数理解析研究所講究録
第 189 巻 1973 年 60-69

PREPARATIONS FOR FORMAL STUDY OF A SIMPLE PROGRAM INCLUDING FLOATING-POINT ARITHMETIC

Kazuo USHIJIMA

Faculty of Engineering Kyushu University, Fukuoka

1. Introduction

Almost all programs solving scientific and engineering problems are not free form floating-point arithmetic. It is difficult to study these programs formally for many reasons. In floating-point arithmetic round-off errors are not avoidable and several laws in real arithmetic do not always hold. Moreover round-off mechanism is often in a different way on every processor and floating-point operations are often imprecisely defined. The round-off mechanism is dependent not only on machine environment but also on software environment, since most of those programs mentioned above are written in high level languages such as FORTRAN, ALGOL or PL/I. In spite of standardization of those languages, actual specifications are partly accomplished in rather different ways on different processors. For example ISO or JIS FORTRAN [1, 2] says that the resultant element of combination of two real elements is of type real, but in reality there exist a number of processors where the resultant element is of type double precision. Moreover precise information is hardly open to us how compilers of those languages are implemented, although there exist in language standards a number of points such that they are left to compiler implementors, for example, to adopt chopping or rounding in floating-point operations in FORTRAN.

To overcome these difficulties Good and London [3] defined computer interval arithmetic and proved the correctness of this implementation on a specified machine by the inductive assertion method described by Floyd and Knuth. In this paper we shall present a program with simple floating-point operations. It has no loops and is independent of numerical algorithms for the purpose of excluding logical and numerical difficulties. In the course of examining every process of this program, we shall make preparations for correct implementation of such programs mentioned above.

2. A program including simple floating-point operations

Let A, B and X be real type variables and Y and Z be double precision type variables. The following floating-point numbers are assigned to A and B:

A = 1.0, 2.0, 4.0, 8.0 and

B = $\pm 2^{-i}$ for i = 10, 11,....

If we call a FORTRAN function INDEX (see below), which value (-1, 0 or 1 for the statement number 1, 2 or 3 respectively) is returned for every pair of A and B?

```
      FUNCTION INDEX(A,B)
      DOUBLE PRECISION Y, Z
      X = (1.0+B)*A
      Y = (1.0+DBLE(B))*A
      Z = 1.0/Y
      IF(SNGL(1.0/Z)-X) 1, 2, 3
    1 INDEX = -1
      RETURN
    2 INDEX =  0
      RETURN
    3 INDEX =  1
      RETURN
      END
```

In order to clarify every process of computation in executing this program on particular computers, so many points must be considered as under.

(1) Floating-point numbers treated in FORTRAN are always normalized. Any normalized floating-point number has the form

$$\pm.d_1d_2\ldots d_t \times \beta^e$$

where base-β digits $d_1, d_2, \ldots, d_t$ satisfy the inequalities

$$1 \leq d_1 \leq \beta - 1$$

$$0 \leq d_i \leq \beta - 1 \quad \text{for } i = 2, 3, \ldots, t$$

and e is an integer with a specified interval. In this paper β is restricted to 2 and 16 in consideration of actual machines.

(2) Though ISO or JIS FORTRAN says that a real datum is a processor approximation to the value of a real number, $2^{\pm i}$ can be represented without any round-off as normalized floating-point numbers within the specified interval of the exponent part e so long as $\beta = 2$ or 16. Therefore it is assumed that four constants 1.0, 2.0, 4.0 and 8.0 are exactly converted to machine numbers by every compiler. Moreover assignment of 2^{-i} to B must be carefully executed. We assume that the statement

B = 1.0/(2.0**I)   for I = 10, 11,...

does make no deviation between B and 2^{-I}. Thus, to evaluate the function INDEX, the values of A and B are assumed to be exactly assigned as normalized floating-point numbers.

