) If $w + y > +K$, set $F = 2$, and the result to $[a,+K]$.

iii) If $v + x < -K$ and $w + y > +K$, set $F = 3$ and the result to $[-K, +K]$.

By this scheme, the best possible results are computed.

In similar fashion we can define on $\mathfrak{I}_M$ the operations $-$, $\boxtimes$, $/$ as follows:

$$[v,w] - [x,y] = \left[\max \{ z \mid z \leq v - y, z \in M\}, \min \{ z \mid z \geq w - x, z \in M\}\right] \tag{4.2}$$

provided $-K \leq v - y$ and $w - x \leq +K$.

$$[v,w] \boxtimes [x,y] = \left[\max \{ z \mid z \leq \min (vx, vy, wx, wy), z \in M\}, \min \{ z \mid z \geq \max (vx, vy, wx, wy), z \in M\}\right] \tag{4.3}$$

provided $-K \leq \min (vx, vy, wx, wy)$ and $\max (vx, vy, wx, wy) \leq +K$.

$$[v,w] / [x,y] = \left[\max \{ z \mid z \leq \min (v/x, v/y, w/x, w/y), z \in M\}, \min \{ z \mid z \geq \max (v/x, v/y, w/x, w/y), z \in M\}\right] \tag{4.4}$$

provided $0 \notin [x,y]$ and $-K \leq \min (v/x, v/y, w/x, w/y)$ and $\max (v/x, v/y, w/x, w/y) \leq +K$.

The operations $\boxtimes$ and $/$ can be analyzed according to sign exactly as shown in Tables 1 and 2. The treatment of the flag F for violations of magnitude in these operations and the results produced are given in

Table 3. In this table the interval [a,b] is the result produced by any of Eqs. 4.

Table 3. Magnitude violation flag.

F	Operation	Result	Caused by
0	$+, -, \boxtimes, /$	[a,b]	No violation
1	+	[-K,b]	Violation left
2	+	[a,+K]	Violation right
3	+	[-K,+K]	Violation left and right
4	-	[-K,b]	Violation left
5	-	[a,+K]	Violation right
6	-	[-K,+K]	Violation left and right
7	$\boxtimes$	[-K,b]	Violation left
8	$\boxtimes$	[a,+K]	Violation right
9	$\boxtimes$	[-K,+K]	Violation left and right
10	/	[-K,b]	Violation left
11	/	[a,+K]	Violation right
12	/	[-K,+K]	Violation left and right
13	/	[-K,+K]	Violation left and right due to $0 \in [x,y]$

Let us now note some of the properties of rational interval arithmetic as defined by Eqs. 4 and Table 3. First note that the relation A - B = A + (-B) in Lemma 1 may not hold in $\mathfrak{I}_M$, because $x \in M$ does not necessarily imply that $-x \in M$. Therefore, we state the following lemma for $\mathfrak{I}_M$.

Lemma 2. Suppose that M has the property that for every $x \in M$ then $-x \in M$. Then for all $A, B \in \mathfrak{I}_M$,

$$A - B = A + (-B)$$

provided $F = 0$. If $F \neq 0$, the result of the above operation is correct if F is increased by three. (F is set by $+$.)

Proof. Let $A = [v,w]$ and $B = [x,y]$ be in $\mathfrak{I}_M$. Then by our assumption about M, $-B = [-y, -x] \in \mathfrak{I}_M$; so $A + (-B)$ is well defined on $\mathfrak{I}_M$. Now apply Eq. 4.1 to get

$$[v,w] + [-y, -x] = [\overline{a},\overline{b}]$$

and suppose $F = 0$ (from $+$). Because $F = 0$ we know that $-K \leq v - y$ and $w - x \leq +K$ which implies that $A - B$ does not commit a magnitude violation; hence $F = 0$ is also the correct value of F (from $-$). Now let $[a,b]$ be the result of applying Eq. 4.2 to $A - B$. Now we must show that $[\overline{a},\overline{b}] = [a,b]$, i.e. $\overline{a} = a$ and $\overline{b} = b$.

$$\overline{a} = \max \{ z \mid z \leq v + (-y),\ z \in M \}$$

$$= \max \{ z \mid z \leq v - y,\ z \in M \} = a.$$

A similar relation shows that $b = \overline{b}$.

Now suppose $F = 1$ as a result of $[v,w] + [-y,-x]$. This implies that $v + (-y) < -K$ and $w + (-x) \leq +K$. Increase F by 3 obtaining $F = 4$. Now $v + (-y) < -K$ implies $v - y < -K$ which is a magnitude violation in the left endpoint of subtract. Now since $F = 4$, it agrees with Table 3 and $\overline{b} = b$ as argued just previously. Similar arguments hold for $F = 2$ and $F = 3$. This completes the proof.

Lemma 3. Rational interval addition is commutative.

Proof. Let $A = [v,w]$ and $B = [x,y]$ be in $\mathfrak{I}_M$. Suppose $F = 0$.

$$\begin{aligned} A + B &= \left[\max\{ z \mid z \leq v + x,\ z \in M\}, \min\{ z \mid z \geq w + y, z \in M\}\right] \\ &= \left[\max\{ z \mid z \leq x + v,\ z \in M\}, \min\{ z \mid z \geq y + w, z \in M\}\right] \\ &= B + A. \end{aligned}$$

Now suppose that in computing $A + B$, F is set to 1. This implies that $v + x < -K$. Hence $x + v < -K$ and $B + A$ also has a left violation. Also if $F = 1$ from $A + B$, then $w + y \leq +K$ which implies that $y + w \leq +K$ and the right endpoint of $B + A$ is computed as required. Similar results hold for $F = 2$ and $F = 3$.

Lemma 4. Rational interval multiplication is commutative.

Proof. Let $A = [v,w]$ and $B = [x,y]$ be in $\mathfrak{I}_M$. Suppose $F = 0$.

$$\begin{aligned} A \boxtimes B &= \left[\max\{ z \mid z \leq \min(vx, vy, wx, wy),\ z \in M\},\right. \\ &\qquad \left.\min\{ z \mid z \geq \max(vx, vy, wx, wy)\}\right] \\ &= \left[\max\{ z \mid z \leq \min(xv, yv, xw, yw),\ z \in M\},\right. \\ &\qquad \left.\min\{ z \mid z \geq \max(xv, yv, xw, yw)\}\right] \\ &= B \boxtimes A \end{aligned}$$

Arguments like those of Lemma 3 hold for the values of F being 7, 8, 9. This completes the proof.

The associative law does not hold for rational interval addition. Suppose

$M = \{0, 1, 2, \ldots, 9\} \cup \{100, 200, \ldots, 900\}$. Let $A = B = [1,1]$ and $C = [100,100]$. Then

$$\begin{aligned}(A + B) + C &= ([1,1] + [1,1]) + [100,100] \\ &= [2,2] + [100,100] \\ &= [100,200]\end{aligned}$$

while

$$\begin{aligned}A + (B + C) &= [1,1] + ([1,1] + [100,100]) \\ &= [1,1] + [100,200] \\ &= [100,300].\end{aligned}$$

This example is not as artificial as it may first appear. An analogous situation can occur with floating point numbers on a computer.

Neither does the associative law hold for multiplication. Suppose $M = \{0, .1, .2, \ldots, .9\} \cup \{1, 2, \ldots, 9\} \cup \{100, 200, \ldots, 900\}$. Let $A = [2,2]$, $B = [.9, .9]$ and $C = [100,100]$. Then

$$\begin{aligned}(AB)\,C &= ([2,2]\,[.9, .9])\,[100,100] \\ &= [1,2]\,[100,100] \\ &= [100,200]\end{aligned}$$

while

$$\begin{aligned}A\,(BC) &= [2,2]\,([.9, .9]\,[100,100]) \\ &= [2,2]\,[9,100] \\ &= [9,200].\end{aligned}$$

The rational interval arithmetic defined by Eqs. 4 and Table 3 is also inclusion monotonic. However, the strict inclusion monotonicity of real interval arithmetic is not preserved. Consider addition. Eq. 4.1 makes it clear that if $A \subseteq C$ and $B \subseteq D$, then certainly $A + B \subseteq C + D$. However, it may be that $A \subset C$ and $B \subset D$ and that $A + B = C + D$ rather than $A + B \subset C + D$ as in real interval arithmetic. For example, suppose that $M = \{-900, -800, \ldots, -100\} \cup \{-9, -8, \ldots, -1\} \cup \{0,1,2,\ldots,9\} \cup \{100,200,\ldots,900\}$. Let $A = [-3,3]$, $B = [-4,4]$, $C = [-7,7]$, and $D = [-8,8]$ so that $A \subset C$ and $B \subset D$. Then $A + C = [-3,3] + [-7,7] = [-100,100] = [-4,4] + [-8,8] = B + D$.

A digital computer operates entirely on a finite set of rational numbers. Hence by implementing Eqs. 4 and Table 3 we can produce an interval arithmetic that can be used on a digital computer to produce guaranteed error bounding. When implemented, the interval arithmetic will have the properties of rational interval arithmetic which we have discussed. The rest of this paper then is concerned with a set of Algol procedures which implement Eqs. 4 and Table 3 on the B5500; and with proving that, in fact, these procedures correctly perform these specified operations.

II. THE B5500 ENVIRONMENT

The following discussion is directly related to the properties of the Burroughs B5500 digital computer. However, several of the points are applicable or extendable to computers with similar properties.

The Set M.

Any digital computer is restricted, for all practical purposes, so that it can operate directly only on a finite set of rational numbers. This follows from the fact that any "arithmetic register" can contain only a finite number of discrete digits. Also most computers make a distinction between floating-point and fixed-point numbers. Since most numerical computation is done in the floating-point mode, we will consider interval arithmetic only on these numbers. Further, on most computers a floating-point number must be "normalized" so that the arithmetic registers will operate on it correctly. (Normalization may mean different things on different machines.) It becomes apparent that for most machines it would not be practical to implement an interval arithmetic, as previously defined, that would operate on <u>all</u> possible machine numbers. Hence the interval arithmetic is restricted to a subset of the machine numbers.

Every number that can be stored in the B5500 is stored in a 48 bit word. We shall call this bit configuration an <u>operand</u> to distinguish it from the number or value it represents. The first three binary digits of

the operand are viewed independently by the B5500. The rest of the bits are interpreted by the machine as a string of 15 octal digits. Thus a B5500 operand of 48 bits may be viewed conceptually as follows:

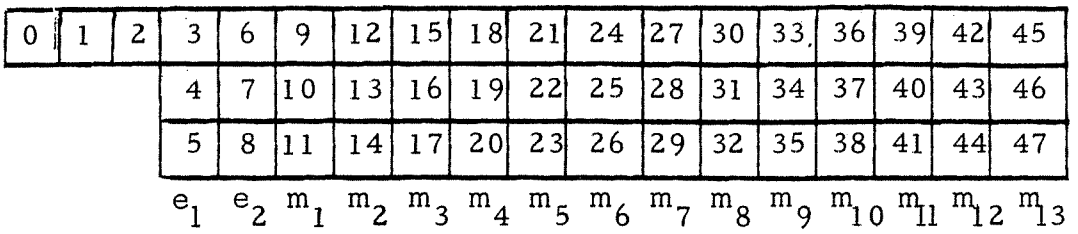

Figure 1. B5500 numeric operand.

Bit 0 is a flag bit which is always 0 for a B5500 operand so as to distinguish it from an instruction. Bits 1 and 2 are sign bits of the mantissa and exponent, respectively; 0 denotes +, and 1 denotes -. The first two octal digits, e_1, e_2 of the 15 digit string denote the magnitude of the exponent, and the remaining octal digits, $m_1,\ldots,m_{13}$, denote the magnitude of the mantissa. The octal point is to the right of m_{13}. Thus the mantissa is an ordered set $(m_1,m_2,\ldots,m_{13})$ of non-negative octal digits, 0, 1, 2, 3, 4, 5, 6, 7; likewise the exponent is an ordered pair of non-negative octal digits (e_1,e_2). If we denote the sign of the mantissa, bit 1, by s_m, and the sign of the exponent, bit 2, by s_e, the numeric operand will then be represented in our notation as $(s_m, s_e, (e_1,e_2), (m_1,\ldots,m_{13}))$. By conceptually associating the sign of the exponent with each digit of the exponent, and likewise for the mantissa we can represent the operand as the pair (e,m) where e

denotes the entire signed exponent and m the entire mantissa with associated sign. Note that both e and m are integers.

$$e = e_1 8 + e_2 \tag{5.1}$$

and

$$m = \sum_{i=1}^{13} m_i 8^{13-i} . \tag{5.2}$$

Even though these and later expressions involve mixed base 8 and base 10 notation, the intent seems clearer than if we were to write the 8's as 10_8 or just plain 10. For the rational number represented by the numeric operand (e,m) we use the notation r(e,m). On the B5500 this number is given as follows:

$$r(e,m) = m\, 8^e = \left(\sum_{i=1}^{13} m_i 8^{13-i} \right) 8^{e_1 8 + e_2} . \tag{6}$$

A B5500 numeric operand is said to be <u>normalized</u> if and only if the first <u>octal</u> digit of the mantissa, m_1, is not zero. For example, the operand (+, +, (1,4), (1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)) is normalized, whereas (+, +, (0, 0), (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)) is not. Let us denote by N' the set of all normalized operands,

$$N' = \{ (e,m) \mid m_1 \neq 0 \} . \tag{7}$$

Let Z be the operand whose exponent and mantissa are all zeros and whose sign bits are both +. The rational number represented by this operand Z is zero. Now let $N = N' \cup \{Z\}$. The set M of rational

numbers on which we shall implement the B5500 interval arithmetic is the following:

$$M = \{ x \mid x = r(e,m),\ (e,m) \in N \}. \tag{8}$$

It can easily be verified that there is a one-to-one correspondence between elements of N and M, and that the elements of M are distinct. Also $+K = 7777777777777 \cdot 8^{77}$ and $-K = -(+K)$. For later purposes we also need to distinguish the elements $0^+ = \epsilon = 1000000000000 \cdot 8^{-77}$ and $0^- = -\epsilon = -1000000000000 \cdot 8^{-77}$.

Integer Operands.

Some computers make a distinction between fixed-point (integer) and floating-point operands. On the B5500 this distinction is made only conceptually. The 48 bit word is interpreted exactly as for floating-point, so that an integer operand is simply a numeric operand whose exponent is zero. Hence the B5500 has only one set of arithmetic commands which work equally well on integer, floating-point or mixed operands. Note that, in general, integer operands are not normalized. Hence most non-zero integer operands are not in N'. However, for every such integer operand which is not in N', there is an equivalent, normalized operand which is in N'. Thus all integers which can be represented on the B5500 by integer operands are in M.

Double-Precision Operands.

