Data Driven Calculation Histories to Minimize IEEE-754 Floating-point Computational Error

by

Lawrence E. Shafer

A Final Dissertation Report
submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy

The Graduate School of Computer and Information Sciences
Nova Southeastern University

2004

NSU NOVA SOUTHEASTERN UNIVERSITY
Graduate School of Computer and Information Sciences

DISSERTATION APPROVAL FORM
Doctoral Programs

CANDIDATE'S NAME Larry Shafer **SSN** ____________

PROGRAM (circle one) **DCTE DISS DISC DCIS [✓ CISD]** **Email** shaferl @nsu.nova.edu

DEGREE (circle one) **[✓ Ph.D.] Ed.D.**

DISSERTATION TOPIC Data Driven Calculation Histories to Minimize IEEE-754 Floating-point Computational Error.

☒ **Final Dissertation Report** We hereby approve the candidate's Final Dissertation Report (attached).

ADVISOR/CHAIR Michael Laszlo ____________ 11/22/04
Name (print) Signature Date

MEMBER ____________
Name (print) Signature Date

MEMBER Sumitra Mukherjee ____________ 11/22/04
Name (print) Signature Date

Junping Sun ____________ 11/22/04

------------Filed with the Program Office------------

ASSISTANT DEAN/ DIRECTOR Dr. Eric S. Ackerman ____________ 12/8/04
Name Signature Date

DEAN Dr. Edward Lieblein ____________ 12-9-04
Name Signature Date

An Abstract of a Dissertation Submitted to Nova Southeastern University in Partial Fulfillment of the Requirements for the Degree of Doctor of Philosophy

Data Driven Calculation Histories to Minimize IEEE-754 Floating-point Computational Error

by
Lawrence E. Shafer

November 17, 2004

The widely implemented and used IEEE-754 Floating-point specification defines a method by which floating-point values may be represented in fixed-width storage. This fixed-width storage does not allow the exact value of all rational values to be stored. While this is an accepted limitation of using the IEEE-754 specification, this problem is compounded when non-exact values are used to compute other values. Attempts to manage this problem have been limited to software implementations that require special programming at the source code level. While this approach works, the problem coder must be aware of the software and explicitly write high-level code specifically referencing it. The entirety of a calculation is not available to the special software so optimum results can not always be obtained when the range of operand values is large. This dissertation proposes and implements an architecture that uses integer algorithms to minimize precision loss in complex floating-point calculations. This is done using run-time calculation operand values at a simulated hardware level. These calculations are coded in a high-level language such that the coder is not knowledgeable about the details of how the calculation is performed.

Acknowledgements

I am grateful to my dissertation advisor, Dr. Michael Laszlo, for his patience in reviewing my ideas and papers and for his excellent remarks, comments, and suggestions helping me to refine my work.

I also thank my dissertation committee members, Dr. Junping Sun and Dr. Sumitra Mukherjee for their review of my work and their comments helping me to refine it.

Special thanks go to the Graduate School of Computer and Information Sciences for its understanding and granting me the time extensions necessary to complete this work.

I thank also Dr. Jon Inouye and Dr. S. Ken Burnell for their moral support. They helped to keep me focused.

I am grateful to my employer, the Fort Lauderdale office of Gabriel, Roeder, Smith and Company, for providing me a place to work when noises in my residential neighborhood became distracting.

Much of the credit toward my developing the topic of precision in floating-point calculations goes to the late Dr. Rollie Guild. Dr. Guild helped me take a rough idea and focus on it so I could eventually refine it to a subject I felt worthwhile for this work.

Table of Contents

Chapters

Appendix

List of Figures

List of Tables

List of Listings

Chapter 1

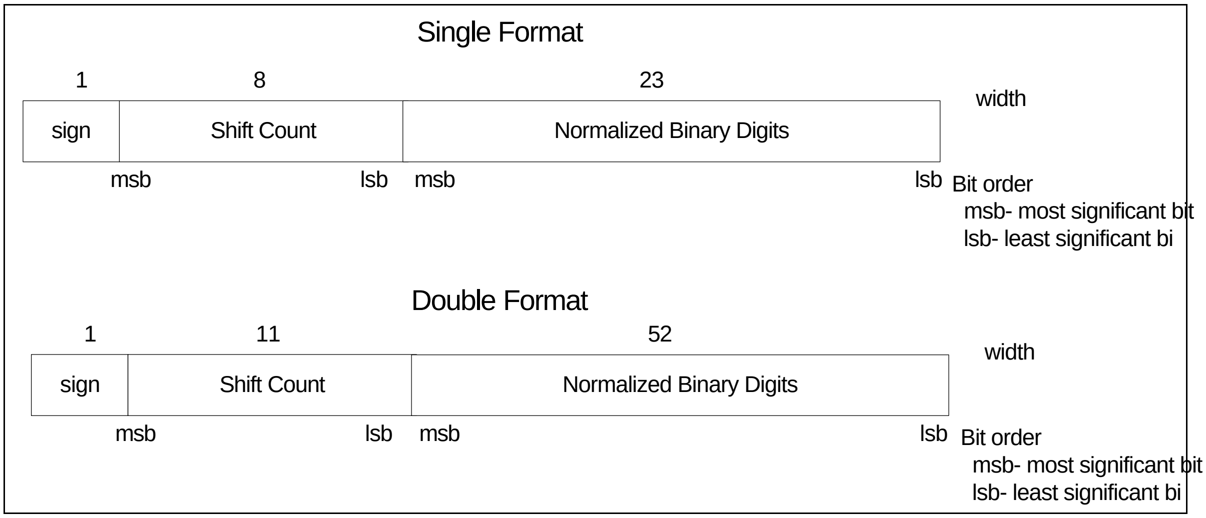
Introduction

Problem Statement and Goal

Numerical computing is necessary to all scientific and technical disciplines, including physics and engineering, disciplines in which numerical accuracy is paramount. In engineering, for example, stresses on a load-bearing column are computed using numerical methods. If the computational errors fall beyond acceptable limits, the column may collapse and cause damage to property and loss of life could result.

The ANSI/IEEE-754 standard [ANSI/IEEE] ("IEEE-754" or "Standard") is the most common floating-point implementation in digital computing machines. This standard defines two binary storage formats, similar in structure, for the storage of floating-point values. The *single* format is 32 bits wide (see Figure 1). The most significant bit is one if the floating-point value is negative, otherwise it is zero. The next most significant eight bits are a biased value that is a shift count for the data value in the remaining (least significant) 23 binary digits. A one bit is assumed before the last 23 bits since the Standard requires the values be stored normalized in this fashion.

Figure 1- IEEE-754 Floating-point Formats



The second storage format is the *double* format that is a total of 64 bits wide (see Figure 1). The only differences between the double format and the single format are that in the double format 11 bits are allotted for the shift count and 52 bits are set aside for the binary representation of the floating-point value.

The fundamental problem with calculations performed on digital computing machines involves the storage of floating point numbers in fixed-width storage elements. The fixed-width storage requirement of the Standard offers only an approximation to irrational numbers and rational numbers whose decimal portion does not terminate or is too long. This results in a truncation of digits beyond the capacity of the storage element and a resultant loss of precision in representing the value of that number. The difference between the desired value and the actual represented value can be quantified as *error*.

Error occurs in two phases of the programming/calculation cycle. A compiler must convert what are usually radix 10 floating-point values to the binary representation of the IEEE-754 standard. Unless the fractional portion can be multiplied by an integral power of two resulting in a value that can be contained in the floating-point storage width without any truncated bits, the compiler must truncate or round the value and precision is lost.

Precision is also lost during run time in the floating-point processor when an operation on one or two operands results in a floating-point value that is wider than its storage width allows. This often happens in multiplications (a division is multiplication by a mathematical inverse) that create a result that requires storage as wide as the sum of the widths of the values. It also occurs in additions (an addition is also a subtraction of values of different signs) since additions create a result that must span the width of the most significant bit of either operand and the least significant bit of either operand. This type of error is considered a *precision problem*.

Precision problems occur in additions if the shift counts are different. The storage width of the exact value of the result is the original storage width plus the difference in the shift counts (because it is necessary to get the “decimal points” to line up). If the difference in shift counts is greater than the storage width of the result, the floating-point value with the lesser shift count will be appended at the end of the other value and will have no effect on the result. Otherwise, bits will be truncated and precision will be lost. The optimum sequence of value to be multiplied can only be determined during run time when their values are known. The process of re-ordering a floating-point calculation based on the value of its operands is called *dynamic computational readjustment*.

A similar precision problem occurs when multiplying two floating-point values. Their product may require twice the storage width of a single floating-point value. Storage overflow occurs when storing the results in the storage width of a single floating-point value. Error can be predicted by breaking each floating-point operand into a two element vector with the first element being the more significant half of the storage bits and the second element being the remaining (lesser significant) half of the storage bits. The greater the sum of the inner cross-products for the multiplicands of a given multiplication, the greater the error. A series of multiplications would have to be reordered based upon such an error test to minimize floating-point error. This can only be done during run time when the values of the operands are known. This is a part of dynamic computation readjustment.

Imprecision is also introduced by formulaic expressions. For example, if *a* and *b* were sufficiently close to each other, the expression (*a* + *b*) * (*a* - *b*) would be computed more precisely as (*a***2 - *b***2) [Goldberg]. Determining the more accurate expression is not possible if a compiler does not know the values of *a* or *b*, but can be accomplished only during run time when the values of *a* and *b* are known. Modifying the expression to fit the data is called *adaptive reformulation*.

The precision problem is complicated when a floating-point value containing a loss of precision is used in subsequent arithmetic operations, since the error may propagate and grow as more operations are performed. This sort of error is known as *propagation error*. In quadrature problems, for example, propagation errors may grow quite rapidly unless steps are taken to remedy the situation [Sterbenz].

Floating-point computations are performed serially from floating-point instructions generated by a compiler from source code. Source code symbolically represents the floating-point calculations and operations that are to be performed. For example, consider the assignment operation $a = (b + c) / 2$. On first glance, it might seem that whatever the values of *b* and *c* are, the value stored in *a* would be accurate. The problem arises from the fact that the mapping of real numbers to a set of digital floating-point values is many-to-one. If *b* is the digital floating-point value adjacent to *c* in the digital floating-point set, *a* in this calculation would be either the value *b* or *c*, depending upon the rounding/truncation policy of the floating-point engine.

It is possible, however, that *b* and *c* are themselves imprecise due to previous calculations. If the sequence used to compute their values and the values of the operands used were known, it might be possible to determine a more precise result. This could be done if the calculation sequence and the values of the operands of the calculation sequence were stored. This stored sequence of operations is called a *calculation history*.

My hypothesis is that by maintaining floating-point calculation histories it is possible to minimize imprecision of a given floating-point calculation. A floating-point processor can use the calculation history to dynamically restructure a calculation by dynamic computation readjustment and adaptive functional restructuring. The broad goal of this dissertation is to explore this hypothesis.

The specific goal of this dissertation is to develop the theory behind a general data-driven computing architecture using calculation histories. Calculation histories would be used to minimize computational error in floating-point systems using the IEEE-754 Standard. This theory would be implemented in a test system. This dissertation also reports on the effectiveness of this implementation of floating-point calculation histories and the promise of this technique generally.

Relevance and Need for the Study

A natural use of electronic computing machinery is in mathematical computations. Employing electronic machines to perform routine tasks greatly speeds up their completion. There is a natural tendency to rely on machines to do one's work whenever possible. Too often, there is blind reliance on the results of employing machines when the results seem reasonable, which may lead to catastrophic results when the calculated results are not sufficiently precise.

Two classes of electronic computing machines are in common use. Analog computing machines deal with a continuous range of values; any value within their useful range is possible. Digital computing machines deal with sets of incremental values. These machines are more commonplace, less expensive, and more flexible. Numeric values in digital machines are either wholly integer or real; real numbers are the sum of an integer and a fractional portion (although the fractional portion and/or integer portion may be zero). The difference in value of adjacent elements in the set is the value of the least significant storage bit. This dissertation concerns itself only with digital computing machines.

Intel, a longtime major fabricator of digital processing chips, described its first dedicated processor for real values in a manual published in 1981[Rash]. This manual describes floating-point formats for real values that eventually became standardized in the IEEE-754 Standard [ANSI/IEEE]. The availability of cheap memory, fast processors, and high-speed floating-point division algorithms has made the IEEE-754 floating-point formats most commonly used in mathematical calculations. Along with the storage formats of real numbers, the Intel manual described the floating-point processor's instruction set. This freed programmers from having to rely on software procedures since a floating-point instruction set had been incorporated in hardware. The Intel floating-point unit has wide processing registers. The Standard relies on narrower fixed-width storage; this is the cause of the problem that will be investigated by this dissertation.

While it is possible to program floating-point calculations directly using a floating-point processor's instruction set, it is much easier to use a high-level programming language. A compiler for a high-level programming language could take mathematical expressions and convert them into an instruction sequence for a floating-point processor. The first compiler implementation designed primarily to perform this conversion was the Fortran compiler. The first FORTRAN compiler is described by [Backus]. This early compiler showed that a high-level language could not only correctly convert a mathematical expression into the correct sequence of instructions for a floating-point processor but that it is possible to perform certain optimizations on the calculation [Padua].

The use of high-level languages greatly facilitated the solution of complex problems. Imprecision problems were noticed, however, in the problems of determining the area under curves and in determining the roots of equations. These problems are typified by a large number of cumulative calculations. When a large number of iterative calculations are performed, the precision loss propagates. Results lose greater precision and thereby acquire greater error from their true value; in general, the greater the number of calculations, the greater the error. The cause of this was known to the early pioneers; it was that they had to perform calculations using guard bits that were lost when the results are stored in finite storage widths. These truncated values were then used to compute further values. [Henrici] described in 1966 one attempt to develop models of how error was introduced and propagated in the solutions of differential equations. [Linz] develops a method for summing values to maintain precision. Instead of doing a sequential summation of a series of numbers, he describes how generating a sum by successive summations on number pairs greatly reduces summation error. This requires explicit coding in a program. The programmer must be aware of the nature of the problem to be solved and be able to determine and code the optimal calculations necessary.

The magnitude of truncation can be known only at run time when a calculation's operations and operands are known. If a calculation consists of several operations, re-ordering the operations may improve the precision of the result. Breaking an operation into several operations where the operands are restructured may improve the precision of the result. This can be done only when the calculation and its operands are known. Current floating-point processors operate without regard to the whole of the calculation and do not determine if a more precise calculation is possible.

A more general architecture is needed to provide the floating-point processor with information so more precise results may be obtained. This requires that more information from the high-level compiler be available to the floating-point unit. It also requires the development of metrics that determine how a calculation is to be restructured, if necessary, for improved precision.

This dissertation develops an architecture that meets these requirements.

Barriers and Issues

Early computing software used guard digits to determine how to appropriately adjust the result of a floating-point calculation for storage. With the advent of very large-scale fabrication techniques, it has become possible to incorporate in hardware many of the calculations that were previously done in software. These large-scale fabrication techniques also make it possible to construct numeric processors that compute results using multiple registers with much larger widths than are allotted for their storage in primary memory. This permits a sequence of calculations to be retained in a numeric processor without truncating the results for primary memory storage before being used again. It also allows greater precision than theory considered before the development of multi-register floating-point processors. It is likely that the precision generated from using a current floating-point processor will generate results requiring greater precision than early theory considered. While the increase in precision for small computations may not be noticeable, it is likely for long and iterative computations it will. Whether this occurs is an issue that is investigated in this dissertation.

The main hypothesis of this dissertation is that improvements in maintaining precision can result from replacing previously computed values with their calculation histories. This leads to even longer calculation histories; these must eventually be shortened with minimal loss of precision. This could be done by adaptive reformulation and/or by constant folding. Developing a method to compact calculation histories is a goal of this dissertation.

Floating-point values are being represented by their calculation histories. The computed value of a floating-point variable is expected to be stored at a known memory location. This allows the address of this memory location to be passed as a parameter to a procedure where its value may be read or written. Since many procedures require the address of a floating-point value, the manner of storing the result of a calculation history must allow the memory location of the value resulting from a calculation history to be known in a production environment.

This work proposes, implements, and tests an architecture that requires a compiler to convert a calculation into a more complex but more accurate calculation for the floating-point processor. To save precision, a compiler may break a floating-point constant that requires extra precision into two or more different values that jointly represent the original floating-point constant but no values of which require extra precision; this would require that a compiler restructure the calculation. This opposes the trends of optimizing compilers to generate simpler code and, consequently, has been left up to software coders to handle, as in [Brent] for example. When it is done by a compiler, it would have to be a compile-time decision rather than a run-time decision that affects the precision of all subsequent computations.

Another issue is to rapidly determine how the calculation is to be restructured by the floating-point processor. Although integer methods may be deployed at run time by decomposing a floating-point value into three component parts, they require an integer computation unit and CPU time during which that integer computation unit can not be used for other calculations. For example, the floating-point processor may have to determine the best sequence to multiply three values. The technique of separating each value into the sum of a precisely computed value and an imprecisely computed value requires a decomposition for the precise and imprecise components of each value, three multiplies, and three compares on their shift counts. For speed, integer-based algorithms must be developed to perform this operation as rapidly as possible. Developing these algorithms is a part of this work.

Storing the new calculation history raises a further issue. A new calculation history requires storage in the form of program memory. A superior implementation as a floating-point processor pipeline step might be able to use cache memory. Additional memory management is needed to destroy the calculation history when it goes out of scope. This can result in fragmented memory and requires processing time to coalesce that fragmented memory. While memory management for calculation histories is more complicated than simply storing a calculated result at a fixed location, in some cases it may be a worthwhile trade-off. The use and management of memory is one of the costs of the calculation that receives comment in the Results section.

It may not be desirable that some results be maintained as calculation histories. This is true for simple calculations of the type $a = b \; op \; c$, where *op* is a primitive arithmetic operator such as the add or multiply operator and *b* and *c* are constants. It is also true if a special algorithm is being coded and no further “improvement” on the calculation is wanted as in the Kahan algorithm [Goldberg]. Only the compiler and linker (in case of calculations passed from one module to another) can determine this and must support mechanisms to do so. A high-level language must support the ability to selectively inhibit the use of calculation histories.

Compilers generate floating-point binary code for the target processor. They assume that the code will be executed by the floating-point processor without alteration. Calculation histories however, require preprocessing before a calculation is executed on a floating-point processor. Consequently, they require a different representation in the code of an application. This also requires that a compiler provide the full calculation to the preprocessor. Proposing this special representation is part of this work.

Elements, Hypotheses, Theories, or Research Questions to be Investigated

The primary hypothesis of this dissertation is that, by maintaining enough of the history of a calculation and applying certain algorithms to restructure the calculation, the precision of the results of subsequent calculations is increased.

The basis of this hypothesis is that the storage format of the IEEE-754 floating-point restricts the number of significant digits (bits) of any floating-point value to maximum width. Should the results of a calculation require a greater number of significant digits, as a multiplication might require, precision would be lost. However, [Stoutemyer] illustrates how formulaic expression affects resultant precision and how, in some cases, certain equivalent formulaic expressions are more precise and should be substituted. At run time, given enough information about the nature of a calculation it might be practical to substitute equivalent expressions to minimize precision loss. This is one hypothesis that is investigated by this dissertation.

[Stoutemyer] also shows that catastrophic cancellation can be a result of the values of both operands of a calculation. A hypothesis is that, by breaking at least one operand into two or more operands, the likelihood of catastrophic cancellation may be reduced.

This dissertation hypothesizes that a compiler can be developed to extract enough information about a calculation for a preprocessor to be able to optimize the calculation.

An element of this study is to show how this could be done. Another element is to show how a preprocessor could use this information to minimize precision loss of a calculation.

Limitations and Delimitations of the Study

The primary work of this dissertation is to explore the use of calculation histories in minimizing precision loss in a calculation. Since an IEEE-754 floating-point calculation returns only a single value, this is done for single floating-point variables rather than subscripted values in an array. This reduces the test system to its simplest form since what can be done to a single floating-point value can be applied to any element of a floating-point array.

FORTRAN 90 is the language upon which the proposed architecture is based. Many of its features, such as those concerning allocatable arrays and referencing shapes, are not needed for this study and are not be used. Appendix A describes the language that is used.

Recursive “C” language procedures are extensively used for this test system. Available memory restricts the size of some calculations when they exceed a large number of floating-point operations.

Definition of Terms

Adaptive Reformulation- replacing a regular expression with an equivalent expression based on the operators of the regular expression.

Additive Chain- a sequential series of add and subtract operations.

Additive Chain Component- a series of additive operand chains that is summed together to produce a single value.

Additive Operand Chain- a series of successive arithmetic operations where the only operation is either addition or subtraction.

Arithmetic Expression- an expression where the only operators are the arithmetic operators for add, subtract, multiply, and divide and the operands are integer or real values.

Arithmetic Operator- a binary operator which is either an arithmetic add, subtract, multiply, or divide.

Bottom-Up Pruning- the process of removing the deepest reduction elements from a calculation tree to limit the number of reduction elements in a calculation.

Binary Operator- an operator that acts on two operands.

Calculation Chain- a sequence of reduction tree elements where the operands of the leaf element are either both reduction tree elements or both populated operands. All superior reduction elements have one data operand and one reduction as an operand.

Calculation Definition- The sequence of stack operations and operands defining a regular expression.

Calculation History- The tree of arithmetic operations with the final values of operands to compute the value of a regular expression.

Calculation History Assignment- a calculation in which the final calculation tree is to be saved.

Calculation History Tree- the tree generated when the top node is the production and the operands, when they are reductions, are nodes.

Calculation History Variable- A named variable whose past calculation history is to be saved.

Calculation Tokens- records between the “Begin Code” and “End Code” records of a calculation. These records are either operands or operators.

Catastrophic Cancellation- the case when a subtraction or division on two numbers very close to each other results in a misleading value.

Central Processing Unit- a hardware unit that processes the machine language instructions of a software application. Abbreviated as “CPU.”

Compiler- a software application that converts a human-readable problem into a machine-executable form.

Direct Assignment- A value assignment where the value of a single operand is being assigned a target variable.

Dynamic Computational Readjustment- re-ordering a floating-point calculation based on the value of its operands.

Floating-point Processor or Floating-point Unit- a hardware unit that processes floating-point calculations. Abbreviated as "FPU."

Floating-point Processor or Floating-point Unit Registers- the storage elements used to store operands and calculated results within the FPU. The storage width of these registers is usually greater than the storage width of program memory.

Floating-point Value- a member of the set of values that is defined by a floating-point specification.   In this dissertation, a floating-point value is a member of the set of 64 bit double-precision values of the IEEE-754 specification.

Grammar- a set of rules that define a language.

Grafting- replacing an operand with the calculation tree used to compute the value of the operand.

Machine Language- in a digital processing machine, streams of binary values that direct its operation.

Multiplicative Operand Chain- a series of successive operations where the only operations are multiplication and division.

Named Value- A variable or constant to which a unique symbolic name has been assigned.

Operator Classification- the process of determining the number of calculation elements of each arithmetic operation used to compute the value of a reduction tree.

Parser- the section of a compiler that processes streams of source language characters. The output of a Parser is a stream of Tokens.

Precision Retention- ordering a series of arithmetic operations so the precision of all values is reflected in the result of that calculation.

Program Memory- the memory in which an application is run and is used to store application specific data. This memory is typically Central Processing Unit memory which has a lesser storage width than the FPU uses internally.

Propagation Error- the difference in a calculated result using an imprecisely computed value and using the correct value as an operand.

Pruning- the process of removing reduction elements from a calculation tree to reduce its size.

Reduction- creating a binary operation on two operands on a stack producing a new a single value. The name is given because it reduces the size of the stack. The three stack elements are replaced by a reduction element.

Reduction Assignment- A value assignment that is the result of at least one arithmetic operator.

Reduction Element- a binary structure with the operation as the node and operands as leaves. An operand may also be a reduction element. The node and its leaves have been removed by a stack reduction.

Reduction Tree Save Element- the record of a Reduction Tree Element that is saved to persistent storage. It contains information in addition to the Reduction Tree Element to reconstruct all its data.

Regular Expression- an arithmetic expression which may include arithmetic expressions enclosed in left and right parentheses or a single operand.

Result index- the zero-based index to an array into which intermediate results of a calculation history assignment are stored.

Stack- Sequential list of operations and operands constituting a calculation. In this dissertation, a Stack is of the Last In, First Out stack class.

Stack Element- An operation or operand that is pushed onto a stack.

Tokens- Smallest meaningful groups of source stream characters that are classed by usage in a grammar.

Value Assignment- the act of copying into another location a known or computed value.

Well Formed Calculation- a regular expression that leaves one result and no unused tokens to compute a value.

Summary

The resultant precision of a floating-point calculation is highly dependent upon the operations in which a calculation is performed and the run-time values of operands of each operation of that calculation.

High-level compilers convert formulaic expressions for calculations into a fixed sequence of operations on operands compatible with a floating-point processor. Although the sequence of operations is static, the values of their operands are not. Equivalent formulaic expressions using IEEE-754 floating-point values yield results of varying precision using the same operand values.

Precision loss may be minimized by being able to modify the calculation by restructuring it and the operands run time. Accomplishing this requires more information than the traditional methods supply the floating-point processor. It also requires additional methods of processing the entirety of the calculation since this is not currently done.

An architecture in which a high-level compiler provides additional information to an intermediate step was proposed. This would allow all the operations and all the values of the operands of a single calculation to be reorganized in a way to minimize precision loss.

Calculation histories were proposed to preserve enough of the previous computation of values so that when the value is used in a subsequent calculation that subsequent calculation may be restructured to produce a more precise result. This would be accomplished by developing an architecture to properly restructure a calculation to minimize precision loss.

Chapter 2

Review of the Literature

Historical Overview of the Theory and Research Literature

The design of today's arithmetic processors can be traced to Charles Babbage's proposed computing machine in the nineteenth century. [Hartree] describes Babbage's machine as consisting of three functional units. The first was a "store" consisting of registers that contained the operands of a calculation; the second unit was the "mill" that performed the mathematical operations on the data contained in the store and could transfer resultant values to the store; and the third unit was not named but controlled the sequence of instructions performed by the mill. These units are not unlike the memory, floating-point processing hardware/software, and software programs of today's digital workstations. Dr. Hartree notes that one of the earliest computing machines, the ENIAC, had this basic design.

Intel described its first dedicated processor for real values in a manual published in 1981 [Rash]. This manual describes floating-point formats for real values that eventually became standardized in the IEEE-754 Standard [ANSI/IEEE]. The availability of cheap memory, fast processors, and high-speed floating-point division algorithms has made the IEEE-754 floating-point formats most commonly used in mathematical calculations. Along with the storage formats of real numbers, the Intel manual described the floating-point processor's instruction set. This freed programmers from having to rely on software procedures since a floating-point instruction set had been incorporated in hardware.

While it is possible to program floating-point calculations directly using a floating-point processor's instruction set, it is much easier to use a high-level programming language. A compiler for a high-level programming language could take mathematical expressions and convert them into an instruction sequence for a floating-point processor. The first compiler implementation designed primarily to perform this conversion was the FORTRAN compiler. The first FORTRAN compiler is described by [Backus]. This early compiler did more than to show that a high-level language could correctly convert a mathematical expression into a functionally equivalent sequence of instructions for a floating-point processor. The compiler also showed it is possible to perform certain optimizations on the calculation [Padua].

FORTRAN, typical of most computing languages, could handle only certain types of real values. A 1997 FORTRAN language reference manual [Lahey] describes only two precisions of real values. If greater precision were required in FORTRAN, it would be necessary to define a new data type using the existing data types in FORTRAN and to develop called procedures to manipulate the new data type. This was accomplished in 1978. [Brent] describes a package to perform multiple-precision on real values. The real values are stored as integer arrays and calls are made to the appropriate procedure to perform a desired computation. The need to explicitly code for the multiple-precision was eliminated by use of a compiler building tool described by [Brent 1980]; this allowed a programmer to treat multiple-precision mathematic as an intrinsic Fortran data type. This required an initial pass to create the regular FORTRAN code and a few lines of code were needed to be added by the programmer to initialize the multiple-precision logic. [Cilie and Corporaal] later describe how this could be done to C programming language code without requiring an initial pass by a post-processing tool or special code added by the programmer.

Two areas in which an imprecision problem was first noticed were in the problems of determining the area under curves and in determining the roots of equations. These problems are typified by a large number of cumulative calculations. When a large number of iterative calculations are performed, the results can lose precision and thereby differ from their true value; in general, the greater the number of calculations, the greater the difference. The cause of such error was known to these early pioneers; it was that they had to perform calculations using guard bits that were lost when the results are stored in finite storage widths. These truncated values were then used to compute further values. [Henrici] described in 1966 one attempt to develop models of how error was introduced and propagated in the solutions of differential equations. He decomposed the contributions to computational error into round off, truncation, and propagation errors and developed formulae to predict their amounts. Subsequent work in this area was presented in papers by [Hull and Swenson], [Mutrie et al.], [Brown], and [Goldberg]. Based upon error analysis, [Brown, 1981] describes accurate vector solving algorithms. [Linz] develops a method for summing values to maintain precision. Instead of doing a sequential summation of a series of numbers, he describes how generating a sum by successive summations on number pairs greatly reduces summation error.

[Lyon] considers the effect of data measurements on the accuracy of computed values but develops tests if results relate to the data on which they are based. He also develops expressions to adjust for the accumulated error in quadrature methods.

A more comprehensive paper [Stoutemyer] illustrates how formulaic expression affects resultant precision and how, in some cases, certain equivalent formulaic expressions are more precise and should be substituted. Stoutemyer breaks error up into three components:

- *inherent error* -- results from the computational algorithm as used in transcendental functions,
- *analytic error* -- the difference from the true resultant value with the true value(s) of the operand(s) and the true computed resultant value with the operand(s) from memory, and
- *generated error* -- the difference from the true resultant value with the operand(s) from memory and the computed resultant value with the operand(s) from memory.

Doing so, he develops error expressions to determine error limits for mathematical expressions that he incorporates in a program named *Reduce*. The author uses this program to predict areas of *catastrophic cancellation* in which mathematical combinations of operand values may generate unacceptable computational error. Equivalent expressions are developed to avoid the computational error in these regions.

The use of high-level languages masks many of the problems intrinsic in floating-point. [Goldberg] recognizes *catastrophic cancellation* described by [Stoutemyer]. He notes that performing a computation exactly and then rounding it gives more accurate result than does using a single guard digit. This is significant since today's floating-point processors use much wider words for computation than for storage. Goldberg emphasizes that particular care must be given to the rounding philosophy. The practice of rounding up increases the error in successive additions of positive values. Depending upon the calculation and data, several rounding options must be available. Two methods of retaining precision are described to avoid the round-off problem: storing the values in arrays where greater storage is available than the standard storage width similar to [Brent], and storing the value using the standard storage widths as an array of values the sum of which is the exact result. The latter is similar to maintaining a calculation history where the implied operation is addition. Goldberg also recognizes that algorithms to minimize or compensate for round-off error exist. He describes the Kahan summation formula that requires disabling all optimizations to correctly operate.

[Bush] discusses situations in which comparisons between real numbers fail as a result of a difference in storage and calculation precision. He notes the loss of precision when two floating-point numbers close in value to each other are subtracted and the probable loss of a unit of value when the integer portion of a floating-point value is truncated when stored as an integer. The latter occurs as a result of the floating-point unit's rounding policy and requires a special procedure to convert a floating-point value to the nearest integer.

Previous Approaches to Resolve this Problem and Their Shortcomings

One approach proposed to address the imprecision problem is *interval mathematics*. Interval mathematics realizes that floating-point results cannot always be adequately stored as a single value because of the need to truncate a result into a fixed-width storage element. Instead, interval mathematics stores the result as two values. One value represents the lower bound of the result, and is the result rounded toward negative infinity; the other value represents the upper bound of the result, and is the result rounded toward positive infinity. These two values are the limits (*end points*) of a closed interval within which the actual value of the result is guaranteed to lie. Smaller intervals between these end points implies greater accuracy. While this approach does not solve the imprecision problem, it provides a guaranteed range in which the result resides. *Reliability in Computing* [Moore], a compendium of papers, describes the implementation and overall considerations of this approach. Papers therein describe how interval data types have been adapted to the FORTRAN and C programming languages.

In his Master's thesis, Kouji Ouchi [Ouchi] describes implementing a system to compute values to a specified precision. This was implemented as a series of classes written in the C++ programming language. The implementation, named "Real/Expr," provides a class, named "BigFloat," for an arbitrary precision data type. A "BigFloat" value is defined as a three element tuple <*m*, *err*, *exp*>, where *m* is the mantissa, *err* is the error, and *exp* is the base two exponent of the value. It is significant to note that error is tracked as part of the computed value. A variable in a C++ program may be declared as a "BigFloat" and used in regular calculations. A second class, named "Expr," is defined so a calculation may be performed using "BigFloat" operations (defined for add, subtract, multiply, divide, and square root). An "Expr" calculation is a directed acyclic graph ("DAG" or tree) of algebraic expressions using the allowable "BigFloat" operations. To calculate the result of an "Expr," a programmer declares a variable of class "Expr" and assigns to that variable an algebraic expression.

Once a calculation has been assigned to an "Expr" variable, its value is computed based on the DAG. The class does not perform any further analysis on the tree but performs the calculation. As it performs the calculation, it uses internal algorithms to track and limit the computed error. The end result may be a value of the desired precision, but based upon the nature of its operands may possess error from the imprecisely computed values of operands.

Summary of What is Known and Unknown about the Topic

The causes of computational loss of precision have been well established and studied. A number of approaches have been implemented to minimize loss of computational precision. Algorithms such as Kahan's summation formula have to be coded. Subroutines that implement extended precision values have to be developed and explicitly called. Interval mathematics have been proposed as a possible solution. They also require special subroutines.

These existing methods tackle the problem only from the source code. They do not take advantage of floating-point processors' capabilities and do not consider the effect of the precision of the operands on individual calculation results.

The research illustrates that there are alternatives, such as equivalent expressions, to calculations that may have catastrophic results. Implementing a solution that utilizes these alternatives has yet to be proposed. The improvements achieved with such a solution are yet unknown.

The Contribution This Study Will Make to the Field

This dissertation will establish new groundwork in precision computing by manipulating run-time data. This will be accomplished by creating a framework into which new algorithms to minimize precision loss can be implemented, tested, and evaluated that run-time data can be more effectively restructured.

The results of this dissertation will establish whether such a data-driven architecture based on calculation histories may or may not be effective.

The implementation and study of the proposed framework will provide an assessment in the improvements in computational precision to be gained through such calculation manipulation. The additional overhead required will be measurable and will allow a cost to benefit analysis of the methods employed.

The primary contribution will be the development of algorithms to make such an implementation possible. These will be founded upon the existing literature but implemented in a functional computational system.

Chapter 3

Methodology

Research Methods

Overview of Procedures Employed

The current architecture of computing systems was described in Chapter 1. Chapter 1 also described the imprecision introduced by storing real values in fixed-width storage elements. This architecture entailed two elements. The first was the high-level language in which the computing problem was defined. The computations are performed using values in fixed-width storage. The second was the floating-point processor that performed the calculations. Floating-point processors process calculations use internal registers wider than the data supplied for the calculations. A high precision computed value would thus have its least significant bits truncated to fit into fixed-width storage. When subsequently used as an operand in a calculation, a truncated value could cause error in the final computed value.

Recent attempts to deal with this problem were described in Chapter 2. They entailed the programmer either coding a problem in certain ways, or having to use special routines to deal with extended precision values. Either way, once the problem was encoded, there was no way to modify the calculation if operands used real time caused loss of precision or created error.

The goal of this dissertation, stated in Chapter 1, was to show that a combination of a high-level compiler and a low-level preprocessor could overcome these limitations by the use of what are called “calculation histories.” A calculation history represents the value of a variable by the mathematical operations and operands to compute the variable’s most recent value.

“Real world” computational problems are defined using high-level programming languages like “C” or FORTRAN. Consequently, the first step was to develop a high-level language in which to describe a calculation problem. A high-level compiler, developed for this dissertation, output a problem defined in this high-level language similar to FORTRAN to a textual object file for a processing engine to process. This was done by developing a stand-alone program using the “C” programming language.

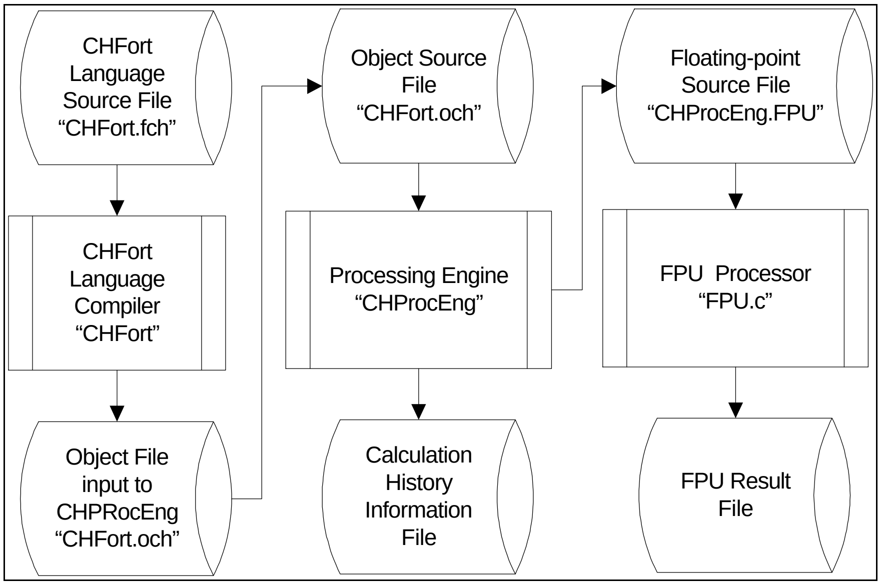
The second step was to develop a processing engine functioning as a virtual machine to perform the problem executing the output of the high-level compiler using the “C” programming language. The virtual machine performed the programmed mathematical calculations using “C” language statements. It also generated floating-point code for the floating-point processor (“FPU”), but it did not execute them.

The third step was to develop a “C” language program in which to process the floating-point code created by the virtual machine. The virtual machine output a “C” language file containing the floating-point code that was inserted into a base program. The modified base program was compiled and executed. The output of the floating-point program was compared to the corresponding output from the preprocessor as well as from either a “C” language baseline program or a computed mathematical expression.

System Design

The test system of this dissertation, shown in Figure 2, consisted of three units. The first unit was the high-level language compiler named “CHFort.” Test problems were defined in an ASCII text file named “CHFort.fch.” The “CHFort” compiler generates an ASCII object file, named “CHFort.och,” representing the program. The second unit was the virtual machine/processing engine, named “CHProcEng,” which uses the file “CHFort.och” as input. The virtual machine/processing engine “CHProcEng” performs an initial scanning pass on this ASCII object file to create internal structures for its processing. After it completes the initial scanning pass, CHProcEng executes the code. During code execution, CHProcEng tracks the initial structure and final structure of each calculation history variable as each calculation history variable is computed.

Figure 2- The Three System Processing Units of the Test System


CHFort Language Source File “CHFort.fch”
CHFort Language Compiler “CHFort”
Object File input to CHPRocEng “CHFort.och”
Object Source File “CHFort.och”
Processing Engine “CHProcEng”
Calculation History Information File
Floating-point Source File “CHProcEng.FPU”
FPU Processor “FPU.c”
FPU Result File

CHProcEng produces a text file named “CHProcEng.FPU” consisting of mixed assembly and “C” language code. This code was cut and pasted into the program named “FPU.c.” Program “FPU.c” was compiled. When executed, program “FPU” performed the calculation using the workstation’s FPU. The results were output in decimal and binary formats. Calculation results were directed into a text file for reporting.

Compiler Design

Overview

The high-level language compiler, named “CHFort,” was a subset of the FORTRAN 90 language. This subset possesses none of the language features not necessary for this work. An extension was added to the subset to declare a floating-point variable as a history variable. An attribute “History” may also be given for a floating-point data type if the variable is to be treated as a calculation history variable. Appendix A describes the language that was implemented in this study.

The compiler’s output object file was the CHProcEng source file named “CHFort.och.” A “CHFort” source file is shown in Listing 1. The corresponding CHProcEng source object file is illustrated in Appendix B.

Listing 1- Sample CHFort Source Program

```
Double Precision, History :: dYSum, dXMid
Double Precision dX0, dSpan, dSpanIncs,
dSpanDelta
dX0 = 0.
dSpan = 1.
dSpanIncs = 199.
dSpanDelta = dSpan /dSpanIncs
dXMid0 = dX0 + dSpanDelta / 2.
dXMid = dXMid0
dYSum = 0.
100 Continue
        If (dXMid .gt. dSpan) GoTo 999
        dYSum = dYSum + dXmid
        dXMid = dXMid + dSpanDelta
        Go To 100
999 Continue
End
```

Statement Processing

The compiler was implemented similar to a “Multibox Parser” [Dyadkin]. This compiler architecture is more suitable than the combination of Lex and Yacc for complex grammars. There are three boxes to the compiler. The first box creates a linked list of source statements. The linked list of source statements is necessary since a statement may be continued onto subsequent lines and only complete statements are processed. Comments may be present but have to be ignored. A limited parser is implemented in this step to determine when no additional source statements are necessary to complete the linked source statement list. When a complete statement has been read in and no continuation lines are indicated, the source statement linked list is passed to the second box.

The second box parses the source statement linked list into source language tokens. Since a source statement block may contain more than one statement, tokens are created only for the next unprocessed statement in the source statement block. The end of a source statement is indicated by no more unused source statement tokens or the existence of a token denoting the end of a source statement. The latter case requires continuing to process additional source statement tokens in the source statement block.

The third box processes the tokens in the statement in which tokens were generated the previous step. This consists of identifying the statement class of the source statement and performing the functions necessary to process it.

The procedure “procMakeExpr,” Appendix C, is used extensively in the third box. Its purpose is to reduce the arithmetic statements into more easily processed structures.

Named Variables and Arithmetic Statements

Named variables are handled separately from arithmetic statements. A named variable may, in CHFort, be defined at any time. All named variables and values, regardless of their declaration, are treated as 64-bit real (double precision) values. Named variables may be dimensioned in their type declaration but were not implemented as dimensioned variables in arithmetic expressions. This is a limitation of the compiler.

Tokenizer Design

The tokenizer implements a state based single character Look Ahead policy. Initially, the tokenizer is in an undefined token state. In the undefined token state, the input statement is scanned for the first character that initiates a token. A unique number is assigned to that token state. That token state becomes the next state of the tokenizer. Subsequent characters are tested if they belong to that token state. If they do not, the next token state is changed to the undefined state and a flag is set so the tokenizer will reuse the last obtained character. A difference in the current state and the next state of the tokenizer signals the generation of the token. Each generated token is appended to a linked list of tokens for that statement.

Two versions of the tokenizer exist in CHFort. A reduced tokenizer was implemented to determine statement blocks within a source file. The full tokenizer was implemented when processing single programming statements.

Parser Design

The parser implements a single token Look Ahead Left Recursive ("LALR") grammar. This grammar generates a calculation definition in such a way that it can be parsed and executed on a stack machine. For example, Listing 2 shows the calculation stack generated by CHFort for the expression "(A + B) * (A - B) + 5." The procedure implementing the LALR grammar used to produce this grammar is the "procMakeExpr" procedure shown in Appendix C. The procedure uses operator precedence and, where precedences are equal, operator associativity to determine when to reduce the token stack [Aho et al.].

Listing 2- LALR Stack for (A + B) * (A - B) + 5

```
push String A
push String B
plus
push String A
push String B
minus
mpy
push  Number 5
plus
```

In Listing 2, a "String" indicates that the following text is the name of a variable; "Number" indicates that the following text is the value of a number. The single text fields are the token descriptions of the applicable operation on the most recent values on the stack ("Stack"). An operation token has a precedence of zero or greater. Operand tokens have a precedence of negative one.

The parser operates by creating a "Stack element" from the next successive token in the source token linked list. This Stack element is pushed on the "Stack," which is actually a linked list. A Stack reduction results in the removal of Stack elements from the Stack and the generation of a “reduction element.”

If the last Stack element is an operator token, then a Stack reduction may occur if the three preceding Stack elements are, successively, operand, operator, and operand Stack elements. The previous operator must have a greater precedence, or, when precedences are equal, a left associativity.

Parenthetical expressions force a Stack reduction when the closing right parenthesis is pushed on the Stack. All reductions left on the Stack are performed beginning with the Stack element preceding the right parenthesis Stack element ending with the starting left parenthesis. One Stack element replaces all Stack elements comprising the parenthetical expression.

The pushing onto the Stack of an unexpected token forces all remaining reductions. This will leave the unexpected token as the topmost Stack element. Depending upon the context in which the expression is processed, this may be treated as an error.

When a correctly formed expression is fully processed, only one Stack element remains. If this Stack element points to a reduction element, the expression is called a “reduction assignment.” Otherwise, there is only one operand that is on the Stack; this is called a “direct assignment.” If the expression is improperly formed, more than one Stack element remains. This is treated as an error.

Named Values

Named values represent variables and constants. A named constant is specified by the attribute “Parameter.” Variables are allowed to be dimensioned as arrays. If a named constant is dimensioned, the dimensions are ignored. When a named variable is assigned an initial value, it cannot be assigned array dimensions. If a named constant is not assigned an initial value, a value of zero is automatically assigned. Although the language allows integers to be declared, they are treated as double precision real values.

The attribute “History” in a data declaration causes the variable to be flagged as a history variable.

The definition of each named value is stored in a binary file name “NamedVals.dat.” This file is overwritten by subsequent applications of the CHFort language.

Computational Value Assignment Expressions

Giving the result of an arithmetic expression to a named variable is called a *value assignment*. A value assignment is an expression of the form “*A* = (*B* + *C*) * 5*(*dValue* + *1*).” Parentheses are optional in a value assignment. The right side of value assignments is processed by the procedure “procMakeExpr,” listed in Appendix C. Value assignments are stored in a code file which is an ASCII text file named “RecCode.dat.” After a source file has been processed, these value assignments are copied into the output object file. Procedure “procMakeExpr” returns a Stack element, which notes either a direct assignment or reduction assignment.

The value assignment is written to the code file as it would appear in the object file input to CHProcEng. This is done by writing a “BeginCode” record with the name of the variable receiving the assignment. The procedure “procOutputCode,” Listing 3, is called to output a stack form of the expression. This is ended by writing an “EndCode” record.

If a direct assignment is being saved, the procedure “procOutputCode” outputs a “push” record to the code file. Otherwise, a reduction assignment is being saved. Recursive procedure “procCreateStackFromReductions,” shown in Listing 4, saves a reduction assignment.

Listing 3- Procedure “procOutputCode”

```
_proc int procOutputCode(ShiftElement_struct *pThisSE)
{ /* procOutputCode- Creates code records for project */
    if (pThisSE->pReduxNode)
    { /* Reduction calculation */
        procCreateStackFromReductions(pThisSE->pReduxNode);
    } /* Reduction calculation */
    else
    { /* Direct assignment */
        fprintf(m_fhCode, "push %s %s\n",
           pszTokenDescription(pThisSE-
>TokenElement.nTokenType), pThisSE->TokenElement.pszValue);
    } /* Direct assignment */
    return 0;
} /* procOutputCode- Creates code records for project */ Do
```

Listing 4- Recursive Procedure “procCreateStackFromReductions”

```
_proc int
procCreateStackFromReductions(ReductElement_struct
*pThisRE)
{ /* procCreateStackFromReductions */
    int nArgsIx;
    /**/
    for (nArgsIx = 0; nArgsIx < 2; nArgsIx++)
    { /* Push this side onto the stack */
        if (pThisRE->pArgsRE[nArgsIx] == NULL)
        { /* Operand is data */
            fprintf(m_fhCode, "  push %s %s\n",
               pszTokenDescription(pThisRE-
>ArgsToken[nArgsIx].nTokenType),
               pThisRE->ArgsToken[nArgsIx].pszValue);
        } /* Operand is data */
        else
        { /* Operand is a Reduction */
            procCreateStackFromReductions(pThisRE-
>pArgsRE[nArgsIx]);
        } /* Operand is a Reduction */
    } /* Push this side onto the stack */
    fprintf(m_fhCode, "  %s\n",
       pszTokenDescription(pThisRE->OpToken.nTokenType));
    return 0;
} /* procCreateStackFromReductions */
```

Calculation Definitions

Calculation definitions generated are prefixed in the object file with a “Begin Code” record and terminated in the object file with an “EndCode” record. Between the “BeginCode” and “EndCode” records are records of the operands and operators that comprise the calculation. These records represent operations on a stack-oriented FPU to compute the expression (see Listing 2).

The first field on an operand record is the phrase “push.” This is followed by a token description (either “String” or “Number”) that gives the nature of the operand, which is the third field. A “Number” is a string of numeric digits. The “String” fields indicate variable names.

An operator record consists only of one field. This is the description of the token representing the operation. For example, addition is represented by the token string “plus.”

A calculation definition is converted by the processing engine into a tree structure for processing by the processing engine’s algorithms. Listing 5 shows a trivial program to track the history variables “A” and “C.”

Listing 5- Sample Source to Test Calculation History Variables

```
! Declare variables A and C as real and
! with the (calculation) history
attribute
!
Real, History :: A, C
!
A = 2
I = 0
001 I = I + 1
If (I .gt. 3) GoTo 002
    B = A + I
    C = (A + B) * (A - B) + 5
    A = C
GoTo 001
002 Continue
End
```

The “CHFort” compiler generates an object file similar to that of Appendix D.

Virtual Machine/Processing Engine

Operational Overview

The processing engine program, “CHProcEng,” uses the output object file of the high-level compiler as the program instruction stream. A program counter is not implemented as in hardware digital processing units. The file offset of each record in the object file acts as an instruction address. The initial value of the “program counter” is the offset of the first executable statement in the object file. Unless a branch is executed, the program counter advances to the file offset of the next statement following the statement referenced by the program counter. Object code statement labels refer to the object file offset of the following record. Executing a branch to a program label causes the file offset of the record in the object file after the label statement to be the current program counter. A linked list of label names and next record offsets is created by CHProcEng for rapid execution.

Object File Statements

The input object file consists of two sections. A section begins with a record containing in the first to columns the string “<<.” The start of a section name follows this and ends at the string “>>.” The start of one section signals the end of the previous section.

The first section defines the named values that are explicitly defined in the source problem file. This is indicated by the header record “<<VariablesValues>>.” A named value begins with a “Name:” record and terminates with the occurrence of the next “Name” record or end of the section. The fields in a name definition are:

- *Name*: The name of the value
- *IsFixed*: 0 if a variable, 1 if a constant value
- *IsHist*: 1 if a calculation history value, 0 if not
- *InitialValue*: Optional, the initial value assigned the name
- *DataType*: an integer giving the data type of the variable (usually 5- indicating a double precision floating-point value)
- *DimsCount*: The number of array dimensions
- *Indexes*: a comma-delimited list of the array dimensions (Optional)

The second section is the code section. This is indicated by the section header record “<<ProgramCode>>.” The code section contains textual records that describe how the problem is executed. The records supported in the code section are:

- *Label*: Gives a named reference to the following code record

- *GoTo*: Followed by a Label reference, changes the next record to be executed to be that following this named reference.
- *GoToCond*: Followed by a Label reference, a test condition, and a variable name. Tests the variable for the condition. If the condition is found true, executes a “GoTo” the named reference; otherwise, the instruction flow is not altered.
- *“BeginCode”.”EndCode” Sequence:* Creates a calculation definition to be performed. A “BeginCode” record may be “BeginCode”, “BeginBoolCode”, or “BeginArgCode.” Between the “BeginCode” and “EndCode” records are the operations and the operands defining the calculation. The name of the target that receives the value of the calculation is on the “BeginCode” record following the “BeginCode” string.

Calculation Definitions

Each calculation definition starts with a “Begin Code” record and terminates with an “End Code” record. The “Begin Code” record differs depending upon the type of calculation being defined. If a regular calculation is being defined, the “Begin Code” record is the string “BeginCode.” The string “BeginIndexCode” is used to start the definition of the calculation of an array index. The string “BeginBoolCode” begins the definition of any calculation that does not require calculation histories.

All calculation definitions are defined by the offset of the record in the object file of the “BeginCode” record. CHProcEng maintains a linked list giving the names of each variable with at least one calculation definition. This is the “defined calculation list.” Since more than one calculation may be given for a variable, a linked list of what are called “defined calculation elements” containing the particular calculation details is maintained for each variable in the defined calculation list. Pointers to the first and last entries of the defined calculation element for each variable name are maintained for each name in the defined calculation list. A calculation definition may be that of an array index, function argument, named variable, or array element of a named variable.

Index and function argument calculation definitions are maintained in the defined calculation list under their own reserved names of “d__Indexes” and “d__Args” respectively. They neither are subject to history calculation substitution nor are stored as calculation histories. Indexed variables and function arguments were not used in any of the test cases of this dissertation.

Internally, a calculation definition is a tree of the linked reduction elements created from the calculation tokens in the object file. This linked list of reduction elements comprises the reduction tree of the calculation. Calculation tokens are specified by the operators and operands between the “Begin Code” and “End Code” records. Each calculation token is pushed onto the Stack as a Stack element. A Stack element is described in Listing 6.

Listing 6- Stack Element Structure

```
typedef struct tagStackElement_struct
{ /* StackElement_struct */
    struct tagStackElement_struct *pPrev,
*pNext;
    int nValType;
    /* nValType-
         -1 not known
          0 hard coded number
          1 named variable- fixed constant
          2 named variable- process variable
          3 named variable- history variable
          5 operation
          6 reduction
          7 Raw (in DataValue)
          8 Function Call
    */
    int nSourceType;
    /* nSourceType- Reference only
         1 Variable Name
         2 Number
         5 Operator
         6 Reduction
         7 Raw Data in DataValue
    */
    void *pReduxElement;
    TokenData_struct TokenData;
    ResType_Struct DataValue;
    FlPtTuple_struct FlPtTuple;
} StackElement_struct;
```

The last Stack element being an operator may cause a Stack reduction. The two Stack elements preceding the operator Stack element are assumed to be its operands. The adjacent Stack element is the right operand and its preceding Stack element is the left operand. A reduction element, described in Listing 7, is formed from these three Stack elements. The two operand Stack elements are removed from the Stack. The “pReduxElement” property of the operator Stack element contains the location of the new reduction tree element and is no longer NULL. This is done until all tokens for the calculation have been pushed onto the Stack.

Listing 7- Reduction Element

```
typedef struct tagReduxElement_struct
{ /* ReduxElement_struct */
    struct tagReduxElement_struct *pPrev, *pNext;
    int nREIx;
    int nArgsType;
    StackElement_struct SEOperator, SEOperands[2];
    int nParentREIx, nOperandsREIx[2];
    int nParentArgIx;
    struct tagReduxElement_struct *pREParent,
*pREOperands[2];
    int nGenIx, nDepth;
    void *pCalcChain;
    void *pFlattenExprs[2];
    FlPtTuple_struct FlPtTuple;
} ReduxElement_struct;
```

When all calculation tokens have been processed and the calculation has been well formed, the Stack consists of only one Stack element. The value of the “pReduxElement” property of the remaining Stack element determines the nature of the calculation. If the value is NULL, the value of the assignment is that of the Stack element’s “TokenData” property, Listing 8, and the calculation is a “direct assignment.” Otherwise, a binary tree of reduction elements is present and the calculation is a “reduction assignment.” The reduction tree is the calculation tree for the expression and the Stack element’s “pReduxElement” property points to the top element of the calculation’s reduction tree.

Listing 8- The “TokenData” Structure

```
typedef struct tagTokenData_Struct
{ /* TokenData_Struct */
    /*
        Source information
    */
    TokenTypes_Enum nTokenType;
    char *pszField;
    DataTypes_Enum nDataType;
    /*
        Result value
    */
    Variant_struct DataValue;
    /**/
} TokenData_Struct;
```

Each time the calculation definition is executed for a calculation history variable; three versions of the calculation are saved. The first version is the calculation definition with all calculation histories and all variables replaced with their current value. The second version is the calculation definition after all arguments that are history variables are replaced by their final “computation tree.” The third version is the optimized version of the calculation as it was executed by the floating-point engine. These are saved in a permanent file named “CHProcEng.dat” with other information for reporting purposes. Also saved in the same permanent file for reporting with the optimized version are the values of the calculation performed internally on the reduction tree before the calculation is subjected to restructuring.

Stored Calculation Representation

A Calculation Tree is stored in a binary file named “RdxTrees.tmp.” The first “record” of the stored representation is a calculation header. This header is described in Listing 9. The value of the property “nElesCount” provides the number of stored reduction tree elements for the calculation. If this number is zero, the calculation is a direct assignment and only one reduction tree element is stored; the token value is stored in the operator Stack element. Otherwise, the calculation is a reduction assignment and the value of the property “nElesCount”contains the number of reduction tree save elements, Listing 10, which are stored following the header. Before the calculation is saved, the reduction tree element property “nREIx” is assigned sequentially beginning with zero starting with the most distant reduction tree element to uniquely identify each stored reduction tree element.

Listing 9- Reduction Tree Element Save Header

```
typedef struct tagReduxSaveHdr_struct
{ /* ReduxSaveHdr_struct */
    int nElesCount, nMinIx, nMaxIx;
    int nTopREIx;
} ReduxSaveHdr_struct;
```

Listing 10- Reduction Tree Element Save Element

```
/*
Saved Reduction Tree record
*/
typedef struct ReduxEleSaved_struct
{ /* ReduxEleSaved_struct */
    ReduxElement_struct SavedRE;
    int nOpREIx, nArgsREIx[2];
    ReduxEleSavedREIxs_struct REIxsSaved;
    TokenTypes_Enum nOpTokenType,
nArgsTokenType[2];
    char szOpToken[m_nMaxTokenSize + 1],
szArgsToken[2][m_nMaxTokenSize];
} ReduxEleSaved_struct;
```

Loading a Calculation into Memory

Calculation histories are loaded into memory only when a target variable is a calculation history variable. There are no calculation histories to load if the calculation is a direct assignment and the operand is not a calculation history variable, or it is a reduction assignment and no operand is a calculation history variable. Regardless the type of assignment, a calculation definition is always loaded from persistent storage into memory.

A calculation definition or tree is loaded into memory from persistent storage by providing the file offset to the saved calculation history to be loaded. The calculation header at this offset is read. The number of reduction tree elements is given by the property “nElesCount” of the recorded calculation header.

Since the original memory offsets stored with the calculation history are no longer useful, a restore array is allocated with that number of elements to hold the new memory offsets of each new reduction tree element. This array is populated with the memory offset where each new reduction tree element was loaded into memory. After all reduction tree elements have been loaded from storage, the “pREParent” and “pREOperands” property values are tested for NULL. If one is not found to be NULL, the “nParentREIx” or “nOperandsREIx”, respectively, property values give the index in the restore array of the correct memory offset. The memory offset replaces the corresponding “pREParent” or “pREOperands” property value.

If the value of calling parameter “nOp” is zero, an original calculation definition is being loaded or the calculation history of a direct assignment is being loaded. The procedure does no more.

If the value of calling parameter “nOp” is two, a calculation history is replacing a calculation tree element operand. Replacing an operand with a calculation history is called “grafting.” The calling parameter “pThisSE” points to a Stack element of the operand being grafted. Grafting a reduction tree to an operand causes the Stack element’s “pReduxElement” property to point to the top of the reduction tree and the value of property “nValType” to be set to six, which denotes a reduction.

Populating a Calculation

Initially, once a calculation is loaded, the values of its operands have not been determined. The Stack element operands of each reduction tree element maintained their references to the items with the values to be used in their place. The operands of each reduction tree element are populated depending upon the type of the operand. Table 1 gives the types of operands that may be present in a reduction tree element.

Table 1- Types of Operands Supported in a Reduction Tree Element Operand

Type	Description
0	Coded Number
1	Named Constant
2	Named Stored Variable
3	Named History Variable
5	Operation on Operands
6	Reduction Tree Element
7	Binary Value
8	Result of a Function Call

If the target of a calculation history assignment is a calculation history variable (a type three operand) and a named variable operand of a Stack element is a calculation history variable, the most recent calculation used to create the operand replaces the Stack element operand. When the replacing operand is a reduction assignment, a grafting of the variable’s last saved calculation history occurs, and the length of the reduction tree is extended. A saved reduction assignment calculation is populated already with computational values so it is not traversed for repopulation.

A type seven operand already contains the value for calculating. Types zero, one, two, and eight operand values are loaded into the “DataValue” element of the “Tokendata” structure and the type is set to seven. This also happens for type three operands unless the calculation is a calculation history assignment. The floating-point tuple of each type seven operand is populated at this time.

The reduction tree elements of the original calculation are populated in linked list order, beginning with the first reduction tree element of the calculation and ending with the last reduction tree element of the initial calculation. When additional reduction tree elements replace Stack element operands, they are appended to the reduction element linked list after the last reduction tree element of the initial calculation.

The Reduction Analysis Element

To enable further processing of a reduction tree, an additional processing element had to be developed. This was the r*eduction analysis element*, described in Listing 11. There is a one-to-one correspondence between reduction tree elements and reduction analysis elements.

Listing 11- Reduction Analysis Element

```
typedef struct tagReduxAnalyzeElement_struct
{ /* ReduxAnalyzeElement_struct */
    struct tagReduxAnalyzeElement_struct *pPrev, *pNext;
    struct tagReduxAnalyzeElement_struct *pParent, *pArgs[2];
    ReduxElement_struct *pRE, *pParentRE;
    int nParentArgIx;
    int nReduxAnalyzeCounts[m_nReduxAnalyzeCountsSize];
    int nDist, nGenIx;
    int nResIx;
    int nMaxRELen;
    FlattenExpr_struct *pFlattenedExpr;
} ReduxAnalyzeElement_struct;
```

The memory offset of the reduction tree element to which a given reduction analysis element applies is the “pRE” property. The “pParent” property points to the reduction analysis element to which it is an argument. Whether it is the left or right argument is given by the “nParentArgIx” property. Property “nParentArgIx” contains the value zero for the left argument and the value one if it is the right argument. The “pArgs” property points to the reduction analysis elements representing the reduction tree elements that are operands to its “pRE”s element.

Instead of processing reduction tree elements directly, each reduction tree element is referenced by the “pRE” property of its reduction analysis element.

