27 Bits Are Not Enough for 8-Digit Accuracy

I. Bennett Goldberg*
General Electric Company, Cambridge, Mass.

From the inequality $10^8 < 2^{27}$, we are likely to conclude that we can represent 8-digit decimal floating-point numbers accurately by 27-bit floating-point numbers. However, we need 28 significant bits to represent some 8-digit numbers accurately. In general, we can show that if $10^p < 2^{q-1}$, then q significant bits are always enough for p-digit decimal accuracy. Finally, we can define a compact 27-bit floating-point representation that will give 28 significant bits, for numbers of practical importance.

1. Introduction

This paper shows that even though $10^8 < 2^{27}$, 27 significant bits are not enough for 8-digit[1] floating-point accuracy. In other words, an input routine may convert an 8-digit floating-point number into the corresponding 27-bit floating-point number, and an output routine may convert the 27-bit number back to the form of the 8-digit number; but the final result is not necessarily the original number (even if we allow the routines to fudge the 27th bit and the 8th digit).

To show how the situation arises, we first consider the simple case, of representing 2 digits by 7 bits, in fixed-point notation. Then, we discuss the general case, of representing p digits by q bits, in floating-point notation. We prove that if $10^p < 2^{q-1}$, then q bits are enough for p-digit accuracy. Applying this rule to the 8-digit problem ($10^8 < 2^{28-1}$), we conclude that 28 bits are enough for 8-digit accuracy. Finally, we define a compact 27-bit floating-point representation that will give us the 28th significant bit for free, for all numbers of practical importance.

Although we can generalize the discussion to any pair of radices, we consider floating-point numbers of radix 2 and 10 only, for simplicity. We shall ignore the sign, since it is not used in a radix conversion.

* Resident visitor at Bell Telephone Laboratories, Murray Hill, N. J., December 1965–January 1967.

[1] We shall abbreviate binary digit as "bit" and decimal digit as "digit." We shall omit "significant" where its intent is obvious.

2. The Simple Case

We can represent each of the 90 decimal integers 10, 11, ⋯, 98, 99 by the *equivalent* 7-bit integer, with no question regarding accuracy, as follows:

1 0	0 0 0 1 0 1 0
1 1	0 0 0 1 0 1 1
1 2	0 0 0 1 1 0 0
⋯	⋯
9 7	1 1 0 0 0 0 1
9 8	1 1 0 0 0 1 0
9 9	1 1 0 0 0 1 1

We can represent each of the 90 decimal fractions .10, .11, ⋯, .98, .99 represented by the *approximate* binary fraction rounded to 7 places:

.1 0	.0 0 0 1 1 0 1
.1 1	.0 0 0 1 1 1 0
.1 2	.0 0 0 1 1 1 1
⋯	⋯
.9 7	.1 1 1 1 1 0 0
.9 8	.1 1 1 1 1 0 1
.9 9	.1 1 1 1 1 1 1

The binary rounding error cannot exceed $\pm 1/256$. Therefore, a decimal fraction and its approximate binary fraction cannot be more than 1/256 apart. Since $1/256 < .005$, we conclude that the binary fraction must convert back to the original decimal fraction, if we round to 2 decimal places. (See Section 3 for a more rigorous discussion of rounding.)

However, a problem arises when we try to represent the 20 decimal numbers 8.0, 8.1, ⋯, 9.8, 9.9 by 7 bits. The integer digit requires 4 bits:

8.0	1 0 0 0.X X X
8.1	1 0 0 0.X X X
⋯	⋯
9.8	1 0 0 1.X X X
9.9	1 0 0 1.X X X

No matter how we compute the fraction bits XXX, there can be only 16 distinct binary numbers. Hence, we can recover only 16 of the original 20 decimal numbers. Floating-point notation does not change the situation. The binary numbers become $.1000XXX \cdot 2^4$ and $.1001XXX \cdot 2^4$. Therefore, 7 bits are not enough for 2-digit accuracy.

The same problem arises trying to represent the 10,000,000 decimal numbers 9000000.0, 9000000.1, ⋯, 9999999.8, 9999999.9 by 27 bits. The integer part requires 24 bits ($2^{23} = 8{,}388{,}608$), and there can be only 8,000,000 distinct binary numbers. Therefore 27 bits are not enough for 8-digit accuracy.

If another significant bit were used in either example, then we would have twice as many binary numbers (more than enough), at intervals of $1/16 < 1/10$. Then we could represent the fraction digit.

3. The General Case

We have seen from the 2-digit case that if $10^p < 2^q$, then q bits may not be enough for p-digit accuracy. Now

we may ask whether $q+1$ bits are enough. In other words, if $10^p < 2^{q-1}$, are q bits enough for p-digit accuracy?

