THE SYMMETRIC LEVEL-INDEX SYSTEM OF COMPUTER ARITHMETIC

By

CHING-WEI CHAO

Bachelor of Science

Oklahoma State University

Stillwater, Oklahoma

1992

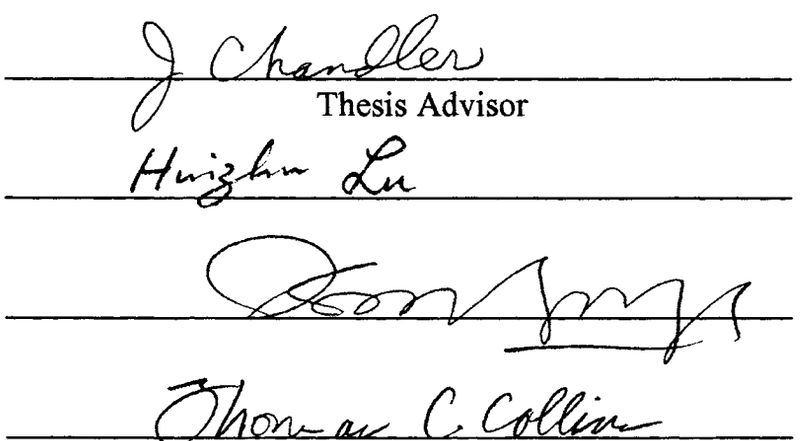
Submitted to the faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements
for the Degree of
MASTER of SCIENCE
May, 1994

OKLAHOMA STATE UNIVERSITY

THE SYMMETRIC LEVEL-INDEX SYSTEM OF

COMPUTER ARITHMETIC

Thesis Approved:



Thesis Advisor

Dean of Graduate College

ACKNOWLEDGMENTS

For getting through this graduate project, it would not be possible without the support of my professors, family, and friends. I gratefully thanks my major adviser, Dr. John P. Chandler, who gave me the guidance and encouragement throughout this project and gave me the most kindness advise and criticism in my thesis. I also thank my graduate committee, Dr. Kaylkkalth M. George and Dr. Hulzhu Lu, for their precious time spending on my project.

A special thanks to my dearest wife, Katherine, whose patient and moral support to help me keep the end goal constantly in sight. Also, I thank my parents for instilling in me the importance of learning and for supporting all my career and decisions. Finally, to all the friends who had been given me their helpful hands, I sincerely thanks them.

TABLE OF CONTENTS

LIST OF TABLES

Table Page

Table Page

LIST OF FIGURES

CHAPTER I

INTRODUCTION

This thesis will present an implementation of the symmetric level-index (sli) system of a computer arithmetic, which can solve the problem of computer overflow and underflow.

The early electronic computers often represented real numbers by using a fixed point system; it is the simplest system of representing real numbers in computers. The fixed point system stores numbers into fixed numbers of binary places, decimal places, or hexadecimal places. For example: in the decimal fixed point system, it can use one digit to represent the sign of the number and the reset of digits to represent the integer and decimal part. But unfortunately the range of numbers that can be represented in fixed point arithmetic is very limited. Often the number is very large; it cannot be represented by fixed point arithmetic. Therefore the floating point system is widely used in computing because it allows large and small numbers to be represented. For a given word length, numbers of much greater magnitude can be accommodated in floating point form than in fixed point form. However, the arithmetic operations in the floating point system are more complicated than in the fixed point system, and take a longer time to perform. The other drawback is that some precision in a floating point system needs to be sacrificed to accommodate the exponent. But overflow or underflow happen more

rarely in floating point systems than they do in fixed point systems.

In a floating point system each real number x is represented in the form

$$x = m \times r^e,$$

where r is the radix, e is a signed integer and m is a signed rational fraction; often e is called the exponent and m is called the coefficient (also called the mantissa), respectively [Sterbenz 1974]. Usually r=2 or 16 in a computer. A base of r=10 is used in every hand calculator.

Suppose $r = 2$, as is often the case.

Then each $x \in R$ can be written in the form

$$x = m \times 2^e,$$

where m ($0 \leq |m| < 1$) is a fraction and e is an integer.

We can always write

$$|m| = b_1 * 2^{-1} + b_2 * 2^{-2} + \ldots + b_i * 2^{-i} + \ldots$$

and

$$|e| = c_1 * 2^0 + c_2 * 2^1 + \ldots + c_j * 2^{j-1} + \ldots$$

t is the number of digits in the mantissa (base r) in the system.

Thus every x is a fraction with denominator equal to an integral power of the base.

For a thirty-two bit computer, suppose twenty-four bits are reserved for the representation of $|m|$, six bits are reserved for the representation of $|e|$, and two bits for the signs of m and e. The structure of this computer word can be represented as follows:

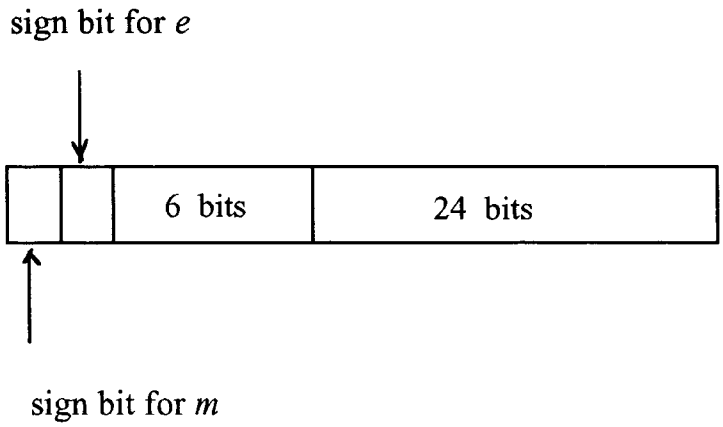


Figure 1. The Structure of A Floating Point System.

In this thirty-two bit computer, we can represent real numbers for which

$$|m| = b_1 * 2^{-1} + b_2 * 2^{-2} + ... + b_{24} * 2^{-24};$$

and

$$|e| = c_1 * 2^0 + c_2 * 2^1 + c_3 * 2^2 + ... + c_6 * 2^6;$$

For example:

$$X = 12 = \frac{12}{16} \times 2^4;$$

$$|m| = \frac{12}{16} = 0.11000...000_2; \qquad e = 4 = 100_2;$$

Therefore, $x = 12$ has the following representation in the floating point system:

0 0 000100 11000...000

The set of computer-representable numbers is also known as the set of floating point numbers, since the binary point can be made to vary merely by changing the exponent. There are only a finite number of floating point numbers in the computer, so it must contain a largest element and a smallest positive element.

$$e_{max} = 111111_2 = 2^6 - 1 = 63;$$

and

$$e_{\min} = -111111_2 = -2^6 + 1 = -63;$$

But if we use the two's complement representation to represent the exponent, then we can get $e_{\max} = 63$ and $e_{\min} = -64$.

The following are the values of the mantissa (m):

$$|m_{\max}| = 1 \times 2^{-1} + 1 \times 2^{-2} + \ldots + 1 \times 2^{-23} + 1 \times 2^{-24} = 1 - 2^{-24};$$

and

$$|m_{\min}| = 0 \times 2^{-1} + 0 \times 2^{-2} + \ldots + 0 \times 2^{-23} + 1 \times 2^{-24} = 2^{-24};$$

Therefore, we can get the largest number in the system:

$$|X_{\max}| = (1 - 2^{-24}) \times 2^{63} \cong 2^{63};$$

and the smallest positive number:

$$|X_{\min}| = 2^{-24} \times 2^{-63} = 2^{-87};$$

or

$$|X_{\min}| = 2^{-24} \times 2^{-64} = 2^{-88} \quad \text{(by using two's complement);}$$

They can be represented in the floating point system as following:

0 0 111111 111...111 the largest number: $\cong 2^{63}$;

0 1 111111 000...001 the smallest number: $\cong 2^{-87}$ or 2^{-88};

However, the floating point numbers need to be normalized when they are stored in the registers. If the floating point representation is normalized, then m satisfies

$$r^{-1} \le |m| < 1$$

That is, there is no leading zeros in the coefficient (except the entire number is zero to represent zero). For example:

$0.1100...000_2 \times 2^{23}$ is normalized whereas

$0.01100...000_2 \times 2^{24}$ is not.

Therefore, in the thirty-two bit floating point system obtained by using two's

complement, the largest and smallest positive normalized numbers will be as follows:

$$|X_{\max}| = 2^{63} \times (1 - 2^{-24}) \cong 2^{63} \qquad \text{(the largest number)}$$

and

$$|X_{\min}| = 2^{-1} \times 2^{-64} = 2^{-65} \qquad \text{(the smallest positive number)}$$

If we try to produce a number with absolute value greater than $X_{\max}$, the exponent spill is called exponent overflow or floating point overflow (also simply called overflow); or if his absolute value number is less than the $X_{\min}$, the exponent spill is called exponent underflow or floating point underflow (also simply called underflow).

During the last several decades, the floating point system has been steadily improved and extended (referring to the IEEE floating point standard [IEEE Standard 1985]); however the range of representable numbers is still not able to satisfy the scientists, and overflow and underflow remain difficult to overcome. One solution to try to get a good approximate real number involves the logarithmic number systems in which numbers are represented by their iterated logarithms (the level) relative to some base.

About the error in the floating point system: most of the numbers can be assumed to be approximate values after the computation of the arithmetic; they are usually not the same as the real numbers. In a floating point system, we always use the relative error to interpret the precision. The definitions of the absolute error and the relative error are as follows:

absolute error = true value - approximate value

relative error = (true value - approximate value)/(true value)

If we want to represent any real number in the floating point system, first this real number has to be approximated by $m \times r^e$ which has described before. Due to the fact

that the system has only limited space to store the mantissa, most real numbers can merely be represented by approximate values. The floating point system can use chopping or rounding off a number to get the result. The relative precision of a computer is measured by the machine epsilon which is defined as [Shampine & Allen 1973]:

ε (machine epsilon) ≡ the smallest x such that $1.0 \oplus x > 1.0$

(⊕ indicates the floating point approximation to the arithmetic addition operation)

In a rounding machine, machine epsilon can be written as:

$$\varepsilon = \frac{1}{2} \times r^{1-t};$$

(r is the base of the machine; t is the number of digits in the mantissa)

For $r = 2$, t can be calculated by the following program:

```
      X = 1.0D0
10    IF (1.0D0+X .GT. 1.0D0) THEN
        X = X/2.0D0
        GO TO 10
      ELSE
        EPS = X*2.0D0
      ENDIF
      T = - DLOG(EPS)/DLOG(2.0D0)
```

In a chopping machine, machine epsilon is:

$$\varepsilon = r^{1-t}.$$

The above program computes EPS correctly for both chopping and rounding, for $r = 2$, 8, or 16, or any integral power of two. Therefore, we need to determine whether a machine chops or rounds.

The following steps can help us to solve this problem:

Step 1: Use the above program to get the value of machine epsilon (ε).

Step 2: Compute (1.0D0 + nε) - 1.0D0 = w;

If n = 3 then

1. If w = 3ε, then the machine chops.

2. If w = 4ε, then the machine rounds.

or if n = 5 then

1. If w = 5ε, then the machine chops.

2. If w = 6ε, then the machine rounds.

Etc.

For example:

In the Sequent and IBM 3090 machines (eps= 2.22044604925031E-16), for n = 3, w is equal to 3×eps (= 6.66133814775093924E-16); these machines are chopping to represent numbers. In the VAX (eps=1.38777878078144568E-17), for n = 3, w is equal to 4×eps (= 5.5511151231257827E-17); it is rounding to represent numbers.

If the ratio (relative error)/(machine epsilon) is small that means there is good accuracy for an algorithm. This is how we will evaluate the results in the implementation of the sli system.

There is another phenomenon in every floating point system that we need to point out:

large number + small number = large number.

For example: 1.0 + 1.0E-20 = 1.0

That is because that every floating point system has consecutive values in it:

For every floating point number x except the largest, there is a smallest floating point number y such that $y > x$. Call the distance between two consecutive floating point numbers the gap length. For $x = m * r$ with a t-digit (base r) mantissa m, gap length $\approx r^{e-t}$. If gap length $> 2\pi$, it makes no sense to compute $\sin(x)$ or $\cos(x)$, and attempting to do so gives a system error. This occurs when x is of the order of $2\pi/\varepsilon$; for REAL arithmetic this is around $x \cong 10^7$ or so, for double precision arithmetic when x is around 10^{17} or so. Also, when x is large, e^x has poor relative accuracy because its slope is very large [Shampine & Allen 1973].

CHAPTER II

LITERATURE REVIEW

The Level-Index System

Now I want to introduce the level-index (li) system which led to symmetric level-index (sli) system. The concept of the li system is to continue taking natural logarithms (of a value) n times (the level) until the result (the index f) is in the interval between zero and one (0,1) [Clenshaw & Olver 1987].

For example:

Let X be any nonnegative real number and

$$X = e^{e^{\cdot^{\cdot^{f}}}},$$

where the process of exponentiation is performed n (n is a nonnegative integer) times. The integer n is called the level of X and the fractional part f is called the index of X. We can express the index f as:

$$f = \ln(\ln(\ln(...(\ln X)))).$$

If the number is normalized in the li system, then f has to satisfy

$$0 \le f < 1.$$

we can express X in the li system as

$$X = [n/f] = \phi(l+f;$$

where $\phi(x)$ is the generalized exponential function defined by

$$\phi(x) = x, \qquad 0 \le x < 1;$$

and

$$\phi(x) = e^{\phi(x-1)}, \qquad x \ge 1.$$

Inversely to $X = \phi(x)$ is the generalized logarithm $x = \psi(X)$ which is defined by

$$\psi(X) = X, \qquad 0 \le X < 1;$$

and

$$\psi(X) = \psi(\ln X) + 1, \quad X \ge 1.$$

Explicitly,

$$\psi(X) = n + \ln^{(n)} X, \qquad X \ge 0,$$

where $\ln^{(n)} X$ denotes the nth (nonnegative integer) repeated logarithm of X and the n is determined uniquely by

$$0 \le \ln^{(n)} X < 1.$$

For example: if $X = 123456$ then $\ln(\ln(\ln X)) = 0.90081452$, approximately. The level n is equal to 3 and the index f is 0.90081452, so $x = 3.90081452$.

The Symmetric Level-Index System

The symmetric level-index system uses a symmetric manner to represent the very large numbers with a uniform precision and the very small numbers by reciprocation [Clenshaw & Turner 1988]. In the sli system, we can represent a nonzero real number X which can be represented by

$$X = s(X)\phi(x)^{r(X)},$$

and

$$r(X) = +1, \quad \text{if } |X| \geq 1; \quad \text{or}$$

$$r(X) = -1, \quad \text{if } |X| < 1;$$

where $s(X)$ is the sign of X and $r(X)$ is a reciprocation indicator which indicates whether X is a large number (greater or equal to one) or a small number (less than one). The functions of ϕ and ψ are the same as used in the li system.