Calculation Flattening

Grafting reduction tree elements onto a reduction tree increases the complexity of the calculation. To minimize this effect of grafting, a reduction assignment was subject to *Flattening*.

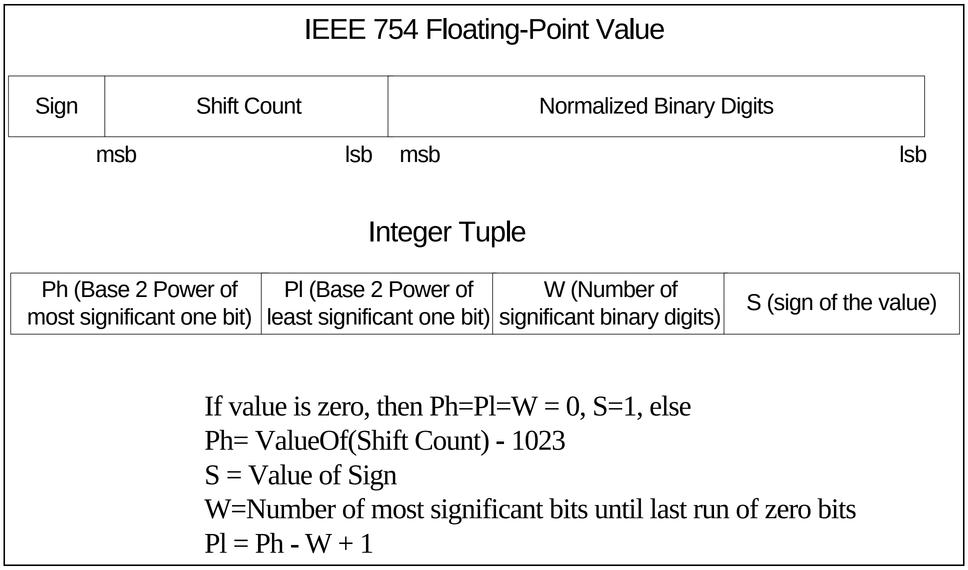
In Flattening, each reduction tree element is assigned an operation class based on the operator of its operands. Addition and Subtraction are one class; these have a Class Value of zero. Multiplication and Division are the other class; these have a Class Value of one. Assigning these class values is called *Operator Classification*. Associated with each reduction tree element is an integer array “nReduxAnalyzeCounts.” This array consists of five (the value of defined constant “m_nReduxAnalyzeCountsSize”) elements. Each element is the count of each operator type (Other, Add, Subtract, Multiply, or Divide, respectively) of all reduction tree elements below it.

After a reduction assignment is populated with final calculation values, each reduction tree element is visited top down. When the “Other” count is zero and there are elements of only one class present with at least two reduction analysis elements below, the subordinate reduction element tree is flattened by procedure “procFlattenExpr,” Appendix E. This linearizes all the operand values of all calculations into one contiguous chain of values and operations.

Floating-Point Tuples

Generating the most precise result for a calculation requires reordering the sequence in which operations are performed. Integer-based algorithms are used to maximize speed. A floating-point number is represented by a tuple of four integer values. The first integer value (“Ph”) is the power of two of the most significant non-zero bit, the hidden one bit. The second integer value ("Pl”) is the power of two of the least significant non-zero bit. The third integer value (“W”) is the number of significant bits. This latter value (“W”) is the difference plus one of the values of “Ph” and “Pl.” The fourth integer value (“S”) is the sign of value. These are denoted by <Ph, Pl, W, S>. For example, the value 1 is represented by T(1) = <0,0,1,1>. The value -17.25 is represented by T(-17.25) = <4,-2,7,-1>. T(0) is defined to be <0, 0, 0, 1>. Figure 3 shows how these values are obtained from the IEEE-754 storage definition.

Figure 3 - Relationship Between IEEE-754 Floating-Point Value and Integer Tuple


IEEE 754 Floating-Point Value
Sign
Shift Count
Normalized Binary Digits
msb
lsb
msb
lsb
Integer Tuple
Ph (Base 2 Power of most significant one bit)
Pl (Base 2 Power of least significant one bit)
W (Number of significant binary digits)
S (sign of the value)

If value is zero, then Ph=Pl=W = 0, S=1, else
Ph= ValueOf(Shift Count) - 1023
S = Value of Sign
W=Number of most significant bits until last run of zero bits
Pl = Ph - W + 1

The widths may be used in computing the width of the result. The instances where the widths do not exceed the storage width of a floating-point number are not affected by the floating-point unit's round-off policy. These operations may be performed in any order. Table 2 shows the maximum (worst case) widths for the arithmetic operations on two positive floating-point values.

Table 2- Worst Case Tuple Limits on Arithmetic Operation on Two Positive Floating-Point Values

Operation	Ph	Pl	S
A + B	Max(A.Ph, B.Ph)+1	Min(A.Pl, B.Pl)	A.S * B.S
A – B	Max(A.Ph, B.Ph)+1	Min(A.Pl, B.Pl)	If A >= B then 1, else -1
A * B	A.Ph + B.Ph	A.Pl + B.Pl	A.S * B.S
A / B	A.Ph - B.Ph	Undefined	A.S * B.S

Examples of these operations can be seen in Figure 4.

Figure 4 – Examples of Tuple Mathematics

Sum-
Decimal: 13.25 + .625 = 13.875
Binary: 1101.01 + 0.101 = 1101.111
Tuple: <3, -2, 6, 1> + <-1, -3, 3, 1> = <3, -3, 7, 1>
Worst Case Limits: = <Max(3,-2) + 1, Min(-2,-3),W=(Ph - Pl + 1), A.S * B.S> = <4, -3, 8, 1>

Difference-
Decimal: 13.25 - .625 = 12.625
Binary: 1101.01 - 0.101 = 1001.101
Tuple: <3, -2, 6, 1> - <-1, -3, 3, 1> = <3, -3, 7, 1>
Worst Case Limits: = <Max(3,-2), Min(-2,-3),W=(Ph - Pl + 1), 1) = <4, -3, 8, 1>

Product-
Decimal: 13.25 * .625 = 8.28125
Binary: 1101.01 + 0.101 = 1000.01001
Tuple: <3, -2, 6, 1> - <-1, -3, 3, 1> = <3, -3, 7, 1>
Worst Case Limits: = <3 + -1 + 1, -2 + -3,W=(Ph - Pl + 1), 1 * 1> = <3, -5, 9, 1>

Division-
Decimal: 13.25 / .625 = 21.2
Binary: 1101.01 / 0.101 = 10101.0011001100110011001100110011001100110011001100110011…
Tuple: <3, -2, 6, 1> / <-1, -3, 3, 1> = <4, -48, 53,1> Note: Limited only by storage width of double format
Worst Case Limits: = <3, Undefined, Undefined, 1 * 1> = <4, Undefined, Undefined, 1>.

Actual values may create values whose tuples contain smaller widths. Examples of this are shown in Figure 5.

Figure 5 - Examples of Exceptions to Worst Case Tuple Limits on Arithmetic Operation on Two Positive Floating-Point Values

```
Sum- (Width of one rather than eight)
  Decimal:  31 + 33 = 64
  Binary: 11111 + 100001 = 1000000
  Tuple: <4, 0, 5, 1> + <5, 0, 6, 1> = <6, 6, 1, 1>
  Worst Case Limits: = <Max(4,5) + 1, Min(0,0),W=(Ph - Pl + 1), A.S *
B.S> =
    <6, 0, 8, 1>

Difference: (Width of zero rather than seven)
  Decimal:  8.125 - 8.125 = 0
  Binary: 1000.001 - 1000.001 = 0.0
  Tuple: <3, -3, 7, 1> - <3, -3, 7, 1>  = <0,  0, 0, 1>
  Worst Case Limits: = <Max(3,3), Min(-3,-3),W=(Ph - Pl + 1), Sign(>) =
    <3, -3, 7, 1>
```

Calculation Chains

A calculation chain is defined as a series of consecutive reductions in which no more than one operand is the result of the preceding calculation. The other operand must be a known value or variable name. Listing 12 shows the structure of a calculation chain element. A calculation chain begins either at a reduction tree element where both operands are known values or at a reduction where both operands are calculated results (reduction tree elements). The calculation chain will terminate either at the top of the reduction tree element or at a reduction tree element where it and another reduction tree are the operands. In the latter case, the reduction tree element where both calculation chains are operands starts a new calculation chain.

Listing 12- Calculation Chain Element

```
typedef struct tagCalcChain_struct
{ /* CalcChain_struct */
    struct tagCalcChain_struct *pPrev, *pNext;
    ReduxElement_struct *pLeafRE, *pNodeRE, *pNextRE;
    int nMaxDepth;
    int nGenIx;
    int nResIndex;
    int nDist;
    ResType_Struct Value;
    struct tagCalcChain_struct *pParentCC, *pPrevCCs[2];
    struct tagCalcChain_struct *pPrevCalcCC, *pNextCalcCC;
    int nParentCCArgIx;
    int nElesCount;
    FlPtTuple_struct FlPtTuple;
} CalcChain_struct;
```

There may be no reduction tree elements in a calculation chain. This is the case when both operands to a calculation chain element are calculation chain elements. In this case, the reduction tree element beginning the calculation chain (the “pLeafRE” reduction element) is also the calculation chain element’s “pNodeRE” reduction element.

Calculation chain elements are assigned a sequential result index (property “nResIndex”) that gives the relative sequence in which the calculation chain elements are processed later to generate floating-point processor code. The result index is generated based on the distance a calculation chain element is from the top calculation chain element. This distance is stored in the calculation chain element property “nDist.” The most distant calculation chain elements have the lowest index. The calculation chain elements with the same distance from the top calculation chain element have sequentially generated index numbers. The calculation chain Element property “nGenIx” contains the zero-based iteration number the distance from the top element is computed. It is initially set to negative one to denote the distance to the top element has not been computed yet.

Algorithms Employed

Adding a value of 2^{50} to a value of 2^{-50} yields a value that is 101 bits wide, possibly exceeding the width of the FPU’s internal registers. Since only the most significant bits are stored, the 2^{-50} value is effectively ignored. However, in a calculation, there may be other addends that, when added with the 2^{-50} value, cause a change the sum from a value of 2^{50} to a different value. If the operation of adding a value of 2^{50} to a value of 2^{-50} is performed first, this carry may not occur and an incorrect value might result.

This algorithm assures that the values of all operands contribute to the result for what are called *additive chains*. An additive chain is defined as a sequence of additive floating-point operations. An additive chain is broken into two additive chain components- an increasing component consisting of all increasing operations and a decreasing component consisting of all decreasing operations. The value of an additive chain is the sum of the two components.

In the case of an additive chain, all operators are either add or subtract operations. An additive operation either causes a sum to either increase or decrease. Either adding a positive value or subtracting a negative value increases the value of a sum. These operations are called *increasing operations*. Either subtracting a positive value or adding a negative value decreases the value of a sum. These operations are called *decreasing operations*.

The first floating-point operation of an additive chain component is on the operand with the lowest magnitude of its least significant bit. The next floating-point operation of the component is the operand with the lowest magnitude of its least significant bit from the remaining operands. Since the component consists of only either increasing or decreasing operations, there is no likelihood for catastrophic cancellation in the component. Since the values with the least significant (right) bits are being summed first, and since carried values propagate from right to left, a precise sum is constantly maintained.

Calculation Tree Length Reduction

After several generations of the same history variable calculation, the number of reduction tree elements in the calculation tree becomes very large. In the case where most independent variables in the calculation are themselves calculation history variables, this number may grow exponentially. The need for some method of truncation maintaining the precision of the calculation becomes apparent. This is done by what is called “Bottom-Up Pruning.”

Bottom-Up Pruning is performed as follows. A threshold variable named “m_MaxREsCount” is set that limits the number of reduction tree elements. If the number of reduction tree elements in the finally populated reduction tree exceed this threshold, the excess reduction tree elements are removed. A trait of each calculation chain element is that it contains the resultant value of the reduction tree elements below it. Simply put, the computed value of a calculation chain element is the computed value of its reduction tree. If the computed value of that calculation chain replaces the operand of its parent reduction element, that calculation chain element with all its reduction tree elements can be removed from the tree

The procedure “procTruncateCCs,” Listing 13, performs “Bottom-Up Pruning” until the threshold count is reached. The topmost calculation chain element is never affected by this. If only the topmost calculation chain element remained, no further pruning continues even if the topmost calculation chain element has more than the threshold number of reduction tree elements. Restricting the topmost calculation chain element to the threshold count is a problem for future work.

Listing 13- Procedure “procTruncateCCs”

```
_proc int procTruncateCCs()
{/*procTruncateCCs- makes sure REs do not exceed threshold count */
    int nCountCCs, nMinGenIx, nCCREsCount;
    int n1TotalElesDeleted, n2TotalElesDeleted, n3TotalElesDeleted;
    ReduxElement_struct *pLeafRE;
    n3TotalElesDeleted = 0;
    for (;;)
    { /* Determine number of Calc Chains */
        CalcChain_struct *pNextCC, *pThisCC, *pParentCC;
        nCountCCs = 0; nMinGenIx = -1; nCCREsCount = 0;
        procCountCCs(m_pTopCC, &nCountCCs, &nMinGenIx, &nCCREsCount);
        if (nCCREsCount <= m_MaxREsCount) break;
        n2TotalElesDeleted = 0;
        for (pNextCC = m_pFirstCC; pNextCC != NULL; )
        { /* Test if this is a bottom node */
            pThisCC = pNextCC;
            pNextCC = pThisCC->pNext;
            if (pThisCC->nGenIx == nMinGenIx)
            { /* This CC can be  pruned */
                pParentCC = pThisCC->pParentCC;
                if (pParentCC!=NULL)/* Do not prune the top node */
                { /* This is not the top node */
                    ReduxElement_struct *pParentRE, *pNodeRE;
                    pParentRE = pThisCC->pNextRE;
                    pNodeRE = pThisCC->pNodeRE;
                    pParentRE->pREOperands[pThisCC->nParentCCArgIx] = NULL;
                    pParentRE->SEOperands[pThisCC->nParentCCArgIx].pReduxElement =
NULL;
                    pParentRE->SEOperands[pThisCC->nParentCCArgIx].nValType = 7;
                    pParentRE->SEOperands[pThisCC->nParentCCArgIx].DataValue =
pNodeRE->SEOperator.DataValue;
                    pParentRE->SEOperands[pThisCC->nParentCCArgIx].FlPtTuple =
pNodeRE->FlPtTuple;
                    pParentRE->nArgsType =
                      ((pParentRE->SEOperands[0].pReduxElement == NULL) ? 0: 1) +
                      ((pParentRE->SEOperands[1].pReduxElement == NULL) ? 0: 2);
                    pParentCC->pPrevCCs[pThisCC->nParentCCArgIx] = NULL;
                } /* This is not the top node */
                n2TotalElesDeleted += pThisCC->nElesCount + 1;
            } /* This CC can be  pruned */
        } /* Test if this is a bottom node */
        n3TotalElesDeleted += n2TotalElesDeleted;
    } /* Determine number of Calc Chains */
    return 0;
}/*procTruncateCCs- makes sure REs do not exceed threshold count */
```

The unfortunate effect of having to remove reduction tree elements is that the calculations that went to create the value of the calculation chain element are lost. The value of the calculation chain element is the truncated storage value of the computed value of its reduction tree. Consequently, accuracy in subsequent calculations may suffer because of this truncation. This was examined in Test Case Six.

Regular Calculation Computation

Once a calculation has been loaded into memory from its stored definition, it is immediately executed using previously computed values. The value returned at this time is called its "regular value" and is the variable *dReg* in the "Results" section. This is the traditional value of the calculation. The *dReg* value is computed using previously computed *dReg* values; it is not computed using values from calculation histories.

A direct assignment returns the value of the operand as the calculated result.

A reduction assignment is computed by the recursive procedure "procResultValue," Appendix F. The value of a reduction assignment is calculated recursively as the topmost reduction tree element's operator on its operands.

Calculation History Computation

A calculation history assignment calculation requires several steps. After the regular value of the calculation has been computed, the calculation tree is populated with the previously computed values of its variable operands. If an operand is a calculation history variable, the operand is replaced with the previously computed value only if it was a direct assignment of a non-calculation history value. Otherwise, it is replaced by the operand's last saved calculation history.

Reduction analysis elements are created recursively starting with the reduction tree element at the top of the reduction tree.

The procedure "procCountREOpTypes," Listing 14, populates each reduction analysis element with the number of reduction analysis elements in its reduction analysis sub tree with the count of each operator type. This is part of *Operator Classification*.

The recursive procedure “procFlattenRAE,” Listing 15, determines if the reduction tree below its referenced reduction tree element can be flattened based on this operator classification. If so, the reduction tree below it is flattened and the “pFlattenRAE” property of the reduction analysis element is set to point to it. Otherwise, the “pFlattenRAE” property remains NULL.

Listing 14- Procedure “procCountREOpTypes”

```
_proc int procCountREOpTypes()
{ /* procCountREOpTypes- Sets up metrics values for a reduction
tree */
    ReduxAnalyzeElement_struct *pThisRAE, *pParentRAE;
    ReduxElement_struct *pThisRE;
    int nArgIx;
    for (pThisRAE = m_pFirstRAE; pThisRAE != NULL; pThisRAE =
pThisRAE->pNext)
    { /* Type this operator token */
        pThisRE = pThisRAE->pRE;
        nArgIx = 0;
        switch (pThisRE->SEOperator.TokenData.nTokenType)
        { /* switch (nTokenType) */
        case nTT_Plus: nArgIx = 1; break;
        case nTT_Minus: nArgIx = 2; break;
        case nTT_StarSingle: nArgIx = 3; break;
        case nTT_DivSingle: nArgIx = 4; break;
        } /* switch (nTokenType) */
        if (nArgIx >= 0)
        { /* Got a valid Operator */
            pThisRAE->nReduxAnalyzeCounts[nArgIx]++;
            for (pParentRAE = pThisRAE->pParent; pParentRAE !=
NULL; pParentRAE = pParentRAE->pParent)
              pParentRAE->nReduxAnalyzeCounts[nArgIx]++;
        } /* Got a valid Operator */
    } /* Type this operator token */
    return m_nRAEsCount;
} /* procCountREOpTypes- Sets up metrics values for a reduction
tree */
```

Listing 15- Procedure “procFlattenRAE”

```
_proc int procFlattenRAE(ReduxAnalyzeElement_struct
*pFlattenRAE, int *nCntUnFlattened)
{ /* procFlattenRAE- */
    int nCntClass1, nCntClass2, nGo, nRetVal;
    nCntClass1 = pFlattenRAE->nReduxAnalyzeCounts[1] +
pFlattenRAE->nReduxAnalyzeCounts[2];
    nCntClass2 = pFlattenRAE->nReduxAnalyzeCounts[3] +
pFlattenRAE->nReduxAnalyzeCounts[4];
    nRetVal = 0;
    nGo = ((nCntClass1 > 0)? 1: 0) + ((nCntClass2 > 0)? 2: 0);
    if (nGo != 0)
    { /* Arguments remain */
        if (nGo == 3)
        { /* Not ready yet */
            *nCntUnFlattened += 2;
            if (pFlattenRAE->pArgs[0])
            { /* Must check this argument */
                procFlattenRAE(pFlattenRAE->pArgs[0],
nCntUnFlattened);
            } /* Must check this argument */
            if (pFlattenRAE->pArgs[1])
            { /* Must check this argument */
                procFlattenRAE(pFlattenRAE->pArgs[1],
nCntUnFlattened);
            } /* Must check this argument */
        } /* Not ready yet */
        else
        { /* Children all of one or the other */
            pFlattenRAE->pFlattenedExpr =
procFlattenExpr(pFlattenRAE->pRE);
            nRetVal = 1;
        } /* Children all of one or the other */
    } /* Arguments remain */
    return nRetVal;
} /* procFlattenRAE- */
```

Once this is accomplished, the elements of each flattened reduction analysis element tree are restructured into a linked list of the operands used to create that flattened expression. The reduction tree elements and their reduction analysis elements that were part of the original expression are removed and a new reduction tree is created from the flattened expression.

Precision retention is performed at this point. This is performed on an *additive* operand chain components by ordering the operations with the lowest values of their least significant bits first. This assures that all carries from previous operations in the operand chain would be reflected in the final computed value. For a *multiplicative* chain, operations were sequenced by increasing number of significant bits. Operands with values of positive one are removed from the operand list for a multiplicative chain.

This new reduction tree replaces the operand of the parent reduction tree element. A new reduction analysis element tree is created starting with the top of the new reduction tree. The parent property of the new reduction analysis element tree is set to that of the original reduction analysis element and its parent's operand property set to point to the top of the new reduction analysis element tree.

Calculation chains are created based on the new reduction tree in no particular order. An initial order is accomplished by determining the calculation chains that are operands to their parents by recursive procedure "procCreateCCLinkedList," Listing 16. The structure of the calculation chain tree parallels the structure of the reduction tree.

Listing 16- Procedure "procCreateCCLinkedList"

```
_proc int procCreateCCLinkedList(CalcChain_struct *pTopCC)
{ /* procCreateCCLinkedList- Create linked list of CCs used in
calculation */
    int nArgIx;
    LinkAppendNewX(pTopCC, m_pFirstCalcCC, m_pLastCalcCC,
pPrevCalcCC, pNextCalcCC);
    for (nArgIx = 0; nArgIx < 2; nArgIx++)
    { /* Add children, if any, to list */
        CalcChain_struct *pArgCC;
        pArgCC = pTopCC->pPrevCCs[nArgIx];
        if (pArgCC)
        { /* Add this one to linked list */
            procCreateCCLinkedList(pArgCC);
        } /* Add this one to linked list */
    } /* Add children, if any, to list */
    return 0;
} /* procCreateCCLinkedList- Create linked list of CCs used in
calculation */
```

Floating-point Code Generation

The procedure “procCreateFPUCode,” Appendix G, generates the assembly language equivalents of the machine language that would be generated if the calculation were performed immediately in the floating-point processor. It assumes a two element floating-point stack and an Intel ™ 80x87 floating-point processor. Operations are performed on the top element of the stack. The second element of the floating-point stack acts only as an operand. Loading a value onto the top of the floating-point stack pushes whatever was on the top of the floating-point stack to the second element of the floating-point stack. The second element of the floating-point stack may be used as an implicit operand in almost all two-operand floating-point arithmetic operations.

The procedure “procCreateFPUCode” generates floating-point by processing calculation chains in a “staging array” named “pCCsSorted.” Code generation begins with the longest unprocessed calculation chain tree. Once the bottom-most calculation chain element has been found, its pointer is appended to the initially empty staging array of calculation chain element pointers. The order in which calculation chain element pointers are inserted into this staging array is the order in which their floating-point code is generated. Once a pointer to the starting calculation chain element has been inserted into the staging array, the calculation chain element is flagged as having been processed and its sibling operand is tested. If the sibling is not a calculation chain element or has already been processed, the parent calculation chain element is flagged as being done. Otherwise, this process is repeated for the sibling calculation chain element beginning with its bottom-most calculation chain element.

When a calculation chain element is flagged as being done, its parent calculation chain element was tested. If there is no parent calculation chain element, all calculation chain elements have been staged and the process terminated. Otherwise, if both of its parent's operands have been processed or its sibling operand is not a calculation chain element, the parent is flagged as being done. Otherwise, the sibling starts the next calculation chain to be generated beginning with its bottom-most calculation chain element.

Floating-point unit code is generated by sequentially processing the calculation chain element pointers in the staging array. Calculation chains are processed by creating floating-point instructions for each of its reduction elements in the calculation chain. For the first floating-point instruction, the left operand of the first reduction element is loaded onto the floating-point stack. Subsequent operands of the calculation chain act on the top of the stack.

If the previous calculation chain element has the same parent, the calculation chain element following the current calculation chain element is the parent calculation chain element. In this case, after the reduction tree elements in each calculation chain element have been processed, the value of neither calculation chain has been stored into program memory. Their values remain on the floating-point stack to be operated on by the operator of the next calculation chain element. This avoids precision loss associated with storing the value in a floating-point register into program memory. The top of the stack is saved into program memory for future use only when the current calculation chain element is not the sibling to its predecessor, is not the sibling to its successor, and the successor is not its parent.

As floating-point unit instructions are being generated, they are written to a temporary file named “FPUCode.tmp.” The instructions refer to operands, none of which has been defined in the temporary file. The operand values are maintained by procedure “procFPUOperandValue,” Appendix H, as a linked list. When an operand value is passed it, the procedure returns the zero-based count of the operand’s occurrence in the operand linked list if it finds the value in the linked list. Otherwise, the operand is appended to the linked list and the number of values tested is returned.

After all the FPU code has been generated, a procedure to execute it and display it is stored in a file named “CHProcEng.FPU.” This procedure consists of four components- data operand values, result value storage, assembly language code to perform the calculation, and “C” language code to output the results. This file is inserted later into a “C” language “main” module named “FPU.c,” Appendix I, for execution.

A “C” language union named “IEEE754Real8_struct,” Listing 17, defines storage for the floating-point operand values in the program “FPU.c”, into which the file named“ CHProcEng.FPU” is inserted. Floating-point data operand values are expressed as 32-bit integer elements using the “lVals” property. It is more accurate to use the hexadecimal representation of the binary 32-bit integer components than an ASCII representation of the decimal values and subject them to possible rounding during output to decimal conversion.

Listing 17- “C” Language Union “IEEE754Real8_struct”

```
typedef union /* IEEE754Real8_struct */
{ /* IEEE754Real8_struct */
    long lVals[2];
    double dVal;
} IEEE754Real8_struct;
```

Storage in the “CHProcEng.FPU” file for the results of each calculation chain Element’s value calculation is allocated by defining an “IEEE754Real8” variable named “dRes” with appended the decimal representation of its sequential index. Each operand value is assigned a variable name starting with the lower-case letter “d” with the decimal representation of its occurrence sequence in the linked list maintained by procedure “procFPUOperandValue.”

The Floating-Point Unit Processor Program

It was not possible to directly process the floating-point processor code in CHProcEng. When executed directly in CHProcEng, floating-point processor code generated “on the fly” corrupted the floating-point stack. This resulted in fatal run-time errors. It was necessary to create a separate and simpler program to avoid this problem.

This was remedied by breaking the program into two source files. File “CHProcEng.FPU” contained the procedures to perform the calculations generated by the CHProcEng application. A second file, named “FPU.c,” Appendix I, was the primary source file to the process floating-point processor operations. The contents of file “CHProcEng.FPU” were inserted into a section preceding the “main” procedure of program “FPU.c.” Procedure declarations at the beginning of the source file were deleted or added as necessary; the same happened to the procedure calls that were part of the “main” procedure.

The edited “FPU.c” program was compiled by the batch procedure “MKCProg.bat,” Listing 18, at the DOS prompt and executed. The output was redirected into a file named “FPU.out.”

Listing 18- Batch Procedure “MKCProg.bat”

```
@echo off
if "%1" == "" GoTo NoJob
set CSrc=%1
path C:\Program Files\Borland\CBuilder6\bin;%WinDir%\System32
:CompileSource
   bcc32 -O -5 %CSrc%.c  >%CSrc%.err
   if errorlevel 1 goto Errors
   goto OK
:Errors
   c:\windows\notepad %CSrc%.err
   goto cleanup
:OK
   if exist %CSrc%.ok_ del %CSrc%.ok_
   if exist %CSrc%.ok copy %CSrc%.ok %CSrc%.ok_
   copy %CSrc%.c %CSrc%.ok
:cleanup
   for %%a in (%CSrc%) do if exist %%a.obj del %%a.obj
   goto Fini
:NoJob
   echo You must specify an C Language Source File  Name
 :Fini
```

Specific Procedures Employed

Procedure Step 1- Test System Build

Each program module was created using a text editor. At various phases, each program was saved and compiled with default input file names. If no command line file names were present, the default file names would be used. A DOS window was opened. The batch procedure named “MkCProg.bat” compiled the program module and created the executable program file. The program was executed in the DOS window with its output being redirected to a persistent storage file for debugging analysis.

If a problem was detected with the DOS execution, the program was compiled and executed as a Borland C++ Builder “C” project. This allowed pinpointing the regions that needed addressing. The program module was modified to correct the problem and testing resumed as in the previous paragraph.

Procedure Step 2- Testing the System

Test cases were chosen. The problem testing each case to be examined was encoded as a CHFort text file. The CHFort text file was compiled by the CHFort application to generate the object text file used as input to the CHProcEng virtual machine application.

The CHProcEng virtual machine generated a binary log file containing test case results of all calculations that were executed by the virtual machine application. The CHProcEng virtual machine application also created a floating-point unit execution file containing procedures to perform the same calculations using an Intel 80x87 floating-point unit.

A binary to text conversion program was run on the binary log file generated by CHProcEng to output a formatted textual result file for the virtual machine results.

The floating-point unit execution file was inserted into a “C” language program file containing a “main” procedure. The “C” language program file containing the “main” procedure was compiled and executed. Its output was redirected into a persistent storage file. This output file contained the results of each calculation as it was executed on the target floating-point unit.

A “C” language program with the same calculations as the CHFort test case was created. This file had additional code to output formatted results. It was compiled and executed. Its output was redirected to a persistent storage file.

Procedure Step 3- Results Analysis

A “C” language program was modified to generate a formatted output depending upon the nature of the test case. This program matched the results from the log, floating-point, and baseline files and output formatted results for individual calculations. The formatted results output was redirected to a persistent storage file. This persistent storage file supplied the contents of the results tables in Chapter 4. Analyzing the formatted results output provided information leading to the findings in Chapter Four.

Formats for Presenting Results

Full results are presented as structured tables. Each table consists of individual calculations and calculation details. The calculation number is displayed. If a predicted value was computed, it is displayed as either the complete value (using variable name “dValue”) or the fractional value of interest (using variable name “dFrac”). Below this line are header lines for each of the decimal values displayed below it. The decimal values are displayed on a single line for the result of the “C” language baseline program, the CHProcEng computed result, and the floating-point unit computed result.

Individual lines following the decimal values display the binary values of each of these results. A label identical to that above its decimal value precedes the binary value.

Within a test case analysis, the results table may be subset or the results are displayed in a different format. If that is the case, the full results table is displayed as an appendix. The results displayed with the test case are abbreviated to present only the information needed to discuss the findings.

Resource Requirements

Development System

Off the shelf tools were used to develop the compiler and processing engine.

The Borland C++ compiler version 5.0 was used to develop the compiler for the FORTRAN source code. The source of the FORTRAN language grammar is a subset of the FORTRAN grammar described by the Lahey Fortran 90 Language Reference [Lahey]. The processing engine was developed using the same C++ compiler.

The developmental operating system was Microsoft Windows XP™.

The text editor used was Visual SlickEdit Version 8.0 from SlickEdit, Inc.

Test System

Each test case was performed on a Microsoft Windows XP system.

The source (CHFort.fch) editor was the same Visual Slick Edit program used in the development system.

Each program ran in a DOS window.

Reliability and Validity

The source code used to perform the primary features required in this architecture have been listed in this chapter or the in an appendix.

The source files used for each of the test cases are displayed with their test cases. Appendixes supply any other supporting information.

Summary

This chapter described the general computing architecture that was developed to address the issue of maintaining run-time precision in propagated calculations. This architecture required communicating the nature of a calculation from the programmer to the processor. This entailed a front-end compiler, a processing engine, and a floating-point processor interface program.

The compiler was shown to be a Look-Ahead, Left Recursive parsing compiler. The compiler created an output file that was the input to the processing engine. The calculation code was stored in the output file in a stack format generated by a LALR grammar.

The processing engine was shown to process two different types of assignments-direct assignments and reduction assignments. The direct assignments were calculated in the traditional method. Reduction assignments were shown to be processed with more complexity. This complexity required procedures to store and load computed calculation histories, reduce the complexity of the calculation, restructure the calculation, compute the results traditionally, and generate floating-point processor code. The names of these procedures were given and explanations of their operation were presented in this chapter.

Operations on run-time data were described to prepare the calculation for integer-based algorithms. These operations were Operation Classification and Flattening. The use of calculation chains and floating-point tuples was discussed to restructure a calculation into a series of calculations without requiring intermediate storage. An integer-based algorithm was described do structure a calculation chain so operand precision would not be lost in a calculation.

A floating-point processor application was described to process the floating-point processor code generated by the processing engine. This was shown to be a small “C” language program combining “C” language code and assembly language.

Chapter 4

Results

Overview

This chapter presents a discussion of test cases using the calculation history system described in Chapter 3. All test cases were developed by developing a problem text file in the CHFort language. The problem text file was input to the CHFort compiler to create a textual output object file in the input format of the CHProcEng application. The text of the output object file was executed by the CHProcEng computing application.

Results were computed for six test cases. Each test targeted a particular aspect of the calculation history approach and is separately described. The nature and goal of each test case is described in the test case. The input and results are presented in Listings, Tables, and Appendixes. The findings are analyzed by textual commentary.

Many test cases required multiple experiments to establish conclusive results. Some experiments required minor alterations to the CHProcEng virtual machine application. These changes were to test different algorithms and thresholds.

Predicted values were determined for the computational results of each experiment. In some cases, the predicted values were determined by mathematical expressions. In the results tables, these are the values named "dValue" or "dFrac." The "dFrac" values were displayed when only the fractional part of a resultant value was of interest.

Many of the experiments were executed using a "C" language baseline program. The "C" language baseline program was considered representative of the "regular approach" toward mathematical computing. Such a program would generate the same computational results as would a computational language like FORTRAN. The values of the regular approach in the result tables are given by the values name *dReg*.

Each experiment entailed computing the results of a calculation using calculation histories. Two values were computed for each calculation. A value was computed if the calculation was performed entirely with the CHProcEng software application using "case" statements and "C" programming language constructs. For example, the operation $A = B + C – D$ would be performed in three separate steps: 1. $dRes = B + C$, 2. $dRes = dRes – D$, and 3. $A = dRes$. Each time the value *dRes* is computed, it must be truncated from the full width value in the floating-point unit registers to the smaller width storage of program memory. The software approach of computing a calculation based on calculation histories would be, therefore, sensitive to the number of floating-point operations. The software computed value of a computation is represented in the result tables as the value "dRes."

The other value came from assembly language code generated by CHProcEng for the floating-point unit ("FPU"). The FPU was considered a stack machine with two stack elements. Floating-point operations were structured so that as many operations as possible were performed on the top stack element without intermediate storage.

The CHProcEng virtual machine produced the assembly language code as a separate procedure for the computation of the value of every calculation history variable. These procedures were copied into a base "C" language program. After being compiled, this "C" language program executed each computation completely within the FPU. For example, the operation $A = B + C – D$ would be performed in four separate steps: 1. fld *B*, 2. fadd *C*, 3. fsub *D*, and 4. fstp *A*. The only truncation would be incurred when storing the final result to the variable *A*. The results were captured in a text file. The value named *dFPU* shows this value in the results tables. The target goal of this dissertation is that the value named *dFPU* be either the floating-point value of the exact value, or, if it does not exist, the nearest floating-point value to the exact value.

Scientific notation is used extensively in this chapter. The value .006123 can be expressed in scientific notation as 6.123e-3. The value 6.123e3 is in decimal form the value 6123. Also, the character “*” is used to denote multiplication.

Test Case One

This test case tested the general validity of using calculation histories in a summation problem. The summation algorithm maintains precision by processing successive additions based on the power of two of the least significant non-zero bit. This method assures all significant bits contribute to the final sum. Precision loss is also reduced because Flattening creates long computational chains avoiding unnecessary truncation of intermediate values for program storage. The series of calculations is performed using the full width of the floating-point unit’s registers.

This test case was designed to be similar to a problem to integrate the area beneath a curve using the MidPoint estimation rule [Smith and Minton]. The MidPoint estimation rule divides the region to be integrated into equal, contiguous widths. The area that each width contributes to the total area is given as a product of its fixed width and the value of the function at the mid-point of that width. For a straight-line function, the value of the function of the next contiguous width is a constant value added to the value of the preceding width. As a result, the area is computed as the sum of a series of additions.

The problem chosen had a width of one and a slope of one. The line was divided into fifty equal intervals. This particular choice of values generated values that could be predicted without complicated methods.

The source code is shown in Listing AA-1. It should be noted that the only operation used to create the sums *dYSum* and *dXMid* is addition. The source code was input to the CHFort compiler. The text of the output object file, which is listed in Appendix AA-1, was input to the CHProcEng virtual machine application. The “C” language baseline program is shown in Listing AA-2. Appendix AA-2 displays the full results for the test.

Listing AA-1- CHFort Source Code of Test Case One

```
Double Precision, History :: dYSum, dXMid
Double Precision dX0, dSpan, dSpanIncs,
dSpanDelta
dX0 = 0.
dSpan = 1.
dSpanIncs = 50.
dSpanDelta = dSpan /dSpanIncs
dXMid0 = dX0 + dSpanDelta / 2.
dXMid = dXMid0
dYSum = 0.
100 Continue
If (dXMid .gt. dSpan) GoTo 999
dYSum = dYSum + dXmid
dXMid = dXMid + dSpanDelta
Go To 100
999 Continue
End
```

Listing AA-2- “C” Language Baseline Program for Test Case One

```
#include "FPU.h"
int main()
{ /* main */
    IEEE754Real8_struct Y;
    double dYSum, dXMid, dXMid0;
    double dX0, dSpan, dSpanIncs, dSpanDelta;
    int n;
    char *pszBits;
    /**/
    dX0 = 0.;
    dSpan = 1.;
    dSpanIncs = 50.;
    dSpanDelta = dSpan /dSpanIncs;
    dXMid0 = dX0 + dSpanDelta / 2.;
    dXMid = dXMid0;
    dYSum = 0.;
    for ( n = 0;dXMid  <= dSpan;)
    {
        dYSum = dYSum + dXMid;
        dXMid = dXMid + dSpanDelta;
        Y.dVal = dYSum;
        printf("\nCalculation index: %i\n", ++n);
        pszBits = procIEE754DblToBin(Y.dVal);
        printf(" dCProg: %25.20g, Binary: %s\n", Y.dVal,
pszBits);
        free(pszBits);
    }
    return 0;
    /* Number of Store Operations: 1 */
} /* main */
```

Two cases of interest are when the sum should be .25 and 25. The significance of these values is that the fractional portion of their binary IEEE-754 representation is zero. These cases, seen in Table AA-1, occur at the iterations five and fifty. Complete test results are seen in Appendix AA-2.

Table AA-1- Special Results of Test Case One

```
Iteration Number: 5
           dReg                       dRes                       dFPU
           0.25              0.2499999999999999445               0.25
dReg: +0.010000000000000000000000000000000000000000000000000000
dRes: +0.0011111111111111111111111111111111111111111111111111110
dFPU: +0.010000000000000000000000000000000000000000000000000000
Original REs Count: 1, Computational REs Count: 14, Saved REs Count: 14

Iteration Number: 50
           dReg                       dRes                       dFPU
   25.00000000000000355       24.99999999999955591                25
dReg: +11001.000000000000000000000000000000000000000000000001
dRes: +11000.111111111111111111111111111111111111111110000011
dFPU: +11001.000000000000000000000000000000000000000000000000
Original REs Count: 1, Computational REs Count: 1274, Saved REs Count: 1274
```

The regularly computed variable *dReg* shows near complete accuracy. This is most likely a result of the triviality of the calculation and the narrow range of the addends. The values of the variable *dRes* show significant error due to precision loss. Although its values are based on calculation histories, each subsequent calculation requires more storage truncations than its previous calculation since each calculation requires performing all previous calculations. To compute the values of *dRes*, procedure “procResultValue” performs arithmetic operations one at a time storing the result of one arithmetic operation into program memory and reloading again from program memory when the result is needed. The operands resulting from previous calculations were also truncated values. A result is an increasing number of storage truncations causing an increasing loss in accuracy.

Flattening, used to compute the values of variable *dFPU,* sequenced the addends into a sequence by which no precision was lost. The result was a calculation of sequential addition operations completely performed within the FPU using the its wide internal registers. There was no intermediate storage to program memory to cause precision loss. All results of the calculation history floating-point unit calculation (variable *dFPU*), shown in Tables AA-1, show precise calculations. This is true when the regularly computed value *dReg* differs from the floating-point unit value *dFPU*. This finding illustrates that the calculation history approach does have merit and should be investigated further.

Test Case Two

The calculation history approach maintains computational precision by reassembling a calculation by adaptive reformulation. This allows a calculation to be restructured into a more precise calculation. This dissertation has developed an algorithm to assure that the significant bits of all operand values in a series of mathematical operations contribute without precision loss. This algorithm uses Operator Classification and Flattening. Operator Classification groups mathematical operations into two operational groups. Theoretically, the sequential operations in each group can be reordered without affecting the resultant value. Flattening is employed to linearize the reduction trees in a calculation so that sequential chains of operations of each group can be isolated.

Once a chain of sequential calculations has been flattened, it can be performed completely within the FPU. The sequence of operations does not have to be dictated by the way the calculation was originally encoded. In fact, the sequence may be modified by the nature of the operators and their operands. This reordering is the primary focus of this dissertation. The primary assertion of this dissertation is that operand values affect the result of a calculation. Reordering the calculation based on the run-time value of the operands can improve the accuracy of the calculation.

In the case of an additive chain, all operators are either add or subtract operations. Depending upon the sign of the operand, each will either increase or decrease the value of a computed result. The virtual machine can process the chain as either one sequential sum of values or as the sum of two sequential sums (summations) of values. One purpose of this test case was to determine if splitting the calculation into two separate components affects the calculated result.

By design, the virtual machine first performs operations that increase the value of the result before operations that decrease the value of the result. To assure all operands contribute to the result, first performing operations where the operands have the lower magnitude of its least significant bit before operations with greater magnitudes of its least significant bit assures that the full precision of all operands are reflected in the sum. If the results of an operation exceed the width of the floating-point register, bits would be truncated in the least significant bit positions and would be, therefore, insignificant to the result. Because operands are ordered in ascending magnitude of the least significant bit, operands do not have significant bits in the dropped positions. This particular ordering, consequently, should cause no loss of accuracy in subsequent calculations.

This test case tested restructuring a series of additions and subtractions as one continuous series of add and subtract operations. Three experiments were performed.

Experiment One used only one sum to compute the result. The primary sequence of the calculation was the ascending magnitude of the least significant bit of an operand’s value. On equal comparisons, increasing operations were performed before decreasing operations. The Intel 80x87 FPU processes internally with a precision of 64 bits [Intel]. The IEEE-754 standard allows a maximum of 53 (one “hidden” and 52 storage) bits for a double-precision value. Since all operations on the sum are in increasing magnitude of the least-significant bit value, no precisions loss should occur.

Experiment Two was similar to Experiment One in that only one sum was used to compute the result. Experiment Two entailed performing the computation as a single series of operations ordered first by the sign of the addend value and then by increasing least-significant bit magnitude. This should create a large positive value to which initially small negative values will be added. If the precision of the sum of the increasing values exceeds 64 bits, addend values with bit magnitudes less then the sum causes round-to-nearest truncation on the addend values. The default truncation in an 80x87 FPU is round-to-nearest truncation [Intel]. This will result in an inaccurate result.

Experiment Three performed the same ordering as Experiment One. But it separated the increasing value operations into one summing component and the decreasing value operations into a second summing component. The value of the result was the sum of these two components. Separating the calculation into two components, each component ordered by ascending value of least significant bits, allowed each component to be computed exactly using the full width of the floating-point unit registers.

The questions examined were:

1. Is it effective to implement Operator Classification and Flattening?
2. Does splitting the computation into an operation on two component parts instead of just one make a difference?
3. Is each method consistently accurate?

In each experiment, the populated calculation histories of each computation were flattened into one continuous series of operations. The result of Flattening is a computation that can be performed without intermediate storage truncation to program memory. It is performed completely on the top element of the floating-point stack. No values are pushed to other floating-point stack elements. The full width of the floating-point unit's registers is used for the computation.

The source code is shown in Listing AB-1. The "C" language baseline code is shown in Listing AB-2. The source code was input to the CHFort compiler. The text of the output object file, which is listed in Appendix AB-1, was input to the CHProcEng virtual machine application.

The values of *1 / 7* and *2 / 7* were chosen since as decimal fractions they are non-terminating. The integer addend of *10000* was chosen to increase the significant bit distance from the integer portion to the fractional portion of the value of the result. The actual value at any iteration *n* can be computed as (*10000n – n/7*). Every seventh iteration should be an integer value. Table AB-1 shows the last eight bits of every seventh result. Full results are shown in Appendix AB-2, where results of Experiment One are in the line with the "a:" prompt, the results of Experiment Two are on the lines with the "b:" prompt, and the results of Experiment Three are on the lines with the "c:" prompt.

Listing AB-1- CHFort Source Code of Test Case Two

```
Double Precision, History ::  Y
A = 1 / 7
B = 2 / 7
Y = 0
N = 0
100 Continue
If (N .ge. 100) GoTo 900
    Y = Y + 10000. + A - B
    N = N + 1
    Go To 100
900 Continue
End
```

Listing AB-2- “C” Language Baseline Program for Test Case Two

```
#include "FPU.h"
int main()
{ /* main */
    IEEE754Real8_struct Y;
    double A, B;
    int n;
    char *pszBits;
    /**/
    A = 1. / 7.;
    B = 2. / 7.;
    Y.dVal = 0.;
    for ( n = 0; n < 100;)
    {
        Y.dVal = Y.dVal +10000. + A - B;
        printf("\nCalculation index: %i\n", ++n);
        pszBits = procIEE754DblToBin(Y.dVal);
        printf(" dCProg: %25.20g, Binary: %s\n", Y.dVal, pszBits);
        free(pszBits);
    }
    return 0;
} /* main */
```

Table AB-1- Computed Least Significant Bits for Test Case Two

		Experiment One		Experiment Two		Experiment Three	
		1 sum, lsb,		1 sum, sign of		2 sums,	
Iter-	CProg	sign of result		result,lsb,#bits		lsb, #nbits	
ation	dReg	dRes	dFPU	dRes	dFPU	dRes	dFPU
7	00000000	00000000	00000000	00000010	00000000	00000000	00000000
14	00000000	00000000	00000000	00000010	00000000	00000000	00000000
21	11111101	00000000	00000000	00000011	00000000	00000000	00000000
28	11111110	00000000	00000000	11110100	00000000	00000000	00000000
35	00000000	00000000	00000000	11110001	00000000	00000000	00000000
42	00000010	00000000	00000000	11101110	00000000	00000000	00000000
49	00000100	00000000	00000000	11101011	00000000	00000000	00000000
56	00000011	00000000	00000000	00010000	00000000	00000000	00000000
63	00000100	00000000	00000000	00010010	00000000	00000000	00000000
70	00000101	00000000	00000000	00010100	00000000	00000000	00000000
77	00000110	00000000	00000000	00010110	00000000	00000000	00000000
84	00000111	00000000	00000000	00011000	00000000	00000000	00000000
91	00001000	00000000	00000000	00011010	00000000	00000000	00000000
98	00001001	00000000	00000000	00011100	00000000	00000000	00000000

The integer portion of all computed values was either that of the correct value, or, where not equal, that of one less than the correct value. The most significant bit shown in Table AB-1 for a value extends left to the decimal point. If it is a one (“1”), the computed value was less than the correct value. Otherwise, the value is equal (if all zeroes) or greater than the correct value.

The floating-point unit values (values *dFPU*) of the calculation histories consistently retain complete precision regardless of number of floating-point stack elements used. The full program storage width of each operand operated on the value already on the floating-point stack. Since all floating-point operations took place on the top register of the floating-point stack, no truncation to program memory was performed. The full width of the sum was maintained at all times, except for final storage to program memory.

Despite occasional agreement with the predicted values, the values (values *dReg*) generated by the "C" language program (regular approach) show increasing error from the target zero value. Accumulative affects of updating a value already truncated by storage program memory can be seen by the values of *dReg*. This is likely due to a result of a round-to-nearest truncation the floating-point unit performs to store a value from its wide registers to narrower program storage.

The values of *dRes* were generated by the virtual machine software executing a calculation in a load, operate, and store fashion for each operation of a calculation. This was only a problem in Experiment Two where operations with lesser significant operands were performed first. What appears to be a form of catastrophic cancellation seemed to be occurring. Operands that were very close to each other were being added and subtracted. The result was that a number of lesser significant bits had to be discarded when added to a much larger value (in this case, the summing variable). Significance that would have caused a more properly stored value was lost.

Computing by software the values of *dRes* initially by increasing/decreasing value function instead initially by value of the least significant bit generated far more accurate results. A review of the results in Appendix AB-2 reveals the values of *dReg* are within a bit value of the floating-point unit computed value of *dFPU* for experiments one and three.

Separating operations depending upon their effect on the result generated precise values. This approach minimizes the possibility of catastrophic cancellation that can adversely affect a result. Also, the proper choice of an algorithm to order the sequence of floating-point operations can assure a more consistently precise result. This especially matters if a computation system is devised to be completely software driven.

The difference in accuracies of each of the methods can be seen by the results in Table AB-1. Implementing Operator Classification and Flattening made it possible to perform the restructuring to use the full precision of each operand to generate a more accurate result.

Test Case Three

Test Case Three was designed to determine if multiplication created a difference in computed values. It is similar to Test Case One, but additions are replaced with multiplications. The source code for this test case is shown Listing AC-1. The initial values in the test case were chosen to be functions of prime values so lesser bit differences would be noticeable. The effect of this test is to develop a calculation that has a value that is a product of a sequence of values. This tested the system to determine the difference in precision that the work in using calculation histories to compute a value would reveal as the lower limits of the double precision value range were approached.

Listing AC-1- Source Code of Test Case Three

```
Double Precision, History :: dYSum, dXMid
Double Precision dX0, dSpan, dSpanIncs,
dSpanDelta
dX0 = 0.
nCnt = 0
dSpan = 1.
dSpanIncs = 199.
dSpanDelta = dSpan /dSpanIncs
dXMid0 = dX0 + dSpanDelta / 2.
dXMid = dXMid0
dYSum = 1.
100 Continue
        If (nCnt .ge. 30) GoTo 999
        dYSum = dYSum * dXmid
        dXMid = dXMid * dSpanDelta
        nCnt  = nCnt + 1
        Go To 100
999 Continue
End
```

The text of the object file to perform this test is listed in Appendix AC. The “C” language baseline program is listed in Listing AC-2. The target variable is *dYSum* of the “C” language program.

Listing AC-2- “C" Baseline Source Code of Test Case Three

```
#include "FPU.h"
int main()
{ /* main */
    IEEE754Real8_struct Y;
    double dYSum, dXMid, dXMid0;
    double dX0, dSpan, dSpanIncs, dSpanDelta;
    int n;
    char *pszBits;
    /**/
    dX0 = 0.;
    dSpan = 1.;
    dSpanIncs = 199.;
    dSpanDelta = dSpan /dSpanIncs;
    dXMid0 = dX0 + dSpanDelta / 2.;
    dXMid = dXMid0;
    dYSum = 1.;
    for ( n = 0; n< 30;)
    {
        dYSum = dYSum * dXMid;
        dXMid = dXMid * dSpanDelta;
        Y.dVal = dYSum;
        printf("\nCalculation index: %i\n", ++n);
        pszBits = procIEE754DblToBin(Y.dVal);
        printf(" dCProg: %25.20g, Binary: %s\n", Y.dVal, pszBits);
        free(pszBits);
    }
    return 0;
    /* Number of Store Operations: 1 */
} /* main */
```

After fifteen iterations, the binary representation became excessively long for reporting. Also, the baseline program generated a zero value after the iteration fifteen. Consequently, only results for the first fifteen iterations are shown in the Table AC-1.

Table AC-1- Results of Test Case Three

```
Iteration Number: 1
          dReg                          dRes                          dFPU
 0.002512562814070351796   0.002512562814070351796   0.002512562814070351796
dReg: +0.0000000010100100101010011100111100011101100101101000001100111
dRes: +0.0000000010100100101010011100111100011101100101101000001100111
dFPU: +0.0000000010100100101010011100111100011101100101101000001100111

Iteration Number: 2
          dReg                          dRes                          dFPU
 3.17234768575332889e-08   3.17234768575332889e-08  3.172347685753329552e-08
dReg: +0.(24 "0"s)10001000010000000101010011101000110111110010101000010
dRes: +0.(24 "0"s)10001000010000000101010011101000110111110010101000010
dFPU: +0.(24 "0"s)10001000010000000101010011101000110111110010101000011

Iteration Number: 3
          dReg                          dRes                          dFPU
2.012757967860900278e-15  2.012757967860900672e-15  2.012757967860900672e-15
dReg: +0.(48 "0"s)10010001000010001101010011000011111110110110101111011
dRes: +0.(48 "0"s)10010001000010001101010011000011111110110110101111100
dFPU: +0.(48 "0"s)10010001000010001101010011000011111110110110101111100

Iteration Number: 4
          dReg                          dRes                          dFPU
6.417254353090552741e-25  6.417254353090555496e-25  6.417254353090555496e-25
dReg: +0.(90 "0"s)11000110100110101011101000100001110011000000111011101
dRes: +0.(90 "0"s)11000110100110101011101000100001110011000000111100000
dFPU: +0.(90 "0"s)11000110100110101011101000100001110011000000111100000

Iteration Number: 5
          dReg                          dRes                          dFPU
1.0281438345454555286e-36 1.0281438345454561968e-36 1.0281438345454560298e-36
dReg: +0.(119 "0"s)10101110111011011111101011010111001111111110110110011
dRes: +0.(119 "0"s)10101110111011011111101011010111001111111110110110111
dFPU: +0.(119 "0"s)10101110111011011111101011010111001111111110110110110

Iteration Number: 6
          dReg                          dRes                          dFPU
8.277618780353700364e-51  8.277618780353705111e-51  8.277618780353705111e-51
dReg: +0.(166 "0"s)11000110001101011010011111001101100000010111100111100
dRes: +0.(166 "0"s)11000110001101011010011111001101100000010111101000000
dFPU: +0.(166 "0"s)11000110001101011010011111001101100000010111101000000

Iteration Number: 7
          dReg                          dRes                          dFPU
 3.34891319699938547e-67  3.348913196999386788e-67  3.348913196999387446e-67
dReg: +0.(220 "0"s)10010000011101010101101111101100011101111111011101001
dRes: +0.(220 "0"s)10010000011101010101101111101100011101111111011101011
dFPU: +0.(220 "0"s)10010000011101010101101111101100011101111111011101100

Iteration Number: 8
          dReg                          dRes                          dFPU
 6.80846617099066642e-86  6.808466170990670707e-86  6.808466170990672136e-86
dReg: +0.(282 "0"s)10000111011100001011100111110001000101011010111110001
dRes: +0.(282 "0"s)10000111011100001011100111110001000101011010111110100
dFPU: +0.(282 "0"s)10000111011100001011100111110001000101011010111110101

Iteration Number: 9
```

```
          dReg                      dRes                      dFPU
6.955712140446614388e-107 6.955712140446619228e-107 6.955712140446620438e-107
dReg: +0.(352 "0"s)10100011010110111001110011010000100101010000011100000
dRes: +0.(352 "0"s)10100011010110111001110011010000100101010000011100100
dFPU: +0.(352 "0"s)10100011010110111001110011010000100101010000011100101

Iteration Number: 10
          dReg                      dRes                      dFPU
3.57092592097006905e-130  3.570925920970071452e-130 3.570925920970072254e-130
dReg: +0.(430 "0"s)1111101011101110010111001010111100100110110010011010
dRes: +0.(430 "0"s)1111101011101110010111001010111100100110110010100000
dFPU: +0.(430 "0"s)1111101011101110010111001010111100100110110010100010

Iteration Number: 11
          dReg                      dRes                      dFPU
9.212277351119226108e-156 9.212277351119233354e-156 9.212277351119234388e-156
dReg: +0.(515 "0"s)11111100111101100011000010101001011000010111001111011
dRes: +0.(515 "0"s)11111100111101100011000010101001011000010111010000010
dFPU: +0.(515 "0"s)11111100111101100011000010101001011000010111010000011

Iteration Number: 12
          dReg                      dRes                      dFPU
1.194263315236431116e-183 1.194263315236431952e-183 1.19426331523643237e-183
dReg: +0.(607 "0"s)10100010011000101011010010000000110010010001000100010
dRes: +0.(607 "0"s)10100010011000101011010010000000110010010001000100110
dFPU: +0.(607 "0"s)10100010011000101011010010000000110010010001000101000

Iteration Number: 13
          dReg                      dRes                      dFPU
7.780009066252295885e-214 7.780009066252300832e-214 7.780009066252302481e-214
dReg: +0.(707 "0"s)10000110000110010111010110101110010010000011001101000
dRes: +0.(707 "0"s)10000110000110010111010110101110010010000011001101011
dFPU: +0.(707 "0"s)10000110000110010111010110101110010010000011001101100

Iteration Number: 14
          dReg                      dRes                      dFPU
2.546871523969335746e-246 2.546871523969336762e-246 2.546871523969337778e-246
dReg: +0.(815 "0"s)10001110011101011100110111101111011000000001101010111
dRes: +0.(815 "0"s)10001110011101011100110111101111011000000001101011001
dFPU: +0.(815 "0"s)10001110011101011100110111101111011000000001101011011

Iteration Number: 15
          dReg                      dRes                      dFPU
4.189680407597540046e-281 4.189680407597540046e-281 4.189680407597543104e-281
dReg: +0.(931 "0"s)1000010101100010000000100111110010110110001101001000
dRes: +0.(931 "0"s)1000010101100010000000100111110010110110001101001000
dFPU: +0.(931 "0"s)1000010101100010000000100111110010110110001101001101
```

A review of the results in Table AC-1 indicates that the calculation history results are at the most three bits larger than the regular results. On iteration two, its result is one least significant bit greater than the regular value. This may be due to the round-off truncation on the computed values creating iteration two. By iteration six, the difference has gone to three least significant bits, which is where it remains for the remaining test calculations. Since calculation history results are computed in one continuous floating-point calculation, calculation history results would be the more reliable. For example, the

FPU code to compute the result of iteration seven is shown in Listing AC-3. It shows only one floating-point load operation and only one floating-point store operation. All other operations are an action from a value in program memory onto a much wider value at the top of the floating-point stack. This lack of intermediate truncation into program memory implies greater precision than the regular approach.

Listing AC-3- FPU Code for Iteration Seven

```
int proc_15() /* Iteration 7 of dYSum */
{ /* proc_15 */
    IEEE754Real8_struct d0 = {0xe3b2d067l, 0x3f649539l}; /*
0.002512562814070351796 */
    IEEE754Real8_struct d1 = {0xe3b2d067l, 0x3f749539l}; /*
0.005025125628140703592 */
    IEEE754Real8_struct dRes0;
    char *pszBits;
    { /* add to Area so far */
    asm
        { /* Do FPU stuff */
            fld  d0; /*   0.002512562814070351796 */
            fmul  d0; /*   0.002512562814070351796 */
            fmul d0; /*   0.002512562814070351796 */
            fmul d0; /*   0.002512562814070351796 */
            fmul d0; /*   0.002512562814070351796 */
            fmul d0; /*   0.002512562814070351796 */
            fmul d0; /*   0.002512562814070351796 */
            fmul d1; /*   0.005025125628140703592 */
            fmul d1; /*   0.005025125628140703592 */
            fmul d1; /*   0.005025125628140703592 */
            fmul d1; /*   0.005025125628140703592 */
            fmul d1; /*   0.005025125628140703592 */
            fmul d1; /*   0.005025125628140703592 */
            fmul d1; /*   0.005025125628140703592 */
            fmul d1; /*   0.005025125628140703592 */
            fmul d1; /*   0.005025125628140703592 */
            fmul d1; /*   0.005025125628140703592 */
            fmul d1; /*   0.005025125628140703592 */
            fmul d1; /*   0.005025125628140703592 */
            fmul d1; /*   0.005025125628140703592 */
            fmul d1; /*   0.005025125628140703592 */
            fmul d1; /*   0.005025125628140703592 */
            fmul d1; /*   0.005025125628140703592 */
            fmul d1; /*   0.005025125628140703592 */
            fmul d1; /*   0.005025125628140703592 */
            fmul d1; /*   0.005025125628140703592 */
            fmul d1; /*   0.005025125628140703592 */
            fmul d1; /*   0.005025125628140703592 */
            fstp dRes0; /* Result  */
        } /* Do FPU stuff */
    } /* add to Area so far */
    printf("\nCalculation index: 15\n");
    pszBits = procIEE754DblToBin(dRes0.dVal);
    printf("   dFPU: %25.20g, Binary: %s\n", dRes0.dVal, pszBits);
    free(pszBits);
    return 0;
    /* Number of Store Operations: 1 */
} /* proc_15 */
```

Test Case Four

Test Case Four addressed the issue of adding small numbers to large numbers. There is a maximum number of 53 significant (one "hidden" and 52 storage) bits in the stored double precision value. If the highest order bit has a value of one, the least significant bit can not have a value less than 2^{-52} which is a decimal value of 2.220e-16. Repetitive adding values less than 2.2e-16 to a value of one tests how accurately each approach computes the value of a summation calculation.

Three experiments were tried. Each experiment added a fixed value addend *A* to an initial sum of one. These began with a value of 1.e-18 and increased successively by a multiple of ten in the subsequent experiments. One hundred iterations were performed for each experiment. The CHFort source code for the Experiment One using 1.e-18 is shown in Listing AD-1. The text of the input file to the CHProcEng virtual machine is shown in Appendix AD-1. The only changes in the listings for the other two experiments are in the value of the constant addend *A*. The "C" language baseline code for Experiment One is shown in Listing AD-2.

Listing AD-1- CHFort Source Code of Experiment One

```
Double Precision, History ::  Y
A = 1e-18
Y = 1
N = 0
100 Continue
If (N .ge. 100) GoTo 900
    Y = Y + A
    N = N + 1
    Go To 100
900 Continue
End
```

Listing AD-2- “C” Language Baseline Program for Experiment One

```
#include "FPU.h"
int main()
{ /* main */
    IEEE754Real8_struct Y;
    double A;
    int n;
    char *pszBits;
    /**/
    A = 1.e-18;
    Y.dVal = 1.;
    for ( n = 0; n < 100;)
    {
        Y.dVal = Y.dVal + A;
        printf("\nCalculation index: %i\n", ++n);
        pszBits = procIEE754DblToBin(Y.dVal);
        printf(" dCProg: %25.20g, Binary: %s\n", Y.dVal, pszBits);
        free(pszBits);
    }
    return 0;
    /* Number of Store Operations: 1 */
} /* main */
```

Results of the first and last iterations of Experiment One using the addend value 1.e-18 are shown in Table AD-1. The omitted results have identical values. These results show no effect when double precision floating-point summation is performed regularly by the “C” language program or using calculation histories. This result is reasonable. Using round-to-nearest truncation, at least 111 iterations would be required for the first change in the least significant bit. Subsequent changes would require at least 222 iterations.

