To the Graduate Council:

I am submitting herewith a thesis written by John L. Sigre entitled "Exact Computation Using Extended-Precision Single-Modulus Residue Arithmetic." I have examined the final copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Computer Science.

Robert Todd Gregory

Robert T. Gregory, Major Professor

We have read this thesis
and recommend its acceptance:

Michael G. Thomason

D. Jeagin

Accepted for the Council:

L Evan Peck

Vice Chancellor
Graduate Studies and Research

Thesis
#3
.5597
Cop. 2

EXACT COMPUTATION USING EXTENDED-PRECISION SINGLE-MODULUS RESIDUE ARITHMETIC

A Thesis

Presented for the

Master of Science

Degree

The University of Tennessee, Knoxville

John L. Smyre

March 1983

3066081

Copyright by John L. Smyre 1983
All Rights Reserved

DEDICATION

This work is dedicated to all those who dare to seek the beauty of intellectual discovery.

ACKNOWLEDGEMENTS

I wish especially to thank Dr. Robert T. Gregory for serving as the major professor on my thesis committee. I am deeply grateful for his valuable assistance and continued interest in this work over many months, as well as for the example and inspiration which he unfailingly provided.

My thanks also go to Dr. Michael Thomason and Dr. Terry Feagin for their kind and helpful service on my thesis committee. I am grateful to Ms. Heather Drummond for critically reading the manuscript.

ABSTRACT

We consider a means of error-free computation that overcomes well-known difficulties in using fixed word-length digital computers for solving ill-conditioned problems and for using numerically unstable algorithms. The technique uses extended-precision single-modulus residue arithmetic to perform computations in an environment totally free of machine roundoff error.

We perform computations using the set of order-N Farey fractions

$$F_N = \{ a/b: 0 \leq a \leq N, 0 < |b| \leq N, \text{ and } \gcd(a,b) = 1 \} ,$$

which we map onto integer residues in the finite field

$$I_p = [0, 1, 2, \ldots , p-1] ,$$

where p is a prime such that

$$p \geq 2N^2 + 1 \quad .$$

We replace arithmetic operations on rational numbers in F_N by corresponding operations on integers in I_p. The mapping between F_N and I_p is one-to-one and onto, and consequently an integer result in I_p can be mapped onto its rational equivalent in F_N. If we choose N (and p) large enough, we can perform computations on rational numbers with very large numerators and denominators by performing the corresponding computations using integers in I_p and be guaranteed that the result in F_N is exact.

We discuss algorithms and coding of routines for extended-precision two's complement integer arithmetic and illustrate their use by means of specific examples. We then describe software routines to implement two equivalent forms of exact computation, (1) extended-precision single-modulus residue arithmetic using Mersenne

prime moduli and (2) rational arithmetic in which numerator and denominator are stored as separate integers. The routines allow us to retrieve results of as many as 1458 decimal numerator or denominator digits in a residue arithmetic computation (twice that number, 2918, in a rational arithmetic computation).

We apply extended-precision residue and rational arithmetic to four numerical algorithms: (a) Gaussian elimination--back substitution solution to simultaneous linear equations, (b) the Newton-Raphson method for obtaining roots of an equation, (c) the Milne-Simpson method for solving first order linear differential equations, and (d) Maclaurin series approximations. We present the exact results of specific computations and compare and contrast memory and execution time requirements using residue and rational arithmetic.

The major results of the study are twofold. First, exact computation produces enormous improvement in numerical accuracy over hardware integer and floating-point arithmetic, albeit vastly increasing computer memory and time requirements. Secondly, residue arithmetic computations necessitate significantly larger execution times (by factors of 2 to 12) and somewhat larger computer memory requirements than do equivalent rational arithmetic computations.

Further research needs include the development of additional software to facilitate exact arithmetic use, the development of hardware geared to exact computation, research on detection of pseudo-overflow in residue arithmetic, and the determination of the most efficient means of incorporating exact computation in specific iterative numerical algorithms.

TABLE OF CONTENTS

CHAPTER PAGE

I. INTRODUCTION. 1
A. Motivation for Error-Free Computation
B. Error-Free Computation Using Residue Arithmetic

II. PRINCIPLES OF SINGLE-MODULUS RESIDUE ARITHMETIC. 7
A. Integer Residue System
B. Residue Computations Involving Rational Operands
C. Inverse Mapping Using Symmetric Residues
1. How the Inverse Mapping is Performed
2. Difficulties with Implementing this Inverse Mapping
D. An Extended Euclidean Algorithm for the Inverse Mapping

III. MULTIPLE-PRECISION INTEGER ARITHMETIC. 32
A. Introduction
B. Assembler Macros
C. Integer Arithmetic Subroutines--Descriptions and Algorithms
D. Computational Results
1. Choquet series
2. Direct computation of the elements of the inverse Hilbert matrix