(3) In the process of computation of the right part in line 3 or 4 of the function, there exists in almost all processors such i that

$$1 + 2^{-i} \neq 1$$

and

$$1 + 2^{-(i+1)} = 1$$

hold. It is expected to be as follows:

if $\beta = 2$,

$i = t - 1$	in case of chopped operation
$i = t$	in case of rounded operation

and if $\beta = 16$,

$i = 4 \times (t - 1)$	in case of chopped operation
$i = 4 \times (t - 1) + 1$	in case of rounded operation.

(4) The resultant type of combination between type real and type double precision such as the right part in line 4 or 5 of the function is of type double precision.

(5) It is assumed that the lower part is filled with zero when a single precision argument is expressed in double precision form by the intrinsic function DBLE.

(6) In line 3 or 4 of the function, put the result of $1 + 2^{-i}$ to be C, then if $\beta = 2$, the mantissa of the result of $C \times A$ is the same as that of C, but the exponent part is added 0, 1, 2 or 3 according as A = 1.0, 2.0, 4.0 or 8.0. Therefore the returned value INDEX is not dependent on the value of A. If $\beta = 16$ on the other hand, the exponent part of $C \times A$ is equal to that of C and the mantissa of $C \times A$ is made by shifting that of C by 0, 1, 2 or 3 bits to the left according as A = 1.0, 2.0, 4.0 or 8.0, since the most significant digit d_1 of C is equal to 0001 in binary digit. It is imagined that the order between shift operation (multiplication of A) and round-off operation (addition of 2^{-i} to

1.0) will delicately affect the result.

(7) If $B < 0$, the computation of $(1 - 2^{-i}) \times A$ is more complicated than that of $(1 + 2^{-i}) \times A$. For all values of A in case $\beta = 2$ or $A = 1.0$ in case $\beta = 16$, the exponent part of $(1 - 2^{-i}) \times A$ is less than that of A so long as $1 - 2^{-i} < 1$. For $A = 2.0, 4.0, 8.0$ in case $\beta = 16$ the exponent part is equal to that of 1.0, because

$$1 < A - A \times 2^{-i} < 16.$$

Moreover it is dependent on machine eivironment whether there exists such i that

$$1 - 2^{-i} \neq 1$$

and

$$1 - 2^{-(i+1)} = 1$$

hold. For $i \leqq t - 1$ in case $\beta = 2$ and $i \leqq 4 \times (t - 1)$ in case $\beta = 16$, $1 - 2^{-i}$ is exactly expressible. Otherwise several cases are considered. At floating-point subtraction, in one case, if the difference of the exponents of both operands is greater than or equal to $t + \alpha (\alpha \geq 0)$, then the larger one is set to the result of the operation with a proper sign. On another machine the result of subtraction $1 - 2^{-i}$ in chopped operation is always less than 1.0, even if i is much greater than t or $4 \times t$ according as $\beta = 2$ or 16.

(8) Let p be the length of mantissa of a single precision floating-point number in binary digit and q be that of a double precision floating-point number. In the process of computation of the right part of line 5:

$$(1 + 2^{-i})^{-1} = 1 - 2^{-i} + 2^{-2\times i} - 2^{-3\times i} \ldots,$$

to what extent can this series be covered within double precision q bits of Z? We must consider as many cases as $2 \times i \gtreqless q$, $3 \times i \gtreqless q$ etc.

It should be noted that the round-off operation in the neighbourhood of the q-th bit is not always the same as that in the neighbourhood of the p-th bit.

(9) Computation of SNGL(1.0/Z). Operation of the function SNGL is only described in ISO or JIS FORTRAN to obtain most significant part of double precision argument. Therefore it is dependent on the compiler implementor whether chopped operation or rounded operation is adopted in SNGL operation. In another processor where the floating-point arithmetic is always done in double precision, SNGL operation may often be a synonym of no operation.