Table AD-1- Results of Experiment One

```
Iteration Number: 1, dFrac: 1.0e-18
           dReg                       dRes                       dFPU
            1                          1                          1
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000000000
dFPU: +1.0000000000000000000000000000000000000000000000000000

Iteration Number: 100, dFrac: 1.0e-16
           dReg                       dRes                       dFPU
            1                          1                          1
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000000000
dFPU: +1.0000000000000000000000000000000000000000000000000000
```

Results for the other two experiments, the second using an addend *A* value of 1.e-17 and the third using an addend of 1.e-16, are shown in Appendixes AD-2 and AD-3, respectively, and are summarized in Tables AD-2 and AD-3, respectively. Tables AD-2 and AD-3 show the real value of the fractional part for a given iteration, what was computed for the fractional part by each of the three tested methods, and the six least significant bits of the fractional part by each of the three tested methods.

Table AD-2- Results of Experiment Two

```
     Iter-   dFrac         dReg               dRes               dFPU
     ation              Frac Binary        Frac Binary         Frac Binary
     1        1e-17         0 000000           0 000000           0 000000
     2        2e-17         0 000000           0 000000           0 000000
     3        3e-17         0 000000           0 000000           0 000000
     4        4e-17         0 000000           0 000000           0 000000
     5        5e-17         0 000000           0 000000           0 000000
     6        6e-17         0 000000           0 000000           0 000000
     7        7e-17         0 000000           0 000000           0 000000
     8        8e-17         0 000000           0 000000           0 000000
     9        9e-17         0 000000           0 000000           0 000000
    10        1e-16         0 000000           0 000000           0 000000
    11      1.1e-16         0 000000           0 000000           0 000000
    12      1.2e-16         0 000000  2.2204e-16 000001  2.2204e-16 000001
    13      1.3e-16         0 000000  2.2204e-16 000001  2.2204e-16 000001
    14      1.4e-16         0 000000  2.2204e-16 000001  2.2204e-16 000001
    15      1.5e-16         0 000000  2.2204e-16 000001  2.2204e-16 000001
    16      1.6e-16         0 000000  2.2204e-16 000001  2.2204e-16 000001
    17      1.7e-16         0 000000  2.2204e-16 000001  2.2204e-16 000001
    18      1.8e-16         0 000000  2.2204e-16 000001  2.2204e-16 000001
    19      1.9e-16         0 000000  2.2204e-16 000001  2.2204e-16 000001
    20        2e-16         0 000000  2.2204e-16 000001  2.2204e-16 000001
    21      2.1e-16         0 000000  2.2204e-16 000001  2.2204e-16 000001
    22      2.2e-16         0 000000  2.2204e-16 000001  2.2204e-16 000001
    23      2.3e-16         0 000000  2.2204e-16 000001  2.2204e-16 000001
    24      2.4e-16         0 000000  2.2204e-16 000001  2.2204e-16 000001
    25      2.5e-16         0 000000  2.2204e-16 000001  2.2204e-16 000001
    26      2.6e-16         0 000000  2.2204e-16 000001  2.2204e-16 000001
    27      2.7e-16         0 000000  2.2204e-16 000001  2.2204e-16 000001
    28      2.8e-16         0 000000  2.2204e-16 000001  2.2204e-16 000001
    29      2.9e-16         0 000000  2.2204e-16 000001  2.2204e-16 000001
    30        3e-16         0 000000  2.2204e-16 000001  2.2204e-16 000001
    31      3.1e-16         0 000000  2.2204e-16 000001  2.2204e-16 000001
    32      3.2e-16         0 000000  2.2204e-16 000001  2.2204e-16 000001
    33      3.3e-16         0 000000  2.2204e-16 000001  2.2204e-16 000001
    34      3.4e-16         0 000000  4.4409e-16 000010  4.4409e-16 000010
    35      3.5e-16         0 000000  4.4409e-16 000010  4.4409e-16 000010
    36      3.6e-16         0 000000  4.4409e-16 000010  4.4409e-16 000010
    37      3.7e-16         0 000000  4.4409e-16 000010  4.4409e-16 000010
    38      3.8e-16         0 000000  4.4409e-16 000010  4.4409e-16 000010
    39      3.9e-16         0 000000  4.4409e-16 000010  4.4409e-16 000010
    40        4e-16         0 000000  4.4409e-16 000010  4.4409e-16 000010
    41      4.1e-16         0 000000  4.4409e-16 000010  4.4409e-16 000010
    42      4.2e-16         0 000000  4.4409e-16 000010  4.4409e-16 000010
    43      4.3e-16         0 000000  4.4409e-16 000010  4.4409e-16 000010
    44      4.4e-16         0 000000  4.4409e-16 000010  4.4409e-16 000010
    45      4.5e-16         0 000000  4.4409e-16 000010  4.4409e-16 000010
    46      4.6e-16         0 000000  4.4409e-16 000010  4.4409e-16 000010
    47      4.7e-16         0 000000  4.4409e-16 000010  4.4409e-16 000010
    48     4.8e-16         0 000000  4.4409e-16 000010  4.4409e-16 000010
    49     4.9e-16         0 000000  4.4409e-16 000010  4.4409e-16 000010
    50       5e-16         0 000000  4.4409e-16 000010  4.4409e-16 000010
    51     5.1e-16         0 000000  4.4409e-16 000010  4.4409e-16 000010
    52     5.2e-16         0 000000  4.4409e-16 000010  4.4409e-16 000010
    53     5.3e-16         0 000000  4.4409e-16 000010  4.4409e-16 000010
    54     5.4e-16         0 000000  4.4409e-16 000010  4.4409e-16 000010
    55     5.5e-16         0 000000  4.4409e-16 000010  4.4409e-16 000010
    56     5.6e-16         0 000000  6.6613e-16 000011  6.6613e-16 000011
    57     5.7e-16         0 000000  6.6613e-16 000011  6.6613e-16 000011
    58     5.8e-16         0 000000  6.6613e-16 000011  6.6613e-16 000011
    59     5.9e-16         0 000000  6.6613e-16 000011  6.6613e-16 000011
    60       6e-16         0 000000  6.6613e-16 000011  6.6613e-16 000011
    61     6.1e-16         0 000000  6.6613e-16 000011  6.6613e-16 000011
    62     6.2e-16         0 000000  6.6613e-16 000011  6.6613e-16 000011
    63     6.3e-16         0 000000  6.6613e-16 000011  6.6613e-16 000011
    64     6.4e-16         0 000000  6.6613e-16 000011  6.6613e-16 000011
    65     6.5e-16         0 000000  6.6613e-16 000011  6.6613e-16 000011
    66     6.6e-16         0 000000  6.6613e-16 000011  6.6613e-16 000011
    67     6.7e-16         0 000000  6.6613e-16 000011  6.6613e-16 000011
    68     6.8e-16         0 000000  6.6613e-16 000011  6.6613e-16 000011
    69     6.9e-16         0 000000  6.6613e-16 000011  6.6613e-16 000011
    70       7e-16         0 000000  6.6613e-16 000011  6.6613e-16 000011
    71     7.1e-16         0 000000  6.6613e-16 000011  6.6613e-16 000011
    72     7.2e-16         0 000000  6.6613e-16 000011  6.6613e-16 000011
    73     7.3e-16         0 000000  6.6613e-16 000011  6.6613e-16 000011
    74     7.4e-16         0 000000  6.6613e-16 000011  6.6613e-16 000011
    75     7.5e-16         0 000000  6.6613e-16 000011  6.6613e-16 000011
    76     7.6e-16         0 000000  6.6613e-16 000011  6.6613e-16 000011
    77     7.7e-16         0 000000  6.6613e-16 000011  6.6613e-16 000011
    78     7.8e-16         0 000000  8.8818e-16 000100  8.8818e-16 000100
    79     7.9e-16         0 000000  8.8818e-16 000100  8.8818e-16 000100
    80       8e-16         0 000000  8.8818e-16 000100  8.8818e-16 000100
    81     8.1e-16         0 000000  8.8818e-16 000100  8.8818e-16 000100
    82     8.2e-16         0 000000  8.8818e-16 000100  8.8818e-16 000100
    83     8.3e-16         0 000000  8.8818e-16 000100  8.8818e-16 000100
    84     8.4e-16         0 000000  8.8818e-16 000100  8.8818e-16 000100
    85     8.5e-16         0 000000  8.8818e-16 000100  8.8818e-16 000100
    86     8.6e-16         0 000000  8.8818e-16 000100  8.8818e-16 000100
    87     8.7e-16         0 000000  8.8818e-16 000100  8.8818e-16 000100
    88     8.8e-16         0 000000  8.8818e-16 000100  8.8818e-16 000100
    89     8.9e-16         0 000000  8.8818e-16 000100  8.8818e-16 000100
    90       9e-16         0 000000  8.8818e-16 000100  8.8818e-16 000100
    91     9.1e-16         0 000000  8.8818e-16 000100  8.8818e-16 000100
    92     9.2e-16         0 000000  8.8818e-16 000100  8.8818e-16 000100
    93     9.3e-16         0 000000  8.8818e-16 000100  8.8818e-16 000100
    94     9.4e-16         0 000000  8.8818e-16 000100  8.8818e-16 000100
    95     9.5e-16         0 000000  8.8818e-16 000100  8.8818e-16 000100
    96     9.6e-16         0 000000  8.8818e-16 000100  8.8818e-16 000100
    97     9.7e-16         0 000000  8.8818e-16 000100  8.8818e-16 000100
    98     9.8e-16         0 000000  8.8818e-16 000100  8.8818e-16 000100
    99     9.9e-16         0 000000  8.8818e-16 000100  8.8818e-16 000100
   100       1e-15         0 000000  1.1102e-15 000101  1.1102e-15 000101
```

Table AD-3- Results of Experiment Three

Iter-	dFrac	dReg		dRes		dFPU	
ation		Frac	Binary	Frac	Binary	Frac	Binary
1	1e-16	0	000000	0	000000	0	000000
2	2e-16	0	000000	0	000000	2.2204e-16	000001
3	3e-16	0	000000	2.2204e-16	000001	2.2204e-16	000001
4	4e-16	0	000000	4.4409e-16	000010	4.4409e-16	000010
5	5e-16	0	000000	4.4409e-16	000010	4.4409e-16	000010
6	6e-16	0	000000	6.6613e-16	000011	6.6613e-16	000011
7	7e-16	0	000000	6.6613e-16	000011	6.6613e-16	000011
8	8e-16	0	000000	8.8818e-16	000100	8.8818e-16	000100
9	9e-16	0	000000	8.8818e-16	000100	8.8818e-16	000100
10	1e-15	0	000000	1.1102e-15	000101	1.1102e-15	000101
11	1.1e-15	0	000000	1.1102e-15	000101	1.1102e-15	000101
12	1.2e-15	0	000000	1.1102e-15	000101	1.1102e-15	000101
13	1.3e-15	0	000000	1.3323e-15	000110	1.3323e-15	000110
14	1.4e-15	0	000000	1.3323e-15	000110	1.3323e-15	000110
15	1.5e-15	0	000000	1.5543e-15	000111	1.5543e-15	000111
16	1.6e-15	0	000000	1.5543e-15	000111	1.5543e-15	000111
17	1.7e-15	0	000000	1.7764e-15	001000	1.7764e-15	001000
18	1.8e-15	0	000000	1.7764e-15	001000	1.7764e-15	001000
19	1.9e-15	0	000000	1.9984e-15	001001	1.9984e-15	001001
20	2e-15	0	000000	1.9984e-15	001001	1.9984e-15	001001
21	2.1e-15	0	000000	1.9984e-15	001001	1.9984e-15	001001
22	2.2e-15	0	000000	2.2204e-15	001010	2.2204e-15	001010
23	2.3e-15	0	000000	2.2204e-15	001010	2.2204e-15	001010
24	2.4e-15	0	000000	2.4425e-15	001011	2.4425e-15	001011
25	2.5e-15	0	000000	2.4425e-15	001011	2.4425e-15	001011
26	2.6e-15	0	000000	2.6645e-15	001100	2.6645e-15	001100
27	2.7e-15	0	000000	2.6645e-15	001100	2.6645e-15	001100
28	2.8e-15	0	000000	2.8866e-15	001101	2.8866e-15	001101
29	2.9e-15	0	000000	2.8866e-15	001101	2.8866e-15	001101
30	3e-15	0	000000	3.1086e-15	001110	3.1086e-15	001110
31	3.1e-15	0	000000	3.1086e-15	001110	3.1086e-15	001110
32	3.2e-15	0	000000	3.1086e-15	001110	3.1086e-15	001110
33	3.3e-15	0	000000	3.3307e-15	001111	3.3307e-15	001111
34	3.4e-15	0	000000	3.3307e-15	001111	3.3307e-15	001111
35	3.5e-15	0	000000	3.5527e-15	010000	3.5527e-15	010000
36	3.6e-15	0	000000	3.5527e-15	010000	3.5527e-15	010000
37	3.7e-15	0	000000	3.7748e-15	010001	3.7748e-15	010001
38	3.8e-15	0	000000	3.7748e-15	010001	3.7748e-15	010001
39	3.9e-15	0	000000	3.9968e-15	010010	3.9968e-15	010010
40	4e-15	0	000000	3.9968e-15	010010	3.9968e-15	010010
41	4.1e-15	0	000000	3.9968e-15	010010	3.9968e-15	010010
42	4.2e-15	0	000000	4.2188e-15	010011	4.2188e-15	010011
43	4.3e-15	0	000000	4.2188e-15	010011	4.2188e-15	010011
44	4.4e-15	0	000000	4.4409e-15	010100	4.4409e-15	010100
45	4.5e-15	0	000000	4.4409e-15	010100	4.4409e-15	010100
46	4.6e-15	0	000000	4.6629e-15	010101	4.6629e-15	010101
47	4.7e-15	0	000000	4.6629e-15	010101	4.6629e-15	010101
48	4.8e-15	0	000000	4.885e-15	010110	4.885e-15	010110
49	4.9e-15	0	000000	4.885e-15	010110	4.885e-15	010110
50	5e-15	0	000000	5.107e-15	010111	5.107e-15	010111
51	5.1e-15	0	000000	5.107e-15	010111	5.107e-15	010111
52	5.2e-15	0	000000	5.107e-15	010111	5.107e-15	010111
53	5.3e-15	0	000000	5.3291e-15	011000	5.3291e-15	011000
54	5.4e-15	0	000000	5.3291e-15	011000	5.3291e-15	011000
55	5.5e-15	0	000000	5.5511e-15	011001	5.5511e-15	011001
56	5.6e-15	0	000000	5.5511e-15	011001	5.5511e-15	011001
57	5.7e-15	0	000000	5.7732e-15	011010	5.7732e-15	011010
58	5.8e-15	0	000000	5.7732e-15	011010	5.7732e-15	011010
59	5.9e-15	0	000000	5.9952e-15	011011	5.9952e-15	011011
60	6e-15	0	000000	5.9952e-15	011011	5.9952e-15	011011
61	6.1e-15	0	000000	5.9952e-15	011011	5.9952e-15	011011
62	6.2e-15	0	000000	6.2172e-15	011100	6.2172e-15	011100
63	6.3e-15	0	000000	6.2172e-15	011100	6.2172e-15	011100
64	6.4e-15	0	000000	6.4393e-15	011101	6.4393e-15	011101
65	6.5e-15	0	000000	6.4393e-15	011101	6.4393e-15	011101
66	6.6e-15	0	000000	6.6613e-15	011110	6.6613e-15	011110

```
   67     6.7e-15         0 000000  6.6613e-15 011110  6.6613e-15 011110
   68     6.8e-15         0 000000  6.8834e-15 011111  6.8834e-15 011111
   69     6.9e-15         0 000000  6.8834e-15 011111  6.8834e-15 011111
   70       7e-15         0 000000  7.1054e-15 100000  7.1054e-15 100000
   71     7.1e-15         0 000000  7.1054e-15 100000  7.1054e-15 100000
   72     7.2e-15         0 000000  7.1054e-15 100000  7.1054e-15 100000
   73     7.3e-15         0 000000  7.3275e-15 100001  7.3275e-15 100001
   74     7.4e-15         0 000000  7.3275e-15 100001  7.3275e-15 100001
   75     7.5e-15         0 000000  7.5495e-15 100010  7.5495e-15 100010
   76     7.6e-15         0 000000  7.5495e-15 100010  7.5495e-15 100010
   77     7.7e-15         0 000000  7.7716e-15 100011  7.7716e-15 100011
   78     7.8e-15         0 000000  7.7716e-15 100011  7.7716e-15 100011
   79     7.9e-15         0 000000  7.9936e-15 100100  7.9936e-15 100100
   80       8e-15         0 000000  7.9936e-15 100100  7.9936e-15 100100
   81     8.1e-15         0 000000  7.9936e-15 100100  7.9936e-15 100100
   82     8.2e-15         0 000000  8.2157e-15 100101  8.2157e-15 100101
   83     8.3e-15         0 000000  8.2157e-15 100101  8.2157e-15 100101
   84     8.4e-15         0 000000  8.4377e-15 100110  8.4377e-15 100110
   85     8.5e-15         0 000000  8.4377e-15 100110  8.4377e-15 100110
   86     8.6e-15         0 000000  8.6597e-15 100111  8.6597e-15 100111
   87     8.7e-15         0 000000  8.6597e-15 100111  8.6597e-15 100111
   88     8.8e-15         0 000000  8.8818e-15 101000  8.8818e-15 101000
   89     8.9e-15         0 000000  8.8818e-15 101000  8.8818e-15 101000
   90       9e-15         0 000000  9.1038e-15 101001  9.1038e-15 101001
   91     9.1e-15         0 000000  9.1038e-15 101001  9.1038e-15 101001
   92     9.2e-15         0 000000  9.1038e-15 101001  9.1038e-15 101001
   93     9.3e-15         0 000000  9.3259e-15 101010  9.3259e-15 101010
   94     9.4e-15         0 000000  9.3259e-15 101010  9.3259e-15 101010
   95     9.5e-15         0 000000  9.5479e-15 101011  9.5479e-15 101011
   96     9.6e-15         0 000000  9.5479e-15 101011  9.5479e-15 101011
   97     9.7e-15         0 000000    9.77e-15 101100    9.77e-15 101100
   98     9.8e-15         0 000000    9.77e-15 101100    9.77e-15 101100
   99     9.9e-15         0 000000   9.992e-15 101101   9.992e-15 101101
  100       1e-14         0 000000   9.992e-15 101101   9.992e-15 101101
```

The least significant bit changes in Experiment Two, Table AD-2, at the 12th iteration rather than the 22nd. This is a result of the floating-point unit’s round-to-nearest truncation philosophy. Subsequently, they increment on iterations 34, 56, and 78. These iterations are spaced exactly 22 apart; since the addend in this experiment is ten times the magnitude of that of Experiment One, the iteration spacing is one-tenth that of Experiment One. It must be noted that adding 1.e-17 only affects a change in the least significant bit after it has been added twenty-two times. Adding a value of 12.e-17 to one has the same effect as adding a value of 23.e-17 to a value of one.

Similarly, in Experiment Three, Table AD-3, the iteration spacing is one-hundredth that of Experiment One since the addend is one hundred times the magnitude of the first experiment’s addend. The binary results should show an initial change after one (the integer portion of 2.22 / 2) iteration. Subsequently, the binary results should show a change after two (the integer portion of 2.22) iterations. Since the iteration spacing (2.22) contains a fraction, after every five changes in the binary results, a result may be additionally repeated. The results shown in Table AD-3 show exactly this pattern. Since 100 iterations were performed, there should be 100 / 2.22 changes; this change count has an integer value of 45. The value of the least significant bits of the one-hundredth iteration is exactly 45.

With the exception of the second iteration in Experiment Three, the *dRes* value of each iteration, computed using calculation histories computed by the CHProcEng program, matched the value computed by the floating-point unit in all experiments. In contrast, the “C” language program was totally defective. It computed for every iteration a value of zero in all experiments. Because the most significant bit of the double-precision addend was less than one-half the least significant bit of the double-precision sum, the value of the sum computed by the “C” language program remained unchanged.

Test Case Five

Test Case Five built on the results of Test Case Four. While Test Case Four tested the addition of a fixed addend to a summation, Test Case Five tested the case of adding an increasing addend to a summation. The addend began as 1.e-17 which is one order of magnitude smaller than any value that can be added to a value of one and result in a different value. Since the addend is maintained as a separate value, any change in its stored value would not be affected by the stored value of the summation. Because the addend is so small compared to the sum, the result of every addition to the sum is subject to truncation to storage round off.

Two experiments were performed with different methods of modifying the value of the addend. Experiment One added 1.e-17 to it after each iteration. For iteration *n*, the addend would have the value (*n * n + n*) / *2 * 1.e-17*. This is the predicted value and is shown as variable *dFrac* in Table AE-1. The full results of Experiment One are shown in Appendix AE-1. Listing AE-1 shows the CHFort input file used for Experiment One. Appendix AE-2 shows the text of the object input file to CHProcEng that was generated. Listing AE-2 shows the source code of the "C" language baseline program.

Listing AE-1- CHFort Source Code of Test Case Five, Experiment One

```
Double Precision, History ::  Y
A = 1e-17
Y = 1
N = 0
100 Continue
If (N .ge. 100) GoTo 900
    Y = Y + A
    N = N + 1
    A = A + 1e-17
    Go To 100
900 Continue
End
```

Listing AE-2- “C” Language Baseline Program for Test Case Five, Experiment One

```
#include "FPU.h"
int main()
{ /* main */
    IEEE754Real8_struct Y;
    double A;
    double dPrevY, dASum;
    int n;
    char *pszBits;
    /**/
    A = 1.e-17;
    Y.dVal = 1.;
    dASum = 0;
    for ( n = 0; n < 100;)
    {
        dPrevY = Y.dVal;
        Y.dVal = Y.dVal + A;
        printf("\nCalculation index: %i\n", ++n);
        pszBits = procIEE754DblToBin(Y.dVal);
        dASum += A;
        printf(" dCProg: %25.20g, Binary: %s\n", Y.dVal, pszBits);
        printf("  Prev Y: %25.20g, A: %25.20g, ASum: %25.20g\n", dPrevY,
A, dASum);
        free(pszBits);
        A += 1.e-17;
    }
    return 0;
    /* Number of Store Operations: 1 */
} /* main */
```

Table AE-1- Results for Experiment One

Iter-	dFrac	dReg		dRes		dFPU	
ation		Frac	Binary	Frac	Binary	Frac	Binary
1	2e-17	0	00000000	0	00000000	0	00000000
2	4e-17	0	00000000	0	00000000	0	00000000
3	7e-17	0	00000000	0	00000000	0	00000000
4	1.1e-16	0	00000000	0	00000000	0	00000000
5	1.6e-16	0	00000000	2.2204e-16	00000001	2.2204e-16	00000001
6	2.2e-16	0	00000000	2.2204e-16	00000001	2.2204e-16	00000001
7	2.9e-16	0	00000000	2.2204e-16	00000001	2.2204e-16	00000001
8	3.7e-16	0	00000000	4.4409e-16	00000010	4.4409e-16	00000010
9	4.6e-16	0	00000000	4.4409e-16	00000010	4.4409e-16	00000010
10	5.6e-16	0	00000000	4.4409e-16	00000010	4.4409e-16	00000010
11	6.7e-16	0	00000000	6.6613e-16	00000011	6.6613e-16	00000011
12	7.9e-16	2.2204e-16	00000001	8.8818e-16	00000100	8.8818e-16	00000100
13	9.2e-16	4.4409e-16	00000010	8.8818e-16	00000100	8.8818e-16	00000100
14	1.06e-15	6.6613e-16	00000011	1.1102e-15	00000101	1.1102e-15	00000101
15	1.21e-15	8.8818e-16	00000100	1.1102e-15	00000101	1.1102e-15	00000101
16	1.37e-15	1.1102e-15	00000101	1.3323e-15	00000110	1.3323e-15	00000110
17	1.54e-15	1.3323e-15	00000110	1.5543e-15	00000111	1.5543e-15	00000111
18	1.72e-15	1.5543e-15	00000111	1.7764e-15	00001000	1.7764e-15	00001000
19	1.91e-15	1.7764e-15	00001000	1.9984e-15	00001001	1.9984e-15	00001001
20	2.11e-15	1.9984e-15	00001001	1.9984e-15	00001001	1.9984e-15	00001001
21	2.32e-15	2.2204e-15	00001010	2.2204e-15	00001010	2.2204e-15	00001010
22	2.54e-15	2.4425e-15	00001011	2.4425e-15	00001011	2.4425e-15	00001011
23	2.77e-15	2.6645e-15	00001100	2.6645e-15	00001100	2.6645e-15	00001100
24	3.01e-15	2.8866e-15	00001101	3.1086e-15	00001110	3.1086e-15	00001110
25	3.26e-15	3.1086e-15	00001110	3.3307e-15	00001111	3.3307e-15	00001111
26	3.52e-15	3.3307e-15	00001111	3.5527e-15	00010000	3.5527e-15	00010000
27	3.79e-15	3.5527e-15	00010000	3.7748e-15	00010001	3.7748e-15	00010001
28	4.07e-15	3.7748e-15	00010001	3.9968e-15	00010010	3.9968e-15	00010010
29	4.36e-15	3.9968e-15	00010010	4.4409e-15	00010100	4.4409e-15	00010100
30	4.66e-15	4.2188e-15	00010011	4.6629e-15	00010101	4.6629e-15	00010101
31	4.97e-15	4.4409e-15	00010100	4.885e-15	00010110	4.885e-15	00010110
32	5.29e-15	4.6629e-15	00010101	5.3291e-15	00011000	5.3291e-15	00011000
33	5.62e-15	4.885e-15	00010110	5.5511e-15	00011001	5.5511e-15	00011001
34	5.96e-15	5.3291e-15	00011000	5.9952e-15	00011011	5.9952e-15	00011011
35	6.31e-15	5.7732e-15	00011010	6.2172e-15	00011100	6.2172e-15	00011100
36	6.67e-15	6.2172e-15	00011100	6.6613e-15	00011110	6.6613e-15	00011110

37	7.04e-15	6.6613e-15	00011110	7.1054e-15	00100000	7.1054e-15	00100000
38	7.42e-15	7.1054e-15	00100000	7.3275e-15	00100001	7.3275e-15	00100001
39	7.81e-15	7.5495e-15	00100010	7.7716e-15	00100011	7.7716e-15	00100011
40	8.21e-15	7.9936e-15	00100100	8.2157e-15	00100101	8.2157e-15	00100101
41	8.62e-15	8.4377e-15	00100110	8.6597e-15	00100111	8.6597e-15	00100111
42	9.04e-15	8.8818e-15	00101000	9.1038e-15	00101001	9.1038e-15	00101001
43	9.47e-15	9.3259e-15	00101010	9.5479e-15	00101011	9.5479e-15	00101011
44	9.91e-15	9.77e-15	00101100	9.992e-15	00101101	9.992e-15	00101101
45	1.036e-14	1.0214e-14	00101110	1.0436e-14	00101111	1.0436e-14	00101111
46	1.082e-14	1.0658e-14	00110000	1.088e-14	00110001	1.088e-14	00110001
47	1.129e-14	1.1102e-14	00110010	1.1324e-14	00110011	1.1324e-14	00110011
48	1.177e-14	1.1546e-14	00110100	1.1768e-14	00110101	1.1768e-14	00110101
49	1.226e-14	1.199e-14	00110110	1.2212e-14	00110111	1.2212e-14	00110111
50	1.276e-14	1.2434e-14	00111000	1.2657e-14	00111001	1.2657e-14	00111001
51	1.327e-14	1.2879e-14	00111010	1.3323e-14	00111100	1.3323e-14	00111100
52	1.379e-14	1.3323e-14	00111100	1.3767e-14	00111110	1.3767e-14	00111110
53	1.432e-14	1.3767e-14	00111110	1.4211e-14	01000000	1.4211e-14	01000000
54	1.486e-14	1.4211e-14	01000000	1.4877e-14	01000011	1.4877e-14	01000011
55	1.541e-14	1.4655e-14	01000010	1.5321e-14	01000101	1.5321e-14	01000101
56	1.597e-14	1.5321e-14	01000101	1.5987e-14	01001000	1.5987e-14	01001000
57	1.654e-14	1.5987e-14	01001000	1.6431e-14	01001010	1.6431e-14	01001010
58	1.712e-14	1.6653e-14	01001011	1.7097e-14	01001101	1.7097e-14	01001101
59	1.771e-14	1.7319e-14	01001110	1.7764e-14	01010000	1.7764e-14	01010000
60	1.831e-14	1.7986e-14	01010001	1.8208e-14	01010010	1.8208e-14	01010010
61	1.892e-14	1.8652e-14	01010100	1.8874e-14	01010101	1.8874e-14	01010101
62	1.954e-14	1.9318e-14	01010111	1.954e-14	01011000	1.954e-14	01011000
63	2.017e-14	1.9984e-14	01011010	2.0206e-14	01011011	2.0206e-14	01011011
64	2.081e-14	2.065e-14	01011101	2.0872e-14	01011110	2.0872e-14	01011110
65	2.146e-14	2.1316e-14	01100000	2.1538e-14	01100001	2.1538e-14	01100001
66	2.212e-14	2.1982e-14	01100011	2.2204e-14	01100100	2.2204e-14	01100100
67	2.279e-14	2.2649e-14	01100110	2.2871e-14	01100111	2.2871e-14	01100111
68	2.347e-14	2.3315e-14	01101001	2.3537e-14	01101010	2.3537e-14	01101010
69	2.416e-14	2.3981e-14	01101100	2.4203e-14	01101101	2.4203e-14	01101101
70	2.486e-14	2.4647e-14	01101111	2.4869e-14	01110000	2.4869e-14	01110000
71	2.557e-14	2.5313e-14	01110010	2.5535e-14	01110011	2.5535e-14	01110011
72	2.629e-14	2.5979e-14	01110101	2.6201e-14	01110110	2.6201e-14	01110110
73	2.702e-14	2.6645e-14	01111000	2.7089e-14	01111010	2.7089e-14	01111010
74	2.776e-14	2.7311e-14	01111011	2.7756e-14	01111101	2.7756e-14	01111101
75	2.851e-14	2.7978e-14	01111110	2.8422e-14	10000000	2.8422e-14	10000000
76	2.927e-14	2.8644e-14	10000001	2.931e-14	10000100	2.931e-14	10000100
77	3.004e-14	2.931e-14	10000100	2.9976e-14	10000111	2.9976e-14	10000111
78	3.082e-14	3.0198e-14	10001000	3.0864e-14	10001011	3.0864e-14	10001011
79	3.161e-14	3.1086e-14	10001100	3.153e-14	10001110	3.153e-14	10001110
80	3.241e-14	3.1974e-14	10010000	3.2419e-14	10010010	3.2419e-14	10010010
81	3.322e-14	3.2863e-14	10010100	3.3307e-14	10010110	3.3307e-14	10010110
82	3.404e-14	3.3751e-14	10011000	3.3973e-14	10011001	3.3973e-14	10011001
83	3.487e-14	3.4639e-14	10011100	3.4861e-14	10011101	3.4861e-14	10011101
84	3.571e-14	3.5527e-14	10100000	3.5749e-14	10100001	3.5749e-14	10100001
85	3.656e-14	3.6415e-14	10100100	3.6637e-14	10100101	3.6637e-14	10100101
86	3.742e-14	3.7303e-14	10101000	3.7303e-14	10101000	3.7303e-14	10101000
87	3.829e-14	3.8192e-14	10101100	3.8192e-14	10101100	3.8192e-14	10101100
88	3.917e-14	3.908e-14	10110000	3.908e-14	10110000	3.908e-14	10110000
89	4.006e-14	3.9968e-14	10110100	3.9968e-14	10110100	3.9968e-14	10110100
90	4.096e-14	4.0856e-14	10111000	4.0856e-14	10111000	4.0856e-14	10111000
91	4.187e-14	4.1744e-14	10111100	4.1966e-14	10111101	4.1966e-14	10111101
92	4.279e-14	4.2633e-14	11000000	4.2855e-14	11000001	4.2855e-14	11000001
93	4.372e-14	4.3521e-14	11000100	4.3743e-14	11000101	4.3743e-14	11000101
94	4.466e-14	4.4409e-14	11001000	4.4631e-14	11001001	4.4631e-14	11001001
95	4.561e-14	4.5297e-14	11001100	4.5519e-14	11001101	4.5519e-14	11001101
96	4.657e-14	4.6185e-14	11010000	4.6629e-14	11010010	4.6629e-14	11010010
97	4.754e-14	4.7073e-14	11010100	4.7518e-14	11010110	4.7518e-14	11010110
98	4.852e-14	4.7962e-14	11011000	4.8406e-14	11011010	4.8406e-14	11011010
99	4.951e-14	4.885e-14	11011100	4.9516e-14	11011111	4.9516e-14	11011111
100	5.051e-14	4.996e-14	11100001	5.0404e-14	11100011	5.0404e-14	11100011

Experiment Two doubled the value of the addend after each iteration. For iteration *n*, the addend would have the value $(2^n - 1) * 1.e-17$. This is the predicted value shown as variable *dFrac* in Table AE-2. The full results of Experiment Two are shown in Appendix AE-3. Listing AE-3 shows the CHFort input file used for Experiment Two.

Appendix AE-4 shows the text of the object input file to CHProcEng that was generated.

Listing AE-4 shows the source code of the “C” language baseline program.

Listing AE-3 CHFort Source Code of Test Case Five, Experiment Two

```
Double Precision, History ::  Y
A = 1e-17
Y = 1
N = 0
100 Continue
If (N .ge. 100) GoTo 900
    Y = Y + A
    N = N + 1
    A = A + 1e-17
    Go To 100
900 Continue
End
```

Listing AE-4- “C” Language Baseline Program for Test Case Five, Experiment Two

```
#include "FPU.h"
int main()
{ /* main */
    IEEE754Real8_struct Y;
    double A;
    double dPrevY, dASum;
    int n;
    char *pszBits;
    /**/
    A = 1.e-17;
    Y.dVal = 1.;
    dASum = 0;
    for ( n = 0; n < 100;)
    {
        dPrevY = Y.dVal;
        Y.dVal = Y.dVal + A;
        printf("\nCalculation index: %i\n", ++n);
        pszBits = procIEEE754DblToBin(Y.dVal);
        dASum += A;
        printf(" dCProg: %25.20g, Binary: %s\n", Y.dVal, pszBits);
        printf("  Prev Y: %25.20g, A: %25.20g, ASum: %25.20g\n", dPrevY,
A, dASum);
        free(pszBits);
        A += 1.e-17;
    }
    return 0;
    /* Number of Store Operations: 1 */
} /* main */
```

Table AE-2- Selected Results for Experiment Two

```
Iteration Number: 1, dFrac: 1.0000000000000001e-17
         dReg                     dRes                   dFPU
          1                      1                      1
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000000000
dFPU: +1.0000000000000000000000000000000000000000000000000000

Iteration Number: 2, dFrac: 3.0000000000000001e-17
         dReg                     dRes                   dFPU
          1                      1                      1
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000000000
dFPU: +1.0000000000000000000000000000000000000000000000000000

Iteration Number: 3, dFrac: 7.0000000000000003e-17
         dReg                     dRes                   dFPU
          1                      1                      1
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000000000
dFPU: +1.0000000000000000000000000000000000000000000000000000

Iteration Number: 4, dFrac: 1.5000000000000002e-16
         dReg                     dRes                   dFPU
          1             1.000000000000000222   1.000000000000000222
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000000001
dFPU: +1.0000000000000000000000000000000000000000000000000001

Iteration Number: 5, dFrac: 3.1000000000000001e-16
         dReg                     dRes                   dFPU
 1.000000000000000222   1.000000000000000222   1.000000000000000222
dReg: +1.0000000000000000000000000000000000000000000000000001
dRes: +1.0000000000000000000000000000000000000000000000000001
dFPU: +1.0000000000000000000000000000000000000000000000000001

Iteration Number: 6, dFrac: 6.3000000000000008e-16
         dReg                     dRes                   dFPU
1.0000000000000004441  1.0000000000000006661  1.0000000000000006661
dReg: +1.0000000000000000000000000000000000000000000000000010
dRes: +1.0000000000000000000000000000000000000000000000000011
dFPU: +1.0000000000000000000000000000000000000000000000000011

Iteration Number: 96, dFrac: 792281625142.64343
         dReg                     dRes                   dFPU
 792281625143.6433106   792281625143.6433106   792281625143.6434326
dReg: +1011100001110111101010100011001000110111.1010010010110
dRes: +1011100001110111101010100011001000110111.1010010010110
dFPU: +1011100001110111101010100011001000110111.1010010010111

Iteration Number: 97, dFrac: 1584563250285.2869
         dReg                     dRes                   dFPU
 1584563250286.286621   1584563250286.286621   1584563250286.286865
dReg: +10111000011101111010101000110010001101110.010010010110
dRes: +10111000011101111010101000110010001101110.010010010110
dFPU: +10111000011101111010101000110010001101110.010010010111

Iteration Number: 98, dFrac: 3169126500570.5737
         dReg                     dRes                   dFPU
 3169126500571.573242   3169126500571.573242   3169126500571.57373
dReg: +101110000111011110101010001100100011011011.10010010110
dRes: +101110000111011110101010001100100011011011.10010010110
dFPU: +101110000111011110101010001100100011011011.10010010111

Iteration Number: 99, dFrac: 6338253001141.1475
         dReg                     dRes                   dFPU
 6338253001142.146484   6338253001142.146484   6338253001142.147461
dReg: +1011100001110111101010100011001000110110110.0010010110
dRes: +1011100001110111101010100011001000110110110.0010010110
dFPU: +1011100001110111101010100011001000110110110.0010010111

Iteration Number: 100, dFrac: 12676506002282.295
         dReg                     dRes                   dFPU
 12676506002283.29297   12676506002283.29297   12676506002283.29492
dReg: +10111000011101111010101000110010001101101011.010010110
dRes: +10111000011101111010101000110010001101101011.010010110
dFPU: +10111000011101111010101000110010001101101011.010010111
```

Experiment One, Table AE-1, shows that the calculation history approach provides the first valid result at iteration five. This is correct since the value 16/22 is at least equal to one-half (.5). The “C” language program (regular approach) still computed zero until the iteration twelve when its result made its first change. Between iteration five and 12 the regular approach was 100% in error. At iteration five, the least significant bits were correctly computed by the calculation history approach as four (rounded to nearest 79/22.2). The regular approach computed a value of one. Fortunately, this significant discrepancy (75% error!) did not last long. Two iterations later, the difference in the value of least significant bits different changed to one and remained within that difference plus or minus one bit subsequently. The regular approach always computed a value less than the correct value.

Experiment Two, Table AE-2, tested adding a rapidly increasing, but initially small, addend value to a sum that was initially much larger than its initial value. The first change in the least significant bit was correctly detected by the calculation history approach in iteration four. Since round-to-nearest truncation was applied when storing computed results into program memory, this occurred when the value of the addend was at least one-half. This was reached when the addend was 15.e-17. Since one least significant unit bit value equals 22.2e-17, the value 15.e-17 / 22.2e-17 is greater than one-half least significant bit value at iteration four.

The regular approach detected this at iteration five when the value of one was reached when the addend value first became at least one (31 units / 22.2 units per bit). The floating-point values (*dFPU*) of the calculation history approach exactly computed the predicted value with the single exception of iteration nine. The regular approach values (*dReg*) agreed about one-half the time with *dFPU* values. However, when there was a difference, the *dReg* values were one least significant bit value lower than the *dFPU* values.

Both experiments indicate calculation history superiority at calculations with large and small operand values. Calculation histories generate the most precise values when a regular approach would effectively disregard operands in a calculation to generate an incorrect value. This test case also bears out that calculation histories consistently return the precise value based on all operands.

Test Case Six

This test case consisted of Newton's Method [Smith and Minton] to compute the square root of a number. Newton's Method is an iterative process in which the one dependent variable is the result of the previous calculation. An example to compute the square root of nine (9) using the CHFort language is shown in Listing AF-1. The initial value of the variable *X* could be anything other than zero. In this case, it is the same value as that of the target variable *B*.

Listing AF-1- CHFort Implementation for Square Root of 9 by Newton's Method

```
Double Precision, History :: X, Xp
B = 9.
n = 0
X = B
100 Continue
        Xp = (X + B / X) / 2.
        X = XP
        n = n + 1
If (n .le. 10) GoTo 100
```

The reduction tree for this calculation is shown in Figure AF-1. One characteristic of this type of problem is that successive iterations of the reduction tree can be developed by replacing each node containing the independent variable with the previous reduction tree for the calculation. If the first calculation is shown in Figure AF-1, the second calculation is shown in Figure AF-2. Only the nodes with the independent variable *X* have been replaced with the preceding calculation. Each iteration doubles the number of nodes containing the initial value of the independent variable.

Figure AF-1- Reduction Tree for Newton’s Method Square Root Calculation

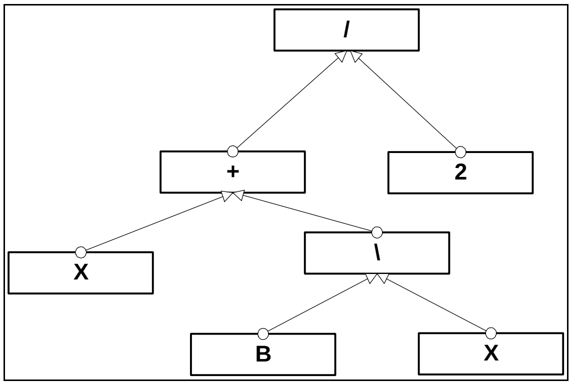
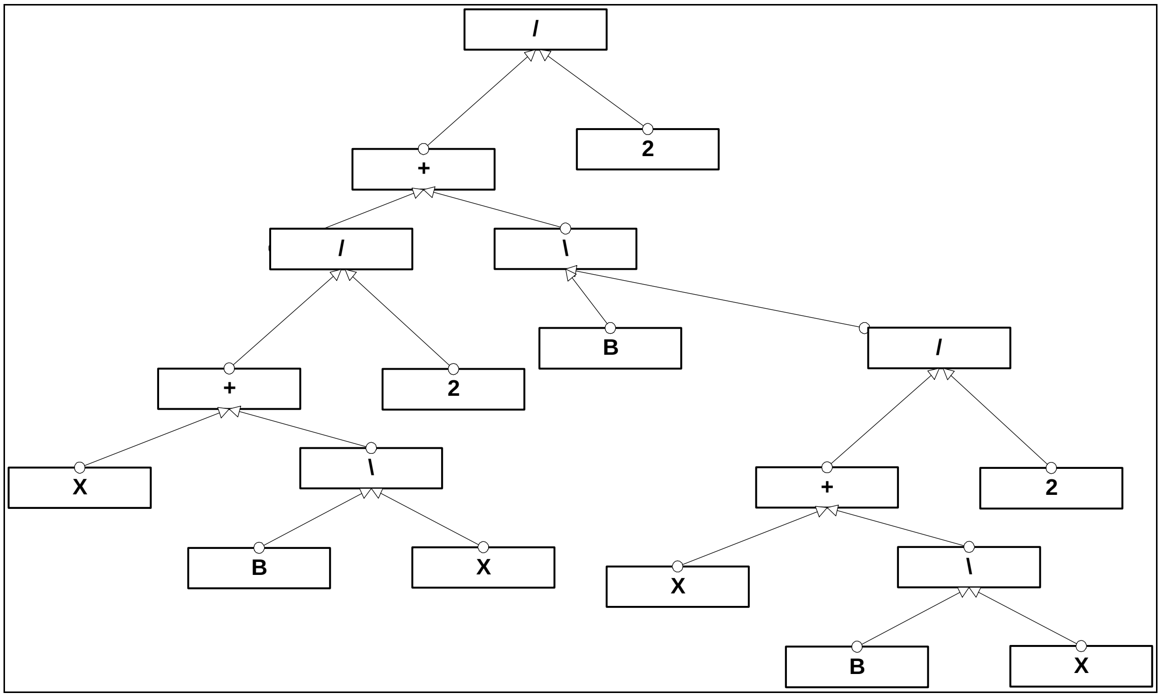
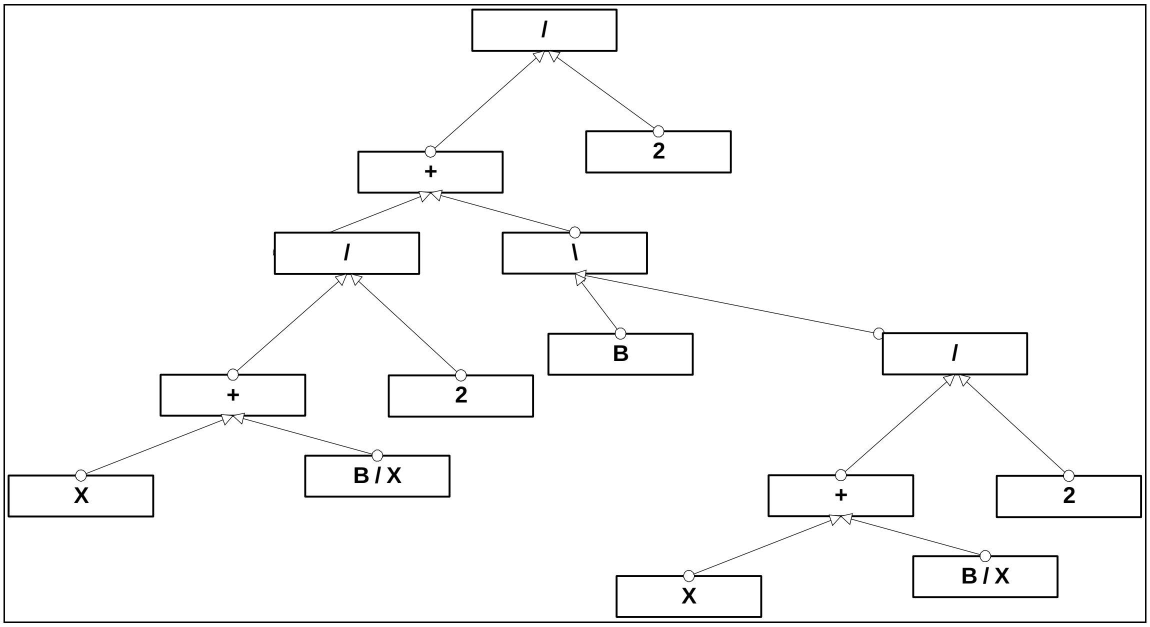


Figure AF-2- Reduction Tree for After Substitution of Computed Variables



It can be seen that after several iterations of computing, the calculation tree will contain a large number of reduction tree elements. Bottom-Up Pruning can be applied to reduce the number of reduction tree elements in the calculation tree. A maximum number of reduction tree elements must be set. If this number of elements is exceeded, the bottom-most elements are removed and their computed value replaces the parent calculation. For example, in Figure AF-2 there are 11 computation elements. If the maximum number were nine, the bottom-most two elements would have to be removed. The result would be the calculation tree shown in Figure AF-3. This saves values of the “pruned” elements in their parent element operands, so the value of their computation is not lost. The saved value is, however, truncated when it is stored in the value of the parent element operand.

Figure AF-3- Reduction Tree Truncated by Bottom-Up Pruning



Three experiments were tried. Experiment One tested the calculation history model computing a simple square root of a value. This value was chosen to be the square of the integer three which was nine. The CHFort source code for this experiment is shown in Listing AF-1. The text of the object code created for this experiment is shown in Appendix AF-1. The text of the “C” baseline program is shown in Listing AF-2.

Listing AF-2- “C" Language Baseline Program Source

```
#include "FPU.h"
int main()
{ /* main */
    IEEE754Real8_struct Y;
    double B, X, XP;
    int n;
    char *pszBits;
    /**/
    B = 9.;
    X = B;
    for (n = 0; n <= 11; n)
    { /* Compute a new value */
        XP = (X + B / X) / 2e0;
        X = XP;
        Y.dVal = XP;
        printf("\nCalculation index: %i\n", ++n);
        pszBits = procIEE754DblToBin(Y.dVal);
        printf(" dCProg: %25.20g, Binary: %s\n", Y.dVal, pszBits);
        free(pszBits);
    } /* Compute a new value */
    return 0;
    /* Number of Store Operations: 1 */
} /* main */
```

The results of Experiment One are shown in Table AF-1. Only the first seven iterations are listed since after the iteration six, the computed value does not change for either method. The number of calculations to arrive at the constant result was small and the values of the operands were close. Consequently, there was little opportunity for the two methods to compute a dissimilar value.

Table AF-1- Results of Experiment One

```
Iteration Number: 1
           dReg                       dRes                       dFPU
            5                          5                          5
dReg: +101.00000000000000000000000000000000000000000000000000
dRes: +101.00000000000000000000000000000000000000000000000000
dFPU: +101.00000000000000000000000000000000000000000000000000

Iteration Number: 2
           dReg                       dRes                       dFPU
   3.399999999999999911       3.399999999999999911       3.399999999999999911
dReg: +11.011001100110011001100110011001100110011001100110011
dRes: +11.011001100110011001100110011001100110011001100110011
dFPU: +11.011001100110011001100110011001100110011001100110011

Iteration Number: 3
           dReg                       dRes                       dFPU
   3.023529411764705799       3.023529411764705799       3.023529411764705799
dReg: +11.000001100000011000000110000001100000011000000110000
dRes: +11.000001100000011000000110000001100000011000000110000
dFPU: +11.000001100000011000000110000001100000011000000110000

Iteration Number: 4
           dReg                       dRes                       dFPU
   3.000091554131380178       3.000091554131380178       3.000091554131380178
dReg: +11.000000000000011000000000000001100000000000000110000
dRes: +11.000000000000011000000000000001100000000000000110000
dFPU: +11.000000000000011000000000000001100000000000000110000

Iteration Number: 5
           dReg                       dRes                       dFPU
   3.000000001396983862       3.000000001396983862       3.000000001396983862
dReg: +11.000000000000000000000000000001100000000000000000000
dRes: +11.000000000000000000000000000001100000000000000000000
dFPU: +11.000000000000000000000000000001100000000000000000000

Iteration Number: 6
           dReg                       dRes                       dFPU
            3                          3                          3
dReg: +11.000000000000000000000000000000000000000000000000000
dRes: +11.000000000000000000000000000000000000000000000000000
dFPU: +11.000000000000000000000000000000000000000000000000000

Iteration Number: 7
           dReg                       dRes                       dFPU
            3                          3                          3
dReg: +11.000000000000000000000000000000000000000000000000000
dRes: +11.000000000000000000000000000000000000000000000000000
dFPU: +11.000000000000000000000000000000000000000000000000000
```

Iteration six computed as a calculation history contains references only to the values used in the first computation. This can be seen in Listing AF-3. Using 137 floating-point instructions it computes the exact value independent of preceding calculations. In fact, by simply replacing the value of variable *d0* with another value in the sixth or seventh calculation, the square root of that value can be almost exactly computed.

Listing AF-3- Floating-Point Unit Code for Experiment One, Iteration Six

```
{ /* proc_13- Iteration 6 of Test Case 6, Experiment 1 */
    IEEE754Real8_struct d0 = {0x0l, 0x40220000l}; /*              9 */
    IEEE754Real8_struct d1 = {0x0l, 0x40000000l}; /*              2 */
    IEEE754Real8_struct dRes2;
    IEEE754Real8_struct dRes6;
    IEEE754Real8_struct dRes9;
    IEEE754Real8_struct dRes14;
    IEEE754Real8_struct dRes17;
    IEEE754Real8_struct dRes21;
    IEEE754Real8_struct dRes24;
    IEEE754Real8_struct dRes30;
    IEEE754Real8_struct dRes33;
    IEEE754Real8_struct dRes37;
    IEEE754Real8_struct dRes40;
    IEEE754Real8_struct dRes45;
    IEEE754Real8_struct dRes48;
    IEEE754Real8_struct dRes52;
    IEEE754Real8_struct dRes55;
    IEEE754Real8_struct dRes62;
    char *pszBits;
    { /* add to Area so far */
    asm
        { /* Do FPU stuff */
             /* This is in Appendix AF-2 */
        } /* Do FPU stuff */
    } /* add to Area so far */
    printf("\nCalculation index: 13\n");
    pszBits = procIEE754DblToBin(dRes62.dVal);
    printf("   dFPU: %25.20g, Binary: %s\n", dRes62.dVal, pszBits);
    free(pszBits);
    return 0;
    /* Number of Store Operations: 16 */
} /* proc_13 */
```

Experiments Two and Three served two purposes. The first purpose was to compare the effectiveness of the calculation history approach to the regular approach. This was accomplished in the second experiment, referred to in the results tables after the prompt “b.” This is a regular calculation without limiting the number of reduction tree elements in the saved calculation. The second purpose was to determine if flattening had any effect on the results; this was accomplished in Experiment Three in which the desired maximum number of saved reduction tree elements was one.

The only difference between these two experiments and the first is that ten (10) was the number that had its square root computed. The square root of ten is a product of the square roots of two and five which are both irrational numbers in that the fractional part of their values is unending and nonrepetitive. Having no common divisor other than one, the same would be true of their product. Once a floating-point value that is equal or adjacent to (if there is no exact floating-point value) the correct square root has been reached, subsequent iterations can change only the value of the least significant bit of the computed value.

The results of Experiments Two and Three are displayed in Table AF-2.

Similar to Experiment One, the value of the square root was reached at the iteration six. Beyond this iteration, the regular approach computed the same value. But the calculation history floating-point processor did not. Its value, *dFPU*, computed by calculation histories, oscillated between two adjacent floating-point values. The *dFPU* lower value, 3.162277660168379079, is less than the value, *dReg*, given by the regular approach, 3.162277660168379523. They would be equal if the least significant bit in *dFPU* were a zero bit. The upper value of *dFPU*, however, is the same as the *dReg* value. The correct value for the square root of ten is 3.162277660168379332 which is between the two values between which *dFPU* oscillated. The calculation history approach seemed to realize it can not compute exactly the correct value so it alternated between the appropriate two contiguous floating-point values.

The effect of pruning is visible by comparing the variable *dFPU* of experiments two and three. These are the “b:” and “c:” lines, respectively, in Table AF-2. The “Saved REs Count” denotes the number of reduction tree elements used to recreate the value of the calculation history variable the next time the variable is used in a calculation history computation. Pruning removes the actual values of operands to create the variable’s most recent value, so a truncated value is used instead of the original operands.

Table AF-2- Results of Experiment Two

```
             b:                                  c:
        Experiment Two                     Experiment Three
m_nReduxAnalyzeCountsSize=unlimited  m_nReduxAnalyzeCountsSize=1

Iteration Number: 1
         dReg                     dRes                   dFPU
         5.5           b:          5.5                    5.5
                       c:          5.5                    5.5
b: dRes: +101.10000000000000000000000000000000000000000000000000
c: dRes: +101.10000000000000000000000000000000000000000000000000
   dReg: +101.10000000000000000000000000000000000000000000000000
b: dFPU: +101.10000000000000000000000000000000000000000000000000
c: dFPU: +101.10000000000000000000000000000000000000000000000000
b: Program REs Count: 3, Populated REs Count: 3, Saved REs Count: 3
c: Program REs Count: 3, Populated REs Count: 3, Saved REs Count: 3

Iteration Number: 2
         dReg                     dRes                   dFPU
 3.659090909090909172  b:  3.659090909090909172   3.659090909090909172
                       c:  3.659090909090909172   3.659090909090909172
b: dRes: +11.101010001011101000101110100010111010001011101000110
c: dRes: +11.101010001011101000101110100010111010001011101000110
   dReg: +11.101010001011101000101110100010111010001011101000110
b: dFPU: +11.101010001011101000101110100010111010001011101000110
c: dFPU: +11.101010001011101000101110100010111010001011101000110
b: Program REs Count: 3, Populated REs Count: 9, Saved REs Count: 9
c: Program REs Count: 3, Populated REs Count: 9, Saved REs Count: 2

Iteration Number: 3
         dReg                     dRes                   dFPU
 3.196005081874647047  b:  3.196005081874647047   3.196005081874647047
                       c:  3.196005081874647047   3.196005081874647047
b: dRes: +11.001100100010110101100011100110001000000001011100100
c: dRes: +11.001100100010110101100011100110001000000001011100100
   dReg: +11.001100100010110101100011100110001000000001011100100
b: dFPU: +11.001100100010110101100011100110001000000001011100100
c: dFPU: +11.001100100010110101100011100110001000000001011100100
b: Program REs Count: 3, Populated REs Count: 21, Saved REs Count: 21
c: Program REs Count: 3, Populated REs Count: 7, Saved REs Count: 2

Iteration Number: 4
         dReg                     dRes                   dFPU
 3.162455622803890254  b:  3.16245562280388981    3.16245562280388981
                       c:  3.16245562280388981    3.162455622803890254
b: dRes: +11.001010011001011010110001000100101111111001111000000
c: dRes: +11.001010011001011010110001000100101111111001111000000
   dReg: +11.001010011001011010110001000100101111111001111000001
b: dFPU: +11.001010011001011010110001000100101111111001111000000
c: dFPU: +11.001010011001011010110001000100101111111001111000001
b: Program REs Count: 3, Populated REs Count: 45, Saved REs Count: 45
c: Program REs Count: 3, Populated REs Count: 7, Saved REs Count: 2

Iteration Number: 5
         dReg                     dRes                   dFPU
 3.16227766517567499   b:  3.16227766517567499    3.16227766517567499
                       c:  3.16227766517567499    3.16227766517567499
b: dRes: +11.001010011000101100000011101110000110011001111110110
c: dRes: +11.001010011000101100000011101110000110011001111110110
   dReg: +11.001010011000101100000011101110000110011001111110110
b: dFPU: +11.001010011000101100000011101110000110011001111110110
c: dFPU: +11.001010011000101100000011101110000110011001111110110
b: Program REs Count: 3, Populated REs Count: 93, Saved REs Count: 93
c: Program REs Count: 3, Populated REs Count: 7, Saved REs Count: 2

Iteration Number: 6
         dReg                     dRes                   dFPU
 3.162277660168379523  b:  3.162277660168379079   3.162277660168379079
                       c:  3.162277660168379079   3.162277660168379523
```

```
b: dRes: +11.001010011000101100000111010110110100101101101010010
c: dRes: +11.001010011000101100000111010110110100101101101010010
   dReg: +11.001010011000101100000111010110110100101101101010011
b: dFPU: +11.001010011000101100000111010110110100101101101010010
c: dFPU: +11.001010011000101100000111010110110100101101101010011
b: Program REs Count: 3, Populated REs Count: 189, Saved REs Count: 189
c: Program REs Count: 3, Populated REs Count: 7, Saved REs Count: 2

Iteration Number: 7
         dReg                      dRes                    dFPU
 3.162277660168379523  b:  3.162277660168379079   3.162277660168379523
                       c:  3.162277660168379079   3.162277660168379523
b: dRes: +11.001010011000101100000111010110110100101101101010010
c: dRes: +11.001010011000101100000111010110110100101101101010010
   dReg: +11.001010011000101100000111010110110100101101101010011
b: dFPU: +11.001010011000101100000111010110110100101101101010011
c: dFPU: +11.001010011000101100000111010110110100101101101010011
b: Program REs Count: 3, Populated REs Count: 381, Saved REs Count: 381
c: Program REs Count: 3, Populated REs Count: 7, Saved REs Count: 2

Iteration Number: 8
         dReg                      dRes                    dFPU
 3.162277660168379523  b:  3.162277660168379079   3.162277660168379079
                       c:  3.162277660168379079   3.162277660168379523
b: dRes: +11.001010011000101100000111010110110100101101101010010
c: dRes: +11.001010011000101100000111010110110100101101101010010
   dReg: +11.001010011000101100000111010110110100101101101010011
b: dFPU: +11.001010011000101100000111010110110100101101101010010
c: dFPU: +11.001010011000101100000111010110110100101101101010011
b: Program REs Count: 3, Populated REs Count: 765, Saved REs Count: 765
c: Program REs Count: 3, Populated REs Count: 7, Saved REs Count: 2

Iteration Number: 9
         dReg                      dRes                    dFPU
 3.162277660168379523  b:  3.162277660168379079   3.162277660168379523
                       c:  3.162277660168379079   3.162277660168379523
b: dRes: +11.001010011000101100000111010110110100101101101010010
c: dRes: +11.001010011000101100000111010110110100101101101010010
   dReg: +11.001010011000101100000111010110110100101101101010011
b: dFPU: +11.001010011000101100000111010110110100101101101010011
c: dFPU: +11.001010011000101100000111010110110100101101101010011
b: Program REs Count: 3, Populated REs Count: 1533, Saved REs Count:
1533
c: Program REs Count: 3, Populated REs Count: 7, Saved REs Count: 2

Iteration Number: 10
         dReg                      dRes                    dFPU
 3.162277660168379523  b:  3.162277660168379079   3.162277660168379079
                       c:  3.162277660168379079   3.162277660168379523
b: dRes: +11.001010011000101100000111010110110100101101101010010
c: dRes: +11.001010011000101100000111010110110100101101101010010
   dReg: +11.001010011000101100000111010110110100101101101010011
b: dFPU: +11.001010011000101100000111010110110100101101101010010
c: dFPU: +11.001010011000101100000111010110110100101101101010011
b: Program REs Count: 3, Populated REs Count: 3069, Saved REs Count:
3069
c: Program REs Count: 3, Populated REs Count: 7, Saved REs Count: 2

Iteration Number: 11
         dReg                      dRes                    dFPU
 3.162277660168379523  b:  3.162277660168379079   3.162277660168379523
                       c:  3.162277660168379079   3.162277660168379523
b: dRes: +11.001010011000101100000111010110110100101101101010010
c: dRes: +11.001010011000101100000111010110110100101101101010010
   dReg: +11.001010011000101100000111010110110100101101101010011
b: dFPU: +11.001010011000101100000111010110110100101101101010011
c: dFPU: +11.001010011000101100000111010110110100101101101010011
b: Program REs Count: 3, Populated REs Count: 6141, Saved REs Count:
2557
c: Program REs Count: 3, Populated REs Count: 7, Saved REs Count: 2
```

The overall effect of pruning was noticeable in that the calculation of variable *dFPU* settled to one fixed value, rather than oscillating between two values. This may be a result of the method used to compute the stored, truncated values. Because it can not directly execute FPU instructions, procedure “procResultValue” computes the values replacing reduction tree elements in program CHProcEng. The result of each reduction tree element operation performed is subject to truncation for program storage. These truncated values determined by software are the values saved with a pruned calculation history. These values could more accurate if the operations were performed as one series of operations in the FPU’s registers.

