RC 15683 (#69737) 4/19/90
Computer Science/Mathematics 10 pages

Research Report

Fast and Perfectly Rounding Decimal/Hexadecimal Conversions

Gordon Slishman

IBM Research Division
T. J. Watson Research Center
Yorktown Heights, NY 10598

IBM Research Division
Almaden • T.J. Watson • Toyko • Zurich

§1 Motivation

Floating point conversion codes, generally embedded in input/output services, facilitate communication between user and computer, the user conversant in base ten and the computer most fluent in base 2^i, $i \in \{1, 2, \ldots\}$.

With the advent of System/360 in 1964, conversions between base ten and base sixteen entered a chaotic period. Errors and inconsistencies abounded: Compilers converted data inconsistently with one another; a number compiled might differ from the same number input to the compiled program; checkpoint/restart worked reliably only with unformatted I/O; and merely printing the number 1.0 yielded outputs ranging from .99998 to 1.0. The discontent as measured by semiannual Share resolutions seemed to peak with Share 33, in 1969, when four of the top five outstanding resolutions addressed conversion problems [3].

Improved conversion codes appeared in the early 1970's with the advent of extended precision floating point architecture and System/370. In the course of adding extended precision subroutines to its math libraries, IBM rewrote conversion codes to achieve consistency and accuracy "near perfection," perfection defined as rounding to nearest machine number or, for ties, to the nearest of greater magnitude. Conversion resolutions quickly disappeared from the proceedings of Share.

As customers seemed content, IBM only twiddled with its conversion codes until 1987, when a FORTRAN customer complained that 610.75 printed as 610.7 (while 111.75 printed as 111.8). This customer complaint instigated a fresh look at *fast* and *perfectly rounding* floating point conversion.

§2 Theory

Matula's Base Conversion Theorem [1,2,3] states some inherent limits of perfectly rounding floating point conversion. The theorem implies that for a given number of decimal digits and a given number of hexadecimal digits, conversion can be one-to-one in at most one direction and onto in at most one direction but never in one direction both one-to-one and onto. Specifically, for S/370 floating point numeration, the Base Conversion Theorem states the minimum number of decimal digits required for reliable checkpoint/restart, which allows users to convert intermediate results to decimal, inspect those results, and restart execution without perturbation. See "§7 Checkpoint/Restart" on page 8 for further discussion of checkpoint/restart. Interested readers can consult Matula's work directly for a statement of the Base Conversion Theorem, which bounds the utility of a perfectly rounding conversion code but does not deal with its design or implementation.

§3 Prior System/370 Codes

As of this date, no two of the following IBM products convert floating point data identically: ACRITH, APL2, Assembler-H, PL/1, VM/CP, and VSFORTRAN. Worse, none always converts correctly. The PL/1 compiler even precludes checkpoint/restart by limiting double precision output to 16 decimal digits rather than the required 18 (see "§7 Checkpoint/Restart" on page 8). Of the products listed, the VSFORTRAN conversions coded by Kuki and Ascoly [4] offer the best precision/performance tradeoff. Their code runs fastest, usually rounds correctly, and evidently, never errs by more than one unit in the last place (*ulp*). But in rounding 111.75 to 111.8 and 112.75 to 112.7, VSFORTRAN's conversion code still leaves room for improvement.

The following IBM Research algorithms as implemented to date convert between single, double and extended precision System/370 hexadecimal and one- to thirty-five-digit decimal floating point representations. The Research codes surpass the speed of VSFORTRAN's floating point conversion codes on average and *always* round correctly.

§4 Hexadecimal to Decimal Floating Point Conversion

Let hexadecimal representations of the integral powers of 10 from 10^{-111} to 10^{+130} reside in a 6960-byte table, including a 256-byte logarithm table and a 464-byte table of offsets to the various powers. The non-negative powers are exactly representable rather compactly, because every fourth power of ten adds a trailing zero to the hexadecimal mantissa. The longest non-negative power requires only 76 hexadecimal digits of mantissa, while each negative power is truncated after 42 digits, spanning three double precision parts.

To convert the machine number X, with infinite decimal representation $.d_1d_2d_3 \ldots \times 10^K$, to a nearest D-digit decimal representaion (rounding ties away from zero), the first step is to compute K. In S370 floating point numeration there are only 128 hexadecades to which a positive normalized machine number can belong, and each hexadecade can span no more than three decades, the ratio of two consecutive powers of 16 being less than 100. The exponent part of the floating point characteristic provides an offset into the 256-byte logarithm table and selects the first of the three decades to which X may belong. Because K is the smallest integer such that $X < 10^K$, at most two comparisons determine K precisely. (If X is too big for the first decade selected and for the second, then X has to belong to the third.)

