Invited Paper

Data Length Independent Real Number Representation Based on Double Exponential Cut

HOZUMI HAMADA*

A new internal representation is proposed for real numbers. It has been named URR for Universal Representation of Real numbers. This approach is based on a bisection method which is applied to real number intervals. With this method, the point of division increases or decreases in a double exponential manner in the global range.

The main characteristics of the method are as follows. First, overflow/underflow does not, in practice, occur. Second, since the data format does not depend on the length but on the value of the data, a transformation operation is virtually not needed between systems of long and short data. Finally, only one bit of resolution is lost compared with the fixed point form. In addition, arithmetic operations are slightly complicated compared with conventional representation, but they present no special difficulties.

This new method is thus the most suitable internal form as an interface not only between computers but also between computers and digital systems which deal with real numbers or physical (scalar) values.

1. Introduction

Some important shortcomings in the floating point form of internal representation for real values in computers have been pointed out. For instance, overflows and underflows cause problems for programmers. In addition, because of the variety of data format standards, there is no compatibility between computers having differing data formats.

W. Kahan and J. Palmer[3] have proposed one format as an IEEE standard to avoid the latter problem. However, their format does not conclusively solve the first problem.

R. Morris[2] has proposed a system which ensures data precision by shortening the length of the exponent part when the absolute value of the exponent is small. S. Matsui and M. Iri[5] have extended this concept. They proposed a system whereby overflow/underflow does not occur in practice. Their method extends the exponent to the entire data word, if necessary, using an additional field that specifies the length of the exponent part. The method has its significant merits, but the conversion from one type of data to another is quite complicated if the lengths are different. This paper gives an improved description of a new system called URR(for Universal Representation of Real numbers), originally reported in[6], which is free from all the drawbacks described above.

*Central Research Laboratory, Hitachi Ltd., Kokubunji, Tokyo 185, Japan.

2. Motivation

Consider a program in Pascal, which is supposed to calculate the absolute value of a complex number. Suppose that type identifier complex is defined as follows:

type complex=**record** re, im:real **end.**

A first cut might be:

function absolute (z: complex):real;
begin absolute:=sqrt(sqr(z.re)+sqr(z.im)) **end.**

However, this program is unacceptable in conventional computers, because an overflow or underflow may occur. Thus, for example, if the real part z.re of z, is the order of magnitude 10^{40}, the square of z.re overflows in IBM S/360[4] or equivalent, even though the result can be represented. To prevent an overflow, the program must be modified as follows:

function absolute (z: complex):real;
begin if abs(z.re)>abs(z.im)
  then absolute: =abs(z.re)*sqrt(1+sqr(z.im/z.re))
  else absolute: =abs(z.im)*sqrt(1+sqr(z.re/z.im))
end.

This program is, even at the expense of the undesirable divisions, still unacceptable because either z.im/z.re, z.re/z.im, or their square might underflow. One acceptable program is as follows:

function absolute(z: complex):real;
  const c=4096;
  var b: Boolean; x, y: real;
begin if abs(z.re)>abs(z.im)
  then begin x:=abs(z.re); y:=abs(z.im) **end**
  else begin x:=abs(z.im); y:=abs(z.re) **end;**

if $x<1$ **then** $b:=x<c*y$ **else** $b:=x/c<y$;
if b **then** absolute:=x*sqrt(1+sqr(y/x))
else absolute:=x **end**.

The constant c in this program is a special number such that, if x/y is greater than c, the $1+\mathrm{sqr}(y/x)$ can not be distinguished from 1 in single-precision floating-point representation.

In conventional computers, the program must be this complicated. In contrast, the first simple program would be acceptable, if an overflow/underflow-free computer existed. Programming would become easier if we could use such a computer.

3. Design Criteria

In a system of the representation of real numbers, the following three conditions appears to be mandatory:

—It should be quite easy to convert from the exponent and mantissa as binary numbers to this form, and vice versa. This condition means that complicated means, such as logarithmic functions, should not be used.
—Overflows and underflows should rarely, if at all, occur.
—Format specifications shoul not depend on the data length.