Chapter 5

Conclusions, Implications, Recommendations, and Summary

Conclusions:

Comparing the results of the calculation history approach to those of regular “C” language program approach, it is seen that the target goal of maintaining a more precise calculation has been met.

The regular “C” language approach displayed precision loss on calculations that required more than the 53 bits allowed to represent an 8-byte real floating-point value. In fact, as the calculation of a value progressed, the precision loss became more pronounced. On the other hand, the calculation history approach, because of its ability to dynamically restructure a calculation, showed complete precision retention as the repetitive calculation of a value progressed.

The ability to replace elements of a calculation tree with run-time created calculation tree elements provides an effective method of modifying a calculation into a more accurate structure. That ability alone is not sufficient to attain maximum precision retention. Data-sensitive algorithms must govern the modified structure and its sequence of operations.

Implications:

The regular approach of computing a floating-point value cannot be relied upon to deliver a correct value for all encoded calculations. Even when a calculation may appear to provide a reasonable answer, it may not be close. To retain maximum precision in a calculation, a different approach is needed.

The superiority of the calculation history approach indicates that calculations must be structured in a way that, as the values of the operands change, the structure of the calculation must change.

Recommendations:

The architecture should be further developed. This dissertation concentrated on addition and subtraction. Multiplication and division resulting in values with widths greater than 53 bits should be investigated further.

The ability to handle function calls as variables should be developed. Unary operations will add greatly to the utility of the architecture. Their support should be added.

The reliance on reduction trees is the processing engine's greatest strength. The operator of a reduction tree element may give a clue if a catastrophic cancellation or some other computational disaster might occur. For example, the function $f(x) = x - sin(x)$ fails near $x = 0$. A minus operator with a variable x left operand and a right operand pointing to $sin(x)$ indicates to the preprocessor a possible catastrophic cancellation which might cause serious inaccuracy. If the history of the value of x were known, the calculation might be restructured more accurately. This scenario needs to be explored and developed.

Summary:

The primary hypothesis of this dissertation was that, by maintaining the calculational past history of a variable apart from the variable's value, an equivalent calculation using that variable can be constructed to minimize precision loss. While the calculations are encoded in a high-level language before compile time, precision loss minimization can be accomplished only at run time. Currently, sufficient information to minimize precision loss is not supplied the run-time machine. A stated goal of this dissertation was to establish an architecture that would allow calculations to be coded at a high-level but, when executed, be able to be restructured based on operand values to minimize precision loss. The architecture developed for this dissertation consisted of a compiler, a run-time application, and a utility application to perform calculations using an 80x87 floating-point unit.

The compiler, named "CHFort," was constructed as a subset of the FORTRAN programming language. The source code was processed using Look-Ahead-Left-Recursive methods. This produced a calculation as a stack oriented series of instructions in a textual output object file. The textual output file also included properties for each of the named variables and values the calculation referenced. This provided sufficient knowledge about the structure of a calculation in an easily understood format without specifying exactly how the calculation was to be performed.

The development of an application as a run-time machine or virtual machine was part of the architecture proposed by this dissertation. This run-time application, named "CHProcEng," used the output object file to perform the processing of the code given in the original high-level source file. The application itself could not execute the actual floating-point instructions, although it generated the machine level code to do so. A separate application was developed to do the actual machine level floating-point processing.

Test cases were coded in a high-level language in a source text file. The source text file was input to the CHFort compiler. The CHFort compiler generated a textual output object file containing all the information of the source text file in a manner the CHProcEng run-time machine could interpret it. This output object file transmitted each calculation that would be performed in stack machine records. This allowed the calculation to be exactly reconstructed by the run-time application. The run-time application was able to convert the stack machine records into a computation tree for internal use. The structure of this computation tree lent itself more readily to the restructuring.

Before calculating the value of a calculation history variable, operands on the calculation tree that were designated as calculation history variables were replaced by their stored calculation tree. Each calculation tree element held an operation and its left and right operands. The applicable operand was replaced by a pointer to the top of that history variable's last saved computation tree. All other variables were replaced by their most recent values.

Once the values of the operands of a calculation tree were populated, the calculation tree could be restructured. This was accomplished by Operand Classification and Flattening. Operand Classification assigned one of two classes of mathematical operations to each calculation tree element. Flattening linearized the calculation tree elements of a given class. This linearization allowed different integer-based algorithms to restructure the calculation depending upon the values of the operands using floating-point tuples. Floating-point tuples separate the floating-point value into three fundamental components. The first two components are the base two powers of the most and least significant bits. The third component, the number of significant bits, is computed from the first two components. The fourth component is the sign of the value. If the value is zero, the sign is zero. A new calculation tree was formed from the restructured calculation tree.

This was followed by the development of calculation chains in which the result of the previous operation is an operand to the current operation. Calculation chains were calculation tree segments that allowed their computation to be performed entirely within the floating-point unit without intermediate storage. The information for each calculation chain was contained in a calculation chain element. The value of each calculation chain element was the computed value of its calculation tree segment. This allowed what was called “Bottom-Up Pruning.” In the case of saving an excessively large calculation history, Bottom-Up Pruning allowed the calculation tree operand consisting of a calculation chain to be replaced by the value of its calculation chain element. This would cause the calculation elements to be removed from the calculation tree and would reduce the number of calculation tree elements.

The calculation was performed by two methods. The first method was computing inside the run-time application using “C" language construct such as $dRes = A + B$ on each calculation element. This caused the truncated results of each mathematical operation to be stored. The second method was performing the calculation using the floating-point unit. This was the “calculation history” approach. The floating-point instructions were generated to minimize intermediate truncation of temporary results to program storage. These floating-point instructions were executed in a separate program.

A baseline set of calculations was performed by writing an equivalent “C" language program and executing it. This was called the “regular approach” and provided the results of programming a test case traditionally. Also, the predicted value for some test cases could be determined by mathematical formula.

The results from each of the methods compared were compiled for each calculation of interest into a results table. The results of this table were analyzed in Chapter 4. Calculations performed using the calculation history approach, with the appropriately chosen integer algorithms, were able to compute a precise result. Calculations performed traditionally lost precision, or, in some cases, never reached a valid calculated value.

The conclusion was that arriving at a completely precise result requires more information than is provided by current programming approaches. A high-level front end must provide the execution machine with additional information to improve the precision of the calculated result. The execution machine must apply algorithms to properly restructure the operands and operators of a calculation before the calculation is performed.

The architecture developed for this dissertation was effective in providing a framework in which integer algorithms were implemented and tested successfully. The software units that were part of this framework can serve as a foundation for further research and testing toward maximum precision retention in more complex calculations.

Appendix A

Pseudo YACC CHFort Language Statement Definition

```
LEGEND-
Quoted Strings are those between double quotes:
  Example: "ARE" represents the character string "ARE."
Square Brackets indicate a series of tokens acting as one entity.
Angle Brackets denote a set from which only one item is selected.
  Example: Only a single value of 0, or 1 or 3 can be used from <0, 1,
    3>.
"+" after an entity specifies zero or one occurrence of the preceding
    entity.
"*" after an entity specifies zero or more occurrences of the preceding
    entity.

%token STRING  /* variable name */
%token NUMBER /* either an integer or floating-point value */
%token LOGIAL /* a logical value ".TRUE" or ".FALSE"
%token BOOL_COMP /* ".LT." | ".LE." | ".EQ." | ".NE." | ".GE." | ".GT."
    */
%token ARITH_OPERATOR /* "+" | "-" | "*" | "/"  "*" | "**"  */
/* precedences in ascending order */
%prec "+", "-"
%prec "*", "/"
%prec "**"
/* associativity */
%left "+", "-"
%left "*", "/"
%right "**"  /* raise to a power */

EXPR
    : EXPR ARITH_OPERATOR EXPR
    | EXPR BOOL_COMP EXPR
    | NUMBER
    | NAME
    | NAME '(' expression ')'
    ;

DIMENSION_SPEC : "(" EXPR [, EXPR]* ")"
VALUE_ASSIGNMENT : "=" EXPR

DATATYPE_ATTRIBUTE
   : "HISTORY"
   | "PARAMETER"
   ;
VARIABLE_LIST
   : STRING [DIMENSION_SPEC]+ VALUE_ASSIGNMENT*
   | VARIABLE_LIST ["," STRING [DIMENSION_SPEC]+ VALUE_ASSIGNMENT*]*
   ;

INTEGER_Declaration
   : 'INTEGER' '::' |
   | "INTEGER" ["," DATATYPE_ATTRIBUTE]* "::" VARIABLE_LIST
   | "INTEGER" VARIABLE_LIST
   ;
```

```
REAL_Declaration
   : "REAL" '::' |
   | "REAL" ["," DATATYPE_ATTRIBUTE]* "::" VARIABLE_LIST
   | "REAL" VARIABLE_LIST
   | "DOUBLE PRECISION" '::' |
   | " DOUBLE PRECISION " ["," DATATYPE_ATTRIBUTE]* "::" VARIABLE_LIST
   | " DOUBLE PRECISION " VARIABLE_LIST
   ;
LOGICAL_Declaration
   : 'LOGICAL' '::' |
   | "LOGICAL" ["(KIND=" EXPR ")"]+ ["," DATATYPE_ATTRIBUTE]* "::"
VARIABLE_LIST
   | "LOGICAL" VARIABLE_LIST
   ;

IF_START_Statement : "IF" "(" EXPR ")" "THEN"
                   | "IF" "(" EXPR ")" "THEN" EXPR  ;
IF_COND_Statement : "ELSE" "IF" "(" EXPR ")" "THEN" ;
IF_ELSE_Statement : "ELSE" ;
IF_END_Statement : "END" "IF" ;
COMMENT_Statement : "!" [(any character)]*

ASSIGN_Statement : STRING “=” EXPR

CHFort_Statement
    : NAME '=' EXPR
    | INTEGER_Declaration
    | REAL_Declaration
    | LOGICAL_Declaration
    | IF_START_Statement
    | IF_COND_Statement
    | IF_ELSE_Statement
    | IF_END_Statement
    | ASSIGN_Statement
    | COMMENT_Statement
    ;

/* Source record layout */
Statement_First : [CHFort_Statement] ;
ContinuedStatement_First : [First Part of CHFort_Statement] “ &” ;
ContinuedStatement_Continued
: [“ &”] [Remaining Part of CHFort_Statement] “ &” ;
```

Appendix B

Sample CHFort Output Object File

```
<<variablesvalues>>
Name: dYSum
Value:
IsHistory: 1
IsFixed: 0
DataType: 5
nDimsCount: 0
Name: dXMid
Value:
IsHistory: 1
IsFixed: 0
DataType: 5
nDimsCount: 0
Name: dX0
Value:
IsHistory: 0
IsFixed: 0
DataType: 5
nDimsCount: 0
Name: dSpan
Value:
IsHistory: 0
IsFixed: 0
DataType: 5
nDimsCount: 0
Name: dSpanIncs
Value:
IsHistory: 0
IsFixed: 0
DataType: 5
nDimsCount: 0
Name: dSpanDelta
Value:
IsHistory: 0
IsFixed: 0
DataType: 5
nDimsCount: 0
<<ProgramCode>>
Label StartProg
BeginCode dX0
push Number 0
EndCode
BeginCode dSpan
push Number 1
EndCode
BeginCode dSpanIncs
push Number 199
EndCode
BeginCode dSpanDelta
  push String dSpan
  push String dSpanIncs
  DivSingle
```

```
EndCode
BeginCode dXMid0
  push String dX0
  push String dSpanDelta
  push Number 2
  DivSingle
  Plus
EndCode
BeginCode dXMid
push String dXMid0
EndCode
BeginCode dYSum
push Number 0
EndCode
Label 100
BeginBoolCode dIf_Bool_1_0
  push String dXMid
  push String dSpan
  Minus
EndCode
GoToCond 999 GreaterThan dIf_Bool_1_0
BeginCode dYSum
  push String dYSum
  push String dXmid
  Plus
EndCode
BeginCode dXMid
  push String dXMid
  push String dSpanDelta
  Plus
EndCode
GoTo 100
Label 999
Label EndProg
```

Appendix C

Procedure “procMakeExpr”

```
_proc ShiftElement_struct *procMakeExpr(int nOpFlags, int *nEndWhy, int nStartTEIx, int
    *nTermTEIx)
{ /* procMakeExpr- Builds a calculation expression */
    /* nEndWhy
        0 End Of Data
        1 End of Expression (nTermClass == 1)
        -1 End Other
    */
    int nTermClass, nBoolClass;
    /* nTermClass
        0 Arithmetic expression
        1 Terminate expression on ")"
       nBoolClass
        0 Not a boolean expression
        1 This is a boolean expression
    */
    ShiftElement_struct *pTopSE;
    TokenElement_Struct *pThisTE, *pNextTE; int nThisTEIx;
    int nGetTE;
    int nShiftsCount, nSEIx;
    int nReduxsCount;
    int nPrensCount;
    int nArgType;
    /* nArgType
         0 Force End of Calculation
         1 Operand
         2 Operator
         3 Left Parenthesis
         4 Right Parenthesis
    */
    ShiftElement_struct *pNewSE;
    int nReduxOp;
    ReductElement_struct *pNewRE;
    /**/
    ShiftElement_struct *pPrev1SE, *pPrev2SE, *pPrev3SE;
    int nIsEOFxnCall;
    /*
        Announce procdure
    */
    nTermClass = nOpFlags & 0x000f;
    nBoolClass = (nOpFlags >> 4) & 0x000f;
    nShiftsCount = 0; nSEIx = 0;
    nReduxsCount = 0;
    nPrensCount = 0;
    /**/
    m_pFirstSE = NULL; m_pLastSE = NULL;
    m_nShiftElementsCount = 0;
    /**/
    for (nGetTE = 2, nThisTEIx = nStartTEIx;;)
    { /* Process all tokens in expression */
        if (nGetTE)
        { /* Provide a Token Element */
            /**/
            if (nGetTE == 1) nThisTEIx++;
            /**/
            nArgType = 0; /* Default to End Expression */
            if (nThisTEIx >= m_nTokenElementsCount)
            { /* No more tokens */
                nArgType = 0;
                *nEndWhy = 0;
            } /* No more tokens */
            else
            { /* At least one other token */
                *nEndWhy = -1;
                pThisTE = procGetTokenElementFromIx(nThisTEIx);
                if ((pNextTE = pThisTE->pNext) != NULL)
                { /* This may be a "function call" */
                    if ((pThisTE->nTokenType == nTT_String) && (pNextTE->nTokenType ==
    nTT_PrenL))
                    { /* This is a "function call" */
                        pNextTE->nTokenType = nTT_ArgTaker;
                        free(pNextTE->pszValue);
                        strcpy(pNextTE->pszValue, pThisTE->pszValue);
                        pThisTE = pNextTE; nThisTEIx++;
                    } /* This is a "function call" */
                } /* This may be a "function call" */
                if ((nTermClass == 1) && (nPrensCount == 0) && (pThisTE->nTokenType ==
    nTT_PrenR))
                { /* Right side of expression encountered */
                    nArgType = 0;
                    *nEndWhy = 1;
                } /* Right side of expression encountered */
                else
                { /* Not forced termination */
                    switch (pThisTE->nTokenType)
                    { /* switch (pThisTE->nTokenType) */
                    case nTT_String: case nTT_StringSingle: case nTT_Number: case
    nTT_StringQuote:
                        nArgType = 1; break;
                    case nTT_Plus: case nTT_Minus: case nTT_StarSingle:
                    case nTT_DivSingle: case nTT_BConjAnd: case nTT_BConjOr:
                    case nTT_LT: case nTT_LE: case nTT_EQ:
                    case nTT_GE: case nTT_GT: case nTT_NE:
                        nArgType = 2; break;
                    case nTT_PrenL: nTT_ArgTaker:
                        nArgType = 3; break;
                    case nTT_PrenR:
                        if (nPrensCount > 0) nArgType = 4; break;
                    } /* switch (pThisTE->nTokenType) */
                } /* Not forced termination */
            } /* At least one other token */
        } /* Provide a Token Element */
        nGetTE = 1;
        /*
            Treat token element
        */
        pNewSE = malloc(sizeof(ShiftElement_struct));
        LinkAppendNew(pNewSE, m_pFirstSE, m_pLastSE);
        m_nShiftElementsCount++;
        /**/
        pNewSE->nArgType = nArgType;
        pNewSE->nReduxIx = -1;
        pNewSE->nSEIx = nSEIx++;
        pNewSE->pPrevShiftElement = NULL;
        pNewSE->pNextShiftElement = NULL;
        pNewSE->pReduxNode = NULL;
        pNewSE->pFirstArg = NULL;
        pNewSE->pLastArg = NULL;
        pNewSE->nArgsCount = 0;
        /**/
        if (nArgType == 0)
        { /* End of Expression */
            /**/
            pNewSE->TokenElement.nAssoc = 0;
            pNewSE->TokenElement.nPrecedence = 0;
            pNewSE->TokenElement.nTokenType = nTT_EndExpr;
            pNewSE->TokenElement.pszValue = NULL;
            *nTermTEIx = nThisTEIx;
            /**/
        } /* End of Expression */
        else
        { /* Input Token */
            /**/
            memcpy(&pNewSE->TokenElement, pThisTE, sizeof(TokenElement_Struct));
            /**/
        } /* Input Token */
        /**/
        if (pNewSE->TokenElement.nTokenType == nTT_PrenL) nPrensCount++;
        if (pNewSE->TokenElement.nTokenType == nTT_PrenR) nPrensCount--;
        /*
        ***************************************************************************
        *
        * Perform all reductions forced by this Shift Element
        *
        ***************************************************************************
        */
        for (;;)
        { /* Perform all reductions forced by this Shift Element */
            /*
                Make the job below easier
            */
            pPrev1SE = NULL; pPrev2SE = NULL; pPrev3SE = NULL;
            if (m_nShiftElementsCount >= 1) pPrev1SE = m_pLastSE->pPrev;
            if (m_nShiftElementsCount >= 2) pPrev2SE = pPrev1SE->pPrev;
            if (m_nShiftElementsCount >= 3) pPrev3SE = pPrev2SE->pPrev;
            /*
                Test if operator forces a reduction
            */
            nReduxOp = 0;
            if (m_pLastSE->nArgType == 2)
            { /* May have to reduce an operand to new operator */
                if (m_nShiftElementsCount > 3) nReduxOp = 1;
            } /* May have to reduce an operand to new operator */
            /*
                Test if Parenthesis forces a reduction
            */
            else if (m_pLastSE->nArgType == 4)
            { /* Test if parenthesis forces a reduction */
                if (m_nShiftElementsCount > 4) nReduxOp = 2;
            } /* Test if parenthesis forces a reduction */
            /*
                End of Input
            */
            else if (m_pLastSE->nArgType == 0)
            { /* End of Input */
                if (m_nShiftElementsCount > 3) nReduxOp = 3;
            } /* End of Input */
            /**/
            if (nReduxOp == 0) break;
            /*
                Must have three preceding terms to reduce
            */
            if (m_nShiftElementsCount < 4)
                nReduxOp = 0;
            else
            { /* May force Last On to stack reduction */
                if ((pPrev1SE->nArgType != 1) ||
                    (pPrev2SE->nArgType != 2) ||
                    (pPrev3SE->nArgType != 1))
                { /* Not reducible yet */
                    nReduxOp = 0;
                } /* Not reducible yet */
                else
                { /* Check if operators force a reduction */
                    int nDiffPrec;
                    /**/
                    nDiffPrec = -1;
                    if (nReduxOp == 1)
                    { /* Must check precedence and/or associativity */
                        nDiffPrec = m_pLastSE->TokenElement.nPrecedence - pPrev2SE-
    >TokenElement.nPrecedence;
                        if (nDiffPrec == 0)
                        { /* Force if left associative */
                            if (pPrev2SE->TokenElement.nAssoc < 0) nDiffPrec = -1;
                        } /* Force if left associative */
                    } /* Must check precedence and/or associativity */
                    if (nDiffPrec >= 0) nReduxOp = 0;
                } /* Check if operators force a reduction */
            } /* May force Last On to stack reduction */
            /**/
            if (nReduxOp == 0) break;
            /*
                Must reduce the stack
            */
            if (nReduxOp)
            { /* Reduce the argument to an operator */
                int nArgIx;
                /**/
                pNewRE = malloc(sizeof(ReductElement_struct));
                LinkAppendNew(pNewRE, m_pFirstRE, m_pLastRE);
                m_nReductElementsCount++;
                /**/
                memcpy(&pNewRE->OpToken, &pPrev2SE->TokenElement,
    sizeof(TokenElement_Struct));
                memcpy(&pNewRE->ArgsToken[0], &pPrev3SE->TokenElement,
    sizeof(TokenElement_Struct));
                memcpy(&pNewRE->ArgsToken[1], &pPrev1SE->TokenElement,
    sizeof(TokenElement_Struct));
                pNewRE->pParentRE = NULL;
                pNewRE->pArgsRE[0] = NULL;
                pNewRE->pArgsRE[1] = NULL;
                /**/
                for (nArgIx = 0; nArgIx < 2; nArgIx++)
                { /* Operand Shift Element may be a Reduction Element */
                    ShiftElement_struct *pArgSE;
                    ReductElement_struct *pArgRE;
                    /**/
                    pArgSE = nArgIx? pPrev1SE: pPrev3SE;
                    pArgRE = pArgSE->pReduxNode;
                    if (pArgRE)
                    { /* Operand is a reduction */
                        pArgRE->pParentRE = pNewRE;
                        pArgRE->nParentREIx = nArgIx;
                        pNewRE->pArgsRE[nArgIx] = pArgRE;
                    } /* Operand is a reduction */
                } /* Operand Shift Element is a Reduction Element */
                /*
                    Modify Shift Element to reflect it is a reduction
                */
                pPrev2SE->pReduxNode = pNewRE;
                pPrev2SE->nArgType = 1; /* This is now an operand */
                /*
                    Do not need Operand Shift Elements
                */
                LinkRemove(pPrev1SE, m_pFirstSE, m_pLastSE);
                m_nShiftElementsCount--;
                LinkRemove(pPrev3SE, m_pFirstSE, m_pLastSE);
                m_nShiftElementsCount--;
            } /* Reduce the argument to an operator */
            /**/
        } /* Perform all reductions forced by this Shift Element */
        /*
        ***************************************************************************
        *
        * All reductions by added Shift Element performed
        *
        ***************************************************************************
        */
        /*
            Test if end of Function Call
        */
        nIsEOFxnCall = 0;
        if ((((m_pLastSE->nArgType == 0) || (m_pLastSE->nArgType == 4))) &&
    (m_nShiftElementsCount >= 3))
        { /* May be a function call */
            if (pPrev2SE->TokenElement.nTokenType == nTT_ArgTaker)
            { /* This ends a term of a function call */
                if ((m_pLastSE->nArgType == 0) &&
                    (m_pLastSE->TokenElement.nTokenType == nTT_Comma)) nIsEOFxnCall = 1;
                if ((m_pLastSE->nArgType == 4) &&
                    (m_pLastSE->TokenElement.nTokenType == nTT_PrenR)) nIsEOFxnCall = 2;
            } /* This ends a term of a function call */
        } /* May be a function call */
        /**/
        if (nIsEOFxnCall != 0)
        { /* End an argument of a function call */
            LinkAppendNew(m_pLastSE, pPrev2SE->pFirstArg, pPrev2SE->pLastArg);
            LinkRemove(m_pLastSE, m_pFirstSE, m_pLastSE);
            m_nShiftElementsCount--;
            pPrev2SE->pLastArg = pPrev1SE;
            pPrev2SE->nArgsCount++;
            LinkRemove(pPrev1SE, m_pFirstSE, m_pLastSE);
            pPrev2SE->nArgsCount++;
            m_nShiftElementsCount--;
        } /* End an argument of a function call */
        else if (m_pLastSE->nArgType == 0)
        { /* End of all Expression */
            if (m_nShiftElementsCount != 2)
            { /* Junk left on stack */
                printf("\n!!! Junk left on stack at end of Expression !!!\n");
            } /* Junk left on stack */
            else
            { /* Just one element left before EndExpr Token */
                LinkRemove(m_pLastSE, m_pFirstSE, m_pLastSE);
                m_nShiftElementsCount--;
            } /* Just one element left before EndExpr Token */
            break;
        } /* End of all Expression */
        /*
            End of Parenthetical Expression
        */
        else if (m_pLastSE->nArgType == 4)
        { /* End of Parenthetical Expression */
            if (m_nShiftElementsCount >= 3)
            { /* These must be '(', Operand, ')' */
                if ((pPrev1SE->nArgType == 1) && (pPrev2SE->nArgType == 3))
                { /* Can reduce the last three terms */
                    LinkRemove(m_pLastSE, m_pFirstSE, m_pLastSE);
                    m_nShiftElementsCount--;
                    LinkRemove(pPrev2SE, m_pFirstSE, m_pLastSE);
                    m_nShiftElementsCount--;
                } /* Can reduce the last three terms */
                else
                { /* Syntax error */
                    printf("\n-- Can not remove parentheses --\n");
                    exit(-1);
                } /* Syntax error */
            } /* These must be '(', Operand, ')' */
        } /* End of Parenthetical Expression */
        /*
            Determine what to do with this Element
        */
    } /* Process all tokens in expression */
    pTopSE = m_pLastSE;
    /**/
    return pTopSE;
} /* procMakeExpr- Builds a calculation expression */
```

Appendix D

Output from CHFort Compiler

```
<<variablesvalues>>
Name: A
Value:
IsHistory: 1
IsFixed: 0
DataType: 5
nDimsCount: 0
Name: C
Value:
IsHistory: 1
IsFixed: 0
DataType: 5
nDimsCount: 0
<<ProgramCode>>
Label StartProg
BeginCode A
push Number 2
EndCode
BeginCode I
push Number 0
EndCode
Label 001
BeginCode I
  push String I
  push Number 1
  Plus
EndCode
BeginBoolCode dIf_Bool_1_0
  push String I
  push Number 3
  Minus
EndCode
GoToCond 2 GreaterThan dIf_Bool_1_0
BeginCode B
  push String A
  push String I
  Plus
EndCode
BeginCode C
  push String A
  push String B
  Plus
  push String A
  push String B
  Minus
  StarSingle
  push Number 5
  Plus
EndCode
BeginCode A
push String C
EndCode
GoTo 1
Label 002
Label EndProg
```

Appendix E

Procedure “procFlattenExpr”

```
_proc FlattenExpr_struct *procFlattenExpr(ReduxElement_struct *pFlattenRE)
{ /* procFlatten- Flatten a reduction chain */
    /**/
    FlattenExpr_struct *pNewFE;
    int nArgsDn, nArgMask;
    /**/
    for (nArgsDn = 0, nArgMask = 1; nArgsDn < 2; nArgsDn++, nArgMask <<= 1)
    { /* Test if must create a new expression */
        if (pFlattenRE->SEOperands[nArgsDn].pReduxElement)
        { /* This is a reduction */
            if (pFlattenRE->pFlattenExprs[nArgsDn] == NULL)
            { /* must flatten this reduction */
                pFlattenRE->pFlattenExprs[nArgsDn] = procFlattenExpr((ReduxElement_struct
     *)pFlattenRE->SEOperands[nArgsDn].pReduxElement);
            } /* must flatten this reduction */
        } /* This is a reduction */
        else
        { /* This is a value */
            /*
                Create a single element Flatten Expression
            */
            FlattenOperand_struct *pNewFO;
            FlattenTerm_struct *pNewFT;
            TokenData_struct *pArgTD;
            /**/
            pArgTD = &pFlattenRE->SEOperands[nArgsDn].TokenData;
            /*
                Create the expression
            */
            pNewFE = MemGet(sizeof(FlattenExpr_struct));
            LinkAppendNew(pNewFE, m_pFirstFE, m_pLastFE);
            pNewFE->pFirstFT = NULL; pNewFE->pLastFT = NULL;
            m_nFlattenExprsCount++;
            /*
                Create the term element
            */
            pNewFT = MemGet(sizeof(FlattenTerm_struct));
            LinkAppendNew(pNewFT, pNewFE->pFirstFT, pNewFE->pLastFT);
            pNewFE->nTermsCount = 1;
            pNewFT->pFirstFO = NULL; pNewFT->pLastFO = NULL;
            /*
                Create the term's single operand
            */
            pNewFO = MemGet(sizeof(FlattenOperand_struct));
            LinkAppendNew(pNewFO, pNewFT->pFirstFO, pNewFT->pLastFO);
            pNewFT->nElesCount = 1;
            /**/
            pNewFO->nOpTokenType = nTT_Plus;
            memcpy(&pNewFO->Value, &pFlattenRE->SEOperands[nArgsDn].DataValue.DataValue,
     sizeof(Variant_struct));
            pNewFO->Value.enumDataType = pArgTD->nDataType;
            procIEE754Real8ToTuple(pNewFO->Value.dVal, &pNewFO->FlPtTuple);
            /**/
            pFlattenRE->pFlattenExprs[nArgsDn] = pNewFE;
            /**/
        } /* This is a value */
    } /* Test if must create a new expression */
    /*
        Create New Term from both Terms
    */
    { /* Procedure to create new expression from a reduction's operands */
        int nExprsDn;
        int nTermsDo, nTermsDn;
        union {
```

```
        struct  {FlattenExpr_struct *pExp0FE; FlattenExpr_struct *pExp1FE; int
nExp0TermsCount; int nExp1TermsCount;};
        struct {FlattenExpr_struct *pExpsFE[2]; nExpsTermsCount[2];};
    } Exprs;
    TokenTypes_Enum nCalcTT;
    /*
        Create Expression to be developed
    */
    pNewFE = MemGet(sizeof(FlattenExpr_struct));
    LinkAppendNew(pNewFE, m_pFirstFE, m_pLastFE);
    pNewFE->pFirstFT = NULL; pNewFE->pLastFT = NULL;
    pNewFE->nTermsCount = 0;
    /*
        Set up Operand Expressions
    */
    Exprs.pExp0FE = pFlattenRE->pFlattenExprs[0];
    Exprs.nExp0TermsCount = Exprs.pExp0FE->nTermsCount;
    Exprs.pExp1FE = pFlattenRE->pFlattenExprs[1];
    Exprs.nExp1TermsCount = Exprs.pExp1FE->nTermsCount;
    /*
        Set up operation
    */
    nCalcTT = pFlattenRE->SEOperator.TokenData.nTokenType;
    /*
        Create new expression
    */
    if ((nCalcTT == nTT_Plus) || (nCalcTT == nTT_Minus))
    { /* Add or subtract operation */
        /*
            Concatenate strings and maybe modify second (subtract only)
        */
        for (nExprsDn = 0; nExprsDn < 2; nExprsDn++)
        { /* Append this operand's expression to new expression */
            FlattenExpr_struct *pThisFE;
            FlattenTerm_struct *pNewFT, *pThisFT;
            /**/
            pThisFE = pFlattenRE->pFlattenExprs[nExprsDn];
            nTermsDo = pThisFE->nTermsCount;
            /**/
            for (nTermsDn = 0, pThisFT = pThisFE->pFirstFT;
                 nTermsDn < nTermsDo;
                 nTermsDn++, pThisFT = pThisFT->pNext)
            { /* Append this expressions terms to new expression's terms */
                FlattenOperand_struct *pNewFO, *pThisFO;
                int nElesCount, nElesDn;
                /**/
                nElesCount = pThisFT->nElesCount;
                /**/
                pNewFT = MemGet(sizeof(FlattenTerm_struct));
                LinkAppendNew(pNewFT, pNewFE->pFirstFT, pNewFE->pLastFT);
                pNewFE->nTermsCount++;
                /**/
                pNewFT->nElesCount = 0;
                pNewFT->pFirstFO = NULL;
                pNewFT->pLastFO = NULL;
                /*
                    Add this expressions term's Elements to new Term
                */
                for (nElesDn = 0, pThisFO = pThisFT->pFirstFO;
                     nElesDn < nElesCount;
                     nElesDn++, pThisFO = pThisFO->pNext)
                { /* Append this operand to new term's operands */
                    TokenTypes_Enum nEleTT;
                    /**/
                    pNewFO = MemGet(sizeof(FlattenOperand_struct));
                    LinkAppendNew(pNewFO, pNewFT->pFirstFO, pNewFT->pLastFO);
                    pNewFT->nElesCount++;
                    /*
                        Populate new element
                    */
                    nEleTT = pThisFO->nOpTokenType;
                    if (nElesDn == 0)
                    { /* First element of term */
                        if (nExprsDn == 1)
                        { /* First element of second expression */
                            if (nCalcTT == nTT_Minus)
                            { /* Must reverse sign of first element */
                                if (nEleTT == nTT_Plus) nEleTT = nTT_Minus;
                                else nEleTT = nTT_Plus;
                            } /* Must reverse sign of first element */
                        } /* First element of second expression */
                    } /* First element of term */
                    pNewFO->nOpTokenType = nEleTT;
                    pNewFO->Value = pThisFO->Value;
                    procIEE754Real8ToTuple(pNewFO->Value.dVal, &pNewFO->FlPtTuple);
                    /**/
                } /* Append this operand to new term's operands */
            } /* Append this expressions terms to new expression's terms */
            /*pNewFE->nTermsCount++;*/
        } /* Append this operand's expression to new expression */
        /**/
    } /* Add or subtract operation */
    else if ((nCalcTT == nTT_StarSingle) || (nCalcTT == nTT_DivSingle))
    { /* Multiply or divide operation */
        int n0TermsDn, n1TermsDn;
        FlattenTerm_struct *p0FT;
        /*
            First term multiplies the second term
        */
        p0FT = ((FlattenExpr_struct *)pFlattenRE->pFlattenExprs[0])->pFirstFT;
        for (n0TermsDn = 0;
             n0TermsDn < Exprs.nExp0TermsCount;
             n0TermsDn++, p0FT = p0FT->pNext)
        { /* Term of First Expression */
            int n0ElesCount, n1TermsCount, n1TermsDn;
            int n0ElesDo, n0ElesDn;
            FlattenOperand_struct *p0FO, *p1FO;
            FlattenTerm_struct *p1FT;
            /**/
            n0ElesCount = p0FT->nElesCount;
            n1TermsCount = Exprs.nExp1TermsCount;
            /**/
            for (n1TermsDn = 0, p1FT = Exprs.pExp1FE->pFirstFT;
                 n1TermsDn < n1TermsCount;
                 n1TermsDn++, p1FT = p1FT->pNext)
            { /* Create Term the product of these two terms */
                FlattenTerm_struct *pNewFT;
                int n1ElesCount, nNewElesCount, nElesDn;
                FlattenOperand_struct *pTermFO;
                /**/
                n1ElesCount = p1FT->nElesCount;
                nNewElesCount = n1ElesCount + n0ElesCount;
                /**/
                pNewFT = MemGet(sizeof(FlattenTerm_struct));
                LinkAppendNew(pNewFT, pNewFE->pFirstFT, pNewFE->pLastFT);
                pNewFE->nTermsCount++;
                pNewFT->pFirstFO = NULL; pNewFT->pLastFO = NULL;
                pNewFT->nElesCount = nNewElesCount;
                /*
                    Append Elements to new term
                */
                for (nElesDn = 0;
                     nElesDn < nNewElesCount;
                     nElesDn++)
                { /* Append an element */
                    TokenTypes_Enum nEleTT;
                    FlattenOperand_struct *pNewFO;
                    /**/
                    if (nElesDn == 0)
                    { /* First element of first term */
                        pTermFO = p0FT->pFirstFO;
                        nEleTT = pTermFO->nOpTokenType;
```

```
                            if (pTermFO->nOpTokenType == p1FT->pFirstFO->nOpTokenType)
                            { /* Same signs of each expression */
                                nEleTT = nTT_Plus;
                            } /* Same signs of each expression */
                            else
                            { /* Different signs of each expression */
                                nEleTT = nTT_Minus;
                            } /* Different signs of each expression */
                        } /* First element of first term */
                        else if (nElesDn == n0ElesCount)
                        { /* First element of second term */
                            pTermFO = p1FT->pFirstFO;
                            nEleTT = pTermFO->nOpTokenType;
                            nEleTT = nCalcTT;
                        } /* First element of second term */
                        else
                        { /* Susequent element of current term */
                            pTermFO = pTermFO->pNext;
                            nEleTT = pTermFO->nOpTokenType;
                            if (nElesDn >= n0ElesCount) nEleTT = nCalcTT;
                        } /* Susequent element of current term */
                        /*if (nElesDn >= n0ElesCount) nEleTT = nCalcTT;*/
                        /**/
                        pNewFO = MemGet(sizeof(FlattenOperand_struct));
                        LinkAppendNew(pNewFO, pNewFT->pFirstFO, pNewFT->pLastFO);
                        /**/
                        pNewFO->nOpTokenType = nEleTT;
                        pNewFO->Value = pTermFO->Value;
                        /**/
                    } /* Append an element */
                    /**/
                } /* Create Term the product of these two terms */
            } /* Term of First Expression */
        } /* Multiply or divide operation */
        /**/
    } /* Procedure to create new expression from a reduction's operands */
    /*
        Child Expressions are no longer needed
    */
    { /* Remove Expression */
        FlattenExpr_struct *pThisFE;
        int nExprIx;
        /**/
        for (nExprIx = 0; nExprIx < 2; nExprIx++)
        { /* Show this Expression */
            /*
                Remove all terms from this expression
            */
            pThisFE = pFlattenRE->pFlattenExprs[nExprIx];
            procFreeFlattenExpr(pThisFE);
            pFlattenRE->pFlattenExprs[nExprIx] = NULL;
        } /* Show this Expression */
    } /* Remove Expression */
    /*
        Expressions Removed
    */
    return pNewFE;
    /**/
} /* procFlattenExpr- Flatten a reduction chain */
```

Appendix F

Procedure “procResultValue”

```
_proc ResType_Struct procResultValue(ReduxElement_struct *pThisRE, int nOp)
{ /* procResultValue- Returns result data type */
    ResType_Struct ResType, ResType0, ResType1;
    /**/
    int nArgIx;
    ReduxElement_struct *pArgRE;
    int nOpFlag0, nOpFlag1;
    /**/
    nOpFlag0 = nOp & 1;
    nOpFlag1 = nOp & 2;
    /**/
    for (nArgIx = 0; nArgIx < 2; nArgIx++)
    { /* process an argument */
        TokenTypes_Enum nArgTT;
        DataTypes_Enum nDataType;
        StackElement_struct *pArgSE;
        /**/
        pArgRE = pThisRE->SEOperands[nArgIx].pReduxElement;
        pArgSE = &pThisRE->SEOperands[nArgIx];
        if ((nOpFlag0 + nOpFlag1) != nOp)
        { /* problem in pointers */
            printf("\nProblem with Operand pointer agreement- nOpFlag0: %i, nOpFlag1: %i,
    nOp: %i\n", nOpFlag0, nOpFlag1, nOp);
        } /* problem in pointers */
        if (pArgRE)
        { /* Get data from operation */
            ResType = procResultValue2(pArgRE,  nOp);
            ResType.DataValue.enumDataType = nDT_Real8;
            ResType.nDataType =  nDT_Real8;
        } /* Get data from operation */
        else
        { /* Get data from data */
            /**/
            pArgSE = &pThisRE->SEOperands[nArgIx];
            if (pArgSE->nValType == 7)
            { /* Already reduced */
                nDataType = nDT_Real8;
                ResType.DataValue.dVal = pArgSE->DataValue.DataValue.dVal;
                ResType.DataValue = pArgSE->DataValue.DataValue;
            } /* Already reduced */
            else if ((nArgTT = pArgSE->TokenData.nTokenType) == nTT_Number)
            { /* numerical constant */
                double dValue;
                char *pNextCh;
                /**/
                nDataType = nDT_Real8;
                ResType.DataValue.enumDataType = nDataType;
                /**/
                ResType.DataValue.dVal = strtod(pArgSE->TokenData.pszToken, &pNextCh);
                ResType.DataValue.enumDataType = nDT_Real8;
                ResType.nDataType = nDT_Real8;
                /**/
            } /* numerical constant */
            else if ((nArgTT == nTT_StringSingle) || (nArgTT == nTT_String))
            { /* may be either constant or variable */
                UsedName_struct *pUN;
                /**/
                if ((pUN = procFindUsedName(pArgSE->TokenData.pszToken)) == NULL)
                { /* Name not indexed first pass */
                    printf("\n!!! Name not indexed first pass: %s\n", pArgSE-
    >TokenData.pszToken);
                    exit(-1);
                } /* Name not indexed first pass */
                /**/
                nDataType = pUN->ResValue.nDataType;
                ResType.DataValue.enumDataType = nDataType;
                ResType.DataValue.dVal = (nOpFlag1 == 0)?
                  pUN->RegValue.DataValue.dVal:
                  pUN->ResValue.DataValue.dVal;
                /**/
            } /* may be either constant or variable */
            else
            { /* not supported datatype */
                printf("!!! Bad token type Side i: %s, %i!!!\n",
                  nArgIx, pszTokenDescription(nArgTT), nArgTT);
                exit(-1);
            } /* not supported datatype */
            ResType.nIsFixed = 1;
            ResType.nResErr = 0;
            ResType.nDataType = nDataType;
            ResType.DataValue.enumDataType = nDataType;
        } /* Get data from data */
        /**/
        if (nArgIx == 0) ResType0 = ResType;
        else ResType1 = ResType;
    } /* process an argument */
    /*
        Determine resultant data type
    */
    { /* Determine final result type */
        double dArg0, dArg1, dRes;
        DataTypes_Enum nOpDataType;
        TokenTypes_Enum nOpTokenType;
        StackElement_struct *pOperator;
        /**/
        { /* earlier results OK */
            /*
                Determine Data Type of the result of this operation
            */
            pOperator = &pThisRE->SEOperator;
            nOpDataType = nDT_None;
            nOpTokenType = pOperator->TokenData.nTokenType;
            if ((nOpTokenType == nTT_BConjAnd) || (nOpTokenType == nTT_BConjOr))
            { /* Boolean Conjunction returns a boolean */
                nOpDataType = nDT_Bool;
            } /* Boolean Conjunction returns a boolean */
            else if ((nOpTokenType == nTT_Plus) || (nOpTokenType == nTT_Minus) ||
              (nOpTokenType == nTT_StarSingle) || (nOpTokenType == nTT_DivSingle))
            { /* Numerical operation */
                nOpDataType = nDT_Real8;
            } /* Numerical operation */
            else if ((nOpTokenType >= nTT_LT) && (nOpTokenType <= nTT_GT))
            { /* Numerical operation returns a boolean */
                nOpDataType = nDT_Bool;
            } /* Numerical operation returns a boolean */
            else
            { /* Invalid Token Type for Operator */
                printf("\n!!! Invalid Token type for operator- value: %i, desc: %s
    !!!\n",
                  nOpTokenType, pszTokenDescription(nOpTokenType));
                exit(-1);
            } /* Invalid Token Type for Operator */
            ResType.nDataType = nOpDataType;
            /*
                Compute result
            */
            if (ResType0.DataValue.enumDataType != nDT_Real8)
            { /* Must convert to Real8 */
                double dVal;
                /**/
                dVal = 0.;
                if (ResType0.DataValue.enumDataType == nDT_Integer) dRes =
    ResType0.DataValue.lVal;
                if (ResType0.DataValue.enumDataType == nDT_Long) dRes =
    ResType0.DataValue.lVal;
```

```
            ResType0.DataValue.dVal = dVal;
            ResType0.DataValue.enumDataType = nDT_Real8;
        } /* Must convert to Real8 */
        if (ResType1.DataValue.enumDataType != nDT_Real8)
        { /* Must convert to Real8 */
            double dVal;
            /**/
            dVal = 0.;
            if (ResType1.DataValue.enumDataType == nDT_Integer) dRes =
ResType1.DataValue.lVal;
            if (ResType1.DataValue.enumDataType == nDT_Long) dRes =
ResType1.DataValue.lVal;
            ResType1.DataValue.dVal = dVal;
            ResType1.DataValue.enumDataType = nDT_Real8;
        } /* Must convert to Real8 */
        dArg0 = ResType0.DataValue.dVal;
        dArg1 = ResType1.DataValue.dVal;
        /**/
        if (((nOpTokenType == nTT_BConjAnd) || (nOpTokenType == nTT_BConjOr)) ||
          ((nOpTokenType >= nTT_LT) && (nOpTokenType <= nTT_GT)))
        { /* Boolean Conjunction returns a boolean */
            int nArg0, nArg1;
            /**/
            nOpDataType = nDT_Bool;
            nArg0 = (dArg0 == 0.)? 0: 1;
            nArg1 = (dArg1 == 0.)? 0: 1;
            dRes = 0.;
            if ((nOpTokenType == nTT_BConjAnd) || (nOpTokenType == nTT_BConjOr))
            { /* Conjunctive Operator */
                if (nOpTokenType == nTT_BConjAnd)
                { /* Both must be true */
                    if (nArg0 & nArg1) dRes = 1.;
                } /* Both must be true */
                else
                { /* Either can be true */
                    if (nArg0 | nArg1) dRes = 1.;
                } /* Either can be true */
            } /* Conjunctive Operator */
            else
            { /* Comparison of the arguments */
                switch (nOpTokenType)
                { /* switch (nOpTokenType) */
                case nTT_LT: if (dArg0 < dArg1) dRes = 1.; break;
                case nTT_LE: if (dArg0 <= dArg1) dRes = 1.; break;
                case nTT_EQ: if (dArg0 == dArg1) dRes = 1.; break;
                case nTT_GE: if (dArg0 >= dArg1) dRes = 1.; break;
                case nTT_GT: if (dArg0 > dArg1) dRes = 1.; break;
                } /* switch (nOpTokenType) */
            } /* Comparison of the arguments */
        } /* Boolean Conjunction returns a boolean */
        else if ((nOpTokenType == nTT_Plus) || (nOpTokenType == nTT_Minus) ||
          (nOpTokenType == nTT_StarSingle) || (nOpTokenType == nTT_DivSingle))
        { /* Numerical operation */
            nOpDataType = nDT_Real8;
            switch (nOpTokenType)
            { /* switch (nOpTokenType) */
            case nTT_Plus: dRes = dArg0 + dArg1; break;
            case nTT_Minus: dRes = dArg0 - dArg1; break;
            case nTT_StarSingle: dRes = dArg0 * dArg1; break;
            case nTT_DivSingle:
                if (dArg1 == 0.) dRes = 1.;
                else dRes = dArg0 / dArg1;
                break;
            } /* switch (nOpTokenType) */
        } /* Numerical operation */
        else
        { /* Invalid Token Type for Operator */
            printf("\n!!! Invalid Token type for operatio- value: %i, desc: %s
!!!\n",
              nOpTokenType, pszTokenDescription(nOpTokenType));
            exit(-1);
            } /* Invalid Token Type for Operator */
            /*
                Set up return values
            */
            ResType.nIsFixed = 1;
            ResType.nResErr = 0;
            ResType.DataValue.dVal = dRes;
            ResType.nDataType = nDT_Real8;
            if (nOpFlag0 == 1)
            { /* Update operator with result of this calculation */
                pThisRE->SEOperator.DataValue =  ResType;
                pThisRE->SEOperator.DataValue.DataValue = ResType.DataValue;
                pThisRE->SEOperator.nValType = 7;
            } /* Update operator with result of this calculation */
            /**/
        } /* earlier results OK */
    } /* Determine Data Types and Values of Arguments */
    /**/
    return ResType;
    /**/
} /* procResultValue- Returns result data type */
```

Appendix G

Procedure “procCreateFPUCode”

```
_proc double procCreateFPUCode()
{ /* procCreateFPUCode- Creates machine code for calculation */
    double dRes;
    /**/
    typedef struct tagWorkEle_struct /* WorkEle_struct */
    { /* WorkEle_struct */
        CalcChain_struct *pCC, *pCCArgs[2], *pCCPapa;
        struct tagWorkEle_struct *pArgs[2], *pPapa;
        int nPapaIx;
        int nStatus; /* 0 not done, 1 doing, 2 done */
        int nResIx;
        int nArgsDn;
    } WorkEle_struct;
    /**/
    int nWorkElesCount;
    WorkEle_struct *pWorkEles, *pWorkEle;
    WorkEle_struct *pDoing, *pDone, *pPapa, *pTop, *pValue;
    int nWorkElesIx, nWorkElesJx;
    /**/
    FILE *fhFPUCode;
    char *pszFPUCode = "FPUCode.tmp";
    char szAsmText[100];
    /**/
    int nThisResIx;
    int nArgIx;
    CalcChain_struct *pThisCC, *pArg0CC, *pArg1CC, *pPrevCC, *pNextCC;
    int nArgsType;
    double *pdResults;
    int nRUElesCount;
    unsigned char *puchResultsUsed;
    long double ldFAcc;
    char *pszCodeIndent = "            ";
    CalcChain_struct **pCCsSorted, **pCCSorted;
    int nNextArgIx, nPrevArgIx;
    int nThisIsSibling, nNextIsSibling;
    int nThisIsParent, nNextIsParent;
    int nNextIsPrevRes, nThisIsPrevRes;
    int nResStoreCount;
    /**/
    int nResIx;
    /**/
    dRes = 0.;
    /**/
    nWorkElesCount = 0;
    for (pThisCC = m_pFirstCC; pThisCC != NULL; pThisCC = pThisCC->pNext)
    { /* Count this element */
        nWorkElesCount++;
    } /* Count this element */
    printf(" Number of FPU Work Elements: %i\n", nWorkElesCount);
    if (nWorkElesCount <= 0) return 0;
    /*
    ****************************************************************************
    *
    *   Create work array
    *
    ****************************************************************************
    */
    pWorkEles = MemGet(nWorkElesCount * sizeof(WorkEle_struct));
    /**/
    nRUElesCount = nWorkElesCount + 1;
    puchResultsUsed = MemGet(nRUElesCount * sizeof(unsigned char));
    memset(puchResultsUsed, 0, nRUElesCount * sizeof(unsigned char));
    /**/
    if ((fhFPUCode = fopen(pszFPUCode, "w+")) == NULL)
    { /* Could not open Calculation Assembly file */
        printf("\nCould not open Calculation Assembly Temporary file\n");
        exit(-1);
    } /* Could not open Calculation Assembly file */
    /**/
    pTop = NULL; /* Top CC/Work Element */
    for (pWorkEle = pWorkEles, pThisCC = m_pFirstCC; pThisCC != NULL; pThisCC = pThisCC-
    >pNext, pWorkEle++)
    { /* Create Sort Element for this CC element */
        pWorkEle->pCC = pThisCC;
        if (pThisCC == m_pTopCC) pTop = pWorkEle;
        pWorkEle->nStatus = 0;
        pWorkEle->nArgsDn = 0;
        pWorkEle->nResIx = -1;
        pWorkEle->pArgs[0] = NULL;
        pWorkEle->pArgs[1] = NULL;
        pWorkEle->pCCArgs[0] = pThisCC->pPrevCCs[0];
        pWorkEle->pCCArgs[1] = pThisCC->pPrevCCs[1];
        pWorkEle->pCCPapa = pThisCC->pParentCC;
        pWorkEle->pPapa = NULL;
        pWorkEle->nPapaIx = pThisCC->nParentCCArgIx;
    } /* Create Sort Element for this CC element */
    /*
    ****************************************************************************
    *
    *   Populate work array
    *
    ****************************************************************************
    */
    for (nWorkElesIx = 0; nWorkElesIx < nWorkElesCount; nWorkElesIx++)
    { /* find associated elements for this eleement */
        CalcChain_struct *pArgsCC;
        WorkEle_struct *pThis, *pTest;
        int nArgsIx;
        /**/
        pThis = pWorkEles + nWorkElesIx;
        for (nArgsIx = 0; nArgsIx < 2; nArgsIx++)
        { /* find references to this argument */
            if ((pArgsCC = pThis->pCCArgs[nArgsIx]) != NULL)
            { /* This element takes at least one argument element */
                for (nWorkElesJx = 0; nWorkElesJx < nWorkElesCount; nWorkElesJx++)
                { /* Check this Work Element */
                    pTest = pWorkEles + nWorkElesJx;
                    if (pTest->pCC == pArgsCC)
                    { /* This argument is operand to another */
                        pTest->pPapa = pThis;
                        pTest->nPapaIx = nArgsIx;
                        pThis->pArgs[nArgsIx] = pTest;
                        break;
                    } /* This argument is operand to another */
                } /* Check this Work Element */
            } /* This element takes at least one argument element */
        } /* find references to this argument */
    } /* find associated elements for this eleement */
    /*
    ****************************************************************************
    *
    *   Create output order of calculaton
    *
    ****************************************************************************
    */
    pCCsSorted = MemGet(sizeof(pCCSorted) * (nWorkElesCount));
    pDone = NULL;
    pDoing = NULL;
    pValue = NULL;
    nResIx = 0;
    for (;;)
    { /* Treat a CC */
        /*
            Ended a CC element
        */
        if (pDone)
        { /* Just finished a CC elemet */
            int nArgIx;
            CalcChain_struct *pArgCC;
            /**/
            /*printf("Done: %x\n", pDone);*/
            pDone->pCC->nResIndex = nResIx;
            *(pCCsSorted + nResIx++) = pDone->pCC;
            pDone->nStatus = 2;
            if ((pPapa = pDone->pPapa) == NULL)
            { /* All finished */
                break;
            } /* All finished */
            else
            { /* This has a parent */
                nArgIx = 1- pDone->pCC->nParentCCArgIx;
                pDone = NULL;
                if (++pPapa->nArgsDn == 2)
                { /* Finishes off papa */
                    pDone = pPapa;
                } /* Finishes off papa */
                else
                { /* Has one more argument to do */
                    if ((pTop = pPapa->pArgs[nArgIx]) == NULL)
                    { /* This is a value element */
                        pValue = pPapa;
                    } /* This is a value element */
                } /* Has one more argument to do */
            } /* This has a parent */
        } /* Just finished a CC elemet */
        /*
            Find bottom of next argument
        */
        if (pTop)
        { /* Look for longest unused CC */
            WorkEle_struct *pThis, *pTest;
            int nMaxDepth, nThisDepth, nArgSide;
            /**/
            nMaxDepth = -1; pDoing = NULL;
            for (nWorkElesIx = 0, pThis = pWorkEles; nWorkElesIx < nWorkElesCount;
    nWorkElesIx++, pThis++)
            { /* This may be already in use */
                if (pThis->nStatus == 0)
                { /* This is a possibility */
                    nThisDepth = 0;
                    for (pTest = pThis; pTest != NULL; pTest = pTest->pPapa)
                    { /* Check agains top element */
                        if (pTest != pTop)
                        { /* Not it, must go up an element */
                            nThisDepth++;
                        } /* Not it, must go up an element */
                        else
                        { /* This is it */
                            if ((nMaxDepth < 0) || ((nMaxDepth >= 0) && (nThisDepth >
    nMaxDepth)))
                            { /* New distance to top */
                                nMaxDepth = nThisDepth;
                                pDoing = pThis;
                                nArgSide = pThis->nPapaIx;
                            } /* New distance to top */
                            if ((nArgSide != 0) && (nThisDepth == nMaxDepth))
                            { /* If this is the left side, keep it */
                                if (pThis->nPapaIx == 0)
                                { /* This is it, so far */
                                    pDoing = pThis;
                                    nArgSide = 0;
                                } /* This is it, so far */
                            } /* If this is the left side, keep it */
                        } /* This is it */
                    } /* Check agains top element */
                } /* This is a possibility */
            } /* This may be already in use */
            /**/
            if (pDoing == NULL) break;
            pTop = NULL;
            /**/
        } /* Look for longest unused CC */
        /*
            Generate code for a Value element
        */
        if (pValue)
        { /* Generate code for a Value element */
            pDone = pValue;
            printf("Value: %x\n", pValue);
            pValue = NULL;
        } /* Generate code for a Value element */
        /*
            Generate code for a CC element
        */
        if (pDoing)
        { /* Generate code for a CC element */
            pDoing->nStatus = 1;
            pDone = pDoing;
            pDoing = NULL;
        } /* Generate code for a CC element */
    } /* Treat a CC */
    /*
    ****************************************************************************
    *
    *   Generate FPU Code
    *
    ****************************************************************************
    */
    pdResults = NULL;
    procFPUOperandValue(-1, 0);
    nResStoreCount = 0;
    /**/
    pdResults = MemGet(sizeof(double) * (nWorkElesCount + 1));
    memset(pdResults, 0, sizeof(double) * (nWorkElesCount + 1));
    /**/
    nNextIsSibling = 0;
    nNextIsParent = 0;
    nNextIsPrevRes = 0;
    /**/
    nNextArgIx = -1;
    for (nThisResIx = 0; nThisResIx < nResIx; nThisResIx++)
    { /* Create calculation chain for this Chain */
        ReduxElement_struct *pLeafRE, *pTestRE;
        /*
            Look for the correct calculation chain
        */
        nPrevArgIx = nNextArgIx;
        /**/
        pPrevCC = pThisCC;
        pThisCC = *(pCCsSorted + nThisResIx);
        /**/
        nThisIsPrevRes = nNextIsPrevRes;
        nNextIsPrevRes = 0;
        nThisIsSibling = nNextIsSibling;
        nNextIsSibling = 0;
        pNextCC = NULL;
        if ((nThisResIx + 1) < nResIx)
        { /* Must check next CC element */
            pNextCC = *(pCCsSorted + nThisResIx + 1);
            if (pNextCC->pParentCC == pThisCC->pParentCC) nNextIsSibling = 1;
            if ((pArg0CC = pNextCC->pPrevCCs[0]) != NULL)
            { /* Next CC element has a CC element for left arg */
                if (pArg0CC->nResIndex == nThisResIx) nNextIsPrevRes = 1;
            } /* Next CC element has a CC element for left arg */
        } /* Must check next CC element */
        nThisIsParent = nNextIsParent;
        nNextIsParent = nThisIsSibling;
        if ((nThisIsSibling) || (nNextIsSibling))
```

```
{ /* Must treat normally */
    nNextIsPrevRes = 0;
} /* Must treat normally */
/**/
pTestRE = (pLeafRE = pThisCC->pLeafRE);
pArg0CC = NULL; pArg1CC = NULL;
if (pTestRE->SEOperands[0].pReduxElement) pArg0CC = pThisCC->pPrevCCs[0];
if (pTestRE->SEOperands[1].pReduxElement) pArg1CC = pThisCC->pPrevCCs[1];
/**/
for (;;)
{ /* Generate instruction for this reduction (pTestRE) */
    TokenTypes_Enum nOpTT;
    StackElement_struct *pArg1SE, *pArg0SE;
    int nArgsType;
    char szOp[15], szSwapOp[15];
    double dArg2;
    /**/
    nOpTT = pTestRE->SEOperator.TokenData.nTokenType;
    pArg0SE = &pTestRE->SEOperands[0];
    pArg1SE = &pTestRE->SEOperands[1];
    /**/
    nArgsType = (pArg0SE->pReduxElement? 1: 0) + (pArg1SE->pReduxElement? 2: 0);
    /*
        Determine the operation
    */
    switch (nOpTT)
    { /* Determine operation */
    case nTT_Plus:
        strcpy(szSwapOp, "fadd");
        strcpy(szOp, "fadd"); break;
    case nTT_Minus:
        strcpy(szSwapOp, "fsubr");
        strcpy(szOp, "fsub"); break;
    case nTT_StarSingle:
        strcpy(szSwapOp, "fmul");
        strcpy(szOp, "fmul"); break;
    case nTT_DivSingle:
        strcpy(szSwapOp, "fdivr");
        strcpy(szOp, "fdiv"); break;
    default:
        strcpy(szSwapOp, "NoOp");
        strcpy(szOp, "(NoOp)");
    } /* Determine operation */
    szAsmText[0] = '\0';
    /*
        Determine the calculation
    */
    if (pTestRE == pLeafRE)
    { /* First instruction in calculation chain */
        /**/
        if (nThisIsParent)
        { /* This should perform only the operation */
            if (pPrevCC->nParentCCArgIx == 0)
            { /* Must swap operations */
                strcpy(szOp, szSwapOp);
            } /* Must swap operations */
            sprintf(szAsmText, "%s;   /* Parent of top two of stack */", szOp);
            fprintf(fhFPUCode, "%s%s\n", pszCodeIndent, szAsmText);
        } /* This should perform only the operation */
        else if (nThisIsPrevRes)
        { /* This should perform only the operation */
            if (pPrevCC->nParentCCArgIx == 0)
            { /* Must swap operations */
                strcpy(szOp, szSwapOp);
            } /* Must swap operations */
            /*
                Right side of operation
            */
            szAsmText[0] = '\0';
            if (pArg1CC)
            { /* Right operand is a previous CC */
```

```
                    sprintf(szAsmText, "%s  dRes%i; /* Result- nThisIsPrevRes: %i
*/", szOp, pArg1CC->nResIndex, nThisIsPrevRes);
                } /* Right operand is a previous CC */
                else
                { /* Right operand is a data value */
                    sprintf(szAsmText, "%s  d%i; /* %20.15f- nThisIsPrevRes: %i */",
szOp,
                       procFPUOperandValue(0, pArg1SE->DataValue.DataValue.dVal),
                       pArg1SE->DataValue.DataValue.dVal, nThisIsPrevRes);
                } /* Right operand is a data value */
                if (szAsmText[0] != '\0') fprintf(fhFPUCode, "%s%s\n", pszCodeIndent,
szAsmText);
            } /* This should perform only the operation */
            else
            { /* Must do something with this operand */
                /*
                    Left side of operation
                */
                if (pArg0CC)
                { /* Left operand is a previous CC */
                    if (nPrevArgIx != 0)
                      sprintf(szAsmText, "fld  dRes%i; /* Result- nThisIsPrevRes: %i,
nNextIsPrevRes: %i */", pArg0CC->nResIndex, nThisIsPrevRes, nNextIsPrevRes);
                } /* Left operand is a previous CC */
                else
                { /* Left operand is a data value */
                    sprintf(szAsmText, "fld  d%i; /* %20.15f */",
                       procFPUOperandValue(0, pArg0SE->DataValue.DataValue.dVal),
                       pArg0SE->DataValue.DataValue.dVal);
                } /* Left operand is a data value */
                fprintf(fhFPUCode, "%s%s\n", pszCodeIndent, szAsmText);
                /*
                    Right side of operation
                */
                szAsmText[0] = '\0';
                if (pArg1CC)
                { /* Right operand is a previous CC */
                    sprintf(szAsmText, "%s  dRes%i; /* Result */", szOp, pArg1CC-
>nResIndex);
                } /* Right operand is a previous CC */
                else
                { /* Right operand is a data value */
                    sprintf(szAsmText, "%s  d%i; /* %20.15f */", szOp,
                       procFPUOperandValue(0, pArg1SE->DataValue.DataValue.dVal),
                       pArg1SE->DataValue.DataValue.dVal);
                } /* Right operand is a data value */
                fprintf(fhFPUCode, "%s%s\n", pszCodeIndent, szAsmText);
            } /* Must do something with this operand */
        } /* First instruction in calculation chain */
        else
        { /* Subsequent instruction in calculation chain */
            /*
                Get arguments
            */
            dArg2 = 0.;
            switch (nArgsType)
            { /* Process argument type for this reduction */
            case 0: /* Both operands */
                strcpy(szAsmText, "(Both operands)");
                break;
            case 1: /* Reduction, operand */
                dArg2 = pArg1SE->DataValue.DataValue.dVal;
                sprintf(szAsmText, "%s d%i; /* %20.15f */", szOp,
procFPUOperandValue(0, dArg2), dArg2);
                break;
            case 2: /* operand, reduction */
                dArg2 = pArg0SE->DataValue.DataValue.dVal;
                sprintf(szAsmText, "%s d%i; /* %20.15f */", szSwapOp,
procFPUOperandValue(0, dArg2), dArg2);
                break;
            case 3: /* reduction, reduction */
```

```
                strcpy(szAsmText, "(Both reductions)");
            } /* Process argument type for this reduction */
            fprintf(fhFPUCode, "%s%s\n", pszCodeIndent, szAsmText);
        } /* Subsequent instruction in calculation chain */
        /*
            Perform operation on contents of facc
        */
        /*procFPUOperandValue(0, dArg2);*/
        /*
            This Reduction handled
        */
        if (pTestRE == pThisCC->pNodeRE) break;
        pTestRE = pTestRE->pREParent;
        if (pTestRE == NULL) break;
        /**/
    } /* Generate instruction for this reduction (pTestRE) */
    if ((nNextIsSibling == 0) && (nThisIsSibling == 0) && (nNextIsPrevRes == 0))
    { /* Next CC element has another parent */
        fprintf(fhFPUCode, "%sfstp dRes%i; /* Result- nThisIsPrevRes: %i,
nNextIsPrevRes: %i  */\n",
         pszCodeIndent, nThisResIx, nThisIsPrevRes, nNextIsPrevRes);
        *(puchResultsUsed + nThisResIx) = 1;
        nResStoreCount++;
    } /* Next CC element has another parent */
    /*
        This Calculation Chain code created
    */
} /* Create calculation chain for this Chain */
fclose(fhFPUCode);
/*
*****************************************************************************
*
*   Create Floating-Point Procedure
*
*****************************************************************************
*/
{ /* Copy code to file */
    #define nMaxBlockSize 4096
    int nGot;
    char *psBlock;
    int nLastRes;
    /**/
    if ((fhFPUCode = fopen(pszFPUCode, "r")) == NULL)
    { /* Could not open Calculation Assembly file */
        printf("\nCould not open Calculation Assembly Temporary file\n");
        exit(-1);
    } /* Could not open Calculation Assembly file */
    psBlock = MemGet(nMaxBlockSize);
    fprintf(m_fhCalcAsm, "int proc_%i()\n{ /* proc_%i */\n", m_nC, m_nC);
    /**/
    procFPUOperandValue(-2, 0);
    nLastRes = 0;
    for (nArgIx = 0; nArgIx < m_nCCLastResIx + 1; nArgIx++)
    { /* Define storage location for chain result */
        if (*(puchResultsUsed + nArgIx) == 1)
        { /* This is needed results storage (a truncation) */
            fprintf(m_fhCalcAsm, "    IEEE754Real8_struct dRes%i;\n", nArgIx);
            nLastRes = nArgIx;
        } /* This is needed results storage (a truncation) */
    } /* Define storage location for chain result */
    /**/
    fprintf(m_fhCalcAsm, "    char *pszBits;\n\     { /* add to Area so far */\n");
    fprintf(m_fhCalcAsm, "    asm\n        { /* Do FPU stuff */\n");
    /**/
    for (;feof(fhFPUCode) == 0;)
    { /* Read and write an FPU operation */
        if (fgets(szAsmText, sizeof(szAsmText), fhFPUCode) <= 0) break;
        fprintf(m_fhCalcAsm, "%s", szAsmText);
    } /* Read and write an FPU operation */
    /**/
    fprintf(m_fhCalcAsm, "        } /* Do FPU stuff */\n");
```

```
        fprintf(m_fhCalcAsm, "    } /* add to Area so far */\n");
        /**/
        fprintf(m_fhCalcAsm, "    printf(\"\\nCalculation index: %i\\n\");\n", m_nC);
        fprintf(m_fhCalcAsm, "    pszBits = procIEE754DblToBin(dRes%i.dVal);\n",
    nLastRes);
        fprintf(m_fhCalcAsm, "    printf(\"    Res: %%25.20g, Binary: %%s\\n\",
    dRes%i.dVal, pszBits);\n", nLastRes);
        fprintf(m_fhCalcAsm, "    free(pszBits);\n");
        fprintf(m_fhCalcAsm, "    return 0;\n");
        fprintf(m_fhCalcAsm, "    /* Number of Store Operations: %i */\n",
    nResStoreCount);
        fprintf(m_fhCalcAsm, "} /* proc_%i */\n", m_nC);
        /**/
        free(psBlock);
        fclose(fhFPUCode);
        /**/
    } /* Copy code to file */
    /*
    ****************************************************************************
    *
    *   Release resources
    *
    ****************************************************************************
    */
    free(puchResultsUsed);
    free(pWorkEles);
    if (pdResults) free(pdResults);
    free(pCCsSorted);
    /**/
    procFPUOperandValue(-1, 0);
    /**/
    return dRes;
    /**/
} /* procCreateFPUCode- Creates machine code for calculation */
```