For example, large and small numbers are represented in the sli system as follows:

1. Large number:

Let $X = 123456$

=> $s(X) = +1, \quad r(X) = +1.$

=> $f = \ln(\ln(\ln 123456)) = 0.90081452.$

=> $x = 3.90081452.$

=> the level = 3, and the index = 0.90081452.

Inversely:

$$(+1) \times \phi(3.90081452)^{(+1)} = e^{\phi(2.90081452)}$$

$$= e^{e^{\phi(1.90081452)}} = e^{e^{e^{\phi(0.90081452)}}} = e^{e^{e^{(0.90081452)}}}$$

= 123455.9983, to calculator accuracy.

Transforming X to x and back to X has given an error of 0.0017 due to the finite precision of the calculations.

2. Small number:

Let $X = 0.000123456$

=> $s(X) = +1, \quad r(X) = -1.$

$\Rightarrow f = \ln(\ln(\ln \frac{1}{0.000123456})) = 0.7871761211,$

$\Rightarrow x = 3.7871761211$

$\Rightarrow$ the level =3, and the index = 0.7871761211,

Inversely:

$$(+1) \times \phi(3.7871761211)^{(-1)} = e^{\phi(2.7871761211)^{(-1)}}$$

$$= e^{e^{\phi(1.7871761211)^{(-1)}}} = e^{e^{e^{\phi(0.7871761211)^{(-1)}}}} = (e^{e^{e^{(0.7871761211)}}})^{-1}$$

= 0.0001234559039, to calculator accuracy.

Here the error is 0.961E-10.

The following structures (Figure 2 and Figure 3) are used in the li and sli systems to represent the number in a 64-bit word [Olver & Turner 1987]:

Level-Index System:

One bit for the sign of the number;

three bits for the level;

sixty bits for the index;

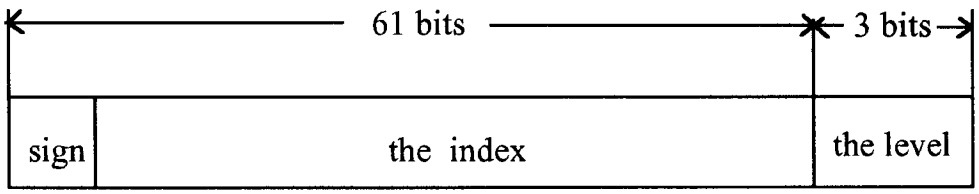


Figure 2. Structure of the li system number

Symmetric Level-Index System:

Two bits for the sign and the reciprocation sign of the number;

three bits for the level;

fifty-nine bits for the index;

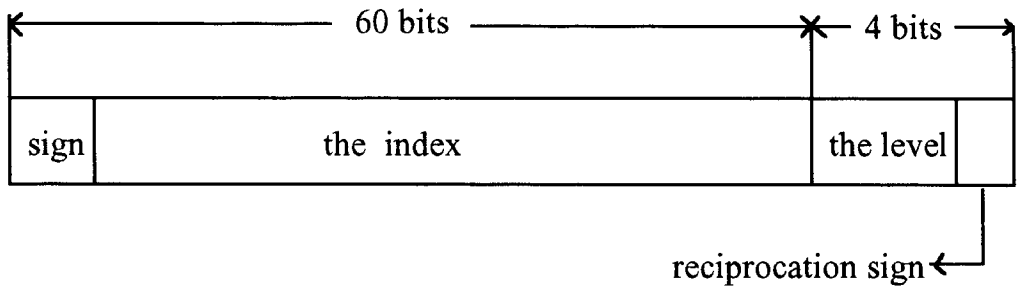


Figure 3. Structure of the sli system number

Actually, the number of bits for the level in the li or sli system is arbitrary, but three bits is in some ways the best choice. In the next chapter, I will discuss this in detail.

Algorithms for the SLI System

In this section, I will introduce the algorithms for the four basic operations (+,-,*,/) in the sli system. The algorithms of addition and subtraction are the fundamental operations in the sli system. The multiplication and addition operations are equivalent operations in a different level (n) (for instance in the large arithmetic, the operation of multiplication is $\phi(x)\phi(y) = e^{\phi(x-1)+\phi(y-1)}$; therefore multiplication is one level lower than addition); there is a similar relation between the operations of division and subtraction [Clenshaw & Olver 1987]. The following are the assumptions that will be used for the algorithms in the sli system.

Assumptions:

1. Let X and Y be two nonzero real numbers and $X \geq Y$.

2. By the sli system definition:

$$X = \phi(x), \quad x = n + f;$$

$$Y = \phi(y), \quad y = m + g.$$

3. Z is the result after the operation (+,-,*,/).

$$Z = \phi(z), \quad z = p + h.$$

Then, we must consider the following operations:

$\phi(z) = \phi(x) + \phi(y)$, (large arithmetic) or

$\phi(z) = \phi(x) + 1/\phi(y)$, (mixed arithmetic) or

$1/\phi(z) = 1/\phi(x) + 1/\phi(y)$. (small arithmetic)

4. Large arithmetic: both operand numbers are greater than one in magnitude.

5. Mixed arithmetic: one operand number is greater than one, the other is smaller than one in magnitude.

6. Small arithmetic: both operand numbers are smaller than one in magnitude.

The algorithm for addition/subtraction is as follows [Clenshaw & Olver 1988]:

Step 1: Initialize the reciprocation indicator of the result Z.

Set $r(Z) := r(X)$.

Step 2: The different algorithms for large, mixed, and small arithmetic:

(1). To get the sequences of $\{a_j\}$:

$$a_n := e^{-f}; \quad a_{j-1} := \exp(-1/a_j) \quad (j = n-2, n-3, ..., 0);$$

(2). To get the sequences of $\{b_j\}$ and $\{c_j\}$:

// Large Arithmetic //

if $(r(X) = r(Y) = 1)$ then

$b_m := a_m * e^g;$

$b_{j-1} := \exp[(b_j - 1)/a_j] \qquad (j = m, m-1, ...2);$

$c'' := 1 \pm b_1;$ // "+" for addition; "-" for subtraction //

// Mixed Arithmetic //

else if $(r(X) * r(Y) = -1)$ then

$b_m := e^{-g};$

$b_{j-1} := \exp(-1/b_j) \qquad (j = n, n-1, ..., 2);$

$c'' := 1 \pm a_1 * b_1;$

// Small Arithmetic //

else $(r(X) = r(Y) = -1)$ then

if $(m > n)$ then

$d_m := e^{-g};$

$d_{m-1} := \exp(-1/d_m) \quad (j = m, m-1, ..., n+2, n+1);$

$b_n := d_n/a_n;$

else $b_m := \exp(f - \exp^{(m-n)} * g);$

$b_{j-1} := \exp[(b_j - 1)/(a_j * b_j)] \qquad (j = n, n-1, ..., 2);$

$c' := 1 \pm b_1;$

Step 3: To compute h, or c_1; if finished then to get z:

// Large or Mixed Arithmetic //

if $(c'' < a_1)$ then

$r(Z) := -1;$

$h_1 := -\ln(c''/a_1);$ go to step 5;

else if $(n = 1)$ then

$h_1 := f + \ln c''$; go to step 5;

else $c_1 := 1 + a_1 * \ln c''$;

// Small Arithmetic //

if $(a_1 * c' > 1)$ then

$r(Z) := 1$;

$z := 1 + \ln(a_1 * c')$; finish;

else if $(n = 1)$ then

$h_1 := f - \ln c'$; go to step 5;

else $c_1 := a_1 * \ln c'$;

Step 4: To get the result z, if in step 3 we cannot get it:

For $(j = 2, 3, ..., n-1)$

if $(c_j < a_j)$ then $z := j + c_j / a_j$; finish;

else $c_{j+1} := 1 + a_{j+1} \times \ln c_j$;

if $(c_n < a_n)$ then $z := n - 1 + c_n / a_n$; finish;

else $h_{n+1} := f + \ln c_n$;

Step 5: if $(h_j \geq 1)$ then

compute $h_j := \ln h_{j-1}$; until $h_j \in [0, 1)$;

(The Result) z := (The Level) j + (The Index) h_j;

The algorithm for multiplication is as follows:

if $(n > 0$ and $m > 0)$ then

1. Taking logarithms for X and Y, we obtain

$\phi(z-1) = \phi(x-1) + \phi(y-1)$;

2. Use the addition algorithm to implement.

else if $(n > 0$ and $m = 0)$ then

1. Set $\phi(y) = Y$; the sequence c_0 is equal to Y.

2. The rest of the operation uses the addition algorithm to implement.

else if $(n = 0$ and $m = 0)$ then

(The Index) h := (The Index of X) f * (The Index of Y) g;

The algorithm for division proceeds as follows:

if $(n > 0$ and $m > 0)$ then

1. Taking logarithms for X and Y, we obtain

$$\phi(z-1) = \phi(x-1) - \phi(y-1);$$

2. Use the subtraction algorithm to implement.

else if $(n > 0$ and $m = 0)$ then

1. The sequence of c_1 is equal to $1 + a_1 * \ln(1/g)$.

2. $\phi(z-1) = \phi(x-1) + \ln(1/g)$;

Use the addition algorithm to implement.

else if $(n = 0$ and $m = 0)$ then

$$\phi(z-1) = \ln f + \ln(1/g);$$

The algorithm for large arithmetic in the sli system is very similar to the algorithm of the li system which is described detail in [Clenshaw & Olver 1987]. The following two examples show mixed and small arithmetic in the sli system, using calculator accuracy.

1. Mixed arithmetic:

$X = 4000.0D0$; $Y = 0.004D0$; operation="+";

From the sli system definition:

=> $s(X) = +1$; $r(X) = +1$; and $s(Y) = +1$; $r(Y) = -1$;

=> $n = 3$; $f = 0.7493093176353089$;

$m = 3$; $g = 0.5356991864069362$;

=> $s(Z) = +1$;

Step 1. $r(Z) = 1$;

Step 2.

=> $a_1 = 2.5000000000000000E-04$; $a_2 = 0.1205683644772228$;

$a_3 = 0.4726929206839799$;

=> $b_1 = 4.0000000000000018E-03$; $b_2 = 0.1811114874987057$;

$b_3 = 0.5852599412604286$;

=> $c" = 1.000001000000000$;

=> Step 3. $c_2 = 1.000000120568304$;

=> Step 4. $c_3 = 1.000000056991780$;

$h_3 = 0.7493093746270876$;

=> Step 5. $Z = 3.7493093746270876$;

The Result = $e^{e^{e^{(0.7493093746270876)}}} = 4000.004000000007$.

(True value = 4000.004)

2. Small arithmetic:

$X = 0.00000002$; $Y = 0.06$; operation = "+";

From the definition of the sli system, X has to be greater than Y. Therefore we need to exchange X and Y in the operations of addition and subtraction if X is less than Y. But in the operation of multiplication and division, first, we have to determine which if-then-else statement needs to be selected, then follow the algorithms of addition and

subtraction to get the result. The sign of the result $s(Z)$ comes from the $s(X)$ and $s(Y)$ n the operations of multiplication and division. If the operation is addition or subtraction, then the sign of the result needs to be found by comparing X and Y in magnitude.

=> $X = 0.06;$ $Y = 0.00000002;$

=> $s(X) = +1;$ $r(X) = -1;$ and $s(Y) = +1;$ $r(Y) = -1;$

=> $n = 3;$ $f = 3.3819156282812881E-02;$

$m = 4;$ $g = 5.4577257566556847E-02;$

=> $s(Z) = +1;$

Step 1. $r(Z) = -1;$

Step 2.

=> $a_1 = 6.0000000000000000E-02;$ $a_2 = 0.3554404602366818;$

$a_3 = 0.9667463188278800;$

=> $b_1 = 3.3333333333333428E-07;$ $b_2 = 0.1587028847921498;$

$b_3 = 0.35977755521432298;$

=> $c' = 1.000000333333333;$

Step 3. $c_2 = 0.9999998815198662;$

Step 4.

=> $c_3 = 0.9999998854597601;$

=> $h_3 = 3.3819041742566474E-02;$

Step 5. $Z = 3.033819041742567;$

The Result = $1/e^{e^{e^{(3.3819041742566474E-02)}}}$ = 6.0000019999999976E-02.

(True value = 6.000002E-02).

CHAPTER III

RESULTS AND DISCUSSION

RESULTS

The sli algorithms have been implemented in A.N.S.I. Standard FORTRAN 77. The program was compiled on Sequent, VAX, and IBM 3090 machines. All of the results are obtained in double precision. Appendix A contains the test data and results from the different operations (+,-,*,/) and different arithmetic (large, mixed, and small) on different machines. These machines' double precision epsilons (ε) are close to 10^{-16}. The Sequent machine uses chopped binary; the machine epsilon is equal to 2.2204460492503131E-16. The VAX uses rounded binary; the machine epsilon is equal to 1.3877787807814457E-17. And the IBM 3090 uses chopped hexadecimal; the machine epsilon is equal to 2.2204460492503131E-16. If the relative error after the computation in the sli system is less than ten times the machine epsilon then the accuracy of the algorithm is good; but if the relative error is close to one thousand times the machine epsilon then the result is not as accurate. In the tables of the result (in APPENDIX C), most of the values of (relative error)/(machine epsilon) are between zero and five hundred. In general we will find that if X and Y are larger or smaller (if the ratio of X to Y is near ε or $1/\varepsilon$) then the value of (relative error)/(machine epsilon)

will be large. Since the result are obtained after many series of computations for logarithms, exponents, and others operations, they might lose much of their precision.

In the sli system, the multiplication of small arithmetic uses the ordinary fixed point multiplication to get the result; here we use $Z = X * Y$ to represent the fixed point computation. The multiplication of small arithmetic has less relative error than others' arithmetic operation because it does not require many series of computations which might lose its precision. The program (in APPENDIX D) for the implementation of the sli system is uses double precision for the index of sli numbers and sequences of $\{a_j\}$, $\{b_j\}$, and $\{c_j\}$; the level of sli numbers is declared by integer.

DISCUSSION

The advantages of the sli system:

1. Efficient, relative to the li system:

   The sli system uses reciprocals of the li numbers to represent the small numbers; however, the li system needs to use the fixed point system to represent the small numbers at level zero. That means the sli system removes the level zero to another levels, therefore it can avoid many of the special cases [Clenshaw & Turner 1988]. For example:

   In the sli system, to compute two of the small numbers only needs one operation; but in the li system there will be more operations required.

$$\frac{1}{\phi(x)} + \frac{1}{\phi(y)} = \frac{\phi(x)+\phi(y)}{\phi(x)*\phi(y)};$$

   This computation needs three li operations (+,*,/) in the li system, whereas, the sli

system needs only one operation; and the result is also less satisfactory in the li system because the operations on the larger or smaller of x and y can affect the accuracy.

