Higher Radix Floating Point Representations

Paul Johnstone
Telerate Systems Incorporated
1017 Pleasant Street
New Orleans, LA 70115

Frederick E. Petry
Dept. of Computer Science
Tulane University
New Orleans, LA 70118

Abstract

This paper examines the feasibility of higher radix floating point representations, and in particular, decimal based representations. Traditional analyses of such representations have assumed the format of a floating point datum to be roughly identical to that of traditional binary floating point encodings such as the IEEE P754 task group standard representations. We relax this restriction and propose a method of encoding higher radix floating point data with range, precision, and storage requirements comparable to those exhibited by traditional binary representations. Results from McKeeman's Maximum and Average Relative Representational Error (MRRE and ARRE) analyses, Brent's RMS error evaluation, Matula's ratio of significance space and gap functions, and Brown and Richman's exponent range estimates are extended to accommodate the proposed representation. A decimal alternative to traditional binary representations is proposed and the behavior of such a system is contrasted with that of a comparable binary system.

Higher radix floating point representations, and in particular, representations using non-binary commensurable radices have generally been eschewed by researchers and systems' designers as inferior to traditional binary floating point encodings. The criticisms against such non-binary systems generally focus on their algorithmic intractability and their relatively poorer error performance vis-a-vis binary systems. However, the interest in non-binary systems is not merely theoretical: conversions between floating point representations in environments in which data are supplied, processed, and presented in different floating point bases has a nontrivial computational cost and potential for generating significant errors in conversion [20]. It is proposed in this paper that many of the algorithmic problems raised in earlier research have become moot given new technologies, and that methods of encoding higher radix floating point representations exist that exhibit error performance comparable to that of traditional binary encodings.

Representation

Floating point representations may be viewed as a hybrid form of scientific notation designed to offer considerable range without the storage requirement that a fixed point encoding would require. Therefore, an idealized view of a floating point representation [23] is of the form:

$$X = (s, e, f) = (-1)^s * f * \beta^e, \quad (1)$$

where,

β – base of representation
f – a base β magnitude
e – an exponent referenced to base β
s – algebraic sign, $0 \le s \le 1$

Additionally, many floating point representations will specify that the significand, f, is encoded in a "normalized" format, viz.,

$$\beta^n > f \ge \beta^{n-1} \quad (2)$$

and, further, n has been assumed to be zero, yielding significands in the range $(1,\beta^{-1}]$. In binary systems, such forms possess an inherent advantage in that the leading bit assumes the role of a sentinel; i.e. it is not a meaningful datum, and therefore need not be stored explicitly. Higher radices do not possess this convenient feature since their "digits" do not encode integrally in a binary storage medium. For example, if one assumes an encoding analogous to that of a traditional binary representation, the leading bits of a decimal significand (i.e. the exponent is referenced to base 10) encoded in binary exhibit the forms:

$.1x_{p-1}x_{p-2}x_{p-3}x_{p-4}\ldots$ x_0 $1 > f \ge 1/2$, (3)
$.01x_{p-2}x_{p-3}x_{p-4}\ldots$ x_0 $1/2 > f \ge 1/4$, (4)
$.001x_{p-3}x_{p-4}\ldots$ x_0 $1/4 > f \ge 1/8$, (5)
$.0001x_{p-4}\ldots$ x_0 $1/8 > f \ge 1/10$. (6)

Given a fixed floating point word size, the explicit presence of the leading sentinel and non-significant zero bits obviously degrades the error performance of non-binary systems that encode a floating point datum in such fashion. Some authors have noted that a higher radix effectively allows the use of bits from the exponent field in the significand field increasing its precision and error performance. Such a method, however, will generally try to satisfy the equation

$$2^{2^i} = \beta^{2^j} \quad (7)$$

where

$\beta > 2$,
i - binary exponent size (in bits),
j - base β exponent size (in bits),

to maintain a similar range of representation. The number of higher radix exponent bits, j, would sensibly be chosen to be either $i - \lfloor \log_2 \log_2 \beta \rfloor$, or $i - \lceil \log_2 \log_2 \beta \rceil$. For a decimal representation such as the example outlined above, a similar exponent range could be achieved by moving one or two bits from the exponent field to the significand. The decimal significand must obviously still store non-significant leading digits vis-a-vis the binary system. Other researchers have shown that such an accommodation will still yield error performance inferior to that of a binary representation.

We propose in this paper an alternative method of encoding a higher radix floating point datum that demonstrates error behavior similar to that of a conventionally encoded binary representation without requiring additional storage or suffering significant reductions in exponent range. In particular, the case of a base 10 representation will be presented as an example of this technique, and referred to as the Decimal Floating Point (DFP) format. Our method allows that more sophisticated hardware and algorithms may be necessary to process the proposed format.

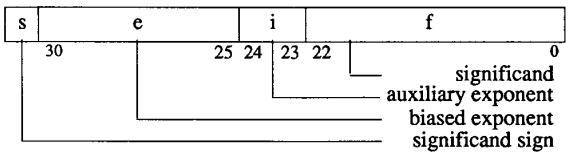

Figure 1. Proposed Decimal Floating Point Format

We begin by examining further the behavior of the decimal significand shown above (3-6). From the perspective of the worst case error, it is observed that four bits are used to define only four different states of the significand. Further, these bits are not used for storage of significant data, but only to hold place value. It is proposed then, that the four possible states of the significand outlined above be encoded as a single two bit field within the floating point word. This field can be described as an alignment field or auxiliary exponent. A proposed partitioning for a single precision floating point word analogous to that specified in the IEEE standard appears in Figure 1.