Appendix H

Procedure “procFPUOperandValue”

```
_proc int procFPUOperandValue(int nOp, double dVal)
{ /* procFPUOperandValue- handles values used as FPU operands */
    /**/
    typedef struct tagDataOperand_struct
    { /* DataOperand_struct */
        struct tagDataOperand_struct *pPrev, *pNext;
        IEEE754Real8_struct Value;
    } DataOperand_struct;
    /**/
    static DataOperand_struct *pFirstDO, *pLastDO;
    static int nDOsCount;
    IEEE754Real8_struct TestVal;
    int nDOsIx, nTestIx;
    DataOperand_struct *pThisDO, *pTestDO, *pNewDO;
    /**/
    nDOsIx = 0;
    if (nOp == 1)
    { /* Initialize Values */
        pFirstDO = NULL; pLastDO = NULL; nDOsCount = 0;
    } /* Initialize Values */
    else if (nOp == -1)
    { /* Close Values */
        for (; pFirstDO !=  NULL;)
        { /* Release this value */
            pTestDO = pFirstDO;
            LinkRemove(pTestDO, pFirstDO, pLastDO);
            free(pTestDO);
            nDOsCount--;
        } /* Release this value */
    } /* Close Values */
    else if (nOp == 0)
    { /* Find/Insert a value */
        TestVal.dVal = dVal;
        nDOsIx = -1;
        for (pTestDO = pFirstDO, nTestIx = 0; pTestDO != NULL; nTestIx++, pTestDO =
    pTestDO->pNext)
        { /* Test if this is the value */
            if (pTestDO->Value.lVals[0] != TestVal.lVals[0]) continue;
            if (pTestDO->Value.lVals[1] != TestVal.lVals[1]) continue;
            nDOsIx = nTestIx;
            break;
        } /* Test if this is the value */
        if (nDOsIx == -1)
        { /* Must add another value */
            pNewDO = MemGet(sizeof(DataOperand_struct));
            LinkAppendNew(pNewDO, pFirstDO, pLastDO);
            nDOsIx = nDOsCount++;
            pNewDO->Value.dVal = dVal;
        } /* Must add another value */
    } /* Find/Insert a value */
    else if (nOp == -2)
    { /* Output Variables to calculation file */
        for (pTestDO = pFirstDO, nTestIx = 0; pTestDO != NULL; nTestIx++, pTestDO =
    pTestDO->pNext)
        { /* Test if this is the value */
            fprintf(m_fhCalcAsm, "    IEEE754Real8_struct d%i = {0x%lxl, 0x%lxl}; /*
    %20.15f */\n",
              nTestIx, pTestDO->Value.lVals[0], pTestDO->Value.lVals[1], pTestDO-
    >Value.dVal);
        } /* Test if this is the value */
    } /* Output Variables to calculation file */
    /**/
    return nDOsIx;
    /**/
} /* procFPUOperandValue- handles values used as FPU operands */
```

Appendix I

Program “FPU.c”

```
#include <stdlib.h>
#include <stdio.h>
#include <malloc.h>
#include <memory.h>
#include <string.h>
/*
***********************************************************************************

    Section 1- IEEE 754 structures

************************************************************************************
*/
typedef union /* IEEE754Real8_struct */
{ /* IEEE754Real8_struct */
    long lVals[2];
    double dVal;
} IEEE754Real8_struct;
/**/
char *procIEE754DblToBin(double dVal);
/*
***********************************************************************************

    Section 2- Other stuff

************************************************************************************
*/
#define TRUE -1
#define FALSE 0
/*
***********************************************************************************

    Section 3- Support procedures

************************************************************************************
*/
char *procIEE754DblToBin(double dVal)
{ /* procIEE754DblToBin- Convert binary IEEE 754 double to decimal output */
    /*FptDouble_struct fpdTest;*/
    IEEE754Real8_struct fpdTest;
    char *pszOut;
    int nBiaseExp;
    int nPwr2;
    int nSignBit;
    int nCntIntBits, nCntFracBits;
    int nCntIntLost, nCntFracLost;
    int nPwr2Low, nPwr2High;
    char *pszIntVal, *pszFracVal;
    char *szRes;
    /**/
    int nIntPwr2, nFracPwr2;
    long lAndMasks[] =
    {
        0x00000001l, 0x00000002l, 0x00000004l, 0x00000008l,
        0x00000010l, 0x00000020l, 0x00000040l, 0x00000080l,
        0x00000100l, 0x00000200l, 0x00000400l, 0x00000800l,
        0x00001000l, 0x00002000l, 0x00004000l, 0x00008000l,
        0x00010000l, 0x00020000l, 0x00040000l, 0x00080000l,
        0x00100000l, 0x00200000l, 0x00400000l, 0x00800000l,
        0x01000000l, 0x02000000l, 0x04000000l, 0x08000000l,
        0x10000000l, 0x20000000l, 0x40000000l, 0x80000000l
    };
    pszOut = NULL;
    /**/
    fpdTest.dVal = dVal;
```

```
/**/
nBiaseExp = (fpdTest.lVals[1] >> 20) & 0x7ff;
nSignBit = (fpdTest.lVals[1] >> 31) & 0x1l;
nPwr2 = nBiaseExp - 1023;
/**/
nCntIntLost = 0; nCntFracLost = 0;
nIntPwr2 = 0; nFracPwr2 = 0;
if (nBiaseExp == 0)
{ /* maybe zero */
    nCntIntBits = 0;
    nCntFracBits = 0;
} /* maybe zero */
if (nPwr2 >= 0)
{ /* Greater than one */
    nCntIntBits = nPwr2 + 1;
    if ((nCntFracBits = 52 - nPwr2) <= 0) nCntIntLost = 52 - nPwr2;
    if (nCntFracBits < 0) nCntFracBits = 0;
    if ((nCntFracLost = nPwr2) > 52) nCntFracLost = 52;
    nIntPwr2 = nCntIntLost;
    nFracPwr2 = -1;
} /* Greater than one */
else
{ /* Less than one */
    nCntIntBits = 0;
    nCntFracBits = 53;
    nFracPwr2 = nBiaseExp;
} /* Less than one */
nPwr2High = nPwr2;
nPwr2Low = nPwr2 - 52;
{ /* Create integer and fractional parts */
    /*
        This moves the Least Significant bits of a Long Value
    */
    int nFptBitsDo, nFptBitsDn, nNewFpt, nThisFpt, nFptBitsLeft, nFptRShift;
    long lFptVal;
    int nTgtBitsDo, nTgtBitsDn, nNewTgt, nThisTgt, nTgtBitsLeft;
    int nMoveDo;
    long lMoveVal, lNewVal;
    /*
        Source
    */
    nFptBitsDo = 1;
    nThisFpt = 0;
    nFptBitsDn = 0;
    lFptVal = 1l;
    /*
        Target
    */
    nNewTgt = 1;
    nTgtBitsDo = 1;
    nThisTgt = 0;
    nTgtBitsDn = 0;
    /*
        Output
    */
    pszIntVal = NULL;
    pszFracVal = NULL;
    /**/
    if (nBiaseExp != 0)
    { /* Alternate approach */
        int nNewSrc, nThisSrc;
        int nSrcCntDo, nSrcCntDn, nSrcCntXfr, nSrcCntWidth;
        int lSrcVal;
        int nNewTgt, nThisTgt;
        int nTgtCntDo, nTgtCntDn, nTgtCntXfr;
        int nMovCntDo, nMovCntDn;
        /**/
        nNewSrc = 1; nNewTgt = 1;
        szRes = NULL;
        for (;;)
        { /* Move a block of bits */
```

```
            /*
                Initialize source
            */
            if (nNewSrc != 0)
            { /*  New source row */
                nSrcCntDn = 0;
                switch (nNewSrc)
                {
                case 1:
                    nSrcCntWidth = 1; nSrcCntDo = 1; lSrcVal = 1l; break;
                case 2:
                    nSrcCntWidth = 32; nSrcCntDo = 20; lSrcVal = fpdTest.lVals[1] &
0xfffff; break;
                case 3:
                    nSrcCntWidth = 32; nSrcCntDo = 32; lSrcVal = fpdTest.lVals[0];
break;
                }
                nThisSrc = nNewSrc;
                nNewSrc = 0;
                nSrcCntDn = 0;
            } /*  New source row */
            nSrcCntXfr = nSrcCntDo - nSrcCntDn;
            /*
                Initialize target
            */
            if (nNewTgt != 0)
            { /*  New source row */
                switch (nNewTgt)
                {
                case 1: /* Integer side */
                    nTgtCntDo = nCntIntBits;
                    break;
                case 2:
                    nTgtCntDo = nCntFracBits;
                    break;
                }
                nTgtCntDn = 0;
                szRes = NULL;
                if (nTgtCntDo > 0)
                {
                    szRes = malloc(nTgtCntDo + 1);
                    memset(szRes, '\0', nTgtCntDo + 1);
                }
                if (nNewTgt == 1) pszIntVal = szRes;
                if (nNewTgt == 2) pszFracVal = szRes;
                nThisTgt = nNewTgt;
                nNewTgt = 0;
            } /*  New source row */
            nTgtCntXfr = nTgtCntDo - nTgtCntDn;
            /*
                Perform move
            */
            nMovCntDo = (nSrcCntXfr <= nTgtCntXfr)? nSrcCntXfr: nTgtCntXfr;
            if (nMovCntDo > 0)
            { /* Perform move */
                int nSrcShift, nMoveIx;
                long int lMoveVal, lBitVal;
                char chBitVal;
                /**/
                nSrcShift = nSrcCntXfr - nMovCntDo;
                lMoveVal = lSrcVal;
                if (nSrcShift > 0) lMoveVal >>= nSrcShift;
                for (nMoveIx = nMovCntDo; nMoveIx > 0;)
                {
                    if (nTgtCntDn < nTgtCntDo)
                    {
                        nMoveIx--;
                        lBitVal = (lMoveVal & lAndMasks[nMoveIx]);
                        chBitVal =(lBitVal == 0)? '0': '1';
                        *(szRes + nTgtCntDn++) = chBitVal;
                    }
                    }
                    nSrcCntDn += nMovCntDo;
                } /* Perform move */
                /*
                    Check for next pass
                */
                if (nSrcCntDn == nSrcCntDo) nNewSrc = nThisSrc + 1;
                if (nTgtCntDn == nTgtCntDo) nNewTgt = nThisTgt + 1;
                if ((nNewSrc == 4) || (nNewTgt == 3)) break;
            } /* Move a block of bits */
        } /* Alternate approach */
        /**/
        { /* Create Returned result */
            int nLenOut, nIx;
            int nLenSign, nLenInt,  nLenFrac, nLenLead;
            /**/
            nLenInt = (pszIntVal? strlen(pszIntVal): 0);
            nLenFrac = (pszFracVal? strlen(pszFracVal): 0);
            nLenLead = 0;
            nLenSign = 1;
            if ((nBiaseExp != 0) && (nPwr2 < 0)) nLenLead = -1 - nPwr2;
            /*
                Create Output String
            */
            nLenOut = 1 + nLenSign + (nLenInt? nLenInt: 1) + nLenLead + nLenFrac + 1;
            /**/
            pszOut = malloc(nLenOut);
            memset(pszOut, '\0', nLenOut);
            strcpy(pszOut, (nSignBit? "-": "+"));
            if (nLenInt == 0) strcat(pszOut, "0");
            else strcat(pszOut, pszIntVal);
            strcat(pszOut, ".");
            for (nIx = 0; nIx < nLenLead; nIx++) strcat(pszOut, "0");
            if (pszFracVal) strcat(pszOut, pszFracVal);
            /**/
        } /* Create Returned result */
        /**/
        if (pszIntVal) free(pszIntVal);
        if (pszFracVal) free(pszFracVal);
    } /* Create integer and fractional parts */
    /**/
    return pszOut;
    /**/
} /* procIEE754DblToBin- Convert binary IEEE 754 double to decimal output */
/*
    Floating-Point Processor Application
*/
int proc_2(void);
int proc_3(void);
int proc_4(void);
      .
      .
      .
int proc_100(void);
int proc_101(void);
/**/
/* Contents of "CHProcEng.FPU" go here */
/**/
int main()
{ /* main- primary entry point */
    proc_2();
    proc_3();
    proc_4();
      .
      .
      .
    proc_100();
    proc_101();
    /**/
    return 0;
} /* main- primary entry point */
```

Appendix AA-1- CHFort Object Code of Test Case One

```
<<variablesvalues>>
Name: dYSum
Value:
IsHistory: 1
IsFixed: 0
DataType: 5
nDimsCount: 0
Name: dXMid
Value:
IsHistory: 1
IsFixed: 0
DataType: 5
nDimsCount: 0
Name: dX0
Value:
IsHistory: 0
IsFixed: 0
DataType: 5
nDimsCount: 0
Name: dSpan
Value:
IsHistory: 0
IsFixed: 0
DataType: 5
nDimsCount: 0
Name: dSpanIncs
Value:
IsHistory: 0
IsFixed: 0
DataType: 5
nDimsCount: 0
Name: dSpanDelta
Value:
IsHistory: 0
IsFixed: 0
DataType: 5
nDimsCount: 0
<<ProgramCode>>
Label StartProg
BeginCode dX0
push Number 0
EndCode
BeginCode dSpan
push Number 1
EndCode
BeginCode dSpanIncs
push Number 50
EndCode
BeginCode dSpanDelta
  push String dSpan
  push String dSpanIncs
  DivSingle
EndCode
BeginCode dXMid0
  push String dX0
  push String dSpanDelta
  push Number 2
  DivSingle
  Plus
EndCode
BeginCode dXMid
push String dXMid0
EndCode
BeginCode dYSum
push Number 0
EndCode
Label 100
BeginBoolCode dIf_Bool_1_0
```

```
  push String dXMid
  push String dSpan
  Minus
EndCode
GoToCond 999 GreaterThan dIf_Bool_1_0
BeginCode dYSum
  push String dYSum
  push String dXmid
  Plus
EndCode
BeginCode dXMid
  push String dXMid
  push String dSpanDelta
  Plus
EndCode
GoTo 100
Label 999
Label EndProg
```

Appendix AA-2- Comparative Results of Test Case One

```
Iteration Number: 1
           dReg                       dRes                       dFPU
 0.0100000000000000000208    0.0100000000000000000208    0.0100000000000000000208
dReg: +0.000000101000111101011100001010001111010111000010100011110 11
dRes: +0.00000010100011110101110000101000111101011100001010001111011
dFPU: +0.00000010100011110101110000101000111101011100001010001111011
Original REs Count: 1, Computational REs Count: 1, Saved REs Count: 1

Iteration Number: 2
           dReg                       dRes                       dFPU
  0.04000000000000000083     0.04000000000000000083     0.04000000000000000083
dReg: +0.000010100011110101110000101000111101011100001010001111011
dRes: +0.000010100011110101110000101000111101011100001010001111011
dFPU: +0.000010100011110101110000101000111101011100001010001111011
Original REs Count: 1, Computational REs Count: 3, Saved REs Count: 2

Iteration Number: 3
           dReg                       dRes                       dFPU
  0.08999999999999999667     0.09000000000000001054     0.08999999999999999667
dReg: +0.00010111000010100011110101110000101000111101011100001010
dRes: +0.00010111000010100011110101110000101000111101011100001011
dFPU: +0.00010111000010100011110101110000101000111101011100001010
Original REs Count: 1, Computational REs Count: 5, Saved REs Count: 5

Iteration Number: 4
           dReg                       dRes                       dFPU
  0.1600000000000000033      0.1600000000000000033      0.1600000000000000033
dReg: +0.0010100011110101110000101000111101011100001010001111011
dRes: +0.0010100011110101110000101000111101011100001010001111011
dFPU: +0.0010100011110101110000101000111101011100001010001111011
Original REs Count: 1, Computational REs Count: 9, Saved REs Count: 9

Iteration Number: 5
           dReg                       dRes                       dFPU
           0.25              0.2499999999999999445               0.25
dReg: +0.010000000000000000000000000000000000000000000000000000
dRes: +0.0011111111111111111111111111111111111111111111111111110
dFPU: +0.010000000000000000000000000000000000000000000000000000
Original REs Count: 1, Computational REs Count: 14, Saved REs Count: 14

Iteration Number: 6
           dReg                       dRes                       dFPU
  0.3599999999999999867      0.3600000000000000422      0.3599999999999999867
dReg: +0.010111000010100011110101110000101000111101011100001010
dRes: +0.010111000010100011110101110000101000111101011100001011
dFPU: +0.010111000010100011110101110000101000111101011100001010
Original REs Count: 1, Computational REs Count: 20, Saved REs Count: 20

Iteration Number: 7
           dReg                       dRes                       dFPU
  0.4899999999999999911      0.4900000000000001576      0.4899999999999999911
dReg: +0.011111010111000010100011110101110000101000111101011100
dRes: +0.011111010111000010100011110101110000101000111101011111
dFPU: +0.011111010111000010100011110101110000101000111101011100
Original REs Count: 1, Computational REs Count: 27, Saved REs Count: 27

Iteration Number: 8
           dReg                       dRes                       dFPU
  0.6400000000000000133      0.6400000000000002354      0.6400000000000000133
dReg: +0.10100011110101110000101000111101011100001010001111011
dRes: +0.10100011110101110000101000111101011100001010001111101
dFPU: +0.10100011110101110000101000111101011100001010001111011
Original REs Count: 1, Computational REs Count: 35, Saved REs Count: 35

Iteration Number: 9
           dReg                       dRes                       dFPU
  0.8100000000000000533      0.8100000000000003864      0.8100000000000000533
dReg: +0.11001111010111000010100011110101110000101000111101100
```

```
dRes: +0.11001111010111000010100011110101110000101000111101111
dFPU: +0.11001111010111000010100011110101110000101000111101100
Original REs Count: 1, Computational REs Count: 44, Saved REs Count: 44

Iteration Number: 10
           dReg                       dRes                       dFPU
            1                1.0000000000000004441                1
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000000010
dFPU: +1.0000000000000000000000000000000000000000000000000000
Original REs Count: 1, Computational REs Count: 54, Saved REs Count: 54

Iteration Number: 11
           dReg                       dRes                       dFPU
   1.209999999999999964       1.210000000000000631       1.209999999999999964
dReg: +1.0011010111000010100011110101110000101000111101011100
dRes: +1.0011010111000010100011110101110000101000111101011111
dFPU: +1.0011010111000010100011110101110000101000111101011100
Original REs Count: 1, Computational REs Count: 65, Saved REs Count: 65

Iteration Number: 12
           dReg                       dRes                       dFPU
   1.439999999999999947       1.440000000000000835       1.439999999999999947
dReg: +1.0111000010100011110101110000101000111101011100001010
dRes: +1.0111000010100011110101110000101000111101011100001110
dFPU: +1.0111000010100011110101110000101000111101011100001010
Original REs Count: 1, Computational REs Count: 77, Saved REs Count: 77

Iteration Number: 13
           dReg                       dRes                       dFPU
   1.689999999999999947       1.690000000000001057       1.689999999999999947
dReg: +1.1011000010100011110101110000101000111101011100001010
dRes: +1.1011000010100011110101110000101000111101011100001111
dFPU: +1.1011000010100011110101110000101000111101011100001010
Original REs Count: 1, Computational REs Count: 90, Saved REs Count: 90

Iteration Number: 14
           dReg                       dRes                       dFPU
   1.959999999999999964       1.960000000000001297       1.959999999999999964
dReg: +1.1111010111000010100011110101110000101000111101011100
dRes: +1.1111010111000010100011110101110000101000111101100010
dFPU: +1.1111010111000010100011110101110000101000111101011100
Original REs Count: 1, Computational REs Count: 104, Saved REs Count: 104

Iteration Number: 15
           dReg                       dRes                       dFPU
           2.25               2.250000000000001332               2.25
dReg: +10.010000000000000000000000000000000000000000000000000
dRes: +10.010000000000000000000000000000000000000000000000011
dFPU: +10.010000000000000000000000000000000000000000000000000
Original REs Count: 1, Computational REs Count: 119, Saved REs Count: 119

Iteration Number: 16
           dReg                       dRes                       dFPU
   2.560000000000000053       2.56000000000000183        2.560000000000000053
dReg: +10.100011110101110000101000111101011100001010001111011
dRes: +10.100011110101110000101000111101011100001010001111111
dFPU: +10.100011110101110000101000111101011100001010001111011
Original REs Count: 1, Computational REs Count: 135, Saved REs Count: 135

Iteration Number: 17
           dReg                       dRes                       dFPU
   2.890000000000000124       2.890000000000001901       2.890000000000000124
dReg: +10.111000111101011100001010001111010111000010100011111
dRes: +10.111000111101011100001010001111010111000010100100011
dFPU: +10.111000111101011100001010001111010111000010100011111
Original REs Count: 1, Computational REs Count: 152, Saved REs Count: 152

Iteration Number: 18
           dReg                       dRes                       dFPU
   3.240000000000000213       3.240000000000002434       3.240000000000000213
```

```
dReg: +11.0011110101110000101000111101011100001010001111 01100
```

```
dReg: +11.001111010111000010100011110101110000101000111101100
dRes: +11.001111010111000010100011110101110000101000111110001
dFPU: +11.001111010111000010100011110101110000101000111101100
Original REs Count: 1, Computational REs Count: 170, Saved REs Count: 170

Iteration Number: 19
           dReg                       dRes                       dFPU
   3.61000000000000032        3.61000000000002984        3.60999999999999876
dReg: +11.100111000010100011110101110000101000111101011100010
dRes: +11.100111000010100011110101110000101000111101011101000
dFPU: +11.100111000010100011110101110000101000111101011100001
Original REs Count: 1, Computational REs Count: 189, Saved REs Count: 189

Iteration Number: 20
           dReg                       dRes                       dFPU
            4                 4.00000000000002664                 4
dReg: +100.0000000000000000000000000000000000000000000000000
dRes: +100.0000000000000000000000000000000000000000000000011
dFPU: +100.0000000000000000000000000000000000000000000000000
Original REs Count: 1, Computational REs Count: 209, Saved REs Count: 209

Iteration Number: 21
           dReg                       dRes                       dFPU
   4.41000000000000142        4.40999999999994813        4.41000000000000142
dReg: +100.0110100011110101110000101000111101011100001010010 0
dRes: +100.0110100011110101110000101000111101011100001001111 0
dFPU: +100.0110100011110101110000101000111101011100001010010 0
Original REs Count: 1, Computational REs Count: 230, Saved REs Count: 230
```

```
   7.290000000000000036       7.289999999999933422       7.290000000000000036
dReg: +111.0100101000111101011100001010001111010111000010100 1
dRes: +111.0100101000111101011100001010001111010110111101111 0
dFPU: +111.0100101000111101011100001010001111010111000010100 1
Original REs Count: 1, Computational REs Count: 377, Saved REs Count: 377

Iteration Number: 28
           dReg                       dRes                       dFPU
   7.839999999999999858       7.83999999999992081        7.839999999999999858
dReg: +111.1101011100001010001111010111000010100011110101110 0
dRes: +111.1101011100001010001111010111000010100011110000001 1
dFPU: +111.1101011100001010001111010111000010100011110101110 0
Original REs Count: 1, Computational REs Count: 405, Saved REs Count: 405

Iteration Number: 29
           dReg                       dRes                       dFPU
   8.410000000000000142       8.409999999999909548       8.410000000000000142
dReg: +1000.011010001111010111000010100011110101110000101001 0
dRes: +1000.011010001111010111000010100011110101110000001111 1
dFPU: +1000.011010001111010111000010100011110101110000101001 0
Original REs Count: 1, Computational REs Count: 434, Saved REs Count: 434

Iteration Number: 30
           dReg                       dRes                       dFPU
            9                 8.999999999999896972                9
dReg: +1001.000000000000000000000000000000000000000000000000 0
dRes: +1000.111111111111111111111111111111111111111111100011 0
dFPU: +1001.000000000000000000000000000000000000000000000000 0
Original REs Count: 1, Computational REs Count: 464, Saved REs Count: 464

Iteration Number: 31
           dReg                       dRes                       dFPU
   9.609999999999999432       9.609999999999883968       9.609999999999999432
dReg: +1001.100111000010100011110101110000101000111101011100 0
dRes: +1001.100111000010100011110101110000101000111100111011 1
dFPU: +1001.100111000010100011110101110000101000111101011100 0
Original REs Count: 1, Computational REs Count: 495, Saved REs Count: 495

Iteration Number: 32
           dReg                       dRes                       dFPU
  10.240000000000000213      10.239999999999870539      10.240000000000000213
dReg: +1010.001111010111000010100011110101110000101000111101 1
dRes: +1010.001111010111000010100011110101110000101000011001 0
dFPU: +1010.001111010111000010100011110101110000101000111101 1
Original REs Count: 1, Computational REs Count: 527, Saved REs Count: 527

Iteration Number: 33
           dReg                       dRes                       dFPU
   10.89000000000000057       10.88999999999985668       10.89000000000000057
dReg: +1010.111000111101011100001010001111010111000010100100 0
dRes: +1010.111000111101011100001010001111010111000001111011 1
dFPU: +1010.111000111101011100001010001111010111000010100100 0
Original REs Count: 1, Computational REs Count: 560, Saved REs Count: 560

Iteration Number: 34
           dReg                       dRes                       dFPU
   11.5600000000000005        11.5599999999998424        11.5600000000000005
dReg: +1011.100011110101110000101000111101011100001010001111 1
dRes: +1011.100011110101110000101000111101011100001001100011 0
dFPU: +1011.100011110101110000101000111101011100001010001111 1
Original REs Count: 1, Computational REs Count: 594, Saved REs Count: 594

Iteration Number: 35
           dReg                       dRes                       dFPU
          12.25               12.24999999999982769              12.25
dReg: +1100.010000000000000000000000000000000000000000000000 0
dRes: +1100.001111111111111111111111111111111111111111100111 1
dFPU: +1100.010000000000000000000000000000000000000000000000 0
Original REs Count: 1, Computational REs Count: 629, Saved REs Count: 629

Iteration Number: 36
```

```
           dReg                       dRes                       dFPU
   12.96000000000000085       12.95999999999981256       12.96000000000000085
dReg: +1100.1111010111000010100011110101110000101000111101100
dRes: +1100.1111010111000010100011110101110000101000110000010
dFPU: +1100.1111010111000010100011110101110000101000111101100
Original REs Count: 1, Computational REs Count: 665, Saved REs Count: 665

Iteration Number: 37
           dReg                       dRes                       dFPU
   13.69000000000000128        13.689999999999797        13.6899999999999995
dReg: +1101.1011000010100011110101110000101000111101011100010
dRes: +1101.1011000010100011110101110000101000111101001101111
dFPU: +1101.1011000010100011110101110000101000111101011100001
Original REs Count: 1, Computational REs Count: 702, Saved REs Count: 702

Iteration Number: 38
           dReg                       dRes                       dFPU
   14.44000000000000128       14.43999999999978101       14.4399999999999995
dReg: +1110.0111000010100011110101110000101000111101011100010
dRes: +1110.0111000010100011110101110000101000111101001100110
dFPU: +1110.0111000010100011110101110000101000111101011100001
Original REs Count: 1, Computational REs Count: 740, Saved REs Count: 740

Iteration Number: 39
           dReg                       dRes                       dFPU
   15.21000000000000085       15.2099999999997646        15.21000000000000085
dReg: +1111.0011010111000010100011110101110000101000111101100
dRes: +1111.0011010111000010100011110101110000101000101100111
dFPU: +1111.0011010111000010100011110101110000101000111101100
Original REs Count: 1, Computational REs Count: 779, Saved REs Count: 779

Iteration Number: 40
           dReg                       dRes                       dFPU
            16                15.99999999999974776                16
dReg: +10000.000000000000000000000000000000000000000000000000
dRes: +1111.1111111111111111111111111111111111111111101110010
dFPU: +10000.000000000000000000000000000000000000000000000000
Original REs Count: 1, Computational REs Count: 819, Saved REs Count: 819

Iteration Number: 41
           dReg                       dRes                       dFPU
   16.80999999999999872       16.80999999999973227       16.80999999999999872
dReg: +10000.110011110101110000101000111101011100001010001111
dRes: +10000.110011110101110000101000111101011100001001000100
dFPU: +10000.110011110101110000101000111101011100001010001111
Original REs Count: 1, Computational REs Count: 860, Saved REs Count: 860

Iteration Number: 42
           dReg                       dRes                       dFPU
   17.64000000000000057       17.6399999999997128        17.64000000000000057
dReg: +10001.101000111101011100001010001111010111000010100100
dRes: +10001.101000111101011100001010001111010111000001010011
dFPU: +10001.101000111101011100001010001111010111000010100100
Original REs Count: 1, Computational REs Count: 902, Saved REs Count: 902

Iteration Number: 43
           dReg                       dRes                       dFPU
   18.49000000000000199       18.48999999999969646       18.49000000000000199
dReg: +10010.011111010111000010100011110101110000101000111110
dRes: +10010.011111010111000010100011110101110000100111101000
dFPU: +10010.011111010111000010100011110101110000101000111110
Original REs Count: 1, Computational REs Count: 945, Saved REs Count: 945

Iteration Number: 44
           dReg                       dRes                       dFPU
   19.36000000000000298       19.35999999999967614       19.35999999999999943
dReg: +10011.010111000010100011110101110000101000111101011101
dRes: +10011.010111000010100011110101110000101000111100000001
dFPU: +10011.010111000010100011110101110000101000111101011100
Original REs Count: 1, Computational REs Count: 989, Saved REs Count: 989
```

```
Iteration Number: 45
           dReg                       dRes                       dFPU
   20.25000000000000355       20.24999999999965894              20.25
dReg: +10100.010000000000000000000000000000000000000000000001
dRes: +10100.001111111111111111111111111111111111111110100000
dFPU: +10100.010000000000000000000000000000000000000000000000
Original REs Count: 1, Computational REs Count: 1034, Saved REs Count: 1034

Iteration Number: 46
           dReg                       dRes                       dFPU
   21.1600000000000037        21.15999999999963776       21.16000000000000014
dReg: +10101.001010001111010111000010100011110101110000101010
dRes: +10101.001010001111010111000010100011110101101111000011
dFPU: +10101.001010001111010111000010100011110101110000101001
Original REs Count: 1, Computational REs Count: 1080, Saved REs Count: 1080

Iteration Number: 47
           dReg                       dRes                       dFPU
   22.09000000000000341       22.08999999999961972       22.08999999999999986
dReg: +10110.000101110000101000111101011100001010001111011000
dRes: +10110.000101110000101000111101011100001010001101101100
dFPU: +10110.000101110000101000111101011100001010001111010111
Original REs Count: 1, Computational REs Count: 1127, Saved REs Count: 1127

Iteration Number: 48
           dReg                       dRes                       dFPU
   23.0400000000000027        23.03999999999959769       23.03999999999999915
dReg: +10111.000010100011110101110000101000111101011100001011
dRes: +10111.000010100011110101110000101000111101011010011001
dFPU: +10111.000010100011110101110000101000111101011100001010
Original REs Count: 1, Computational REs Count: 1175, Saved REs Count: 1175

Iteration Number: 49
           dReg                       dRes                       dFPU
   24.01000000000000156       24.00999999999957879       24.01000000000000156
dReg: +11000.000000101000111101011100001010001111010111000011
dRes: +11000.000000101000111101011100001010001111010101001100
dFPU: +11000.000000101000111101011100001010001111010111000011
Original REs Count: 1, Computational REs Count: 1224, Saved REs Count: 1224

Iteration Number: 50
           dReg                       dRes                       dFPU
   25.00000000000000355       24.99999999999955591                25
dReg: +11001.000000000000000000000000000000000000000000000001
dRes: +11000.111111111111111111111111111111111111111110000011
dFPU: +11001.000000000000000000000000000000000000000000000000
Original REs Count: 1, Computational REs Count: 1274, Saved REs Count: 1274
```

Appendix AB-1- CHFort Object Code of Test Case Two

```
<<variablesvalues>>
Name: Y
Value:
IsHistory: 1
IsFixed: 0
DataType: 5
nDimsCount: 0
<<ProgramCode>>
Label StartProg
BeginCode A
  push Number 1
  push Number 7
  DivSingle
EndCode
BeginCode B
  push Number 2
  push Number 7
  DivSingle
EndCode
BeginCode Y
push Number 0
EndCode
BeginCode N
push Number 0
EndCode
Label 100
BeginBoolCode dIf_Bool_1_0
  push String N
  push Number 100
  Minus
EndCode
GoToCond 900 GreaterOrEqual dIf_Bool_1_0
BeginCode Y
  push String Y
  push Number 10000
  Plus
  push String A
  Plus
  push String B
  Minus
EndCode
BeginCode N
  push String N
  push Number 1
  Plus
EndCode
GoTo 100
Label 900
Label EndProg
```