Two 48 bit words are used to contain a B5500 double-precision operand. The most significant word appears as a single-precision operand, i.e. it has the same exponent, mantissa, and two sign fields. The least significant word has its exponent and two sign fields all zeros, and the mantissa of the double-precision operand is continued by using the mantissa of the least significant word. Thus a double-precision operand is a pair of numeric operands $((e,m^{(1)}), (0,m^{(2)}))$ where $e = (e_1, e_2)$, $m^{(1)} = (m_1,\ldots,m_{13})$, and where $m^{(2)} = (m_{14},\ldots,m_{26})$ with associated sign bits found in the first word. (Note that the least significant word always appears as a non-negative integer if viewed independently of the most significant word.) For a double-precision operand we will use the notation $(e,m^{(1)}, m^{(2)})$, a simplification of the above. The rational number represented by this operand is

$$r(e,m^{(1)}, m^{(2)}) = (m^{(1)} + m^{(2)})\, 8^e = \left(\sum_{i=1}^{26} m_i 8^{13-i} \right) 8^{e_1 8 + e_2}. \qquad (9)$$

As with single-precision, a double-precision operand is said to be <u>normalized</u> if and only if $m_1 \neq 0$.

Let $u = r(e,m^{(1)}, m^{(2)})$. Then we make the following definitions:

$$\underline{u} = \max \{ z \mid z \leq u,\ z \in M \} \qquad (10.1)$$

$$\overline{u} = \min \{ z \mid z \geq u,\ z \in M \}. \qquad (10.2)$$

Note that $\underline{u}$ and $\overline{u}$ are rational numbers, not operands. Henceforth we will also use the following notational conventions. The + operator of r (e,m) + r (e',m') denotes real addition, and likewise for double-precision operands. The + operator of (e,m) + (e',m') denotes single-precision addition as performed by the B5500 arithmetic unit. Also $(e,m^{(1)}, m^{(2)}) + (e', m^{(1)'}, m^{(2)'})$ denotes B5500 double-precision addition. The same holds for the other operators -, ⊠, /. Also we will make use of the unnormalized single-precision operand (e,1), i.e. the mantissa is 0000000000001.

We now present some lemmas concerning relations among elements of the sets M, N, double-precision operands and the numbers represented by them.

Lemma 5. Let $x \in M$, and let $(e,m) \in N$ such that $x = r(e,m)$. Suppose $x \neq 0$. Then

(i) $x^{+} = r((e,m) + (e,1))$ provided $r(e,m) \neq +K$ and $r(e,m) \neq 0^{-}$.

(ii) $x^{-} = r((e,m) - (e,1))$ provided $r(e,m) \neq -K$ and $r(e,m) = 0^{+}$.

Proof. Consider first (i). First of all we claim that the single-precision addition (e,m) + (e,1) is within the capability of the B5500 even though (e,1) is unnormalized, and we further claim that r ((e,m) + (e,1)) = r (e,m) + r (e,1). Notice that the exponents of the two operands are equal in magnitude and sign. Now suppose that the sign of M is +. This implies that $m > 0$ since $m_1 \neq 0$. The sign of 1 in (e,1) is also +.

Now the B5500 add proceeds as follows. The exponent of the answer, e_a, is $e_a = e$. Then the mantissas are added. Suppose the sum does not exceed thirteen octal digits. Then the mantissa of the answer m_a is $m_a = m+1$. Thus the resulting operand is $(e, m+1)$. See (2, p.6-9). Hence the following equalities hold:

$$\begin{aligned} r\,(\,(e,m) + (e,1)\,) &= r\,(e, m+1) \\ &= (m+1)\,8^e \\ &= m\,8^e + 1 \cdot 8^e \\ &= r\,(e,m) + r\,(e,1) \end{aligned} \tag{11}$$

Notice also that if m is normalized, then so is $m+1$.

Now suppose that when the mantissas are added the sum exceeds 13 octal digits. Then the fourteen digit result is shifted right one place, rounded to thirteen digits, and the exponent increased by one. Now, the sum of m and 1 can exceed thirteen octal digits only if $m = 7777777777777$. Thus the answer would be $(e+1,\ 1000000000000)$. This is a computable operand provided $e < 77$. If $e = 77$, then $r\,(e,m) = +K$; but this case has been excluded. Now

$$\begin{aligned} r\,(\,(e,m) + (e,1)\,) &= r\,(\,(e,7777777777777) + (e,1)\,) \\ &= r\,(e+1,\ 1000000000000) \\ &= (8^{12})\ 8^{e+1} \\ &= 8^{13}\ 8^e \\ &= (8^{13} - 1 + 1)\,8^e \\ &= (8^{13} - 1)\,8^e + 1 \cdot 8^e \end{aligned}$$

$$= 7777777777777 \cdot 8^e + 1 \cdot 8^e$$

$$= m\, 8^e + 1 \cdot 8^e$$

$$= r(e,m) + r(e,1).$$

Also again the result is normalized. Now suppose that the sign of m is -, hence $m < 0$. In this case the exponent of the result is e and the mantissa is the difference between m and 1, i.e. $m + 1$ $(m < 0)$. Hence the resulting operand is $(e, m+1)$. Since m and 1 are of opposite signs, the new mantissa will be of smaller magnitude than m. Therefore the new mantissa does not exceed thirteen digits, so Eqs. 11 apply and $r((e,m) + (e,1)) = r(e,m) + r(e,1)$. In this case, however, it is possible for the result operand $(e, m+1)$ to be unnormalized. This happens if and only if $m = -1000000000000$ in which case the result operand is $(e, 0777777777777)$. However, even in this case $r(e, m+1) \in M$ if $e \neq -77$ because an equivalent normalized operand is $(e-1,\ 7777777777770)$. We have ruled out the case $(-77,\ -1000000000000)$ because the number represented by this operand is 0^-. Summarizing, (i) follows readily: Since (e,m) is normalized,

$$\begin{aligned} x^+ &= (m+1)\, 8^e \\ &= m\, 8^e + 8^e \\ &= r(e,m) + r(e,1) \\ &= r((e,m) + (e,1)) \end{aligned} \tag{13}$$

provided $x \neq +K$ and $x \neq 0^-$. Notice that it is essential that (e,m) be normalized before the addition of $(e,1)$. Otherwise a smaller single-precision upper bound can be obtained for $r(e,m)$ by first normalizing (e,m) and then adding $(e,1)$. Part (ii) follows from precisely the same kind of argument.

Lemma 6. Let $(e,m^{(1)}, m^{(2)})$ be a normalized double-precision operand and let $u = r(e,m^{(1)}, m^{(2)})$. Then $\overline{u}$ and $\underline{u}$ (Eqs. 10) are

$$\overline{u} = \begin{cases} r((e,m^{(1)}) + (e,1)) & \text{if } m^{(2)} > 0, r(e,m^{(1)}) \neq +K, r(e,m^{(1)}) \neq 0^- \\ r(e,m^{(1)}) & \text{if } m^{(2)} \leq 0 \end{cases}$$

$$\underline{u} = \begin{cases} r((e,m^{(1)}) - (e,1)) & \text{if } m^{(2)} < 0, r(e,m^{(1)}) \neq -K, r(e,m^{(1)}) \neq 0^+ \\ r(e,m^{(1)}) & \text{if } m^{(2)} \geq 0. \end{cases}$$

Proof. Let $x = r(e,m^{(1)})$. Consider first $\overline{u}$. Suppose $m^{(2)} > 0$, $r(e,m^{(1)}) \neq +K$, and $r(e,m^{(1)}) \neq 0^-$. Then since $m^{(2)} > 0$, $m^{(1)} > 0$. We now claim that $x < u < x^+$. If this is so, $\overline{u} = \min \{ z \mid z \geq u, z \in M \}$ $= x^+$ and $x^+ = r((e,m^{(1)}) + (e,1))$ by Lemma 5. To see that $x < u < x^+$ recall that the octal point is between $m^{(1)}$ and $m^{(2)}$ and observe the following:

$$x = r(e,m^{(1)}) = m^{(1)}\,8^e < (m^{(1)} + m^{(2)})\,8^e = u < (m^{(1)} + 1)\,8^e$$

$$= m^{(1)}\,8^e + 8^e = r(e,m^{(1)}) + r(e,1) = r((e,m^{(1)}) + (e,1)) = x^+. \quad (14)$$

Now suppose $m^{(2)} = 0$. In this case $u = r(e,m^{(1)},0) = r(e,m^{(1)}) = x$, so

$$\overline{u} = \min\{\, z \mid z \geq u,\ z \in M \,\} = x = r(e,m^{(1)}).$$

Finally suppose that $m^{(2)} < 0$. Then also $m^{(1)} < 0$. Now we have that $x^- < u < x$, because

$$x^- = r((e,m^{(1)}) - (e,1)) = r(e,m^{(1)}) - r(e,1) = m^{(1)}\,8^e - 8^e$$

$$< m^{(1)}\,8^e + m^{(2)}\,8^e = u < m^{(1)}\,8^e = r(e,m^{(1)}) = x. \quad (15)$$

Hence $\overline{u} = \min\{\, z \mid z \geq u,\ z \in M \,\} = x = r(e,m^{(1)})$.

Precisely the same kind of arguments hold for $\underline{u}$. This comples the proof.

Suppose (e,m) and $(\overline{e},\overline{m})$ are normalized, single-precision operands. Then we shall say that their mantissas are <u>disjoint</u> if and only if $|\, e - \overline{e}\,| > 12_{10}$. (The term is motivated by the shifting necessary in an addition to align the octal points. Cf. the alignments in the next proof.) In the following lemma, we will show that by using double-precision addition, an exact sum can be computed for normalized, single-precision operands whose mantissas are not disjoint.

Lemma 7. Suppose (e,m) and $(\overline{e},\overline{m})$ are single-precision operands in N whose mantissas are not disjoint. Form the double-precision operands $(e,m,0)$ and $(\overline{e},\overline{m},0)$, and suppose $(e,m,0) + (\overline{e},\overline{m},0) = (a,b,c)$ and no exponent overflow or underflow occurs. Then $r(e,m) + r(\overline{e},\overline{m}) = r(a,b,c)$.

Proof. Assume without loss of generality that $0 \leq e - \overline{e} \leq 12$. Notice that since $e - \overline{e} \leq 12$, the mantissas after being aligned for addition must "overlap" by at least one digit. The alignments are as follows:

$e - \overline{e} = 0$

$$\begin{array}{l} m_1\ m_2\ m_3 \ldots m_{13}\ \overbrace{0\ .\ .\ .\ 0}^{13\text{ zeros}} \\ \overline{m}_1\ \overline{m}_2\ \overline{m}_3 \quad \overline{m}_{13}\ 0 \qquad\qquad 0 \end{array}$$

$e - \overline{e} = 1$

$$\begin{array}{l} m_1\ m_2\ m_3 \ldots m_{13}\ 0\ .\ .\ .\ 0 \\ \overline{m}_1\ \overline{m}_2 \quad \overline{m}_{12}\ \overline{m}_{13}\ \underbrace{0 \qquad\qquad 0}_{12\text{ zeros}} \end{array}$$

$e - \overline{e} = 2$

$$\begin{array}{l} m_1\ m_2\ m_3 \ldots m_{13}\ 0\ .\ .\ .\ 0 \\ \overline{m}_1 \quad \overline{m}_{11}\ \overline{m}_{12}\ \overline{m}_{13}\ \underbrace{0 \qquad 0}_{11\text{ zeros}} \end{array}$$

$\vdots$

$e - \overline{e} = 12$

$$\begin{array}{l} m_1\ m_2\ m_3 \ldots m_{13}\ 0\ .\ .\ .\ 0 \\ \overline{m}_1\ \overline{m}_2\ \overline{m}_3 \qquad \overline{m}_{13}\ 0 \end{array}$$

After alignment, the mantissa $\overline{m}$ has been shifted right by $e - \overline{e}$ octal places. From the above, we see that the resulting sums contain at least $13_{10} + (e - \overline{e})$ octal digits counting leading zeros. Let us denote by $\hat{m}$ the shifted mantissa of $\overline{m}$. Now if m and $\hat{m}$ are of different signs, essentially a subtraction occurs between m and $\hat{m}$. In this case $m + \hat{m}$ can contain <u>at most</u> $13 + (e - \overline{e})$ digits; since $e - \overline{e}$ can be at most 12, $m + \hat{m}$ can contain at most 25 octal digits. Now since the B5500 result (a,b,c) can contain 26 octal digits in the mantissa, the lemma is proved for m and $\overline{m}$ of opposing signs.

Now suppose m and $\overline{m}$ are of the same sign. Then since m and $\hat{m}$ are added, a carry digit is possible. We will show that for any m and $\hat{m}$, this carry is at most one digit so that in any resulting mantissa we can have at most $13 + (e - \overline{e}) + 1 \leq 13 + 12 + 1 = 26$ digits. Hence again the result can be expressed exactly in (a,b,c). The greatest possible carry must occur when m and $\overline{m}$ are as large as possible and when $e = \overline{e}$. Thus let $m_1 = m_2 = \ldots = \overline{m}_1 = \overline{m}_2 = \ldots = 7$. Now (e,m) represents $7777777777777 \cdot 8^e$ and so does $(\overline{e},\overline{m})$. So adding these values,

$$7777777777777 \cdot 8^e + 7777777777777 \cdot 8^e$$

$$< 10000000000000 \cdot 8^e + 7777777777777 \cdot 8^e$$

$$= 17777777777777 \cdot 8^e .$$

Hence the carry in adding $m + \hat{m}$ is at most one digit. This completes the proof.

Next we wish to prove a similar result for double-precision multiplication. The lemma rests on the following assumption: In general, a B5500 double-precision multiply on the operands $(e,m^{(1)}, m^{(2)}) \boxtimes (\overline{e},\overline{m}^{(1)}, \overline{m}^{(2)})$ may generate an error in the result as large as one in the twenty-fifth digit position (2, p. 6-12). However, we have <u>assumed</u> that no error is generated for operands of the form $(e,m,0) \boxtimes (\overline{e},\overline{m},0)$. This was verified experimentally for $(0,7777777777777,0) \boxtimes (0,7777777777777,0)$ and $(0,1000000000001,0) \boxtimes (0,1000000000001,0)$.

Lemma 8. Suppose (e,m) and $(\overline{e},\overline{m})$ are single-precision operands in N. Form the double-precision operands $(e,m,0)$ and $(\overline{e},\overline{m},0)$, and suppose $(e,m,0) \boxtimes (\overline{e},\overline{m},0) = (a,b,c)$ and no exponent overflow or underflow occurs. Then $r(a,b,c) = r(e,m) \boxtimes r(\overline{e},\overline{m})$.

Proof. The B5500 double-precision multiply adds exponents and multiplies mantissas. The fact that no overflow or underflow occurs insures that the exponent can be added. It remains to show that the product of the mantissas contains at most 26 octal digits. Without loss of generality, assume both mantissas are greater than zero. Thus

$$m \boxtimes \overline{m} \leq (8^{13} - 1) \boxtimes (8^{13} - 1) < 8^{26}.$$

But in octal, 8^{26} is a 1 followed by 26 zeros, which is the smallest possible, positive 27 digit octal number with a non-zero leading digit. Now $m \boxtimes \overline{m}$ is less than this number by an integer amount. Hence $m \boxtimes \overline{m}$ must have less than 27 octal digits, i.e. less than or equal 26. This completes the proof (under the previously mentioned assumption).