2. Symmetric, and resistant to underflow as well as overflow:

The range in the thirty-two bit the floating point system is only $[2^{-65}, 2^{63})$; but the sli system at least can represent the numbers from $2^{2^{2^{65536}}}$ to $2^{-2^{2^{65536}}}$. These numbers are much larger or smaller than in any floating point system. The comparison table of the overflow and underflow limits is as follows:

TABLE 1

THE COMPARISON OF OVERFLOW AND UNDERFLOW

	32-Bit Word		64-Bit Word	
	Overflow	Underflow	Overflow	Underflow
VAX fl. pt.	2^{127}	2^{-129}	2^{127}	2^{-129}
level-index sys.	$2^{2^{65536}}$	2^{-28}	$2^{2^{65536}}$	2^{-60}
symmetric level-index sys.	$2^{2^{2^{65536}}}$	$2^{-2^{2^{65536}}}$	$2^{2^{2^{65536}}}$	$2^{-2^{2^{65536}}}$

3. Closure:

Both the li and sli systems are closed under the operations of addition, subtraction, multiplication, and division, other than division by zero, when their operations are used with any finite precision arithmetic [Lozier & Olver 1990]. Because the subset of the representable numbers interval [-M, M] and arbitrary constant M is large enough (the sli system by using three bits for the level can represent the very wide range of numbers referred to in APPENDIX B). That means they are free from the

problems of overflow and underflow.

The disadvantages of the sli system:

1. The sli system has many arithmetic operations that are more complicated and slower to implement (than the fixed point and floating point systems) because of the computation of the exponentials and logarithms [Clenshaw & Olver 1984].

   If x, y, and z have the following relation in the li or sli system:

   $$\phi(z) = \phi(x) \pm \phi(y) \qquad (x \geq y),$$

   then the computation of the algorithms are based on the three equations below:

   $$a_j = 1/\phi(x-j);$$

   $$b_j = \phi(y-j)/\phi(x-j);$$

   $$c_j = \phi(z-j)/\phi(x-j);$$

   If software or hardware is used to implement the sli system by parallelizing the computation of the sequences $\{a_j\}$, $\{b_j\}$, and $\{c_j\}$ this can save much of the execution time.

2. Some people argue that a disadvantage of the sli system is that integers cannot be represented exactly within it [Turner 1989].

   Generally, the fixed point system uses the absolute precision (absolute error) and the floating point system uses the relative precision (relative error) to measure the error.

   The li and sli systems use the generalized precision to measure the error:

   Let $X = \phi(x)$. If $x < 1$, that means in level zero, to measure the precision in the li system we use the absolute precision α. If $1 \leq x < e$, that means in level one, then the precision of the li or sli system is measured using the relative precision α to

represent; and if $x \geq e$, which means it is at high levels, then both systems are using the generalized precision to represent the error. The generalized precision (gp) is defined as follows [Clenshaw & Olver 1984]:

Let x and $\bar{x}$ be positive numbers:

$$|\psi(x) - \psi(\bar{x})| \leq \alpha$$

where ψ is the generalized logarithm function which was described in Chapter II.

Now, we can rewrite the gp as:

$$x \cong \bar{x};\ \mathrm{gp}(\alpha) \quad \text{or}$$

$$X = \phi(\bar{x});\quad |x - \bar{x}| \leq \alpha.$$

In a thirty-two bit computer, the sli system provides the uniform precision gp (2^{-27}). And a sixty-four bit computer can provide the uniform precision $\mathrm{gp}(2^{-59})$ [Clenshaw, Olver & Turner 1989].

There is another thing about the operation of sli arithmetic we need to discuss:

If a small number is added to or subtracted from a large number, then the result of this operation would just be the large number. (Similar phenomena occur in all finite recision representations.)

For example:

$$a = \phi(4.5123456) = 4.98706924E + 87$$

$$b = \phi(3.9123456) = 173366.835$$

$$a + b = \phi(4.5123456) + \phi(3.9123456) = \phi(4.5123456) = a.$$

Comparing these two numbers, 'b' is a small number compared to 'a', a large number. After the operation of the sli addition arithmetic, the result remains unchanged equal to 'a'. At this precision, this addition is trivial. But if the computer can provide more

precision for the index (f), then we may be able to distinguish the difference of the result. Now, suppose the li and sli systems' internal arithmetic base of the computer is r, and the index part is stored to d r-nary places. Then if we add any positive representable number $\phi(x)$ to itself, then the stored sum will not exceed x as long as

$$\phi(x + cr^{-d}) > 2\phi(x).$$

Here c is a positive constant. By using the above equation, we can derive the following results [Lozier & Olver 1990]:

$$\phi(5) + \phi(x) \cong \phi(5); \quad \text{gp}(2^{-27}) \qquad \text{when } x \leq 5$$

If there is a positive number at level 5 added to $\phi(x)$, x is another positive number which is smaller than the first number at level 5. The result will be: if the index is not more than twenty-seven bits, then the result of the addition will be the same as the larger number. One consequence of this is that $\phi(5) + \phi(5) = \phi(5)$ at this precision. The corresponding bound level for multiplication and division is at the sixth level, because they are one level below compared to addition and subtraction in the sli system.

$$\phi(6) * \phi(6) \cong \phi(6); \quad \text{gp}(2^{-27}) \qquad \text{when } x \leq 6.$$

$$\text{Also, } \phi(6) + \phi(x) \cong \phi(6); \quad \text{gp}(2^{-5500000}) \qquad \text{when } x \leq 6.$$

Under the same circumstance, at level 6, only if the index exceeds 5.5 million bits can the result be distinguished from the larger number. The same consequences for multiplication and division happen at level 7:

$$\phi(6) * \phi(6) \cong \phi(6).$$

The table of results for operations on large numbers in the sli level and index implementation using the Sequent machine is in APPENDIX C. From the results, we can see that addition (or subtraction) in sli arithmetic is almost trivial at level 5; and it is

trivial at level six, seven, or even more. If we have the same two sli numbers (x) for operands in addition and the result is R (= $\psi(\phi(x)+\phi(x))$), then from the program we can plot (R-x) as the y-axis and x as the x-axis as follows.

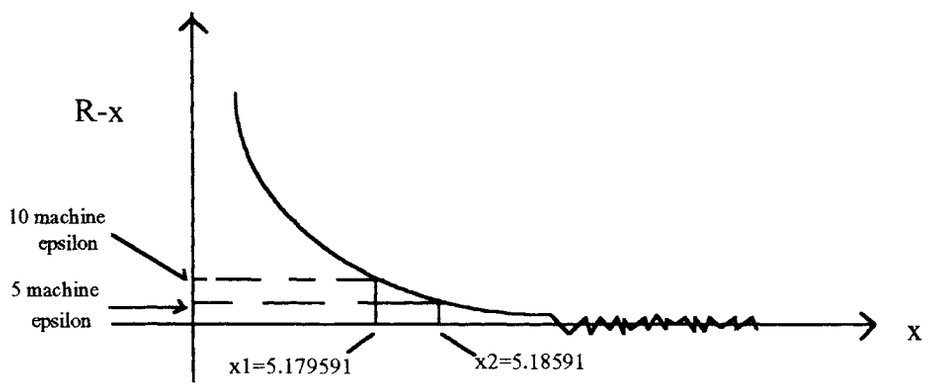


Figure 4. Graph of (R-x) vs. x

The difference between the operations of multiplication (or division) and addition (or subtraction) in sli arithmetic is one level only.

For example:

$$\psi(\phi(3)+\phi(3)) = 3.2046791426805$$

$$\psi(\phi(4)*\phi(4)) = 4.2046791426805$$

$$\psi(\phi(4)+\phi(4)) = 4.0161875057657$$

$$\psi(\phi(5)*\phi(5)) = 5.0161875057657$$

From the above example, we find that addition in level 3 and multiplication in level 4 of sli numbers have the same index; and addition in level 4 and multiplication in level 5 of sli numbers also have the same index.

Therefore, in a thirty-two bit, sixty-four bit or even-more-bit sli system, up to

5.5×10^6–bit words, all addition or subtraction at level 6 is trivial and all multiplication at level 7 is trivial. At level 6, $x \oplus x = x$ for all x. As a matter of fact, the numbers represented by the sli system in level 5, 6, or 7 are enormous. Therefore, using three bits for the level to represent from level 1 to level 8 is sufficient to represent all sli numbers, for index lengths up to 5.5×10^6 bits, At present 5.5×10^6 bits is an impractically great length for most computations.

The conclusions above hold only for the four basic arithmetic operations of addition, subtraction, multiplication, and division. They do not hold for exponentiation, because e^x always adds one to $\psi(X)$:

$$\psi(e^x) = \psi(X) + 1.$$

Beyond the point at which multiplication becomes trivial (where $X * X = X$ to machine precision), however, so much precision is lost that many results become very strange and even paradoxical. Just the fact that $X * X = X$ for a huge number X is exceedingly counterintuitive. The reason for this is that the gap length at these values greatly exceeds X^2; X is the sli number closest to X^2, at this point and using this precision.

Other paradoxical results also occur: for very large X values.

$$\psi(X^X) = \psi(e^{\ln(X^X)}) = 1 + \psi(\ln X^X) = 1 + \psi(X \ln X)$$

and because $\ln X$ is small compared to X and $X^2 = X$, $X \ln X = X$ so that

$$\psi(X^X) = 1 + \psi(X) = \psi(e^x).$$

Hence $X^X = e^X$, and because $e^X < X! < X^X$ for large real numbers, we have $e^X = X! = X^X$ for all very large sli numbers.

CHAPTER IV

CONCLUSION

The fixed point system can only represent a very limited range of numbers; the floating point system is still not closed under the basic arithmetic operations of addition, subtraction, multiplication, and division (excluding division by zero), although overflow and underflow are much less prevalent. Now, the sli and li systems have the solution to the problems of overflow and underflow; and the sli system is more efficient than the li system. The disadvantage of the sli system is losing the speed of floating point and some precision but these prices are acceptable for the adoption of the floating point system instead of the fixed point system. Meanwhile, the speed of the modern computer is increasing; the lower speed of sli operations may be acceptable. The adoption of sli hardware would allow users to avoid overflow and underflow completely, which would be quite desirable.

BIBLIOGRAPHY

Allen, Richard C. and Shampine, Lawrence F., Jr. Numerical Computing: An Introduction. W. B. Saunders Company, Philadelphia, London, Toronto, 1973.

Clenshaw, C. W. and Olver, F. W. J., "Level-Index Arithmetic Operations," Society for Industrial and Applied Mathematics Journal of Numerical Analysis, 24 (1987), pp. 470-485.

Clenshaw, C. W. and Olver, F. W. J., "The Symmetric Level-Index System," IMA Journal of Numerical Analysis, 8 (1988), pp. 517-526.

Clenshaw, C. W. and Olver, F. W. J., "Beyond Floating Point," Journal of the Association for Computing Machinery, Vol. 31, No. 2, April 1984, pp. 319-328.

Clenshaw, C. W., Olver, F. W. J. and Turner, Peter R., "Generalized Exponential and Logarithmic Functions," Comput. Math. Appl. Part B, 12 (1986), pp. 1091-1101.

Clenshaw, C. W., Olver, F. W. J. and Turner, Peter R., "Level-Index Arithmetic: An Introductory Survey," Numerical Analysis and Parallel Processing, (Turner, P. R., Ed.) pp. 95-168, Springer, Berlin-Heidelberg-New York-Tokyo-Hong Kong: Lecture Notes in Mathematics 1397, 1989.

Demmel, J. W. "On Error Analysis in Arithmetic with Varying Relative Precision," in Proceedings Eighth Symposium on Computer Arithmetic, (M. J. Irwin and R. Stefanelli, eds.) IEEE Computer Society Press, Washington, DC, 1987, pp. 148-152.

Devlin, Keith, "Computers and Mathematics," Notices of The American Mathematical Society, Vol. 38, No. 4, April 1991.

Feldstein A. and Turner, Peter R., "Overflow, Underflow, and Severe Loss of Significance in Floating-Point Addition and Subtraction," IMA Journal of Numerical Analysis, 6 (1986), pp. 241-251.

Gregory, Robert Todd, Error-Free Computation, Robert E. Krieger Publishing Company, Huntington, New York, 1980.

Hamada, H., "URR: Universal Representation of Real Numbers," New Generation Computing, OHM-Sha, Springer-Verlag, Berlin, New York, 1983, pp. 205-209.

IEEE Standard 754, Binary floating-point arithmetic, The Institute of Electrical and Electronic Engineers, New York, 1985.

Lozier, D. W. and Olver, F. W. J. "Closure and Precision in Level-Index Arithmetic," SIAM Journal of Numerical Analysis Vol. 27, No. 5, October 1990, pp. 1295-1304.

Matsui, S. and Iri, M., "An Overflow/Underflow-Free Floating-Point Representation of Numbers," Journal of Information Process., 4 (1981), pp. 123-133.

Olver, F. W. J. and Turner, Peter R., "A Closed Computer Arithmetic," Proceedings of The Eighth Symposium on Computer Arithmetic, Como, ITALY, May 1987, pp. 139-143.

Olver, F. W. J. and Turner, Peter R., "Implementation of Level-Index Arithmetic Using Partial Table Look-Up," Proceedings Eighth Symposium on Computer Arithmetic, (M. J. Irwin and R. Stefanelli, eds.) IEEE Computer Society Press, Washington, DC, 1987, pp. 144-147.

Sterbenz, Pat H. Floating-Point Computation, Prentice-Hall, Englewood Cliffs, N.J. 1974.

Turner, Peter R. "Towards a Fast Implementation of Level-Index Arithmetic," Bull. Inst. Math. Appl., 22 (1986), pp. 188-191.

Turner, Peter R., "A Software Implementation of SLI Arithmetic," Proceedings of the Ninth Symposium on Computer Arithmetic, (Ercegovac and Swartxlander, eds.) IEEE Computer Society, Washington DC, 1989, pp. 18-24.

Turner, Peter R. "Algorithms for The Elementary Functions in Level-Index Arithmetic," in Scientific Software Systems, J. C. Mason and M. G. Cox, eds., Chapman and Hall, London, 1990, pp. 123-134.