So defined, the DFP format will represent a floating point datum, X, as

$$X = (-1)^s * f * 10^{e-q} * 2^{i-3}, \tag{8}$$

where q is a bias of 32_{10}. It is also proposed, for purposes of error control, that the DFP system be implemented with the radix to the right of the significand. The digit complementation of the binary auxiliary exponent, i, serves to preserve the integer ordering of floating point words and the value of i is assigned for the possible significand states as follows:

a) $1x_{p-1}x_{p-2}x_{p-3}x_{p-4}\cdots$ x_0 **i=3** $2^p>f\geq 2^{p-1}$, (9)
b) $01x_{p-1}x_{p-2}x_{p-3}x_{p-4}\cdots$ x_0 **i=2** $2^{p-1}>f\geq 2^{p-2}$, (10)
c) $001x_{p-1}x_{p-2}x_{p-3}x_{p-4}\cdots$ x_0 **i=1** $2^{p-2}>f\geq 2^{p-3}$, (11)
d) $0001x_{p-1}x_{p-2}x_{p-3}x_{p-4}\cdots$ x_0 **i=0** $2^{p-3}>f\geq 2^{p-3}/10$. (12)

Since the binary exponent field scales the significand by less than the base, the useful integer ordering (a)>(b)>(c)>(d) is preserved. The size of the binary exponent also allows the DFP format to carry a full 24 bits of precision within the DFP encoding. Because the place value of the lsb is dependent on the auxiliary exponent, i, the value of the "unit of the last place" [21] varies based upon the value of i. This necessarily complicates the error analysis of the DFP format, however, we demonstrate below results comparable to an equivalent binary representation.

Non-binary radices other than decimal can also use this technique by noting that an auxiliary exponent may be created for any base by reserving $\log_2 \log_2 \beta$ bits of the exponent for this purpose. Base 10, besides its obvious interest, is particularly amenable to this technique in that it gives up little exponent range in its implementation.

Performance

In this section the performance of the DFP system is measured by the traditional analyses of floating point systems. Results from Matula's examination of significance spaces [25, 26], Brown and Richman's consideration of the chosen base and exponent range [6], McKeeman's relative error analysis [27], and Brent's RMS error criteria [5] are extended to the DFP format. In each case the performance exhibited by the DFP system is compared with the expected result for the IEEE binary single precision format.

Significance Space Analysis

We adapt Matula's analyses [25, 26] of significance space density and relative gap between constituents of a significance space to the somewhat more unorthodox format of the DFP system. Use of these results, however, requires a more practical view of the implementation of floating point representations. Specifically, practical systems are assumed to be implemented with binary devices regardless of the logical base of representation. This is the case when implementing the proposed Decimal Floating Point format and Matula's results will, therefore, be adapted to such an encoding.

Matula [26] defines several concepts for evaluating the relative performance of floating point representations of different bases. He uses the term "significance space" to denote the set of representable real numbers for a given base and significand size. Such sets disregard exponent range and are infinite. The density of membership within a finite range, however, is different for each base and significand length considered and a finite membership ratio of two different bases [26] is, therefore, posed as

$$\frac{|S_\delta^n|}{|S_\beta^m|} = \lim_{M\to\infty} \left| \frac{\left\{ d \mid d\in S_\delta^n,\ \frac{1}{M} \leq |d| \leq M \right\}}{\left\{ b \mid b\in S_\beta^n,\ \frac{1}{M} \leq |b| \leq M \right\}} \right|, \tag{9}$$

$$= \lim_{M\to\infty} \frac{2(2\log_\delta M + \varepsilon_1)(\delta-1)\delta^{m-1}}{2(2\log_\beta M + \varepsilon_2)(\beta-1)\beta^{n-1}},\ |\varepsilon_1|, |\varepsilon_2| \leq 2, \tag{10}$$

where S_β^n = Base β, n digit, significance space.

Completing the limit of eq. (10) [26], we see

$$= \frac{(\delta-1)\delta^{m-1}}{(\beta-1)\beta^{n-1}} \log_\delta \beta, \tag{11}$$

but applying this formula to the IEEE format and the DFP format is somewhat problematic in that the DFP system is not defined with a specific number of decimal digits. If one ascribes meaning to a nonintegral number of digits, $\log_{10} 2^{24}$ can be substituted as a reasonable estimate. Evaluating eq. (11), then, produces the disappointing result of .5418.. or a significance space only 54% as dense as that of a comparable binary system. Signifance space density, then, if accepted as a valid measure of the static error properties of a floating point representation, appears to depend on the magnitude of the base as well as the precision of the significand. Binary systems obviously optimize the former criterion.

Equation (11), however, does not address the increased density made possible by the use of an auxiliary exponent. In the proposed system, 24 significant bits of precision are always retained. This has been accomplished by absorbing leading zeros and the implicit msb in the auxiliary exponent. Re-examining then, the derivation of eq. (11) [26], one sees that the terms $(\delta-1)\delta^{m-1}$ and $(\beta-1)\beta^{n-1}$ describe the number of significand states that are assumed over the intervals $(\delta^i,\delta^{i-1}]$ and $(\beta^i,\beta^{i-1}]$. Consider then, encoding a base β system on δ state devices, where $\beta>\delta$, and an auxiliary exponent of adequate size preserves the trailing bits of the significand. The number of significand states for such a system is, then,

$$\lfloor \log_\delta \beta \rfloor\, \delta^{m-1} + \left\lceil \delta^m - \frac{\delta^{m+\lfloor \log_\delta \beta \rfloor}}{\beta} \right\rceil,$$

and, for a base β system imlemented on δ–ary devices, eq. (10) is recast in the form:

$$= \lim_{M\to\infty} \frac{2(2\log_\delta M + \varepsilon_1)(\delta-1)\delta^{m-1}}{2(2\log_\beta M + \varepsilon_2)\left(\lfloor \log_\delta \beta \rfloor\, \delta^{m-1} + \left\lceil \delta^m - \frac{\delta^{m+\lfloor \log_\delta \beta \rfloor}}{\beta} \right\rceil\right)}$$

$$= \frac{\log_\delta \beta\ (\delta-1)}{\lfloor \log_\delta \beta \rfloor + \delta - \delta^{1+\lfloor \log_\delta \beta \rfloor - \log_\delta \beta}}. \qquad (12)$$

Evaluating this result for the DFP format (specifically, δ=2 and β=10), the ratio of significance space density is .977, indicating a more populous distribution of significands under the proposed format. This may initially seem counterintuitive, however, Matula's analyses evaluate the behavior of infinite sets and do not examine the effect on exponent range as do, say, Brown and Richman [6] below. Figure 2 depicts the behavior of eq. (12) with δ=2 and β the independent variable. We consider the unusual behavior of (12) further after adapting Matula's relative gap function to include the DFP format.