Next we need a result concerning the behavior of double-precision division when applied to single-precision operands. We are faced with the octal analog of decimal $1/3 = .333 \ldots$. Accordingly, there is no result for division corresponding to the exact results of addition and multiplication given in Lemmas 7 and 8. A result sufficient for our needs, however, can be obtained by considering only when the least significant word of the quotient assumes its extreme values, namely 0000000000000 and 7777777777777.

Lemma 9. Let (e_a, a) and (e_b, b) be operands in N where $(e_b, b) \neq Z$. Form the double-precision operands $(e_a, a, 0)$ and $(e_b, b, 0)$, and suppose that $(e_a, a, 0) / (e_b, b, 0) = (d, c^{(1)}, c^{(2)})$, and no exponent overflow or underflow occurs. Also suppose, as is the case on the B5500, that

i) $(d, c^{(1)}, c^{(2)})$ is normalized or zero.

ii) the resulting quotient mantissa $c_1, c_2, \ldots, c_{26}$ contains the first 26 non-zero digits of the actual quotient $r(e_a, a, 0) / r(e_b, b, 0)$

(2, pp. 6-12--6-13). Finally suppose that

$$\frac{r(e_a, a, 0)}{r(e_b, b, 0)} = r(d, c^{(1)}, c^{(2)}) + R$$

where R is a real number of arbitrary precision. Then

(i) if $c^{(2)} = 0$ (i.e. $c_{14} = c_{15} = \ldots = c_{26} = 0$), then $R = 0$

(ii) if $c_{14} = c_{15} = \ldots = c_{26} = 7$, then the (infinite) quotient does not consist of all 7's beyond c_{26}.

Proof. If $r(e_a, a) = 0$, (i) is clear and (ii) does not apply. Therefore assume $r(e_a, a) \neq 0$. Let the mantissas be denoted as follows: a is the string of 13 digits $a_1 a_2 \ldots a_{13}$, b is $b_1 b_2 \ldots b_{13}$, $c^{(1)}$ is $c_1 c_2 \ldots c_{13}$, $c^{(2)}$ is $c_{14} c_{15} \ldots c_{26}$, and the total quotient mantissa is the string $c_1 c_2 \ldots c_{13} c_{14} c_{15} \ldots c_{26}$. Because $(e_a, a, 0)$, $(e_b, b, 0)$, and $(d, c^{(1)}, c^{(2)})$ are normalized, we have that $a_1 \neq 0$, $b_1 \neq 0$, and $c_1 \neq 0$. Since there is no overflow or underflow, we may ignore the exponents, i.e. the digits of the quotient mantissa do not depend on the exponents.

We prove first (i). Let $c^{(2)} = 0$. Consider the division schematically shown by the process of long division:

$$
\begin{array}{l}
\qquad\qquad\qquad\quad c_1 \cdots\cdots\cdots\cdots c_{13} \\
b_1 \ldots b_{13} \;\overline{\big|\, a_1 \cdots\cdots\cdots\cdots a_{13}\; 0 \ldots 0} \\
\qquad\quad -(c_1 \ldots c_{13}) \boxtimes (b_1 \ldots b_{13}) \\
\qquad\qquad\qquad\quad s
\end{array}
$$

where s, of up to 13 digits, is the remainder of the division after $c^{(1)}$ has appeared in the quotient. While c_1 may indeed appear over a_2, we have, without loss of generality shown it over a_1. Suppose $s \neq 0$. Then we may continue to being down zeros from the (augmented) dividend until the divisor of only 13 digits will go into $s\ 0 \ldots 0$ of at least 14 digits. Thus a non-zero digit would appear as one of c_{14}, $c_{15}, \ldots, c_{26}$. But this contradicts $c^{(2)} = 0$. Hence $s = 0$, and therefore $R = 0$.

For the proof of (ii) let $c^{(2)} = 7 \ldots . 7$, and suppose the quotient does consist of all 7's beyond c_{26}. Consider the same division as before but with $c_{14} = 7$ placed in the quotient and the next remainder t shown:

$$
\begin{array}{l}
\qquad\qquad\qquad\quad c_1 \cdots\cdots\cdots\cdots c_{13}\ 7 \\
b_1 \ldots b_{13} \;\overline{\big|\, a_1 \cdots\cdots\cdots\cdots a_{13}\; 0 \ldots 0} \\
\qquad\quad -(c_1 \ldots c_{13}) \boxtimes (b_1 \ldots b_{13}) \\
\qquad\qquad\qquad\quad s\ 0 \\
\qquad\qquad\qquad\quad -7 \boxtimes (b_1 \ldots b_{13}) \\
\qquad\qquad\qquad\qquad\quad t
\end{array}
$$

where t may be of up to 13 digits. But in order to obtain the infinite string of 7's in the quotient t must equal s. Since the quantity s 0

has relative value $10_8 \boxtimes s$ as compared to $7 \boxtimes (b_1 \ldots b_{13})$, we have from the subtraction which gave t

$$10_8 \boxtimes s - 7 \boxtimes (b_1 \ldots b_{13}) = t = s$$

or

$$s = b_1 \ldots b_{13}.$$

This says that c_{13} should be increased by 1. $c^{(2)}$ of the quotient then becomes 0's, a contradiction to $c^{(2)} = 7 \ldots . 7$. This completes the proof.

III. THE ALGOL ENVIRONMENT

Extended Algol.

The interval arithmetic package presented here consists of five procedures, three initialization statements and appropriate global declarations. This code uses several of the features of B5500 Extended Algol (1). We shall now list and explain the five non-Algol 60 features that are used.

1. Conditional statement.

In Algol 60,

< conditional statement > ::= < if statement > |
< if statement > ELSE < statement > |
< if clause > < for statement > |
< label > : < conditional statement >

< if statement > ::= < if clause > < unconditional statement >

In Extended Algol,

< conditional statement > ::= < if statement > |
< if statement > ELSE < statement > |
< label > : < conditional statement >

< if statement > ::= < if clause > < statement >

In short, an Algol 60 < if statement > requires an unconditional statement while Extended Algol allows any statement. Thus

Extended Algol allows the nesting of IF THEN ELSE statements without bracketing the THEN and ELSE clauses in BEGIN...END. The problem of any so-called dangling ELSE is resolved (recursively) by matching each ELSE with the nearest previous unmatched THEN in the source string "unprotected" by a BEGIN END pair. To cause the desired pairings, ELSE < dummy statement > can be inserted. The result, ELSE ELSE, appears in this code. The flow of control for certain conditional statements is the subject of Lemma 13.

2. The MONITOR declaration and < fault statement >.

This is a means of detecting exponent overflow, and taking appropriate action. The declaration

MONITOR EXPOVR

sets aside the identifier EXPOVR as a reserved word (in the block where this declaration occurs) for use in the < fault statement >. In general, < fault statement > ::= < fault type > ← < designational expression >. As used in these procedures it specifies to < fault statement > ::= EXPOVR ← < label >. After execution of a < fault statement >, the < label > is used as a recovery point when an exponent overflow occurs. For example, consider the following code segment:

```
EXPOVR ← L 1;
    .
    .
    .
A ← B + C;
    .
    .
    .
L 1:
```

If, in evaluating the expression B + C, an exponent overflow occurs, control is transferred immediately to the label L 1. The recovery point label may be changed at any point in the program by executing a new < fault statment >. On the B5500 only the arithmetic operations can generate an exponent overflow. On some machines the relational operators, for instance, may generate overflow; this is not true on the B5500.

3. The FILL statement.

Only one such statement is used: FILL Q7QLIMNO [*] WITH OCT0777777777777777, OCT1771000000000000. This is a means of inserting octal constants into Q7QLIMNO[0] and Q7QLIMNO[1]. The value of Q7QLIMNO[0] is +K and the value of Q7QLIMNO[1] is 0^+. It is also true that -Q7QLIMNO[0] = -K and -Q7QLIMNO[1] = 0^-.

4. The DOUBLE statement.

Consider the example, DOUBLE (A, B, P, Q, +, ←, H, L). This statement performs a (hardware) double-precision addition. The variables A, B are taken as the first operand, P, Q as the second, and the result is placed in H, L. The most significant word of the first operand is A, and the mantissa of B is considered as an extension of the mantissa of A. The rest of B is ignored. The same is true of P, Q and of H, L. In placing the result in H, L, the non-mantissa part of L is set to zeros. The + can be replaced

by ⊠ or / to perform multiplication or division, similarly. When operating on normalized operands, the DOUBLE operators always produce either a normalized result or zero. An exponent overflow can be detected within a DOUBLE statement by means of the < fault statement > discussed before. If an underflow occurs, we have assumed that the following actions are taken:

(i) the B5500 sets an underflow interrupt.

(ii) the Algol-compiled code tests the interrupt, and if it is set, the result H, L is set to all zeros.

We could not find this assumption explicitly stated anywhere. However, experimentation both at the hardware and software levels, and discussion with Burroughs representatives tend to verify these assumptions.

5. The point (.) operator.

This is a combination of two features of Extended Algol, the DEFINE declaration and the partial word designator, and provides a means of accessing word subfields of contiguous bits. The point operator and a field name, applied to the numeric operand $A = (s_m, s_e, (e_1, e_2), (m_1,\ldots,m_{13}))$, yields the corresponding subfield. That is,

$A.Q7QEXPONENT = e_1e_2$

$A.Q7QMANTISSA = m_1m_2,\ldots,m_{13}$

$A.Q7QMSIGN = s_m$

$A.Q7QXSIGN = s_e$

A.Q7QM1213 = $m_{12}m_{13}$

A.Q7QM1 = m_1

A.Q7QSEXPONENT = $s_e e_1 e_2$

The DEFINE feature allows the use of field names rather than requiring numeric notation. When arithmetic is performed using any of these subfields as an operand, the bits covered by the field are treated as an unsigned (non-negative) integer.

Global Declarations and Initialization.

With the one exception of NORMALIZE, all of the procedures of this package have no parameters and, therefore, operate only on global variables. (This technique is employed to minimize execution time.) All of these global variables, with the one exception of INTERFLOW, are prefixed with Q7Q in order to minimize the possibility of a nomenclature conflict when these procedures are inserted into a program. However, for the remainder of this paper, all Q7Q prefixes are dropped except in the actual program listing in the Appendix. The following declarations, or equivalent ones, are necessary:

```
BOOLEAN       LEFT, SUBFLAG

INTEGER       EXP, EXPA, EXPB, I, INTERFLOW

REAL          H, HS, L, LA, LB, LS, OPL1, OPL2, OPR1, OPR2,
              RA, RB, RNDR, S, T, X, Y

REAL ARRAY    LIMNO [0 : 1]
```

```
DEFINE      EXPONENT = [3 : 6] #, MANTISSA = [9 : 39] #,
            MSIGN = [1 : 1] #, M1 = [9 : 3] #, M1213 =
            [42 : 6] #, SEXPONENT = [2 : 7] #, XSIGN =
            [2 : 1] #
```

In addition to the preceding declarations, the interval arithmetic package consists of five procedures, ADD, SUB, MLT, DIV, and NORMALIZE, and three initialization statements. (The procedure names are also prefixed with Q7Q, and only NORMALIZE has a formal parameter.) The three initialization statements are

```
SUBFLAG ← FALSE

INTERFLOW ← 0
```

and

```
FILL LIMNO [*] WITH OCT0777777777777777,OCT1771000000000000
```

and must be executed before the execution of any of the procedures. Recall that the FILL statement initializes LIMNO [0] to +K and LIMNO [1] to $+\epsilon = 0^{+}$.

General Strategy of Implementation.

ADD, SUB, MLT, and DIV operate on the intervals A and B and return the result in B. A is represented by the two global variables LA and RA containing the left and right endpoints of A, respectively. Similarly B is represented by LB and RB. For all four procedures, the form of the operation is therefore B ← A op B. If, and only if, a magnitude violation occurs during the execution of a procedure, the

global variable INTERFLOW (no Q7Q prefix) is set to the value of F given in Table 3. (If no violation occurs, INTERFLOW is left unaltered rather than set to zero, provided INTERFLOW > 0 upon entry. If INTERFLOW < 0 upon entry, it is immediately set to zero.) It will be noticed that a magnitude violation corresponds to the machine condition of exponent overflow. The problem of exponent underflow is taken into account directly by Eqs. 4. On the B5500 we know that if underflow occurs, the exact result y satisfies the relation $0^- < y < 0^+$ and the appropriate max or min can be found as required by Eqs. 4.

ADD, SUB, MLT, and DIV operate on the variables LA, RA, LB, and RB which are operands of the set N. If upon entry to any of the procedures, LA, RA, LB, or RB is not an operand of the set N, it is replaced by an operand in N. The following is done for each of the four operands not already in N:

(i) If the mantissa of the operand is all zeros, the operand is replaced by the operand Z.

(ii) The operand is normalized if possible--the mantissa is shifted left and the exponent reduced until $m_1 \neq 0$.

It is possible that the operand cannot be normalized. In this case the operands are replaced by the operands shown in Table 4.

Table 4. Replacements for unnormalizable operands. Z^+ and Z^- are elements of N such that $r(Z^-) = 0^-$ and $R(Z^+) = 0^+$. $r(Z) = 0$.

Operand	Sign of Operand	
	+	-
LA	Z	Z^-
RA	Z^+	Z
LB	Z	Z^-
RB	Z^+	Z

Thus henceforth in the text, we assume that LA, RA, LB, RB are elements of the set N.

Once having these single-precision operands in normalized form, the operations of Eqs. 4 are performed, in general, in double-precision. These double-precision results are then appropriately rounded back to single-precision according to Lemma 6 to satisfy the max and min operations of Eqs. 4. It is possible for the procedures to produce an unnormalized result. However, if this unnormalized result is operated on again by one of the procedures it will be normalized before such later operations begin by the algorithm stated previously. The results of the procedures are not normalized because the B5500 single-precision arithmetic operations do not require normalization.

The reader may ask justifiably why these procedures were written as they were. The Q7Q prefixes and the global procedure arguments certainly do not make the procedures appealing for direct use. The

answer is that the procedures were not designed for direct usage. They are to be part of an Interval Algol language which is currently being designed and implemented (by Good). The language starts with Algol and defines a new data type called interval. The initial implementation of this language is a translator from Interval Algol to Algol; interval arithmetic expressions are translated into calls on these procedures. The prefixes minimize nomenclature conflicts, and the global procedure arguments allow the generation of a relatively efficient translation. To make direct use of the interval arithmetic in these procedures more appealing pending an implementation of Interval Algol, parameterized procedures which call the Q7Q procedures have been prepared. For example, INTADD(C, A, B) computes the interval C, namely $C \leftarrow A + B$, by first setting LA, LB, RA and RB and then calling Q7QADD.

IV. THE PROOFS OF THE CODE

The General Strategy of Proof and Notation.

The remainder of this paper is concerned with proving the correctness of the four procedures ADD, SUB, MLT, and DIV. Some groundwork, including notation, toward this goal has already been laid in the preceding, but more is needed.

The actual Algol code appears in the Appendix. Words in all capital letters in the text refer to program identifiers. In the proofs, the Algol code is displayed in upper case type. Any three digit numbers after a line of code or in the text refer to sequence numbers in the code listing. We have suppressed irrelevant zeros, of course. We remind the reader that we have dropped the Q7Q prefix from all names.