APPENDIX A

THE RESULTS AND RELATIVE ERRORS OF SLI ARITHMETIC IN FOUR BASIC OPERATIONS

TABLE 2

Machine Epsilon (ε) = 2.2204460492503131E-16
Double Precision Large Arithmetic
Operation: Addition (+)
Sequent Computer

X	Y	Approximate Value	True Value	Relative Error	$\|R.E.\|$
1.0E+12	5	1.00000000000497644E+12	1,000,000,000,005	+2.356E-14	106.1
1.0E+10	5	1.00000000050000572E+10	10,000,000,005	5.720E-15	25.8
5.0E+06	4,000	5.00399999999997485E+06	5,004,000	-5.026E-15	22.7
50,000	5	5.00050000000000072E+04	50,005	+1.44E-16	0.8
50,000	400	5.04000000000003419E+04	50,400	+6.784E-15	30.6
500	20	5.20000000000000909E+02	520	+1.811E-15	8.2
500	400	9.00000000000000909E+02	900	+1.01E-15	4.6
100	5	1.05000000000000056E+02	105	+5.333E-16	2.4
20	20	3.99999999999999289E+01	40	-1.778E-15	8
5	4	9.00000000000000177E+00	9	+1.967E-16	0.9

TABLE 3

Double Precision Large Arithmetic
Operation: Multiplication (*)
Sequent Computer

X	Y	Approximate Value	True Value	Relative Error	|R.E.|/
1.0E+12	5	4.99999999999990234E+12	5,000,000,000,000	-1.953E-14	88
1.0E+10	5	5.00000000000003356E+10	50,000,000,005	+6.712E-15	30.3
5.0E+06	4,000	1.99999999999999046E+10	20,000,000,000	-4.77E-15	21.5
50,000	400	1.99999999999997653E+07	20,000,000	-1.174E-14	52.9
50,000	5	2.49999999999998952E+05	250,000	-4.192E-15	18.9
500	400	1.99999999999998632E+05	200,000	-6.84E-15	30.8
500	20	9.99999999999999090E+03	10,000	-9.1E-16	4.1
100	5	4.99999999999998976E+02	500	-2.048E-15	9.3
20	20	4.00000000000000227E+02	400	+5.67E-16	2.6
5	4	2.00000000000000071E+01	20	+3.55E-16	1.6

TABLE 4

Double Precision Large Arithmetic
Operation: Division (/)
Sequent Computer

X	Y	Approximate Value	True Value	Relative Error	$\|R.E.\|/\varepsilon$
1.0E+12	5	1.99999999999997131E+11	2.0E+11	-1.435E-14	64.6
1.0E+10	5	1.99999999999999523E+09	2.0E+09	-2.385E-15	10.8
5.0E+06	4,000	1.25000000000000864E+03	1,250	+1.088E-15	4.9
50,000	5	9.99999999999999090E+03	10,000	-9.1E-16	4.1
50,000	400	1.24999999999999829E+02	1.25	-1.368E-15	6.2
500	20	2.49999999999999964E+01	25	-1.44E-16	0.7
500	400	1.25000000000000133E+00	1.25	+1.064E-15	4.8
100	5	2.00000000000000071E+01	20	+3.55E-16	1.6
50	25	1.99999999999999982E+00	2	-9.0E-17	0.9
5	4	1.24999999999999977E+00	1.25	-1.84E-16	0.8

TABLE 5

Double Precision Mixed Arithmetic
Operation: Addition (+)
Sequent Computer

X	Y	Approximate Value	True Value	Relative Error	$\|R.E.\|$
1.0E+12	0.05	1.0000000000000061E+12	1,000,000,000,000.05	-4.39E-14	197.7
1.0E+10	0.05	1.0000000000050276E+10	10,000,000,000.05	2.36E-14	124.2
1.0E+08	0.05	1.0000000004999964E+08	100,000,000.05	-3.61E-15	16.3
5.0E+06	0.05	5.0000000050000213E+06	5,000,000.05	4.26E-15	19.2
500	5.0E-07	5.0000000050000079E+02	5.00000000500E+02	1.59E-15	7.2
50	1.0E-06	5.0000000999999976E+01	5.00000100000E+01	-4.78E-16	2.2
50	1.0E-08	5.0000000009999972E+01	5.00000000100E+01	-5.52E-16	2.5
5	1.0E-10	5.0000000000999991E+00	5.00000000010E+00	-1.76E-16	0.8
1	5.0E-12	1.0000000000050000E+00	1.00000000000E+00	0	0
5	0.5	5.4999999999999991E+00	5.5	-1.62E-16	0.8

TABLE 6

Double Precision Mixed Arithmetic
Operation: Multiplication (*)
Sequent Computer

X	Y	Approximate Value	True Value	Relative Error	$\|R.E.\|/\varepsilon$
1.0E+12	0.5	5.00000000000014587E+11	5.0E+11	+2.92E-14	131.4
1.0E+08	0.5	4.99999999999996349E+07	5.0E+09	-1.27E-14	57.2
5.0E+06	1.0E-03	5.00000000000000363E+03	5.0E+03	+7.26E-16	3.3
5,000	1.0E-03	5.00000000000000363E+03	5	+7.10E-16	3.2
5,000	1.0E-08	4.99999999999996802E-05	5.0E-05	-6.40E-15	28.8
5	0.5	2.49999999999999955E+00	2.5	-1.8E-16	0.8
1	1.0E-05	9.99999999999999688E-06	1.0E-05	-3.12E-16	1.4
5	1.0E-08	5.00000000000016431E-08	5.0E-08	+3.29E-14	14.8
5	1.0E-10	4.99999999999986943E-10	5.0E-10	-2.61E-14	117.6
5	1.0E-12	4.99999999999984190E-12	5.0E-12	-3.16E-14	142.4

TABLE 7

Double Precision Mixed Arithmetic
Operation: Division (/)
Sequent Computer

X	Y	Approximate Value	True Value	Relative Error	$\|R.E.\|/\varepsilon$
1.0E+12	0.5	1.99999999999993798E+12	2.0E+12	-3.10E-14	139.7
1.0E+10	0.5	1.99999999999999046E+10	2.0E+10	-4.77E-15	21.5
1.0E+08	0.5	2.00000000000000000E+08	2.0E+08	0	0
5.0E+06	0.01	4.99999999999996960E+08	5.0E+08	-6.08E-15	27.4
5,000	1.0E-04	4.99999999999996349E+07	5.0E+07	-7.30E-15	32.9
500	1.0E-08	5.00000000000003356E+10	5.0E+10	+6.73E-15	30.3
50	0.2	2.50000000000000341E+02	250	+1.36E-15	0.6
5	0.5	9.99999999999999822E+00	10	-1.78E-15	8
5	1.0E-05	4.99999999999998835E+05	5.0E+05	-2.33E-15	1.1
5	1.0E-12	4.99999999999990234E+12	5.0E+12	-1.95E-14	88

TABLE 8

Double Precision Small Arithmetic
Operation: Addition (+)
Sequent Computer

X	Y	Approximate Value	True Value	Relative Error	$\|R.E.\|/$
0.5	0.1	6.00000000000000000E-01	0.6	0	0
0.6	0.4	9.99999999999999822E-01	1	-1.78E-16	0.8
0.5	0.01	5.10000000000000071E-01	5.10E-01	1.39E-16	0.6
0.5	1.0E-06	5.00000100000000050E-01	5.000001E-01	1.0E-16	0.5
0.5	1.0E-10	5.00000000010000008E-01	5.0000000001E-01	0.16E-01	5E-2
0.5	2.0E-12	5.00000000000200000E-01	5.000000000001E-01	0	0
1.0E-05	4.0E-05	4.99999999999996802E-05	5.0E-05	-6.37E-15	28.8
1.0E-05	5.0E-10	1.00005000000000143E-05	1.00005E-05	1.43E-15	6.5
1.0E-12	1.0E-12	1.99999999999994173E-12	2.0E-12	-2.91E-14	131.2
3.0E-12	2.0E-12	4.99999999999984190E-12	5.0E-12	-3.16E-14	142.4

TABLE 9

Double Precision Small Arithmetic
Operation: Multiplication (*)
Sequent Computer

X	Y	Approximate Value	True Value	Relative Error	$\|R.E.\|/\varepsilon$
0.2	0.8	1.60000000000000035E-01	0.16	+2.19E-16	1
0.09	0.03	2.69999999999999975E-03	2.7E-03	-9.26E-17	0.4
0.03	1.0E-05	3.00000000000000044E-07	3.0E-07	+1.47E-16	0.7
0.03	1.0E-10	3.00000000000000044E-12	3.0E-12	+1.47E-16	0.7
0.03	1.0E-12	2.99999999999999964E-14	3.0E-14	-1.08E-16	0.6
3.0E-05	2.0E-05	6.00000000000000088E-10	6.0E-10	+1.47E-16	0.7
3.0E-04	3.0E-05	9.00000000000000000E-09	9.0E-09	0	0
3.0E-08	3.0E-04	8.99999999999999982E-12	9.0E-12	-2.0E-17	0.1
1.0E-10	5.0E-03	4.99999999999999911E-13	5.0E-13	-1.78E-16	0.8
0.5	3.0E-12	1.50000000000000017E-12	1.5E-12	+1.13E-16	0.5

TABLE 10

Double Precision Small Arithmetic
Operation: Division (/)
Sequent Computer

X	Y	Approximate Value	True Value	Relative Error	$\|R.E.\|/\varepsilon$
5.0E-12	2.0E-10	2.49999999999999911E-02	2.5E-02	-3.56E-15	16.1
5.0E-08	5.0E-05	1.00000000000000466E-03	1.0E-03	+4.66E-15	21
5.0E-04	0.02	2.50000000000000710E-03	2.5E-03	+2.84E-15	12.8
0.5	0.02	2.49999999999999964E+01	25	-1.44E-16	0.7
0.9	0.3	3.00000000000000088E+00	3	+2.93E-16	1.3
9.9E-02	9.9E-04	1.00000000000000127E+02	100	+1.27E-15	5.7
0.9	0.02	4.49999999999999715E+01	45	-6.33E-16	2.9
0.8	2.0E-05	3.99999999999998981E+04	4,000	-2.59E-15	11.5
5.5E-02	1.0E-05	5.49999999999998908E+03	5,500	-1.99E-15	9
0.5	2.0E-09	2.49999999999997109E+08	2.5E+08	-1.16E-14	52.1

TABLE 11

Machine Epsilon (ε) = 1.3877787807814457E-17
Double Precision Large Arithmetic
Operation: Addition (+)
VAX

X	Y	Approximate Value	True Value	Relative Error	$\|R.E.\|/\varepsilon$
1.0E+12	5	1.0000000000005003937E+12	1,000,000,000,005	+3.94E-15	283.64
1.0E+10	5	1.0000000000499998665E+10	10,000,000,005	-1.34E-15	96.18
5.0E+06	4,000	5.0040000000000009313E+06	5,004,000	+1.86E-15	134.08
50,000	5	5.0005000000000002728E+04	50,005	+5.46E-16	39.31
50,000	400	5.0400000000000002910E+04	50,400	+5.77E-16	41.6
500	20	5.2000000000000000284E+02	520	+5.46E-17	3.94
500	400	9.0000000000000000455E+02	900	+5.06E-17	36.43
100	5	1.0499999999999999929E+02	105	-6.76E-17	5.05
20	20	4.0000000000000000266E+01	40	+6.65E-17	4.97
5	4	9.0000000000000000222E+00	9	+2.46E-17	1.84

TABLE 12

Double Precision Large Arithmetic
Operation: Multiplication (*)
VAX

X	Y	Approximate Value	True Value	Relative Error	$\|R.E.\|/\varepsilon$
1.0E+12	5	5.0000000000000366E+12	5.0E+12	+7.32E-17	5.27
1.0E+10	5	5.0000000000000002575E+10	5.0E+10	+5.15E-16	37.1
5.0E+06	4,000	1.999999999999997425E+10	2.0E+10	-1.29E-15	92.8
50,000	400	1.999999999999999674E+07	2.0E+07	-1.63E-16	11.74
50,000	5	2.499999999999997817E+05	250,000	-8.73E-16	62.91
500	400	1.999999999999999673E+05	200,000	-1.64E-16	11.78
500	20	9.999999999999999773E+03	10,000	-2.27E-17	1.64
100	5	5.000000000000001173E+02	500	+2.35E-16	16.9
20	20	4.000000000000000639E+02	400	+1.60E-16	11.51
5	4	1.999999999999999956E+01	20	-2.20E-17	1.59

TABLE 13

Double Precision Large Arithmetic
Operation: Division (/)
VAX

X	Y	Approximate Value	True Value	Relative Error	$\|R.E.\|/\varepsilon$
1.0E+12	5	2.000000000000009384E+11	2.0E+11	+4.692E-15	338.04
1.0E+10	5	2.000000000000001431E+09	2.0E+09	+7.155E-16	51.55
5.0E+06	4,000	1.250000000000000199E+03	1,250	+1.592E-16	11.47
50,000	5	9.999999999999999773E+03	10,000	-2.270E-17	1.64
50,000	400	1.249999999999999929E+02	1.25	-5.680E-17	4.09
500	20	2.499999999999999933E+01	25	-2.680E-17	1.93
500	400	1.250000000000000250E+00	1.25	+2.000E-16	14.41
100	5	2.000000000000000266E+01	20	+1.330E-16	9.58
50	25	2.000000000000000111E+00	2	+5.550E-17	4
5	4	1.249999999999999944E+00	1.25	-4.480E-17	3.23

TABLE 14

Double Precision Mixed Arithmetic
Operation: Addition (+)
VAX

X	Y	Approximate Value	True Value	Relative Error	$\|R.E.\|/\varepsilon$
1.E+12	0.05	1.0000000000000572E+12	1,000,000,000,000.05	+7.22E-15	520.17
1.E+10	0.05	1.0000000000049973E+10	10,000,000,000.05	-2.72E-15	196.18
1.E+08	0.05	1.0000000005000022E+08	100,000,000.05	+2.22E-15	160.23
5.E+06	0.05	5.0000000050000116E+06	5,000,000.05	+2.31E-15	166.71
500	5.E-07	5.0000000049999994E+02	5.00000000500E+02	-1.30E-16	9.39
50	1.E-06	5.0000000999999997E+01	5.00000100000E+01	-6.82E-17	4.91
50	1.E-08	5.0000000010000007E+01	5.00000001000E+01	+1.41E-16	10.14
5	1.E-10	5.0000000001000000E+00	5.00000000010E+00	+1.60E-18	0.12
1	5.E-12	1.0000000000050000E+00	1.00000000000E+00	0	0
5	0.5	5.5000000000000002E+00	5.5	+4.36E-17	3.14