K established, the second step is to compute $X \times 10^{D-K} + 0.5 = d_1d_2 \ldots d_D.d_{D+1} \ldots + 0.5$ with sufficient precision that the truncation error is less than $.0001_{16}$. Such precision can be achieved because X is assumed exact, 10^{D-K} is tabulated fully or truncated after no fewer than 42 hexadecimal digits, and 0.5 is a machine number. If the fractional part of the product is less than $.FFFF_{16}$, then adding in a correction for the truncation error wouldn't affect the integer part, where the D decimal digits materialize as a hexadecimal integer. Almost all, precisely 65535 of 65536, fractional parts compare favorably, and then the Convert-to-Decimal instruction completes the conversion rapidly and perfectly, nine decimal digits at a time.

Rarely, when the fractional part of the product equals or exceeds $.FFFF_{16}$, one of two possible subcases exits: Either $D - K$ is negative, or not. The easier subcase is the latter, because then 10^{D-K} is an integer, and $X \times 10^{D-K} + 0.5$ is computable exactly.

The negative subcase needs slight recasting because 10^{D-K} has no finite hexadecimal representation. Recast thusly:

$$X \times 10^{D-K} \geq INT(X \times 10^{D-K}) + 0.5 \Leftrightarrow X \geq (INT(X \times 10^{D-K}) + 0.5) \times 10^{K-D}.$$

$INT(X \times 10^{D-K})$ is computable because $0.5 - .0001_{16} < FRAC(X \times 10^{D-K}) \leq 0.5$, and 10^{K-D} is an integer. Therefore, the right side of the second inequality is computable without error to direct the rounding correctly.

§5 Decimal to Hexadecimal Floating Point Conversion

Let's start with the same 6960-byte table previously described for hexadecimal to decimal conversion. Here our problem is to compute nearest machine number $X = .d_1d_2 \ldots d_D \times 10^K + 0.5ulp$. Step one is to convert the decimal integer $d_1d_2 \ldots d_D$ to hex by means of the Convert-to-Binary instruction, nine digits at a time.

K given, next compute $d_1d_2 \ldots d_D \times 10^{K-D} + 0.5ulp$ with sufficient precision that the truncation error is less than $.0001_{16}ulp$. Such precision can be achieved because the decimal fraction is assumed exact, 10^{K-D} is tabulated fully or truncated after no fewer than 42 hexadecimal digits, and $0.5ulp$ is a machine number. If the truncation of the product to the target precision would chop less than $.FFFF_{16}ulp$, then adding in a correction for all truncation errors committed in multiplication and accumulation wouldn't affect the product digits to be retained. Therefore, these two steps convert decimal floating point to hexadecimal perfectly and rapidly, save once in 65536.

When the fractional part of the product equals or exceeds $.FFFF_{16}ulp$ one of two possible subcases exits: Either $K - D$ is negative, or not. The easier subcase is again the latter, because then 10^{K-D} is an integer, and the product is computable exactly.

The negative subcase again needs slight recasting because 10^{K-D} has no finite hexadecimal representation. Recast thusly:

$$d_1d_2 \ldots d_D \times 10^{K-D} \geq TRUNC(d_1d_2 \ldots d_D \times 10^{K-D}) + 0.5ulp$$

$$\Leftrightarrow$$

$$d_1d_2 \ldots d_D \geq (TRUNC(d_1d_2 \ldots d_D \times 10^{K-D}) + 0.5ulp) \times 10^{D-K}$$

$TRUNC(d_1d_2 \ldots d^D \times 10^{K-D})$ is computable, because $0.5 - 0.0001_{16}ulp < choppedpart \leq 0.5ulp$, and 10^{D-K} is an integer. Therefore the right side of the second inequality is computable exactly to direct the rounding correctly.

§6 Exact Computability

Rarely, 1 case in 65536, when we have to compute products exactly to choose the correct rounding, a fixed-point accumulator of 200 hexadecimal digits more than suffices. The assumed hexadecimal point resides after the first 64 hex digits, and before the last 136.

To see the sufficiency, consider first hexadecimal to decimal conversion. If $D - K$ is non-negative, then the exact product $X \times 10^{D-K}$ consists of fewer than 28 plus 76 hex digits (worst case: extended precision times a positive power of 10 with a 76 hex digit mantissa) and lies between 1 and 10^{35} . If $D - K$ is negative, then X is greater than 1, and again the product consists of fewer than 104 hex digits and lies between 1 and 16^{63}. These products easily fit in the defined accumulator.

Decimal to hexadecimal conversion involves products formed from an integer of at most 30 hex digits (35 decimal) and a power of 10 of fewer than 76 hex digits. The product operands are scaled so that the significant digits of the product are left-aligned in the fraction of the accumulator. The product, with fewer than 106 hex digits, easily fits in the 136 hex digit fraction.