In the following proofs and discussion the following notation is used on Algol variables. Suppose X is an Algol variable whose content represents the number $x \in M$. Then X^+ denotes a variable whose content represents x^+, and X^- denotes a variable whose content represents x^-. If Y is another Algol variable, then (X,Y) denotes a double-precision operand using X as the most significant word, and Y as the least. Also in the following, the symbol ϵ stands for 0^+ and $-\epsilon$ stands for 0^-. Finally r (X) denotes the real number represented by X, and similarly for r (X,Y).

The first task in proving the correctness of the procedures is to state precisely what it is we wish to prove. While this could be incorporated into the main proofs, it seems easier and clearer to state this separately. The stating of what we want to prove occurs at different levels. The way in which the proofs at the various levels are combined into the final proof is illustrated schematically in Figure 2.

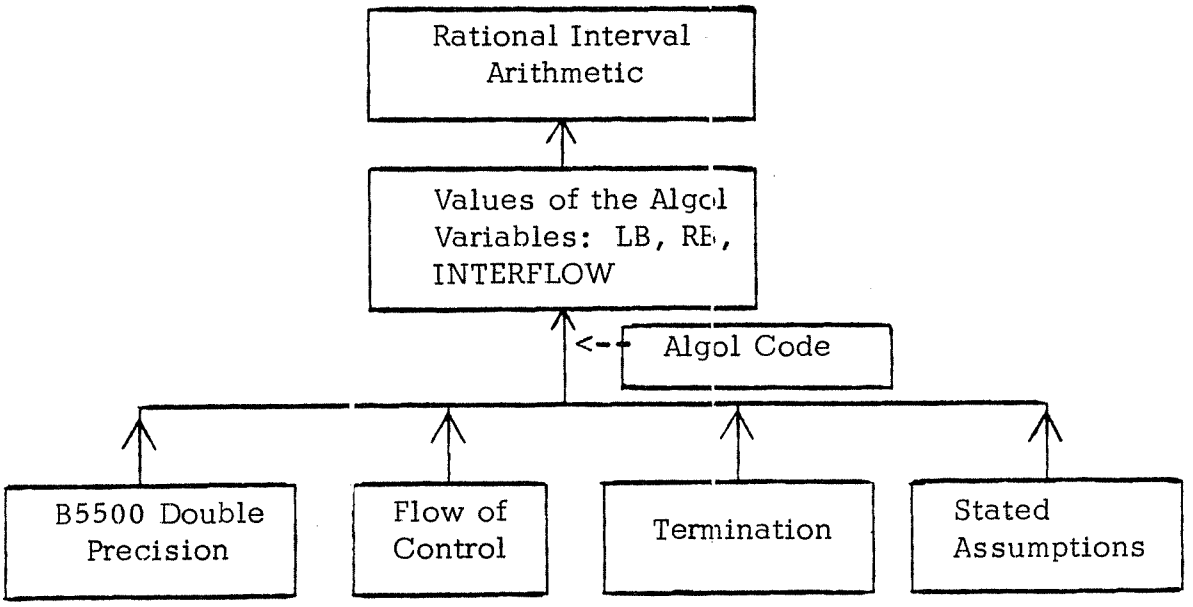

Figure 2. General Proof Strategy.

Thus, the lemmas and assumptions of the bottom row of Figure 2 are applied directly to the Algol code to show that the variables have certain values upon exit from a procedure. Another set of lemmas shows that these values are indeed the ones required to satisfy the definition of rational interval arithmetic.

More specifically, Lemmas 10, 11 and 12 are the connections between the Algol variables and rational interval arithmetic, i.e. Tables 5, 6 and 7 present what the results of the B5500 implementation of interval arithmetic should or ought to be. The proofs of these lemmas are based, then, on the definition and development of rational interval arithmetic and previous lemmas. We are not, at this point, stating anything about the results that the implementation yields.

Once Lemmas 10, 11 and 12 have stated this task precisely, the problem of the proof of the procedures is significantly reduced. But in no sense is the proof problem eliminated or even made trivial. We must still verify that the implementation, in all cases, produces the stated, desired results of Tables 5, 6 and 7.

Assumptions.

We list here in one place the assumptions we must make in order to complete the proofs. Nearly all of them have been empirically verified or copied from a manual or both, but they have not been proved as has the rest of the material. We have assumed:

1. DOUBLE normalizes all results or else gives 0. Overflow, but not underflow, is detected by the fault statement.
2. Unary minus and assignment operators are exact, namely:
   (i) $-(e,m) = (e,m)$ except for the sign bit, of course.
   (ii) $(e,m) \leftarrow (\overline{e},\overline{m})$ implies $(e,m) = (\overline{e},\overline{m})$.

3. Only arithmetic operations and specifically no relational operators can cause overflow or underflow. Also the relational operators give correct results for any operand, normalized or not.

4. The Algol compiler is correct according to the definition of Algol. We further assume that it or the hardware handles all problems of +0 and -0. For example, if A is a simple variable, A = 0 is true if and only if all digits of the mantissa of A are zero, and A $>$ 0 if and only if A represents a positive quantity.

5. All results which DOUBLE cannot normalize because its exponent would become $< -77_8$ are set to 0 (underflow).

6. Double precision real divide does no rounding.

7. Double precision multiply involving operands of the form (e,m,0) and $(\overline{e},\overline{m},0)$ produces exact results (cf. Lemma 8).

8. Definition of SIGN function:

$$\text{SIGN (A E)} = \begin{cases} +1 & \text{if } AE > 0 \\ 0 & \text{if } AE = 0 \\ -1 & \text{if } AE < 0 \end{cases}$$

where AE is an arithmetic expression.

9. DOUBLE always terminates.

10. SIGN always terminates.

Required Values of the Variables LB and RB.

The next three lemmas show what values LB and RB must have at the end of each procedure in order to satisfy Eqs. 4 and Table 3.

The results of these lemmas are given in Tables 5, 6 and 7. Each of Tables 5 and 6 is really two tables combined--one for computing left endpoints LB and one for computing right endpoints RB. If we let, in all its occurrences, the letter D (standing for Direction) be L and ignore the RB column, we have a table for computing LB. Similarly, if we let D be R and ignore the LB column, we compute RB. The inequalities in Table 5, for example $DA < S < DA^+$, reflect the dual nature of the table. If D = L, this inequality says that DA (= LA) is a lower bound and since $S < DA^+$ (= LA^+), LA is the best machine representable lower bound. IF D = R, the roles of DA and DA^+ are reversed.

Lemma 10. Table 5 gives the values of LB and RB for the sum interval in the case when the mantissas of the summands are disjoint. Recall that we wish to compute [LB, RB] ← [LA, RA] + [LB, RB] where the original [LA, RA] and [LB, RB] correspond to [v, w] and [x, y], respectively, of Eq. 4.1.

Table 5. Values of LB and RB for Sum Interval, Disjoint Mantissas.

	Original Endpoints		DA Bigger in Magnitude			DB Bigger in Magnitude		
	DA	DB	Bounds on Sum S	LB	RB	Bounds on Sum S	LB	RB
1	> 0	> 0	$DA < S < DA^+$	LA	RA^+	$DB < S < DB^+$	LB	RB^+
2	> 0	$= 0$	$S = DA$	LA	RA	$S = DA$	LA	RA
3	> 0	< 0	$DA^- < S < DA$	LA^-	RA	$DB < S < DB^+$	LB	RB^+
4	$= 0$	> 0	$S = DB$	LB	RB	$S = DB$	LB	RB
5	$= 0$	$= 0$	$S = 0$	0	0	$S = 0$	0	0
6	$= 0$	< 0	$S = DB$	LB	RB	$S = DB$	LB	RB
7	< 0	> 0	$DA < S < DA^+$	LA	RA^+	$DB^- < S < DB$	LB^-	RB
8	< 0	$= 0$	$S = DA$	LA	RA	$S = DA$	LA	RA
9	< 0	< 0	$DA^- < S < DA$	LA^-	RA	$DB^- < S < DB$	LB^-	RB

Proof. Lines 2, 4, 6, 8. Since one operand is 0, the result is the non-zero operand.

Line 5. Both operands are 0 so the result is either one, namely 0. Let Q stand for the quantity bigger in magnitude, and consider the effect of the quantity smaller in magnitude. The idea is similar to the proof of Lemma 6 and so is given without details.

Line 1. S exceeds Q but is less than Q^+.

Lines 3, 7. If $Q > 0$, S exceeds Q^- but is less than Q. If $Q < 0$, S exceeds Q but is less than Q^+.

Line 9. S exceeds Q^- but is less than Q.

Lemma 11. If there is exponent underflow in addition, Table 6 gives the values of LB and RB for the sum interval.

Table 6. Values of LB and RB for Sum Interval, Exponent Underflow.

	Original Endpoints		Comparison of DB, DA if of opposite signs	Bounds on Sum S	LB	RB
	DA	DB				
1	> 0	> 0	-	$0 < S < \epsilon$	0	ϵ
2	> 0	$= 0$	-	$0 < S < \epsilon$	0	ϵ
3	> 0	< 0	$-DB > DA$	$-\epsilon < S < 0$	$-\epsilon$	0
4	> 0	< 0	$-DB < DA$	$0 < S < \epsilon$	0	ϵ
5	$= 0$	> 0	-	$0 < S < \epsilon$	0	ϵ
6	$= 0$	< 0	-	$-\epsilon < S < 0$	$-\epsilon$	0
7	< 0	> 0	$DB > -DA$	$0 < S < \epsilon$	0	ϵ
8	< 0	> 0	$DB < -DA$	$-\epsilon < S < 0$	$-\epsilon$	0
9	< 0	$= 0$	-	$-\epsilon < S < 0$	$-\epsilon$	0
10	< 0	< 0	-	$-\epsilon < S < 0$	$-\epsilon$	0

Proof. Note first that $-DB = DA$ (includes $DA = DB = 0$) would imply an exact result of 0, not underflow.

Lines 1, 2, 5, 6, 9, 10 are clear since the summands are not of opposite signs. In fact, with normalized operands, underflow is impossible in these cases.

Lines 3, 4, 7, 8 follow by considering the sign of the real arithmetic sum under the constraint of the third column. For example, in line 3 $-DB > DA$ says DB is more negative than DA is positive and hence the real sum is negative.

Lemma 12. If the double-precision result (H, L) is exact or if, in division, any inexactness may be ignored, Table 7 gives the values of LB and RB for the result interval.

Table 7. Values of LB and RB for Exact Double-Precision Result.

	L	H	Should result be exact zero?	Bounds on result R	LB	RB
1	0	0	No	Underflow, i.e.. $-\epsilon < R < \epsilon$	$-\epsilon$, 0 based on each operation	0, ϵ
2	0	0	Yes	$R = 0$	0	0
3	0	$\neq 0$	-	$R = H$	H	H
4	$\neq 0$	> 0	-	$H < R < H^+$	H	H^+
5	$\neq 0$	< 0	-	$H^- < R < H$	H^-	H
6	$\neq 0$	$= 0$	-	Conditions are impossible		

Proof. Line 1. These are the conditions for underflow (cf. assumption 5).

Line 2. The YES entry says the result is 0.

Line 3. $r(H,0) = r(H)$ so the result is contained in a single-precision word.

Line 4. $H > 0$ and $L \neq 0$ implies the result exceeds H but is less than H^+.

Line 5. $H < 0$ and $L \neq 0$ implies the result is less than H but exceeds H^-.

Line 6. Impossible else the result is not normalized without any underflow.

The entries in the "LB" and "RB" columns, namely the values of LB and RB as best bounds, follow immediately from the "Bounds on result R" column. This completes the proof.

Flow of Control through IF Nests.

The next lemma enables us to verify flow of control through certain IF - THEN - ELSE constructions. The lemma is needed because of such formidable appearing conditional statements as the one at 222-269. We feel compelled to offer some sort of proof that, in such a conditional statement, the flow of control is as we claim. Lemma 13, below, is our response to that compulsion.

The recursive nature of Algol constructions makes the conditional statements in the code very powerful and elegant. But the recursion also makes it somewhat difficult to state the lemma succinctly. Thus we shall not attempt to state the most general result possible. In fact, there will be a few conditional statements in the code to which the lemma will not apply directly. But additional arguments in each such case can give the desired conclusion.

The aim of the lemma is to state a sufficient set of conditions which guarantee that upon the execution of certain statements, control immediately leaves the conditional statement. For this purpose we need to define a suitable class of statements which appear within or as part of a conditional statement. Special problems are posed by compound statements and the constituent statements of a compound statement.

For a conditional statement C, we define the set $\mathfrak{R}_C$ of statements. The statement $S \in \mathfrak{R}_C$ if S is part of the conditional C in the sense

that S appears immediately after a THEN or an ELSE. That is, S is a statement allowed by the Extended Algol syntax in the two underlined places:

< if clause > < statement >
where < if clause > ::= IF < Boolean expression > THEN

or

< if statement > ELSE < statement >

S may be the dummy statement between two adjacent ELSE's. If, however, S is a compound statement, no constituent statement of S may belong to $\mathfrak{R}_c$. A single statement made compound solely for the purpose of making a COMMENT is not considered compound for the present purpose. See, e.g. lines 250-260.

Questions of the definition of $\mathfrak{R}_c$ may well arise. They should be resolved by noting the conditional statements of the procedures to which the lemma is (to be) applied:

NORMALIZE	185-192	
ADD	222-269	including Corollary 13.1 applied to 245-269
	282-329	including Corollary 13.1 applied to 305-329
MLT	370-431	modifying 403
	435-449	
	458-472	
DIV	600-655	deleting 625-629, 631-632 and 655
	658-672	
	682-697	

Lemma 13. Let C be a single conditional statement with the same number of IF's, THEN's, and ELSE's. As a restriction on $\mathfrak{R}_C$, assume that if $S \in \mathfrak{R}_C$, then S is not a GO TO statement and, further assume that if S is a compound statement, S does not contain a constituent conditional statement. Then, after the execution of each $S \in \mathfrak{R}_C$ is completed, control pases directly to the first statement T following C with no intervening statements being executed.

Proof. The lemma might be considered obvious from the recursive definition of conditional statements and its semantics. Nevertheless, we present the following proof by induction on the number L of IF's in C. Assume first there are no conditional arithmetic or Boolean expressions in C. If L = 1 (e.g. C might be IF A = 0 THEN B ← 1 ELSE B ← 2), $\mathfrak{R}_C$ contains two elements and the lemma follows directly from the definition of the conditional statement. Assume the lemma holds for $L \geq 1$, and consider C with L + 1 IF's.

Find within C a conditional statement U such that U contains precisely one each of IF, THEN and ELSE, i.e. no conditional statements appear within U. Because the IF's, THEN's and ELSE's are nested, U must exist. If U does not exist, i.e. if each conditional statement contains more than one IF, THEN and ELSE, the nesting is non-terminating or infinite. Then there is an infinite number of IF's which is clearly absurd.

