July 25, 1973

TO; W. Kahan

FROM: P. H. Sterbenz

PROPOSAL FOR DECIMAL FLOATING-POINT ARITHMETIC

by

THE COMPUTATION SPECIAL INTEREST GROUP

Specifications for Decimal Floating-Point Arithmetic

I. Format:

Single-Precision = 8 bytes
Double-Precision = 16 bytes

Mantissa length:

Single-Precision: 6-1/2 bytes = 13 decimal digits
Double-Precision: 14-1/2 bytes = 29 decimal digits

High order 1-1/2 byte:

Sign, invalid indicator, characteristic

Representation

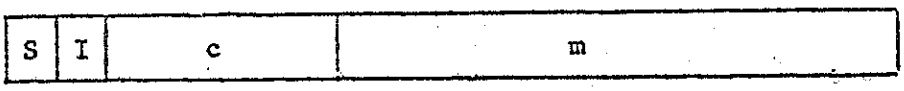
$$x = \pm 10^{e} m$$

e = exponent
m = mantissa (13 decimal digits for single-precision,
29 decimal digits for double-precision)

c = characteristic = e + 512

Allow 10 bits for c, so

$$0 \leq c \leq 1023$$
$$-512 \leq e \leq 511$$



S = sign
I = invalid indicator
c = characteristic
m = mantissa

Use "sign and true magnitude" representation for negative numbers.

Scaling of the mantissa

The mantissa of the floating-point number is written with the decimal point to the right of the leading digit. Thus, in the representation

$$x = \pm 10^{e} m$$

we have

$$-512 \leq e \leq 511$$

$$0 \leq m < 10$$

Normalized Numbers

(a) A non-zero number is normalized if the high order digit of its mantissa is non-zero. That is, if it is stored as

$$x = \pm 10^{e} m,$$

where $1 \leq m < 10$.

(b) Normalized zero:

sign +
characteristic = 0
mantissa = 0
invalid bit "off"

Unnormalized Number:

A number is unnormalized if it is not a normalized zero, but the high order digit of its mantissa is zero.

Rounding

Let x be a real number and let S be the set of p digit floating-decimal numbers. (Here p = 13 for single-precision and p = 29 for double-precision.) Let x_L and x_R be the left and right "neighbors" of x in S. That is:

x_L is the (algebraically) largest number in S which is $\leq x$

x_R is the (algebraically) smallest number in S which is $\geq x$

Then

(1) $x_L \leq x \leq x_R$.

(2) If x is in S, $x_L = x = x_R$.

(3) If x is not in S, $x_L < x < x_R$.

We define three operations, CHOP(x), ROUNDH(x) and ROUNDU(x) which round x to S, ROUNDH using a "half-adjust" round and ROUNDU using an "unbiased" round. The operations CHOP(x), ROUNDH(x), and ROUNDU(x) will always produce one of the neighbors x_L, x_R of x.

Here

(1) CHOP(x) produces the neighbor x_L or x_R having the smaller absolute value. (That is, it chops off all digits beyond the first p digits in x.)

(2) If one of the neighbors x_L or x_R is closer to x than the other one is, both ROUNDH(x) and ROUNDU(x) will produce the neighbor which is closest to x.

(3) If x_L and x_R are equally close to x:

(a) ROUNDH(x) will produce the neighbor having the larger absolute value.

(b) ROUNDU(x) will produce the neighbor whose low order digit is even.

Note: These definitions imply that all rounding and chopping takes place after normalization (and after right shifting to compensate for overflow in the add magnitude cases of addition and subtraction).

In general terms:

Need:

For CHOP(x): need first p digits of x
For ROUNDH(x): need first p+1 digits of x
For ROUNDU(x): need first p+1 digits of x and an indication of whether or not x has any non-zero digits to the right of the first p+1 digits.

General approach for rounding:

For ROUNDH(x):

(1) Obtain x normalized and chopped to p+1 digits.
(2) If $x \neq 0$, add 5 to the p+1-st digit of the absolute value of the mantissa, allowing carries to propagate. If this produces a high order carry, adjust the exponent and right shift the mantissa.
(3) Chop result to p digit.