Matula's relative gap function describes the relative distance between constituents of a significance space:

$$\Gamma_\beta^n(x) = \frac{\min\{b|b>x, b\in S_\beta^n\} - \max\{b|b\le x, b\in S_\beta^n\}}{x}, \qquad (13)$$

and, when plotted, provides a convenient visual model of the behavior of different significance spaces. To address the DFP format, however, the range of significands must be considered as four separate intervals. This is necessary since the value of the "unit of the last place" [21] differs based on the state of the auxiliary exponent. From the decimal example above, the possible significand states are:

a) $1x_{p-1}x_{p-2}x_{p-3}x_{p-4}\cdots$ x_0 **i=3** $2^p > f \ge 2^{p-1}$, (14)
b) $01x_{p-1}x_{p-2}x_{p-3}x_{p-4}\cdots$ x_0 **i=2** $2^{p-1} > f \ge 2^{p-2}$, (15)
c) $001x_{p-1}x_{p-2}x_{p-3}x_{p-4}\cdots$ x_0 **i=1** $2^{p-2} > f \ge 2^{p-3}$, (16)
d) $0001x_{p-1}x_{p-2}x_{p-3}x_{p-4}\cdots$ x_0 **i=0** $2^{p-3} > f \ge 2^{p-3}/10$, (17)

where the place value of x_0 is: 1 in case a), 1/2 in case b), 1/4 in case c), and 1/8 in case d). The *relative* magnitude of x_0, however, remains a constant 2^{-p}. The relative gap function for the DFP format, then, is a sawtooth function similar to that of a comparable binary system. The discontinuities occur, here, at 10^x, $10^x/2$, $10^x/4$, and $10^x/8$. Figure 3 depicts this behavior for a p bit DFP system and a comparable binary system on a log-log scale.

Figures 2 and 3 possess several interesting features. It becomes obvious that the relative gap functions for p bit binary and DFP systems, while exhibiting different periods, have identical maximum and minimum values. It can also be seen that the relative gap of the DFP system will periodically reach a discontinuity other than at the point of maximum relative error. This minor cycle naturally occurs because a denormalized DFP significand begins at the point $2^p/10$ instead of at an integral power of two. This discontinuity somewhat obviates the earlier result with respect to significance space density in that the "average" relative

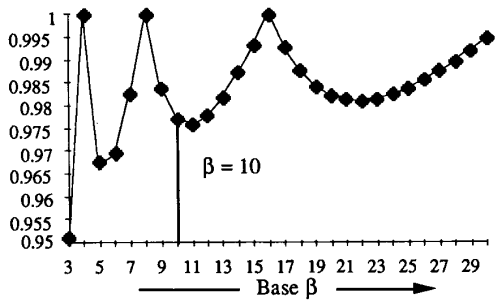

Figure 2. Significance Space Density Ratio – eq. (8), δ=2

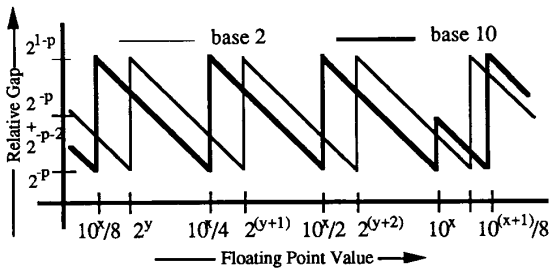

Figure 3. Relative Gap of DFP and Binary Representations

distance between elements of the DFP is system is a bit less than a comparable binary system.

The minor cycle in the gap function of the DFP system can also be used to further elaborate on the phenomena observed in Figure 2. Figure 2 has two curious features. The first of these is the periodic behavior exhibited as the base approaches a power of two. From Figure 3, it is apparent that the distance from the maximum or minimum point at which a minor cycle occurs is determined by the proximity of the base to an integral power of two. Bases slightly less than a power of two will have a minor cycle almost as large as a major cycle, and, therefore, will exhibit average error performance nearly equal to that of a binary system. Similarly, bases slightly greater than a power of two will have a minor cycle of negligible size and will also have average error performance comparable to a binary system.

The second feature of interest in Figure 2 is the apparent trend towards diminished utility for higher radices. Figure 3 demonstrates graphically that a higher radix will insert minor cycles into the gap function less frequently and diminish their effect on the average error performance of the representation. Therefore, the minor cycle interposed in the gap function in systems using an auxiliary exponent has the beneficial effect of reducing the average relative error.

We infer from these results, then, that the utility of systems employing an auxiliary exponent is diminished in systems with large radices or systems in which the radix is in close proximity to an integral power of the base implementation.

Exponent Range

Brown and Richman [6] examine the choice of floating point base by considering the exponent range available to systems of similar accuracy and word size. Our nomenclature is temporarily adjusted to match the authors', *viz.*,

- N – floating point word size, exclusive of sign bits
- r – number of storage device states on which a floating point word is encoded
- β – the base of representation
- q – the number of r state devices used to encode the significand
- i – $\log_r \beta$, where $\beta = r^i$.

The authors' use of the term mantissa is also acknowledged as synonymous with the more contemporary term, significand.

The accuracy of floating point systems is measured by Brown and Richman with a maximum relative error criteria. In a q device base $\beta = r^i$ system, the error function, ε, can be represented by either a maximum relative gap criteria, or a maximum relative error function. Both exhibit proportional behavior (relative gap is generally twice the relative error), therefore, either concept can be employed.

Exponent range, E(q,i), is essentially defined by the bits "remaining" in a floating point word after the significand and signs have been allocated. In an N, r–state device base β floating point word with q "digits" of significand (mantissa), the exponent range is

$$E(q,i) = i\,(r^{N-q} - 1), \tag{18}$$

allowing exponents in the range $r^{-E(q,i)}$ to $r^{E(q,i)}$. Sensible designs will assure q≥i.

By definition, floating point systems of the form (q,r^i) and $(q-i+1,r)$ will have comparable maximum relative error (or gap) since an r^i representation will potentially have i–1 leading zeros and require the same number of additional significand bits to achieve a comparable worst case error [27]. Exponent range, therefore, will be diminished and can be compared by considering the ratio of the exponent range functions,

$$\frac{E(q-i+1,1)}{E(q,i)} = \frac{r^{i-1}}{i}\left(1+\frac{1-r^{1-i}}{r^{N-q}-1}\right) \approx \frac{r^{i-1}}{i}. \tag{19}$$

Systems with base $\beta = r^i$, where i>1, have a reduced exponent range to achieve accuracy comparable to a system in which β = r. The reduced exponent range is necessarily implied by the fact that there may be $\lfloor \log_r \beta \rfloor$ leading zeros in the significand. This, in turn, requires greater significand size to achieve maximum relative error properties comparable to a smaller base. The extension of the significand must naturally come at the expense of the exponent. Table 1 [6] lists values derived by Brown and Richman by applying eq. (19) to different base and device size combinations.