Appendix AB-2- Results of Test Case Two

```
          a:                  b:                  c:
     Experiment One     Experiment Two     Experiment Three
     1 sum, lsb,        1 sum, sign of        2 sums,
     sign of result     result,lsb,#bits      lsb, #nbits

Iteration Number: 1, Value: 9999.857142857143117
         dReg                     dRes                   dFPU
 9999.857142857143117  a:  9999.857142857143117   9999.857142857143117
                       b:  9999.857142857143117   9999.857142857143117
                       c:  9999.857142857143117   9999.857142857143117
a: dRes: +10011100001111.110110110110110110110110110110110110111
b: dRes: +10011100001111.110110110110110110110110110110110110111
c: dRes: +10011100001111.110110110110110110110110110110110110111
   dReg: +10011100001111.110110110110110110110110110110110110111
a: dFPU: +10011100001111.110110110110110110110110110110110110111
b: dFPU: +10011100001111.110110110110110110110110110110110110111
c: dFPU: +10011100001111.110110110110110110110110110110110110111

Iteration Number: 2, Value: 19999.71428571428623
         dReg                     dRes                   dFPU
 19999.71428571428623  a:  19999.71428571428623   19999.71428571428623
                       b:  19999.71428571428623   19999.71428571428623
                       c:  19999.71428571428623   19999.71428571428623
a: dRes: +100111000011111.10110110110110110110110110110110110111
b: dRes: +100111000011111.10110110110110110110110110110110110111
c: dRes: +100111000011111.10110110110110110110110110110110110111
   dReg: +100111000011111.10110110110110110110110110110110110111
a: dFPU: +100111000011111.10110110110110110110110110110110110111
b: dFPU: +100111000011111.10110110110110110110110110110110110111
c: dFPU: +100111000011111.10110110110110110110110110110110110111

Iteration Number: 3, Value: 29999.57142857142753
         dReg                     dRes                   dFPU
 29999.57142857142753  a:  29999.57142857142753   29999.57142857142753
                       b:  29999.57142857143117   29999.57142857142753
                       c:  29999.57142857143117   29999.57142857142753
a: dRes: +111010100101111.10010010010010010010010010010010010010
b: dRes: +111010100101111.10010010010010010010010010010010010011
c: dRes: +111010100101111.10010010010010010010010010010010010011
   dReg: +111010100101111.10010010010010010010010010010010010010
a: dFPU: +111010100101111.10010010010010010010010010010010010010
b: dFPU: +111010100101111.10010010010010010010010010010010010010
c: dFPU: +111010100101111.10010010010010010010010010010010010010

Iteration Number: 4, Value: 39999.42857142857247
         dReg                     dRes                   dFPU
 39999.42857142857247  a:  39999.42857142857247   39999.42857142857247
                       b:  39999.42857142855792   39999.42857142857247
                       c:  39999.42857142857247   39999.42857142857247
a: dRes: +1001110000111111.0110110110110110110110110110110110111
b: dRes: +1001110000111111.0110110110110110110110110110110110101
c: dRes: +1001110000111111.0110110110110110110110110110110110111
   dReg: +1001110000111111.0110110110110110110110110110110110111
a: dFPU: +1001110000111111.0110110110110110110110110110110110111
b: dFPU: +1001110000111111.0110110110110110110110110110110110111
c: dFPU: +1001110000111111.0110110110110110110110110110110110111

Iteration Number: 5, Value: 49999.2857142857174
         dReg                     dRes                   dFPU
 49999.2857142857174   a:  49999.28571428571013   49999.2857142857174
                       b:  49999.28571428570285   49999.2857142857174
                       c:  49999.2857142857174    49999.2857142857174
a: dRes: +1100001101001111.0100100100100100100100100100100100100
b: dRes: +1100001101001111.0100100100100100100100100100100100011
c: dRes: +1100001101001111.0100100100100100100100100100100100101
   dReg: +1100001101001111.0100100100100100100100100100100100101
a: dFPU: +1100001101001111.0100100100100100100100100100100100101
b: dFPU: +1100001101001111.0100100100100100100100100100100100101
```

```
c: dFPU: +1100001101001111.0100100100100100100100100100100100101

Iteration Number: 6, Value: 59999.14285714285506
         dReg                     dRes                  dFPU
 59999.14285714286234  a:  59999.14285714285506   59999.14285714285506
                       b:  59999.14285714284051   59999.14285714285506
                       c:  59999.14285714286234   59999.14285714285506
a: dRes: +1110101001011111.0010010010010010010010010010010010010
b: dRes: +1110101001011111.0010010010010010010010010010010010000
c: dRes: +1110101001011111.0010010010010010010010010010010010011
   dReg: +1110101001011111.0010010010010010010010010010010010011
a: dFPU: +1110101001011111.0010010010010010010010010010010010010
b: dFPU: +1110101001011111.0010010010010010010010010010010010010
c: dFPU: +1110101001011111.0010010010010010010010010010010010010

Iteration Number: 7, Value: 69999
         dReg                     dRes                  dFPU
        69999          a:         69999                 69999
                       b:  69999.0000000000291          69999
                       c:         69999                 69999
a: dRes: +10001000101101111.000000000000000000000000000000000000
b: dRes: +10001000101101111.000000000000000000000000000000000010
c: dRes: +10001000101101111.000000000000000000000000000000000000
   dReg: +10001000101101111.000000000000000000000000000000000000
a: dFPU: +10001000101101111.000000000000000000000000000000000000
b: dFPU: +10001000101101111.000000000000000000000000000000000000
c: dFPU: +10001000101101111.000000000000000000000000000000000000

Iteration Number: 8, Value: 79998.85714285714494
         dReg                     dRes                  dFPU
 79998.85714285714494  a:  79998.85714285714494   79998.85714285714494
                       b:  79998.85714285717404   79998.85714285714494
                       c:  79998.85714285714494   79998.85714285714494
a: dRes: +10011100001111110.110110110110110110110110110110110111
b: dRes: +10011100001111110.110110110110110110110110110110111001
c: dRes: +10011100001111110.110110110110110110110110110110110111
   dReg: +10011100001111110.110110110110110110110110110110110111
a: dFPU: +10011100001111110.110110110110110110110110110110110111
b: dFPU: +10011100001111110.110110110110110110110110110110110111
c: dFPU: +10011100001111110.110110110110110110110110110110110111

Iteration Number: 9, Value: 89998.71428571428987
         dReg                     dRes                  dFPU
 89998.71428571428987  a:  89998.71428571428987   89998.71428571428987
                       b:  89998.71428571431898   89998.71428571428987
                       c:  89998.71428571427532   89998.71428571428987
a: dRes: +10101111110001110.101101101101101101101101101101101110
b: dRes: +10101111110001110.101101101101101101101101101101110000
c: dRes: +10101111110001110.101101101101101101101101101101101101
   dReg: +10101111110001110.101101101101101101101101101101101110
a: dFPU: +10101111110001110.101101101101101101101101101101101110
b: dFPU: +10101111110001110.101101101101101101101101101101101110
c: dFPU: +10101111110001110.101101101101101101101101101101101110

Iteration Number: 10, Value: 99998.57142857143481
         dReg                     dRes                  dFPU
 99998.57142857143481  a:  99998.57142857142026   99998.57142857143481
                       b:  99998.57142857147846   99998.57142857143481
                       c:  99998.57142857143481   99998.57142857143481
a: dRes: +11000011010011110.100100100100100100100100100100100100
b: dRes: +11000011010011110.100100100100100100100100100100101000
c: dRes: +11000011010011110.100100100100100100100100100100100101
   dReg: +11000011010011110.100100100100100100100100100100100101
a: dFPU: +11000011010011110.100100100100100100100100100100100101
b: dFPU: +11000011010011110.100100100100100100100100100100100101
c: dFPU: +11000011010011110.100100100100100100100100100100100101

Iteration Number: 11, Value: 109998.4285714285652
         dReg                     dRes                  dFPU
 109998.4285714285797  a:  109998.4285714285797   109998.4285714285652
                       b:  109998.4285714286088   109998.4285714285652
                       c:  109998.4285714285652   109998.4285714285652
a: dRes: +11010110110101110.011011011011011011011011011011011100
b: dRes: +11010110110101110.011011011011011011011011011011011110
c: dRes: +11010110110101110.011011011011011011011011011011011011
   dReg: +11010110110101110.011011011011011011011011011011011100
a: dFPU: +11010110110101110.011011011011011011011011011011011011
b: dFPU: +11010110110101110.011011011011011011011011011011011011
c: dFPU: +11010110110101110.011011011011011011011011011011011011

Iteration Number: 12, Value: 119998.2857142857101
         dReg                     dRes                   dFPU
 119998.2857142857247  a:  119998.2857142857101   119998.2857142857101
                       b:  119998.2857142857683   119998.2857142857101
                       c:  119998.2857142857247   119998.2857142857101
a: dRes: +11101010010111110.010010010010010010010010010010010010
b: dRes: +11101010010111110.010010010010010010010010010010010110
c: dRes: +11101010010111110.010010010010010010010010010010010011
   dReg: +11101010010111110.010010010010010010010010010010010011
a: dFPU: +11101010010111110.010010010010010010010010010010010010
b: dFPU: +11101010010111110.010010010010010010010010010010010010
c: dFPU: +11101010010111110.010010010010010010010010010010010010

Iteration Number: 13, Value: 129998.1428571428551
         dReg                     dRes                   dFPU
 129998.1428571428696  a:  129998.1428571428551   129998.1428571428551
                       b:  129998.1428571429133   129998.1428571428551
                       c:  129998.1428571428551   129998.1428571428551
a: dRes: +11111101111001110.001001001001001001001001001001001001
b: dRes: +11111101111001110.001001001001001001001001001001001101
c: dRes: +11111101111001110.001001001001001001001001001001001001
   dReg: +11111101111001110.001001001001001001001001001001001010
a: dFPU: +11111101111001110.001001001001001001001001001001001001
b: dFPU: +11111101111001110.001001001001001001001001001001001001
c: dFPU: +11111101111001110.001001001001001001001001001001001001

Iteration Number: 14, Value: 139998
         dReg                     dRes                   dFPU
        139998         a:         139998                 139998
                       b:  139998.0000000000582          139998
                       c:         139998                 139998
a: dRes: +100010001011011110.00000000000000000000000000000000000
b: dRes: +100010001011011110.00000000000000000000000000000000010
c: dRes: +100010001011011110.00000000000000000000000000000000000
   dReg: +100010001011011110.00000000000000000000000000000000000
a: dFPU: +100010001011011110.00000000000000000000000000000000000
b: dFPU: +100010001011011110.00000000000000000000000000000000000
c: dFPU: +100010001011011110.00000000000000000000000000000000000

Iteration Number: 15, Value: 149997.8571428571304
         dReg                     dRes                   dFPU
 149997.8571428571304  a:  149997.8571428571595   149997.8571428571304
                       b:  149997.8571428571886   149997.8571428571304
                       c:  149997.8571428571304   149997.8571428571304
a: dRes: +100100100111101101.11011011011011011011011011011011100
b: dRes: +100100100111101101.11011011011011011011011011011011101
c: dRes: +100100100111101101.11011011011011011011011011011011011
   dReg: +100100100111101101.11011011011011011011011011011011011
a: dFPU: +100100100111101101.11011011011011011011011011011011011
b: dFPU: +100100100111101101.11011011011011011011011011011011011
c: dFPU: +100100100111101101.11011011011011011011011011011011011

Iteration Number: 16, Value: 159997.7142857142899
         dReg                     dRes                   dFPU
 159997.7142857142608  a:  159997.7142857142899   159997.7142857142899
                       b:  159997.7142857143481   159997.7142857142899
                       c:  159997.7142857142899   159997.7142857142899
a: dRes: +100111000011111101.10110110110110110110110110110110111
b: dRes: +100111000011111101.10110110110110110110110110110111001
c: dRes: +100111000011111101.10110110110110110110110110110110111
   dReg: +100111000011111101.10110110110110110110110110110110110
a: dFPU: +100111000011111101.10110110110110110110110110110110111
```

```
b: dFPU: +100111000011111101.10110110110110110110110110110110111
c: dFPU: +100111000011111101.10110110110110110110110110110110111

Iteration Number: 17, Value: 169997.5714285714202
         dReg                     dRes                   dFPU
 169997.5714285713912  a:  169997.5714285714202   169997.5714285714202
                       b:  169997.5714285715076   169997.5714285714202
                       c:  169997.5714285714494   169997.5714285714202
a: dRes: +101001100000001101.10010010010010010010010010010010010
b: dRes: +101001100000001101.10010010010010010010010010010010101
c: dRes: +101001100000001101.10010010010010010010010010010010011
   dReg: +101001100000001101.10010010010010010010010010010010001
a: dFPU: +101001100000001101.10010010010010010010010010010010010
b: dFPU: +101001100000001101.10010010010010010010010010010010010
c: dFPU: +101001100000001101.10010010010010010010010010010010010

Iteration Number: 18, Value: 179997.4285714285798
         dReg                     dRes                   dFPU
 179997.4285714285215  a:  179997.4285714285798   179997.4285714285798
                       b:  179997.428571428638    179997.4285714285798
                       c:  179997.4285714285506   179997.4285714285798
a: dRes: +101011111100011101.01101101101101101101101101101101110
b: dRes: +101011111100011101.01101101101101101101101101101110000
c: dRes: +101011111100011101.01101101101101101101101101101101101
   dReg: +101011111100011101.01101101101101101101101101101101100
a: dFPU: +101011111100011101.01101101101101101101101101101101110
b: dFPU: +101011111100011101.01101101101101101101101101101101110
c: dFPU: +101011111100011101.01101101101101101101101101101101110

Iteration Number: 19, Value: 189997.2857142857101
         dReg                     dRes                   dFPU
 189997.2857142856519  a:  189997.2857142857101   189997.2857142857101
                       b:  189997.2857142857974   189997.2857142857101
                       c:  189997.2857142857101   189997.2857142857101
a: dRes: +101110011000101101.01001001001001001001001001001001001
b: dRes: +101110011000101101.01001001001001001001001001001001100
c: dRes: +101110011000101101.01001001001001001001001001001001001
   dReg: +101110011000101101.01001001001001001001001001001000111
a: dFPU: +101110011000101101.01001001001001001001001001001001001
b: dFPU: +101110011000101101.01001001001001001001001001001001001
c: dFPU: +101110011000101101.01001001001001001001001001001001001

Iteration Number: 20, Value: 199997.1428571428696
         dReg                     dRes                   dFPU
 199997.1428571427823  a:  199997.1428571428405   199997.1428571428696
                       b:  199997.1428571429569   199997.1428571428696
                       c:  199997.1428571428696   199997.1428571428696
a: dRes: +110000110100111101.00100100100100100100100100100100100
b: dRes: +110000110100111101.00100100100100100100100100100101000
c: dRes: +110000110100111101.00100100100100100100100100100100101
   dReg: +110000110100111101.00100100100100100100100100100100010
a: dFPU: +110000110100111101.00100100100100100100100100100100101
b: dFPU: +110000110100111101.00100100100100100100100100100100101
c: dFPU: +110000110100111101.00100100100100100100100100100100101

Iteration Number: 21, Value: 209997
         dReg                     dRes                   dFPU
 209996.9999999999127  a:         209997                 209997
                       b:  209997.0000000000873          209997
                       c:         209997                 209997
a: dRes: +110011010001001101.00000000000000000000000000000000000
b: dRes: +110011010001001101.00000000000000000000000000000000011
c: dRes: +110011010001001101.00000000000000000000000000000000000
   dReg: +110011010001001100.11111111111111111111111111111111101
a: dFPU: +110011010001001101.00000000000000000000000000000000000
b: dFPU: +110011010001001101.00000000000000000000000000000000000
c: dFPU: +110011010001001101.00000000000000000000000000000000000

Iteration Number: 22, Value: 219996.8571428571304
         dReg                     dRes                   dFPU
 219996.8571428570431  a:  219996.8571428571595   219996.8571428571304
```

```
                       b:  219996.8571428572177   219996.8571428571304
                       c:  219996.8571428571304   219996.8571428571304
a: dRes: +110101101101011100.11011011011011011011011011011011100
b: dRes: +110101101101011100.11011011011011011011011011011011110
c: dRes: +110101101101011100.11011011011011011011011011011011011
   dReg: +110101101101011100.11011011011011011011011011011011000
a: dFPU: +110101101101011100.11011011011011011011011011011011011
b: dFPU: +110101101101011100.11011011011011011011011011011011011
c: dFPU: +110101101101011100.11011011011011011011011011011011011

Iteration Number: 23, Value: 229996.7142857142899
         dReg                     dRes                   dFPU
 229996.7142857141735  a:  229996.7142857142899   229996.7142857142899
                       b:  229996.7142857143772   229996.7142857142899
                       c:  229996.7142857142899   229996.7142857142899
a: dRes: +111000001001101100.10110110110110110110110110110110111
b: dRes: +111000001001101100.10110110110110110110110110110111010
c: dRes: +111000001001101100.10110110110110110110110110110110111
   dReg: +111000001001101100.10110110110110110110110110110110011
a: dFPU: +111000001001101100.10110110110110110110110110110110111
b: dFPU: +111000001001101100.10110110110110110110110110110110111
c: dFPU: +111000001001101100.10110110110110110110110110110110111

Iteration Number: 24, Value: 239996.5714285714202
         dReg                     dRes                   dFPU
 239996.5714285713038  a:  239996.5714285714202   239996.5714285714202
                       b:  239996.5714285715367   239996.5714285714202
                       c:  239996.5714285714494   239996.5714285714202
a: dRes: +111010100101111100.10010010010010010010010010010010010
b: dRes: +111010100101111100.10010010010010010010010010010010110
c: dRes: +111010100101111100.10010010010010010010010010010010011
   dReg: +111010100101111100.10010010010010010010010010010001110
a: dFPU: +111010100101111100.10010010010010010010010010010010010
b: dFPU: +111010100101111100.10010010010010010010010010010010010
c: dFPU: +111010100101111100.10010010010010010010010010010010010

Iteration Number: 25, Value: 249996.4285714285798
         dReg                     dRes                   dFPU
 249996.4285714284342  a:  249996.4285714285798   249996.4285714285798
                       b:  249996.428571428667    249996.4285714285798
                       c:  249996.4285714285506   249996.4285714285798
a: dRes: +111101000010001100.01101101101101101101101101101101110
b: dRes: +111101000010001100.01101101101101101101101101101110001
c: dRes: +111101000010001100.01101101101101101101101101101101101
   dReg: +111101000010001100.01101101101101101101101101101101001
a: dFPU: +111101000010001100.01101101101101101101101101101101110
b: dFPU: +111101000010001100.01101101101101101101101101101101110
c: dFPU: +111101000010001100.01101101101101101101101101101101110

Iteration Number: 26, Value: 259996.2857142857101
         dReg                     dRes                   dFPU
 259996.2857142855646  a:  259996.2857142857101   259996.2857142857101
                       b:  259996.2857142858266   259996.2857142857101
                       c:  259996.2857142857101   259996.2857142857101
a: dRes: +111111011110011100.01001001001001001001001001001001001
b: dRes: +111111011110011100.01001001001001001001001001001001101
c: dRes: +111111011110011100.01001001001001001001001001001001001
   dReg: +111111011110011100.01001001001001001001001001001000100
a: dFPU: +111111011110011100.01001001001001001001001001001001001
b: dFPU: +111111011110011100.01001001001001001001001001001001001
c: dFPU: +111111011110011100.01001001001001001001001001001001001

Iteration Number: 27, Value: 269996.1428571428405
         dReg                     dRes                   dFPU
 269996.1428571427241  a:  269996.1428571428405   269996.1428571428405
                       b:  269996.1428571422002   269996.1428571428405
                       c:  269996.1428571428987   269996.1428571428405
a: dRes: +1000001111010101100.0010010010010010010010010010010010
b: dRes: +1000001111010101100.0010010010010010010010010010000111
c: dRes: +1000001111010101100.0010010010010010010010010010010011
   dReg: +1000001111010101100.0010010010010010010010010010010000
```

```
a: dFPU: +1000001111010101100.0010010010010010010010010010010010
b: dFPU: +1000001111010101100.0010010010010010010010010010010010
c: dFPU: +1000001111010101100.0010010010010010010010010010010010

Iteration Number: 28, Value: 279996
         dReg                     dRes                   dFPU
 279995.9999999998836  a:         279996                 279996
                       b:  279995.9999999993015          279996
                       c:         279996                 279996
a: dRes: +1000100010110111100.0000000000000000000000000000000000
b: dRes: +1000100010110111011.1111111111111111111111111111110100
c: dRes: +1000100010110111100.0000000000000000000000000000000000
   dReg: +1000100010110111011.1111111111111111111111111111111110
a: dFPU: +1000100010110111100.0000000000000000000000000000000000
b: dFPU: +1000100010110111100.0000000000000000000000000000000000
c: dFPU: +1000100010110111100.0000000000000000000000000000000000

Iteration Number: 29, Value: 289995.8571428571595
         dReg                     dRes                   dFPU
 289995.8571428570431  a:  289995.8571428571595   289995.8571428571595
                       b:  289995.8571428564028   289995.8571428571595
                       c:  289995.8571428571013   289995.8571428571595
a: dRes: +1000110110011001011.1101101101101101101101101101101110
b: dRes: +1000110110011001011.1101101101101101101101101101100001
c: dRes: +1000110110011001011.1101101101101101101101101101101101
   dReg: +1000110110011001011.1101101101101101101101101101101100
a: dFPU: +1000110110011001011.1101101101101101101101101101101110
b: dFPU: +1000110110011001011.1101101101101101101101101101101110
c: dFPU: +1000110110011001011.1101101101101101101101101101101110

Iteration Number: 30, Value: 299995.7142857142608
         dReg                     dRes                   dFPU
 299995.7142857142026  a:  299995.714285714319    299995.7142857142608
                       b:  299995.7142857135041   299995.7142857142608
                       c:  299995.7142857142608   299995.7142857142608
a: dRes: +1001001001111011011.1011011011011011011011011011011100
b: dRes: +1001001001111011011.1011011011011011011011011011001110
c: dRes: +1001001001111011011.1011011011011011011011011011011011
   dReg: +1001001001111011011.1011011011011011011011011011011010
a: dFPU: +1001001001111011011.1011011011011011011011011011011011
b: dFPU: +1001001001111011011.1011011011011011011011011011011011
c: dFPU: +1001001001111011011.1011011011011011011011011011011011

Iteration Number: 31, Value: 309995.5714285714202
         dReg                     dRes                   dFPU
 309995.571428571362   a:  309995.5714285714202   309995.5714285714202
                       b:  309995.5714285706636   309995.5714285714202
                       c:  309995.5714285714202   309995.5714285714202
a: dRes: +1001011101011101011.1001001001001001001001001001001001
b: dRes: +1001011101011101011.1001001001001001001001001000111100
c: dRes: +1001011101011101011.1001001001001001001001001001001001
   dReg: +1001011101011101011.1001001001001001001001001001001000
a: dFPU: +1001011101011101011.1001001001001001001001001001001001
b: dFPU: +1001011101011101011.1001001001001001001001001001001001
c: dFPU: +1001011101011101011.1001001001001001001001001001001001

Iteration Number: 32, Value: 319995.4285714285798
         dReg                     dRes                   dFPU
 319995.4285714285215  a:  319995.4285714285798   319995.4285714285798
                       b:  319995.4285714277648   319995.4285714285798
                       c:  319995.4285714285798   319995.4285714285798
a: dRes: +1001110000111111011.0110110110110110110110110110110111
b: dRes: +1001110000111111011.0110110110110110110110110110101001
c: dRes: +1001110000111111011.0110110110110110110110110110110111
   dReg: +1001110000111111011.0110110110110110110110110110110110
a: dFPU: +1001110000111111011.0110110110110110110110110110110111
b: dFPU: +1001110000111111011.0110110110110110110110110110110111
c: dFPU: +1001110000111111011.0110110110110110110110110110110111

Iteration Number: 33, Value: 329995.2857142857392
         dReg                     dRes                   dFPU
 329995.285714285681   a:  329995.285714285681    329995.2857142857392
                       b:  329995.2857142849243   329995.2857142857392
                       c:  329995.2857142857392   329995.2857142857392
a: dRes: +1010000100100001011.0100100100100100100100100100100100
b: dRes: +1010000100100001011.0100100100100100100100100100010111
c: dRes: +1010000100100001011.0100100100100100100100100100100101
   dReg: +1010000100100001011.0100100100100100100100100100100100
a: dFPU: +1010000100100001011.0100100100100100100100100100100101
b: dFPU: +1010000100100001011.0100100100100100100100100100100101
c: dFPU: +1010000100100001011.0100100100100100100100100100100101

Iteration Number: 34, Value: 339995.1428571428405
         dReg                     dRes                   dFPU
 339995.1428571428405  a:  339995.1428571428405   339995.1428571428405
                       b:  339995.1428571420256   339995.1428571428405
                       c:  339995.1428571428987   339995.1428571428405
a: dRes: +1010011000000011011.0010010010010010010010010010010010
b: dRes: +1010011000000011011.0010010010010010010010010010000100
c: dRes: +1010011000000011011.0010010010010010010010010010010011
   dReg: +1010011000000011011.0010010010010010010010010010010010
a: dFPU: +1010011000000011011.0010010010010010010010010010010010
b: dFPU: +1010011000000011011.0010010010010010010010010010010010
c: dFPU: +1010011000000011011.0010010010010010010010010010010010

Iteration Number: 35, Value: 349995
         dReg                     dRes                   dFPU
        349995         a:         349995                 349995
                       b:  349994.9999999991269          349995
                       c:         349995                 349995
a: dRes: +1010101011100101011.0000000000000000000000000000000000
b: dRes: +1010101011100101010.1111111111111111111111111111110001
c: dRes: +1010101011100101011.0000000000000000000000000000000000
   dReg: +1010101011100101011.0000000000000000000000000000000000
a: dFPU: +1010101011100101011.0000000000000000000000000000000000
b: dFPU: +1010101011100101011.0000000000000000000000000000000000
c: dFPU: +1010101011100101011.0000000000000000000000000000000000

Iteration Number: 36, Value: 359994.8571428571595
         dReg                     dRes                   dFPU
 359994.8571428571595  a:  359994.8571428571595   359994.8571428571595
                       b:  359994.8571428562282   359994.8571428571595
                       c:  359994.8571428571013   359994.8571428571595
a: dRes: +1010111111000111010.1101101101101101101101101101101110
b: dRes: +1010111111000111010.1101101101101101101101101101011110
c: dRes: +1010111111000111010.1101101101101101101101101101101101
   dReg: +1010111111000111010.1101101101101101101101101101101110
a: dFPU: +1010111111000111010.1101101101101101101101101101101110
b: dFPU: +1010111111000111010.1101101101101101101101101101101110
c: dFPU: +1010111111000111010.1101101101101101101101101101101110

Iteration Number: 37, Value: 369994.7142857142608
         dReg                     dRes                   dFPU
 369994.714285714319   a:  369994.714285714319    369994.7142857142608
                       b:  369994.7142857133294   369994.7142857142608
                       c:  369994.7142857142608   369994.7142857142608
a: dRes: +1011010010101001010.1011011011011011011011011011011100
b: dRes: +1011010010101001010.1011011011011011011011011011001011
c: dRes: +1011010010101001010.1011011011011011011011011011011011
   dReg: +1011010010101001010.1011011011011011011011011011011100
a: dFPU: +1011010010101001010.1011011011011011011011011011011011
b: dFPU: +1011010010101001010.1011011011011011011011011011011011
c: dFPU: +1011010010101001010.1011011011011011011011011011011011

Iteration Number: 38, Value: 379994.5714285714202
         dReg                     dRes                   dFPU
 379994.5714285714785  a:  379994.5714285714202   379994.5714285714202
                       b:  379994.5714285704889   379994.5714285714202
                       c:  379994.5714285714202   379994.5714285714202
a: dRes: +1011100110001011010.1001001001001001001001001001001001
b: dRes: +1011100110001011010.1001001001001001001001001000111001
c: dRes: +1011100110001011010.1001001001001001001001001001001001
```

```
   dReg: +1011100110001011010.1001001001001001001001001001001010
a: dFPU: +1011100110001011010.1001001001001001001001001001001001
b: dFPU: +1011100110001011010.1001001001001001001001001001001001
c: dFPU: +1011100110001011010.1001001001001001001001001001001001

Iteration Number: 39, Value: 389994.4285714285798
         dReg                     dRes                  dFPU
 389994.428571428638   a:  389994.4285714285798   389994.4285714285798
                       b:  389994.4285714275902   389994.4285714285798
                       c:  389994.4285714285798   389994.4285714285798
a: dRes: +1011111001101101010.0110110110110110110110110110110111
b: dRes: +1011111001101101010.0110110110110110110110110110100110
c: dRes: +1011111001101101010.0110110110110110110110110110110111
   dReg: +1011111001101101010.0110110110110110110110110110111000
a: dFPU: +1011111001101101010.0110110110110110110110110110110111
b: dFPU: +1011111001101101010.0110110110110110110110110110110111
c: dFPU: +1011111001101101010.0110110110110110110110110110110111

Iteration Number: 40, Value: 399994.2857142857392
         dReg                     dRes                  dFPU
 399994.2857142857974  a:  399994.285714285681    399994.2857142857392
                       b:  399994.2857142847497   399994.2857142857392
                       c:  399994.2857142857392   399994.2857142857392
a: dRes: +1100001101001111010.0100100100100100100100100100100100
b: dRes: +1100001101001111010.0100100100100100100100100100010100
c: dRes: +1100001101001111010.0100100100100100100100100100100101
   dReg: +1100001101001111010.0100100100100100100100100100100110
a: dFPU: +1100001101001111010.0100100100100100100100100100100101
b: dFPU: +1100001101001111010.0100100100100100100100100100100101
c: dFPU: +1100001101001111010.0100100100100100100100100100100101

Iteration Number: 41, Value: 409994.1428571428405
         dReg                     dRes                  dFPU
 409994.1428571429569  a:  409994.1428571428405   409994.1428571428405
                       b:  409994.142857141851    409994.1428571428405
                       c:  409994.1428571428987   409994.1428571428405
a: dRes: +1100100000110001010.0010010010010010010010010010010010
b: dRes: +1100100000110001010.0010010010010010010010010010000001
c: dRes: +1100100000110001010.0010010010010010010010010010010011
   dReg: +1100100000110001010.0010010010010010010010010010010100
a: dFPU: +1100100000110001010.0010010010010010010010010010010010
b: dFPU: +1100100000110001010.0010010010010010010010010010010010
c: dFPU: +1100100000110001010.0010010010010010010010010010010010

Iteration Number: 42, Value: 419994
         dReg                     dRes                  dFPU
 419994.0000000001164  a:         419994                419994
                       b:  419993.9999999989522         419994
                       c:         419994                419994
a: dRes: +1100110100010011010.0000000000000000000000000000000000
b: dRes: +1100110100010011001.1111111111111111111111111111101110
c: dRes: +1100110100010011010.0000000000000000000000000000000000
   dReg: +1100110100010011010.0000000000000000000000000000000010
a: dFPU: +1100110100010011010.0000000000000000000000000000000000
b: dFPU: +1100110100010011010.0000000000000000000000000000000000
c: dFPU: +1100110100010011010.0000000000000000000000000000000000

Iteration Number: 43, Value: 429993.8571428571595
         dReg                     dRes                  dFPU
 429993.8571428572759  a:  429993.8571428571595   429993.8571428571595
                       b:  429993.8571428560536   429993.8571428571595
                       c:  429993.8571428571013   429993.8571428571595
a: dRes: +1101000111110101001.1101101101101101101101101101101110
b: dRes: +1101000111110101001.1101101101101101101101101101011011
c: dRes: +1101000111110101001.1101101101101101101101101101101101
   dReg: +1101000111110101001.1101101101101101101101101101110000
a: dFPU: +1101000111110101001.1101101101101101101101101101101110
b: dFPU: +1101000111110101001.1101101101101101101101101101101110
c: dFPU: +1101000111110101001.1101101101101101101101101101101110

Iteration Number: 44, Value: 439993.7142857142608
```

```
         dReg                     dRes                  dFPU
 439993.7142857144354  a:  439993.714285714319    439993.7142857142608
                       b:  439993.7142857131548   439993.7142857142608
                       c:  439993.7142857142608   439993.7142857142608
a: dRes: +1101011011010111001.1011011011011011011011011011011100
b: dRes: +1101011011010111001.1011011011011011011011011011001000
c: dRes: +1101011011010111001.1011011011011011011011011011011011
   dReg: +1101011011010111001.1011011011011011011011011011011110
a: dFPU: +1101011011010111001.1011011011011011011011011011011011
b: dFPU: +1101011011010111001.1011011011011011011011011011011011
c: dFPU: +1101011011010111001.1011011011011011011011011011011011

Iteration Number: 45, Value: 449993.5714285714202
         dReg                     dRes                  dFPU
 449993.5714285715949  a:  449993.5714285714202   449993.5714285714202
                       b:  449993.5714285703143   449993.5714285714202
                       c:  449993.5714285714202   449993.5714285714202
a: dRes: +1101101110111001001.1001001001001001001001001001001001
b: dRes: +1101101110111001001.1001001001001001001001001000110110
c: dRes: +1101101110111001001.1001001001001001001001001001001001
   dReg: +1101101110111001001.1001001001001001001001001001001100
a: dFPU: +1101101110111001001.1001001001001001001001001001001001
b: dFPU: +1101101110111001001.1001001001001001001001001001001001
c: dFPU: +1101101110111001001.1001001001001001001001001001001001

Iteration Number: 46, Value: 459993.4285714285798
         dReg                     dRes                  dFPU
 459993.4285714287544  a:  459993.4285714285798   459993.4285714285798
                       b:  459993.4285714274156   459993.4285714285798
                       c:  459993.4285714285798   459993.4285714285798
a: dRes: +1110000010011011001.0110110110110110110110110110110111
b: dRes: +1110000010011011001.0110110110110110110110110110100011
c: dRes: +1110000010011011001.0110110110110110110110110110110111
   dReg: +1110000010011011001.0110110110110110110110110110111010
a: dFPU: +1110000010011011001.0110110110110110110110110110110111
b: dFPU: +1110000010011011001.0110110110110110110110110110110111
c: dFPU: +1110000010011011001.0110110110110110110110110110110111

Iteration Number: 47, Value: 469993.2857142857392
         dReg                     dRes                  dFPU
 469993.2857142859138  a:  469993.285714285681    469993.2857142857392
                       b:  469993.2857142845751   469993.2857142857392
                       c:  469993.2857142857392   469993.2857142857392
a: dRes: +1110010101111101001.0100100100100100100100100100100100
b: dRes: +1110010101111101001.0100100100100100100100100100010001
c: dRes: +1110010101111101001.0100100100100100100100100100100101
   dReg: +1110010101111101001.0100100100100100100100100100101000
a: dFPU: +1110010101111101001.0100100100100100100100100100100101
b: dFPU: +1110010101111101001.0100100100100100100100100100100101
c: dFPU: +1110010101111101001.0100100100100100100100100100100101

Iteration Number: 48, Value: 479993.1428571428405
         dReg                     dRes                  dFPU
 479993.1428571430734  a:  479993.1428571428405   479993.1428571428405
                       b:  479993.1428571416764   479993.1428571428405
                       c:  479993.1428571428987   479993.1428571428405
a: dRes: +1110101001011111001.0010010010010010010010010010010010
b: dRes: +1110101001011111001.0010010010010010010010010001111110
c: dRes: +1110101001011111001.0010010010010010010010010010010011
   dReg: +1110101001011111001.0010010010010010010010010010010110
a: dFPU: +1110101001011111001.0010010010010010010010010010010010
b: dFPU: +1110101001011111001.0010010010010010010010010010010010
c: dFPU: +1110101001011111001.0010010010010010010010010010010010

Iteration Number: 49, Value: 489993
         dReg                     dRes                  dFPU
 489993.0000000002328  a:         489993                489993
                       b:  489992.9999999987776         489993
                       c:         489993                489993
a: dRes: +1110111101000001001.0000000000000000000000000000000000
b: dRes: +1110111101000001000.1111111111111111111111111111101011
```

```
c: dRes: +1110111101000001001.0000000000000000000000000000000000
   dReg: +1110111101000001001.0000000000000000000000000000000100
a: dFPU: +1110111101000001001.0000000000000000000000000000000000
b: dFPU: +1110111101000001001.0000000000000000000000000000000000
c: dFPU: +1110111101000001001.0000000000000000000000000000000000

Iteration Number: 50, Value: 499992.8571428571595
         dReg                      dRes                   dFPU
 499992.8571428573923  a:  499992.8571428571595   499992.8571428571595
                       b:  499992.8571428558789   499992.8571428571595
                       c:  499992.8571428571013   499992.8571428571595
a: dRes: +1111010000100011000.1101101101101101101101101101101110
b: dRes: +1111010000100011000.1101101101101101101101101101011000
c: dRes: +1111010000100011000.1101101101101101101101101101101101
   dReg: +1111010000100011000.1101101101101101101101101101110010
a: dFPU: +1111010000100011000.1101101101101101101101101101101110
b: dFPU: +1111010000100011000.1101101101101101101101101101101110
c: dFPU: +1111010000100011000.1101101101101101101101101101101110

Iteration Number: 51, Value: 509992.7142857142608
         dReg                      dRes                   dFPU
 509992.7142857145518  a:  509992.714285714319    509992.7142857142608
                       b:  509992.7142857129802   509992.7142857142608
                       c:  509992.7142857142608   509992.7142857142608
a: dRes: +1111100100000101000.1011011011011011011011011011011100
b: dRes: +1111100100000101000.1011011011011011011011011011000101
c: dRes: +1111100100000101000.1011011011011011011011011011011011
   dReg: +1111100100000101000.1011011011011011011011011011100000
a: dFPU: +1111100100000101000.1011011011011011011011011011011011
b: dFPU: +1111100100000101000.1011011011011011011011011011011011
c: dFPU: +1111100100000101000.1011011011011011011011011011011011

Iteration Number: 52, Value: 519992.5714285714202
         dReg                      dRes                   dFPU
 519992.5714285717113  a:  519992.5714285714202   519992.5714285714202
                       b:  519992.5714285701397   519992.5714285714202
                       c:  519992.5714285714202   519992.5714285714202
a: dRes: +1111110111100111000.1001001001001001001001001001001001
b: dRes: +1111110111100111000.1001001001001001001001001000110011
c: dRes: +1111110111100111000.1001001001001001001001001001001001
   dReg: +1111110111100111000.1001001001001001001001001001001110
a: dFPU: +1111110111100111000.1001001001001001001001001001001001
b: dFPU: +1111110111100111000.1001001001001001001001001001001001
c: dFPU: +1111110111100111000.1001001001001001001001001001001001

Iteration Number: 53, Value: 529992.4285714285216
         dReg                      dRes                   dFPU
 529992.4285714288708  a:  529992.428571428638    529992.4285714285216
                       b:  529992.4285714302678   529992.4285714285216
                       c:  529992.4285714285216   529992.4285714285216
a: dRes: +10000001011001001000.011011011011011011011011011011100
b: dRes: +10000001011001001000.011011011011011011011011011101010
c: dRes: +10000001011001001000.011011011011011011011011011011011
   dReg: +10000001011001001000.011011011011011011011011011011110
a: dFPU: +10000001011001001000.011011011011011011011011011011011
b: dFPU: +10000001011001001000.011011011011011011011011011011011
c: dFPU: +10000001011001001000.011011011011011011011011011011011

Iteration Number: 54, Value: 539992.285714285681
         dReg                      dRes                   dFPU
 539992.2857142860302  a:  539992.285714285681    539992.285714285681
                       b:  539992.2857142875436   539992.285714285681
                       c:  539992.2857142857974   539992.285714285681
a: dRes: +10000011110101011000.010010010010010010010010010010010
b: dRes: +10000011110101011000.010010010010010010010010010100010
c: dRes: +10000011110101011000.010010010010010010010010010010011
   dReg: +10000011110101011000.010010010010010010010010010010101
a: dFPU: +10000011110101011000.010010010010010010010010010010010
b: dFPU: +10000011110101011000.010010010010010010010010010010010
c: dFPU: +10000011110101011000.010010010010010010010010010010010
```

```
Iteration Number: 55, Value: 549992.1428571428405
         dReg                     dRes                   dFPU
 549992.1428571431898  a:  549992.1428571428405   549992.1428571428405
                       b:  549992.1428571447032   549992.1428571428405
                       c:  549992.1428571428405   549992.1428571428405
a: dRes: +10000110010001101000.001001001001001001001001001001001
b: dRes: +10000110010001101000.001001001001001001001001001011001
c: dRes: +10000110010001101000.001001001001001001001001001001001
   dReg: +10000110010001101000.001001001001001001001001001001100
a: dFPU: +10000110010001101000.001001001001001001001001001001001
b: dFPU: +10000110010001101000.001001001001001001001001001001001
c: dFPU: +10000110010001101000.001001001001001001001001001001001

Iteration Number: 56, Value: 559992
         dReg                     dRes                   dFPU
 559992.0000000003492  a:         559992                 559992
                       b:  559992.0000000018626          559992
                       c:         559992                 559992
a: dRes: +10001000101101111000.000000000000000000000000000000000
b: dRes: +10001000101101111000.000000000000000000000000000010000
c: dRes: +10001000101101111000.000000000000000000000000000000000
   dReg: +10001000101101111000.000000000000000000000000000000011
a: dFPU: +10001000101101111000.000000000000000000000000000000000
b: dFPU: +10001000101101111000.000000000000000000000000000000000
c: dFPU: +10001000101101111000.000000000000000000000000000000000

Iteration Number: 57, Value: 569991.8571428571595
         dReg                     dRes                   dFPU
 569991.8571428575088  a:  569991.8571428571595   569991.8571428571595
                       b:  569991.8571428590222   569991.8571428571595
                       c:  569991.8571428571595   569991.8571428571595
a: dRes: +10001011001010000111.110110110110110110110110110110111
b: dRes: +10001011001010000111.110110110110110110110110111000111
c: dRes: +10001011001010000111.110110110110110110110110110110111
   dReg: +10001011001010000111.110110110110110110110110110111010
a: dFPU: +10001011001010000111.110110110110110110110110110110111
b: dFPU: +10001011001010000111.110110110110110110110110110110111
c: dFPU: +10001011001010000111.110110110110110110110110110110111

Iteration Number: 58, Value: 579991.714285714319
         dReg                     dRes                   dFPU
 579991.7142857146682  a:  579991.714285714319    579991.714285714319
                       b:  579991.7142857161816   579991.714285714319
                       c:  579991.7142857142026   579991.714285714319
a: dRes: +10001101100110010111.101101101101101101101101101101110
b: dRes: +10001101100110010111.101101101101101101101101101111110
c: dRes: +10001101100110010111.101101101101101101101101101101101
   dReg: +10001101100110010111.101101101101101101101101101110001
a: dFPU: +10001101100110010111.101101101101101101101101101101110
b: dFPU: +10001101100110010111.101101101101101101101101101101110
c: dFPU: +10001101100110010111.101101101101101101101101101101110

Iteration Number: 59, Value: 589991.5714285714784
         dReg                     dRes                   dFPU
 589991.5714285718277  a:  589991.571428571362    589991.5714285714784
                       b:  589991.5714285734575   589991.5714285714784
                       c:  589991.5714285714784   589991.5714285714784
a: dRes: +10010000000010100111.100100100100100100100100100100100
b: dRes: +10010000000010100111.100100100100100100100100100110110
c: dRes: +10010000000010100111.100100100100100100100100100100101
   dReg: +10010000000010100111.100100100100100100100100100101000
a: dFPU: +10010000000010100111.100100100100100100100100100100101
b: dFPU: +10010000000010100111.100100100100100100100100100100101
c: dFPU: +10010000000010100111.100100100100100100100100100100101

Iteration Number: 60, Value: 599991.4285714285216
         dReg                     dRes                   dFPU
 599991.4285714289872  a:  599991.428571428638    599991.4285714285216
                       b:  599991.4285714305006   599991.4285714285216
                       c:  599991.4285714285216   599991.4285714285216
a: dRes: +10010010011110110111.011011011011011011011011011011100
```

```
b: dRes: +10010010011110110111.01101101101101101101101101101100
c: dRes: +10010010011110110111.01101101101101101101101101101011
   dReg: +10010010011110110111.01101101101101101101101101101111
a: dFPU: +10010010011110110111.01101101101101101101101101101011
b: dFPU: +10010010011110110111.01101101101101101101101101101011
c: dFPU: +10010010011110110111.01101101101101101101101101101011

Iteration Number: 61, Value: 609991.285714285681
         dReg                      dRes                   dFPU
 609991.2857142861467  a:  609991.285714285681    609991.285714285681
                       b:  609991.2857142877765   609991.285714285681
                       c:  609991.2857142857974   609991.285714285681
a: dRes: +10010100111011000111.01001001001001001001001001010010
b: dRes: +10010100111011000111.01001001001001001001001001010100
c: dRes: +10010100111011000111.01001001001001001001001001010011
   dReg: +10010100111011000111.01001001001001001001001001010110
a: dFPU: +10010100111011000111.01001001001001001001001001010010
b: dFPU: +10010100111011000111.01001001001001001001001001010010
c: dFPU: +10010100111011000111.01001001001001001001001001010010

Iteration Number: 62, Value: 619991.1428571428405
         dReg                      dRes                   dFPU
 619991.1428571433062  a:  619991.1428571428405   619991.1428571428405
                       b:  619991.142857144936    619991.1428571428405
                       c:  619991.1428571428405   619991.1428571428405
a: dRes: +10010111010111010111.00100100100100100100100100100001
b: dRes: +10010111010111010111.00100100100100100100100100101011
c: dRes: +10010111010111010111.00100100100100100100100100100001
   dReg: +10010111010111010111.00100100100100100100100100100101
a: dFPU: +10010111010111010111.00100100100100100100100100100001
b: dFPU: +10010111010111010111.00100100100100100100100100100001
c: dFPU: +10010111010111010111.00100100100100100100100100100001

Iteration Number: 63, Value: 629991
         dReg                      dRes                   dFPU
 629991.0000000004656  a:         629991                 629991
                       b:  629991.0000000020955          629991
                       c:         629991                 629991
a: dRes: +10011001110011100111.00000000000000000000000000000000
b: dRes: +10011001110011100111.00000000000000000000000000010010
c: dRes: +10011001110011100111.00000000000000000000000000000000
   dReg: +10011001110011100111.00000000000000000000000000000100
a: dFPU: +10011001110011100111.00000000000000000000000000000000
b: dFPU: +10011001110011100111.00000000000000000000000000000000
c: dFPU: +10011001110011100111.00000000000000000000000000000000

Iteration Number: 64, Value: 639990.8571428571595
         dReg                      dRes                   dFPU
 639990.8571428576252  a:  639990.8571428571595   639990.8571428571595
                       b:  639990.857142859255    639990.8571428571595
                       c:  639990.8571428571595   639990.8571428571595
a: dRes: +10011100001111110110.11011011011011011011011011010111
b: dRes: +10011100001111110110.11011011011011011011011011101001
c: dRes: +10011100001111110110.11011011011011011011011011010111
   dReg: +10011100001111110110.11011011011011011011011011011011
a: dFPU: +10011100001111110110.11011011011011011011011011010111
b: dFPU: +10011100001111110110.11011011011011011011011011010111
c: dFPU: +10011100001111110110.11011011011011011011011011010111

Iteration Number: 65, Value: 649990.714285714319
         dReg                      dRes                   dFPU
 649990.7142857147846  a:  649990.714285714319    649990.714285714319
                       b:  649990.7142857164144   649990.714285714319
                       c:  649990.7142857142026   649990.714285714319
a: dRes: +10011110101100000110.10110110110110110110110110101110
b: dRes: +10011110101100000110.10110110110110110110110111000000
c: dRes: +10011110101100000110.10110110110110110110110110101101
   dReg: +10011110101100000110.10110110110110110110110110110010
a: dFPU: +10011110101100000110.10110110110110110110110110101110
b: dFPU: +10011110101100000110.10110110110110110110110110101110
c: dFPU: +10011110101100000110.10110110110110110110110110101110
```

```
Iteration Number: 66, Value: 659990.5714285714784
         dReg                     dRes                   dFPU
 659990.5714285719441  a:  659990.571428571362    659990.5714285714784
                       b:  659990.5714285736904   659990.5714285714784
                       c:  659990.5714285714784   659990.5714285714784
a: dRes: +10100001001000010110.100100100100100100100100100100100
b: dRes: +10100001001000010110.100100100100100100100100100111000
c: dRes: +10100001001000010110.100100100100100100100100100100101
   dReg: +10100001001000010110.100100100100100100100100100101001
a: dFPU: +10100001001000010110.100100100100100100100100100100101
b: dFPU: +10100001001000010110.100100100100100100100100100100101
c: dFPU: +10100001001000010110.100100100100100100100100100100101

Iteration Number: 67, Value: 669990.4285714285216
         dReg                     dRes                   dFPU
 669990.4285714291036  a:  669990.428571428638    669990.4285714285216
                       b:  669990.4285714307334   669990.4285714285216
                       c:  669990.4285714285216   669990.4285714285216
a: dRes: +10100011100100100110.011011011011011011011011011011100
b: dRes: +10100011100100100110.011011011011011011011011011101110
c: dRes: +10100011100100100110.011011011011011011011011011011011
   dReg: +10100011100100100110.011011011011011011011011011100000
a: dFPU: +10100011100100100110.011011011011011011011011011011011
b: dFPU: +10100011100100100110.011011011011011011011011011011011
c: dFPU: +10100011100100100110.011011011011011011011011011011011

Iteration Number: 68, Value: 679990.285714285681
         dReg                     dRes                   dFPU
 679990.2857142862631  a:  679990.285714285681    679990.285714285681
                       b:  679990.2857142880094   679990.285714285681
                       c:  679990.2857142857974   679990.285714285681
a: dRes: +10100110000000110110.010010010010010010010010010010010
b: dRes: +10100110000000110110.010010010010010010010010010100110
c: dRes: +10100110000000110110.010010010010010010010010010010011
   dReg: +10100110000000110110.010010010010010010010010010010111
a: dFPU: +10100110000000110110.010010010010010010010010010010010
b: dFPU: +10100110000000110110.010010010010010010010010010010010
c: dFPU: +10100110000000110110.010010010010010010010010010010010

Iteration Number: 69, Value: 689990.1428571428405
         dReg                     dRes                   dFPU
 689990.1428571434226  a:  689990.1428571428405   689990.1428571428405
                       b:  689990.1428571451688   689990.1428571428405
                       c:  689990.1428571428405   689990.1428571428405
a: dRes: +10101000011101000110.001001001001001001001001001001001
b: dRes: +10101000011101000110.001001001001001001001001001011101
c: dRes: +10101000011101000110.001001001001001001001001001001001
   dReg: +10101000011101000110.001001001001001001001001001001110
a: dFPU: +10101000011101000110.001001001001001001001001001001001
b: dFPU: +10101000011101000110.001001001001001001001001001001001
c: dFPU: +10101000011101000110.001001001001001001001001001001001

Iteration Number: 70, Value: 699990
         dReg                     dRes                   dFPU
 699990.0000000005821  a:         699990                 699990
                       b:  699990.0000000023283          699990
                       c:         699990                 699990
a: dRes: +10101010111001010110.000000000000000000000000000000000
b: dRes: +10101010111001010110.000000000000000000000000000010100
c: dRes: +10101010111001010110.000000000000000000000000000000000
   dReg: +10101010111001010110.000000000000000000000000000000101
a: dFPU: +10101010111001010110.000000000000000000000000000000000
b: dFPU: +10101010111001010110.000000000000000000000000000000000
c: dFPU: +10101010111001010110.000000000000000000000000000000000

Iteration Number: 71, Value: 709989.8571428571595
         dReg                     dRes                   dFPU
 709989.8571428577416  a:  709989.8571428571595   709989.8571428571595
                       b:  709989.8571428594878   709989.8571428571595
                       c:  709989.8571428571595   709989.8571428571595
```

```
a: dRes: +10101101010101100101.11011011011011011011011011011110111
b: dRes: +10101101010101100101.110110110110110110110110111001011
c: dRes: +10101101010101100101.11011011011011011011011011011110111
   dReg: +10101101010101100101.1101101101101101101101101101111100
a: dFPU: +10101101010101100101.11011011011011011011011011011110111
b: dFPU: +10101101010101100101.11011011011011011011011011011110111
c: dFPU: +10101101010101100101.11011011011011011011011011011110111

Iteration Number: 72, Value: 719989.714285714319
         dReg                        dRes                  dFPU
 719989.714285714901   a:  719989.714285714319    719989.714285714319
                       b:  719989.7142857166473   719989.714285714319
                       c:  719989.7142857142026   719989.714285714319
a: dRes: +10101111110001110101.1011011011011011011011011011101110
b: dRes: +10101111110001110101.1011011011011011011011011110000010
c: dRes: +10101111110001110101.1011011011011011011011011011101101
   dReg: +10101111110001110101.1011011011011011011011011101110011
a: dFPU: +10101111110001110101.1011011011011011011011011011101110
b: dFPU: +10101111110001110101.1011011011011011011011011011101110
c: dFPU: +10101111110001110101.1011011011011011011011011011101110

Iteration Number: 73, Value: 729989.5714285714784
         dReg                        dRes                  dFPU
 729989.5714285720606  a:  729989.571428571362    729989.5714285714784
                       b:  729989.5714285739232   729989.5714285714784
                       c:  729989.5714285714784   729989.5714285714784
a: dRes: +10110010001110000101.1001001001001001001001001001100100
b: dRes: +10110010001110000101.1001001001001001001001001001111010
c: dRes: +10110010001110000101.1001001001001001001001001001100101
   dReg: +10110010001110000101.1001001001001001001001001001101010
a: dFPU: +10110010001110000101.1001001001001001001001001001100101
b: dFPU: +10110010001110000101.1001001001001001001001001001100101
c: dFPU: +10110010001110000101.1001001001001001001001001001100101

Iteration Number: 74, Value: 739989.4285714285216
         dReg                        dRes                  dFPU
   739989.42857142922  a:  739989.428571428638    739989.4285714285216
                       b:  739989.4285714309662   739989.4285714285216
                       c:  739989.4285714285216   739989.4285714285216
a: dRes: +10110100101010010101.0110110110110110110110110110111100
b: dRes: +10110100101010010101.0110110110110110110110110111110000
c: dRes: +10110100101010010101.0110110110110110110110110110111011
   dReg: +10110100101010010101.0110110110110110110110110111100001
a: dFPU: +10110100101010010101.0110110110110110110110110110111011
b: dFPU: +10110100101010010101.0110110110110110110110110110111011
c: dFPU: +10110100101010010101.0110110110110110110110110110111011

Iteration Number: 75, Value: 749989.285714285681
         dReg                        dRes                  dFPU
 749989.2857142863795  a:  749989.285714285681    749989.285714285681
                       b:  749989.2857142882422   749989.285714285681
                       c:  749989.2857142857974   749989.285714285681
a: dRes: +10110111000110100101.0100100100100100100100100100010010
b: dRes: +10110111000110100101.0100100100100100100100100101010000
c: dRes: +10110111000110100101.0100100100100100100100100100010011
   dReg: +10110111000110100101.0100100100100100100100100100011000
a: dFPU: +10110111000110100101.0100100100100100100100100100010010
b: dFPU: +10110111000110100101.0100100100100100100100100100010010
c: dFPU: +10110111000110100101.0100100100100100100100100100010010

Iteration Number: 76, Value: 759989.1428571428405
         dReg                        dRes                  dFPU
 759989.142857143539   a:  759989.1428571428405   759989.1428571428405
                       b:  759989.1428571454016   759989.1428571428405
                       c:  759989.1428571428405   759989.1428571428405
a: dRes: +10111001100010110101.0010010010010010010010010010010001
b: dRes: +10111001100010110101.0010010010010010010010010010111111
c: dRes: +10111001100010110101.0010010010010010010010010010010001
   dReg: +10111001100010110101.0010010010010010010010010010011111
a: dFPU: +10111001100010110101.0010010010010010010010010010010001
b: dFPU: +10111001100010110101.0010010010010010010010010010010001
```

```
c: dFPU: +10111001100010110101.001001001001001001001001001001001

Iteration Number: 77, Value: 769989
         dReg                     dRes                   dFPU
 769989.0000000006985  a:         769989                 769989
                       b:  769989.0000000025612          769989
                       c:         769989                 769989
a: dRes: +10111011111111000101.000000000000000000000000000000000
b: dRes: +10111011111111000101.000000000000000000000000000010110
c: dRes: +10111011111111000101.000000000000000000000000000000000
   dReg: +10111011111111000101.000000000000000000000000000000110
a: dFPU: +10111011111111000101.000000000000000000000000000000000
b: dFPU: +10111011111111000101.000000000000000000000000000000000
c: dFPU: +10111011111111000101.000000000000000000000000000000000

Iteration Number: 78, Value: 779988.8571428571595
         dReg                     dRes                   dFPU
 779988.857142857858   a:  779988.8571428571595   779988.8571428571595
                       b:  779988.8571428597206   779988.8571428571595
                       c:  779988.8571428571595   779988.8571428571595
a: dRes: +10111110011011010100.110110110110110110110110110110111
b: dRes: +10111110011011010100.110110110110110110110110111001101
c: dRes: +10111110011011010100.110110110110110110110110110110111
   dReg: +10111110011011010100.110110110110110110110110110111101
a: dFPU: +10111110011011010100.110110110110110110110110110110111
b: dFPU: +10111110011011010100.110110110110110110110110110110111
c: dFPU: +10111110011011010100.110110110110110110110110110110111

Iteration Number: 79, Value: 789988.714285714319
         dReg                     dRes                   dFPU
 789988.7142857150174  a:  789988.714285714319    789988.714285714319
                       b:  789988.7142857168801   789988.714285714319
                       c:  789988.7142857142026   789988.714285714319
a: dRes: +11000000110111100100.101101101101101101101101101101110
b: dRes: +11000000110111100100.101101101101101101101101110000100
c: dRes: +11000000110111100100.101101101101101101101101101101101
   dReg: +11000000110111100100.101101101101101101101101101110100
a: dFPU: +11000000110111100100.101101101101101101101101101101110
b: dFPU: +11000000110111100100.101101101101101101101101101101110
c: dFPU: +11000000110111100100.101101101101101101101101101101110

Iteration Number: 80, Value: 799988.5714285714784
         dReg                     dRes                   dFPU
 799988.571428572177   a:  799988.571428571362    799988.5714285714784
                       b:  799988.571428574156    799988.5714285714784
                       c:  799988.5714285714784   799988.5714285714784
a: dRes: +11000011010011110100.100100100100100100100100100100100
b: dRes: +11000011010011110100.100100100100100100100100100111100
c: dRes: +11000011010011110100.100100100100100100100100100100101
   dReg: +11000011010011110100.100100100100100100100100100101011
a: dFPU: +11000011010011110100.100100100100100100100100100100101
b: dFPU: +11000011010011110100.100100100100100100100100100100101
c: dFPU: +11000011010011110100.100100100100100100100100100100101

Iteration Number: 81, Value: 809988.4285714285216
         dReg                     dRes                   dFPU
 809988.4285714293364  a:  809988.428571428638    809988.4285714285216
                       b:  809988.4285714311991   809988.4285714285216
                       c:  809988.4285714285216   809988.4285714285216
a: dRes: +11000101110000000100.011011011011011011011011011011100
b: dRes: +11000101110000000100.011011011011011011011011011110010
c: dRes: +11000101110000000100.011011011011011011011011011011011
   dReg: +11000101110000000100.011011011011011011011011011100010
a: dFPU: +11000101110000000100.011011011011011011011011011011011
b: dFPU: +11000101110000000100.011011011011011011011011011011011
c: dFPU: +11000101110000000100.011011011011011011011011011011011

Iteration Number: 82, Value: 819988.285714285681
         dReg                     dRes                   dFPU
 819988.285714286496   a:  819988.285714285681    819988.285714285681
                       b:  819988.285714288475    819988.285714285681
                       c:  819988.2857142857974   819988.285714285681
a: dRes: +11001000001100010100.010010010010010010010010010010010
b: dRes: +11001000001100010100.010010010010010010010010010101010
c: dRes: +11001000001100010100.010010010010010010010010010010011
   dReg: +11001000001100010100.010010010010010010010010010011001
a: dFPU: +11001000001100010100.010010010010010010010010010010010
b: dFPU: +11001000001100010100.010010010010010010010010010010010
c: dFPU: +11001000001100010100.010010010010010010010010010010010

Iteration Number: 83, Value: 829988.1428571428405
         dReg                     dRes                   dFPU
 829988.1428571436554  a:  829988.1428571428405   829988.1428571428405
                       b:  829988.1428571456345   829988.1428571428405
                       c:  829988.1428571428405   829988.1428571428405
a: dRes: +11001010101000100100.001001001001001001001001001001001
b: dRes: +11001010101000100100.001001001001001001001001001100001
c: dRes: +11001010101000100100.001001001001001001001001001001001
   dReg: +11001010101000100100.001001001001001001001001001010000
a: dFPU: +11001010101000100100.001001001001001001001001001001001
b: dFPU: +11001010101000100100.001001001001001001001001001001001
c: dFPU: +11001010101000100100.001001001001001001001001001001001

Iteration Number: 84, Value: 839988
         dReg                     dRes                   dFPU
 839988.0000000008149  a:         839988                 839988
                       b:  839988.000000002794           839988
                       c:         839988                 839988
a: dRes: +11001101000100110100.000000000000000000000000000000000
b: dRes: +11001101000100110100.000000000000000000000000000011000
c: dRes: +11001101000100110100.000000000000000000000000000000000
   dReg: +11001101000100110100.000000000000000000000000000000111
a: dFPU: +11001101000100110100.000000000000000000000000000000000
b: dFPU: +11001101000100110100.000000000000000000000000000000000
c: dFPU: +11001101000100110100.000000000000000000000000000000000

Iteration Number: 85, Value: 849987.8571428571595
         dReg                     dRes                   dFPU
 849987.8571428579744  a:  849987.8571428571595   849987.8571428571595
                       b:  849987.8571428599534   849987.8571428571595
                       c:  849987.8571428571595   849987.8571428571595
a: dRes: +11001111100001000011.110110110110110110110110110110111
b: dRes: +11001111100001000011.110110110110110110110110111001111
c: dRes: +11001111100001000011.110110110110110110110110110110111
   dReg: +11001111100001000011.110110110110110110110110110111110
a: dFPU: +11001111100001000011.110110110110110110110110110110111
b: dFPU: +11001111100001000011.110110110110110110110110110110111
c: dFPU: +11001111100001000011.110110110110110110110110110110111

Iteration Number: 86, Value: 859987.714285714319
         dReg                     dRes                   dFPU
 859987.7142857151339  a:  859987.714285714319    859987.714285714319
                       b:  859987.714285717113    859987.714285714319
                       c:  859987.7142857142026   859987.714285714319
a: dRes: +11010001111101010011.101101101101101101101101101101110
b: dRes: +11010001111101010011.101101101101101101101101110000110
c: dRes: +11010001111101010011.101101101101101101101101101101101
   dReg: +11010001111101010011.101101101101101101101101101110101
a: dFPU: +11010001111101010011.101101101101101101101101101101110
b: dFPU: +11010001111101010011.101101101101101101101101101101110
c: dFPU: +11010001111101010011.101101101101101101101101101101110

Iteration Number: 87, Value: 869987.5714285714784
         dReg                     dRes                   dFPU
 869987.5714285722934  a:  869987.571428571362    869987.5714285714784
                       b:  869987.5714285743888   869987.5714285714784
                       c:  869987.5714285714784   869987.5714285714784
a: dRes: +11010100011001100011.100100100100100100100100100100100
b: dRes: +11010100011001100011.100100100100100100100100100111110
c: dRes: +11010100011001100011.100100100100100100100100100100101
   dReg: +11010100011001100011.100100100100100100100100100101100
a: dFPU: +11010100011001100011.100100100100100100100100100100101
```

```
b: dFPU: +11010100011001100011.100100100100100100100100100100101
c: dFPU: +11010100011001100011.100100100100100100100100100100101

Iteration Number: 88, Value: 879987.4285714285216
         dReg                     dRes                  dFPU
 879987.4285714294528  a:  879987.428571428638    879987.4285714285216
                       b:  879987.4285714314319   879987.4285714285216
                       c:  879987.4285714285216   879987.4285714285216
a: dRes: +11010110110101110011.011011011011011011011011011011100
b: dRes: +11010110110101110011.011011011011011011011011011110100
c: dRes: +11010110110101110011.011011011011011011011011011011011
   dReg: +11010110110101110011.011011011011011011011011011100011
a: dFPU: +11010110110101110011.011011011011011011011011011011011
b: dFPU: +11010110110101110011.011011011011011011011011011011011
c: dFPU: +11010110110101110011.011011011011011011011011011011011

Iteration Number: 89, Value: 889987.285714285681
         dReg                     dRes                  dFPU
 889987.2857142866124  a:  889987.285714285681    889987.285714285681
                       b:  889987.2857142887078   889987.285714285681
                       c:  889987.2857142857974   889987.285714285681
a: dRes: +11011001010010000011.010010010010010010010010010010010
b: dRes: +11011001010010000011.010010010010010010010010010101100
c: dRes: +11011001010010000011.010010010010010010010010010010011
   dReg: +11011001010010000011.010010010010010010010010010011010
a: dFPU: +11011001010010000011.010010010010010010010010010010010
b: dFPU: +11011001010010000011.010010010010010010010010010010010
c: dFPU: +11011001010010000011.010010010010010010010010010010010

Iteration Number: 90, Value: 899987.1428571428405
         dReg                     dRes                  dFPU
 899987.1428571437718  a:  899987.1428571428405   899987.1428571428405
                       b:  899987.1428571458673   899987.1428571428405
                       c:  899987.1428571428405   899987.1428571428405
a: dRes: +11011011101110010011.001001001001001001001001001001001
b: dRes: +11011011101110010011.001001001001001001001001001100011
c: dRes: +11011011101110010011.001001001001001001001001001001001
   dReg: +11011011101110010011.001001001001001001001001001010001
a: dFPU: +11011011101110010011.001001001001001001001001001001001
b: dFPU: +11011011101110010011.001001001001001001001001001001001
c: dFPU: +11011011101110010011.001001001001001001001001001001001

Iteration Number: 91, Value: 909987
         dReg                     dRes                  dFPU
 909987.0000000009313  a:         909987                909987
                       b:  909987.0000000030268         909987
                       c:         909987                909987
a: dRes: +11011110001010100011.000000000000000000000000000000000
b: dRes: +11011110001010100011.000000000000000000000000000011010
c: dRes: +11011110001010100011.000000000000000000000000000000000
   dReg: +11011110001010100011.000000000000000000000000000001000
a: dFPU: +11011110001010100011.000000000000000000000000000000000
b: dFPU: +11011110001010100011.000000000000000000000000000000000
c: dFPU: +11011110001010100011.000000000000000000000000000000000

Iteration Number: 92, Value: 919986.8571428571595
         dReg                     dRes                  dFPU
 919986.8571428580908  a:  919986.8571428571595   919986.8571428571595
                       b:  919986.8571428601863   919986.8571428571595
                       c:  919986.8571428571595   919986.8571428571595
a: dRes: +11100000100110110010.110110110110110110110110110110111
b: dRes: +11100000100110110010.110110110110110110110110111010001
c: dRes: +11100000100110110010.110110110110110110110110110110111
   dReg: +11100000100110110010.110110110110110110110110110111111
a: dFPU: +11100000100110110010.110110110110110110110110110110111
b: dFPU: +11100000100110110010.110110110110110110110110110110111
c: dFPU: +11100000100110110010.110110110110110110110110110110111

Iteration Number: 93, Value: 929986.714285714319
         dReg                     dRes                  dFPU
 929986.7142857152503  a:  929986.714285714319    929986.714285714319
```

```
                       b:  929986.7142857173458   929986.714285714319
                       c:  929986.7142857142026   929986.714285714319
a: dRes: +11100011000011000010.101101101101101101101101101101110
b: dRes: +11100011000011000010.101101101101101101101101110001000
c: dRes: +11100011000011000010.101101101101101101101101101101101
   dReg: +11100011000011000010.101101101101101101101101101110110
a: dFPU: +11100011000011000010.101101101101101101101101101101110
b: dFPU: +11100011000011000010.101101101101101101101101101101110
c: dFPU: +11100011000011000010.101101101101101101101101101101110

Iteration Number: 94, Value: 939986.5714285714785
         dReg                     dRes                   dFPU
 939986.5714285724098  a:  939986.571428571362    939986.5714285714785
                       b:  939986.5714285746217   939986.5714285714785
                       c:  939986.5714285714785   939986.5714285714785
a: dRes: +11100101011111010010.100100100100100100100100100100100
b: dRes: +11100101011111010010.100100100100100100100100101000000
c: dRes: +11100101011111010010.100100100100100100100100100100101
   dReg: +11100101011111010010.100100100100100100100100100101101
a: dFPU: +11100101011111010010.100100100100100100100100100100101
b: dFPU: +11100101011111010010.100100100100100100100100100100101
c: dFPU: +11100101011111010010.100100100100100100100100100100101

Iteration Number: 95, Value: 949986.4285714285215
         dReg                     dRes                   dFPU
 949986.4285714295693  a:  949986.428571428638    949986.4285714285215
                       b:  949986.4285714316647   949986.4285714285215
                       c:  949986.4285714285215   949986.4285714285215
a: dRes: +11100111111011100010.011011011011011011011011011011100
b: dRes: +11100111111011100010.011011011011011011011011011110110
c: dRes: +11100111111011100010.011011011011011011011011011011011
   dReg: +11100111111011100010.011011011011011011011011011100100
a: dFPU: +11100111111011100010.011011011011011011011011011011011
b: dFPU: +11100111111011100010.011011011011011011011011011011011
c: dFPU: +11100111111011100010.011011011011011011011011011011011

Iteration Number: 96, Value: 959986.285714285681
         dReg                     dRes                   dFPU
 959986.2857142867288  a:  959986.285714285681    959986.285714285681
                       b:  959986.2857142889407   959986.285714285681
                       c:  959986.2857142857974   959986.285714285681
a: dRes: +11101010010111110010.010010010010010010010010010010010
b: dRes: +11101010010111110010.010010010010010010010010010101110
c: dRes: +11101010010111110010.010010010010010010010010010010011
   dReg: +11101010010111110010.010010010010010010010010010011011
a: dFPU: +11101010010111110010.010010010010010010010010010010010
b: dFPU: +11101010010111110010.010010010010010010010010010010010
c: dFPU: +11101010010111110010.010010010010010010010010010010010

Iteration Number: 97, Value: 969986.1428571428405
         dReg                     dRes                   dFPU
 969986.1428571438882  a:  969986.1428571428405   969986.1428571428405
                       b:  969986.1428571461001   969986.1428571428405
                       c:  969986.1428571428405   969986.1428571428405
a: dRes: +11101100110100000010.001001001001001001001001001001001
b: dRes: +11101100110100000010.001001001001001001001001001100101
c: dRes: +11101100110100000010.001001001001001001001001001001001
   dReg: +11101100110100000010.001001001001001001001001001010010
a: dFPU: +11101100110100000010.001001001001001001001001001001001
b: dFPU: +11101100110100000010.001001001001001001001001001001001
c: dFPU: +11101100110100000010.001001001001001001001001001001001

Iteration Number: 98, Value: 979986
         dReg                     dRes                   dFPU
 979986.0000000010477  a:         979986                 979986
                       b:  979986.0000000032596          979986
                       c:         979986                 979986
a: dRes: +11101111010000010010.000000000000000000000000000000000
b: dRes: +11101111010000010010.000000000000000000000000000011100
c: dRes: +11101111010000010010.000000000000000000000000000000000
   dReg: +11101111010000010010.000000000000000000000000000001001
```

```
a: dFPU: +11101111010000010010.000000000000000000000000000000000
b: dFPU: +11101111010000010010.000000000000000000000000000000000
c: dFPU: +11101111010000010010.000000000000000000000000000000000

Iteration Number: 99, Value: 989985.8571428571595
         dReg                     dRes                   dFPU
 989985.8571428582072  a:  989985.8571428571595   989985.8571428571595
                       b:  989985.8571428604191   989985.8571428571595
                       c:  989985.8571428571595   989985.8571428571595
a: dRes: +11110001101100100001.110110110110110110110110110110111
b: dRes: +11110001101100100001.110110110110110110110110110111010011
c: dRes: +11110001101100100001.110110110110110110110110110110111
   dReg: +11110001101100100001.110110110110110110110110111000000
a: dFPU: +11110001101100100001.110110110110110110110110110110111
b: dFPU: +11110001101100100001.110110110110110110110110110110111
c: dFPU: +11110001101100100001.110110110110110110110110110110111

Iteration Number: 100, Value: 999985.714285714319
         dReg                     dRes                   dFPU
 999985.7142857153667  a:  999985.714285714319    999985.714285714319
                       b:  999985.7142857175786   999985.714285714319
                       c:  999985.7142857142026   999985.714285714319
a: dRes: +11110100001000110001.101101101101101101101101101101110
b: dRes: +11110100001000110001.101101101101101101101101110001010
c: dRes: +11110100001000110001.101101101101101101101101101101101
   dReg: +11110100001000110001.101101101101101101101101101110111
a: dFPU: +11110100001000110001.101101101101101101101101101101110
b: dFPU: +11110100001000110001.101101101101101101101101101101110
c: dFPU: +11110100001000110001.101101101101101101101101101101110
```

Appendix AC- CHFort Object Code of Test Case Three

```
<<variablesvalues>>
Name: dYSum
Value:
IsHistory: 1
IsFixed: 0
DataType: 5
nDimsCount: 0
Name: dXMid
Value:
IsHistory: 1
IsFixed: 0
DataType: 5
nDimsCount: 0
Name: dX0
Value:
IsHistory: 0
IsFixed: 0
DataType: 5
nDimsCount: 0
Name: dSpan
Value:
IsHistory: 0
IsFixed: 0
DataType: 5
nDimsCount: 0
Name: dSpanIncs
Value:
IsHistory: 0
IsFixed: 0
DataType: 5
nDimsCount: 0
Name: dSpanDelta
Value:
IsHistory: 0
IsFixed: 0
DataType: 5
nDimsCount: 0
<<ProgramCode>>
Label StartProg
BeginCode dX0
push Number 0
EndCode
BeginCode nCnt
push Number 0
EndCode
BeginCode dSpan
push Number 1
EndCode
BeginCode dSpanIncs
push Number 199
EndCode
BeginCode dSpanDelta
  push String dSpan
  push String dSpanIncs
  DivSingle
EndCode
BeginCode dXMid0
  push String dX0
  push String dSpanDelta
  push Number 2
  DivSingle
  Plus
EndCode
BeginCode dXMid
push String dXMid0
EndCode
BeginCode dYSum
push Number 1
```

```
EndCode
Label 100
BeginBoolCode dIf_Bool_1_0
  push String nCnt
  push Number 30
  Minus
EndCode
GoToCond 999 GreaterOrEqual dIf_Bool_1_0
BeginCode dYSum
  push String dYSum
  push String dXmid
  StarSingle
EndCode
BeginCode dXMid
  push String dXMid
  push String dSpanDelta
  StarSingle
EndCode
BeginCode nCnt
  push String nCnt
  push Number 1
  Plus
EndCode
GoTo 100
Label 999
Label EndProg
```

Appendix AD-1- CHFort Object Code of Test Case Four, Experiment One

```
<<variablesvalues>>
Name: Y
Value:
IsHistory: 1
IsFixed: 0
DataType: 5
nDimsCount: 0
<<ProgramCode>>
Label StartProg
BeginCode A
push Number 1E-18
EndCode
BeginCode Y
push Number 1
EndCode
BeginCode N
push Number 0
EndCode
Label 100
BeginBoolCode dIf_Bool_1_0
  push String N
  push Number 100
  Minus
EndCode
GoToCond 900 GreaterOrEqual dIf_Bool_1_0
BeginCode Y
  push String Y
  push String A
  Plus
EndCode
BeginCode N
  push String N
  push Number 1
  Plus
EndCode
GoTo 100
Label 900
Label EndProg
```