For ROUNDU(x):

(1) Obtain x, normalized and chopped to p+q digits plus a sticky digit. Here q is 1 or 2. The sticky digit is 1 (or $\neq 0$) if x has non-zero digits to the right of the first p+q digits, otherwise it is zero.

(2) (a) If the p+1-st digit of x is not a 5, form ROUNDH(x).

(b) If the p+1-st digit of x is 5, but x has non-zero digits to the right of the first p+1 digits, form ROUNDH(x).

(c) If p+1-st digit of x is 5 and x has no non-zero digits to the right of the first p+1 digits, look at the p-th digit of x.

(i) If p-th digit of x is even, form CHOP(x)
(ii) If p-th digit of x is odd, form ROUNDH(x)

Rounded Arithmetic

We must decide on one of the rounding strategies, ROUNDH or ROUNDU, and then use this technique throughout. Mathematically, ROUNDU is preferable, because there are cases in which it reduces or eliminates bias. ROUNDH is probably easier to implement.

We shall assume that one of the rounding techniques ROUNDH, ROUNDU has been chosen, and we shall designate it by ROUND.

Let $\oplus$, $\ominus$, $*$, $\div$ denote floating-point addition, floating-point subtraction, floating-point multiplication, and floating-point division respectively. Then we want the hardware to produce the following results:

$$x \oplus y = \text{ROUND}\ (x+y)$$

$$x \ominus y = \text{ROUND}\ (x-y)$$

$$x * y = \text{ROUND}\ (xy)$$

$$x \div y = \text{ROUND}\ (x/y)$$

Details of Floating-Decimal Arithmetic

Let x and y be p digit floating-point numbers with

$$x = 10^e m$$

$$y = 10^f n$$

Here m and n are the mantissas of x and y, so they are signed numbers with

$$|m| < 10$$

$$|n| < 10.$$

We do not require that x and y be normalized. We shall assume that x and y are stored with the exponents e and f (that is, with the characteristics e + 512 and f + 512.)

Floating-Point Addition and Subtraction

(1) Compare e and f and interchange x and y if $e < f$. Then we may assume that $e \geq f$.

(2) Shift n to the right e-f places, keeping two guard digits and a sticky digit. That is, produce a p+3 digit number n' of the form

$$n' = x.xxx \ldots xxx$$

defined as follows:

(a) Let n" be a p+2 digit number of the form

$$n'' = x.xxx \ldots xxx$$

which is defined to be the first p+2 digits of $10^{-(e-f)}n$.

(b) The first p+2 digits of n' are the same as those of n".

(c) The p+3 -rd digit of n' is zero if $n'' = 10^{-(e-f)}n$, otherwise it is 1 (or any other convenient non-zero digit). Thus, if a non-zero digit is shifted out of the p+2 -nd position, it "sticks" in the p+3 -rd position.

(3) Add (or subtract if the operation is $\ominus$) the signed numbers m and n' to produce μ'. Since the addition of m to n' may produce a high order carry, μ' is a signed p+4 digit number of the form

$$\mu' = \pm\, xx.xxx \ldots xxx$$

Remark: The answer we want to produce is obtained by suitably normalizing $10^e\mu'$ and rounding it to p digits.

(4) We shall designate the answer by $10^g\mu$, where g and μ are described below.

(5) If $|\mu'| \geq 10$, set $g' = e+1$ and shift μ' one place to the right to produce a p+3 digit number μ'' of the form

$$\mu'' = \pm\ x.xxx\ \ldots\ xxx$$

All digits of μ' participate in this shift, with the low order digit acting as a sticky digit. That is, the low order digit of μ'' is zero if and only if the low order 2 digits of μ'' were zero.

(6) If $1 \leq |\mu| < 10$, let $g'=e$ and let $\mu'' = \mu'$.

(7) If $\mu' = 0$, set the answer to a normalized zero.