Additionally, the following three conditions are desirable:

—The format should be as "natural" as possible. Arbitrary arrangements should be avoided.
—Every bit pattern should correspond to a different value.
—The order of data in this format should be the same as the order in the fixed point form.

4. Formal Definition

The following notations and procedures are provided for definition of URR. An arbitrary bit string, S corresponds to the semi-closed real number interval. That is,

$$S: \{x \mid a \leqq x < b\}.$$

If $U(S)$ is an arbitrary value in the interval corresponding to bit string S, the following relationship holds,

$$a \leqq U(S) < b.$$

Let $S0$ denote a bit string S followed by a "0" bit. Likewise, $S1$ is a bit string S followed by a "1" bit. The interval is then divided by a third value, c, which is determined by the form of S and the value of a and of b. Using c, we have:

$$a \leqq U(S0) < c \quad \text{and} \quad c \leqq U(S1) < b.$$

Let 0^k and 1^k denote a concatenation of k "0" bits and k "1" bits, respectively. Intervals are then cut by the following four steps.

(a) Rough cut
First, these relationships arise:

$$-\infty \leqq U(1) < 0 \quad \text{and}$$
$$0 \leqq U(0) < +\infty.$$

Here, the equality and the inequality to the infinite terms are given to assure formal uniformity. If these intervals are partitioned at -1 and 1, then, the following relationships can be derived.

$$-\infty \leqq U(10) < -1,$$
$$-1 \leqq U(11) < 0,$$
$$0 \leqq U(00) < 1 \quad \text{and}$$
$$1 \leqq U(01) < +\infty.$$

Moreover, if these intervals are further partitioned at -2, -0.5, 0.5 and 2, then the following relationships occur.

$$-\infty \leqq U(100) < -2,$$
$$-2 \leqq U(101) < -1,$$
$$-1 \leqq U(110) < -0.5,$$
$$-0.5 \leqq U(111) < 0,$$
$$0 \leqq U(000) < 0.5,$$
$$0.5 \leqq U(001) < 1,$$
$$1 \leqq U(010) < 2 \quad \text{and}$$
$$2 \leqq U(011) < +\infty.$$

In these eight relationships, the procedure progresses to step(b) for those in which the 2nd bit is equal to the 3rd bit(counting from left to right). Otherwise, it proceeds directly to an equal-difference cut(step(d)).

(b) Double-exponential cut
Let $p^+(m)$ and $p^-(m)$ denote 2^{2^m} and 2^{-2^m}, respectively. If $m \geqq 0$, the interval is partitioned recursively, as follows.

$$-\infty \leqq U(10^{m+2}) < -p^+(m) \quad \text{at } -p^+(m+1),$$
$$-p^-(m) \leqq U(11^{m+2}) < 0 \quad \text{at } -p^-(m+1),$$
$$0 \leqq U(00^{m+2}) < +p^-(m) \quad \text{at } +p^-(m+1)$$
and
$$+p^+(m) \leqq U(01^{m+2}) < +\infty \quad \text{at } +p^+(m+1).$$

Then, the following relationships are arrived at, where the run of 0s or 1s is broken from the 2nd bit to the right.

$$-p^+(m+1) \leqq U(10^{m+2}1) < -p^+(m),$$
$$-p^-(m) \leqq U(11^{m+2}0) < -p^-(m+1),$$
$$+p^-(m+1) \leqq U(00^{m+2}1) < +p^-(m) \quad \text{and}$$
$$+p^+(m) \leqq U(01^{m+2}0) < +p^+(m+1).$$

(c) Equal-ratio cut
The following cut is performed m times in the above case. If the relationship is $a \leqq U(S) < b$, then each interval is divided at $a\sqrt{b/a}$. The following relation-

ships are then obtained:

$$a \leqq U(S0) < a\sqrt{b/a} \quad \text{and} \quad a\sqrt{b/a} \leqq U(S1) < b.$$