Adapting Brown and Richman's analyses to the DFP system requires consideration of its unusual format. In general, a string of leading zeros in a significand can be changed from a unary encoding to a more compact r–ary encoding by adding an r–ary auxiliary exponent. Therefore, eq. (18) can be recast such that the storage base r and the exponent base β need not necessarily belong to the same commensurable family, *viz.*,

$$E'(q,\beta) = \log_r\beta\, r^{\lfloor N-q-\log_r\log_r\beta \rfloor} \approx E(q,i) \tag{20}$$

From the previous examination of Matula's gap function, it's seen that given an equal number of base r significand devices, one can achieve identical maximum and minimum relative gap for different bases if an auxiliary base r exponent exists to account for nonsignificant leading base r "digits" in the significand. The exponent range ratio of eq.(19) can, therefore, be recast as:

$$\frac{E'(q,r)}{E'(q,\beta)} \approx \frac{r^{N-q}}{\log_r\beta\, r^{\lfloor N-q-\log_r\log_r\beta \rfloor}} = r^{\log_r\log_r\beta - \lfloor \log_r\log_r\beta \rfloor}. \tag{21}$$

It therefore follows that

$$1 \le \frac{E'(q,r)}{E'(q,\beta)} \le r, \tag{22}$$

and the somewhat more pessimistic results of Table 1 can be

	r=2 implicit	r=2 explicit	r=3	r=4	r=10
r^0	1	2	1.5	2	5
r^1	1.33	2.67	3	5.33	33.3
r^2	2	4	6.75	16	250
r^3	3.2	6.4	16.2	51.2	2000

Table 1. Brown and Richman's Ratio of Exponent Range

rejected if an appropriate encoding exists and binary devices are employed.

Given, then, the presence of an appropriate auxiliary exponent, as in the DFP format, the impact on exponent range from the use of a higher radix can be limited to a factor of r^{-1} of the ideal. For practical systems (*i.e.* those implemented on binary architectures), the worst case penalty is a 50% decrease in exponent range when compared with a conventional binary floating point representation. In practice, the figure derived from eq. (21) for the DFP system and a comparable binary system is approximately 1.19.., or, the DFP system achieves approximately 84% of the exponent range of a comparable binary system.

Relative Representational Error

In this section, the analyses of McKeeman [27], Cody [9], and Hwang [17] are extended to include the DFP format. Specifically, the Maximum and Average Relative Representational Error (MRRE and ARRE) properties of the Decimal Floating Point System are examined.

To evaluate the MRRE, consider an arbitrary real number x and the floating point representation to which it maps, $x^* = f_x\beta^{e_x}$. The difference between x and x* relative to the magnitude of x gives the relative error in representing x with x*. The MRRE divides the absolute representational error by x to negate the effect of the exponent and to produce a percentage or relative error statistic. Thus stated,

$$\text{MRRE} = \text{Max}\left[\frac{x-x^*}{x}\right]. \tag{23}$$

If x is an arbitrary real number within the range of representation of a floating point system, then the mapping of x is bounded by y* and z*, where $y^* \le x < z^*$, and z* is succ(y*) within the representation. Maximizing the difference between x and x*, and further, assuming the mapping is the best possible, the maximum difference can be approximated by (z*–y*)/2. Similarly, the denominator of (23) can be approximated with y* since the difference is less than 1 unit in the system of representation. Equation (23) can, therefore, be approximated by

$$= \text{Max}\left[\frac{f_{z^*}\beta^{e_{z^*}} - f_{y^*}\beta^{e_{y^*}}}{2\, f_{y^*}\beta^{e_{y^*}}}\right] \tag{24}$$

$$= \text{Max}\left[\frac{((f_{y^*} + 1\text{ "unit of the last place"})\beta^{e_{y^*}}) - f_{y^*}\beta^{e_{y^*}}}{2\, f_{y^*}\beta^{e_{y^*}}}\right] \tag{25}$$

where a "unit of the last place" [21] is the arithmetic difference between two adjacent significands. The definition of the term "unit of the last place" (or ulp) in the numerator derives from that of the significand in the denominator. In a system using n significand states (where n is generally assumed to be some integral power of the base of implementation) and a normalized significand range of $a > f \ge b$, the ulp is defined as a/n. For typical binary cases (*i.e.* $1 > f \ge 1/2$) the ulp is 2^{-p}, where p is the number of bits allocated to the significand.

Maximizing (25) is clearly a function of the minimum value of the significand of the denominator. If a normalized representation in the range $1 > f \ge \beta^{-1}$ is assumed, the denominator of (25) becomes $2\beta^{-1}$. For conventional encodings, then,

$$= \frac{1\text{ ulp }\beta}{2}. \tag{26}$$

If a binary commensurable base [26] is employed (*e.g.* β = 2, 4, 8,

16, *etc.*), and the above outlined normalization applied, the ulp is 2^{-p}, where p is the number of bits allocated to the significand. Conventional binary commensurable representations will, therefore, have an MRRE of $2^{-p-1}\beta = 2^{-p}$. Binary systems that use an implicit msb for the significand obviously have more significand states. The ulp for such systems is 2^{-p-1} and the results above can be reduced by one half.

In the DFP system it is observed that the MRRE can only occur at one of three possible significand states, specifically, when the magnitude of the significand is at a minimum. This corresponds to one of the forms; 1000..., 01000..., 001000..., recognizing that the leading nonsignificant bits are stored in the auxiliary exponent and the place value of the "unit of the last place"[21] is decreased by 1/2 in each case. Applying McKeeman's analysis, it can then be concluded that the MRRE for all these points is 2^{-p-1}, identical to the binary representation.