If desired, U may be found constructively. Since the IF's, THEN's and ELSE's are nested or grouped similarly to parentheses, we may discover the nesting as follows: Count each THEN as +1, each ELSE as -1 and compute a running sum from left to right through C. Nesting implies the sum is always non-negative. Equal numbers of THEN's and ELSE's imply the sum at the end of C is zero. A suitable U is a conditional statement whose THEN has the maximum running sum value. There may be several such U's, but that matters not. By assumption, U is not part of a compound statement.

From the definition of the conditional statement, control clearly passes in one of two ways to the end of U. Modify C, obtaining C', by replacing U with an unconditional statement V (non-GO TO, non-compound, of course) which contains no IF's, THEN's or ELSE's. C' contains precisely L IF's. By the induction assumption the lemma holds for all $S \in \mathcal{R}_{C'}$. But it is easy to see that the lemma now also holds for all $S \in \mathcal{R}_{C}$ since $\mathcal{R}_{C'} = \mathcal{R}_{C} \cup \{V\} - \mathcal{R}_{u}$. That is, control's unconditionally getting to the end of V, and hence to T, is equivalent to control's getting to the end of U in either of two ways and hence to T.

If there are conditional arithmetic or Boolean expressions in C, we may by a similar, but unstated lemma, replace the conditional expressions by a single (unconditional) value and apply the above. No circular reasoning is involved since no conditional

statements may occur within a conditional expression. This completes the proof.

If the reader is unhappy with our abrupt dismissal of conditional expressions, the following argument will suffice for our later proofs: Each of the conditional expressions in this code has only one IF-THEN-ELSE and they are matched as the hyphens show. Therefore, the reader may simply replace the conditional expression by a constant analogously to the replacement of the U by V above. The lemma then applies under the assumption of no conditional expressions. In effect, this argument proves the unstated lemma for L = 1 but in a larger context than the L = 1 case in Lemma 13.

The running sums used in the above proof may also be used to verify constructively the flow of control. When control reaches the end of the statement after a THEN whose sum is A, or when control reaches the end of the statement after an ELSE whose sum is A $>$ 0, find the first succeeding ELSE whose sum is A -1. Control passes to the end of the statement after the designated ELSE. Apply this algorithm repeatedly. It must terminate since at each step we are finding an ELSE whose sum is strictly less than the preceding value and zero is a lower bound on the sum. For an ELSE whose sum is 0, control passes directly to T as required. Note that "the first succeeding ELSE" exists both after a THEN and after an ELSE whose sum is positive since otherwise the running sum is positive at the end.

Corollary 13.1. If a compound statement $S \in \mathfrak{R}_c$ is of the form

BEGIN

DOUBLE (. . .);

< conditional statement > ;

END;

then the conclusion of Lemma 13 also holds for the constituent conditional statement of S. That is, control also goes directly to T.

Proof. Apply Lemma 13 to the constituent conditional statement of the compound statement, and note that it has the same T as the main conditional statement of which S is a part.

The Variables X, Y, S, T, EXPA, EXPB, RNDR, LEFT, SUBFLAG.

Lemma 14. Table 8 gives all the locations in the code where selected variables, or their fields, are set or changed. The use of the point (.) operator denotes that only the named field in the variable is changed. S is an abbreviation for SEXPONENT.

Table 8. All Settings to Selected Variables.

Procedure	X	Y	S	T	EXPA	EXPB	RNDR	LEFT	SUBFLAG†
ADD	206	206	-	-	206	206	206	208	-
					217.MSIGN	218.MSIGN	229.S	211	
					219.M1213	220.M1213	238.S		
					277.MSIGN	278.MSIGN	265.S		
					279.M1213	280.M1213	286.S		
							298.S		
							325.S		
SUB	348	-	-	-	-	-	-	-	347
									352
MLT	*492	-	362	362	-	-	362	364	-
	*512	513					446.S	367	
	*524	-					469.S		
	562	563					530.S		
							579.S		
DIV	-	-	592	592	-	-	592	594	-
							669.S	597	
							694.S		

* If 492, then neither 512 nor 524.

† By 781, SUBFLAG is initially FALSE.

Proof. Examination of the code. Our examination was both manual and with aid of the compiler by undeclaring the variables and noting the error diagnostics. This completes the proof.

We wish to draw several inferences from Table 8:

1a. X and Y are used only in ADD, and S and T only in MLT and DIV to hold LA and RA, respectively.

b. X is used only in SUB for swapping LB and RB.

c. X is used (492) in MLT to save LB in case of overflow trap to ERR7F, or (524) to save LB if there is no overflow trap to ERR7F but never both. LB is needed only for the DOUBLE multiply at 535.

d. X and Y are used as part of the double-precision compare (512-513 and 562-563).

e. In all cases the need for the previous X and Y is gone by the time the new X or Y is set.

2. EXPA and EXPB are set to 0 at 206. Only the MSIGN and M1213 fields are changed at 217-220. Thus only the changes at 277-280 are needed to have EXPA and EXPB serve as integers again.

3. A similar argument to 2 applies to RNDR. It is set to +1 at the start of ADD, MLT and DIV and then only its SEXPONENT field is changed. Hence, when used, it is always the appropriate rounding factor.

4. LEFT is set only at the start of ADD, MLT and DIV as shown.

5. SUBFLAG is set only in SUB.

The Values of INTERFLOW.

Lemma 15. INTERFLOW is set only as follows:

Table 9. All Settings to INTERFLOW *.

Procedure	Start	Overflow traps	Finish
ADD	207	273, 335	340, 341
SUB	-	-	-
MLT	363	454, 478, 494, 544	584
DIV	593	626, 678, 704	710

* By 781, INTERFLOW is initially zero.

Proof. As in Lemma 14, the proof is by examination of the code. This completes the proof.

We now explain the mechanism of setting INTERFLOW. We shall use Table 9. Between calls to the four arithmetic procedures, a non-negative value of INTERFLOW indicates the last overflow error condition in a single arithmetic operation as follows: (cf. Table 3.)

Table 10. Meaning of INTERFLOW Values.

Interflow value	Overflow error condition
0	No error
1	Error left in ADD
2	Error right in ADD
3	Error left and right in ADD
4	Error left in SUB
5	Error right in SUB
6	Error left and right in SUB
7	Error left in MLT
8	Error right in MLT
9	Error left and right in MLT
10	Error left in DIV
11	Error right in DIV
12	Error left and right in DIV
13	Attempt to divide by an interval containing 0

If INTERFLOW is set positive, upon completing a procedure, this implies that the resulting endpoint ($\pm K$) is no longer a bound of the result. Note that INTERFLOW is initialized to zero at 781. If $\pm K$ is still a bound, INTERFLOW is not set. Recall that $-K$ is an upper bound for endpoints $< -K$, and $+K$ is a lower bound for endpoints $> +K$; in fact, both are the best bound that is machine representable. That INTERFLOW is set according to this rule follows by examining each of the overflow traps and observing whether the trap involves a left or right endpoint computation.

Within an interval arithmetic operation, an overflow condition sets INTERFLOW to the negative of the value in Table 10. This enables the operation to know whether a non-zero INTERFLOW value is caused by the current operation (INTERFLOW < 0) or by a previous operation (INTERFLOW > 0). This is necessary if a right overflow of, say ADD, is to tell if an indication of left overflow of ADD is caused by the current operation or not.

Note that INTERFLOW is guaranteed non-negative at the start of each operation since the statement

IF INTERFLOW < 0 THEN INTERFLOW $\leftarrow 0$

is executed by 207 in ADD, by 363 in MLT and by 593 in DIV. (All of INTERFLOW in SUB is handled by ADD since the actual subtraction is done in ADD. See below.)

An examination of each of the overflow traps in Table 9 shows that the proper negative error number is set. Note particularly that a right overflow checks for a left overflow in the current operation and sets INTERFLOW accordingly. Once set negative within an operation, INTERFLOW can change in only two ways. It may be changed to indicate that both a left and right overflow error have occurred, and it will be changed at the end to make the negative number positive. To see the latter, note that in all cases, the statement

IF INTERFLOW $<$ 0 THEN INTERFLOW $\leftarrow$ - INTERFLOW

is executed by 341 in ADD, by 584 in MLT and by 710 in DIV.

The only other INTERFLOW detail is 340 in ADD:

IF INTERFLOW $<$ 0 AND SUBFLAG THEN INTERFLOW $\leftarrow$ INTERFLOW - 3

SUBFLAG is true if and only if ADD has been called by SUB since SUBFLAG is initialized to FALSE and set only as in Table 8. Hence, any ADD error must be converted to the corresponding SUB error by subtracting 3 (before 341 makes INTERFLOW positive). Conversely, if ADD is called by SUB and ADD generated no error, then SUB will report no error.

The Actual Rounding of Variables.

Lemma 5 gives formulas for computing the rounded quantities X^+ and X^-. In the actual code open subroutines are used to accomplish all roundings according to these formulas. But for the purpose of

proof it is convenient to define two closed subroutines (real procedures), ROUNDUP (A) = A^+ and ROUNDDOWN (A) = A^-, where in both cases the parameter A is assumed normalized and non-zero. A procedure body for ROUNDUP (A) would be

```
BEGIN RNDR.SEXPONENT ← A.SEXPONENT;
ROUNDUP ← A + RNDR END
```

and a procedure body for ROUNDDOWN (A) would be

```
BEGIN RNDR.SEXPONENT ← A.SEXPONENT;
ROUNDDOWN ← A - RNDR END
```

A check of the code shows that all roundings are accomplished as if by one of the two above procedure bodies.

Lemma 16. The two procedure bodies above accomplish ROUNDUP (A) = A^+ and ROUNDDOWN (A) = A^-, respectively, provided the parameter A is normalized and non-zero. ROUNDUP ($-\epsilon$) is unnormalized and ROUNDUP (+K) will cause exponent overflow. Similarly, ROUNDDOWN (ϵ) is unnormalized and ROUNDDOWN (-K) will cause exponent overflow. The number represented by all other results is an element of M.

Proof. The statements in the last three sentences of the lemma follow directly from the proof of Lemma 5. To prove that ROUNDUP (A) = A^+ and ROUNDDOWN (A) = A^- provided A is normalized and non-zero, suppose A is of the form (e,m). By Lemma 5 and the second line of each procedure body, we need only show that RNDR is of the form (e,1).

When initially set at the start of each ADD, MLT and DIV, the variable RNDR = (0,1). By Table 8 all subsequent settings of RNDR occur only as in the first line of either procedure body, namely only its SEXPONENT field is altered. In fact, the first line gives RNDR each time the required SEXPONENT e from A = (e,m).

Proof of NORMALIZE.

In the proofs of ADD, SUB, MLT and DIV, it is essential to know that all endpoints are either normalized or zero prior to any computation. In order to guarantee this, the procedure NORMALIZE (A) is used, where A is the endpoint to be normalized and also the result of normalization. We shall prove below that NORMALIZE obeys the defintion: (Cf. the comment in the code for NORMALIZE, lines 163-174, and also the section on General Strategy of Implementation including Table 4.)

IF A = 0, the result A is set to all zeros.

IF $A \neq 0$, normalize A, if possible. If this is not possible, use Table 11.

Table 11. Result of NORMALIZE for non-zero, unnormalizable arguments A.

	LEFT	A	Result
1	True	> 0	0
2	True	< 0	$-\epsilon$
3	False	> 0	ϵ
4	False	< 0	0

It is necessary first to show that Table 11 is what ought to be done if A is unnormalizable. NORMALIZE assumes A is a left endpoint if and only if LEFT = TRUE (and assumes a right endpoint otherwise). Under this assumption, Table 11 follows immediately from Table 4.

We can now prove the correctness of the procedure NORMALIZE. Both the key factor and key problem in NORMALIZE is the FOR statement--the only loop in the entire code. In response, we use basically a proof method due to R. W. Floyd (4). We have stated predicates or propositions in the manner Floyd suggested. As he showed was necessary and sufficient to obtain a proof, our proof consists of verifying the predicates as follows: Each statement S (or better, each box of an equivalent flowchart) is both preceded by one or more antecedent predicates, P_i, i = 1, 2,...n, and succeeded by one or more consequent predicates, Q_j, j = 1, 2,...m. (Join points of control cause $n \geq 2$, and branch points cause $m \geq 2$.) Of course, at a specific point in the actual execution of the code, the unique antecedent and consequent predicate is determined by the flow of control. But we must consider all combinations for a proof. Therefore, along each path of control, we assume the P_i predicate is true before the statement and we show that each time after the statement S is executed the Q_j predicate is true.

In other words, we prove, for each statement S, m ⊠ n separate theorems of the form: P_i and S implies Q_j for all i and j. Fortunately, m and n are here at most 2. Thus, assuming legal input data, if the predicates both have been chosen correctly and successfully verified,

the desired result will hold at the (assumed) termination of the code. In short, each predicate is true each time control passes it since we have shown there is no first false predicate. Proof of termination is handled separately.

Lemma 17. NORMALIZE (A) implements the previous definition.

Proof. Some notation will simplify the presentation. Let A^O be the initial value of A, and let E^O be the integer represented by the value of SEXPONENT of A^O. Hence $r(A^O) = \text{sign}(A)\ r(A.\text{MANTISSA})\ 8^{E^O}$. The superscript zero is to suggest time zero, namely before NORMALIZE starts to compute. Thus the use of A will follow the usual convention that a name denotes its current value. We shall show at various points that we have not altered $r(A^O)$ even though A itself has been altered. Indeed, the purpose of NORMALIZE is to alter A so that it is normalized, if possible, but has the same value.

We number the lines of code consecutively starting at 0 in order to be able to refer to them. (We do not use the sequence numbers as shown in the Appendix.) The predicates or propositions are numbered as m, where m is the line of code after which the predicate appears, and the predicates are further identified by ".n". All predicates with the same m should be considered connected by "Boolean and" to form the single predicate m. They are so subdivided only for ease of reference. The ".n" do not indicate separate antecedent or consequent predicates. The symbol " <== " should be read "since" or "is implied by",

i.e. the passive of "implies" or " $\Longrightarrow$ ". Where a predicate has more than one line of code as its predecessor, the number or numbers over the "$\Longleftarrow$" indicates which of the possible paths of control is assumed.

Note that A.MSIGN is altered only at lines 14 and 21. In both cases control passes immediately to the end of NORMALIZE. We know, therefore, that sign (A) = sign (A^{O}) essentially throughout.

The proof starts on the next page. It appears between statements of the Algol Code.

0 BEGIN

0.1 $| E^{O} | \leq 63$ $\Longleftarrow$ the magnitude of the exponent of A^{O} is represented by two octal digits (2, p. 2-7). Hence $| E^{O} | \leq 77_{8} = 63_{10}$. Note that E^{O} is an integer.

1. IF $A \neq 0$

1.1 r(A.MANTISSA) $\neq 0$ $\Longleftarrow$ 1 and the definition of $\neq$ (2, p. 6-13).