§7 Checkpoint/Restart

Given a perfectly rounding conversion program for S/370, one can easily derive the minimum number of decimal digits to support checkpoint/restart. Let H be an h-digit hexadecimal number and D be a closest d-digit decimal. A perfectly rounding hexadecimal-to-decimal conversion "out" perturbs the original hexadecimal number, H, no more than $0.5ulp$ of the decimal representation. If the perturbation $0.5ulp(D)$ is less than $0.5ulp(H)$, then D must be closer to H than to any other hexadecimal number in the precision of H, and therefore the perfectly rounding conversion "in" will certainly return to H. Expressing this observation algebraically, the first inequality following states a sufficient condition in terms of H and D for checkpoint/restart or one-to-one out-then-in mapping. The subsequent inequalities offer a chain of sufficient conditions, the last of which is in terms of h and d, the number of hexadecimal and decimal digits respectively that guarantees checkpoint/restart for all H and D.

$$H \to D \to H \quad \text{if} \quad 0.5ulp(D) < 0.5ulp(H).$$

$$0.5ulp(D) < 0.5ulp(H) \quad \text{if} \quad ulp(D) \le D \times 10^{1-d} < H \times 16^{-h} < ulp(H).$$

$$ulp(D) \le D \times 10^{1-d} < H \times 16^{-h} < ulp(H) \quad \text{if} \quad 16^h < 10^{d-1} \times H/D.$$

$$16^h < 10^{d-1} \times H/D \quad \text{if} \quad h \log_{10} 16 < d - 1 + \log_{10}(H/D).$$

$$h \log_{10} 16 < d - 1 + \log_{10}(H/D) \quad \text{if} \quad 1 + h \times \log_{10} 16 + 16^{1-h} < d.$$

$$\text{Therefore,} \quad H \to D \to H \quad \text{if} \quad 1 + h \times \log_{10} 16 + 16^{1-h} < d.$$

	hex digits(h)	dec digits (d)
single	6	9
double	14	18
extended	28	35

§8 Performance versus VS FORTRAN 2.3 on 3090 400E

Each of these performance numbers states the best of 10 trials under CMS on a busy system. The FORTRAN interface differs from ours in several respects, and so the comparisons are not strictly "apples to apples." For examples, the FORTRAN code supports the G format and receives its decimal exponent in binary; our code has no G format support but receives its decimal exponent as decimal characters, which must be converted to binary. The following figures merely show that perfect rounding and excellent average performance can coexist in conversion codes.

hexadecimal to decimal

Precision	*Yorktown(ns)*	*VSFORT(ns)*	*speedup*
single to 09 digits	2914	12145	4.17
single to 18 digits	5027	12974	2.58
single to 27 digits	7276	14137	1.94
single to 35 digits	10732	15090	1.41
double to 09 digits	2926	12880	4.40
double to 18 digits	5003	14766	2.95
double to 27 digits	7308	16094	2.20
double to 35 digits	10667	16857	1.58
extended to 09 digits	3081	15384	4.99
extended to 18 digits	5165	17249	3.34
extended to 27 digits	7417	19759	2.66
extended to 35 digits	10884	22091	2.03

decimal to hexadecimal

09 digits to single	2709	6265	2.31
18 digits to single	3502	5974	1.71
27 digits to single	3954	5771	1.46
35 digits to single	4421	5480	1.24
09 digits to double	3711	6728	1.81
18 digits to double	4403	8266	1.88
27 digits to double	4883	7899	1.62
35 digits to double	5662	7566	1.34
09 digits to extended	5572	8626	1.55
18 digits to extended	6869	10332	1.50
27 digits to extended	7999	12263	1.53
35 digits to extended	9490	14111	1.49

§9 Performance versus VS FORTRAN 2.3 on 3090 300S

hexadecimal to decimal

Precision	*Yorktown(ns)*	*VSFORT(ns)*	*speedup*
single to 09 digits	2526	10554	4.18
single to 18 digits	4108	11341	2.76
single to 27 digits	5779	12302	2.13
single to 35 digits	8553	13085	1.53
double to 09 digits	2514	11252	4.48
double to 18 digits	4092	12812	3.13
double to 27 digits	5787	13771	2.38
double to 35 digits	8553	14482	1.69
extended to 09 digits	2639	13337	5.05
extended to 18 digits	4228	14931	3.53
extended to 27 digits	5899	17054	2.89
extended to 35 digits	8687	18937	2.18

Decimal to Hexadecimal

09 digits to single	2245	5399	2.40
18 digits to single	2950	5174	1.75
27 digits to single	3312	4964	1.50
35 digits to single	3767	4777	1.27
09 digits to double	2949	5804	1.97
18 digits to double	3647	7128	1.95
27 digits to double	4076	6802	1.67
35 digits to double	4678	6465	1.38
09 digits to extended	4644	7443	1.60
18 digits to extended	5532	8818	1.59
27 digits to extended	6496	10458	1.61
35 digits to extended	7836	12090	1.54

§10 Bibliography

1. D. Matula, A Formalization of Floating-Point Numeric Base Conversion, *IEEE Transactions on Computers,* **C-19, No. 8,** August 1970
2. D. Matula, Base Conversion Mappings, *Spring Joint Computer Conference, AFIPS Proc.* **vol. 30,** 1967
3. D. Matula, The Base Conversion Theorem, *Proc. Amer. Math Soc.* **vol. 19, no. 3,** 1968
4. H. Kuki and J. Ascoly, FORTRAN extended-precision library, *IBM Systems Journal,* **vol. 10, no. 1,** 1971