(8) If $0 < |\mu| < 1$, let k be the number of leading zero in μ' (not counting the tens position). Then

$$1 \leq |10^k \mu'| < 10$$

Set $g' = e-k$ and shift μ' to the right k places to produce μ''. All digits of μ' participate in this shift.

(9) As a result of steps 5, 7 or 8, we have produced a p+3 digit number μ'' of the form

$$\mu'' = \pm\ x.xxx\ \ldots\ xxx$$

(10) Let $\mu''' = \text{ROUND}(\mu'')$

(11) If step 10 does not produce a high order carry, we have $|\mu'''| < 10$, so we set

$$g = g'$$

$$\mu = \mu'''$$

(12) If step 10 produced a high order carry, set

$$g = g' + 1$$

$$\mu = \frac{\mu'''}{10} = \pm 1$$

Floating-Point Multiplication

1. Prenormalize the operands. If either operand is zero, set the result equal to a normalized zero.

2. Suppose that neither operand is zero. Since the operands are prenormalized, we may write

$$x = 10^e m, \quad 1 \le |m| < 10$$

$$y = 10^f n, \quad 1 \le |n| < 10.$$

3. Let $\mu' = mn$, so $1 \le |\mu'| < 100$. Then μ' may be written as a 2p digit number of the form

$$\mu' = \pm \text{ xx.xxx ... xxx}$$

Form the p+3 digit number μ'' by retaining the high order p+2 digits of μ' and a sticky digit. Thus,

$$\mu'' = \pm \text{ xx.xxx ... xxx}$$

The low order digit of μ'' is zero if and only if μ' has no non-zero digits to the right of the first p+2 digits.

4. We shall designate the answer by $10^g\mu$, where g and μ are described below.

5. If $|\mu''| \ge 10$, let $g' = e+f+1$ and $\mu''' = \mu''/10$

6. If $|\mu''| < 10$, let $g' = e+f$ and $\mu''' = \mu''$

7. Thus μ''' is a p+3 digit number of the form

$$\mu''' = \pm \text{ x.xxx ... xxx}$$

8. Let $\mu^{(iv)} = \text{ROUND}(\mu''')$

9. If step 8 does not produce a high order carry, we have $|\mu^{(iv)}| < 10$, so we set

$$g = g'$$

$$\mu = \mu^{(iv)}$$

10. If step 8 produces a high order carry, set

$$g = g' + 1$$

$$\mu = \frac{\mu^{(iv)}}{10} = 1$$

Remark: If we use ROUNDH instead of ROUNDU, we do not need the sticky digit. Then μ'' and μ''' can be p+2 digit numbers instead of p+3 digit numbers.

Floating-Point Division

1. Prenormalize the operands and provide special treatment for zero operands.

2. The divide command will probably pre-shift the dividend so that no post-normalization is required.

3. Develop a p+2 digit quotient q which is normalized and contains the first p+1 digits of x/y and a sticky digit. The sticky digit is zero if the quotient can be represented exactly in p+1 digits, otherwise it is non-zero.

4. Form ROUND(q)

Remark: If we use ROUNDH instead of ROUNDU, we don't need the sticky digit.

Remark: Instead of developing p+1 digits of the quotient, we can develop p digits of the quotient and a remainder. The rounding procedure is based on whether the remainder is less than, equal to, or greater than half the divisor

Unnormalized Operands:

The operations proceed exactly as described above. Note that the multiply and divide commands prenormalize the operands, but the add and subtract commands do not.

Unnormalized Operation Codes:

It is proposed that in addition to the normalized operation codes described above, there should be operation codes for unnormalized addition and subtraction. These operations omit post-normalization, but they retain rounding. In the add magnitude case, correct adjustment is made if there is a high order carry.

In all of the floating-point arithmetic operations ($\oplus$, $\ominus$, *, ÷), if either of the operands has the invalid bit "ON", the result will have the invalid bit "ON".

Special bit patterns will be used to represent ∞,-∞, and INDETERMINATE. These bit patterns will have the invalid bit "ON" and they will use a bit pattern which is distinct from that of any regular floating-point number.