Appendix AD-2 Results of Test Case Four, Experiment Two

```
Iteration Number: 1, dFrac 1.0000000000000000715e-17
         dReg                   dRes                   dFPU
          1                      1                      1
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000000000
dFPU: +1.0000000000000000000000000000000000000000000000000000

Iteration Number: 2, dFrac 2.000000000000000143e-17
         dReg                   dRes                   dFPU
          1                      1                      1
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000000000
dFPU: +1.0000000000000000000000000000000000000000000000000000

Iteration Number: 3, dFrac 3.00000000000000006e-17
         dReg                   dRes                   dFPU
          1                      1                      1
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000000000
dFPU: +1.0000000000000000000000000000000000000000000000000000

Iteration Number: 4, dFrac 4.000000000000000286e-17
         dReg                   dRes                   dFPU
          1                      1                      1
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000000000
dFPU: +1.0000000000000000000000000000000000000000000000000000

Iteration Number: 5, dFrac 5.000000000000000512e-17
         dReg                   dRes                   dFPU
          1                      1                      1
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000000000
dFPU: +1.0000000000000000000000000000000000000000000000000000

Iteration Number: 6, dFrac 6.000000000000000121e-17
         dReg                   dRes                   dFPU
          1                      1                      1
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000000000
dFPU: +1.0000000000000000000000000000000000000000000000000000

Iteration Number: 7, dFrac 7.000000000000000346e-17
         dReg                   dRes                   dFPU
          1                      1                      1
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000000000
dFPU: +1.0000000000000000000000000000000000000000000000000000

Iteration Number: 8, dFrac 8.000000000000000572e-17
         dReg                   dRes                   dFPU
          1                      1                      1
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000000000
dFPU: +1.0000000000000000000000000000000000000000000000000000

Iteration Number: 9, dFrac 9.000000000000000798e-17
         dReg                   dRes                   dFPU
          1                      1                      1
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000000000
dFPU: +1.0000000000000000000000000000000000000000000000000000

Iteration Number: 10, dFrac 1.0000000000000001023e-16
         dReg                   dRes                   dFPU
          1                      1                      1
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000000000
```

```
dFPU: +1.0000000000000000000000000000000000000000000000000000

Iteration Number: 11, dFrac 1.100000000000000125e-16
         dReg                   dRes                   dFPU
          1                      1                      1
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000000000
dFPU: +1.0000000000000000000000000000000000000000000000000000

Iteration Number: 12, dFrac 1.200000000000000024e-16
         dReg                   dRes                   dFPU
          1             1.000000000000000222   1.000000000000000222
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000000001
dFPU: +1.0000000000000000000000000000000000000000000000000001

Iteration Number: 13, dFrac 1.30000000000000017e-16
         dReg                   dRes                   dFPU
          1             1.000000000000000222   1.000000000000000222
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000000001
dFPU: +1.0000000000000000000000000000000000000000000000000001

Iteration Number: 14, dFrac 1.400000000000000069e-16
         dReg                   dRes                   dFPU
          1             1.000000000000000222   1.000000000000000222
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000000001
dFPU: +1.0000000000000000000000000000000000000000000000000001

Iteration Number: 15, dFrac 1.500000000000000215e-16
         dReg                   dRes                   dFPU
          1             1.000000000000000222   1.000000000000000222
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000000001
dFPU: +1.0000000000000000000000000000000000000000000000000001

Iteration Number: 16, dFrac 1.600000000000000114e-16
         dReg                   dRes                   dFPU
          1             1.000000000000000222   1.000000000000000222
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000000001
dFPU: +1.0000000000000000000000000000000000000000000000000001

Iteration Number: 17, dFrac 1.700000000000000014e-16
         dReg                   dRes                   dFPU
          1             1.000000000000000222   1.000000000000000222
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000000001
dFPU: +1.0000000000000000000000000000000000000000000000000001

Iteration Number: 18, dFrac 1.80000000000000016e-16
         dReg                   dRes                   dFPU
          1             1.000000000000000222   1.000000000000000222
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000000001
dFPU: +1.0000000000000000000000000000000000000000000000000001

Iteration Number: 19, dFrac 1.900000000000000059e-16
         dReg                   dRes                   dFPU
          1             1.000000000000000222   1.000000000000000222
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000000001
dFPU: +1.0000000000000000000000000000000000000000000000000001

Iteration Number: 20, dFrac 2.000000000000000205e-16
         dReg                   dRes                   dFPU
          1             1.000000000000000222   1.000000000000000222
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000000001
dFPU: +1.0000000000000000000000000000000000000000000000000001
```

```
Iteration Number: 21, dFrac 2.100000000000000104e-16
         dReg                   dRes                   dFPU
          1             1.000000000000000222   1.000000000000000222
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000000001
dFPU: +1.0000000000000000000000000000000000000000000000000001

Iteration Number: 22, dFrac 2.20000000000000025e-16
         dReg                   dRes                   dFPU
          1             1.000000000000000222   1.000000000000000222
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000000001
dFPU: +1.0000000000000000000000000000000000000000000000000001

Iteration Number: 23, dFrac 2.300000000000000396e-16
         dReg                   dRes                   dFPU
          1             1.000000000000000222   1.000000000000000222
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000000001
dFPU: +1.0000000000000000000000000000000000000000000000000001

Iteration Number: 24, dFrac 2.400000000000000048e-16
         dReg                   dRes                   dFPU
          1             1.000000000000000222   1.000000000000000222
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000000001
dFPU: +1.0000000000000000000000000000000000000000000000000001

Iteration Number: 25, dFrac 2.500000000000000194e-16
         dReg                   dRes                   dFPU
          1             1.000000000000000222   1.000000000000000222
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000000001
dFPU: +1.0000000000000000000000000000000000000000000000000001

Iteration Number: 26, dFrac 2.60000000000000034e-16
         dReg                   dRes                   dFPU
          1             1.000000000000000222   1.000000000000000222
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000000001
dFPU: +1.0000000000000000000000000000000000000000000000000001

Iteration Number: 27, dFrac 2.699999999999999993e-16
         dReg                   dRes                   dFPU
          1             1.000000000000000222   1.000000000000000222
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000000001
dFPU: +1.0000000000000000000000000000000000000000000000000001

Iteration Number: 28, dFrac 2.800000000000000139e-16
         dReg                   dRes                   dFPU
          1             1.000000000000000222   1.000000000000000222
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000000001
dFPU: +1.0000000000000000000000000000000000000000000000000001

Iteration Number: 29, dFrac 2.900000000000000284e-16
         dReg                   dRes                   dFPU
          1             1.000000000000000222   1.000000000000000222
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000000001
dFPU: +1.0000000000000000000000000000000000000000000000000001

Iteration Number: 30, dFrac 3.00000000000000043e-16
         dReg                   dRes                   dFPU
          1             1.000000000000000222   1.000000000000000222
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000000001
dFPU: +1.0000000000000000000000000000000000000000000000000001
```

```
Iteration Number: 31, dFrac 3.100000000000000083e-16
         dReg                   dRes                   dFPU
          1             1.000000000000000222   1.000000000000000222
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000000001
dFPU: +1.0000000000000000000000000000000000000000000000000001

Iteration Number: 32, dFrac 3.200000000000000229e-16
         dReg                   dRes                   dFPU
          1             1.000000000000000222   1.000000000000000222
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000000001
dFPU: +1.0000000000000000000000000000000000000000000000000001

Iteration Number: 33, dFrac 3.300000000000000375e-16
         dReg                   dRes                   dFPU
          1             1.000000000000000222   1.000000000000000222
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000000001
dFPU: +1.0000000000000000000000000000000000000000000000000001

Iteration Number: 34, dFrac 3.400000000000000028e-16
         dReg                   dRes                   dFPU
          1            1.0000000000000004441  1.0000000000000004441
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000000010
dFPU: +1.0000000000000000000000000000000000000000000000000010

Iteration Number: 35, dFrac 3.500000000000000174e-16
         dReg                   dRes                   dFPU
          1            1.0000000000000004441  1.0000000000000004441
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000000010
dFPU: +1.0000000000000000000000000000000000000000000000000010

Iteration Number: 36, dFrac 3.600000000000000319e-16
         dReg                   dRes                   dFPU
          1            1.0000000000000004441  1.0000000000000004441
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000000010
dFPU: +1.0000000000000000000000000000000000000000000000000010

Iteration Number: 37, dFrac 3.700000000000000465e-16
         dReg                   dRes                   dFPU
          1            1.0000000000000004441  1.0000000000000004441
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000000010
dFPU: +1.0000000000000000000000000000000000000000000000000010

Iteration Number: 38, dFrac 3.800000000000000118e-16
         dReg                   dRes                   dFPU
          1            1.0000000000000004441  1.0000000000000004441
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000000010
dFPU: +1.0000000000000000000000000000000000000000000000000010

Iteration Number: 39, dFrac 3.900000000000000264e-16
         dReg                   dRes                   dFPU
          1            1.0000000000000004441  1.0000000000000004441
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000000010
dFPU: +1.0000000000000000000000000000000000000000000000000010

Iteration Number: 40, dFrac 4.00000000000000041e-16
         dReg                   dRes                   dFPU
          1            1.0000000000000004441  1.0000000000000004441
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000000010
dFPU: +1.0000000000000000000000000000000000000000000000000010

Iteration Number: 41, dFrac 4.100000000000000062e-16
```

```
         dReg                   dRes                   dFPU
          1            1.0000000000000004441  1.0000000000000004441
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000000010
dFPU: +1.0000000000000000000000000000000000000000000000000010

Iteration Number: 42, dFrac 4.200000000000000208e-16
         dReg                   dRes                   dFPU
          1            1.0000000000000004441  1.0000000000000004441
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000000010
dFPU: +1.0000000000000000000000000000000000000000000000000010

Iteration Number: 43, dFrac 4.300000000000000354e-16
         dReg                   dRes                   dFPU
          1            1.0000000000000004441  1.0000000000000004441
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000000010
dFPU: +1.0000000000000000000000000000000000000000000000000010

Iteration Number: 44, dFrac 4.4000000000000005e-16
         dReg                   dRes                   dFPU
          1            1.0000000000000004441  1.0000000000000004441
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000000010
dFPU: +1.0000000000000000000000000000000000000000000000000010

Iteration Number: 45, dFrac 4.500000000000000152e-16
         dReg                   dRes                   dFPU
          1            1.0000000000000004441  1.0000000000000004441
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000000010
dFPU: +1.0000000000000000000000000000000000000000000000000010

Iteration Number: 46, dFrac 4.600000000000000792e-16
         dReg                   dRes                   dFPU
          1            1.0000000000000004441  1.0000000000000004441
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000000010
dFPU: +1.0000000000000000000000000000000000000000000000000010

Iteration Number: 47, dFrac 4.700000000000000444e-16
         dReg                   dRes                   dFPU
          1            1.0000000000000004441  1.0000000000000004441
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000000010
dFPU: +1.0000000000000000000000000000000000000000000000000010

Iteration Number: 48, dFrac 4.800000000000000097e-16
         dReg                   dRes                   dFPU
          1            1.0000000000000004441  1.0000000000000004441
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000000010
dFPU: +1.0000000000000000000000000000000000000000000000000010

Iteration Number: 49, dFrac 4.900000000000000736e-16
         dReg                   dRes                   dFPU
          1            1.0000000000000004441  1.0000000000000004441
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000000010
dFPU: +1.0000000000000000000000000000000000000000000000000010

Iteration Number: 50, dFrac 5.000000000000000388e-16
         dReg                   dRes                   dFPU
          1            1.0000000000000004441  1.0000000000000004441
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000000010
dFPU: +1.0000000000000000000000000000000000000000000000000010

Iteration Number: 51, dFrac 5.100000000000000042e-16
         dReg                   dRes                   dFPU
```

```
          1            1.0000000000000004441  1.0000000000000004441
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000000010
dFPU: +1.0000000000000000000000000000000000000000000000000010

Iteration Number: 52, dFrac 5.20000000000000068e-16
         dReg                   dRes                   dFPU
          1            1.0000000000000004441  1.0000000000000004441
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000000010
dFPU: +1.0000000000000000000000000000000000000000000000000010

Iteration Number: 53, dFrac 5.300000000000000333e-16
         dReg                   dRes                   dFPU
          1            1.0000000000000004441  1.0000000000000004441
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000000010
dFPU: +1.0000000000000000000000000000000000000000000000000010

Iteration Number: 54, dFrac 5.399999999999999986e-16
         dReg                   dRes                   dFPU
          1            1.0000000000000004441  1.0000000000000004441
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000000010
dFPU: +1.0000000000000000000000000000000000000000000000000010

Iteration Number: 55, dFrac 5.500000000000000624e-16
         dReg                   dRes                   dFPU
          1            1.0000000000000004441  1.0000000000000004441
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000000010
dFPU: +1.0000000000000000000000000000000000000000000000000010

Iteration Number: 56, dFrac 5.600000000000000278e-16
         dReg                   dRes                   dFPU
          1            1.0000000000000006661  1.0000000000000006661
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000000011
dFPU: +1.0000000000000000000000000000000000000000000000000011

Iteration Number: 57, dFrac 5.69999999999999993e-16
         dReg                   dRes                   dFPU
          1            1.0000000000000006661  1.0000000000000006661
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000000011
dFPU: +1.0000000000000000000000000000000000000000000000000011

Iteration Number: 58, dFrac 5.800000000000000569e-16
         dReg                   dRes                   dFPU
          1            1.0000000000000006661  1.0000000000000006661
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000000011
dFPU: +1.0000000000000000000000000000000000000000000000000011

Iteration Number: 59, dFrac 5.900000000000000222e-16
         dReg                   dRes                   dFPU
          1            1.0000000000000006661  1.0000000000000006661
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000000011
dFPU: +1.0000000000000000000000000000000000000000000000000011

Iteration Number: 60, dFrac 6.00000000000000086e-16
         dReg                   dRes                   dFPU
          1            1.0000000000000006661  1.0000000000000006661
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000000011
dFPU: +1.0000000000000000000000000000000000000000000000000011

Iteration Number: 61, dFrac 6.100000000000000514e-16
         dReg                   dRes                   dFPU
          1            1.0000000000000006661  1.0000000000000006661
```

```
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000000011
dFPU: +1.0000000000000000000000000000000000000000000000000011

Iteration Number: 62, dFrac 6.200000000000000166e-16
         dReg                   dRes                   dFPU
          1            1.0000000000000006661  1.0000000000000006661
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000000011
dFPU: +1.0000000000000000000000000000000000000000000000000011

Iteration Number: 63, dFrac 6.300000000000000805e-16
         dReg                   dRes                   dFPU
          1            1.0000000000000006661  1.0000000000000006661
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000000011
dFPU: +1.0000000000000000000000000000000000000000000000000011

Iteration Number: 64, dFrac 6.400000000000000458e-16
         dReg                   dRes                   dFPU
          1            1.0000000000000006661  1.0000000000000006661
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000000011
dFPU: +1.0000000000000000000000000000000000000000000000000011

Iteration Number: 65, dFrac 6.50000000000000011e-16
         dReg                   dRes                   dFPU
          1            1.0000000000000006661  1.0000000000000006661
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000000011
dFPU: +1.0000000000000000000000000000000000000000000000000011

Iteration Number: 66, dFrac 6.60000000000000075e-16
         dReg                   dRes                   dFPU
          1            1.0000000000000006661  1.0000000000000006661
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000000011
dFPU: +1.0000000000000000000000000000000000000000000000000011

Iteration Number: 67, dFrac 6.700000000000000402e-16
         dReg                   dRes                   dFPU
          1            1.0000000000000006661  1.0000000000000006661
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000000011
dFPU: +1.0000000000000000000000000000000000000000000000000011

Iteration Number: 68, dFrac 6.800000000000000055e-16
         dReg                   dRes                   dFPU
          1            1.0000000000000006661  1.0000000000000006661
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000000011
dFPU: +1.0000000000000000000000000000000000000000000000000011

Iteration Number: 69, dFrac 6.900000000000000694e-16
         dReg                   dRes                   dFPU
          1            1.0000000000000006661  1.0000000000000006661
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000000011
dFPU: +1.0000000000000000000000000000000000000000000000000011

Iteration Number: 70, dFrac 7.000000000000000347e-16
         dReg                   dRes                   dFPU
          1            1.0000000000000006661  1.0000000000000006661
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000000011
dFPU: +1.0000000000000000000000000000000000000000000000000011

Iteration Number: 71, dFrac 7.100000000000000986e-16
         dReg                   dRes                   dFPU
          1            1.0000000000000006661  1.0000000000000006661
dReg: +1.0000000000000000000000000000000000000000000000000000
```

```
dRes: +1.0000000000000000000000000000000000000000000000000011
dFPU: +1.0000000000000000000000000000000000000000000000000011

Iteration Number: 72, dFrac 7.200000000000000638e-16
         dReg                   dRes                   dFPU
          1            1.0000000000000006661  1.0000000000000006661
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000000011
dFPU: +1.0000000000000000000000000000000000000000000000000011

Iteration Number: 73, dFrac 7.300000000000000291e-16
         dReg                   dRes                   dFPU
          1            1.0000000000000006661  1.0000000000000006661
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000000011
dFPU: +1.0000000000000000000000000000000000000000000000000011

Iteration Number: 74, dFrac 7.40000000000000093e-16
         dReg                   dRes                   dFPU
          1            1.0000000000000006661  1.0000000000000006661
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000000011
dFPU: +1.0000000000000000000000000000000000000000000000000011

Iteration Number: 75, dFrac 7.500000000000000583e-16
         dReg                   dRes                   dFPU
          1            1.0000000000000006661  1.0000000000000006661
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000000011
dFPU: +1.0000000000000000000000000000000000000000000000000011

Iteration Number: 76, dFrac 7.600000000000000236e-16
         dReg                   dRes                   dFPU
          1            1.0000000000000006661  1.0000000000000006661
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000000011
dFPU: +1.0000000000000000000000000000000000000000000000000011

Iteration Number: 77, dFrac 7.700000000000000874e-16
         dReg                   dRes                   dFPU
          1            1.0000000000000006661  1.0000000000000006661
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000000011
dFPU: +1.0000000000000000000000000000000000000000000000000011

Iteration Number: 78, dFrac 7.800000000000000528e-16
         dReg                   dRes                   dFPU
          1            1.0000000000000008882  1.0000000000000008882
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000000100
dFPU: +1.0000000000000000000000000000000000000000000000000100

Iteration Number: 79, dFrac 7.90000000000000018e-16
         dReg                   dRes                   dFPU
          1            1.0000000000000008882  1.0000000000000008882
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000000100
dFPU: +1.0000000000000000000000000000000000000000000000000100

Iteration Number: 80, dFrac 8.000000000000000819e-16
         dReg                   dRes                   dFPU
          1            1.0000000000000008882  1.0000000000000008882
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000000100
dFPU: +1.0000000000000000000000000000000000000000000000000100

Iteration Number: 81, dFrac 8.100000000000000472e-16
         dReg                   dRes                   dFPU
          1            1.0000000000000008882  1.0000000000000008882
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000000100
```

```
dFPU: +1.0000000000000000000000000000000000000000000000000100

Iteration Number: 82, dFrac 8.200000000000000124e-16
         dReg                   dRes                   dFPU
          1            1.0000000000000008882  1.0000000000000008882
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000000100
dFPU: +1.0000000000000000000000000000000000000000000000000100

Iteration Number: 83, dFrac 8.300000000000000764e-16
         dReg                   dRes                   dFPU
          1            1.0000000000000008882  1.0000000000000008882
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000000100
dFPU: +1.0000000000000000000000000000000000000000000000000100

Iteration Number: 84, dFrac 8.400000000000000416e-16
         dReg                   dRes                   dFPU
          1            1.0000000000000008882  1.0000000000000008882
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000000100
dFPU: +1.0000000000000000000000000000000000000000000000000100

Iteration Number: 85, dFrac 8.500000000000001055e-16
         dReg                   dRes                   dFPU
          1            1.0000000000000008882  1.0000000000000008882
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000000100
dFPU: +1.0000000000000000000000000000000000000000000000000100

Iteration Number: 86, dFrac 8.600000000000000708e-16
         dReg                   dRes                   dFPU
          1            1.0000000000000008882  1.0000000000000008882
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000000100
dFPU: +1.0000000000000000000000000000000000000000000000000100

Iteration Number: 87, dFrac 8.70000000000000036e-16
         dReg                   dRes                   dFPU
          1            1.0000000000000008882  1.0000000000000008882
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000000100
dFPU: +1.0000000000000000000000000000000000000000000000000100

Iteration Number: 88, dFrac 8.800000000000001e-16
         dReg                   dRes                   dFPU
          1            1.0000000000000008882  1.0000000000000008882
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000000100
dFPU: +1.0000000000000000000000000000000000000000000000000100

Iteration Number: 89, dFrac 8.900000000000000652e-16
         dReg                   dRes                   dFPU
          1            1.0000000000000008882  1.0000000000000008882
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000000100
dFPU: +1.0000000000000000000000000000000000000000000000000100

Iteration Number: 90, dFrac 9.000000000000000305e-16
         dReg                   dRes                   dFPU
          1            1.0000000000000008882  1.0000000000000008882
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000000100
dFPU: +1.0000000000000000000000000000000000000000000000000100

Iteration Number: 91, dFrac 9.099999999999999958e-16
         dReg                   dRes                   dFPU
          1            1.0000000000000008882  1.0000000000000008882
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000000100
dFPU: +1.0000000000000000000000000000000000000000000000000100
```

```
Iteration Number: 92, dFrac 9.200000000000001583e-16
         dReg                   dRes                   dFPU
          1            1.0000000000000008882  1.0000000000000008882
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000000100
dFPU: +1.0000000000000000000000000000000000000000000000000100

Iteration Number: 93, dFrac 9.300000000000001236e-16
         dReg                   dRes                   dFPU
          1            1.0000000000000008882  1.0000000000000008882
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000000100
dFPU: +1.0000000000000000000000000000000000000000000000000100

Iteration Number: 94, dFrac 9.400000000000000888e-16
         dReg                   dRes                   dFPU
          1            1.0000000000000008882  1.0000000000000008882
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000000100
dFPU: +1.0000000000000000000000000000000000000000000000000100

Iteration Number: 95, dFrac 9.500000000000000541e-16
         dReg                   dRes                   dFPU
          1            1.0000000000000008882  1.0000000000000008882
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000000100
dFPU: +1.0000000000000000000000000000000000000000000000000100

Iteration Number: 96, dFrac 9.600000000000000194e-16
         dReg                   dRes                   dFPU
          1            1.0000000000000008882  1.0000000000000008882
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000000100
dFPU: +1.0000000000000000000000000000000000000000000000000100

Iteration Number: 97, dFrac 9.699999999999999847e-16
         dReg                   dRes                   dFPU
          1            1.0000000000000008882  1.0000000000000008882
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000000100
dFPU: +1.0000000000000000000000000000000000000000000000000100

Iteration Number: 98, dFrac 9.800000000000001472e-16
         dReg                   dRes                   dFPU
          1            1.0000000000000008882  1.0000000000000008882
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000000100
dFPU: +1.0000000000000000000000000000000000000000000000000100

Iteration Number: 99, dFrac 9.900000000000001124e-16
         dReg                   dRes                   dFPU
          1            1.0000000000000008882  1.0000000000000008882
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000000100
dFPU: +1.0000000000000000000000000000000000000000000000000100

Iteration Number: 100, dFrac 1.000000000000000777e-15
         dReg                   dRes                   dFPU
          1            1.0000000000000011102  1.0000000000000011102
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000000101
dFPU: +1.0000000000000000000000000000000000000000000000000101
```

Appendix AD-3 Results of Test Case Four, Experiment Three

```
Iteration Number: 1, dFrac: 9.999999999999999791e-17
         dReg                     dRes                   dFPU
          1                      1                      1
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000000000
dFPU: +1.0000000000000000000000000000000000000000000000000000

Iteration Number: 2, dFrac: 1.999999999999999958e-16
         dReg                     dRes                   dFPU
          1                      1             1.000000000000000222
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000000000
dFPU: +1.0000000000000000000000000000000000000000000000000001

Iteration Number: 3, dFrac: 2.999999999999999937e-16
         dReg                     dRes                   dFPU
          1             1.000000000000000222   1.000000000000000222
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000000001
dFPU: +1.0000000000000000000000000000000000000000000000000001

Iteration Number: 4, dFrac: 3.999999999999999916e-16
         dReg                     dRes                   dFPU
          1            1.0000000000000004441  1.0000000000000004441
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000000010
dFPU: +1.0000000000000000000000000000000000000000000000000010

Iteration Number: 5, dFrac: 5.000000000000000388e-16
         dReg                     dRes                   dFPU
          1            1.0000000000000004441  1.0000000000000004441
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000000010
dFPU: +1.0000000000000000000000000000000000000000000000000010

Iteration Number: 6, dFrac: 5.999999999999999874e-16
         dReg                     dRes                   dFPU
          1            1.0000000000000006661  1.0000000000000006661
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000000011
dFPU: +1.0000000000000000000000000000000000000000000000000011

Iteration Number: 7, dFrac: 6.99999999999999936e-16
         dReg                     dRes                   dFPU
          1            1.0000000000000006661  1.0000000000000006661
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000000011
dFPU: +1.0000000000000000000000000000000000000000000000000011

Iteration Number: 8, dFrac: 7.999999999999999833e-16
         dReg                     dRes                   dFPU
          1            1.0000000000000008882  1.0000000000000008882
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000000100
dFPU: +1.0000000000000000000000000000000000000000000000000100

Iteration Number: 9, dFrac: 9.000000000000000305e-16
         dReg                     dRes                   dFPU
          1            1.0000000000000008882  1.0000000000000008882
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000000100
dFPU: +1.0000000000000000000000000000000000000000000000000100

Iteration Number: 10, dFrac: 1.0000000000000000777e-15
         dReg                     dRes                   dFPU
          1            1.0000000000000011102  1.0000000000000011102
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000000101
```

```
dFPU: +1.0000000000000000000000000000000000000000000000000101

Iteration Number: 11, dFrac: 1.099999999999999928e-15
         dReg                     dRes                   dFPU
          1            1.0000000000000011102  1.0000000000000011102
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000000101
dFPU: +1.0000000000000000000000000000000000000000000000000101

Iteration Number: 12, dFrac: 1.199999999999999975e-15
         dReg                     dRes                   dFPU
          1            1.0000000000000011102  1.0000000000000011102
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000000101
dFPU: +1.0000000000000000000000000000000000000000000000000101

Iteration Number: 13, dFrac: 1.300000000000000022e-15
         dReg                     dRes                   dFPU
          1            1.0000000000000013323  1.0000000000000013323
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000000110
dFPU: +1.0000000000000000000000000000000000000000000000000110

Iteration Number: 14, dFrac: 1.399999999999999872e-15
         dReg                     dRes                   dFPU
          1            1.0000000000000013323  1.0000000000000013323
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000000110
dFPU: +1.0000000000000000000000000000000000000000000000000110

Iteration Number: 15, dFrac: 1.499999999999999919e-15
         dReg                     dRes                   dFPU
          1            1.0000000000000015543  1.0000000000000015543
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000000111
dFPU: +1.0000000000000000000000000000000000000000000000000111

Iteration Number: 16, dFrac: 1.599999999999999967e-15
         dReg                     dRes                   dFPU
          1            1.0000000000000015543  1.0000000000000015543
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000000111
dFPU: +1.0000000000000000000000000000000000000000000000000111

Iteration Number: 17, dFrac: 1.700000000000000014e-15
         dReg                     dRes                   dFPU
          1            1.0000000000000017764  1.0000000000000017764
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000001000
dFPU: +1.0000000000000000000000000000000000000000000000001000

Iteration Number: 18, dFrac: 1.800000000000000061e-15
         dReg                     dRes                   dFPU
          1            1.0000000000000017764  1.0000000000000017764
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000001000
dFPU: +1.0000000000000000000000000000000000000000000000001000

Iteration Number: 19, dFrac: 1.900000000000000108e-15
         dReg                     dRes                   dFPU
          1            1.0000000000000019984  1.0000000000000019984
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000001001
dFPU: +1.0000000000000000000000000000000000000000000000001001

Iteration Number: 20, dFrac: 2.000000000000000156e-15
         dReg                     dRes                   dFPU
          1            1.0000000000000019984  1.0000000000000019984
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000001001
dFPU: +1.0000000000000000000000000000000000000000000000001001
```

```
Iteration Number: 21, dFrac: 2.099999999999999808e-15
         dReg                     dRes                   dFPU
          1            1.0000000000000019984  1.0000000000000019984
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000001001
dFPU: +1.0000000000000000000000000000000000000000000000001001

Iteration Number: 22, dFrac: 2.199999999999999856e-15
         dReg                     dRes                   dFPU
          1            1.0000000000000022204  1.0000000000000022204
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000001010
dFPU: +1.0000000000000000000000000000000000000000000000001010

Iteration Number: 23, dFrac: 2.299999999999999903e-15
         dReg                     dRes                   dFPU
          1            1.0000000000000022204  1.0000000000000022204
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000001010
dFPU: +1.0000000000000000000000000000000000000000000000001010

Iteration Number: 24, dFrac: 2.39999999999999995e-15
         dReg                     dRes                   dFPU
          1            1.0000000000000024425  1.0000000000000024425
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000001011
dFPU: +1.0000000000000000000000000000000000000000000000001011

Iteration Number: 25, dFrac: 2.499999999999999997e-15
         dReg                     dRes                   dFPU
          1            1.0000000000000024425  1.0000000000000024425
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000001011
dFPU: +1.0000000000000000000000000000000000000000000000001011

Iteration Number: 26, dFrac: 2.600000000000000044e-15
         dReg                     dRes                   dFPU
          1            1.0000000000000026645  1.0000000000000026645
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000001100
dFPU: +1.0000000000000000000000000000000000000000000000001100

Iteration Number: 27, dFrac: 2.700000000000000092e-15
         dReg                     dRes                   dFPU
          1            1.0000000000000026645  1.0000000000000026645
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000001100
dFPU: +1.0000000000000000000000000000000000000000000000001100

Iteration Number: 28, dFrac: 2.799999999999999744e-15
         dReg                     dRes                   dFPU
          1            1.0000000000000028866  1.0000000000000028866
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000001101
dFPU: +1.0000000000000000000000000000000000000000000000001101

Iteration Number: 29, dFrac: 2.899999999999999792e-15
         dReg                     dRes                   dFPU
          1            1.0000000000000028866  1.0000000000000028866
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000001101
dFPU: +1.0000000000000000000000000000000000000000000000001101

Iteration Number: 30, dFrac: 2.999999999999999839e-15
         dReg                     dRes                   dFPU
          1            1.0000000000000031086  1.0000000000000031086
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000001110
dFPU: +1.0000000000000000000000000000000000000000000000001110
```

```
Iteration Number: 31, dFrac: 3.0999999999999999886e-15
         dReg                     dRes                     dFPU
          1            1.0000000000000031086  1.0000000000000031086
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000001110
dFPU: +1.0000000000000000000000000000000000000000000000001110

Iteration Number: 32, dFrac: 3.1999999999999999933e-15
         dReg                     dRes                     dFPU
          1            1.0000000000000031086  1.0000000000000031086
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000001110
dFPU: +1.0000000000000000000000000000000000000000000000001110

Iteration Number: 33, dFrac: 3.29999999999999998e-15
         dReg                     dRes                     dFPU
          1            1.0000000000000033307  1.0000000000000033307
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000001111
dFPU: +1.0000000000000000000000000000000000000000000000001111

Iteration Number: 34, dFrac: 3.4000000000000000028e-15
         dReg                     dRes                     dFPU
          1            1.0000000000000033307  1.0000000000000033307
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000001111
dFPU: +1.0000000000000000000000000000000000000000000000001111

Iteration Number: 35, dFrac: 3.5000000000000000075e-15
         dReg                     dRes                     dFPU
          1            1.0000000000000035527  1.0000000000000035527
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000010000
dFPU: +1.0000000000000000000000000000000000000000000000010000

Iteration Number: 36, dFrac: 3.6000000000000000122e-15
         dReg                     dRes                     dFPU
          1            1.0000000000000035527  1.0000000000000035527
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000010000
dFPU: +1.0000000000000000000000000000000000000000000000010000

Iteration Number: 37, dFrac: 3.7000000000000000169e-15
         dReg                     dRes                     dFPU
          1            1.0000000000000037748  1.0000000000000037748
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000010001
dFPU: +1.0000000000000000000000000000000000000000000000010001

Iteration Number: 38, dFrac: 3.8000000000000000216e-15
         dReg                     dRes                     dFPU
          1            1.0000000000000037748  1.0000000000000037748
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000010001
dFPU: +1.0000000000000000000000000000000000000000000000010001

Iteration Number: 39, dFrac: 3.9000000000000000264e-15
         dReg                     dRes                     dFPU
          1            1.0000000000000039968  1.0000000000000039968
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000010010
dFPU: +1.0000000000000000000000000000000000000000000000010010

Iteration Number: 40, dFrac: 4.0000000000000000311e-15
         dReg                     dRes                     dFPU
          1            1.0000000000000039968  1.0000000000000039968
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000010010
dFPU: +1.0000000000000000000000000000000000000000000000010010

Iteration Number: 41, dFrac: 4.0999999999999999569e-15
```

```
         dReg                     dRes                   dFPU
          1            1.0000000000000039968  1.0000000000000039968
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000010010
dFPU: +1.0000000000000000000000000000000000000000000000010010

Iteration Number: 42, dFrac: 4.199999999999999616e-15
         dReg                     dRes                   dFPU
          1            1.0000000000000042188  1.0000000000000042188
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000010011
dFPU: +1.0000000000000000000000000000000000000000000000010011

Iteration Number: 43, dFrac: 4.299999999999999664e-15
         dReg                     dRes                   dFPU
          1            1.0000000000000042188  1.0000000000000042188
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000010011
dFPU: +1.0000000000000000000000000000000000000000000000010011

Iteration Number: 44, dFrac: 4.399999999999999711e-15
         dReg                     dRes                   dFPU
          1            1.0000000000000044409  1.0000000000000044409
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000010100
dFPU: +1.0000000000000000000000000000000000000000000000010100

Iteration Number: 45, dFrac: 4.499999999999999758e-15
         dReg                     dRes                   dFPU
          1            1.0000000000000044409  1.0000000000000044409
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000010100
dFPU: +1.0000000000000000000000000000000000000000000000010100

Iteration Number: 46, dFrac: 4.599999999999999806e-15
         dReg                     dRes                   dFPU
          1            1.0000000000000046629  1.0000000000000046629
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000010101
dFPU: +1.0000000000000000000000000000000000000000000000010101

Iteration Number: 47, dFrac: 4.699999999999999852e-15
         dReg                     dRes                   dFPU
          1            1.0000000000000046629  1.0000000000000046629
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000010101
dFPU: +1.0000000000000000000000000000000000000000000000010101

Iteration Number: 48, dFrac: 4.7999999999999999e-15
         dReg                     dRes                   dFPU
          1             1.000000000000004885   1.000000000000004885
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000010110
dFPU: +1.0000000000000000000000000000000000000000000000010110

Iteration Number: 49, dFrac: 4.899999999999999947e-15
         dReg                     dRes                   dFPU
          1             1.000000000000004885   1.000000000000004885
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000010110
dFPU: +1.0000000000000000000000000000000000000000000000010110

Iteration Number: 50, dFrac: 4.999999999999999994e-15
         dReg                     dRes                   dFPU
          1             1.000000000000005107   1.000000000000005107
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000010111
dFPU: +1.0000000000000000000000000000000000000000000000010111

Iteration Number: 51, dFrac: 5.100000000000000042e-15
         dReg                     dRes                   dFPU
```

```
          1             1.000000000000005107   1.000000000000005107
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000010111
dFPU: +1.0000000000000000000000000000000000000000000000010111

Iteration Number: 52, dFrac: 5.200000000000000088e-15
         dReg                     dRes                   dFPU
          1             1.000000000000005107   1.000000000000005107
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000010111
dFPU: +1.0000000000000000000000000000000000000000000000010111

Iteration Number: 53, dFrac: 5.300000000000000136e-15
         dReg                     dRes                   dFPU
          1            1.0000000000000053291  1.0000000000000053291
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000011000
dFPU: +1.0000000000000000000000000000000000000000000000011000

Iteration Number: 54, dFrac: 5.400000000000000183e-15
         dReg                     dRes                   dFPU
          1            1.0000000000000053291  1.0000000000000053291
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000011000
dFPU: +1.0000000000000000000000000000000000000000000000011000

Iteration Number: 55, dFrac: 5.50000000000000023e-15
         dReg                     dRes                   dFPU
          1            1.0000000000000055511  1.0000000000000055511
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000011001
dFPU: +1.0000000000000000000000000000000000000000000000011001

Iteration Number: 56, dFrac: 5.599999999999999488e-15
         dReg                     dRes                   dFPU
          1            1.0000000000000055511  1.0000000000000055511
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000011001
dFPU: +1.0000000000000000000000000000000000000000000000011001

Iteration Number: 57, dFrac: 5.699999999999999536e-15
         dReg                     dRes                   dFPU
          1            1.0000000000000057732  1.0000000000000057732
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000011010
dFPU: +1.0000000000000000000000000000000000000000000000011010

Iteration Number: 58, dFrac: 5.799999999999999583e-15
         dReg                     dRes                   dFPU
          1            1.0000000000000057732  1.0000000000000057732
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000011010
dFPU: +1.0000000000000000000000000000000000000000000000011010

Iteration Number: 59, dFrac: 5.89999999999999963e-15
         dReg                     dRes                   dFPU
          1            1.0000000000000059952  1.0000000000000059952
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000011011
dFPU: +1.0000000000000000000000000000000000000000000000011011

Iteration Number: 60, dFrac: 5.999999999999999678e-15
         dReg                     dRes                   dFPU
          1            1.0000000000000059952  1.0000000000000059952
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000011011
dFPU: +1.0000000000000000000000000000000000000000000000011011

Iteration Number: 61, dFrac: 6.099999999999999725e-15
         dReg                     dRes                   dFPU
          1            1.0000000000000059952  1.0000000000000059952
```

```
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000011011
dFPU: +1.0000000000000000000000000000000000000000000000011011

Iteration Number: 62, dFrac: 6.199999999999999772e-15
         dReg                     dRes                   dFPU
          1            1.0000000000000062172  1.0000000000000062172
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000011100
dFPU: +1.0000000000000000000000000000000000000000000000011100

Iteration Number: 63, dFrac: 6.29999999999999982e-15
         dReg                     dRes                   dFPU
          1            1.0000000000000062172  1.0000000000000062172
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000011100
dFPU: +1.0000000000000000000000000000000000000000000000011100

Iteration Number: 64, dFrac: 6.399999999999999866e-15
         dReg                     dRes                   dFPU
          1            1.0000000000000064393  1.0000000000000064393
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000011101
dFPU: +1.0000000000000000000000000000000000000000000000011101

Iteration Number: 65, dFrac: 6.499999999999999914e-15
         dReg                     dRes                   dFPU
          1            1.0000000000000064393  1.0000000000000064393
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000011101
dFPU: +1.0000000000000000000000000000000000000000000000011101

Iteration Number: 66, dFrac: 6.599999999999999961e-15
         dReg                     dRes                   dFPU
          1            1.0000000000000066613  1.0000000000000066613
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000011110
dFPU: +1.0000000000000000000000000000000000000000000000011110

Iteration Number: 67, dFrac: 6.700000000000000008e-15
         dReg                     dRes                   dFPU
          1            1.0000000000000066613  1.0000000000000066613
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000011110
dFPU: +1.0000000000000000000000000000000000000000000000011110

Iteration Number: 68, dFrac: 6.800000000000000056e-15
         dReg                     dRes                   dFPU
          1            1.0000000000000068834  1.0000000000000068834
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000011111
dFPU: +1.0000000000000000000000000000000000000000000000011111

Iteration Number: 69, dFrac: 6.900000000000000102e-15
         dReg                     dRes                   dFPU
          1            1.0000000000000068834  1.0000000000000068834
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000011111
dFPU: +1.0000000000000000000000000000000000000000000000011111

Iteration Number: 70, dFrac: 7.00000000000000015e-15
         dReg                     dRes                   dFPU
          1            1.0000000000000071054  1.0000000000000071054
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000100000
dFPU: +1.0000000000000000000000000000000000000000000000100000

Iteration Number: 71, dFrac: 7.100000000000000197e-15
         dReg                     dRes                   dFPU
          1            1.0000000000000071054  1.0000000000000071054
dReg: +1.0000000000000000000000000000000000000000000000000000
```

```
dRes: +1.0000000000000000000000000000000000000000000000100000
dFPU: +1.0000000000000000000000000000000000000000000000100000

Iteration Number: 72, dFrac: 7.200000000000000244e-15
         dReg                     dRes                   dFPU
          1            1.0000000000000071054  1.0000000000000071054
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000100000
dFPU: +1.0000000000000000000000000000000000000000000000100000

Iteration Number: 73, dFrac: 7.299999999999999502e-15
         dReg                     dRes                   dFPU
          1            1.0000000000000073275  1.0000000000000073275
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000100001
dFPU: +1.0000000000000000000000000000000000000000000000100001

Iteration Number: 74, dFrac: 7.400000000000000338e-15
         dReg                     dRes                   dFPU
          1            1.0000000000000073275  1.0000000000000073275
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000100001
dFPU: +1.0000000000000000000000000000000000000000000000100001

Iteration Number: 75, dFrac: 7.499999999999999597e-15
         dReg                     dRes                   dFPU
          1            1.0000000000000075495  1.0000000000000075495
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000100010
dFPU: +1.0000000000000000000000000000000000000000000000100010

Iteration Number: 76, dFrac: 7.600000000000000433e-15
         dReg                     dRes                   dFPU
          1            1.0000000000000075495  1.0000000000000075495
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000100010
dFPU: +1.0000000000000000000000000000000000000000000000100010

Iteration Number: 77, dFrac: 7.699999999999999692e-15
         dReg                     dRes                   dFPU
          1            1.0000000000000077716  1.0000000000000077716
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000100011
dFPU: +1.0000000000000000000000000000000000000000000000100011

Iteration Number: 78, dFrac: 7.800000000000000528e-15
         dReg                     dRes                   dFPU
          1            1.0000000000000077716  1.0000000000000077716
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000100011
dFPU: +1.0000000000000000000000000000000000000000000000100011

Iteration Number: 79, dFrac: 7.899999999999999786e-15
         dReg                     dRes                   dFPU
          1            1.0000000000000079936  1.0000000000000079936
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000100100
dFPU: +1.0000000000000000000000000000000000000000000000100100

Iteration Number: 80, dFrac: 8.000000000000000622e-15
         dReg                     dRes                   dFPU
          1            1.0000000000000079936  1.0000000000000079936
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000100100
dFPU: +1.0000000000000000000000000000000000000000000000100100

Iteration Number: 81, dFrac: 8.09999999999999988e-15
         dReg                     dRes                   dFPU
          1            1.0000000000000079936  1.0000000000000079936
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000100100
```

```
dFPU: +1.0000000000000000000000000000000000000000000000100100

Iteration Number: 82, dFrac: 8.199999999999999138e-15
         dReg                     dRes                   dFPU
          1            1.0000000000000082157  1.0000000000000082157
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000100101
dFPU: +1.0000000000000000000000000000000000000000000000100101

Iteration Number: 83, dFrac: 8.299999999999999975e-15
         dReg                     dRes                   dFPU
          1            1.0000000000000082157  1.0000000000000082157
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000100101
dFPU: +1.0000000000000000000000000000000000000000000000100101

Iteration Number: 84, dFrac: 8.399999999999999233e-15
         dReg                     dRes                   dFPU
          1            1.0000000000000084377  1.0000000000000084377
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000100110
dFPU: +1.0000000000000000000000000000000000000000000000100110

Iteration Number: 85, dFrac: 8.500000000000000069e-15
         dReg                     dRes                   dFPU
          1            1.0000000000000084377  1.0000000000000084377
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000100110
dFPU: +1.0000000000000000000000000000000000000000000000100110

Iteration Number: 86, dFrac: 8.599999999999999328e-15
         dReg                     dRes                   dFPU
          1            1.0000000000000086597  1.0000000000000086597
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000100111
dFPU: +1.0000000000000000000000000000000000000000000000100111

Iteration Number: 87, dFrac: 8.700000000000000164e-15
         dReg                     dRes                   dFPU
          1            1.0000000000000086597  1.0000000000000086597
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000100111
dFPU: +1.0000000000000000000000000000000000000000000000100111

Iteration Number: 88, dFrac: 8.799999999999999422e-15
         dReg                     dRes                   dFPU
          1            1.0000000000000088818  1.0000000000000088818
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000101000
dFPU: +1.0000000000000000000000000000000000000000000000101000

Iteration Number: 89, dFrac: 8.900000000000000258e-15
         dReg                     dRes                   dFPU
          1            1.0000000000000088818  1.0000000000000088818
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000101000
dFPU: +1.0000000000000000000000000000000000000000000000101000

Iteration Number: 90, dFrac: 8.999999999999999516e-15
         dReg                     dRes                   dFPU
          1            1.0000000000000091038  1.0000000000000091038
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000101001
dFPU: +1.0000000000000000000000000000000000000000000000101001

Iteration Number: 91, dFrac: 9.100000000000000352e-15
         dReg                     dRes                   dFPU
          1            1.0000000000000091038  1.0000000000000091038
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000101001
dFPU: +1.0000000000000000000000000000000000000000000000101001
```

```
Iteration Number: 92, dFrac: 9.199999999999999611e-15
         dReg                     dRes                   dFPU
          1            1.0000000000000091038  1.0000000000000091038
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000101001
dFPU: +1.0000000000000000000000000000000000000000000000101001

Iteration Number: 93, dFrac: 9.300000000000000447e-15
         dReg                     dRes                   dFPU
          1            1.0000000000000093259  1.0000000000000093259
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000101010
dFPU: +1.0000000000000000000000000000000000000000000000101010

Iteration Number: 94, dFrac: 9.399999999999999705e-15
         dReg                     dRes                   dFPU
          1            1.0000000000000093259  1.0000000000000093259
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000101010
dFPU: +1.0000000000000000000000000000000000000000000000101010

Iteration Number: 95, dFrac: 9.500000000000000541e-15
         dReg                     dRes                   dFPU
          1            1.0000000000000095479  1.0000000000000095479
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000101011
dFPU: +1.0000000000000000000000000000000000000000000000101011

Iteration Number: 96, dFrac: 9.5999999999999998e-15
         dReg                     dRes                   dFPU
          1            1.0000000000000095479  1.0000000000000095479
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000101011
dFPU: +1.0000000000000000000000000000000000000000000000101011

Iteration Number: 97, dFrac: 9.699999999999999058e-15
         dReg                     dRes                   dFPU
          1             1.00000000000000977    1.00000000000000977
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000101100
dFPU: +1.0000000000000000000000000000000000000000000000101100

Iteration Number: 98, dFrac: 9.799999999999999894e-15
         dReg                     dRes                   dFPU
          1             1.00000000000000977    1.00000000000000977
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000101100
dFPU: +1.0000000000000000000000000000000000000000000000101100

Iteration Number: 99, dFrac: 9.899999999999999153e-15
         dReg                     dRes                   dFPU
          1             1.000000000000009992   1.000000000000009992
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000101101
dFPU: +1.0000000000000000000000000000000000000000000000101101

Iteration Number: 100, dFrac: 9.999999999999999989e-15
         dReg                     dRes                   dFPU
          1             1.000000000000009992   1.000000000000009992
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000101101
dFPU: +1.0000000000000000000000000000000000000000000000101101
```

Appendix AE-1 Results of Test Case Five, Experiment One

```
Iteration Number: 1, dFrac: 2e-17
         dReg                     dRes                   dFPU
          1                      1                      1
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000000000
dFPU: +1.0000000000000000000000000000000000000000000000000000

Iteration Number: 2, dFrac: 4e-17
         dReg                     dRes                   dFPU
          1                      1                      1
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000000000
dFPU: +1.0000000000000000000000000000000000000000000000000000

Iteration Number: 3, dFrac: 7e-17
         dReg                     dRes                   dFPU
          1                      1                      1
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000000000
dFPU: +1.0000000000000000000000000000000000000000000000000000

Iteration Number: 4, dFrac: 1.1e-16
         dReg                     dRes                   dFPU
          1                      1                      1
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000000000
dFPU: +1.0000000000000000000000000000000000000000000000000000

Iteration Number: 5, dFrac: 1.6e-16
         dReg                     dRes                   dFPU
          1             1.000000000000000222   1.000000000000000222
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000000001
dFPU: +1.0000000000000000000000000000000000000000000000000001

Iteration Number: 6, dFrac: 2.2e-16
         dReg                     dRes                   dFPU
          1             1.000000000000000222   1.000000000000000222
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000000001
dFPU: +1.0000000000000000000000000000000000000000000000000001

Iteration Number: 7, dFrac: 2.9e-16
         dReg                     dRes                   dFPU
          1             1.000000000000000222   1.000000000000000222
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000000001
dFPU: +1.0000000000000000000000000000000000000000000000000001

Iteration Number: 8, dFrac: 3.7e-16
         dReg                     dRes                   dFPU
          1            1.0000000000000004441  1.0000000000000004441
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000000010
dFPU: +1.0000000000000000000000000000000000000000000000000010

Iteration Number: 9, dFrac: 4.6e-16
         dReg                     dRes                   dFPU
          1            1.0000000000000004441  1.0000000000000004441
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000000010
dFPU: +1.0000000000000000000000000000000000000000000000000010

Iteration Number: 10, dFrac: 5.6e-16
         dReg                     dRes                   dFPU
          1            1.0000000000000004441  1.0000000000000004441
dReg: +1.0000000000000000000000000000000000000000000000000000
```

```
dRes: +1.0000000000000000000000000000000000000000000000000010
dFPU: +1.0000000000000000000000000000000000000000000000000010

Iteration Number: 11, dFrac: 6.7e-16
         dReg                     dRes                   dFPU
          1            1.0000000000000006661  1.0000000000000006661
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000000011
dFPU: +1.0000000000000000000000000000000000000000000000000011

Iteration Number: 12, dFrac: 7.9e-16
         dReg                     dRes                   dFPU
 1.000000000000000222  1.0000000000000008882  1.0000000000000008882
dReg: +1.0000000000000000000000000000000000000000000000000001
dRes: +1.0000000000000000000000000000000000000000000000000100
dFPU: +1.0000000000000000000000000000000000000000000000000100

Iteration Number: 13, dFrac: 9.2e-16
         dReg                     dRes                   dFPU
1.0000000000000004441  1.0000000000000008882  1.0000000000000008882
dReg: +1.0000000000000000000000000000000000000000000000000010
dRes: +1.0000000000000000000000000000000000000000000000000100
dFPU: +1.0000000000000000000000000000000000000000000000000100

Iteration Number: 14, dFrac: 1.06e-15
         dReg                     dRes                   dFPU
1.0000000000000006661  1.0000000000000011102  1.0000000000000011102
dReg: +1.0000000000000000000000000000000000000000000000000011
dRes: +1.0000000000000000000000000000000000000000000000000101
dFPU: +1.0000000000000000000000000000000000000000000000000101

Iteration Number: 15, dFrac: 1.21e-15
         dReg                     dRes                   dFPU
1.0000000000000008882  1.0000000000000011102  1.0000000000000011102
dReg: +1.0000000000000000000000000000000000000000000000000100
dRes: +1.0000000000000000000000000000000000000000000000000101
dFPU: +1.0000000000000000000000000000000000000000000000000101

Iteration Number: 16, dFrac: 1.37e-15
         dReg                     dRes                   dFPU
1.0000000000000011102  1.0000000000000013323  1.0000000000000013323
dReg: +1.0000000000000000000000000000000000000000000000000101
dRes: +1.0000000000000000000000000000000000000000000000000110
dFPU: +1.0000000000000000000000000000000000000000000000000110

Iteration Number: 17, dFrac: 1.54e-15
         dReg                     dRes                   dFPU
1.0000000000000013323  1.0000000000000015543  1.0000000000000015543
dReg: +1.0000000000000000000000000000000000000000000000000110
dRes: +1.0000000000000000000000000000000000000000000000000111
dFPU: +1.0000000000000000000000000000000000000000000000000111

Iteration Number: 18, dFrac: 1.72e-15
         dReg                     dRes                   dFPU
1.0000000000000015543  1.0000000000000017764  1.0000000000000017764
dReg: +1.0000000000000000000000000000000000000000000000000111
dRes: +1.0000000000000000000000000000000000000000000000001000
dFPU: +1.0000000000000000000000000000000000000000000000001000

Iteration Number: 19, dFrac: 1.91e-15
         dReg                     dRes                   dFPU
1.0000000000000017764  1.0000000000000019984  1.0000000000000019984
dReg: +1.0000000000000000000000000000000000000000000000001000
dRes: +1.0000000000000000000000000000000000000000000000001001
dFPU: +1.0000000000000000000000000000000000000000000000001001

Iteration Number: 20, dFrac: 2.11e-15
         dReg                     dRes                   dFPU
1.0000000000000019984  1.0000000000000019984  1.0000000000000019984
dReg: +1.0000000000000000000000000000000000000000000000001001
dRes: +1.0000000000000000000000000000000000000000000000001001
```

```
dFPU: +1.0000000000000000000000000000000000000000000000001001

Iteration Number: 21, dFrac: 2.32e-15
         dReg                     dRes                   dFPU
1.0000000000000022204  1.0000000000000022204  1.0000000000000022204
dReg: +1.0000000000000000000000000000000000000000000000001010
dRes: +1.0000000000000000000000000000000000000000000000001010
dFPU: +1.0000000000000000000000000000000000000000000000001010

Iteration Number: 22, dFrac: 2.54e-15
         dReg                     dRes                   dFPU
1.0000000000000024425  1.0000000000000024425  1.0000000000000024425
dReg: +1.0000000000000000000000000000000000000000000000001011
dRes: +1.0000000000000000000000000000000000000000000000001011
dFPU: +1.0000000000000000000000000000000000000000000000001011

Iteration Number: 23, dFrac: 2.77e-15
         dReg                     dRes                   dFPU
1.0000000000000026645  1.0000000000000026645  1.0000000000000026645
dReg: +1.0000000000000000000000000000000000000000000000001100
dRes: +1.0000000000000000000000000000000000000000000000001100
dFPU: +1.0000000000000000000000000000000000000000000000001100

Iteration Number: 24, dFrac: 3.01e-15
         dReg                     dRes                   dFPU
1.0000000000000028866  1.0000000000000031086  1.0000000000000031086
dReg: +1.0000000000000000000000000000000000000000000000001101
dRes: +1.0000000000000000000000000000000000000000000000001110
dFPU: +1.0000000000000000000000000000000000000000000000001110

Iteration Number: 25, dFrac: 3.26e-15
         dReg                     dRes                   dFPU
1.0000000000000031086  1.0000000000000033307  1.0000000000000033307
dReg: +1.0000000000000000000000000000000000000000000000001110
dRes: +1.0000000000000000000000000000000000000000000000001111
dFPU: +1.0000000000000000000000000000000000000000000000001111

Iteration Number: 26, dFrac: 3.52e-15
         dReg                     dRes                   dFPU
1.0000000000000033307  1.0000000000000035527  1.0000000000000035527
dReg: +1.0000000000000000000000000000000000000000000000001111
dRes: +1.0000000000000000000000000000000000000000000000010000
dFPU: +1.0000000000000000000000000000000000000000000000010000

Iteration Number: 27, dFrac: 3.79e-15
         dReg                     dRes                   dFPU
1.0000000000000035527  1.0000000000000037748  1.0000000000000037748
dReg: +1.0000000000000000000000000000000000000000000000010000
dRes: +1.0000000000000000000000000000000000000000000000010001
dFPU: +1.0000000000000000000000000000000000000000000000010001

Iteration Number: 28, dFrac: 4.07e-15
         dReg                     dRes                   dFPU
1.0000000000000037748  1.0000000000000039968  1.0000000000000039968
dReg: +1.0000000000000000000000000000000000000000000000010001
dRes: +1.0000000000000000000000000000000000000000000000010010
dFPU: +1.0000000000000000000000000000000000000000000000010010

Iteration Number: 29, dFrac: 4.36e-15
         dReg                     dRes                   dFPU
1.0000000000000039968  1.0000000000000044409  1.0000000000000044409
dReg: +1.0000000000000000000000000000000000000000000000010010
dRes: +1.0000000000000000000000000000000000000000000000010100
dFPU: +1.0000000000000000000000000000000000000000000000010100

Iteration Number: 30, dFrac: 4.66e-15
         dReg                     dRes                   dFPU
1.0000000000000042188  1.0000000000000046629  1.0000000000000046629
dReg: +1.0000000000000000000000000000000000000000000000010011
dRes: +1.0000000000000000000000000000000000000000000000010101
dFPU: +1.0000000000000000000000000000000000000000000000010101
```

```
Iteration Number: 31, dFrac: 4.97e-15
         dReg                     dRes                   dFPU
1.0000000000000044409   1.000000000000004885   1.000000000000004885
dReg: +1.0000000000000000000000000000000000000000000000010100
dRes: +1.0000000000000000000000000000000000000000000000010110
dFPU: +1.0000000000000000000000000000000000000000000000010110

Iteration Number: 32, dFrac: 5.29e-15
         dReg                     dRes                   dFPU
1.0000000000000046629  1.0000000000000053291  1.0000000000000053291
dReg: +1.0000000000000000000000000000000000000000000000010101
dRes: +1.0000000000000000000000000000000000000000000000011000
dFPU: +1.0000000000000000000000000000000000000000000000011000

Iteration Number: 33, dFrac: 5.62e-15
         dReg                     dRes                   dFPU
 1.000000000000004885  1.0000000000000055511  1.0000000000000055511
dReg: +1.0000000000000000000000000000000000000000000000010110
dRes: +1.0000000000000000000000000000000000000000000000011001
dFPU: +1.0000000000000000000000000000000000000000000000011001

Iteration Number: 34, dFrac: 5.96e-15
         dReg                     dRes                   dFPU
1.0000000000000053291  1.0000000000000059952  1.0000000000000059952
dReg: +1.0000000000000000000000000000000000000000000000011000
dRes: +1.0000000000000000000000000000000000000000000000011011
dFPU: +1.0000000000000000000000000000000000000000000000011011

Iteration Number: 35, dFrac: 6.31e-15
         dReg                     dRes                   dFPU
1.0000000000000057732  1.0000000000000062172  1.0000000000000062172
dReg: +1.0000000000000000000000000000000000000000000000011010
dRes: +1.0000000000000000000000000000000000000000000000011100
dFPU: +1.0000000000000000000000000000000000000000000000011100

Iteration Number: 36, dFrac: 6.67e-15
         dReg                     dRes                   dFPU
1.0000000000000062172  1.0000000000000066613  1.0000000000000066613
dReg: +1.0000000000000000000000000000000000000000000000011100
dRes: +1.0000000000000000000000000000000000000000000000011110
dFPU: +1.0000000000000000000000000000000000000000000000011110

Iteration Number: 37, dFrac: 7.04e-15
         dReg                     dRes                   dFPU
1.0000000000000066613  1.0000000000000071054  1.0000000000000071054
dReg: +1.0000000000000000000000000000000000000000000000011110
dRes: +1.0000000000000000000000000000000000000000000000100000
dFPU: +1.0000000000000000000000000000000000000000000000100000

Iteration Number: 38, dFrac: 7.42e-15
         dReg                     dRes                   dFPU
1.0000000000000071054  1.0000000000000073275  1.0000000000000073275
dReg: +1.0000000000000000000000000000000000000000000000100000
dRes: +1.0000000000000000000000000000000000000000000000100001
dFPU: +1.0000000000000000000000000000000000000000000000100001

Iteration Number: 39, dFrac: 7.81e-15
         dReg                     dRes                   dFPU
1.0000000000000075495  1.0000000000000077716  1.0000000000000077716
dReg: +1.0000000000000000000000000000000000000000000000100010
dRes: +1.0000000000000000000000000000000000000000000000100011
dFPU: +1.0000000000000000000000000000000000000000000000100011

Iteration Number: 40, dFrac: 8.21e-15
         dReg                     dRes                   dFPU
1.0000000000000079936  1.0000000000000082157  1.0000000000000082157
dReg: +1.0000000000000000000000000000000000000000000000100100
dRes: +1.0000000000000000000000000000000000000000000000100101
dFPU: +1.0000000000000000000000000000000000000000000000100101
```

```
Iteration Number: 41, dFrac: 8.62e-15
         dReg                     dRes                   dFPU
1.0000000000000084377  1.0000000000000086597  1.0000000000000086597
dReg: +1.0000000000000000000000000000000000000000000000100110
dRes: +1.0000000000000000000000000000000000000000000000100111
dFPU: +1.0000000000000000000000000000000000000000000000100111

Iteration Number: 42, dFrac: 9.04e-15
         dReg                     dRes                   dFPU
1.0000000000000088818  1.0000000000000091038  1.0000000000000091038
dReg: +1.0000000000000000000000000000000000000000000000101000
dRes: +1.0000000000000000000000000000000000000000000000101001
dFPU: +1.0000000000000000000000000000000000000000000000101001

Iteration Number: 43, dFrac: 9.47e-15
         dReg                     dRes                   dFPU
1.0000000000000093259  1.0000000000000095479  1.0000000000000095479
dReg: +1.0000000000000000000000000000000000000000000000101010
dRes: +1.0000000000000000000000000000000000000000000000101011
dFPU: +1.0000000000000000000000000000000000000000000000101011

Iteration Number: 44, dFrac: 9.91e-15
         dReg                     dRes                   dFPU
 1.00000000000000977    1.000000000000009992   1.000000000000009992
dReg: +1.0000000000000000000000000000000000000000000000101100
dRes: +1.0000000000000000000000000000000000000000000000101101
dFPU: +1.0000000000000000000000000000000000000000000000101101

Iteration Number: 45, dFrac: 1.036e-14
         dReg                     dRes                   dFPU
1.0000000000000102141  1.0000000000000104361  1.0000000000000104361
dReg: +1.0000000000000000000000000000000000000000000000101110
dRes: +1.0000000000000000000000000000000000000000000000101111
dFPU: +1.0000000000000000000000000000000000000000000000101111

Iteration Number: 46, dFrac: 1.082e-14
         dReg                     dRes                   dFPU
1.0000000000000106581  1.0000000000000108802  1.0000000000000108802
dReg: +1.0000000000000000000000000000000000000000000000110000
dRes: +1.0000000000000000000000000000000000000000000000110001
dFPU: +1.0000000000000000000000000000000000000000000000110001

Iteration Number: 47, dFrac: 1.129e-14
         dReg                     dRes                   dFPU
1.0000000000000111022  1.0000000000000113243  1.0000000000000113243
dReg: +1.0000000000000000000000000000000000000000000000110010
dRes: +1.0000000000000000000000000000000000000000000000110011
dFPU: +1.0000000000000000000000000000000000000000000000110011

Iteration Number: 48, dFrac: 1.177e-14
         dReg                     dRes                   dFPU
1.0000000000000115463  1.0000000000000117684  1.0000000000000117684
dReg: +1.0000000000000000000000000000000000000000000000110100
dRes: +1.0000000000000000000000000000000000000000000000110101
dFPU: +1.0000000000000000000000000000000000000000000000110101

Iteration Number: 49, dFrac: 1.226e-14
         dReg                     dRes                   dFPU
1.0000000000000119904  1.0000000000000122125  1.0000000000000122125
dReg: +1.0000000000000000000000000000000000000000000000110110
dRes: +1.0000000000000000000000000000000000000000000000110111
dFPU: +1.0000000000000000000000000000000000000000000000110111

Iteration Number: 50, dFrac: 1.276e-14
         dReg                     dRes                   dFPU
1.0000000000000124345  1.0000000000000126565  1.0000000000000126565
dReg: +1.0000000000000000000000000000000000000000000000111000
dRes: +1.0000000000000000000000000000000000000000000000111001
dFPU: +1.0000000000000000000000000000000000000000000000111001

Iteration Number: 51, dFrac: 1.327e-14
```

```
         dReg                     dRes                   dFPU
1.0000000000000128786  1.0000000000000133227  1.0000000000000133227
dReg: +1.0000000000000000000000000000000000000000000000111010
dRes: +1.0000000000000000000000000000000000000000000000111100
dFPU: +1.0000000000000000000000000000000000000000000000111100

Iteration Number: 52, dFrac: 1.379e-14
         dReg                     dRes                   dFPU
1.0000000000000133227  1.0000000000000137668  1.0000000000000137668
dReg: +1.0000000000000000000000000000000000000000000000111100
dRes: +1.0000000000000000000000000000000000000000000000111110
dFPU: +1.0000000000000000000000000000000000000000000000111110

Iteration Number: 53, dFrac: 1.432e-14
         dReg                     dRes                   dFPU
1.0000000000000137668  1.0000000000000142109  1.0000000000000142109
dReg: +1.0000000000000000000000000000000000000000000000111110
dRes: +1.0000000000000000000000000000000000000000000001000000
dFPU: +1.0000000000000000000000000000000000000000000001000000

Iteration Number: 54, dFrac: 1.486e-14
         dReg                     dRes                   dFPU
1.0000000000000142109   1.000000000000014877   1.000000000000014877
dReg: +1.0000000000000000000000000000000000000000000001000000
dRes: +1.0000000000000000000000000000000000000000000001000011
dFPU: +1.0000000000000000000000000000000000000000000001000011

Iteration Number: 55, dFrac: 1.541e-14
         dReg                     dRes                   dFPU
1.0000000000000146549  1.0000000000000153211  1.0000000000000153211
dReg: +1.0000000000000000000000000000000000000000000001000010
dRes: +1.0000000000000000000000000000000000000000000001000101
dFPU: +1.0000000000000000000000000000000000000000000001000101

Iteration Number: 56, dFrac: 1.597e-14
         dReg                     dRes                   dFPU
1.0000000000000153211  1.0000000000000159872  1.0000000000000159872
dReg: +1.0000000000000000000000000000000000000000000001000101
dRes: +1.0000000000000000000000000000000000000000000001001000
dFPU: +1.0000000000000000000000000000000000000000000001001000

Iteration Number: 57, dFrac: 1.654e-14
         dReg                     dRes                   dFPU
1.0000000000000159872  1.0000000000000164313  1.0000000000000164313
dReg: +1.0000000000000000000000000000000000000000000001001000
dRes: +1.0000000000000000000000000000000000000000000001001010
dFPU: +1.0000000000000000000000000000000000000000000001001010

Iteration Number: 58, dFrac: 1.712e-14
         dReg                     dRes                   dFPU
1.0000000000000166533  1.0000000000000170974  1.0000000000000170974
dReg: +1.0000000000000000000000000000000000000000000001001011
dRes: +1.0000000000000000000000000000000000000000000001001101
dFPU: +1.0000000000000000000000000000000000000000000001001101

Iteration Number: 59, dFrac: 1.771e-14
         dReg                     dRes                   dFPU
1.0000000000000173195  1.0000000000000177636  1.0000000000000177636
dReg: +1.0000000000000000000000000000000000000000000001001110
dRes: +1.0000000000000000000000000000000000000000000001010000
dFPU: +1.0000000000000000000000000000000000000000000001010000

Iteration Number: 60, dFrac: 1.831e-14
         dReg                     dRes                   dFPU
1.0000000000000179856  1.0000000000000182077  1.0000000000000182077
dReg: +1.0000000000000000000000000000000000000000000001010001
dRes: +1.0000000000000000000000000000000000000000000001010010
dFPU: +1.0000000000000000000000000000000000000000000001010010

Iteration Number: 61, dFrac: 1.892e-14
         dReg                     dRes                   dFPU
```

```
1.0000000000000186517  1.0000000000000188738  1.0000000000000188738
dReg: +1.0000000000000000000000000000000000000000000001010100
dRes: +1.0000000000000000000000000000000000000000000001010101
dFPU: +1.0000000000000000000000000000000000000000000001010101

Iteration Number: 62, dFrac: 1.954e-14
         dReg                     dRes                   dFPU
1.0000000000000193179  1.0000000000000195399  1.0000000000000195399
dReg: +1.0000000000000000000000000000000000000000000001010111
dRes: +1.0000000000000000000000000000000000000000000001011000
dFPU: +1.0000000000000000000000000000000000000000000001011000

Iteration Number: 63, dFrac: 2.017e-14
         dReg                     dRes                   dFPU
 1.000000000000019984  1.0000000000000202061  1.0000000000000202061
dReg: +1.0000000000000000000000000000000000000000000001011010
dRes: +1.0000000000000000000000000000000000000000000001011011
dFPU: +1.0000000000000000000000000000000000000000000001011011

Iteration Number: 64, dFrac: 2.081e-14
         dReg                     dRes                   dFPU
1.0000000000000206501  1.0000000000000208722  1.0000000000000208722
dReg: +1.0000000000000000000000000000000000000000000001011101
dRes: +1.0000000000000000000000000000000000000000000001011110
dFPU: +1.0000000000000000000000000000000000000000000001011110

Iteration Number: 65, dFrac: 2.146e-14
         dReg                     dRes                   dFPU
1.0000000000000213163  1.0000000000000215383  1.0000000000000215383
dReg: +1.0000000000000000000000000000000000000000000001100000
dRes: +1.0000000000000000000000000000000000000000000001100001
dFPU: +1.0000000000000000000000000000000000000000000001100001

Iteration Number: 66, dFrac: 2.212e-14
         dReg                     dRes                   dFPU
1.0000000000000219824  1.0000000000000222045  1.0000000000000222045
dReg: +1.0000000000000000000000000000000000000000000001100011
dRes: +1.0000000000000000000000000000000000000000000001100100
dFPU: +1.0000000000000000000000000000000000000000000001100100

Iteration Number: 67, dFrac: 2.279e-14
         dReg                     dRes                   dFPU
1.0000000000000226485  1.0000000000000228706  1.0000000000000228706
dReg: +1.0000000000000000000000000000000000000000000001100110
dRes: +1.0000000000000000000000000000000000000000000001100111
dFPU: +1.0000000000000000000000000000000000000000000001100111

Iteration Number: 68, dFrac: 2.347e-14
         dReg                     dRes                   dFPU
1.0000000000000233147  1.0000000000000235367  1.0000000000000235367
dReg: +1.0000000000000000000000000000000000000000000001101001
dRes: +1.0000000000000000000000000000000000000000000001101010
dFPU: +1.0000000000000000000000000000000000000000000001101010

Iteration Number: 69, dFrac: 2.416e-14
         dReg                     dRes                   dFPU
1.0000000000000239808  1.0000000000000242029  1.0000000000000242029
dReg: +1.0000000000000000000000000000000000000000000001101100
dRes: +1.0000000000000000000000000000000000000000000001101101
dFPU: +1.0000000000000000000000000000000000000000000001101101

Iteration Number: 70, dFrac: 2.486e-14
         dReg                     dRes                   dFPU
 1.000000000000024647   1.000000000000024869   1.000000000000024869
dReg: +1.0000000000000000000000000000000000000000000001101111
dRes: +1.0000000000000000000000000000000000000000000001110000
dFPU: +1.0000000000000000000000000000000000000000000001110000

Iteration Number: 71, dFrac: 2.557e-14
         dReg                     dRes                   dFPU
1.0000000000000253131  1.0000000000000255351  1.0000000000000255351
```

```
dReg: +1.0000000000000000000000000000000000000000000001110010
dRes: +1.0000000000000000000000000000000000000000000001110011
dFPU: +1.0000000000000000000000000000000000000000000001110011

Iteration Number: 72, dFrac: 2.629e-14
         dReg                     dRes                   dFPU
1.0000000000000259792  1.0000000000000262013  1.0000000000000262013
dReg: +1.0000000000000000000000000000000000000000000001110101
dRes: +1.0000000000000000000000000000000000000000000001110110
dFPU: +1.0000000000000000000000000000000000000000000001110110

Iteration Number: 73, dFrac: 2.702e-14
         dReg                     dRes                   dFPU
1.0000000000000266454  1.0000000000000270894  1.0000000000000270894
dReg: +1.0000000000000000000000000000000000000000000001111000
dRes: +1.0000000000000000000000000000000000000000000001111010
dFPU: +1.0000000000000000000000000000000000000000000001111010

Iteration Number: 74, dFrac: 2.776e-14
         dReg                     dRes                   dFPU
1.0000000000000273115  1.0000000000000277556  1.0000000000000277556
dReg: +1.0000000000000000000000000000000000000000000001111011
dRes: +1.0000000000000000000000000000000000000000000001111101
dFPU: +1.0000000000000000000000000000000000000000000001111101

Iteration Number: 75, dFrac: 2.851e-14
         dReg                     dRes                   dFPU
1.0000000000000279776  1.0000000000000284217  1.0000000000000284217
dReg: +1.0000000000000000000000000000000000000000000001111110
dRes: +1.0000000000000000000000000000000000000000000010000000
dFPU: +1.0000000000000000000000000000000000000000000010000000

Iteration Number: 76, dFrac: 2.927e-14
         dReg                     dRes                   dFPU
1.0000000000000286438  1.0000000000000293099  1.0000000000000293099
dReg: +1.0000000000000000000000000000000000000000000010000001
dRes: +1.0000000000000000000000000000000000000000000010000100
dFPU: +1.0000000000000000000000000000000000000000000010000100

Iteration Number: 77, dFrac: 3.004e-14
         dReg                     dRes                   dFPU
1.0000000000000293099   1.000000000000029976   1.000000000000029976
dReg: +1.0000000000000000000000000000000000000000000010000100
dRes: +1.0000000000000000000000000000000000000000000010000111
dFPU: +1.0000000000000000000000000000000000000000000010000111

Iteration Number: 78, dFrac: 3.082e-14
         dReg                     dRes                   dFPU
1.0000000000000301981  1.0000000000000308642  1.0000000000000308642
dReg: +1.0000000000000000000000000000000000000000000010001000
dRes: +1.0000000000000000000000000000000000000000000010001011
dFPU: +1.0000000000000000000000000000000000000000000010001011

Iteration Number: 79, dFrac: 3.161e-14
         dReg                     dRes                   dFPU
1.0000000000000310862  1.0000000000000315303  1.0000000000000315303
dReg: +1.0000000000000000000000000000000000000000000010001100
dRes: +1.0000000000000000000000000000000000000000000010001110
dFPU: +1.0000000000000000000000000000000000000000000010001110

Iteration Number: 80, dFrac: 3.241e-14
         dReg                     dRes                   dFPU
1.0000000000000319744  1.0000000000000324185  1.0000000000000324185
dReg: +1.0000000000000000000000000000000000000000000010010000
dRes: +1.0000000000000000000000000000000000000000000010010010
dFPU: +1.0000000000000000000000000000000000000000000010010010

Iteration Number: 81, dFrac: 3.322e-14
         dReg                     dRes                   dFPU
1.0000000000000328626  1.0000000000000333067  1.0000000000000333067
dReg: +1.0000000000000000000000000000000000000000000010010100
```

```
dRes: +1.0000000000000000000000000000000000000000000010010110
dFPU: +1.0000000000000000000000000000000000000000000010010110

Iteration Number: 82, dFrac: 3.404e-14
         dReg                     dRes                   dFPU
1.0000000000000337508  1.0000000000000339728  1.0000000000000339728
dReg: +1.0000000000000000000000000000000000000000000010011000
dRes: +1.0000000000000000000000000000000000000000000010011001
dFPU: +1.0000000000000000000000000000000000000000000010011001

Iteration Number: 83, dFrac: 3.487e-14
         dReg                     dRes                   dFPU
 1.000000000000034639   1.000000000000034861   1.000000000000034861
dReg: +1.0000000000000000000000000000000000000000000010011100
dRes: +1.0000000000000000000000000000000000000000000010011101
dFPU: +1.0000000000000000000000000000000000000000000010011101

Iteration Number: 84, dFrac: 3.571e-14
         dReg                     dRes                   dFPU
1.0000000000000355271  1.0000000000000357492  1.0000000000000357492
dReg: +1.0000000000000000000000000000000000000000000010100000
dRes: +1.0000000000000000000000000000000000000000000010100001
dFPU: +1.0000000000000000000000000000000000000000000010100001

Iteration Number: 85, dFrac: 3.656e-14
         dReg                     dRes                   dFPU
1.0000000000000364153  1.0000000000000366374  1.0000000000000366374
dReg: +1.0000000000000000000000000000000000000000000010100100
dRes: +1.0000000000000000000000000000000000000000000010100101
dFPU: +1.0000000000000000000000000000000000000000000010100101

Iteration Number: 86, dFrac: 3.742e-14
         dReg                     dRes                   dFPU
1.0000000000000373035  1.0000000000000373035  1.0000000000000373035
dReg: +1.0000000000000000000000000000000000000000000010101000
dRes: +1.0000000000000000000000000000000000000000000010101000
dFPU: +1.0000000000000000000000000000000000000000000010101000

Iteration Number: 87, dFrac: 3.829e-14
         dReg                     dRes                   dFPU
1.0000000000000381917  1.0000000000000381917  1.0000000000000381917
dReg: +1.0000000000000000000000000000000000000000000010101100
dRes: +1.0000000000000000000000000000000000000000000010101100
dFPU: +1.0000000000000000000000000000000000000000000010101100

Iteration Number: 88, dFrac: 3.917e-14
         dReg                     dRes                   dFPU
1.0000000000000390799  1.0000000000000390799  1.0000000000000390799
dReg: +1.0000000000000000000000000000000000000000000010110000
dRes: +1.0000000000000000000000000000000000000000000010110000
dFPU: +1.0000000000000000000000000000000000000000000010110000

Iteration Number: 89, dFrac: 4.006e-14
         dReg                     dRes                   dFPU
 1.000000000000039968   1.000000000000039968   1.000000000000039968
dReg: +1.0000000000000000000000000000000000000000000010110100
dRes: +1.0000000000000000000000000000000000000000000010110100
dFPU: +1.0000000000000000000000000000000000000000000010110100

Iteration Number: 90, dFrac: 4.096e-14
         dReg                     dRes                   dFPU
1.0000000000000408562  1.0000000000000408562  1.0000000000000408562
dReg: +1.0000000000000000000000000000000000000000000010111000
dRes: +1.0000000000000000000000000000000000000000000010111000
dFPU: +1.0000000000000000000000000000000000000000000010111000

Iteration Number: 91, dFrac: 4.187e-14
         dReg                     dRes                   dFPU
1.0000000000000417444  1.0000000000000419664  1.0000000000000419664
dReg: +1.0000000000000000000000000000000000000000000010111100
dRes: +1.0000000000000000000000000000000000000000000010111101
```

```
dFPU: +1.000000000000000000000000000000000000000000000010111101

Iteration Number: 92, dFrac: 4.279e-14
         dReg                     dRes                   dFPU
1.0000000000000426326  1.0000000000000428546  1.0000000000000428546
dReg: +1.000000000000000000000000000000000000000000000011000000
dRes: +1.000000000000000000000000000000000000000000000011000001
dFPU: +1.000000000000000000000000000000000000000000000011000001

Iteration Number: 93, dFrac: 4.372e-14
         dReg                     dRes                   dFPU
1.0000000000000435207  1.0000000000000437428  1.0000000000000437428
dReg: +1.000000000000000000000000000000000000000000000011000100
dRes: +1.000000000000000000000000000000000000000000000011000101
dFPU: +1.000000000000000000000000000000000000000000000011000101

Iteration Number: 94, dFrac: 4.466e-14
         dReg                     dRes                   dFPU
1.0000000000000444089   1.000000000000044631   1.000000000000044631
dReg: +1.000000000000000000000000000000000000000000000011001000
dRes: +1.000000000000000000000000000000000000000000000011001001
dFPU: +1.000000000000000000000000000000000000000000000011001001

Iteration Number: 95, dFrac: 4.561e-14
         dReg                     dRes                   dFPU
1.0000000000000452971  1.0000000000000455191  1.0000000000000455191
dReg: +1.000000000000000000000000000000000000000000000011001100
dRes: +1.000000000000000000000000000000000000000000000011001101
dFPU: +1.000000000000000000000000000000000000000000000011001101

Iteration Number: 96, dFrac: 4.657e-14
         dReg                     dRes                   dFPU
1.0000000000000461853  1.0000000000000466294  1.0000000000000466294
dReg: +1.000000000000000000000000000000000000000000000011010000
dRes: +1.000000000000000000000000000000000000000000000011010010
dFPU: +1.000000000000000000000000000000000000000000000011010010

Iteration Number: 97, dFrac: 4.754e-14
         dReg                     dRes                   dFPU
1.0000000000000470735  1.0000000000000475175  1.0000000000000475175
dReg: +1.000000000000000000000000000000000000000000000011010100
dRes: +1.000000000000000000000000000000000000000000000011010110
dFPU: +1.000000000000000000000000000000000000000000000011010110

Iteration Number: 98, dFrac: 4.852e-14
         dReg                     dRes                   dFPU
1.0000000000000479616  1.0000000000000484057  1.0000000000000484057
dReg: +1.000000000000000000000000000000000000000000000011011000
dRes: +1.000000000000000000000000000000000000000000000011011010
dFPU: +1.000000000000000000000000000000000000000000000011011010

Iteration Number: 99, dFrac: 4.951e-14
         dReg                     dRes                   dFPU
1.0000000000000488498  1.0000000000000495159  1.0000000000000495159
dReg: +1.000000000000000000000000000000000000000000000011011100
dRes: +1.000000000000000000000000000000000000000000000011011111
dFPU: +1.000000000000000000000000000000000000000000000011011111

Iteration Number: 100, dFrac: 5.051e-14
         dReg                     dRes                   dFPU
 1.00000000000004996   1.0000000000000504041  1.0000000000000504041
dReg: +1.000000000000000000000000000000000000000000000011100001
dRes: +1.000000000000000000000000000000000000000000000011100011
dFPU: +1.000000000000000000000000000000000000000000000011100011
```