TABLE 15

Double Precision Mixed Arithmetic
Operation: Multiplication (*)
VAX

X	Y	Approximate Value	True Value	Relative Error	$\|R.E./\varepsilon$
1.0E+12	0.5	5.00000000000039215E+11	5.0E+11	+7.843E-15	565.06
1.0E+08	0.5	5.00000000000006054E+07	5.0E+09	+1.211E-15	87.25
5.0E+06	1.0E-03	4.99999999999999432E+03	5.0E+03	-1.154E-16	8.31
5,000	1.0E-03	5.00000000000000222E+03	5	+4.440E-17	3.2
5,000	1.0E-08	4.99999999999999647E-05	5.0E-05	-7.060E-17	5.09
5	0.5	2.49999999999999944E+00	2.5	-2.240E-17	1.61
1	1.0E-05	9.99999999999993618E-06	1.0E-05	-6.382E-16	45.98
5	1.0E-08	5.00000000000016400E-08	5.0E-08	+3.280E-15	236.31
5	1.0E-10	4.99999999999996434E-10	5.0E-10	-7.132E-16	51.38
5	1.0E-12	4.99999999999998789E-12	5.0E-12	-2.422E-16	17.45

TABLE 16

Double Precision Mixed Arithmetic
Operation: Division (/)
VAX

X	Y	Approximate Value	True Value	Relative Error	$\lvert R.E./\varepsilon$
1.0E+12	0.5	2.000000000000012482E+12	2.0E+11	+6.24E-15	449.64
1.0E+10	0.5	1.999999999999997425E+10	2.0E+10	-1.29E-15	92.8
1.0E+08	0.5	2.000000000000006296E+08	2.0E+08	+3.15E-15	235.28
5.0E+06	0.01	4.999999999999995977E+08	5.0E+08	-8.05E-16	57.97
5,000	1.0E-04	5.000000000000006054E+07	5.0E+07	+1.21E-15	87.25
500	1.0E-08	5.000000000000002575E+10	5.0E+10	+5.15E-16	37.1
50	0.2	2.500000000000000071E+02	250	+2.84E-17	2.05
5	0.5	1.000000000000000000E+01	10	0	0
5	1.0E-05	5.000000000000002765E+05	5.0E+05	+5.53E-16	39.84
5	1.0E-12	5.000000000000000366E+12	5.0E+12	+7.32E-17	5.27

TABLE 17

Double Precision Small Arithmetic
Operation: Addition (+)
VAX

X	Y	Approximate Value	True Value	Relative Error	$\|R.E.\|$
0.5	0.1	5.99999999999999992E-01	0.6	-1.38E-17	1
0.6	0.4	1.0000000000000003E+00	1	2.80E-17	2.02
0.5	0.01	5.04999999999999991E-01	5.05E-01	-1.86E-17	1.34
0.5	1.0E-06	5.00000999999999997E-01	5.000001E-01	-2.58E-17	1.86
0.5	1.0E-10	5.00000009999999994E-01	5.0000000001E-01	-1.12E-17	0.81
0.5	2.0E-12	5.00000000001999997E-01	5.000000000001E-01	-5.20E-18	0.37
1.0E-05	4.0E-05	4.99999999999999965E-05	5.0E-05	-7.02E-17	5.06
1.0E-05	5.0E-10	1.00004999999999943E-05	1.00005E-05	-5.71E-16	41.14
1.0E-12	1.0E-12	1.99999999999998431E-12	2.0E-12	-7.85E-15	565.3
3.0E-12	2.0E-12	4.99999999999999889E-12	5.0E-12	-2.42E-16	17.45

TABLE 18

Double Precision Small Arithmetic
Operation: Multiplication (*)
VAX

X	Y	Approximate Value	True Value	Relative Error	$\|R.E.\|/\varepsilon$
0.2	0.8	1.59999999999999999E-01	0.16	-6.25E-19	4.5E-2
0.09	0.03	2.70000000000000035E-03	2.7E-03	+1.30E-17	0.93
0.03	1.0E-05	2.99999999999999997E-07	3.0E-07	-1.00E-18	0.72
0.03	1.0E-10	3.00000000000000041E-12	3.0E-12	+1.37E-17	0.98
0.03	1.0E-12	2.99999999999999996E-14	3.0E-14	-1.33E-18	0.1
3.0E-05	2.0E-05	5.99999999999999960E-10	6.0E-10	-6.67E-18	0.48
3.0E-04	3.0E-05	8.99999999999999940E-09	9.0E-09	-6.67E-18	0.48
3.0E-08	3.0E-04	9.00000000000000021E-12	9.0E-04	+2.33E-18	0.17
1.0E-10	5.0E-03	5.00000000000000026E-13	5.0E-13	+5.20E-18	0.37
0.5	3.0E-12	1.49999999999999995E-12	1.5E-12	-3.33E-18	0.24

TABLE 19

Double Precision Small Arithmetic
Operation: Division (/)
VAX

X	Y	Approximate Value	True Value	Relative Error	$\|R.E.\|/\varepsilon$
5.0E-12	2.0E-10	2.49999999999999835E-02	2.5E-02	-6.60E-17	4.76
5.0E-08	5.0E-05	9.99999999999999124E-04	1.0E-03	-8.76E-17	6.31
5.0E-04	0.02	2.49999999999999835E-03	2.5E-03	-6.60E-17	4.76
0.5	0.02	2.50000000000000222E+01	25	+8.88E-17	6.4
0.9	0.3	2.99999999999999944E+00	3	-1.87E-17	1.35
9.9E-02	9.9E-04	1.00000000000000320E+02	100	+3.20E-16	23.05
0.9	0.02	4.50000000000000178E+01	45	+3.96E-17	2.85
0.8	2.0E-05	4.00000000000001546E+04	4,000	+3.87E-16	27.85
5.5E-02	1.0E-05	5.50000000000001364E+03	5,500	+2.48E-16	17.87
0.5	2.0E-09	2.50000000000000000E+08	2.5E+08	0	0

TABLE 20

Machine Epsilon (ε) = 2.2204460492503131E-16
Double Precision Large Arithmetic
Operation: Addition (+)
IBM 3090

X	Y	Approximate Value	True Value	Relative Error	$\|R.E.\|/\varepsilon$
1.0E+12	5	1.0000000000004948000E+12	1,000,000,000,005	-4.80E-14	216.17
1.0E+10	5	1.0000000000499934000E+10	10,000,000,005	-6.60E-14	297.24
5.0E+06	4,000	5.0039999999999840000E+06	5,004,000	-3.20E-14	143.98
50,000	5	5.0004999999999667000E+04	50,005	-6.66E-14	299.91
50,000	400	5.0399999999999964000E+04	50,400	-7.14E-15	32.17
500	20	5.1999999999999978000E+02	520	-2.31E-15	10.39
500	400	8.9999999999999950000E+02	900	-5.56E-15	25.02
100	5	1.0499999999999995000E+02	105	-4.76E-15	21.45
20	20	3.9999999999999988000E+01	40	-3.00E-15	13.5
5	4	8.9999999999999995000E+00	9	-5.56E-16	2.5

TABLE 21

Double Precision Large Arithmetic
Operation: Multiplication (*)
IBM 3090

X	Y	Approximate Value	True Value	Relative Error	$\|R.E.\|/\varepsilon$
1.0E+12	5	4.99999999999963600E+12	5.0E+12	-7.280E-14	327.86
1.0E+10	5	4.99999999999969600E+10	5.0E+10	-6.080E-14	273.82
5.0E+06	4,000	1.99999999999993300E+10	2.0E+10	-3.350E-14	150.87
50,000	400	1.99999999999995500E+07	2.0E+07	-2.250E-14	101.33
50,000	5	2.49999999999997300E+05	250,000	-1.080E-14	48.64
500	400	1.99999999999998000E+03	200,000	-1.000E-14	45.04
500	20	9.99999999999994800E+05	10,000	-5.200E-14	234.19
100	5	4.99999999999997700E+02	500	-4.600E-15	20.72
20	20	3.99999999999998100E+02	400	-4.750E-15	21.39
5	4	1.99999999999999600E+01	20	-2.000E-15	9.01

TABLE 22

Double Precision Large Arithmetic
Operation: Division (/)
IBM 3090

X	Y	Approximate Value	True Value	Relative Error	$\|R.E.\|/\varepsilon$
1.0E+12	5	1.99999999999886000E+11	2.0E+11	-5.70E-14	256.24
1.0E+10	5	1.99999999999895000E+09	2.0E+09	-5.25E-14	236.44
5.0E+06	4,000	1.24999999999998000E+03	1,250	-1.60E-14	72.06
50,000	5	9.99999999999988000E+03	10,000	-1.20E-15	5.4
50,000	400	1.24999999999998000E+02	1.25	-1.60E-15	7.21
500	20	2.49999999999997000E+01	25	-1.20E-15	5.4
500	400	1.25000000000000000E+00	1.25	0	0
100	5	1.99999999999996000E+01	20	-2.00E-15	9.01
50	25	1.99999999999999000E+00	2	-5.00E-16	2.25
5	4	1.25000000000000000E+00	1.25	0	0

TABLE 23

Double Precision Mixed Arithmetic
Operation: Addition (+)
IBM 3090

X	Y	Approximate Value	True Value	Relative Error	$\|R.E.\|/\varepsilon$
1.0E+12	0.05	9.999999999999777E+11	1,000,000,000,000.05	-2.23E-14	100.43
1.0E+10	0.05	1.000000000004945E+10	10,000,000,000.05	-5.50E-14	247.7
1.0E+08	0.05	1.000000000499957E+08	100,000,000.05	-4.30E-14	193.65
5.0E+06	0.05	5.000000004999821E+06	5,000,000.05	-3.58E-14	161.23
500	5.0E-07	5.000000004999979E+02	5.000000005000E+02	-4.20E-15	18.92
50	1.0E-06	5.000000099999983E+01	5.000001000000E+01	-3.40E-15	15.31
50	1.0E-08	5.000000000999983E+01	5.000000010000E+01	-3.40E-15	15.31
5	1.0E-10	5.000000000099996E+00	5.000000000100E+00	-8.00E-16	3.6
1	5.0E-12	1.000000000004999E+00	1.000000000005E+00	-1.00E-15	4.5
5	0.5	5.499999999999994E+00	5.5	-1.09E-15	4.91

TABLE 24

Double Precision Mixed Arithmetic
Operation: Multiplication (*)
IBM 3090

X	Y	Approximate Value	True Value	Relative Error	$\|R.E\|/\varepsilon$
1.0E+12	0.5	4.999999999999773000E+11	5.0E+11	-4.54E-14	204.46
1.0E+08	0.5	4.999999999999803000E+07	5.0E+09	-3.94E-14	177.44
5.0E+06	1.0E-03	4.999999999999922000E+03	5.0E+03	-1.56E-14	70.26
5,000	1.0E-03	5.000000000000000000E+00	5	0	0
5,000	1.0E-08	5.000000000000005000E-05	5.0E-05	+1.00E-15	4.5
5	0.5	2.500000000000000000E+00	2.5	0	0
1	1.0E-05	1.000000000000004000E-05	1.0E-05	+4.00E-15	18.01
5	1.0E-08	5.000000000000111000E-08	5.0E-08	+2.22E-15	10
5	1.0E-10	5.000000000000171000E-10	5.0E-10	+3.42E-15	15.4
5	1.0E-12	5.000000000000178000E-12	5.0E-12	+3.56E-15	16.03

TABLE 25

Double Precision Mixed Arithmetic
Operation: Division (/)
IBM 3090

X	Y	Approximate Value	True Value	Relative Error	$\|R.E.\|/\varepsilon$
1.0E+12	0.5	1.999999999999888000E+11	2.0E+12	-5.60E-14	252.2
1.0E+10	0.5	1.999999999999905000E+10	2.0E+10	-4.75E-14	213.92
1.0E+08	0.5	1.999999999999879000E+08	2.0E+08	-6.05E-14	272.47
5.0E+06	0.01	4.999999999999791000E+08	5.0E+08	-4.18E-14	188.25
5,000	1.0E-04	4.999999999999839000E+07	5.0E+07	-3.32E-14	149.52
500	1.0E-08	4.999999999999696000E+10	5.0E+10	-6.08E-14	273.82
50	0.2	2.499999999999990000E+02	250	-4.00E-15	18.01
5	0.5	9.999999999999992000E+00	10	-8.00E-16	3.6
5	1.0E-05	4.999999999999947000E+05	5.0E+05	-1.06E-14	47.74
5	1.0E-12	4.999999999999636000E+12	5.0E+12	-7.28E-14	327.86

TABLE 26

Double Precision Small Arithmetic
Operation: Addition (+)
IBM 3090

X	Y	Approximate Value	True Value	Relative Error	$\|R.E\|/\varepsilon$
0.5	0.1	6.000000000000010E-01	0.6	1.67E-16	0.8
0.6	0.4	1.000000000000000E+00	1	0	0
0.5	0.01	5.100000000000000E-01	5.1E-01	0	0
0.5	1.0E-06	5.000010000000000E-01	5.000001E-01	0	0
0.5	1.0E-10	5.000000000100000E-01	5.0000000001E-01	0	0
0.5	2.0E-12	5.000000000001000E-01	5.000000000001E-01	0	0
1.0E-05	4.0E-05	5.000000000000050E-05	5.0E-05	1.00E-15	4.5
1.0E-05	5.0E-10	1.000050000000030E-10	1.00005E-05	3.00E-15	13.5
1.0E-12	1.0E-12	2.000000000000900E-12	2.0E-12	4.50E-15	2.03
3.0E-12	2.0E-12	5.000000000002670E-12	5.0E-12	5.34E-14	240.5

TABLE 27

Double Precision Small Arithmetic
Operation: Multiplication (*)
IBM 3090

X	Y	Approximate Value	True Value	Relative Error	$\|R.E.\|/\varepsilon$
0.2	0.8	1.60000000000000000E-01	0.16	0	0
0.09	0.03	2.69999999999999000E-03	2.7E-03	-3.70E-16	1.67
0.03	1.0E-05	2.99999999999999000E-07	3.0E-07	-3.33E-16	1.5
0.03	1.0E-10	2.99999999999999000E-12	3.0E-12	-3.33E-16	1.5
0.03	1.0E-12	2.99999999999999000E-14	3.0E-14	-3.33E-16	1.5
3.0E-05	2.0E-05	5.99999999999999000E-10	6.0E-10	-1.67E-16	1
3.0E-04	3.0E-05	8.99999999999998000E-09	9.0E-09	-2.22E-16	1
3.0E-08	3.0E-04	8.99999999999998000E-12	9.0E-04	-2.22E-16	1
1.0E-10	5.0E-03	5.00000000000000000E-13	5.0E-13	0	0
0.5	3.0E-12	1.50000000000000000E-12	1.5E-12	0	0