Rules: Let a and b be regular floating-point numbers with $b \neq 0$. Then:

$a \oplus \infty = \infty$
$a \ominus \infty = -\infty$
$a \oplus (-\infty) = -\infty$
$a \ominus (-\infty) = \infty$

$\infty \oplus a = \infty$
$\infty \ominus a = \infty$
$(-\infty) \oplus a = -\infty$
$(-\infty) \ominus a = -\infty$

$b * (\pm\infty) = \pm\infty$ (normal sign control)
$a \div (\pm\infty) = 0$ (normalized zero)
$(\pm\infty) \div 0 = \pm\infty$
$b \div 0 = \pm\infty$

$\infty \oplus \infty = \infty$
$(-\infty) \oplus (-\infty) = -\infty$
$(-\infty) \ominus \infty = -\infty$
$\infty \ominus (-\infty) = \infty$

$\infty \ominus \infty =$ INDETERMINATE
$\infty \oplus (-\infty) =$ INDETERMINATE
$(-\infty) \oplus \infty =$ INDETERMINATE
$(-\infty) \ominus (-\infty) =$ INDETERMINATE

$(\pm\infty) \div (\pm\infty) =$ INDETERMINATE
$0 \div 0 =$ INDETERMINATE
$0 * (\pm\infty) =$ INDETERMINATE
$(\pm\infty) * 0 =$ INDETERMINATE

In comparisons:

$\infty > a$
$-\infty < a$
$\infty > -\infty$

Questions

1. We really don't know the correct sign to give to $(\pm\infty) \div 0$ or $b \div 0$. It is probably reasonable to give the result the same sign as the numerator.

2. Should two ∞'s compare as equal? Should two INDETERMINATE's compare as equal?

Remarks: In considering the speed of the operations, emphasis should be placed on the speed with which the hardware handles the case in which both operands are "valid". If either operand is "invalid", speed may be degraded somewhat.

Special Operation Codes

The following operations should be easy to perform. There probably should be special operation codes for many of them.

A. Sign control:

1. Test sign of x
2. ABS(x)
3. -x

B. Invalid Bit:

1. Test to see whether invalid bit is "ON"
2. Set invalid bit "ON"
3. Set invalid bit "OFF"
4. Extract invalid bit
5. Test to determine whether a number is ∞, $-\infty$, INDETERMINATE, or "other".

C. Dismantling and Assembling:

1. Extract exponent
2. Extract mantissa
3. Assemble

D. Scaling and Shortening

1. Normalize
2. Take integer part
3. Write as unnormalized number with exponent k
4. Chop to k digits
5. Round to k digits
6. Round double-precision number to single-precision.
7. Chop double-precision number to single-precision.
8. Extend single-precision number to double-precision by appending zeros.

E. Mixed Operations

1. Multiply two single-precision numbers to produce a double-precision result.

2. Add a single-precision number to a double-precision number to produce a double-precision result.

F. Reduction

1. SUM
2. PROD
3. DOT
4. POLY

We shall use the term "exponent spill" to include both overflow and underflow.

Comments:

1. Exponent spill should cause an interrupt.

2. We want it to be easy to change the way the software responds to the interrupt. We want to be able to make this change in the middle of a program at the cost of, say, a move.

3. Performance may be degraded when spill occurs.

4. Result produced after spill:

   (a) invalid bit is "ON"

   (b) result should have correct sign, correct mantissa, and a "wrapped-around" characteristic. That is, the characteristic is 1024 too small after overflow and 1024 too large after underflow.

5. For distributed operations, the operation should be completed before the interrupt is taken.

6. For reduction, set the intermediate result to $\pm\infty$ after overflow and 0 after underflow and continue the operation to completion. Take the interrupt at the end of the operation.

Questions

It would be desirable to look at the complete set of operation codes and consider the coding of, say, argument reduction for simple functions or higher-than-double-precision arithmetic. No one is now planning to do this.