2 THEN BEGIN

3 I $\leftarrow$ EXP $\leftarrow$ 0;

4 EXP.MSIGN $\leftarrow$ A.XSIGN;

5 EXP.M1213 $\leftarrow$ A.EXPONENT;

5.1 r (EXP) = E^{O} $\Longleftarrow$ 3, 4 and 5. I.e. EXP denotes a B5500 numeric operand, $(s_m, s_e, (e_1, e_2), (m_1, m_2, \ldots m_{12}, m_{13}))$. Now

$3 \Longrightarrow (+, +, (0,0), (0,0,\ldots,0,0))$,

$4 \Longrightarrow$ (sign E^{O}, +, (0,0), (0,0,...,0,0)),

$5 \Longrightarrow$ (sign E^{O}, +, (0,0), $(0,0,\ldots,e_1,e_2)$) where e_1 is the most significant digit of E^{O} and e_2 the least.

5.2 r(A.MANTISSA) $\neq 0$ $\Longleftarrow$ 1.1 still holds.

5.3 sign (A) r (A.MANTISSA) $8^{r(EXP)} = r(A^{O})$ $\Longleftarrow$ 5.1 and definition of A^{O}.

6 FOR I $\leftarrow$ I + 1 WHILE A.M1 = 0 AND EXP $>$ -63 DO

6.1 r (A.M1) = 0 $\Longleftarrow$ 6.

6.2 $r(EXP) > -63 \leq 6.$

6.3 sign (A) $r(A.MANTISSA)\ 8^{r(EXP)} = r(A^{O}) \leq^{5} 5.3 \leq^{9} 9.2.$

6.4 $r(A.MANTISSA) \neq 0 \leq^{5} 5.2 \leq^{9} 9.3.$

```
7       BEGIN
8       EXP ← EXP - 1;
9       A.MANTISSA ← A.MANTISSA ⊠ 8;
```

9.1 $r(EXP) \geq -63 \leq 6.2$, 8 and EXP is an integer.

9.2 sign (A) $r(A.MANTISSA)\ 8^{r(EXP)} = r(A^{O}) \leq 6.3$, 8 and 9. I.e. the relationship is true at 6.3. In view of 8 and 9 we must make, in 6.3, the substitutions EXP - 1 for EXP and A.MANTISSA ⊠ 8 for A.MANTISSA. Thus,

sign (A) $r(A.MANTISSA \boxtimes 8)\ 8^{r(EXP-1)} =$

sign (A) $r(A.MANTISSA)\, r(8)\ 8^{r(EXP) + r(-1)} =$

sign (A) $r(A.MANTISSA)\ 8 \cdot 8^{r(EXP)}\ 8^{-1} =$

sign (A) $r(A.MANTISSA)\ 8^{r(EXP)} = r(A^{O})$.

9.3 $r(A.MANTISSA) \neq 0 \leq$ 6.4 and 6.1. The latter insures that there is no overflow of the A.MANTISSA field by the multiply at 9. In particular, r(A.MANTISSA) remains non-zero.

```
10      END;
```

10.1 A.M1 $\neq$ 0 or r(EXP) = -63 $<\underline{\underline{5,6}}$ 6, 5.1 and 0.1.

$<\underline{\underline{5,9,6}}$ 6 and 9.1.

10.2 sign (A) r(A.MANTISSA) $8^{r(EXP)}$ = r(A^{O}) $<\underline{\underline{5,6}}$ 5.3.

$<\underline{\underline{5,9,6}}$ 9.2.

11 IF A.M1 = 0

11.1 r(EXP) = -63 $<==$ 10.1 and 11.

11.2 A cannot be normalized $<==$ 11 and 11.1.

12 THEN IF LEFT

13 THEN A $\leftarrow$ IF A $>$ 0

13.1 Line 1 of Table 11. Result is 0.

14 THEN 0

15 ELSE

15.1 A $\leq$ 0. But by 1, A $\neq$ 0. Result is -ϵ from line 2.

16 -ϵ

17 ELSE

17.1 Not LEFT, i.e. LEFT is FALSE.

18 A $\leftarrow$ IF A $>$ 0

18.1 Line 3. Result is ϵ.

19 THEN ϵ

20 ELSE

20.1 A $\leq$ 0. But by 1, A $\neq$ 0. Result is 0 from line 4.

21 0

22 ELSE BEGIN

22.1 A.M1 $\neq 0$.

22.2 $-63 \leq$ r(EXP) $\leq 63 <$==== 0.1, 5.1, the fact that after 5, EXP is changed only at 8, and then 9.1 guarantees it.

23 A.XSIGN ← EXP.MSIGN;

24 A.EXPONENT ← EXP.M1213;

24.1 r (A) = r (A^{O}) <==== 10.2, 23, 24 and 22.2. The 22.2 predicate guarantees that EXP.M1213 is all of EXP that matters. In other words, m_1 through m_{11} are all zero.

24.2 A is normalized <==== 22.1.

25 END

26 END

27 ELSE

27.1 A = 0, i.e. r(A.MANTISSA) = 0 <==== definition of $\neq$ and ELSE.

27.2 r (A^{O}) = 0 <==== 27.1.

28 A ← 0;

28.1 A is set to all zeros <==== definition of "← 0."

29 END NORMALIZE;

We summarize the situation when control reaches line 29 (END NORMALIZE;): If the initial A is zero, A is set to all zeros by lines 27-28. If A $\neq$ 0 but normalizable, 24.1 and 24.2 guarantee a normalized result of equal value. If A $\neq$ 0 and unnormalizable, Table 11 is implemented in lines 11-21.

We must still prove that NORMALIZE terminates. First observe that Lemma 13 does not apply directly to NORMALIZE because the conditional statement 11-25 violates the restriction on $\mathcal{R}_C$. However, Lemma 13 does apply to 11-25 considered as a separate conditional statement. Therefore, if control ever reaches 11, it reaches 26 having made precisely one setting to A.

Now NORMALIZE is just one conditional statement. If A = 0, control pases from 1 to 27 to 28 to the end. If $A \neq 0$, control passes from 1 to 2 through 5, 6 and (to be shown below) to 11. From there, by the above use of Lemma 13, control reaches 26 and hence to the end.

It remains to show that control passes from 6 to 11, i.e. statements 6-10 cannot be executed endlessly. By 5.2, there exists at least one non-zero octal digit in A.MANTISSA. Because 9 is the only place that changes A.MANTISSA within 6-10 (left shift one octal place), 9 could be executed at most 12 times before the non-zero digit would appear as A.M1. Hence A.M1 becomes $\neq 0$ so control passes from 6 to 11.

This completes the proof of NORMALIZE.

Operations Common to ADD, MLT, and DIV.

The four endpoints are normalized at the start of each procedure (except SUB) and before any computation starts. If an endpoint C is zero or is unnormalized, namely C.M1 = 0, NORMALIZE (C) is called.

(208-213 in ADD, 364-369 in MLT and 594-599 in DIV.) LEFT is set correctly in each case. If C = 0, NORMALIZE returns 0. Otherwise NORMALIZE returns C normalized and equal in value, or else the appropriate quantity $-\epsilon$, 0 or ϵ.

With two exceptions the endpoints remain normalized (or normalizable). This follows since the only arithmetic performed on them is by the DOUBLE statement or by the round operations. We are assuming DOUBLE normalizes results, gives 0 directly or else underflows with value 0. Except for ROUNDUP ($-\epsilon$) and ROUNDDOWN (ϵ), all rounds remain normalized (or normalizable). However, except for taking the negative of an endpoint in SUB, any unnormalized endpoint is normalized before any interval computation is performed. This follows by the paragraph above.

Since LA and RA are not to be changed by any of the four operations, they are saved in two variables at the start of each procedure (except SUB where ADD does it). They are restored from the proper variables just prior to exit. The saving variables are never used for another purpose. For the most part this is probably unnecessary. However, we feel that if either LA or RA is unnormalized upon entry to these procedures, these unnormalized values should be restored.

we did not do this, and we even chose not to prove the right endpoints with the one word proof: "similarly." This reflects our attempt to be extra cautious and complete.

0 1 2 3 4 5 6 7 8 9

THEN IF LA $\neq$ 0

Lines 1, 2, 7, 8 of Table 5. LB is LA. ("LB is LA" means the new value of LB is the initial value of LA, and similarly for other such statements.)

THEN LB $\leftarrow$ LA

ELSE

LA = 0. Lines 4, 5. LB is unchanged.

ELSE

LB $<$ 0.

IF LA $\neq$ 0

Lines 3, 9. LB is $\overline{LA}$. LA $\neq$ 0 and, by 209, LA is normalized. Thus we may use ROUNDDOWN.*

THEN LB $\leftarrow$ ROUNDDOWN (LA) 229-230

Overflow to SETR.

ELSE 232

LA = 0. Line 6. LB is unchanged.

ELSE 233

EXPA - EXPB $\leq$ 12.

IF EXPB - EXPA $>$ 12

LA and LB still have disjoint mantissas. LB is larger in magnitude.

0 1 2 3 4 5 6 7 8 9

* LA $\neq \varepsilon$ since it is larger in magnitude and hence EXPA $\geq$ -51.

0 1 2 3 4 5 6 7 8 9

(H,L) ← (LA,0) + (LB,0). Lemma 7 guarantees exact double-precision sum except for underflow. Overflow to SETR. Hence Lemma 12 applies.

IF L = 0

THEN IF H = 0

THEN IF LA ≠ -LB 249

THEN BEGIN COMMENT EXPONENT UNDERFLOW;

(H,L) = (0,0) and LA ≠ -LB implies underflow. 251-259 implements Lemma 11 (Table 6) for left endpoints.

IF LA ≥ 0 251

THEN IF LB > 0

Lines 1, 5. LB is 0.

THEN LB ← 0

ELSE

LB ≤ 0.

LB ←IF -LB > LA

Lines 3, 6. LB is -ϵ.

0 1 2 3 4 5 6 7 8 9

0 1 2 3 4 5 6 7 8 9

END

ELSE

LA = -LB so sum is 0.

LB ← 0

ELSE

H ≠ 0. L = 0 implies H is exact sum.

LB ← H

ELSE

L ≠ 0.

IF H < 0

L ≠ 0 and H < 0 implies LB is H^- by Lemma 12. H ≠ 0 since H < 0. H normalized by operation of DOUBLE. Thus we can use ROUNDDOWN.

THEN LB ← ROUNDDOWN (H) 265-266

Overflow to SETR.

ELSE

H ≥ 0. But H = 0 is impossible. Since L ≠ 0, LB is H by Lemma 12.

LB ← H

END; 269

GO TO ADRIGHT; 270

0 1 2 3 4 5 6 7 8 9

0 1 2 3 4 5 6 7 8 9

Hence control never reached 266 in the first place.

c. Suppose LA and LB have opposite signs. H = -K and $L \neq 0$ implies $(H,L) < -K$. But with opposite signs on the single precision summands, the sum $(H,L) \geq \min (LA,LB) \geq -K$, a contradiction.

In all cases control passes to ADRIGHT for the computation of the right endpoint RB.

We now show that RB is computed correctly. ERDONE becomes the exponent overflow label. Since EXPA and EXPB were only set once in the left endpoint calculation--to hold the respective SEXPONENTs as integers (see Table 8)--they may be and are used for that same purpose for RA and RB at 277-280. Control is thus at 282. 282-329 is a single conditional statement which computes RB. Lemma 13 (including Corollary 13.1 applied 305-329) shows that control passes through 330 to DONE at 338, assuming no exponent overflow. Lemma 13 further shows that RB is set at most once. It is easy to see that RB is set at least once or else unchanged. RB is not altered in the remainder of ADD. 282-292 implements Lemma 10 (Table 5) with RA larger in magnitude, and 293-303 implements Lemma 10 with RB larger in magnitude. 304-329 covers the case of non-disjoint mantissas.

IF EXPA - EXPB $>$ 12 282

0 1 2 3 4 5 6 7 8 9

0 1 2 3 4 5 6 7 8 9

Line 2. RB is RA.

THEN RB ← RA

ELSE

RB ≠ 0. Lines 1, 3. RB is RB^{+}. RB is normalized by 213. Thus we may use ROUNDUP. RB ≠ -ε. Similar to footnote p. 86.

RB ← ROUNDUP (RB) 298-299

Overflow to ERDONE.

ELSE

RA ≤ 0.

IF RB = 0

Lines 5, 8. RB is RA.

THEN RB ← RA

ELSE 303

RB ≠ 0. Lines 4, 6, 7, 9. RB is unchanged.

ELSE

|EXPA - EXPB| ≤ 12, i.e. non-disjoint mantissas.

BEGIN 305

DOUBLE (RA, 0, RB, 0, +, ←, H, L); 306

(H,L) ← (RA,0) + (RB,0). Lemma 7 guarantees

0 1 2 3 4 5 6 7 8 9

0 1 2 3 4 5 6 7 8 9

-RB ≥ RA, but

= is impossible

by 309. Lines

3, 6. RB is 0.

0

ELSE

RA < 0.

IF RB ≤ 0

Lines 9, 10. RB is 0.

THEN RB ← 0

ELSE

RB > 0.

RB ← IF RB > -RA

Line 7. RB is ∈.

THEN ∈

ELSE

RB < -RA. Line

8. LB is 0.

0 319

END

ELSE

0 1 2 3 4 5 6 7 8 9

0 1 2 3 4 5 6 7 8 9

ERDONE:

Control reaches here only if exponent overflow at 287, 299, 306 or 326. Assume overflow at 287. RB $>$ 0 and there is overflow only if RA = +K. Hence result should be +K (Table 3) and is. INTERFLOW is set since +K is not an upper bound. Overflow at 299 follows from that at 287 with the roles of RA and RB reversed.

Assume overflow at 306. Here overflow is possible only if RA ⊠ RB $>$ 0, i.e. they have the same non-zero signs. If both signs are positive, RB is set to +K and INTERFLOW is set. If both signs are negative, RB is set to -K, still a lower bound, the best that is machine representable.

Assume overflow at 326. L $\neq$ 0 and H $>$ 0 implies H must be +K before ROUNDUP operates to cause overflow. Overflow setting of RB should be +K. It is if RA $>$ 0 and RB $>$ 0. We must show that any other sign configuration of RA and RB is impossible.

a. RA $<$ 0 and RB $<$ 0 implies H $<$ 0, a contradiction to H $>$ 0.

b. If either RA or RB is 0, then the same argument holds as in left endpoint case.

0 1 2 3 4 5 6 7 8 9

Proof of SUB.

The proof of SUB is different from the other three. SUB uses the relation of Lemma 2, A - B = A + (-B). It therefore changes the B interval to -B (348-350) and then calls ADD (351). SUB must also set SUBFLAG true (347) before calling ADD and back to FALSE (352) after calling ADD in order that ADD can properly set INTERFLOW (cf. discussion of INTERFLOW). Thus the correctness of ADD implies that SUB correctly performs B ← A - B.

The only part of the above that is not immediately obvious is the changing of the B interval to -B (348-350). We wish to show that B = [u,v] becomes B = [-v,-u]. A simple trace, before and after each statement, of the variables X, LB and RB will suffice:

	X	LB	RB
	?	u	v
X ← LB;			
	u	u	v
LB ← -RB;			
	u	-v	v
RB ← -X;			
	u	-v	-u

This completes the proof of SUB.