TABLE 28

Double Precision Small Arithmetic
Operation: Division (/)
IBM 3090

X	Y	Approximate Value	True Value	Relative Error	$\|R.E.\|/\varepsilon$
5.0E-12	2.0E-10	2.49999999999903000E-02	2.5E-02	-3.88E-14	174.74
5.0E-08	5.0E-05	1.00000000000001000E-03	1.0E-03	+1.00E-15	4.5
5.0E-04	0.02	2.50000000000003000E-03	2.5E-03	+1.20E-15	5.4
0.5	0.02	2.49999999999996000E+01	25	-1.60E-15	7.21
0.9	0.3	2.99999999999998000E+00	3	-6.67E-16	3
9.9E-02	9.9E-04	9.99999999999988000E+01	100	-1.20E-15	5.4
0.9	0.02	4.49999999999993000E+01	45	-1.56E-15	7.01
0.8	2.0E-05	3.99999999999987000E+04	4,000	-3.25E-15	14.64
5.5E-02	1.0E-05	5.49999999999997200E+03	5,500	-5.09E-15	22.93
0.5	2.0E-09	2.49999999999917000E+08	2.5E+08	-3.32E-14	149.52

APPENDIX B

TABLE 29

THE RANGE OF NUMBERS REPRESENTED IN THE SLI SYSTEM

THE LEVEL	THE INDEX	RECIPROCAL SIGN	THE REPRESENTED NUMBER
1	0	1	1
1	0.5	1	1.648721271E+00
1	0	-1	0.1
1	0.5	-1	6.065306596E-01
2	0	1	2.718281828E+00
2	0	-1	3.678794412E-01
2	0.5	1	5.200325765E+00
2	0.5	-1	1.922956455E-01
3	0	1	1.515426224E+01
3	0	-1	6.598803585E-02
3	0.5	1	1.813313036E+02
3	0.5	-1	5.514767611E-03
4	0	1	3.814279104E+06
4	0	-1	2.621727739E-07
4	0.5	1	5.638772181E+78
4	0.5	-1	1.773435720E-79
5	0	1	$\cong 10^{1660000}$
5	0	-1	$\cong (10^{1660000})^{-1}$
6	0	1	$\cong e^{10^{1660000}}$
6	0	-1	$\cong (e^{10^{1660000}})^{-1}$
7	0	1	$e^{e^{e^{e^{e^{e^{e^{0}}}}}}}$
7	0	-1	$(e^{e^{e^{e^{e^{e^{e^{0}}}}}}})^{-1}$

(There is no level zero in the sli system)

APPENDIX C

TABLE 30

THE RESULT OF THE ADDITION AND MULTIPLICATION IN THE SLI SYSTEM (SEQUENT COMPUTER)

LL	FF	MM	GG	(+) RESULT 1	(*) RESULT 2
2	0	2	0	2.526589034139044450E+00	2.693147180559945390E+00
3	0	3	0	3.204679142680520340E+00	3.526589034139044450E+00
3	0.5	3	0.5	3.573149765144861600E+00	3.850949094195776460E+00
4	0	4	0	4.016187505765744340E+00	4.204679142680520340E+00
4	0.5	4	0.5	4.500444723739581930E+00	4.573149765144862040E+00
5	0	5	0	5.000000004411472960E+00	5.016187505765744340E+00
5	0.5	5	0.5	5.500000000000000000E+00	5.500444723739581930E+00
6	0	6	0	6.000000000000000000E+00	6.000000004411472960E+00
6	0.5	6	0.5	6.500000000000000000E+00	6.500000000000000000E+00
7	0	7	0	7.000000000000000000E+00	7.000000000000000000E+00
7	0.5	7	0.5	7.500000000000000000E+00	7.500000000000000000E+00