The ARRE is derived by assuming that significands are distributed with the reciprocal density derived by Hamming [15] and Benford [4] and extended by others [2, 7]. Specifically,

$$ARRE_\beta = \int_{\frac{1}{\beta}}^{1} P(x)\,Q(x)\,dx, \tag{27}$$

$$\text{where}\quad P(x) = \frac{1}{x \operatorname{Ln} \beta}, \quad Q(x) = \frac{1}{4x}. \tag{28, 29}$$

P(x) is the reciprocal probability distribution for significands [15], and Q(x)is the uniform relative error in representing a real number x on the interval $(2^p, 2^p/\beta]$. In statistical terms, (27) evaluates the expected value of $|\delta|$, the magnitude of the relative error. To derive this measure for the DFP system, it is necessary to again divide the range of the significand, $(2^p, 2^p/10]$, into four subintervals corresponding to each of the auxiliary exponent states. The ARRE for each of the intervals $(2^p, 2^p/2]$, $(2^p/2, 2^p/4]$, $(2^p/4, 2^p/8]$, and $(2^p/8, 2^p/10]$ is examined separately. The ARRE for the DFP system, then, can be described as:

$$\int_{\frac{2^p}{10}}^{2^{p-3}} \frac{1}{x \operatorname{Ln}\beta}\,\frac{1}{32x}\,dx + \int_{2^{p-3}}^{2^{p-2}} \frac{1}{x \operatorname{Ln}\beta}\,\frac{1}{16x}\,dx + \tag{30}$$

$$\int_{2^{p-2}}^{2^{p-1}} \frac{1}{x \operatorname{Ln}\beta}\,\frac{1}{8x}\,dx + \int_{2^{p-1}}^{2^{p}} \frac{1}{x \operatorname{Ln}\beta}\,\frac{1}{4x}\,dx$$

$$= \frac{13}{2^{(p+4)} \operatorname{Ln}\beta}. \tag{31}$$

The uniform error on each subinterval is decreased by a factor of two by virtue of the auxiliary exponent. Substituting β=10 in eq. (31), and evaluating (27) for a binary system, a ratio of the ARRE can be calculated for the DFP and binary systems,

$$\frac{ARRE_{DFP}}{ARRE_2} = \frac{\frac{13}{2^{(p+4)} \operatorname{Ln} 10}}{\frac{1}{2^{(p+2)} \operatorname{Ln} 2}} = \frac{13}{4} \operatorname{Log}_{10} 2 \approx .978.$$

Therefore, by the Average Relative Representational Error criteria, the Decimal Floating Point format appears to represent an improvement over the best forms of binary representation.

Relative Variance

Brent's analyses of floating point representations [5] are made relative to a logarithmic system of similar precision. He develops two statistics, both ratios of error in a floating point system over that of an equivalent log system. The first of these is simply a ratio of maximum relative error. He shows that the maximum relative error in the log system is the theoretical minimum that can be achieved by any floating point system. It can be noted, however, that the DFP system has been shown earlier to have worst case behavior comparable to the best binary systems.

The second statistic advocated by Brent is motivated by the observation that the MRRE is a pessimistic measure of the representational error inherent in a given representation, i.e. representational errors are generally tend to cancel one another out in a computational process. He presents several simulations that lend credibility to this argument. Brent prefers to consider a root–mean–square (RMS) measure of error in floating point representations as a basis of comparison. Again, Brent evaluates the RMS error against that of the log system. The log system, however, provides no theoretical minimum for the RMS error, as was seen for maximum relative error. Therefore, results based upon an RMS type of analysis are most useful in direct comparisons of representations employing different bases and encodings.

Brent (and Kaneko and Liu [22]) use the distribution for the uniform error and the reciprocal distribution of significands to develop a distribution function for the relative error. We prefer to calculate the second moment of the magnitude of the relative error directly since it closely parallels the development of the average relative error statistic above (27). It can be assumed that δ, the relative error, is uniformly distributed with mean (approximately) 0. The root function will be ignored since it is a ratio of statistics for the DFP and the equivalent binary system which is of interest. The second moment (or mean–square) can be defined for the DFP system as

$$\int_{\frac{2^p}{10}}^{2^{p-3}} P(x)\,(Q''(x))^2\,dx + \int_{2^{p-3}}^{2^{p-2}} P(x)\,(Q''(x))^2\,dx + \tag{32}$$

$$\int_{2^{p-2}}^{2^{p-1}} P(x)\,(Q'(x))^2\,dx + \int_{2^{p-1}}^{2^{p}} P(x)\,(Q(x))^2\,dx,$$

$$\text{where}\quad (Q(x))^2 = \frac{1}{2^4 x^2}, \quad (Q'(x))^2 = \frac{1}{2^6 x^2},$$

$$(Q''(x))^2 = \frac{1}{2^8 x^2}, \quad (Q'''(x))^2 = \frac{1}{2^{10} x^2},$$

$$= \frac{153}{2^{2p+9} \operatorname{Ln}\beta} \tag{33}$$

and P(x) is the reciprocal density function described above (28). It can be noted that this technique of calculating the second moment of the relative error will differ from Brent's by only a constant factor. A direct comparison of bases will therefore remove this bias.

For more traditional floating point formats (*i.e.* without an auxiliary exponent for higher radices), the second moment can be derived as

$$\int_{\frac{1}{\beta}}^{1} \frac{1}{x \ln \beta}\,\frac{1}{2^{2p+4}x^2} = \frac{\beta^2 - 1}{2^{2p+5} \ln \beta}. \tag{34}$$

Comparing the DFP format to the traditional binary format for equal precision,

$$\frac{\delta_{ms}(dfp)}{\delta_{ms}(2)} = \frac{\frac{153}{2^{2p+9} \ln 10}}{\frac{3}{2^{2p+5} \ln 2}} = \frac{153 \ln 2}{16 \ln 10} \approx .95, \tag{35}$$

an analysis similar to Brent's shows the proposed DFP format will again appear to outperform the traditional binary implementation.

Implementation

In our decision to employ a higher radix floating point representation, such as the DFP format, we have allowed that additional hardware and modifications to traditional algorithms will be necessary. In this section we outline a representative ALU implementation capable of processing DFP encoded data.

Adding the ability to process DFP encoded data to a floating point ALU requires the addition of several data translation functions and arithmetic operators. We propose that DFP encoded data be processed in an unpacked format similar to that of conventional binary floating point data. The auxiliary exponent is removed and the leading bits of a DFP datum are restored as it enters the floating point processing unit and a converse operation is performed when a datum exits the floating point ALU. Figure 4 depicts the proposed internal representation. This format is similar to the decimal significand format proposed by Fenwick [12]. To preserve the error control properties of the DFP format, all arithmetic operations are carried out to p+4 bits of precision (for p bits of DFP significand) to accommodate the potential leading zeros and the msb of a decimal significand.