Thus we may start examining the code (for all cases except 5) in more detail at 433. 435-449 is a single conditional. Applying Lemma 13, control passes through 450 to NOWRIGHT at 455 assuming no exponent overflow. In the code from 434-449 we show that LB is set correctly.

```
0     1     2     3     4

EXPOVR ← ERR7;                                                    433
DOUBLE (OPL1, 0, OPL2, 0, ⊠, ←, H, L);                            434
      (H,L) ← (OPL1,0) ⊠ (OPL2,0). Overflow to ERR7. Except for
      underflow, Lemma 8 guarantees exact double-precision
      product in (H,L). Hence Lemma 12 applies.
IF L = 0                                                          435
THEN IF H = 0
      THEN IF OPL1 ≠ 0 AND OPL2 ≠ 0
            THEN BEGIN COMMENT EXPONENT UNDERFLOW;
                  (H,L) = (0,0) and OPL1 ≠ 0 and OPL2 ≠ 0 implies
                  underflow.
                  LB ← IF SIGN (OPL1) = SIGN (OPL2)
                        THEN 0 ELSE -ϵ;
                        Neither OPL1 nor OPL2 is 0. Hence if the
                        signs agree, then before underflow, the
                        result was between 0 and ϵ, so 0 is the best

0     1     2     3     4
```

0 1 2 3 4

Applying Lemma 13 shows LB is set at most once in 435-449. It is easy to verify that LB is set at least once. LB is not altered in the remainder of MLT since control does not pass through CASE5.

GO TO NOWRIGHT; 450

ERR7:

Control reaches here only if exponent overflow at 434 or 447. Assume overflow at 434. Neither OPL1 nor OPL2 is 0, else no overflow. If like signs, LB should be +K (Table 3) and is so set whence control goes to NOWRIGHT. INTERFLOW is not set since +K is still a lower bound, the best that is machine representable. If unlike signs, LB should be and is set to -K and control goes to NOWRIGHT. In the latter case, INTERFLOW is set since -K is not a lower bound. Now assume overflow at 447. $H \neq 0$ implies neither OPL1 = 0 nor OPL2 = 0. $H < 0$ implies SIGN (OPL1) $\neq$ SIGN (OPL2). The rest follows as in 434 overflow.

In the code from 457-472, we show that RB is set correctly.

NOWRIGHT: 455

EXPOVR ← ERR89;

DOUBLE (OPR1, 0, OPR2, 0, ⊠, ←, H, L); 457

0 1 2 3 4

0 1 2 3 4

RB ← H

ELSE

L ≠ 0.

IF H > 0

L ≠ 0 and H > 0 implies (Lemma 12) RB is H^+. H ≠ 0 and H is normalized. Thus we can use ROUNDUP.

THEN RB ← ROUNDUP (H) 469-470

Overflow to ERR89.

ELSE

H ≤ 0. But H = 0 is impossible. Since L ≠ 0, RB is H.

RB ← H; 472

Applying Lemma 13 (and an easy verification) shows RB is set precisely once in 458-472. RB is not altered in the remainder of MLT since control does not pass through CASE5.

GO TO OVERFLOW;

ERR89:

Control reaches here only if exponent overflow at 457 or 470. As at ERR7, no argument of SIGN is 0. At 457 like signs implies RB ← +K. +K is not an upper bound so INTERFLOW is set. Unlike signs implies -K which is still an upper bound. At 470 H > 0 implies like signs. In all cases control goes to OVERFLOW.

0 1 2 3 4

0 1 2 3 4

$-\epsilon <$ LA ⊠ RB < 0, so the double-precision quantity $(-\epsilon, 0)$ is the best lower bound. L = 0 already.

END;

GO TO NEXTLMLT;

ERR7F:

Control reaches here only if exponent overflow at 484, 498 or 531. First LB is saved as X for right endpoint computation. In the first two cases, the product is negative while in the third $H < 0$. Thus, in all three cases LB should be and is set to -K. Since this is the smallest number we can represent, we have already computed the min, properly rounded. Thus, set INTERFLOW (-K is not a lower bound) and proceed directly to the right endpoint computation.

NEXTLMLT:

DOUBLE (RA, 0, LB, 0, ⊠, ←, HS, LS); 498

(HS,LS) = (RA,0) ⊠ (LB,0). Overflow to ERR7F.

IF HS = 0 AND LS = 0 AND RA ≠ 0

THEN BEGIN COMMENT EXPONENT UNDERFLOW;

HS ← $-\epsilon$

$-\epsilon <$ RA ⊠ LB < 0 so the double-precision quantity $(-\epsilon, 0)$ is the best lower bound. LS = 0 already.

END;

0 1 2 3 4

0 1 2 3 4

Table 8). Set L.MSIGN from H, LS.MSIGN from HS in order to have comparison include signs of the quantities. The SEXPONENT of both L and LS are already 0.

```
X ← L; Y ← LS;
L.MSIGN ←H.MSIGN; LS.MSIGN ← HS.MSIGN;
IF L ≤ LS
THEN
     (H,L) ≤ (HS,LS) so (H,L) is min. Restore L
     only since H is already set.
     L ← X
ELSE
     L > LS so (HS,LS) < (H,L). Move (LS, original
     LS = Y) to (H,L).
     BEGIN
     H ← HS;
     L ← Y
     END
END;                                                    523
```

Note that if the negation of 503 holds, i.e. H < HS, (H,L) is already the min. Thus, in all cases from 503-523, (H,L) ←

0 1 2 3 4

0 1 2 3 4

(H,L) ← (LA,0) ⊠ (X,0) where X is the original LB by 524 if no overflow in computing left endpoint, or by 492 if overflow. Overflow in 535 to ERR8F.

IF H = 0 AND L = 0

LA $<$ 0 and X = LB $<$ 0 by assumption of Case 5.

THEN BEGIN COMMENT EXPONENT UNDERFLOW;

LA $\neq$ 0 and LB $\neq$ 0 and (H,L) = (0,0) implies underflow.

H ← ϵ

ϵ $>$ LA ⊠ LB $>$ 0 so the double-precision quantity (ϵ, 0) is the best upper bound. L = 0 already.

END;

GO TO NEXTRMLT;

ERR8F:

Control reaches here only if exponent overflow at 535, 548 or 580. In the first two cases the product is positive while in the third H $>$ 0. Thus RB is set to +K, not an upper bound so INTERFLOW is set. We have already found the max so control goes to OVERFLOW for the finish, already covered in the non-Case 5 situation.

NEXTRMLT:

DOUBLE (RA, 0, RB, 0, ⊠, ←, HS, LS); 548

0 1 2 3 4

0 1 2 3 4

$H \geq 0$ and $L \neq 0$. But $H = 0$ is impossible if $L \neq 0$. RB is H^+. H is normalized by operation of DOUBLE since $H = 0$ has been ruled out. Hence we may use ROUNDUP.

RB ← ROUNDUP (H); 579-580

Overflow to ERR8F.

Control now passes to OVERFLOW for the finish, already covered in the non-Case 5 situation.

In Case 5, LB is set at 493 if overflow or at either 527 or 531 if no overflow. Similarly RB is set at 543 or either 576 or 580.

This completes the proof of MLT.

0 1 2 3 4

control passes to OVERFLOW.

Thus, we may start examining the code in more detail at 656. 658-672 is a single conditional. Applying Lemma 13, control passes through 673 to REND at 680, assuming no exponent overflow. In the code from 657-672, we show that LB is set correctly.

```
0     1     2     3     4

EXPOVR ← BIGEXPL;                                              656
DOUBLE (OPL1, 0, OPL2, 0, /, ←, H, L);                         657
     (H,L) ← (OPL1, 0)/ (OPL2, 0). Overflow to BIGEXPL.
IF L = 0                                                       658
THEN IF H = 0
     (H,L) = (0,0) means either an exact quotient of 0 or an
     underflow. It is underflow if OPL1 ≠ 0. The underflow will
     be corrected at 662-663.
     THEN IF OPL1 ≠ 0
          THEN BEGIN COMMENT EXPONENT UNDERFLOW;
               (H,L) = (0,0) and OPL1 ≠ 0 implies underflow.
               LB ← IF SIGN (OPL1) = SIGN (OPL2)               662
                    THEN 0 ELSE -ϵ;                            663
                    Neither OPL1 nor OPL2 is 0. Hence if the
                    signs agree, then before underflow, the

0     1     2     3     4
```

0 1 2 3 4

L ≠ 0 and H < 0 implies LB is H^-. H ≠ 0 since H < 0. H is normalized by operation of DOUBLE. Thus we can use ROUNDDOWN.

THEN LB ← ROUNDDOWN (H) 669-670

Overflow to BIGEXPL.

ELSE

H ≥ 0. But H = 0 is impossible. Since L ≠ 0, LB is H.

LB ← H; 672

Applying Lemma 13 (and an easy verification) shows LB is set precisely once in 658-672. LB is not altered in the remainder of DIV.

GO TO REND; 673

BIGEXPL:

Control reaches here only if exponent overflow at 657 or at 670. Assume overflow at 657. OPL1 ≠ 0 else no overflow. OPL2 ≠ 0 by the first part of divide proof. If both signs are alike, LB should be and is set to +K and control goes to REND. INTERFLOW is not set since +K is still a lower bound, the best that is machine representable. If the two signs are different, LB should be and is set to -K and control goes to REND. In the latter case,

0 1 2 3 4

0 1 2 3 4

different signs implies 0 is best upper bound.

END

ELSE

OPR1 = 0 so 0 is exact quotient.

RB ← 0

ELSE

$H \neq 0$. Since L = 0, Lemma 9 (i) implies no inexactness so H is exact result.

RB ← H

ELSE

$L \neq 0$. The statements for the same situation at left endpoints (667) hold. Therefore, Lemma 12 applies.

IF $H > 0$

$L \neq 0$ and $H > 0$ implies RB is H^+. $H \neq 0$ and H is normalized. Thus we can use ROUNDUP.

THEN RB ← ROUNDUP (H) 694-695

Overflow to BIGEXPR.

ELSE

$H \leq 0$. But H = 0 is impossible. Since $L \neq 0$, RB is H.

0 1 2 3 4

Proof of Termination.

It is easy to see that each of the four arithmetic operations terminates. First, we eliminate failure to return from a procedure call as a source of non-termination. The only procedure calls are:

NORMALIZE - called from ADD, MLT and DIV.

DOUBLE - called from ADD, MLT and DIV.

SIGN - called from MLT and DIV.

ADD - called from SUB.

We have proved that NORMALIZE terminates, and we have assumed that DOUBLE and SIGN terminate. Hence, ADD, MLT and DIV cannot fail to terminate because of procedure calls. Since this holds for ADD, it also holds for SUB.

Second, the only other source of non-termination is flow of control within an arithmetic operation. But flow of control is always toward the end of each arithmetic operation except:

DIV - 632 when control goes to 625 but then to OVERFLOW and exit.

MLT - Overflow at 498, 531 when control goes to ERR7F at 490 but then to RIGHT at 533.

MLT - Overflow at 548, 580 when control goes to ERR8F at 541 but then to OVERFLOW and exit.

Thus these three exceptions cannot cause non-termination either singly or jointly. Hence all four arithmetic operations terminate.

2. Unnormalized operands are permitted. Thus both rounding up and rounding down can be accomplished very cleanly by the unnormalized quantity RNDR.

3. The point (·) operator to manipulate subfields of operands.

4. A double-precision divide operator which truncates rather than rounds.

5. Sign-magnitude representation of operands.

6. Overflow trap.

7. Ability to write machine base constants directly (without conversion from decimal).

8. 13 octal digit mantissa and 2 octal digit exponent.

We clearly cannot anticipate all problems that might be encountered in converting our algorithms to all machines using all possible operating systems and languages. Nevertheless, we claim that analogs of each of the features exist on most other computers, namely:

1. The double-precision capability, which is crucial and central to the algorithms, exists or can be programmed on most computers. If necessary, it could be programmed on a digit-by-digit basis starting from first principles. What is important is that subroutines exist which compute

$$(H,L) \leftarrow (C,0) \text{ op } (D,0)$$

where the H and L can later be separately interrogated. Further, the non-mantissa part of the least significant word is considered all zeros.

of code for this case, only when we discovered our wrong assumption. There is a subtle problem if an overflow occurs only in the remultiply but not in the original divide. We had allowed for this. Thus it certainly should be possible to write a DIV if the machine divide rounds.

5. Other number representations may present problems, e.g. representation of negative numbers by complementation. The set M of numbers may change, and overflow and underflow tests may have to be altered.

6. Suppose the overflow trap is unavailable to the implementor at the proper language level. If, for example, an overflow is an unrecoverable error, so be it. And some systems set the results of an overflow to zero! Clearly some special testing may be needed to distinguish this case from a natural result of zero or even from an underflow.

Another interval arithmetic implementation on the B5500, by Lord (5), shows an approach if no overflow trap is available. To guard against overflow, he in effect pretests in a rough way the exponents of the operands to see if "overflow is likely." These tests give sufficient, but not necessary, conditions for non-overflow. Thus, unavoidably some bounds may be overly pessimistic.

7. The current implementation uses octal constants only to set ϵ and $+K$, but these values are never later tested for. Some problems could be avoided by redefining the set M and using

The proofs have another interesting use--serving as documentation of the procedures and, therefore, the algorithms. It has been suggested to us that the statements of the proofs of the five procedures make the code well-commented; we tend to agree. We do not claim that our documentation, which is presented in the form of proofs, is the best possible way to document. Our intent was proof, not documentation. But we do claim that our proofs meet reasonable standards of documentation in the sense that all relevant information is present somewhere. This is true, almost by definition of the meaning of proof.

Finally, if a user asked us to find the error in the procedures because he is not getting correct answers, a not unreasonable reply might be to ask him first to find the error in our proofs.