LL: The level of x

FF: The index of x

MM: The level of y

GG: The index of y

Result 1: The result for $\psi(\phi(x)+\phi(y))$

Result 2: The result for $\psi(\phi(x)*\phi(y))$

APPENDIX D

PROGRAM

```
C ***************************************************************
C *          THE SYMMETRIC LEVEL-INDEX SYSTEM OF                *
C *                  COMPUTER ARITHMETIC                        *
C *                                                             *
C *    PURPOSE: TO IMPLEMENT FOUR BASIC OPERATIONS (ADDITION,   *
C *                SUBTRACTION, MULTIPLICATION, AND DIVISION) OF*
C *                 THE SLI ARITHMETIC.                         *
C *                                                             *
C *      INPUT: 1. X AND Y.                                     *
C *             2. CHOOSE OPERATION.                            *
C *     OUTPUT:  SHOW THE RESULT.                               *
C *                                                             *
C *  C. W. CLENSHAW AND F. W. J. OLVER, "LEVEL-INDEX ARITHMETIC *
C *        OPERATIONS." SOCIETY FOR INDUSTRIAL AND MATHEMATICS*
C *        JOURNAL OF NUMERICAL ANALYSIS, 24 (1987), PP.471-473.*
C *                                                             *
C *  C. W. CLENSHAW AND P. R. TURNER, "THE SYMMETRIC LEVEL-     *
C *        INDEX SYSTEM." INST. MATH. APPL. J. 8 (1988), PP.517-519.*
C *                                                             *
C *  CHING-WEI CHAO, COMPUTER SCIENCE DEPARTMENT,               *
C *        OKLAHOMA STATE UNIVERSITY, 1993.                     *
C *                                                             *
C ***************************************************************
      DOUBLE PRECISION X, Y, TPY, FF, GG, RESULT
      INTEGER ANS, SZ, AE, IN, LP, OP, LMS, RX, RY, LL, MM
      LOGICAL END, ERR
C
C     ******************
C     *  INITIALIZATION *
C     ******************
10    ERR = .FALSE.
      END = .FALSE.
      RESULT = 0.0D0
       LMS = 1
       SZ = 1
       AE = 0
       IN = 5
       LP = 6
C     ************************************
C     *  CHOOSE THE BASIC ARITHMETIC *
C     ************************************
        CALL MENU(NUM,X,Y,TPY,OP,LMS,END,RESULT,ERR,AE,SZ)
        IF (END)  THEN
           WRITE(LP,11)RESULT
11         FORMAT(' RESULT = ', 1PG25.18)
        ENDIF
        IF (ERR .OR. END) GO TO 12
C     ************************************************
```

```
C   * FIND OUT THE LEVEL & INDEX FROM X & Y *
C   ************************************************
      CALL FLMG(X, Y, RX, RY, FF, LL, MM, GG, LMS)
C   ******************************************
C   * SYMMETRIC LEVEL-INDEX ARITHMETIC *
C   ******************************************
      CALL SLI(NUM, X, Y, TPY, OP, LMS, RX, RY, FF, LL, MM, GG, AE, SZ)
12    WRITE(LP,13)
13    FORMAT( ' DO YOU WANT TO CONTINUE(1. YES/2. NO) ')
      READ(IN,14)ANS
14    FORMAT(I2)
      IF (ANS .EQ. 1)  GO TO 10
      STOP
      END
C
    SUBROUTINE MENU (NUM,X,Y,TPY,OP,LMS,END,RESULT,ERR,AE, SZ)
C *                                                                    *
C *  A MENU FOR CHOOSING BASIC ARITHMETIC (+, -, *, /)  AND INPUT X & Y.  *
C *                                                                    *
C *  INPUT QUANTITIES:                                                 *
C *     X: THE FIRST INPUT OPERAND                                     *
C *     Y: THE SECOND INPUT OPERAND                                    *
C *                                                                    *
C *  OUTPUT QUANTITIES:                                                *
C *     NUM: INPUT OPERATOR                                            *
C *     OP  :  THE OPERATION'S FLAG:                                   *
C *          OP = 1   MEANS    ADDITION  ALGORITHM                     *
C *          OP = 2   MEANS    SUBTRACTION  ALGORITHM                  *
C *     TPY: THE ORIGINAL OPERAND Y                                    *
C *     LMS: THE FLAG FOR LARGE, MIXED, OR SMALL OPERATION             *
C *           LMS = 1  MEANS  LARGE OR MIXED  ARITHMETIC               *
C *           LMS = 2  MEANS   SMALL  ARITHMETIC                       *
C *     END: THE FLAG FOR TERMINATING THE OPERATION                    *
C *           END = .TRUE.   MEANS  TERMINATING THE OPERATION          *
C *           END = .FALSE.   MEANS  CONTINUING THE OPERATION          *
C *     RESULT: THE RESULT OF THE OPERATION                            *
C *     ERR: THE FLAG FOR ERROR MESSAGE.                               *
C *          ERR = .TRUE.   MEANS   PRINTING THE ERROR MESSAGE         *
C *          ERR = .FALSE.  MEANS   NO  ERROR  HAPPENING               *
C *     AE : THE FLAG FOR X & Y TAKING  LOGARITHM ONCE BEFORE USING    *
C *         "SLI" SUBROUTINE IN DIVISION OR MULTIPLICATION OPERATION   *
C *           AE = 0   MEANS    NOT  TAKING                            *
C *           AE = 1   MEANS    TAKING                                 *
C *     SZ  : THE RESULT OF THE RECIPROCATION SIGN                     *
C *                                                                    *
C *  ANOTHER SUBROUTINES ARE CONTAINING:                               *
C *       1. SUBROUTINE ADD:   FOR ADDITION'S OPERATION                *
C *       2. SUBROUTINE SUBTR: FOR SUBTRACTION'S OPERATION             *
C *       3. SUBROUTINE MULT: FOR MULTIPLICATION'S OPERATION           *
C *       4. SUBROUTINE DIV:   FOR DIVISION'S OPERATION                *
C *                                                                    *
C *  CHING-WEI CHAO, COMPUTER SCIENCE DEPARTMENT,                      *
C *          OKLAHOMA STATE UNIVERSITY, 1993.                          *
C *                                                                    *
```

```
C ***********************************************************************
      DOUBLE PRECISION X, Y, RESULT, TPY
      INTEGER SZ, AE, NUM, OP, LMS, IN, LP
      LOGICAL END, ERR
C
      IN = 5
      LP = 6
      WRITE(LP,20)
20    FORMAT(' PLEASE INPUT X = ')
      WRITE(LP, 21)
21    FORMAT( '(Input in "E" or "D" Form with an Exponent must be Right Justified in the Format ')
      WRITE(LP,22)
22    FORMAT(' Field; Input in "F" Format (No Exponent) need Not be)  ')
      READ(IN,23)X
23    FORMAT(D20.10)
      WRITE(LP,24)
24    FORMAT(' PLEASE INPUT Y = ')
      WRITE(LP, 21)
      WRITE(LP, 22)
      READ(IN,23)Y
25    WRITE(LP,26)
26    FORMAT(' PLEASE INPUT THE OPERATOR ' )
      WRITE(LP,27)
27    FORMAT(' 1. ADDITION            2. SUBTRACTION ')
      WRITE(LP,28)
28    FORMAT(' 3. MULTIPLICATION  4. DIVISION ')
      READ(IN,29)NUM
29    FORMAT(I1)
C     *********************
C     *   ADDITION        *
C     *********************
      IF (NUM .EQ. 1) THEN
        CALL ADD(X, Y, END, OP, RESULT, SZ)
C     *********************
C     *  SUBTRACTION    *
C     *********************
      ELSE IF (NUM .EQ. 2) THEN
        CALL SUBTR(X, Y, END, OP, RESULT, SZ)
C     *********************
C     *  MULTIPLICATION *
C     *********************
      ELSE IF (NUM .EQ. 3) THEN
        CALL MULT(X, Y, TPY, END, OP, LMS, RESULT, SZ, AE)
C     *********************
C     *  DIVISION       *
C     *********************
      ELSE IF (NUM .EQ. 4) THEN
        CALL DIV(X, Y, TPY, END, ERR, OP, LMS, RESULT, SZ, AE, NUM)
      ELSE
        GO TO 55
      ENDIF
      RETURN
      END
C
```

```
      SUBROUTINE ADD(X, Y, END, OP, RESULT, SZ)
C *                                                                *
C *  PERFORMS ADDITION USING THE SLI SYSTEM OF                     *
C *  COMPUTER ARITHMETIC.                                          *
C *                                                                *
C *  INPUT QUANTITIES:                                             *
C *       X: THE FIRST INPUT OPERAND                               *
C *       Y: THE SECOND INPUT OPERAND                              *
C *                                                                *
C *  OUTPUT QUANTITIES:                                            *
C *      OP: THE OPERATION'S FLAG  (+,-)                           *
C *       = 1  MEANS  ADDITION  ALGORITHM                          *
C *       = 2  MEANS  SUBTRACTION  ALGORITHM                       *
C *      END: THE FLAG FOR TERMINATING THE OPERATION               *
C *       = .TRUE. MEANS TERMINATING THE OPERATION                 *
C *       = .FALSE. MEANS CONTINUING THE OPERATION                 *
C *      RESULT: THE RESULT OF THE OPERATION                       *
C *      SZ      : THE RESULT OF THE RECIPROCATION SIGN            *
C *                                                                *
C *  PURPOSE: 1. TO GET THE SIGN OF THE RESULT.                    *
C *              2. LET X GREATER THAN Y.                          *
C *              3. TO GET THE FLAG OF THE OPERATION               *
C *                                                                *
C *  C. W. CLENSHAW AND P. R. TURNER, "THE SYMMETRIC               *
C *         LEVEL-INDEX SYSTEM", INST. MATH. APPL.                 *
C *          J. 8 (1988), PP.517-519.                              *
C*                                                                 *
C *  CHING-WEI CHAO, COMPUTER SCIENCE DEPARTMENT,                  *
C *           OKLAHOMA STATE UNIVERSITY, 1993.                     *
C *                                                                *
C ******************************************************************
      DOUBLE PRECISION X, Y, TPY, RESULT
      INTEGER SZ, OP
      LOGICAL END
C
C     ******************
C     * SPECIAL CASES  *
C     ******************
      IF (X .EQ. 0 .AND. Y .EQ. 0) THEN
          RESULT = 0
          END = .TRUE.
          GO TO 30
      ELSE IF (X .EQ. 0) THEN
          RESULT= Y
          END = .TRUE.
      ELSE IF (Y .EQ. 0) THEN
          RESULT= X
          END = .TRUE.
      ENDIF
C     ************************
C     *  X GREATER THAN Y    *
C     ************************
      IF (DABS(X) .GE. DABS(Y)) THEN
         IF (X .GE. 0 .AND. Y .GE. 0) THEN
```

```
            OP = 1
          ELSE IF (X .LT. 0 .AND. Y .LT. 0) THEN
            OP = 1
            SZ = -1
          ELSE IF (X .GE. 0 .AND. Y .LT. 0) THEN
            OP = 2
          ELSE IF (X .LT. 0 .AND. Y .GT. 0) THEN
            SZ = -1
            OP = 2
          ENDIF
          X = DABS(X)
          Y = DABS(Y)
C     **********************
C     *  X SMALLER THAN Y  *
C     **********************
        ELSE IF (DABS(X) .LT. DABS(Y)) THEN
            IF (X .GE. 0 .AND. Y .GE. 0) THEN
              OP = 1
            ELSE IF (X .LT. 0 .AND. Y .LT. 0) THEN
              OP = 1
              SZ = -1
            ELSE IF (X .GE. 0 .AND. Y .LT. 0) THEN
              OP = 2
              SZ = -1
            ELSE IF (X .LT. 0 .AND. Y .GT. 0) THEN
              OP = 2
            ENDIF
            TPY = Y
            Y = DABS(X)
            X = DABS(TPY)
        ENDIF
30      RETURN
        END
C
      SUBROUTINE SUBTR(X, Y, END, OP, RESULT, SZ)
C *                                                              *
C *  PERFORMS ADDITION USING THE SLI SYSTEM OF                   *
C *  COMPUTER ARITHMETIC.                                        *
C *                                                              *
C *  INPUT QUANTITIES:                                           *
C *        X: THE FIRST INPUT OPERAND                            *
C *        Y: THE SECOND INPUT OPERAND                           *
C *                                                              *
C *  OUTPUT QUANTITIES:                                          *
C *        OP: THE OPERATION'S FLAG (+,-)                        *
C *         = 1  MEANS  ADDITION  ALGORITHM                      *
C *         = 2  MEANS  SUBTRACTION  ALGORITHM                   *
C *        END: THE FLAG FOR TERMINATE THE OPERATION             *
C *         = .TRUE. MEANS TERMINATING THE OPERATION             *
C *         = .FALSE. MEANS CONTINUING THE OPERATION             *
C *        RESULT: THE RESULT OF THE OPERATION                   *
C *        SZ      : THE RESULT OF THE RECIPROCATION SIGN        *
C *                                                              *
C *  PURPOSE: 1. TO GET THE SIGN OF THE RESULT.                  *
```

```
C *              2. LET X GREATER THAN Y.                          *
C *              3. TO GET THE FLAG OF THE OPERATION               *
C *                                                                *
C *  C. W. CLENSHAW AND P. R. TURNER, "THE SYMMETRIC               *
C *         LEVEL-INDEX SYSTEM." INST. MATH. APPL. J. 8 (1988),    *
C *          PP.517-519.                                           *
C *                                                                *
C *  CHING-WEI CHAO, COMPUTER SCIENCE DEPARTMENT,                  *
C *        OKLAHOMA STATE UNIVERSITY, 1993.                        *
C *                                                                *
C ******************************************************************
      DOUBLE PRECISION X, Y, TPY, RESULT
      INTEGER SZ, OP , LP
      LOGICAL END
C
      LP = 6
C     ******************
C     * SPECIAL CASES  *
C     ******************
      IF (X .EQ. Y) THEN
         WRITE(LP, 40)
40       FORMAT(' RESULT = 0 ')
         END = .TRUE.
         GO TO 45
      ENDIF
C     ************************
C     *  X GREATER THAN Y    *
C     ************************
      IF (DABS(X) .GT. DABS(Y)) THEN
         IF (X .GE. 0 .AND. Y .GE. 0) THEN
            OP = 2
         ELSE IF (X .LT. 0 .AND. Y .LT. 0) THEN
            OP = 2
            SZ = -1
         ELSE IF (X .GE. 0 .AND. Y .LT. 0) THEN
            OP = 1
         ELSE IF (X .LT. 0 .AND. Y .GT. 0) THEN
            OP = 1
            SZ = -1
         ENDIF
         X = DABS(X)
         Y = DABS(Y)
C     ************************
C     *  X SMALLER THAN Y    *
C     ************************
       ELSE IF (DABS(X) .LT. DABS(Y)) THEN
          IF (X .GE. 0 .AND. Y .GE. 0) THEN
             OP = 2
             SZ = -1
          ELSE IF (X .LT. 0 .AND. Y .LT. 0) THEN
             OP = 2
          ELSE IF (X .GE. 0 .AND. Y .LT. 0) THEN
             OP = 1
          ELSE IF (X .LT. 0 .AND. Y .GT. 0) THEN
```

```
            OP = 1
            SZ = -1
          ENDIF
          TPY = Y
          Y = DABS(X)
          X = DABS(TPY)
        ENDIF
45      RETURN
        END
C
        SUBROUTINE MULT(X, Y, TPY, END, OP, LMS, RESULT, SZ, AE)
C *                                                                  *
C *  PERFORMS MULTIPLICATION USING THE SLI SYSTEM OF                 *
C *  COMPUTER ARITHMETIC.                                            *
C *                                                                  *
C *  INPUT QUANTITIES:                                               *
C *       X: THE FIRST INPUT OPERAND                                 *
C *       Y: THE SECOND INPUT OPERAND                                *
C *                                                                  *
C *  OUTPUT QUANTITIES:                                              *
C *       OP  :  THE OPERATION'S FLAG (+,-)                          *
C *        = 1  MEANS  ADDITION  ALGORITHM                           *
C *        = 2  MEANS  SUBTRACTION  ALGORITHM                        *
C *       TPY: TO SAVE ORIGINAL Y OPERAND TEMPORARILY                *
C *       END: THE FLAG FOR TERMINATING THE OPERATION                *
C *        = .TRUE. MEANS TERMINATING THE OPERATION                  *
C *        = .FALSE. MEANS CONTINUING THE OPERATION                  *
C *       LMS: THE FLAG FOR LARGE, MIXED, OR SMALL OPERATION         *
C *        = 1  MEANS  LARGE OR MIXED ARITHMETIC                     *
C *        = 2  MEANS  SMALL ARITHMETIC                              *
C *       RESULT: THE RESULT OF THE OPERATION                        *
C *       SZ       : THE RESULT OF THE RECIPROCATION SIGN            *
C *      AE : THE FLAG FOR X & Y TAKING  LOGARITHM ONCE BEFORE       *
C *            USING "SLI" SUBROUTINE IN DIVISION OR                 *
C *            MULTIPLICATION OPERATION                              *
C *        = 0  MEANS  NOT  TAKING                                   *
C *        = 1  MEANS  TAKING                                        *
C *                                                                  *
C *  PURPOSE: 1. TO GET THE SIGN OF THE RESULT.                      *
C *              2. LET X GREATER THAN Y.                            *
C *              3. IF (X & Y GREATER THAN 1):                       *
C *                    X = DLOG(X)                                   *
C *                    Y = DLOG(Y)                                   *
C *                    OPERATION = "+"                               *
C *                 ELSE IF (X GREATER THAN 1 & Y LESS THAN 1)       *
C *                    C(1) = Y                                      *
C *                    OPERATION = "+"                               *
C *                 ELSE IF (X & Y SMALLER THAN 1)                   *
C *                    RESULT = X*Y                                  *
C *                                                                  *
C *  C. W. CLENSHAW AND F. W. J. OLVER, "LEVEL-INDIX ARITHMETIC      *
C *       OPERATIONS." SOCIETY INDUSTRIAL AND MATHEMATICS            *
C *       JOURNAL OF NUMERICAL ANALYSIS, 24 (1987), PP. 471-473.     *
C *                                                                  *
```

```
C *  CHING-WEI CHAO, COMPUTER SCIENCE DEPARTMENT,                    *
C *            OKLAHOMA STATE UNIVERSITY, 1993.                      *
C *                                                                  *
C ********************************************************************
      DOUBLE PRECISION X, Y, TPY, RESULT
      INTEGER SZ, AE, OP, LMS, LP
      LOGICAL END
C
      LP = 6
C     ******************
C     * SPECIAL CASES  *
C     ******************
      IF (X .EQ. 0 .OR. Y .EQ. 0) THEN
          WRITE(LP, 50)
50        FORMAT(' RESULT = 0 ')
          END = .TRUE.
          GO TO 55
      ENDIF
      OP = 1
      IF (X .GT. 0 .AND. Y .LT. 0)     SZ = -1
      IF (X .LT. 0 .AND. Y .GT. 0)     SZ = -1
C     **********************
C     *  X LARGER THAN Y   *
C     **********************
      IF (DABS(X) .GE. DABS(Y)) THEN
          X = DABS(X)
          Y = DABS(Y)
          IF (X .GE. 1 .AND. Y .GE. 1) THEN
              X = DLOG(X)
              Y = DLOG(Y)
              AE = 1
          ELSE IF (X .GE. 1 .AND. Y .LT. 1) THEN
              LMS = 2
              TPY = Y
          ELSE IF (X .LT. 1 .AND. Y .LT. 1) THEN
              RESULT = X*Y
              END = .TRUE.
          ENDIF
C     ***********************
C     *  X SMALLER THAN Y   *
C     ***********************
      ELSE IF (DABS(X) .LT. DABS(Y)) THEN
          TPY = Y
          Y = DABS(X)
          X = DABS(TPY)
          IF (X .GE. 1 .AND. Y .GE. 1) THEN