THEOREM. *Let x be any decimal floating-point number with p significant digits ($p > 0$). Let y be the corresponding binary floating-point number, rounded to q significant bits. For y, let z be the corresponding decimal floating-point number, rounded to p significant digits. Then, if $10^p < 2^{q-1}$, we have $x = z$, and y is accurate to p significant digits.*

PROOF. Since floating-point zero can be defined uniquely as zero with the smallest possible machine exponent,we may assume that $x > 0$, $y > 0$, and $z > 0$. Therefore, we can find integers m and n, such that $2^{n-1} \leqq x < 2^n$ and $10^{m-1} \leqq y < 10^m$. Since y is rounded to q significant bits, we have:

$$y \leqq x + (2^{n-q})/2 < y + 2^{n-q} \tag{1}$$

$$2^{n-1} \leqq y. \tag{2}$$

The quantity $2^{n-q} = .00\cdots01\cdot2^n$ is the interval between two successive binary numbers $.y_1y_2\cdots y_q\cdot2^n$. If $x = 2^{n-1}$, then $y = 2^{n-1}$.

Since z is rounded to p significant digits, we have:

$$z \leqq y + (10^{m-p})/2 < z + 10^{m-p}. \tag{3}$$

The quantity $10^{m-p} = .00\cdots01\cdot10^m$ is the interval between two successive decimal numbers $.z_1z_2\cdots z_p\cdot10^m$.

Combining (2) with $y < 10^m$ and rephrasing $10^p < 2^{q-1}$, we get:

$$2^{n-1} \leqq y < 10^m \tag{4}$$

$$2^{1-q} < 10^{-p}. \tag{5}$$

Multiplying (4) and (5), we have:

$$2^{n-q} < 10^{m-p}. \tag{6}$$

Rephrasing (1) and (3), we obtain:

$$-(2^{n-q})/2 \leqq x - y < (2^{n-q})/2 \tag{7}$$

$$-(10^{m-p})/2 \leqq y - z < (10^{m-p})/2. \tag{8}$$

Combining (6) with (7), we find:

$$-(10^{m-p})/2 < x - y < (10^{m-p})/2. \tag{9}$$

Adding (8) and (9), we have:

$$-10^{m-p} < x - z < 10^{m-p}. \tag{10}$$

Therefore $x = z$, and y is accurate to p significant digits. This completes the proof of the theorem.

We conclude that if $10^p < 2^{q-1}$ then q bits are enough for p-digit accuracy. The smallest value of q for which this rule applies satisfies the inequalities $2^{q-2} < 10^p < 2^{q-1}$, which are equivalent to $q-2 < p\cdot\log_2 10 < q - 1$. Since $q-2$ is a positive integer, we have $q - 2 = [p\cdot\log_2 10]$, where brackets denote "the integer part of;" in other words:

$$q = [p\cdot\log_2 10] + 2 \cong [3.322\cdot p] + 2. \tag{11}$$

If (11) is applied to $p = 1, 2, \cdots, 28$, then we obtain the following table:

p	1	2	3	4	5	6	7	8	9	10	11	12	13	14
q	5	8	11	15	18	21	25	28	31	35	38	41	45	48
p	15	16	17	18	19	20	21	22	23	24	25	26	27	28
q	51	55	58	61	65	68	71	75	78	81	85	88	91	95

We conclude that 8 bits are sufficient for 2-digit accuracy, and 28 bits for 8-digit accuracy.

To show that 4 bits are not enough for 1-digit accuracy, we consider the following case:

$$.1000\cdot2^{-69} = 8\cdot2^{-73} \cong 8.470\cdot10^{-22} \cong 8\cdot10^{-22}$$

$$.1001\cdot2^{-69} = 9\cdot2^{-73} \cong 9.529\cdot10^{-22} \cong 10\cdot10^{-22}.$$

We cannot recover $9\cdot10^{-22}$, unless there is fudging in the output routine.

4. A Compact 27-Bit Representation

We have seen that 28 bits are enough for 8-digit floating-point accuracy. We now ask whether we can squeeze 28 significant bits into a compact 27-bit representation, for numbers of practical importance.

Let us examine a typical floating-point number $y = .y_1y_2\cdots y_{27}y_{28}\cdot2^n$ with 28 significant bits. We have $y_1 = 1$, unless y = floating-point zero $= .00\cdots00\cdot2^{-e}$, where $-e$ is the smallest possible machine exponent. But we do not use y_1 alone, to distinguish floating-point zero from other floating-point numbers.

Let us now define a compact 27-bit representation for storing 28 significant bits. We must recognize two distinct cases: (1) For $n = -e$, assume that $y_{28} = 0$ and store $y_1y_2\cdots y_{26}y_{27}$. (2) For $n \neq -e$, assume that $y_1 = 1$ and store $y_2y_3\cdots y_{27}y_{28}$.

We can still represent floating-point zero and very small numbers ($.00\cdots01\cdot2^{-e}$ to $.11\cdots11\cdot2^{-e}$) with the same 27-bit precision as before. For all other floating-point numbers, we get the 28th significant bit as a bonus.

If we use the compact 27-bit representation in memory, then we should still use the full 28-bit representation in the arithmetic registers. Otherwise, the normalized floating-point instructions will not be compatible with the other arithmetic instructions. Therefore, each normalized instruction must translate from one representation to another, whenever necessary. On a machine with operation overlap, this translation should normally occur during the execution of another instruction. A delay should occur only when the second of two consecutive normalized instructions fetches the number stored by the first instruction (a normalized store instruction).

Acknowledgment. I would like to thank Dr. R. W. Hamming of Bell Telephone Laboratories for his suggestions and assistance in preparing this paper.

RECEIVED AUGUST, 1966; REVISED OCTOBER, 1966