With a few exceptions, arithmetic processing of the decimal significand is identical to that of a binary equivalent. All the normal binary algorithms (addition, subtraction, multiplication, division, complementation, and ordering) are applicable to the decimal significand data. Prealignment of operands (for addition and subtraction) and postalignment of arithmetic results (rounding and normalization) will require that a decimal significand (or other higher radix, if such is employed) be scaled by the base. The scaling of a significand by a base other than binary, however, requires the addition of data path operators that will multiply and divide a significand by the base. Similarly, the normalization threshold will require a more sophisticated algorithm for detection, since the leading digits of the significand will not be in integral form.

Figure 6 depicts a more detailed organization of a representative floating point arithmetic unit, a simplified version of the Analog Devices ADSP-3212. Adding the DFP operators described above, it is proposed that a floating point ALU similar to that depicted in Figure 6 be considered as a possible implementation of the DFP system. The design of each of the architectural revisions is discussed separately below.

Data Path Translation

Packing and unpacking a DFP datum is straightforward and can be accomplished by conventional encode/decode types of circuitry as shown in Figures 5 and 7. An anomaly of the internal format proposed is that it is possible for a significand stored in an ALU register to have more precision than can be stored in memory. There are two possible methods of rounding to the allowed precision of external memory. In the architecture depicted in Figure 5, a floating point word with excess precision (*i.e.* with more than p significant bits) is simply rounded as it leaves the ALU. This will mean a possible trip through the normalization logic to assure a properly encoded result, however, many ALU designs assure that this is the case for any datum exported.

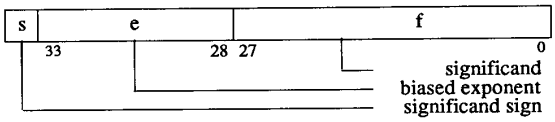

Figure 4. DFP Internal ALU Format

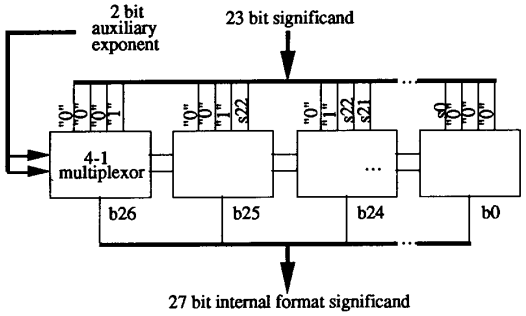

Figure 5. Decimal Floating Point Unpack Operation

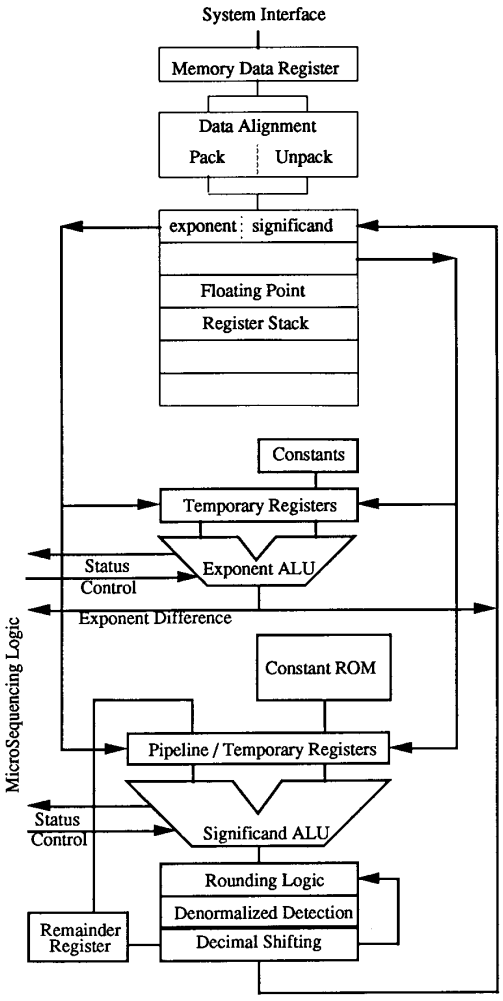

Figure 6. Representative ALU Architecture

An alternative method of rounding is to sacrifice any extra precision after each arithmetic operation by selecting the round, guard, and sticky bits (commonly required by rounding algorithms) on the basis of the leading number of nonsignificant bits. Since this sacrifices precision already available, the method described above is preferred here. System designers will undoubtedly recognize that the preponderance of transfers between the ALU and memory (or the CPU) will involve the simpler unpack operation.

Decimal Base Scaling

Shifting a decimal significand, or any binary incommensurable significand, by the base is more difficult than in a comparable binary oriented system in that simple physical displacement of "digits" must be replaced by a combinatorial function on a binary encoded word. This disadvantage is offset by the relatively lower frequency with which such shift operations must be invoked [36] for higher radices, however, it is arguable that this single limitation has prevented serious prior consideration of non–binary floating point bases. Fortunately, several researchers [1,12,18,29,34] have described techniques by which non–binary scaling of an integer significand can be accomplished in substantially less time and circuit complexity than had previously been achieved. Raghuram [31] presents an excellent summary of these methodologies.

Scaling a significand by base 10 effectively means an integer multiplication or division of the significand by 10. Multiplication by 10 is straightforward and can be decomposed into f*10 = f*2 + f*8. Since a binary shift operation can be performed in the data path, the multiplication of a significand by 10 can be accomplished in time and complexity equivalent to a conventional binary addition. With appropriate lookahead logic, such an addition can be implemented in O(logN) time.

Division by base 10 is a more interesting problem. One cannot distribute the denominator as in multiplication, i.e. $f/10 \neq f/8 + f/2$. However, the problem in this case can be broken down into division of the form f/10 = f/2 * f/5, demonstrating that the difficulty inherent in decimal scaling is a division by five.