APPENDIX

This appendix is a listing of the source code of ADD, SUB, MLT, DIV, and NORMALIZE. It also shows declarations, initialization, and sequence numbers.

```
        COMMENT * * * * * * * * * *  * * * * * * * * * * * * * * *Q7QNORMALIZE;
PROCEDURE Q7QNORMALIZE(A);
REAL            A;
        COMMENT IF THE MANTISSA OF A IS ZERO, THEN SET A TO ALL ZEROS.
                IF A≠0 (MANTISSA≠0), THEN NORMALIZE A IF POSSIBLE.
                THE FOLLOWING ACTIONS ARE TAKEN IF A CAN NOT BE
                NORMALIZED:
                     IF Q7QLEFT IS TRUE, THEN A IS A LEFT ENDPOINT.
                          IF A>0, SET A TO ZERO.
                          IF A<0, SET A TO THE NEGATIVE, NORMALIZED
                               MACHINE NUMBER OF SMALLEST MAGNITUDE.
                     IF Q7QLEFT IS FALSE, THEN A IS A RIGHT ENDPOINT.
                          IF A<0, SET A TO ZERO.
                          IF A>0, SET A TO THE POSITIVE, NORMALIZED
                               MACHINE NUMBER OF SMALLEST MAGNITUDE;
BEGIN
IF A≠0
THEN BEGIN
     Q7QI←Q7QEXP←0;       Q7QEXP.Q7QMSIGN←A.Q7QXSIGN;
     Q7QEXP.Q7QM1213←A.Q7QEXPONENT;
     FOR Q7QI←Q7QI+1 WHILE A.Q7QM1=0 AND Q7QEXP>=-63 DO
          BEGIN
          Q7QEXP←Q7QEXP-1;
```

```
        COMMENT * * * * * * * * * * * * * * * * * * * * * * * * * * * Q7QADD;
PROCEDURE Q7QADD;
                        COMMENT Q7QADD ADDS THE CONTENTS OF THE INTERVAL
                                A AND B REGISTERS AND PUTS THE RESULT
                                IN B.
                                    A REGISTER = [Q7QLA,Q7QRA]
                                    B REGISTER = [Q7QLB,Q7QRB];
BEGIN
LABEL     SETR,ADRIGHT,ERDONE,DONE;
MONITOR         EXPOVR;
Q7QX←Q7QLA;     Q7QY←Q7QRA;     Q7QEXPA←Q7QEXPB←0;        Q7QRNDR←1;
IF INTERFLOW<0 THEN INTERFLOW←0;
Q7QLEFT←TRUE;
IF Q7QLA.Q7QM1=0 THEN Q7QNORMALIZE(Q7QLA);
IF Q7QLB.Q7QM1=0 THEN Q7QNORMALIZE(Q7QLB);
Q7QLEFT←FALSE;
IF Q7QRA.Q7QM1=0 THEN Q7QNORMALIZE(Q7QRA);
IF Q7QRB.Q7QM1=0 THEN Q7QNORMALIZE(Q7QRB);
EXPOVR ← SETR;
        COMMENT ADD LEFT ENDPOINTS FIRST.
                GET EXPONENTS FOR COMPARISON;
Q7QEXPA.Q7QMSIGN ← Q7QLA.Q7QXSIGN;
Q7QEXPB.Q7QMSIGN ← Q7QLB.Q7QXSIGN;
```

```
            THEN Q7QLB←Q7QLA                                                    2420
            ELSE                                                                2430
ELSE                                                                            2440
     BEGIN                                                                      2450
     DOUBLE(Q7QLA,0,Q7QLB,0,+,←,Q7QH,Q7QL);                                     2460
     IF Q7QL=0                                                                  2470
     THEN IF Q7QH=0                                                             2480
          THEN IF Q7QLA≠-Q7QLB                                                  2490
               THEN BEGIN COMMENT EXPONENT UNDERFLOW;                           2500
                    IF Q7QLA≥0                                                  2510
                    THEN IF Q7QLB>0                                             2520
                         THEN Q7QLB←0                                           2530
                         ELSE Q7QLB←IF -Q7QLB>Q7QLA                             2540
                                   THEN -Q7QLIMNO[1] ELSE 0                     2550
                    ELSE IF Q7QLB≤0                                             2560
                         THEN Q7QLB←-Q7QLIMNO[1]                                2570
                         ELSE Q7QLB←IF Q7QLB>-Q7QLA                             2580
                                   THEN 0 ELSE -Q7QLIMNO[1];                    2590
                    END                                                         2600
               ELSE Q7QLB←0                                                     2610
          ELSE Q7QLB←Q7QH                                                       2620
     ELSE IF Q7QH<0                                                             2630
          THEN BEGIN                                                            2640
```

```
                END                                                    2860
          ELSE                                                         2890
     ELSE IF Q7QRA≠0                                                   2900
          THEN Q7QRB←Q7QRA                                             2910
          ELSE                                                         2920
ELSE IF   Q7QEXPB-Q7QEXPA > 12                                         2930
     THEN IF Q7QRA>0                                                   2940
          THEN IF Q7QRB=0                                              2950
               THEN Q7QRB←Q7QRA                                        2960
               ELSE BEGIN                                              2970
                    Q7QRNDR.Q7QSEXPONENT←Q7QRB.Q7QSEXPONENT;           2980
                    Q7QRB←Q7QRB+Q7QRNDR;                               2990
                    END                                                3000
          ELSE IF Q7QRB=0                                              3010
               THEN Q7QRB←Q7QRA                                        3020
               ELSE                                                    3030
     ELSE                                                              3040
          BEGIN                                                        3050
          DOUBLE(Q7QRA,0,Q7QRB,0,+,←,Q7Q4,Q7QL);                       3060
          IF Q7QL=0                                                    3070
          THEN IF Q7Q4=0                                               3080
               THEN IF Q7QRA≠-Q7QRB                                    3090
                    THEN BEGIN COMMENT EXPONENT UNDERFLOW;             3100
```

```
        Q7QRB←Q7QLIMNO[0];                                                    3340
        INTERFLOW←IF INTERFLOW=-1 THEN -3 ELSE -2;                            3350
        END                                                                   3360
    ELSE Q7QRB←-Q7QLIMNO[0];                                                  3370
    DONE:                                                                     3380
    Q7QLA←Q7QX;   Q7QRA←Q7QY;                                                 3390
    IF INTERFLOW<0 AND Q7QSUBFLAG THEN INTERFLOW ← INTERFLOW-3;                3400
    IF INTERFLOW <0 THEN INTERFLOW ← -INTERFLOW;                               3410
    END Q7QADD ;                                                              3420
```

```
        COMMENT * * * * * * * * * * * * * * * * * * * * * * * * * * * * Q7QMLT;
PROCEDURE Q7QMLT;
                  COMMENT B ← A × B;
BEGIN
LABEL          CASE5,ERR7,ERR89,NOWRIGHT,OVERFLOW,RIGHT,ERR7F,ERR8F,
                NEXTLMLT,NEXTRMLT;
MONITOR        EXPOVR;
       COMMENT GET OPERANDS FOR COMPUTING RIGHT AND LEFT ENDPOINTS;
Q7QS←Q7QLA;    Q7QT←Q7QRA;    Q7QRNDR←1;
IF INTERFLOW<0 THEN INTERFLOW←0;
Q7QLEFT←TRUE;
IF Q7QLA.Q7QM1=0 THEN Q7QNORMALIZE(Q7QLA);
IF Q7QLB.Q7QM1=0 THEN Q7QNORMALIZE(Q7QLB);
Q7QLEFT←FALSE;
IF Q7QRA.Q7QM1=0 THEN Q7QNORMALIZE(Q7QRA);
IF Q7QRB.Q7QM1=0 THEN Q7QNORMALIZE(Q7QRB);
IF   Q7QLA < 0
THEN IF   Q7QRA < 0
     THEN IF   Q7QLB < 0
          THEN IF   Q7QRB < 0
               THEN
                    BEGIN COMMENT CASE 9;
                    Q7QDPL1 ← Q7QRA;
```

```
                    Q7QOPR1 ← Q7QLA;                                    4000
                    Q7QOPR2 ← Q7QLB;                                    4010
                    END                                                 4020
                ELSE GO TO CASE5                                        4030
            ELSE                                                        4040
                BEGIN COMMENT CASE 4;                                   4050
                Q7QOPL1 ← Q7QLA;                                        4060
                Q7QOPL2 ← Q7QOPR2 ← Q7QRB;                              4070
                Q7QOPR1 ← Q7QRA;                                        4080
                END                                                     4090
        ELSE IF   Q7QLB < 0                                             4100
            THEN IF   Q7QRB < 0                                         4110
                THEN                                                    4120
                BEGIN COMMENT CASE 3;                                   4130
                Q7QOPL1 ← Q7QRA;                                        4140
                Q7QOPL2 ← Q7QLB;                                        4150
                Q7QOPR1 ← Q7QLA;                                        4160
                Q7QOPR2 ← Q7QRB;                                        4170
                END                                                     4180
            ELSE                                                        4190
                BEGIN COMMENT CASE 2;                                   4200
                Q7QOPL1 ← Q7QOPR1 ← Q7QRA;                              4210
                Q7QOPL2 ← Q7QLB;                                        4220
```

```
          Q7QRNDR.Q7QSEXPONENT←Q7QH.Q7QSEXPONENT;                    4460
          Q7QLB←Q7QH-Q7QRNDR;                                        4470
          END                                                        4480
     ELSE Q7QLB←Q7QH;                                                4490
GO TO NOWRIGHT;                                                      4500
ERR7:                                                                4510
IF SIGN(Q7QOPL1)=SIGN(Q7QOPL2)                                       4520
THEN Q7QLB←Q7QLIMNO[0]                                               4530
ELSE BEGIN Q7QLB← -Q7QLIMNO[0];  INTERFLOW←-7; END;                  4540
NOWRIGHT:                                                            4550
EXPOVR ← ERR89;                                                      4560
DOUBLE(Q7QOPR1,0,Q7QOPR2,0,×,←,Q7QH,Q7QL);                           4570
IF Q7QL=0                                                            4580
THEN IF Q7QH=0                                                       4590
     THEN IF Q7QOPR1≠0 AND Q7QOPR2≠0                                 4600
          THEN BEGIN COMMENT EXPONENT UNDERFLOW;                     4610
               Q7QRB←IF SIGN(Q7QOPR1)=SIGN(Q7QOPR2)                  4620
                    THEN Q7QLIMNO[1] ELSE 0                          4630
               END                                                   4640
          ELSE Q7QRB←0                                               4650
     ELSE Q7QRB←Q7QH                                                 4660
ELSE IF Q7QH>0                                                       4670
     THEN BEGIN                                                      4680
```

```
        Q7QX←Q7QLB;                                                    4920
        Q7QLB←=Q7QLIMNO[0];                                            4930
        INTERFLOW←=7;                                                  4940
        GO TO RIGHT;                                                   4950
        END;                                                           4960
NEXTLMLT:                                                              4970
DOUBLE(Q7QRA,0,Q7QLB,0,×,←,Q7QHS,Q7QLS);                               4980
IF Q7QHS=0 AND Q7QLS=0 AND Q7QRA≠0                                     4990
THEN BEGIN COMMENT EXPONENT UNDERFLOW;                                 5000
        Q7QHS←=Q7QLIMNO[1];                                            5010
        END;                                                           5020
IF      Q7QH ≥ Q7QHS                                                   5030
THEN IF     Q7QHS < Q7QH                                               5040
        THEN                                                           5050
             BEGIN                                                     5060
             Q7QH ← Q7QHS;                                             5070
             Q7QL ← Q7QLS;                                             5080
             END                                                       5090
        ELSE                                                           5100
             BEGIN                                                     5110
             Q7QX ← Q7QL;                                              5120
             Q7QY ← Q7QLS;                                             5130
             Q7QL.Q7QMSIGN←Q7QH.Q7QMSIGN;                              5140
```

```
        Q7QH←Q7QLIMNO[1];                                          5380
        END;                                                       5390
    GO TO NEXTRMLT;                                                5400
    ERR8F:                                                         5410
        BEGIN                                                      5420
        Q7QRB←Q7QLIMNO[0];                                         5430
        INTERFLOW←IF INTERFLOW=-7 THEN -9 ELSE -8;                 5440
        GO TO OVERFLOW;                                            5450
        END;                                                       5460
    NEXTRMLT:                                                      5470
    DOUBLE(Q7QRA,0,Q7QRB,0,×,←,Q7QHS,Q7QLS);                       5480
    IF Q7QHS=0 AND Q7QLS=0 AND Q7QRA≠0 AND Q7QRB≠0                  5490
    THEN BEGIN COMMENT EXPONENT UNDERFLOW;                         5500
        Q7QHS←Q7QLIMNO[1];                                         5510
        END;                                                       5520
    IF   Q7QH ≤ Q7QHS                                              5530
    THEN IF   Q7QHS > Q7QH                                         5540
        THEN                                                       5550
            BEGIN                                                  5560
            Q7QH← Q7QHS;                                           5570
            Q7QL ← Q7QLS;                                          5580
            END                                                    5590
        ELSE                                                       5600
```

```
      IF INTERFLOW<0 THEN INTERFLOW← -INTERFLOW;                    5840
      END Q70MLT ;                                                  5850
```

```
                    Q7QOPR2 ← Q7QRB;                                    6090
                    END                                                 6100
               ELSE                                                     6110
                    BEGIN COMMENT CASE 6;                               6120
                    Q7QOPL1 ← Q7QRA;                                    6130
                    Q7QOPL2 ← Q7QOPR2 ← Q7QRB;                          6140
                    Q7QOPR1 ← Q7QLA;                                    6150
                    END                                                 6160
          ELSE                                                          6170
               BEGIN COMMENT CASE 3;                                    6180
               Q7QOPL1 ← Q7QRA;                                         6190
               Q7QOPL2 ← Q7QRB;                                         6200
               Q7QOPR1 ← Q7QLA;                                         6210
               Q7QOPR2 ← Q7QLB;                                         6220
               END                                                      6230
     ELSE                                                               6240
DIVZERO: BEGIN COMMENT DIVIDE BY B CONTAINING 0;                        6250
          INTERFLOW ← -13;                                              6260
          Q7QLB←-Q7QLIMNO[0]; Q7QRB←Q7QLIMNO[0];                        6270
          GO TO OVERFLOW;                                               6280
          END                                                           6290
ELSE                                                                    6300
     BEGIN                                                              6310
```

```
      END;                                                          6550
EXPOVR ← BIGEXPL;                                                   6560
DOUBLE(Q7QOPL1,0,Q7QOPL2,0,/,←,Q7QH,Q7QL);                          6570
IF Q7QL=0                                                           6580
THEN IF Q7QH=0                                                      6590
     THEN IF Q7QOPL1≠0                                              6600
          THEN BEGIN COMMENT EXPONENT UNDERFLOW;                    6610
               Q7QLB←IF SIGN(Q7QOPL1)=SIGN(Q7QOPL2)                 6620
                    THEN 0 ELSE -Q7QLIMNO[1];                       6630
               END                                                  6640
          ELSE Q7QLB←0                                              6650
     ELSE Q7QLB←Q7QH                                                6660
ELSE IF Q7QH<0                                                      6670
     THEN BEGIN                                                     6680
          Q7QRNDR,Q7QSEXPONENT←Q7QH,Q7QSEXPONENT;                   6690
          Q7QLB←Q7QH-Q7QRNDR;                                       6700
          END                                                       6710
     ELSE Q7QLB←Q7QH;                                               6720
GO TO RENO;                                                         6730
BIGEXPL:                                                            6740
     BEGIN COMMENT OVERFLOW;                                        6750
     IF SIGN(Q7QOPL1)=SIGN(Q7QOPL2)                                 6760
     THEN Q7QLB←Q7QLIMNO[0]                                         6770
```

```
      IF SIGN(Q7QDPR1)=SIGN(Q7QDPR2)                                    7010
      THEN BEGIN                                                        7020
           Q7QRB←Q7QLIMND[0];                                           7030
           INTERFLOW←IF INTERFLOW=-10 THEN -12 ELSE -11;                7040
           END                                                          7050
      ELSE Q7QRB←-Q7QLIMND[0];                                          7060
      END;                                                              7070
OVERFLOW:                                                               7080
Q7QLA←Q7QS;     Q7QRA←Q7QT;                                             7090
IF INTERFLOW<0 THEN INTERFLOW← -INTERFLOW;                              7100
END Q7QDIV ;                                                            7110
```