(d) Equal-difference cut

The following cut is performed an arbitrary number of times. Each interval is divided at $(a+b)/2$ under the same conditions as above. The following relationships are consequently obtained:

$$a \leqq U(S0) < (a+b)/2 \quad \text{and} \quad (a+b)/2 \leqq U(S1) < b.$$

Since these processes show the generating rule for a bit string, they do not necessarily present the entire process of cutting. It is permissible to stop cutting when the desired bit string is arrived at. In addition, since reduction of the interval is performed by a non-fixed of cuts, it is obvious that $U(S0^k)$ at $k \to \infty$ converges to the lower limit of the coresponding interval. This is denoted by $V(S)$. That is, if the relationship is $a \leqq U(S) < b$, then

$$V(S) = \lim_{k \to \infty} U(S0^k) = a.$$

For example the representation of 37 using this method is as follows.

—rough cut

$$2 \leqq U(011) < +\infty,$$

—double-exponential cut

$$4 \leqq U(0111) < +\infty,$$
$$16 \leqq U(01111) < +\infty,$$
$$16 \leqq U(011110) < 256,$$

—equal-ratio cut

$$16 \leqq U(0111100) < 64,$$
$$32 \leqq U(01111001) < 64,$$

—equal-difference cut

$$32 \leqq U(011110010) < 48,$$
$$32 \leqq U(0111100100) < 40,$$
$$36 \leqq U(01111001001) < 40,$$
$$36 \leqq U(011110010010) < 38,$$
$$37 \leqq U(0111100100101) < 38.$$

Consequently, $V(0111100100101)$ is equal to 37.

5. Conventional Explanation

The format defined above can be understood more conventionally in terms of the exponent and the mantissa parts.

(a) The sign bit

The first bit corresponds to the sign of the number in the first step of the rough cut. Therefore, the first bit is considered to be a sign bit.

(b) Mantissa part

An equal-difference cut starts with the interval for which the ratio of the limits equals 2. The bit string for this process is equal to the bit string in which the left-most redundant bit is removed from the bit string generated by the mantissa value. Thus, the bit string generated by the equal-difference cut may be considered to be the mantissa part.

(c) Exponent part

It is assumed that the exponent part is to the right of the sign bit and to the left of the mantissa part. This is that part of the bit string obtained from the last two steps of the rough cut, the double-exponential cut, and the equal-ratio cut. The exponent part is a variable length. If only the rough cut takes place, the exponent part will be 2 bits long. Otherwise, it is $2m+3$ bits long. If the value x is positive, the relationship between the range of the value, the binary form of the exponent value, e, and the bit pattern of the exponent part are as follows:

⋮	⋮	⋮
$p^-(m+1) \leqq x < p^-(m)$	$1\text{—}10\overset{m}{*\text{—}*}$	$0^{m+2}1\overset{m}{*\text{—}*}$
⋮	⋮	⋮
$p^-(2) \leqq x < p^-(1)$	111—110*	0001*
$p^-(1) \leqq x < p^-(0)$	111—1110	001
$p^-(0) \leqq x < 1$	111—1111	01
$1 \leqq x < p^+(0)$	000—0000	10
$p^+(0) \leqq x < p^+(1)$	000-0001	110
$p^+(1) \leqq x < p^+(2)$	000-001*	1110*
⋮	⋮	⋮
$p^+(m) \leqq x < p^+(m+1)$	$0\text{—}01\overset{m}{*\text{—}*}$	$1^{m+2}0\overset{m}{*\text{—}*}$.
⋮	⋮	⋮

Furthermore, the mantissa value, f, can be expressed as:

$$-2 \leqq f < -1, \quad 1 \leqq f < 2.$$

6. Not-A-Number

Consider 0, ± 0, ∞ or $\pm\infty$ as not-a-number. For n bits per word, the following two are special cases:

$$V(00^{n-1}) = 0 \quad \text{and}$$
$$V(10^{n-1}) = -\infty.$$

The values represented by neighboring cyclic bit patterns are as follows:

$$V(11^{n-2}1) = -p^-(n-3),$$
$$V(00^{n-2}1) = +p^-(n-3),$$
$$V(01^{n-2}1) = +p^+(n-3) \quad \text{and}$$
$$V(10^{n-2}1) = -p^+(n-3).$$

With every system, if should be determined whether to give special meaning to these bit patterns or not.