Petry et al [18], Petry [29], and Fenwick [12] have all proposed methods of modified subtraction to perform quinary division. The division of an arbitrary significand X by five can be broken down to the form

$$X = 4Q + Q + R, \tag{28}$$

where R is the remainder and Q is the desired result. More graphically, the majority of attempts at implementing specialized division operators begin by decomposing the binary multiplication process that forms the dividend X,

$$\begin{array}{rl} & Q_nQ_{n-1}\cdots Q_4Q_3Q_2Q_1Q_0 \\ * & \qquad\qquad\qquad 1\ 0\ 1 \\ \hline & Q_nQ_{n-1}\cdots Q_4Q_3Q_2Q_1Q_0 \\ + & Q_nQ_{n-1}\cdots Q_4Q_3Q_2Q_1Q_0 \\ \hline & \qquad\qquad\qquad \cdots X_4X_3X_2X_1X_0 \end{array} \tag{29}$$

Both the Petry [18] and the Schreiber [34] algorithms attempt to extract the quotient from the dividend by noting that portions of the of the quotient will be "exposed" at either end of the dividend and the complete quotient may be obtained by unraveling the addition of the partial products, from the left in Schreiber algorithm, and from the right in the Petry algorithm. Fenwick [12] proposes a similar technique consuming larger portions of the dividend at each step. All of these techniques have generally suffered from the bit serial nature of the algorithms, limiting them to O(N) performance.

More recently, Petry and Raghuram [30] have shown extensions to the Schreiber algorithm which decouple the development of each of the quotient "digits" and allows conventional look-ahead techniques to be employed. While the necessary hardware will exceed that required to perform multiplication by the base, it appears that some constant divisions (including quinary) can be accomplished in O(logN) time.

Significand Threshold Detection

Detection of denormalized significands is somewhat more problematic for representations in which the significand digits are not integrally encoded. This is obviously the case for the DFP format. The smallest DFP significand that can be represented is $2^P/10$ = 00011001100... It is recognized, therefore, that such detection will require a subtraction of the minimum allowable value that a normalized significand may assume. The construction of hardware to perform this task is, however, simpler than that of a full subtraction since the subtrahend is a constant and the borrow from the most significant bit is the only result of interest. In effect, only the borrow lookahead logic is required. Figure 8 depicts a representative implementation where B is an ALU internal format significand and K is the normalization threshold.

Carry or borrow lookahead is typically implemented by assessing the likelihood that bit pairs (of the addend/augend and minuend/subtrahend) of equal place value will generate or propagate borrows (carries). In subtraction, bit pairs of the form 0–0 and 1–1 will obviously propagate borrows and a bit pair of the form 0–1 will generate a borrow. Discrete signals describing these states are generally aggregated for groups of 4 to 8 bits to form group propagate and generate functions. Similarly, these functions can be collected to generate a word borrow function. The necessary time to evaluate borrow generation is reduced, then, from a linear time function to a log time function. Typically, the

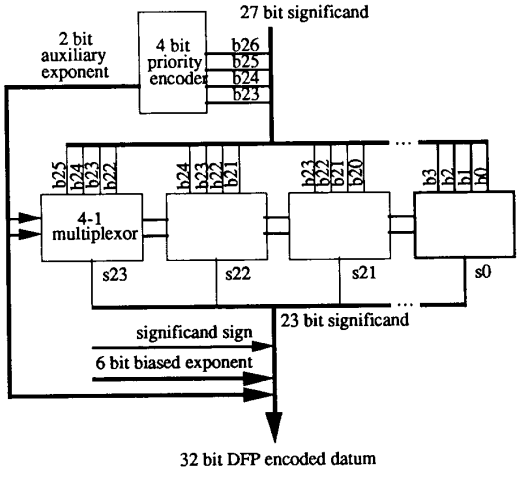

Figure 7. Decimal Floating Point Pack Operation

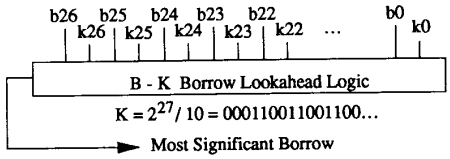

Figure 8. Normalization Threshold Detection

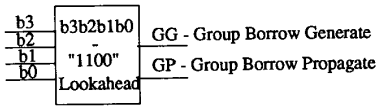

Figure 9. Threshold Detection Borrow Lookahead Term

bit, group, and word propagate and generate functions are used to generate borrows for each bit subtraction stage. In this instance, only the word generate function is of interest inasmuch as the result of the subtraction is of little use. It is, therefore, only necessary to implement the lookahead function.

The regular nature of the normalization threshold (*viz.* ...1100...) make it straightforward to create a logic element for generating the borrow generate and propagate functions a nibble at a time. Figure 9 depicts a sample of such a function. The logic cell in Figure 9 can be implemented with the functions

$$GP = b_3 * b_2 * \neg b_1 * \neg b_0 \quad (30)$$

$$GG = \neg b_3 + \neg b_2 \quad (31)$$

Conventional borrow lookahead logic (i.e, the conjunction of propagate terms and disjunction of generate terms) can be used to collect the group generate and propagate functions.

Conclusions

The choice of floating point base is probably one of the oldest and best researched areas in the study of practical computation. It has traditionally been concerned, more or less, with the engineering of representations that can be implemented with minimal cost and greatest accuracy. Cost, in this case, has been associated with the size of stored data and the complexity of the algorithms that manipulate them. Research in this area has resulted in well understood methods of implementing floating point data and algorithms that have endured very persistently for the better part of twenty years. Little else in the evolution of computer systems has remained nearly so constant.

The implementation of computer systems and the manner in which they are employed, however, has obviously changed significantly in the interval since the conception and initial study of binary floating point representations. Contemporary computers are no longer designed solely on the basis of strict economy of parts or with pure arithmetic bandwidth as the measure of their utility. In essence, we find that the criteria on which the advocacy of binary floating point methods has been built are simply no longer as substantial as they were twenty years ago. Do contemporary computer systems, in fact, have number representation requirements that go beyond the scope of traditional binary floating point representations? The persistence of decimal based calculation in business applications and the growth of non-binary representations in popular spreadsheet and database programs generally offer to support the argument in favor of non-binary floating point alternatives.

In this paper we have considered then a method of encoding non-binary floating point radices that retains many of the advantages of the simpler binary methods. In the case of decimal representation, the proposed method produces a system which requires no additional storage to encode a floating point datum, achieves slightly better accuracy by most established methods of static analysis, and suffers only a small reduction in exponent range. Further, the method employed is easily extensible to other potentially interesting radices and a straightforward method for implementation has been outlined. We conclude, therefore, that the format proposed represents a practical and useful model on which future floating point implementations might successfully be constructed.