              X = DLOG(X)
              Y = DLOG(Y)
              AE = 1
          ELSE IF (X .GE. 1 .AND. Y .LT. 1) THEN
              LMS = 2
              TPY = Y
          ELSE IF (X .LT. 1 .AND. Y .LT. 1) THEN
              RESULT = X*Y
```

```
          END = .TRUE.
        ENDIF
      ENDIF
55    RETURN
      END
C
      SUBROUTINE DIV(X, Y, TPY, END, ERR, OP, LMS,RESULT,SZ,AE, NUM)
C *                                                                  *
C *  PERFORMS DIVISION USING THE SLI SYSTEM OF COMPUTER              *
C *  ARITHMETIC.                                                     *
C *                                                                  *
C *  INPUT QUANTITIES:                                               *
C *       X: THE FIRST INPUT OPERAND                                 *
C *       Y: THE SECOND INPUT OPERAND                                *
C *                                                                  *
C *  OUTPUT QUANTITIES:                                              *
C *       TPY: TO SAVE ORIGINAL OPERAND Y TEMPORARILY                *
C *       END: THE FLAG FOR TERMINATING THE OPERATION                *
C *        = .TRUE. MEANS TERMINATING THE OPERATION                  *
C *        = .FALSE. MEANS CONTINUING THE OPERATION                  *
C *       ERR: THE FLAG FOR ERROR MESSAGE.                           *
C *            ERR = .TRUE.  MEANS  PRINTING THE ERROR MESSAGE       *
C *            ERR = .FALSE.  MEANS  NO  ERROR  HAPPENING            *
C *       OP  :  THE OPERATION'S FLAG (+,-)                          *
C *           OP = 1  MEANS  ADDITION  ALGORITHM                     *
C *           OP = 2  MEANS  SUBTRACTION  ALGORITHM                  *
C *       LMS: THE FLAG FOR LARGE, MIXED, OR SMALL OPERATION         *
C *            LMS = 1   MEANS   LARGE  OR  MIXED  ARITHMETIC        *
C *            LMS = 2   MEANS    SMALL  ARITHMETIC                  *
C *       RESULT: THE RESULT OF THE OPERATION                        *
C *       SZ        : THE RESULT OF THE RECIPROCATION SIGN           *
C *       AE : THE FLAG FOR X & Y TAKING  LOGARITHM ONCE BEFORE      *
C *            USING "SLI" SUBROUTINE IN DIVISION OR                 *
C *            MULTIPLICATION OPERATION                              *
C *            AE = 0   MEANS    NOT  TAKING                         *
C *            AE = 1   MEANS    TAKING                              *
C *       NUM : THE FLAG FOR THE DIVISION USING MULTIPLICATION       *
C *                  OPERATION                                       *
C *                                                                  *
C *  PURPOSE: 1. TO GET THE SIGN OF THE RESULT.                      *
C *               2. IF (X & Y LESS THAN 1)                          *
C *                     Z = DLOG(X)+DLOG(1/Y)                        *
C *                  ELSE IF (X GREATER THAN Y)                      *
C *                     IF (Y GREATER THAN 1)                        *
C *                        X = DLOG(X)                               *
C *                        Y = DLOG(Y)                               *
C *                        OPERATION = "-"                           *
C *                  ELSE IF (X GREATER THAN 1 & Y LESS THAN 1)      *
C *                        X = DLOG(X)                               *
C *                        Y = DLOG(1/Y)                             *
C *                        OPERATION = "+"                           *
C *                  ELSE IF (Y GREATER THAN X)                      *
C *                        SIMILAR AS ABOVE                          *
C *                                                                  *
```

```
C *  C. W. CLENSHAW AND F. W. J. OLVER, "LEVEL-INDEX ARITHMETIC *
C *        OPERATIONS." SOCIETY INDUSTRIAL AND MATHEMATICS     *
C *         JOURNAL OF NUMERICAL ANALYSIS, 24 (1987), PP. 471-473.  *
C *                                                             *
C *  CHING-WEI CHAO, COMPUTER SCIENCE DEPARTMENT,               *
C *           OKLAHOMA STATE UNIVERSITY, 1993.                  *
C *                                                             *
C ***************************************************************
      DOUBLE PRECISION X, Y, TPY, RESULT
      INTEGER SZ, AE, OP, LMS , NUM, LP
      LOGICAL END, ERR
C
       LP = 6
       IF (X .GT. 0 .AND. Y .LT. 0)     SZ = -1
       IF (X .LT. 0 .AND. Y .GT. 0)     SZ = -1
C      ********************
C      * SPECIAL CASES  *
C      ********************
       IF (Y .EQ. 0) THEN
          WRITE(LP, 60)
60        FORMAT(' ERROR! DIVIDED BY ZERO ' )
          ERR = .TRUE.
          GO  TO 65
       ELSE IF (X .EQ. 0) THEN
           RESULT = 0
           END = .TRUE.
          GO  TO 65
      ELSE IF (DABS(X) .EQ. DABS(Y)) THEN
           RESULT = SZ*1
           END = .TRUE.
           GO  TO 65
      ENDIF
      X = DABS(X)
      Y = DABS(Y)
      OP = 1
      LMS = 1
C     ***********************
C     *  X & Y LESS THAN 1  *
C     ***********************
       IF (X .LE. 1 .AND. Y .LE. 1) THEN
           Y = 1.0D0/Y
           LMS = 2
          IF (DABS(X) .GE. DABS(Y)) THEN
              TPY = Y
           ELSE
              TPY = X
           ENDIF
           NUM = 3
C     ***********************
C     *  X GREATER THAN Y   *
C     ***********************
       ELSE IF (X .GE. Y) THEN
          IF (Y. GT. 1) THEN
             X = DLOG(X)
```

```
            Y = DLOG(Y)
            OP = 2
            AE = 1
          ELSE IF (X .GT. 1 .AND. Y .LE. 1) THEN
            X = DLOG(X)
            Y = DLOG(1.0D0/Y)
            AE = 1
          ENDIF
C     ******************
C     * X LESS THAN Y  *
C     ******************
      ELSE IF (X .LT. Y) THEN
        IF (X .GT. 1 .AND. Y .GT. 1) THEN
          Y = 1.0D0/Y
          TPY = Y
          LMS = 2
        ELSE IF (X .LE. 1 .AND. Y .GT. 1) THEN
          Y = 1.0D0/Y
          RESULT = X*Y
          END = .TRUE.
        ENDIF
      ENDIF
65    RETURN
      END
C
      SUBROUTINE  FLMG(X, Y, RX, RY, FF, LL, MM, GG, LMS)
C *                                                                    *
C *  1. TO GET R(X) & R(Y) USING SLI SYSTEM OF COMPUTER ARITHMETIC.    *
C *  2. TO GET THE INDEX & THE LEVEL FROM X & Y USING SLI SYSTEM       *
C *     OF COMPUTER ARITHMETIC.                                        *
C *                                                                    *
C *   INPUT QUANTITIES:                                                *
C *        X: THE FIRST INPUT OPERAND                                  *
C *        Y: THE SECOND INPUT OPERAND                                 *
C *                                                                    *
C *  OUTPUT QUANTITIES:                                                *
C *        RX: THE SIGN OF X                                           *
C *        RY: THE SIGN OF Y                                           *
C *        FF: THE INDEX OF X                                          *
C *        LL: THE LEVEL OF X                                          *
C *        MM: THE LEVEL OF Y                                          *
C *        GG:  THE INDEX OF Y                                         *
C *        LMS: THE FLAG FOR LARGE, MIXED, OR SMALL OPERATION          *
C *         = 1  MEANS  LARGE OR MIXED ARITHMETIC                      *
C *         = 2  MEANS   SMALL ARITHMETIC                              *
C *                                                                    *
C *  C. W. CLENSHAW AND P. R. TURNER, "THE SYMMETRIC LEVEL-INDEX       *
C *           SYSTEM." INST. MATH. APPL.  J. 8 (1988), PP.517-519.     *
C *                                                                    *
C *  CHING-WEI CHAO, COMPUTER SCIENCE DEPARTMENT,                      *
C *             OKLAHOMA STATE UNIVERSITY, 1993.                       *
C *                                                                    *
C ************************************************************************
      DOUBLE PRECISION X, Y, FF, GG, TT
```

```
      INTEGER RX, RY, LMS, LL, MM
C
C    ***************
C    * LET X .GE. Y  *
C    ***************
     IF (X .LT. Y) THEN
        TT = X
        X = Y
        Y = TT
     ENDIF
C    ************
C    * GET R(X) *
C    ************
      IF (X .GE. 1) THEN
         RX = 1
      ELSE
         RX = -1
      ENDIF
C    ************
C    * GET R(Y) *
C    ************
      IF (Y .GE. 1) THEN
         RY = 1
      ELSE
         RY = -1
     ENDIF
C    ****************************************************************
C    * TO FIND OUT WHICH ARE LARGE #S, MIXED #S, OR SMALL #S        *
C    * ARITHMETIC (LMS).                                            *
C    ****************************************************************
     IF (RX .LT. 1 .AND. RY .LT. 1) THEN
         LMS = 3
     ELSE IF (LMS .NE. 2) THEN
         LMS = 1
     ENDIF
C    ****************************************************************
C    *  TO GET THE LL, FF, GG, AND MM FROM THE SMALL NUMBER.*
C    ****************************************************************
      LL = 0
      MM = 0
C    ****************************************
C    * TO GET THE LEVEL & THE INDEX FROM X  *
C    ****************************************
70   IF (X .GE. 1) THEN
         X = DLOG(X)
         LL = LL+1
         GO TO 70
      ELSE IF (LL .EQ. 0) THEN
         IF (X .NE. 0)     X = 1.0D0/X
         IF (X .GE. 1) THEN
71          LL = LL+1
            X = DLOG(X)
         ENDIF
```

```
          IF (X .GE. 1) GO TO 71
        ENDIF
        FF = X
C       ******************************************
C       *  THE GET THE LEVEL & THE INDEX FROM Y. *
C       ******************************************
72      IF (Y .GE. 1) THEN
            Y = DLOG(Y)
            MM = MM+1
            GO TO 72
        ELSE IF (MM .EQ. 0) THEN
            IF (Y .NE. 0) Y = 1.0D0/Y
            IF (Y .GE. 1) THEN
73              MM = MM+1
                Y = DLOG(Y)
            ENDIF
            IF (Y .GE. 1)  GO TO 73
        ENDIF
        GG = Y
        RETURN
        END
C
      SUBROUTINE SLI(NUM,X,Y,TPY,OP,LMS,RX,RY,FF,LL,MM,GG,AE,SZ)
C *                                                                  *
C *   INPUT QUANTITIES:                                              *
C *        NUM: INPUT OPERATOR                                       *
C *        X: THE FIRST INPUT OPERAND                                *
C *        Y: THE SECOND INPUT OPERAND                               *
C *        TPY: THE ORIGINAL OPERAND Y                               *
C *        OP: THE  OPERATION'S FLAG (+,-)                           *
C *         = 1  MEANS  ADDITION  ALGORITHM                          *
C *         = 2  MEANS  SUBTRACTION  ALGORITHM                       *
C *       LMS: THE FLAG FOR LARGE, MIXED, OR SMALL OPERATION         *
C *         = 1   MEANS   LARGE  OR  MIXED  ARITHMETIC               *
C *         = 2   MEANS    SMALL  ARITHMETIC                         *
C *        RX: THE SIGN OF X.                                        *
C *        RY: THE SIGN OF Y.                                        *
C *        FF: THE INDEX OF X.                                       *
C *        LL: THE LEVEL OF X.                                       *
C *        MM: THE LEVEL OF Y.                                       *
C *        GG:  THE INDEX OF Y.                                      *
C *       AE : THE FLAG FOR X & Y TAKING  LOGARITHM ONCE BEFORE      *
C *             USING "SLI" SUBROUTINE IN DIVISION OR MULTIPLICATION *
C *              OPERATION                                           *
C *          = 0   MEANS    NOT  TAKING                            *
C *          = 1   MEANS    TAKING                                 *
C *        SZ : THE RESULT OF THE RECIPROCATION SIGN                 *
C *                                                                  *
C *   ALGORITHM:                                                     *
C *   STEP 1: SET R(Z)=R(X)                                          *
C *   STEP 2: GET THE SEQUENCES OF {A...}.                           *
C *  STEP 3: ACCORDING TO THE LARGE #S, MIXED #S, OR SMALL #S TO     *
C *            GET THE SEQUENCES OF {B...} AND C(1).                 *
C *   STEP 4, 5: TO GET THE RESULT.                                  *
```

```
C *                                                                    *
C *   C. W. CLENSHAW & P. R. TURNER, "THE SYMMETRIC LEVEL-INDEX        *
C *         SYSTEM." INST. MATH. APPL. J. 8 (1988), PP. 517-521.       *
C *                                                                    *
C *   CHING-WEI CHAO, COMPUTER SCIENCE DEPARTMENT,                     *
C *           OKLAHOMA STATE UNIVERSITY, 1993.                         *
C *                                                                    *
C ********************************************************************
DIMENSION A(10), B(10), D(10), H(10), C(10)
      DOUBLE PRECISION X, Y, Z, C1, C2, TPY, FF, GG, RESULT
      DOUBLE PRECISION  A, B, C, D, H, TEMP, TEMPA, TEMPB
      INTEGER SZ, AE, OP, LMS, RX, RY, LL, MM, EE, I, J, K, LP
C
C     **********************
C     ** INITIALIZATION **
C     **********************
      I = 1
      LP = 6
      C1 = 0.0D0
      C2 = 0.0D0
      Z = 0.0D0
 90   A(I) = 0.0D0
      B(I) = 0.0D0
      D(I) = 0.0D0
      C(I) = 0.0D0
      H(I) = 0.0D0
      I = I+1
     IF (I .LE. 10)  GO TO 90
C     ****************************
C     *  STEP 1: SET R(Z)=R(X)       *
C     ****************************
C
100   RZ = RX
C     ******************************************
C     *  STEP 2: GET THE SEQUENCES OF {A...}.     *
C     ******************************************
C
200   A(LL) = DEXP(-FF)
       IF (LL .GE. 2) THEN
           DO 210  J = 2, LL
                K = LL+2-J
                A(K-1) = DEXP(-1.0D0/A(K))
210        CONTINUE
      ENDIF
C     **********************
C     *  LARGE NUMBERS     *
C     **********************
      IF (RX .EQ. 1 .AND. RY .EQ. 1) THEN
          B(MM) = A(MM)*DEXP(GG)
                K = MM+2-J
                B(K-1) = DEXP((B(K)-1)/A(K))
220       CONTINUE
          IF (OP .EQ. 1) THEN
              C2 = 1+B(1)
```

```
            C2 = 1-B(1)
          ENDIF
C     **********************
C     *  MIXED NUMBERS     *
C     **********************
      ELSE IF (RX*RY .EQ. -1)  THEN
          B(MM) = DEXP(-GG)
          IF (MM .GE. 2) THEN
              DO 230 J = 2, MM
                  K = MM+2-J
                  B(K-1) = DEXP(-1.0D0/B(K))
230           CONTINUE
          ENDIF
          IF (LMS .EQ. 2) THEN
              C2 = TPY
          ELSE IF (OP .EQ. 1) THEN
              C2 = 1+A(1)*B(1)
          ELSE IF (OP .EQ. 2) THEN
              C2 = 1-A(1)*B(1)
          ENDIF
C     **********************
C     *  SMALL NUMBERS     *
C     **********************
      ELSE IF (RX .EQ. -1 .AND. RY .EQ. -1) THEN
          IF (MM .GT. LL) THEN
              D(MM) = DEXP(-GG)
              I = LL+1
              IF (MM .GE. I) THEN
                  DO 240 J = I, MM
                      K = MM+I-J
                      D(K-1) = DEXP(-1.0D0/D(K))
240               CONTINUE
              ENDIF
              B(LL) = D(LL)/A(LL)
          ELSE
              TEMP = MM-LL
              TEMP = DEXP(TEMP)
              B(LL) = DEXP(FF-TEMP*GG)
          ENDIF
          IF (LL .GE. 2) THEN
              DO 260 J = 2, LL
                  K = LL+2-J
                  TEMPA = B(K)-1
                  TEMPB = A(K)*B(K)
                  B(K-1) = DEXP(TEMPA/TEMPB)
260           CONTINUE
          ENDIF
          IF (OP .EQ. 1) THEN
              C1 = 1+B(1)
          ELSE IF (OP .EQ. 2) THEN
              C1 = 1-B(1)
          ENDIF
      ENDIF
C     ************************************************
```

```
C   *  STEP 3: GET THE SEQUENCES OF {B...} & C(1)      *
C   ************************************************
C
300   J = 1
C     ****************************************
C     *  FOR LARGE NUMBERS & MIXED NUMBERS  *
C     ****************************************
      IF (LMS .EQ. 1 .OR. LMS .EQ. 2) THEN
         IF (C2 .LT. A(1)) THEN
             RZ = -1
             TEMP = C2/A(1)
             H(1) = -DLOG(TEMP)
             IF (H(1) .LT. 0)      H(1) = 0
            GO TO 500
         ELSE IF (LL .EQ. 1) THEN
            H(1) = FF+DLOG(C2)
            IF (H(1) .LT. 0)      H(1) = 0
           GO TO 500
         ELSE
           C(2) = 1+A(2)*DLOG(C2)
         ENDIF
C     **************************
C     *  FOR SMALL NUMBERS     *
C     **************************
      ELSE
         IF (A(1)*C1 .GT. 1) THEN
             RZ = 1
            Z = 1+DLOG(A(1)*C1)
            GO TO 600
        ELSE IF (LL .EQ. 1) THEN
            IF (C1 .GT. 0) H(1) = FF-DLOG(C1)
            IF (C1 .LE. 0) H(1) = FF
            IF (H(1) .LT. 0)   H(1) = 0
            GO TO 500
        ELSE
            C(2) = 1-A(2)*DLOG(C1)
        ENDIF
      ENDIF
C     ************************************************************
C     *  STEP 4: TO GET THE LEVEL & THE INDEX OF THE RESULT.  *
C     ************************************************************
C
400    J = 2
410   IF (LL .GT. 2) THEN
          IF (C(J) .LT. A(J)) THEN
              Z = J+C(J)/A(J)
              GO TO 600
          ELSE
              C(J+1) = 1+A(J+1)*DLOG(C(J))
          ENDIF
          J = J+1
          IF (J .LE. LL-1)        GO TO 410
       ENDIF
       IF (C(LL) .LT. A(LL)) THEN
```

```
          Z = LL-1 + C(LL)/A(LL)
          GO TO 600
          H(LL) = FF+DLOG(C(LL))
          J = LL
        ENDIF
C   ****************************
C   *    STEP 5: GET THE RESULT  *
C   ****************************
C
500   CNT = 0
510   IF (H(J) .GE. 0 .AND. H(J) .LT. 1) THEN
          Z = CNT+J+H(J)
        ELSE
          H(J) = DLOG(H(J))
          CNT = CNT+1
          GO TO 510
        ENDIF
600   EE = 0
610   IF (Z .GT. 1) THEN
          EE = EE+1
          Z = Z-1
          GO TO 610
        ENDIF
        RESULT = Z
        IF (EE .GE. 1) THEN
          DO 620  K = 1, EE
              RESULT = DEXP(RESULT)
620       CONTINUE
        ENDIF
        IF (NUM .EQ. 3 .AND. LMS .EQ. 1)     GO TO  800
800   RESULT = RESULT**RZ
        IF (AE .EQ. 1)            RESULT = DEXP(RESULT)
        RESULT = RESULT*SZ
        WRITE(LP,810)RESULT
810   FORMAT(' RESULT= ', 1PG25.18)
        RETURN
        END
```

VITA 2

Ching-Wei Chao

Candidate for the Degree of

Master of Science

Thesis: THE SYMMETRIC LEVEL-INDEX SYSTEM OF COMPUTER ARITHMETIC

Major Field: Computer Science

Biographical:

Personal Data: Born in Tainan, Taiwan, Republic of China, March 28, 1963, the son of Chu-Chung Chao and Chu-Fong Hsin.

Education: Earned a bachelor of Science in Computing and information science from Oklahoma State University, May 1992; completed requirements for the Master of Science degree at Oklahoma State University in May, 1994.

Professional Experience: Programmer, Yu-Mei Computer Company, from October 1988 to October 1989.