However, when the patterns are used as quantities for computation, it is convenient for them to be -0, $+0$, $+\infty$ and $-\infty$, respectively. For this reason, infinity is redefined without using a sign as:

$$V(10^{n-1})=\infty.$$

The ± 0 and $\pm\infty$ are symbols for values in the direction of 0 or infinity. They are representations of conventional values that replace an underflow or overflow. However, due to the magnitude in this case, these values virtually do not occur. Note that, $p^{+}(29)$, which represents $+\infty$ for a 32-bit data length, has the power value of 161,614,249, whereas the power value would be 39 in the IEEE standard[4].

7. Evaluation of Errors

Here, characteristics obtained directly from the representation rule will be evaluated through application of the URR method to a fixed data length word. The system of this paper was compared with several typical applications of 64-bit data. (Note that a 64-bit data length is typical with numerical computations) With limited length data, it is impossible to represent all the real values. Errors can occur since real values which bit patterns exactly represent are discrete. The error is the difference between the value intended for the representation, and the value corresponding to the bit pattern. This error is evaluated where the value corresponds to the mid-point between two discrete points.

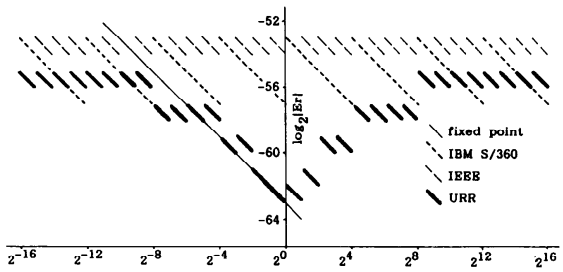

Fig.1 Relative Error(Er), with 64 bits data.

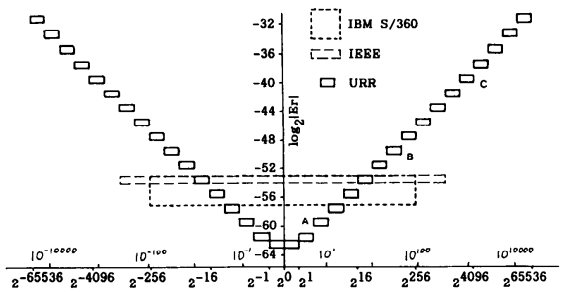

Fig.2 Er over wide range.

Relative error characteristics for several methods are shown in Figure 1. Both axes are logarithmic. Here, at a range of $-2\leqq x<2$, there are errors when an IBM S/360[4] is used, and with the fixed point form. This shows that URR has less precisions in only three cases when compared with the fixed point form. Moreover, the loss of precision is only 1 bit. Fig. 1 can be simplified. The slanted lines standing at the same level can be replaced by the squares which include these lines.

Next, the horizontal axis can be replaced by a double logarithmic scale so as to be able to observe neighbors in infinite and infinitesimal regions. Such a case is shown in Fig. 2. This figure also shows the following three characteristics:

—Region A is a neighbor of ± 1, and URR is superior to the conventional method in regard to precision.
—In region B, URR is not superior.
—In region C, URR can represent real values, but the conventional method cannot.

8. Elementary Operations

For URR, the elementary operations below should be provided. Using a software simulator, it has been verified that these operations are appropriate.

—Changing the sign and obtaining the absolute value
—Addition, subtraction, multiplication and division
—Extracting the exponent value and operation of this item
—Conversion from/to integer type data
—Conversion from/to a decimal value.

The following are specific to this method and its elementary operations. Mathematical elementary functions and the more complicated operations are not explained.

(i) Changing the sign and obtaining the absolute value

Because URR uses 2's complement representation for negative values, the operation for changing the sign is slightly complicated compared with the one using a sign and magnitude. However, because the relation of representation between the original data and its negative is in 2's complement as an integer value, changing the sign can then be performed using the operation for changing the sign of an integer. Obtaining the absolute value is performed using the operation for changing the sign, if the given value is negative.