3. Analysis of the results of the program execution

The results of the program executed in two processors of $\beta = 16$ (FACOM 230-25 and HITAC 8350, where $p = 24 = 4 \times 6$ and $q = 56 = 4 \times 14$) are as follows:

INDEX(A, B) = 1 for

$A = 2.0$ and $B = 2^{-21}$,

$A = 4.0$ and $B = 2^{-21}, 2^{-22}$,

$A = 8.0$ and $B = 2^{-21}, 2^{-22}, 2^{-23}$,

INDEX(A, B) = -1 for

$A = 1.0, 2.0, 4.0, 8.0$ and

$B = -2^{-i}$ for $21 \leq i \leq 52$ ...FACOM 230-25 and

$25 \leq i \leq$ ** ...HITAC 8350

where ** = 52,56,56,53 according as A = 1.0,2.0,4.0,8.0, INDEX(A, B) = 0 otherwise

Here we restrict our investigation about these results within the interval $21 \leq i \leq 24$, where the characteristics of both processors are regarded to be well reflected. We can rewrite the computation of Y for $B > 0$

$$Y = (1 + 2^{-24+k}) \times A \quad \text{where} \quad k = 0, 1, 2, 3.$$
$$= (1 + 16^{-6} \times 2^{k}) \times A$$

Since $16^{-18} << 16^{-14}$, we obtain by neglecting terms smaller than 16^{-18}

$$Z = (1 - (2^{k} \times 16^{-6} - 2^{2\times k} \times 16^{-12})) \times A^{-1}$$

Therefore it is expected in line 6 of the function

$$1.0/Z = (1 + 2^{k} \times 16^{-6}) \times A$$

For $B < 0$ we obtain in the same way

$$1.0/Z = (1 - 2^{k} \times 16^{-6}) \times A.$$

In Table 1 are listed the expected results of the computation of 1.0/Z, SNGL(1.0/Z) and X in both chopped and rounded operations for $B > 0$. By examining all pairs of A and B to make INDEX(A, B) = 1 in Table 1, the results coincide with those in column (1)-(3) in the table. Therefore we can conclude that the single precision floating-point arithmetic in the right part of X is done in chopped operation and SNGL operation is also executed as chopped operation. On the other hand for the results of INDEX(A, B) = -1, where $B < 0$, if we assume based on the above conclusion that whenever the difference of the exponents of both operands is greater than or equal to 6 in the single precision subtraction, the larger one is set directly to the result of the operation with a proper sign, then in the computation of X the result of

A	i	1.0/Z	SNGL(1.0/Z) (a)		X (b)		sign of (a)-(b)			
			chopped (1)	rounded (2)	chopped (3)	rounded (4)	(1) -(3)	(1) -(4)	(2) -(3)	(2) -(4)
1.0	21	$1+8\times16^{-6}$	1	$1+1\times16^{-5}$	1	$1+1\times16^{-5}$	0	-	+	0
	22	$1+4\times16^{-6}$	1	1	1	1	0	0	0	0
	23	$1+2\times16^{-6}$	1	1	1	1	0	0	0	0
	24	$1+1\times16^{-6}$	1	1	1	1	0	0	0	0
2.0	21	$2+1\times16^{-5}$	$2+1\times16^{-5}$	$2+1\times16^{-5}$	2	$2+2\times16^{-5}$	+	-	+	-
	22	$2+8\times16^{-6}$	2	$2+1\times16^{-5}$	2	2	0	0	+	+
	23	$2+4\times16^{-6}$	2	2	2	2	0	0	0	0
	24	$2+2\times16^{-6}$	2	2	2	2	0	0	0	0
4.0	21	$4+2\times16^{-5}$	$4+2\times16^{-5}$	$4+2\times16^{-5}$	4	$4+4\times16^{-5}$	+	-	+	-
	22	$4+1\times16^{-5}$	$4+1\times16^{-5}$	$4+1\times16^{-5}$	4	4	+	+	+	+
	23	$4+8\times16^{-6}$	4	$4+1\times16^{-5}$	4	4	0	0	+	+
	24	$4+4\times16^{-6}$	4	4	4	4	0	0	0	0
8.0	21	$8+4\times16^{-5}$	$8+4\times16^{-5}$	$8+4\times16^{-5}$	8	$8+8\times16^{-5}$	+	-	+	-
	22	$8+2\times16^{-5}$	$8+2\times16^{-5}$	$8+2\times16^{-5}$	8	8	+	+	+	+
	23	$8+1\times16^{-5}$	$8+1\times16^{-5}$	$8+1\times16^{-5}$	8	8	+	+	+	+
	24	$8+8\times16^{-6}$	8	$8+1\times16^{-5}$	8	8	0	0	+	+