References

[1] Artzy, E., Hinds, J.A., Saal,H.J. *A Fast Division Technique for Constant Divisors.* CACM, vol. 19, no. 2, (January 1976), 98–101

[2] Barlow, J. *On the Distribution of Accumulated Roundoff Error in Floating Point Arithmetic.* Proc. of the 5th Symp on Comp. Arithmetic, Ann Arbor, MI, USA, (19 May 1981), New York: IEEE 1981, 100–105,

[3] Bellamy, L.R. *Base 10 Division of Binary Numbers.* IBM Tech. Disc. Bul., vol. 27, no. 4b, (Sept. 1984), 240–243

[4] Benford, F. *The Law of Anomalous Numbers.* Proc. American Phil. Soc., vol. 78, (1938), 551–572.

[5] Brent, R.P. *On the Precision Attainable with Various Floating–point Number Systems.* IEEE Trans. Comput, vol. C–22, no. 6, (Jan 1973), 601–607

[6] Brown, W.S., Richman,P.L. *The Choice of Base,* CACM, vol. 12, no. 10, (Sept 1969), 560–561.

[7] Bustoz, J.A., et al. *Improved Trailing Digit Estimates Applied to Optimal Computer Arithmetic.* JACM, vol. 26, no. 4, (1979), 716–30.

[8] Cody, W.J. *Analysis of Proposals for the Floating–point Standard.* Computer, vol. 14, no. 3, (Jan 1981), 51–62.

[9] Cody, W.J. *Static and Dynamic Numeric Characteristics of Floating Point Arithmetic.* IEEE Trans. Comput., vol C–22, no. 6, (June 1973), 598–601.

[10] Cody, W.J. *A Proposed Radix and Word Length Independent Standard for Floating–point Arithmetic.* SIGNUM Newsletter, vol. 20, no. 1, (Jan. 1985), 37–51.

[11] Coonen, J.T. *An Implementation Guide to a Proposed Standard for Floating–Point Arithmetic.* Computer, vol.13, no. 1, (Jan 1980), 69–79.

[12] Fenwick, P.M. *A Binary Representation for Decimal Numbers.* Aust Comput J., vol.4, no. 4, (Nov. 1972), 146–149.

[13] Gerrity, G.W. *Computer Representation of Real Numbers.* IEEE Trans. Comput., vol.C–31, no. 8, (Aug. 1982), 709–714.

[14] Goldberg, I.B. *27 bits are not enough for 8–digit Accuracy.* CACM, vol. 10, no. 2, (Feb. 1967), 105–106.

[15] Hamming, R.W. *On the Distribution of Numbers.* Bell Sys. Tech. Jour., vol 49, (Oct 1970), 1609–1626.

[16] Ho, R.L. *Approximating Divsions By a Constant.* IBM Tech. Disclosure Bull., vol. 22, no. 4, (Sept. 1979), 1554–1557.

[17] Hwang, K. *Computer Arithmetic: Principles, Architecture, and Design.* Chichester, England: Wiley, 1979, xiii+423.

[18] Johannes, J.D., Pegden,C.D., Petry,F.E. *Decimal Shifting for an Exact Floating Point Representation.* Comput & Electr. Eng., vol. 7, no. 3, (Sept. 1980), 149–155.

[19] Johnstone, P. *Representational Error in Binary and Decimal Numbering Systems.* Proceedings of the 20th SE Regional ACM Conference, (April 1 1982), Knoxville, Tenn., 85–88.

[20] Johnstone, P. *Decimal Floating Point Representation,* Ph.D. Dissertation, Tulane University, May 1988

[21] Kahan, W. *"What is the best base for floating point? Is binary best?"* Dept. of Computer Science lecture notes, UC Berkeley, Dec 1970.

[22] Kaneko, T. and Liu, B. *On the Local Roundoff Error in Floating Point Arithmetic.* JACM, vol. 20, no. 7, (July 1973), 391–398.

[23] Knuth, D.E. *The Art of Computer Programming, Vol. 2.* New York:Addision–Wesley, 1969

[24] Kuki, H. and Cody, W. *A Statistical Study of the Accuracy of Floating Point Number Systems.* CACM, vol. 16, no. 4, (April 1973), 223–230.

[25] Matula, D.W. *In and Out Conversions.* CACM, vol 11, no. 1, (Jan. 1968), 47–50.

[26] Matula, D.W. *A Formalization of Floating Point Numeric Base Conversion.* IEEE Trans. Comput., vol. C–19, no. 8, (Aug. 1970), 681–692.

[27] McKeeman, W.M. *Representation Error for Real Numbers in Binary Computer Arithmetic.* IEEE Trans. on Electron. Comput., vol. EC–16, (Oct. 1967), 582–583

[28] Olver, F.W.J. *Error Bounds for Arith. Operations on Computers without Guard Digits.* IMA J. Numer Anal., vol. 3, no. 2, (April 1983), 153–160

[29] Petry, F.E. *Two's Complement Extension of a Parallel Binary Divsion by Ten.* Electronics Letters, vol. 19, no. 18, (September 1983), pp 718–720

[30] Petry, F.E. and Raghuram, P. *Division Techniques for Integers of the Form $2^n \pm 1$.* Submitted to IEEE Trans. on Computers

[31] Raghuram, P. *Generalized Approaches to Constant Division.* Ph.D. Dissertation, Tulane University, Dec. 1988.

[32] Raimi, R.A. *On the Distribution of First Significant Digits.* Amer. Math. Monthly, vol. 74, no. 2, (Feb. 1969), 342–348.

[33] Salomon, D. *Two Generalized Floating Point Representations.* Byte, vol. 10, no. 9, (Sept. 1985), 154–158

[34] Schreiber, F., and Stefanelli, R. *Two Methods for Fast Integer Binary – BCD Conversion.* Proceedings of the 4th Symposium on Computer Arithmetic, Santa Monica, CA, (June 1975), 200–207,

[35] Sites, R.L. *Serial Binary Division by Ten.* IEEE Trans. on Comput., vol. C–23, no. 12, (Dec. 1974), 1299–1301

[36] Sweeney, D.W. *An Analysis of Floating Point Addition.* IBM Systems Journal, vol. 4., no. 1, (1965), 31-42