(ii) Addition, subtraction, multiplication and division

Unfortunately, a method to directly operate with URR data has not yet been found. The exponent part is variable length, so the operation can be simplified by clearly separating the mantissa part. Supposing a data length of n bits, storages holding exponent and mantissa must both be $n-1$ bits long if no information is to be lost. This means that 2 words must be provided to separate exponent and mantissa values contained in one word. The method for separating URR data into expo-

nent and mantissa is as follows. If the leftmost 3 bits are

$$110,\ 101,\ 010 \text{ or } 001,$$

that is, the 2nd and 3rd bits are different, separation may be done specifically. In all other cases, the border of the exponent and the mantissa part is determined by counting the run of 0's or 1's from the 2nd bit to the right. When the count is n, the exponent part is $2n-1$ bits from the 2nd bit to the right. Because the conventional floating point operation does not require this process, the operation speed of this separation is an important consideration.

The operation for the separated exponent and mantissa is performed in a conventional way. The result also consists of the separated exponent and mantissa, which must then be combined into URR form. This process is also slightly complicated compared with the conventional floating point operation. At first, the mantissa value is normalized and the exponent value is adjusted. If the value of the exponent is equal to 0 or -1, the combining process may be done specifically. But for the other cases, the number of exponent bits is counted, which is done by finding the transition point, 0 to 1 or 1 to 0, from the sign bit to the right. In this case, a negative integer is expressed by 2's complement; this needs a similar process as in the case of the system with a sign and magnitude. For either the separating or combining process, the position of the transition from 0 to 1 or 1 to 0, must be determined; it is hoped that special hardware can be provided for performing this determination.

When the operand has a special value such as 0 or infinity, these separating/combining processes can be omitted by pre-examining for such a value. Moreover, even when the given value is none of these, if it is extreme, for instance, far from the exponent value, it can be considered that these processes can also be ommitted.

(iii) Extracting and operating on the exponent value

It is recommended by the IEEE standard regarding floating point representation that a function logb should be provided. This function returns the value n satisfying

$$2^n \leqq |x| < 2^{n+1}$$

by calling logb (x). In the case of IEEE representation which belongs to the system of sign and magnitude, the relation is as above. Thus, in the case of URR, the same function should be fundamentaly provided.

The exponent value is obtained by this function logb, while the mantissa value is obtained by the function scalb, which is described in the following paragraph. Because the mantissa value is obtained simultaneously by hardware with the exponent value, it is recommended that the mantissa value also be returned in a specified register as x. This is described as follows in Pascal.

function logb (**var** x: real):integer;

Another function scalb which is recommended by the IEEE standard is useful, and should also be provided. Function call scalb(x, $-n$) returns the value in URR form, but depending on some conditions, the fixed point form may be required. In such case, we can use the fact that if the value x is in either interval

$$0.25 \leqq |x| \leqq 1,$$

the bit patterns are completely the same between the URR form and fixed point representation. In order for the exponent value to become a round number at large exponent values, the function must return the integral value and the mantissa value such that

$$0.5 < |x| \leqq 1.$$

As the result, the exponent integer which the function returns satisfies the relation

$$2^{n-1} < |x| \leqq 2^n.$$

(iv) Conversion from/to integer type data

The standard method for separating the exponent and mantissa will use the function logb described above, in which the mantissa is converted to an integer value by bit shifting. In addition, there is a simplified method for some situations. An example is given here for the case of a 64-bit data length. A positive value m, whose decimal point is understood to be located at the right-end of its URR representation, is in the range of

$$2^{50} \leqq m < 2^{51}.$$

If a number x satisfies the condition

$$-2^{49} \leqq x < 2^{49},$$

when a constant c is equal to $2^{49}+2^{50}$, the relation

$$2^{50} \leqq x+c < 2^{51}$$

holds, so the decimal point of $x+c$ is always located at the right-end of its representation. In this case, the leftmost 15 bits are,

$$011111111010 1 \text{ at } x \geqq 0, \quad \text{and}$$

$$011111111010 0 \text{ at } x < 0.$$

Because the URR representation of c is

$$011111111010 1 0 \cdots 0,$$

the following algorithm is obtained:

(a) add c is URR form to x

(b) subtract c as an integer from this result.