Table 1. $(1 + 2^{-i}) \times A$

$(\beta = 16)$

A	i	1.0/Z	SNGL(1.0/Z) chopped	X
1.0	21	$1-8\times16^{-6}$	$1-8\times16^{-6}$	1
	22	$1-4\times16^{-6}$	$1-4\times16^{-6}$	1
	23	$1-2\times16^{-6}$	$1-2\times16^{-6}$	1
	24	$1-1\times16^{-6}$	$1-1\times16^{-6}$	1
2.0	21	$2-1\times16^{-5}$	$2-1\times16^{-5}$	2
	22	$2-8\times16^{-6}$	$2-1\times16^{-5}$	2
	23	$2-4\times16^{-6}$	$2-1\times16^{-5}$	2
	24	$2-2\times16^{-6}$	$2-1\times16^{-5}$	2
4.0	21	$4-2\times16^{-5}$	$4-2\times16^{-5}$	4
	22	$4-1\times16^{-5}$	$4-1\times16^{-5}$	4
	23	$4-8\times16^{-6}$	$4-1\times16^{-5}$	4
	24	$4-4\times16^{-6}$	$4-1\times16^{-5}$	4
8.0	21	$8-4\times16^{-5}$	$8-4\times16^{-5}$	8
	22	$8-2\times16^{-5}$	$8-2\times16^{-5}$	8
	23	$8-1\times16^{-5}$	$8-1\times16^{-5}$	8
	24	$8-8\times16^{-6}$	$8-1\times16^{-5}$	8

Table 2. $(1 - 2^{-i}) \times A$

$(\beta = 16)$

$1 \times 16^0 - 2^k \times 16^{-6}$ is set to 1.0 and we obtain Table 2. The results of FACOM 230-25 coincide with Table 2. Supposing that the subtraction is always significant whenever the difference of the exponents of both operands is less than or equal to at least 6 in single precision and 14 in double precision, we can explain the results of HITAC 8350.

For the results of $\beta = 2$, we show the only one case of NEAC 3200-30 ($p = 23$ and $q = 39$), where every floating-point arithmetic is executed by using subroutines. The results are independent of the value of A, since $\beta = 2$. They are as follows:

INDEX(A, B) = -1 for

$B = +2^{-i}$ $(15 \leq i \leq 19)$ and

$B = -2^{-i}$ $(30 \leq i \leq 38)$,

INDEX(A, B) = 0 otherwise.

INDEX(A, B) = 1 is not realized for any pair of A and B. We can get information about floating-point subroutines on this processor to some extent by analyzing these results as above.

4. Closing remarks

It does not seem so practical and meaningful to discuss the correctness of such a simple program, but investigations in the process of determining meaning of each step of the program by analyzing the execution results have given us several pieces of information about round-off mechanism on particular processors, which will be useful for correct implementation of more complicated algorithms. In general a collection of such simple programs as above [4, 5] is helpful for writing programs on a particular processor or transferring programs from one processor to another. We can make use of the collection to examine characteristics of the processor in advance of programming or reprogramming.

References

1) Draft ISO recommendation FORTRAN, ISO/TC97/SC5, 1965.

2) Japanese Industrial Standard, Programming Language for Computers FORTRAN(level 7000) JIS C 6201 (in Japanese), 1972.

3) Good, D. I., and London, R. L., Computer interval arithmetic: definition and proof of correct implementation, J. ACM 17, 4, 1970, 603-612.

4) Ushijima, K., A test program for a unit round-off error in floating-point arithmetic, Information Processing in Japan, 11, 1971, 176-184.

5) ___________, Test programs for detecting formation sequences of expressions, ibid., 159-168.