Appendix AE-2 CHFort Object Code of Test Case Five, Experiment One

```
<<variablesvalues>>
Name: Y
Value:
IsHistory: 1
IsFixed: 0
DataType: 5
nDimsCount: 0
<<ProgramCode>>
Label StartProg
BeginCode A
push Number 1E-17
EndCode
BeginCode Y
push Number 1
EndCode
BeginCode N
push Number 0
EndCode
Label 100
BeginBoolCode dIf_Bool_1_0
  push String N
  push Number 100
  Minus
EndCode
GoToCond 900 GreaterOrEqual dIf_Bool_1_0
BeginCode Y
  push String Y
  push String A
  Plus
EndCode
BeginCode N
  push String N
  push Number 1
  Plus
EndCode
BeginCode A
  push String A
  push Number 1E-17
  Plus
EndCode
GoTo 100
Label 900
Label EndProg
```

Appendix AE-3 Results of Test Case Five, Experiment Two

```
Iteration Number: 1, dFrac: 1.0000000000000001e-17
         dReg                     dRes                   dFPU
          1                      1                      1
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000000000
dFPU: +1.0000000000000000000000000000000000000000000000000000

Iteration Number: 2, dFrac: 3.0000000000000001e-17
         dReg                     dRes                   dFPU
          1                      1                      1
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000000000
dFPU: +1.0000000000000000000000000000000000000000000000000000

Iteration Number: 3, dFrac: 7.0000000000000003e-17
         dReg                     dRes                   dFPU
          1                      1                      1
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000000000
dFPU: +1.0000000000000000000000000000000000000000000000000000

Iteration Number: 4, dFrac: 1.5000000000000002e-16
         dReg                     dRes                   dFPU
          1             1.000000000000000222   1.000000000000000222
dReg: +1.0000000000000000000000000000000000000000000000000000
dRes: +1.0000000000000000000000000000000000000000000000000001
dFPU: +1.0000000000000000000000000000000000000000000000000001

Iteration Number: 5, dFrac: 3.1000000000000001e-16
         dReg                     dRes                   dFPU
 1.000000000000000222   1.000000000000000222   1.000000000000000222
dReg: +1.0000000000000000000000000000000000000000000000000001
dRes: +1.0000000000000000000000000000000000000000000000000001
dFPU: +1.0000000000000000000000000000000000000000000000000001

Iteration Number: 6, dFrac: 6.3000000000000008e-16
         dReg                     dRes                   dFPU
1.0000000000000004441  1.0000000000000006661  1.0000000000000006661
dReg: +1.0000000000000000000000000000000000000000000000000010
dRes: +1.0000000000000000000000000000000000000000000000000011
dFPU: +1.0000000000000000000000000000000000000000000000000011

Iteration Number: 7, dFrac: 1.2700000000000001e-15
         dReg                     dRes                   dFPU
1.0000000000000011102  1.0000000000000013323  1.0000000000000013323
dReg: +1.0000000000000000000000000000000000000000000000000101
dRes: +1.0000000000000000000000000000000000000000000000000110
dFPU: +1.0000000000000000000000000000000000000000000000000110

Iteration Number: 8, dFrac: 2.55e-15
         dReg                     dRes                   dFPU
1.0000000000000024425  1.0000000000000024425  1.0000000000000024425
dReg: +1.0000000000000000000000000000000000000000000000001011
dRes: +1.0000000000000000000000000000000000000000000000001011
dFPU: +1.0000000000000000000000000000000000000000000000001011

Iteration Number: 9, dFrac: 5.1100000000000006e-15
         dReg                     dRes                   dFPU
 1.000000000000005107   1.000000000000005107   1.000000000000005107
dReg: +1.0000000000000000000000000000000000000000000000010111
dRes: +1.0000000000000000000000000000000000000000000000010111
dFPU: +1.0000000000000000000000000000000000000000000000010111

Iteration Number: 10, dFrac: 1.0230000000000001e-14
         dReg                     dRes                   dFPU
1.0000000000000102141  1.0000000000000102141  1.0000000000000102141
dReg: +1.0000000000000000000000000000000000000000000000101110
dRes: +1.0000000000000000000000000000000000000000000000101110
```

```
dFPU: +1.0000000000000000000000000000000000000000000000101110

Iteration Number: 11, dFrac: 2.047e-14
         dReg                     dRes                   dFPU
1.0000000000000204281  1.0000000000000204281  1.0000000000000204281
dReg: +1.0000000000000000000000000000000000000000000001011100
dRes: +1.0000000000000000000000000000000000000000000001011100
dFPU: +1.0000000000000000000000000000000000000000000001011100

Iteration Number: 12, dFrac: 4.0950000000000005e-14
         dReg                     dRes                   dFPU
1.0000000000000408562  1.0000000000000408562  1.0000000000000408562
dReg: +1.0000000000000000000000000000000000000000000010111000
dRes: +1.0000000000000000000000000000000000000000000010111000
dFPU: +1.0000000000000000000000000000000000000000000010111000

Iteration Number: 13, dFrac: 8.1910000000000001e-14
         dReg                     dRes                   dFPU
1.0000000000000817124  1.0000000000000819345  1.0000000000000819345
dReg: +1.0000000000000000000000000000000000000000000101110000
dRes: +1.0000000000000000000000000000000000000000000101110001
dFPU: +1.0000000000000000000000000000000000000000000101110001

Iteration Number: 14, dFrac: 1.6383000000000002e-13
         dReg                     dRes                   dFPU
1.0000000000001636469  1.0000000000001638689  1.0000000000001638689
dReg: +1.0000000000000000000000000000000000000000001011100001
dRes: +1.0000000000000000000000000000000000000000001011100010
dFPU: +1.0000000000000000000000000000000000000000001011100010

Iteration Number: 15, dFrac: 3.2767000000000001e-13
         dReg                     dRes                   dFPU
1.0000000000003275158  1.0000000000003277378  1.0000000000003277378
dReg: +1.0000000000000000000000000000000000000000010111000011
dRes: +1.0000000000000000000000000000000000000000010111000100
dFPU: +1.0000000000000000000000000000000000000000010111000100

Iteration Number: 16, dFrac: 6.5535000000000003e-13
         dReg                     dRes                   dFPU
1.0000000000006552536  1.0000000000006552536  1.0000000000006552536
dReg: +1.0000000000000000000000000000000000000000101110000111
dRes: +1.0000000000000000000000000000000000000000101110000111
dFPU: +1.0000000000000000000000000000000000000000101110000111

Iteration Number: 17, dFrac: 1.3107100000000002e-12
         dReg                     dRes                   dFPU
1.0000000000013105073  1.0000000000013107293  1.0000000000013107293
dReg: +1.0000000000000000000000000000000000000001011100001110
dRes: +1.0000000000000000000000000000000000000001011100001111
dFPU: +1.0000000000000000000000000000000000000001011100001111

Iteration Number: 18, dFrac: 2.6214300000000001e-12
         dReg                     dRes                   dFPU
1.0000000000026212366  1.0000000000026214586  1.0000000000026214586
dReg: +1.0000000000000000000000000000000000000010111000011101
dRes: +1.0000000000000000000000000000000000000010111000011110
dFPU: +1.0000000000000000000000000000000000000010111000011110

Iteration Number: 19, dFrac: 5.2428700000000003e-12
         dReg                     dRes                   dFPU
1.0000000000052426952  1.0000000000052429172  1.0000000000052429172
dReg: +1.0000000000000000000000000000000000000101110000111011
dRes: +1.0000000000000000000000000000000000000101110000111100
dFPU: +1.0000000000000000000000000000000000000101110000111100

Iteration Number: 20, dFrac: 1.0485750000000001e-11
         dReg                     dRes                   dFPU
1.0000000000104856124  1.0000000000104858344  1.0000000000104858344
dReg: +1.0000000000000000000000000000000000001011100001110111
dRes: +1.0000000000000000000000000000000000001011100001111000
dFPU: +1.0000000000000000000000000000000000001011100001111000
```

```
Iteration Number: 21, dFrac: 2.0971510000000002e-11
         dReg                     dRes                   dFPU
1.0000000000209714468  1.0000000000209714468  1.0000000000209714468
dReg: +1.000000000000000000000000000000000000010111000011101111
dRes: +1.000000000000000000000000000000000000010111000011101111
dFPU: +1.000000000000000000000000000000000000010111000011101111

Iteration Number: 22, dFrac: 4.1943030000000004e-11
         dReg                     dRes                   dFPU
1.0000000000419428936  1.0000000000419431156  1.0000000000419431156
dReg: +1.0000000000000000000000000000000000101110000111011110
dRes: +1.0000000000000000000000000000000000101110000111011111
dFPU: +1.0000000000000000000000000000000000101110000111011111

Iteration Number: 23, dFrac: 8.388607e-11
         dReg                     dRes                   dFPU
1.0000000000838860093  1.0000000000838860093  1.0000000000838860093
dReg: +1.0000000000000000000000000000000001011100001110111101
dRes: +1.0000000000000000000000000000000001011100001110111101
dFPU: +1.0000000000000000000000000000000001011100001110111101

Iteration Number: 24, dFrac: 1.6777215000000002e-10
         dReg                     dRes                   dFPU
1.0000000001677720185  1.0000000001677722405  1.0000000001677722405
dReg: +1.0000000000000000000000000000000010111000011101111010
dRes: +1.0000000000000000000000000000000010111000011101111011
dFPU: +1.0000000000000000000000000000000010111000011101111011

Iteration Number: 25, dFrac: 3.3554431000000003e-10
         dReg                     dRes                   dFPU
 1.000000000335544259   1.000000000335544259   1.000000000335544259
dReg: +1.0000000000000000000000000000000101110000111011110101
dRes: +1.0000000000000000000000000000000101110000111011110101
dFPU: +1.0000000000000000000000000000000101110000111011110101

Iteration Number: 26, dFrac: 6.7108863e-10
         dReg                     dRes                   dFPU
1.0000000006710885181  1.0000000006710887401  1.0000000006710887401
dReg: +1.0000000000000000000000000000001011100001110111101010
dRes: +1.0000000000000000000000000000001011100001110111101011
dFPU: +1.0000000000000000000000000000001011100001110111101011

Iteration Number: 27, dFrac: 1.3421772700000001e-09
         dReg                     dRes                   dFPU
1.0000000013421772582  1.0000000013421772582  1.0000000013421772582
dReg: +1.0000000000000000000000000000010111000011101111010101
dRes: +1.0000000000000000000000000000010111000011101111010101
dFPU: +1.0000000000000000000000000000010111000011101111010101

Iteration Number: 28, dFrac: 2.6843545500000004e-09
         dReg                     dRes                   dFPU
1.0000000026843545164  1.0000000026843545164  1.0000000026843545164
dReg: +1.0000000000000000000000000000101110000111011110101010
dRes: +1.0000000000000000000000000000101110000111011110101010
dFPU: +1.0000000000000000000000000000101110000111011110101010

Iteration Number: 29, dFrac: 5.3687091100000005e-09
         dReg                     dRes                   dFPU
1.0000000053687090329  1.0000000053687090329  1.0000000053687090329
dReg: +1.0000000000000000000000000001011100001110111101010100
dRes: +1.0000000000000000000000000001011100001110111101010100
dFPU: +1.0000000000000000000000000001011100001110111101010100

Iteration Number: 30, dFrac: 1.0737418230000001e-08
         dReg                     dRes                   dFPU
1.0000000107374180658  1.0000000107374182878  1.0000000107374182878
dReg: +1.0000000000000000000000000010111000011101111010101000
dRes: +1.0000000000000000000000000010111000011101111010101001
dFPU: +1.0000000000000000000000000010111000011101111010101001
```

```
Iteration Number: 31, dFrac: 2.147483647e-08
         dReg                     dRes                   dFPU
1.0000000214748363536  1.0000000214748365757  1.0000000214748365757
dReg: +1.0000000000000000000000000101110000111011110101010001
dRes: +1.0000000000000000000000000101110000111011110101010010
dFPU: +1.0000000000000000000000000101110000111011110101010010

Iteration Number: 32, dFrac: 4.2949672950000005e-08
         dReg                     dRes                   dFPU
1.0000000429496729293  1.0000000429496729293  1.0000000429496729293
dReg: +1.0000000000000000000000001011100001110111101010100011
dRes: +1.0000000000000000000000001011100001110111101010100011
dFPU: +1.0000000000000000000000001011100001110111101010100011

Iteration Number: 33, dFrac: 8.5899345910000001e-08
         dReg                     dRes                   dFPU
1.0000000858993458586  1.0000000858993458586  1.0000000858993458586
dReg: +1.0000000000000000000000010111000011101111010101000110
dRes: +1.0000000000000000000000010111000011101111010101000110
dFPU: +1.0000000000000000000000010111000011101111010101000110

Iteration Number: 34, dFrac: 1.7179869183000002e-07
         dReg                     dRes                   dFPU
1.0000001717986917171  1.0000001717986919392  1.0000001717986919392
dReg: +1.0000000000000000000000101110000111011110101010001100
dRes: +1.0000000000000000000000101110000111011110101010001101
dFPU: +1.0000000000000000000000101110000111011110101010001101

Iteration Number: 35, dFrac: 3.4359738367000001e-07
         dReg                     dRes                   dFPU
1.0000003435973836563  1.0000003435973836563  1.0000003435973836563
dReg: +1.0000000000000000000001011100001110111101010100011001
dRes: +1.0000000000000000000001011100001110111101010100011001
dFPU: +1.0000000000000000000001011100001110111101010100011001

Iteration Number: 36, dFrac: 6.8719476735000008e-07
         dReg                     dRes                   dFPU
1.0000006871947673126  1.0000006871947673126  1.0000006871947673126
dReg: +1.0000000000000000000010111000011101111010101000110010
dRes: +1.0000000000000000000010111000011101111010101000110010
dFPU: +1.0000000000000000000010111000011101111010101000110010

Iteration Number: 37, dFrac: 1.37438953471e-06
         dReg                     dRes                   dFPU
1.0000013743895346252  1.0000013743895346252  1.0000013743895346252
dReg: +1.0000000000000000000101110000111011110101010001100100
dRes: +1.0000000000000000000101110000111011110101010001100100
dFPU: +1.0000000000000000000101110000111011110101010001100100

Iteration Number: 38, dFrac: 2.7487790694300001e-06
         dReg                     dRes                   dFPU
1.0000027487790692504  1.0000027487790694725  1.0000027487790694725
dReg: +1.0000000000000000001011100001110111101010100011001000
dRes: +1.0000000000000000001011100001110111101010100011001001
dFPU: +1.0000000000000000001011100001110111101010100011001001

Iteration Number: 39, dFrac: 5.4975581388700003e-06
         dReg                     dRes                   dFPU
1.0000054975581387229  1.0000054975581389449  1.0000054975581389449
dReg: +1.0000000000000000010111000011101111010101000110010001
dRes: +1.0000000000000000010111000011101111010101000110010010
dFPU: +1.0000000000000000010111000011101111010101000110010010

Iteration Number: 40, dFrac: 1.0995116277750001e-05
         dReg                     dRes                   dFPU
1.0000109951162776678  1.0000109951162776678  1.0000109951162776678
dReg: +1.0000000000000000101110000111011110101010001100100011
dRes: +1.0000000000000000101110000111011110101010001100100011
dFPU: +1.0000000000000000101110000111011110101010001100100011

Iteration Number: 41, dFrac: 2.1990232555510003e-05
```

```
         dReg                     dRes                   dFPU
1.0000219902325553356  1.0000219902325555577  1.0000219902325555577
dReg: +1.0000000000000001011100001110111101010100011001000110
dRes: +1.0000000000000001011100001110111101010100011001000111
dFPU: +1.0000000000000001011100001110111101010100011001000111

Iteration Number: 42, dFrac: 4.3980465111030001e-05
         dReg                     dRes                   dFPU
1.0000439804651108933  1.0000439804651111153  1.0000439804651111153
dReg: +1.0000000000000010111000011101111010101000110010001101
dRes: +1.0000000000000010111000011101111010101000110010001110
dFPU: +1.0000000000000010111000011101111010101000110010001110

Iteration Number: 43, dFrac: 8.7960930222070005e-05
         dReg                     dRes                   dFPU
1.0000879609302220086  1.0000879609302220086  1.0000879609302220086
dReg: +1.0000000000000101110000111011110101010001100100011011
dRes: +1.0000000000000101110000111011110101010001100100011011
dFPU: +1.0000000000000101110000111011110101010001100100011011

Iteration Number: 44, dFrac: 0.00017592186044415001
         dReg                     dRes                   dFPU
1.0001759218604440171  1.0001759218604442392  1.0001759218604442392
dReg: +1.0000000000001011100001110111101010100011001000110110
dRes: +1.0000000000001011100001110111101010100011001000110111
dFPU: +1.0000000000001011100001110111101010100011001000110111

Iteration Number: 45, dFrac: 0.00035184372088831005
         dReg                     dRes                   dFPU
1.0003518437208882563  1.0003518437208882563  1.0003518437208882563
dReg: +1.0000000000010111000011101111010101000110010001101101
dRes: +1.0000000000010111000011101111010101000110010001101101
dFPU: +1.0000000000010111000011101111010101000110010001101101

Iteration Number: 46, dFrac: 0.00070368744177663008
         dReg                     dRes                   dFPU
1.0007036874417765127  1.0007036874417767347  1.0007036874417767347
dReg: +1.0000000000101110000111011110101010001100100011011010
dRes: +1.0000000000101110000111011110101010001100100011011011
dFPU: +1.0000000000101110000111011110101010001100100011011011

Iteration Number: 47, dFrac: 0.0014073748835532701
         dReg                     dRes                   dFPU
1.0014073748835532474  1.0014073748835532474  1.0014073748835532474
dReg: +1.0000000001011100001110111101010100011001000110110101
dRes: +1.0000000001011100001110111101010100011001000110110101
dFPU: +1.0000000001011100001110111101010100011001000110110101

Iteration Number: 48, dFrac: 0.0028147497671065502
         dReg                     dRes                   dFPU
1.0028147497671064947  1.0028147497671064947  1.0028147497671064947
dReg: +1.0000000010111000011101111010101000110010001101101010
dRes: +1.0000000010111000011101111010101000110010001101101010
dFPU: +1.0000000010111000011101111010101000110010001101101010

Iteration Number: 49, dFrac: 0.00562949953421311
         dReg                     dRes                   dFPU
1.0056294995342129894  1.0056294995342132115  1.0056294995342132115
dReg: +1.0000000101110000111011110101010001100100011011010100
dRes: +1.0000000101110000111011110101010001100100011011010101
dFPU: +1.0000000101110000111011110101010001100100011011010101

Iteration Number: 50, dFrac: 0.01125899906842623
         dReg                     dRes                   dFPU
 1.011258999068426201   1.011258999068426201   1.011258999068426201
dReg: +1.0000001011100001110111101010100011001000110110101001
dRes: +1.0000001011100001110111101010100011001000110110101001
dFPU: +1.0000001011100001110111101010100011001000110110101001

Iteration Number: 51, dFrac: 0.022517998136852471
         dReg                     dRes                   dFPU
 1.022517998136852402   1.022517998136852402   1.022517998136852402
dReg: +1.0000010111000011101111010101000110010001101101010010
dRes: +1.0000010111000011101111010101000110010001101101010010
dFPU: +1.0000010111000011101111010101000110010001101101010010

Iteration Number: 52, dFrac: 0.045035996273704956
         dReg                     dRes                    dFPU
 1.045035996273704804   1.045035996273705026   1.045035996273705026
dReg: +1.0000101110000111011110101010001100100011011010100100
dRes: +1.0000101110000111011110101010001100100011011010100101
dFPU: +1.0000101110000111011110101010001100100011011010100101

Iteration Number: 53, dFrac: 0.090071992547409913
         dReg                     dRes                    dFPU
 1.090071992547409829   1.090071992547409829   1.090071992547409829
dReg: +1.0001011100001110111101010100011001000110110101001001
dRes: +1.0001011100001110111101010100011001000110110101001001
dFPU: +1.0001011100001110111101010100011001000110110101001001

Iteration Number: 54, dFrac: 0.18014398509481985
         dReg                     dRes                    dFPU
 1.180143985094819659   1.180143985094819881   1.180143985094819881
dReg: +1.0010111000011101111010101000110010001101101010010010
dRes: +1.0010111000011101111010101000110010001101101010010011
dFPU: +1.0010111000011101111010101000110010001101101010010011

Iteration Number: 55, dFrac: 0.36028797018963971
         dReg                     dRes                    dFPU
 1.360287970189639539   1.360287970189639761   1.360287970189639761
dReg: +1.0101110000111011110101010001100100011011010100100101
dRes: +1.0101110000111011110101010001100100011011010100100110
dFPU: +1.0101110000111011110101010001100100011011010100100110

Iteration Number: 56, dFrac: 0.72057594037927941
         dReg                     dRes                    dFPU
  1.7205759403792793     1.7205759403792793     1.7205759403792793
dReg: +1.1011100001110111101010100011001000110110101001001011
dRes: +1.1011100001110111101010100011001000110110101001001011
dFPU: +1.1011100001110111101010100011001000110110101001001011

Iteration Number: 57, dFrac: 1.4411518807585588
         dReg                     dRes                    dFPU
 2.441151880758558601   2.441151880758558601   2.441151880758558601
dReg: +10.011100001110111101010100011001000110110101001001011
dRes: +10.011100001110111101010100011001000110110101001001011
dFPU: +10.011100001110111101010100011001000110110101001001011

Iteration Number: 58, dFrac: 2.8823037615171176
         dReg                     dRes                    dFPU
 3.882303761517117202   3.882303761517117202   3.882303761517117646
dReg: +11.111000011101111010101000110010001101101010010010110
dRes: +11.111000011101111010101000110010001101101010010010110
dFPU: +11.111000011101111010101000110010001101101010010010111

Iteration Number: 59, dFrac: 5.7646075230342353
         dReg                     dRes                    dFPU
 6.764607523034234404   6.764607523034234404   6.764607523034235292
dReg: +110.11000011101111010101000110010001101101010010010110
dRes: +110.11000011101111010101000110010001101101010010010110
dFPU: +110.11000011101111010101000110010001101101010010010111

Iteration Number: 60, dFrac: 11.529215046068471
         dReg                     dRes                    dFPU
 12.52921504606846881   12.52921504606846881   12.52921504606847058
dReg: +1100.1000011101111010101000110010001101101010010010110
dRes: +1100.1000011101111010101000110010001101101010010010110
dFPU: +1100.1000011101111010101000110010001101101010010010111

Iteration Number: 61, dFrac: 23.058430092136941
         dReg                     dRes                    dFPU
 24.05843009213693762   24.05843009213693762   24.05843009213694117
```

```
dReg: +11000.000011101111010101000110010001101101010010010110
dRes: +11000.000011101111010101000110010001101101010010010110
dFPU: +11000.000011101111010101000110010001101101010010010111

Iteration Number: 62, dFrac: 46.116860184273882
         dReg                    dRes                   dFPU
 47.11686018427387524   47.11686018427387524   47.11686018427388234
dReg: +101111.00011101111010101000110010001101101010010010110
dRes: +101111.00011101111010101000110010001101101010010010110
dFPU: +101111.00011101111010101000110010001101101010010010111

Iteration Number: 63, dFrac: 92.233720368547765
         dReg                    dRes                   dFPU
 93.23372036854775047   93.23372036854775047   93.23372036854776468
dReg: +1011101.0011101111010101000110010001101101010010010110
dRes: +1011101.0011101111010101000110010001101101010010010110
dFPU: +1011101.0011101111010101000110010001101101010010010111

Iteration Number: 64, dFrac: 184.46744073709553
         dReg                    dRes                   dFPU
 185.4674407370955009   185.4674407370955009   185.4674407370955294
dReg: +10111001.011101111010101000110010001101101010010010110
dRes: +10111001.011101111010101000110010001101101010010010110
dFPU: +10111001.011101111010101000110010001101101010010010111

Iteration Number: 65, dFrac: 368.93488147419106
         dReg                    dRes                   dFPU
 369.9348814741910019   369.9348814741910019   369.9348814741910587
dReg: +101110001.11101111010101000110010001101101010010010110
dRes: +101110001.11101111010101000110010001101101010010010110
dFPU: +101110001.11101111010101000110010001101101010010010111

Iteration Number: 66, dFrac: 737.86976294838212
         dReg                    dRes                   dFPU
 738.8697629483820038   738.8697629483820038   738.8697629483821174
dReg: +1011100010.1101111010101000110010001101101010010010110
dRes: +1011100010.1101111010101000110010001101101010010010110
dFPU: +1011100010.1101111010101000110010001101101010010010111

Iteration Number: 67, dFrac: 1475.7395258967642
         dReg                    dRes                   dFPU
 1476.739525896764008   1476.739525896764008   1476.739525896764235
dReg: +10111000100.101111010101000110010001101101010010010110
dRes: +10111000100.101111010101000110010001101101010010010110
dFPU: +10111000100.101111010101000110010001101101010010010111

Iteration Number: 68, dFrac: 2951.4790517935285
         dReg                    dRes                   dFPU
 2952.479051793528015   2952.479051793528015   2952.47905179352847
dReg: +101110001000.01111010101000110010001101101010010010110
dRes: +101110001000.01111010101000110010001101101010010010110
dFPU: +101110001000.01111010101000110010001101101010010010111

Iteration Number: 69, dFrac: 5902.9581035870569
         dReg                    dRes                   dFPU
 5903.95810358705603    5903.95810358705603    5903.95810358705694
dReg: +1011100001111.1111010101000110010001101101010010010110
dRes: +1011100001111.1111010101000110010001101101010010010110
dFPU: +1011100001111.1111010101000110010001101101010010010111

Iteration Number: 70, dFrac: 11805.916207174114
         dReg                    dRes                   dFPU
 11806.91620717411206   11806.91620717411206   11806.91620717411388
dReg: +10111000011110.111010101000110010001101101010010010110
dRes: +10111000011110.111010101000110010001101101010010010110
dFPU: +10111000011110.111010101000110010001101101010010010111

Iteration Number: 71, dFrac: 23611.832414348228
         dReg                    dRes                   dFPU
 23612.83241434822412   23612.83241434822412   23612.83241434822776
dReg: +101110000111100.11010101000110010001101101010010010110
```

```
dRes: +101110000111100.11010101000110010001101101010010010110
dFPU: +101110000111100.11010101000110010001101101010010010111

Iteration Number: 72, dFrac: 47223.664828696456
         dReg                     dRes                   dFPU
 47224.66482869644824   47224.66482869644824   47224.66482869645552
dReg: +1011100001111000.1010101000110010001101101010010010110
dRes: +1011100001111000.1010101000110010001101101010010010110
dFPU: +1011100001111000.1010101000110010001101101010010010111

Iteration Number: 73, dFrac: 94447.329657392911
         dReg                     dRes                   dFPU
 94448.32965739289648   94448.32965739289648   94448.32965739291103
dReg: +10111000011110000.010101000110010001101101010010010110
dRes: +10111000011110000.010101000110010001101101010010010110
dFPU: +10111000011110000.010101000110010001101101010010010111

Iteration Number: 74, dFrac: 188894.65931478582
         dReg                     dRes                   dFPU
 188895.659314785793    188895.659314785793    188895.6593147858221
dReg: +101110000111011111.10101000110010001101101010010010110
dRes: +101110000111011111.10101000110010001101101010010010110
dFPU: +101110000111011111.10101000110010001101101010010010111

Iteration Number: 75, dFrac: 377789.31862957164
         dReg                     dRes                   dFPU
 377790.3186295715859   377790.3186295715859   377790.3186295716441
dReg: +1011100001110111110.0101000110010001101101010010010110
dRes: +1011100001110111110.0101000110010001101101010010010110
dFPU: +1011100001110111110.0101000110010001101101010010010111

Iteration Number: 76, dFrac: 755578.63725914329
         dReg                     dRes                   dFPU
 755579.6372591431718   755579.6372591431718   755579.6372591432882
dReg: +10111000011101111011.101000110010001101101010010010110
dRes: +10111000011101111011.101000110010001101101010010010110
dFPU: +10111000011101111011.101000110010001101101010010010111

Iteration Number: 77, dFrac: 1511157.2745182866
         dReg                     dRes                   dFPU
 1511158.274518286344   1511158.274518286344   1511158.274518286576
dReg: +101110000111011110110.01000110010001101101010010010110
dRes: +101110000111011110110.01000110010001101101010010010110
dFPU: +101110000111011110110.01000110010001101101010010010111

Iteration Number: 78, dFrac: 3022314.5490365732
         dReg                     dRes                   dFPU
 3022315.549036572687   3022315.549036572687   3022315.549036573153
dReg: +1011100001110111101011.1000110010001101101010010010110
dRes: +1011100001110111101011.1000110010001101101010010010110
dFPU: +1011100001110111101011.1000110010001101101010010010111

Iteration Number: 79, dFrac: 6044629.0980731463
         dReg                     dRes                   dFPU
 6044630.098073145374   6044630.098073145374   6044630.098073146306
dReg: +10111000011101111010110.000110010001101101010010010110
dRes: +10111000011101111010110.000110010001101101010010010110
dFPU: +10111000011101111010110.000110010001101101010010010111

Iteration Number: 80, dFrac: 12089258.196146293
         dReg                     dRes                   dFPU
 12089259.19614629075   12089259.19614629075   12089259.19614629261
dReg: +101110000111011110101011.00110010001101101010010010110
dRes: +101110000111011110101011.00110010001101101010010010110
dFPU: +101110000111011110101011.00110010001101101010010010111

Iteration Number: 81, dFrac: 24178516.392292585
         dReg                     dRes                   dFPU
 24178517.3922925815    24178517.3922925815    24178517.39229258522
dReg: +1011100001110111101010101.0110010001101101010010010110
dRes: +1011100001110111101010101.0110010001101101010010010110
```

```
dFPU: +1011100001110111101010101.0110010001101101010010010111

Iteration Number: 82, dFrac: 48357032.78458517
         dReg                     dRes                   dFPU
  48357033.784585163     48357033.784585163    48357033.78458517045
dReg: +10111000011101111010101001.110010001101101010010010110
dRes: +10111000011101111010101001.110010001101101010010010110
dFPU: +10111000011101111010101001.110010001101101010010010111

Iteration Number: 83, dFrac: 96714065.569170341
         dReg                     dRes                   dFPU
 96714066.56917032599   96714066.56917032599   96714066.5691703409
dReg: +101110000111011110101010010.10010001101101010010010110
dRes: +101110000111011110101010010.10010001101101010010010110
dFPU: +101110000111011110101010010.10010001101101010010010111

Iteration Number: 84, dFrac: 193428131.13834068
         dReg                     dRes                   dFPU
 193428132.138340652    193428132.138340652    193428132.1383406818
dReg: +1011100001110111101010100100.0010001101101010010010110
dRes: +1011100001110111101010100100.0010001101101010010010110
dFPU: +1011100001110111101010100100.0010001101101010010010111

Iteration Number: 85, dFrac: 386856262.27668136
         dReg                     dRes                   dFPU
 386856263.276681304    386856263.276681304    386856263.2766813636
dReg: +10111000011101111010101000111.010001101101010010010110
dRes: +10111000011101111010101000111.010001101101010010010110
dFPU: +10111000011101111010101000111.010001101101010010010111

Iteration Number: 86, dFrac: 773712524.55336273
         dReg                     dRes                   dFPU
 773712525.553362608    773712525.553362608    773712525.5533627272
dReg: +101110000111011110101010001101.10001101101010010010110
dRes: +101110000111011110101010001101.10001101101010010010110
dFPU: +101110000111011110101010001101.10001101101010010010111

Iteration Number: 87, dFrac: 1547425049.1067255
         dReg                     dRes                   dFPU
 1547425050.106725216   1547425050.106725216   1547425050.106725454
dReg: +1011100001110111101010100011010.0001101101010010010110
dRes: +1011100001110111101010100011010.0001101101010010010110
dFPU: +1011100001110111101010100011010.0001101101010010010111

Iteration Number: 88, dFrac: 3094850098.2134509
         dReg                     dRes                   dFPU
 3094850099.213450432   3094850099.213450432   3094850099.213450909
dReg: +10111000011101111010101000110011.001101101010010010110
dRes: +10111000011101111010101000110011.001101101010010010110
dFPU: +10111000011101111010101000110011.001101101010010010111

Iteration Number: 89, dFrac: 6189700196.4269018
         dReg                     dRes                   dFPU
 6189700197.426900864   6189700197.426900864   6189700197.426901818
dReg: +101110000111011110101010001100101.01101101010010010110
dRes: +101110000111011110101010001100101.01101101010010010110
dFPU: +101110000111011110101010001100101.01101101010010010111

Iteration Number: 90, dFrac: 12379400392.853804
         dReg                     dRes                   dFPU
 12379400393.85380173   12379400393.85380173   12379400393.85380364
dReg: +1011100001110111101010100011001001.1101101010010010110
dRes: +1011100001110111101010100011001001.1101101010010010110
dFPU: +1011100001110111101010100011001001.1101101010010010111

Iteration Number: 91, dFrac: 24758800785.707607
         dReg                     dRes                   dFPU
 24758800786.70760346   24758800786.70760346   24758800786.70760727
dReg: +10111000011101111010101000110010010.101101010010010110
dRes: +10111000011101111010101000110010010.101101010010010110
dFPU: +10111000011101111010101000110010010.101101010010010111
```

```
Iteration Number: 92, dFrac: 49517601571.415215
         dReg                     dRes                   dFPU
 49517601572.41520691   49517601572.41520691   49517601572.41521454
dReg: +101110000111011110101010001100100100.01101010010010110
dRes: +101110000111011110101010001100100100.01101010010010110
dFPU: +101110000111011110101010001100100100.01101010010010111

Iteration Number: 93, dFrac: 99035203142.830429
         dReg                     dRes                   dFPU
 99035203143.83041382   99035203143.83041382   99035203143.83042908
dReg: +1011100001110111101010100011001000111.1101010010010110
dRes: +1011100001110111101010100011001000111.1101010010010110
dFPU: +1011100001110111101010100011001000111.1101010010010111

Iteration Number: 94, dFrac: 198070406285.66086
         dReg                     dRes                   dFPU
 198070406286.6608276   198070406286.6608276   198070406286.6608582
dReg: +10111000011101111010101000110010001110.101010010010110
dRes: +10111000011101111010101000110010001110.101010010010110
dFPU: +10111000011101111010101000110010001110.101010010010111

Iteration Number: 95, dFrac: 396140812571.32172
         dReg                     dRes                   dFPU
 396140812572.3216553   396140812572.3216553   396140812572.3217163
dReg: +101110000111011110101010001100100011100.01010010010110
dRes: +101110000111011110101010001100100011100.01010010010110
dFPU: +101110000111011110101010001100100011100.01010010010111

Iteration Number: 96, dFrac: 792281625142.64343
         dReg                     dRes                   dFPU
 792281625143.6433106   792281625143.6433106   792281625143.6434326
dReg: +1011100001110111101010100011001000110111.1010010010110
dRes: +1011100001110111101010100011001000110111.1010010010110
dFPU: +1011100001110111101010100011001000110111.1010010010111

Iteration Number: 97, dFrac: 1584563250285.2869
         dReg                     dRes                   dFPU
 1584563250286.286621   1584563250286.286621   1584563250286.286865
dReg: +10111000011101111010101000110010001101110.010010010110
dRes: +10111000011101111010101000110010001101110.010010010110
dFPU: +10111000011101111010101000110010001101110.010010010111

Iteration Number: 98, dFrac: 3169126500570.5737
         dReg                     dRes                   dFPU
 3169126500571.573242   3169126500571.573242   3169126500571.57373
dReg: +101110000111011110101010001100100011011011.10010010110
dRes: +101110000111011110101010001100100011011011.10010010110
dFPU: +101110000111011110101010001100100011011011.10010010111

Iteration Number: 99, dFrac: 6338253001141.1475
         dReg                     dRes                   dFPU
 6338253001142.146484   6338253001142.146484   6338253001142.147461
dReg: +1011100001110111101010100011001000110110110.0010010110
dRes: +1011100001110111101010100011001000110110110.0010010110
dFPU: +1011100001110111101010100011001000110110110.0010010111

Iteration Number: 100, dFrac: 12676506002282.295
         dReg                     dRes                   dFPU
 12676506002283.29297   12676506002283.29297   12676506002283.29492
dReg: +10111000011101111010101000110010001101101011.010010110
dRes: +10111000011101111010101000110010001101101011.010010110
dFPU: +10111000011101111010101000110010001101101011.010010111
```

Appendix AE-4 CHFort Object Code of Test Case Five, Experiment Two

```
<<variablesvalues>>
Name: Y
Value:
IsHistory: 1
IsFixed: 0
DataType: 5
nDimsCount: 0
<<ProgramCode>>
Label StartProg
BeginCode A
push Number 1E-17
EndCode
BeginCode Y
push Number 1
EndCode
BeginCode N
push Number 0
EndCode
Label 100
BeginBoolCode dIf_Bool_1_0
  push String N
  push Number 100
  Minus
EndCode
GoToCond 900 GreaterOrEqual dIf_Bool_1_0
BeginCode Y
  push String Y
  push String A
  Plus
EndCode
BeginCode N
  push String N
  push Number 1
  Plus
EndCode
BeginCode A
  push String A
  push String A
  Plus
EndCode
GoTo 100
Label 900
Label EndProg
```

Appendix AF-1- CHFort Object Code of Test Case Six, Experiment One

```
<<variablesvalues>>
Name: X
Value:
IsHistory: 1
IsFixed: 0
DataType: 5
nDimsCount: 0
Name: Xp
Value:
IsHistory: 1
IsFixed: 0
DataType: 5
nDimsCount: 0
<<ProgramCode>>
Label StartProg
BeginCode B
push Number 9
EndCode
BeginCode n
push Number 0
EndCode
BeginCode X
push String B
EndCode
Label 100
BeginCode Xp
  push String X
  push String B
  push String X
  DivSingle
  Plus
  push Number 2
  DivSingle
EndCode
BeginCode X
push String XP
EndCode
BeginCode n
  push String n
  push Number 1
  Plus
EndCode
BeginBoolCode dIf_Bool_1_0
  push String n
  push Number 10
  Minus
EndCode
GoToCond 100 LessOrEqual dIf_Bool_1_0
Label EndProg
```

Appendix AF-2- Floating-Point Unit Code for Iteration Six, Experiment One

```
{ /* proc_13- Iteration 6 of Test Case 6, Experiment 1 */
    IEEE754Real8_struct d0 = {0x0l, 0x40220000l}; /*                         9 */
    IEEE754Real8_struct d1 = {0x0l, 0x40000000l}; /*                         2 */
    IEEE754Real8_struct dRes2;
    IEEE754Real8_struct dRes6;
    IEEE754Real8_struct dRes9;
    IEEE754Real8_struct dRes14;
    IEEE754Real8_struct dRes17;
    IEEE754Real8_struct dRes21;
    IEEE754Real8_struct dRes24;
    IEEE754Real8_struct dRes30;
    IEEE754Real8_struct dRes33;
    IEEE754Real8_struct dRes37;
    IEEE754Real8_struct dRes40;
    IEEE754Real8_struct dRes45;
    IEEE754Real8_struct dRes48;
    IEEE754Real8_struct dRes52;
    IEEE754Real8_struct dRes55;
    IEEE754Real8_struct dRes62;
    char *pszBits;
    { /* add to Area so far */
    asm
        { /* Do FPU stuff */
            fld  d0; /*                         9 */
            fdiv  d0; /*                         9 */
            fadd d0; /*                         9 */
            fdiv d1; /*                         2 */
            fld  d0; /*                         9 */
            fdiv  d0; /*                         9 */
            fadd d0; /*                         9 */
            fdiv d1; /*                         2 */
            fdivr d0; /*                         9 */
            fadd;   /* Parent of top two of stack */
            fdiv d1; /*                         2 */
            fdivr d0; /*                         9 */
            fstp dRes2; /* Result  */
            fld  d0; /*                         9 */
            fdiv  d0; /*                         9 */
            fadd d0; /*                         9 */
            fdiv d1; /*                         2 */
            fld  d0; /*                         9 */
            fdiv  d0; /*                         9 */
            fadd d0; /*                         9 */
            fdiv d1; /*                         2 */
            fdivr d0; /*                         9 */
            fadd;   /* Parent of top two of stack */
            fdiv d1; /*                         2 */
            fadd  dRes2; /* Result */
            fdiv d1; /*                         2 */
            fdivr d0; /*                         9 */
            fstp dRes6; /* Result  */
            fld  d0; /*                         9 */
            fdiv  d0; /*                         9 */
            fadd d0; /*                         9 */
            fdiv d1; /*                         2 */
            fld  d0; /*                         9 */
            fdiv  d0; /*                         9 */
            fadd d0; /*                         9 */
            fdiv d1; /*                         2 */
            fdivr d0; /*                         9 */
            fadd;   /* Parent of top two of stack */
            fdiv d1; /*                         2 */
            fdivr d0; /*                         9 */
            fstp dRes9; /* Result  */
            fld  d0; /*                         9 */
            fdiv  d0; /*                         9 */
            fadd d0; /*                         9 */
            fdiv d1; /*                         2 */
```

```
fld  d0; /*                         9 */
fdiv  d0; /*                         9 */
fadd d0; /*                         9 */
fdiv d1; /*                         2 */
fdivr d0; /*                         9 */
fadd;   /* Parent of top two of stack */
fdiv d1; /*                         2 */
fadd  dRes9; /* Result */
fdiv d1; /*                         2 */
fadd  dRes6; /* Result */
fdiv d1; /*                         2 */
fdivr d0; /*                         9 */
fstp dRes14; /* Result  */
fld  d0; /*                         9 */
fdiv  d0; /*                         9 */
fadd d0; /*                         9 */
fdiv d1; /*                         2 */
fld  d0; /*                         9 */
fdiv  d0; /*                         9 */
fadd d0; /*                         9 */
fdiv d1; /*                         2 */
fdivr d0; /*                         9 */
fadd;   /* Parent of top two of stack */
fdiv d1; /*                         2 */
fdivr d0; /*                         9 */
fstp dRes17; /* Result  */
fld  d0; /*                         9 */
fdiv  d0; /*                         9 */
fadd d0; /*                         9 */
fdiv d1; /*                         2 */
fld  d0; /*                         9 */
fdiv  d0; /*                         9 */
fadd d0; /*                         9 */
fdiv d1; /*                         2 */
fdivr d0; /*                         9 */
fadd;   /* Parent of top two of stack */
fdiv d1; /*                         2 */
fadd  dRes17; /* Result */
fdiv d1; /*                         2 */
fdivr d0; /*                         9 */
fstp dRes21; /* Result  */
fld  d0; /*                         9 */
fdiv  d0; /*                         9 */
fadd d0; /*                         9 */
fdiv d1; /*                         2 */
fld  d0; /*                         9 */
fdiv  d0; /*                         9 */
fadd d0; /*                         9 */
fdiv d1; /*                         2 */
fdivr d0; /*                         9 */
fadd;   /* Parent of top two of stack */
fdiv d1; /*                         2 */
fdivr d0; /*                         9 */
fstp dRes24; /* Result  */
fld  d0; /*                         9 */
fdiv  d0; /*                         9 */
fadd d0; /*                         9 */
fdiv d1; /*                         2 */
fld  d0; /*                         9 */
fdiv  d0; /*                         9 */
fadd d0; /*                         9 */
fdiv d1; /*                         2 */
fdivr d0; /*                         9 */
fadd;   /* Parent of top two of stack */
fdiv d1; /*                         2 */
fadd  dRes24; /* Result */
fdiv d1; /*                         2 */
fadd  dRes21; /* Result */
fdiv d1; /*                         2 */
fadd  dRes14; /* Result */
fdiv d1; /*                         2 */
```

```
fdivr d0; /*                         9 */
fstp dRes30; /* Result  */
fld  d0; /*                         9 */
fdiv  d0; /*                         9 */
fadd d0; /*                         9 */
fdiv d1; /*                         2 */
fld  d0; /*                         9 */
fdiv  d0; /*                         9 */
fadd d0; /*                         9 */
fdiv d1; /*                         2 */
fdivr d0; /*                         9 */
fadd;   /* Parent of top two of stack */
fdiv d1; /*                         2 */
fdivr d0; /*                         9 */
fstp dRes33; /* Result  */
fld  d0; /*                         9 */
fdiv  d0; /*                         9 */
fadd d0; /*                         9 */
fdiv d1; /*                         2 */
fld  d0; /*                         9 */
fdiv  d0; /*                         9 */
fadd d0; /*                         9 */
fdiv d1; /*                         2 */
fdivr d0; /*                         9 */
fadd;   /* Parent of top two of stack */
fdiv d1; /*                         2 */
fadd  dRes33; /* Result */
fdiv d1; /*                         2 */
fdivr d0; /*                         9 */
fstp dRes37; /* Result  */
fld  d0; /*                         9 */
fdiv  d0; /*                         9 */
fadd d0; /*                         9 */
fdiv d1; /*                         2 */
fld  d0; /*                         9 */
fdiv  d0; /*                         9 */
fadd d0; /*                         9 */
fdiv d1; /*                         2 */
fdivr d0; /*                         9 */
fadd;   /* Parent of top two of stack */
fdiv d1; /*                         2 */
fdivr d0; /*                         9 */
fstp dRes40; /* Result  */
fld  d0; /*                         9 */
fdiv  d0; /*                         9 */
fadd d0; /*                         9 */
fdiv d1; /*                         2 */
fld  d0; /*                         9 */
fdiv  d0; /*                         9 */
fadd d0; /*                         9 */
fdiv d1; /*                         2 */
fdivr d0; /*                         9 */
fadd;   /* Parent of top two of stack */
fdiv d1; /*                         2 */
fadd  dRes40; /* Result */
fdiv d1; /*                         2 */
fadd  dRes37; /* Result */
fdiv d1; /*                         2 */
fdivr d0; /*                         9 */
fstp dRes45; /* Result  */
fld  d0; /*                         9 */
fdiv  d0; /*                         9 */
fadd d0; /*                         9 */
fdiv d1; /*                         2 */
fld  d0; /*                         9 */
fdiv  d0; /*                         9 */
fadd d0; /*                         9 */
fdiv d1; /*                         2 */
fdivr d0; /*                         9 */
fadd;   /* Parent of top two of stack */
fdiv d1; /*                         2 */
```

```
            fdivr d0; /*                         9 */
            fstp dRes48; /* Result  */
            fld  d0; /*                         9 */
            fdiv  d0; /*                         9 */
            fadd d0; /*                         9 */
            fdiv d1; /*                         2 */
            fld  d0; /*                         9 */
            fdiv  d0; /*                         9 */
            fadd d0; /*                         9 */
            fdiv d1; /*                         2 */
            fdivr d0; /*                         9 */
            fadd;   /* Parent of top two of stack */
            fdiv d1; /*                         2 */
            fadd  dRes48; /* Result */
            fdiv d1; /*                         2 */
            fdivr d0; /*                         9 */
            fstp dRes52; /* Result  */
            fld  d0; /*                         9 */
            fdiv  d0; /*                         9 */
            fadd d0; /*                         9 */
            fdiv d1; /*                         2 */
            fld  d0; /*                         9 */
            fdiv  d0; /*                         9 */
            fadd d0; /*                         9 */
            fdiv d1; /*                         2 */
            fdivr d0; /*                         9 */
            fadd;   /* Parent of top two of stack */
            fdiv d1; /*                         2 */
            fdivr d0; /*                         9 */
            fstp dRes55; /* Result  */
            fld  d0; /*                         9 */
            fdiv  d0; /*                         9 */
            fadd d0; /*                         9 */
            fdiv d1; /*                         2 */
            fld  d0; /*                         9 */
            fdiv  d0; /*                         9 */
            fadd d0; /*                         9 */
            fdiv d1; /*                         2 */
            fdivr d0; /*                         9 */
            fadd;   /* Parent of top two of stack */
            fdiv d1; /*                         2 */
            fadd  dRes55; /* Result */
            fdiv d1; /*                         2 */
            fadd  dRes52; /* Result */
            fdiv d1; /*                         2 */
            fadd  dRes45; /* Result */
            fdiv d1; /*                         2 */
            fadd  dRes30; /* Result */
            fdiv d1; /*                         2 */
            fstp dRes62; /* Result  */
        } /* Do FPU stuff */
    } /* add to Area so far */
    printf("\nCalculation index: 13\n");
    pszBits = procIEE754DblToBin(dRes62.dVal);
    printf("   dFPU: %25.20g, Binary: %s\n", dRes62.dVal, pszBits);
    free(pszBits);
    return 0;
    /* Number of Store Operations: 16 */
} /* proc_13 */
```

Reference List

Aho, A., Sethi, R., and Ulman, J. (1988). *Compilers Principles, Techniques , and Tools.* Addison-Wesley Publishing Company. Reading, Massachusetts.

ANSI/IEEE (1985). *IEEE Standard for Binary Floating-Point*. New York, NY: The Institute of Electrical and Electronics Engineers, Inc.

Backus, J. et al, (1956). "*The Fortran Automatic Coding System for the IBM 704 EDPM®*." International Business Machines Corporation. New York.

Brent, R. (1978). "Algorithm 524: MP: A Fortran Multiple-Precision Arithmetic Package". *ACM Transactions on Mathematical Software (TOMS).* (1978) 71-81.

Brown, W. (1981). "A Simple but Realistic Model of Floating-Point Computation". *ACM Transactions on Mathematical Software Vol. 7 No. 4.* (1981) 445-480.

Bush, B. (1996). "The Perils of Floating Point". *http://www.lahey.com/float.htm.* Lahey Computer Systems, Inc.

Cilie, A. and Corporaal, H. (1999). "Floating Point to Fixed Point Conversion of C Code". *Computational Geometry*. 229-243.

Dyadkin, L. J. (1995). "Multibox Parsers: No More Handwritten Lexical Analyzers." *IEEE Software*, September 1995, 61-67.

Goldberg, D (1991). "What Every Computer Scientist Should Know About Floating-Point Arithmetic." *ACM Computing Survey*, Vol. 23, NO. 1, 6-48.

Hartree, D.R. (1949). "Modern Calculating Machines". *Endeavour VIII*. IBM Archives. April, 1949.

Henrici, C. (1966). "Theoretical and experimental studies on the accumulation of error in the numerical solution of initial value problems for systems of ordinary differential equations". *Methods of Digital Computing*. 36-44.

Hull, T. E., and Swenson, J. R. (1966). "Tests of Probabilistic Models for Propagation of Roundoff Errors." *Communications of the ACM*, Vol. 9, No. 2, 108-113.

Intel Corporation (2002). *Intel IA-32® Architecture Software Developer's Manual Volume 1: Basic Architecture*. Mt. Prospect, Il.

Lahey Computer Systems, Inc. (1997). *Lahey Fortran 90 Language Reference*. Incline Village, NV, Lahey Computer Systems, Inc..

Linz, P. (1970). “Accurate Floating-Point Summation”. Communications of the *ACM Vol. 13 No. 6.* (June, 1970) 361-362.

Lyon, A. (1970). *Dealing With Data*. Pergamon Press. Oxford.

Moore, R.E. (1988). *Reliability in Computing.* Academic Press, Inc. San Diego, CA.

Mutrie, M., Bartels, R., and Char, B. (1992). “An Approach for Floating-point Error Analysis Using Computer Algebra”. *International Conference on Symbolic and Algebraic Computation.* 284-293.

Ouchi, K (1997). “Real/Expr: Implementation of Exact Computation”. *Masters Thesis*. New York University.

Padua, D. (2000). “The Fortran I Compiler”. *Computing in Science and Engineering.* 70-75.

Rash, B. (1981) *Getting Started with the Numeric Data Processor*. Intel Corporation.

Smith, R. T., Minton, R. B. (2002). *Calculus*. The McGraw-Hill Companies, Inc. New York, NY. 251, 384-386.

Sterbenz, P. H. (1974). *Floating Point Computation*. Englewood Cliffs, N.J. Prentice-Hall, Inc. 123-130.

Stoutemyer, D. (1977). “Automatic Error Analysis Using Computer Algebraic Manipulation”. *ACM Transactions on Mathematical Software*. Vol. 3, No. 1, 26-43.