This algorithm is correct if $-2^{49} \leqq x < 2^{49}$.

The method to convert from integer n to URR is the reverse of the above.

(a) add c as an integer to n

(b) subtract c in URR form from this result.

Thus, this conversion method is valid only if the integer to be converted can be expressed within 50 bits. If not, conversion should be done in accordance with the standard method. This simplified method can also be effectively performed in the conventional representa-

tion.

Hence, URR does not present any disadvantages in this regard.

(v) Conversion from/to a decimal value

Practically speaking, it is impossible to store the values of 10 to all powers such as the conventional method does, because the number of integers to be raised to 10 is by far too many to construct an internal table.

In the case of conversion from decimal to URR, the exponentiation operation is employed as usual. That is, by providing the table of 10^{2^i}, only necessary factors are multiplied. The value of 10 to an arbitrary power is obtained by this method. As the number of multiplications increases, the error increases; however, the result is so distant from 1 that the length of the mantissa is shorter in this case, which it is not actually disadvantageous.

Using another method, the exponent value can be evaluated directly without a table. In this case, the number is normalized within 0.5 and 2 so as to adjust the range. Although the program becomes more complex, the error becomes smaller than the previous method. The capability of choosing either the simple method or the small-error method is a key feature of URR.

In the case of conversion from URR to decimal, the conversion process is almost the same as above. The numbers to be provided are the exponent and mantissa value of 2^{-2^i} as decimal instead of 10^{2^i}. In addition, normalizing the mantissa and adjusting the exponent should be performed for every multiplication. For outputting decimal numbers for a fixed width format, it should be noted that because of the characteristics of URR, it is appropriate for a fewer number of digits to be output for the mantissa when a greater number of digits are output for the exponent.

In any case, there are no special difficulties with these conversion processes.

9. Conclusion

The URR method described here satisfies all the design criteria raised in Chapter 3. Incorporating these criteria, URR has eight major characteristics:

—It is quite easy to convert from an exponent and a mantissa in binary to this form, and vice versa.
—Overflows/underflows do not occur in practice.
—Specifications do not depend on data length.
—No fixed arrangements are needed regarding format.
—In terms of fixed length data, every bit pattern corresponds to a different value.
—Data order is the same as the fixed point form.
—Absolute error is at most only 1 bit worse than the fixed point form.
—By simply extending data length, arbitrary real values can be infinitely approximated.

Moreover, it has become clear that arithmetic operations are slightly complicated compared with conventional representation, but there are no special difficulties.

We have also tested this method by numerical computation using a trial processer coupled with a personal computer. Results showed a reasonable level of error in estimating the length of the mantissa[1], which is dependent on the value.

Acknowledgements

The author wishes to thank Profs. Sin Hitotumatu of the University of Kyoto and Katuya Nakasima of Waseda University for their fruitful suggestions, and Dr. Yasutsugu Takeda, General Manager of the Central Research Laboratory, Hitachi Ltd., for his support of our research activities.

References

1. WILKINSON, J. H. Rounding Errors in Algebraic Processes, Prentice-Hall, Englewood Cliffs, NJ (1963).
2. MORRIS, R. Tapered Floating Point: A New Floating-Point Representation. *IEEE Trans. Comput.*, C-20, 6, June 1971, 1578–1579.
3. KAHAN, W. and PALMER, Jr. On a Proposed Floating-Point Standard, *ACM SIGNUM Newsl. Special Issue*, (Oct. 1979), 13–21.
4. IBM: IBM System/370 Principles of Operation, GA22-7000-8(Oct. 1981).
5. MATSUI, S. and IRI, M. An Overflow/Underflow-Free Floating-Point Representation of Numbers, *J. Inf. Process.*, **4**, (Nov. 1983), 123–133.
6. HAMADA, H. Data Length Independent Real Number Representation Based on Double Exponential Cut II, *Trans. IPS Japan*, **24**, 2, (Mar. 1983) 149–156 [in Japanese].

(Received October 1, 1986; revised February 3, 1987)