IV. MULTIPLE-PRECISION RESIDUE ARITHMETIC. 83
A. Choice of Prime Moduli
B. Descriptions of Residue Arithmetic Routines
1. Initialization and Utilization
2. Subroutine Descriptions

V. MULTIPLE-PRECISION RATIONAL ARITHMETIC. 96
A. Principles of Computation in the Field of Rational Numbers
1. Rational addition
2. Rational subtraction
3. Rational multiplication
4. Rational division
B. Coding and Use of Rational Arithmetic Subroutines
1. Initialization and utilization of routines
2. Description of rational arithmetic subroutines
C. Rational Number Output and Conversion Subroutines
1. Introduction
2. Description of subroutines

CHAPTER PAGE

VI. COMPUTATIONAL RESULTS--RATIONAL VS. RESIDUE ARITHMETIC. . . . 114
A. Introduction
B. Large Systems of Simultaneous Linear Algebraic Equations
1. Background--Hilbert matrices
2. Results--Hilbert coefficient matrices
C. Newton-Raphson Approximation to the Cube Root of a
D. Milne-Simpson Method of Approximate Solutions to First Order, Linear Differential Equations
E. Maclaurin Series Approximation to e^x

VII. CONCLUSIONS: SUGGESTIONS FOR ADDITIONAL RESEARCH. 235
A. Residue vs. Rational Arithmetic
1. Advantages of residue arithmetic
2. Disadvantages of residue arithmetic
B. Additional Software for Exact Computation
C. Hardware and Micro-programmed Features Useful for Exact Computation
D. New Avenues for Applying Numerical Approximation Techniques

LIST OF REFERENCES. 244

APPENDICES. 246
A. ASSEMBLER MACROS. 247
B. MULTIPLE-PRECISION INTEGER ARITHMETIC ROUTINES (ASSEMBLER). 254
C. CHOQUET SERIES PROGRAM (ASSEMBLER). 281
D. INVERSE HILBERT MATRIX PROGRAM (FORTRAN). 285
E. MULTIPLE-PRECISION RESIDUE ARITHMETIC ROUTINES (ASSEMBLER). 289
F. MULTIPLE-PRECISION RATIONAL ARITHMETIC ROUTINES (ASSEMBLER). 306
G. OUTPUT AND CONVERSION SUBROUTINES (ASSEMBLER). 317
H. GAUSSIAN ELIMINATION PROGRAMS FOR HILBERT MATRICES (PL1). 332
I. NEWTON-RAPHSON APPROXIMATION PROGRAMS FOR CUBE ROOT OF A (FORTRAN). 346
J. MILNE-SIMPSON PROGRAMS FOR SOLVING Y' = -Y, Y(0) = 1 (PL1). 352
K. MACLAURIN SERIES PROGRAMS FOR APPROXIMATING E^X (ASSEMBLER). 364

VITA. 374

LIST OF TABLES

TABLE		PAGE
1.	Residues Modulo 997 of Order-22 Farey Fractions a/b for which $0 < \|b\| \leq 11$	14
2.	Residues Modulo 997 of Order-22 Farey Fractions a/b for which $12 \leq \|b\| \leq 22$	15
3.	Choquet Series Computation: Multiple-Precision Integer Arithmetic	59
4.	Direct Computation of the Inverse Hilbert Matrix: Multiple-Precision Integer Arithmetic	62
5.	Mersenne Primes Used by Residue Arithmetic Subroutines	85
6.	Rational/Residue Arithmetic Execution Data: Linear Systems AX=B in which A is a Hilbert Matrix	117
7.	Digits in p(n) of the Exact Newton-Raphson Computation of $a^{1/3}$	131
8.	Execution Data for Residue and Rational Arithmetic: Newton-Raphson Approximations to the Cube Root of a	131
9.	Milne-Simpson Method: Initial-Value Problem $y' = -y$, $y(0) = 1$	138
10.	Maclaurin Series Computation of e^x	213

CHAPTER I

INTRODUCTION

Motivation for Error-Free Computation

Because of their accuracy, large data storage capacity, and extremely high speed, computers are extraordinarily well adapted to performing numerous complex mathematical computations that are either impossible or extremely difficult and time-consuming as hand calculations. Nonetheless, the limitations of fixed-word-length computers in storing numeric values is well-known. Only a finite subset of the field R of real numbers can be represented in fixed-word-length machines. The subset F of floating-point numbers that are computer-representable depends on machine word length, the internal representation chosen for storing floating-point numbers, and the radix (usually radix-two) employed by the machine.

The limitations on the subset F pose a number of problems in performing numerical computations using digital computers:

1. Real operands for use in computations must be stored (represented) as a nearby (hopefully, the nearest) floating-point value which is a member of F. Unfortunately, most common rational numbers cannot be represented exactly as computer-representable numbers. For instance, in a binary machine, only certain fractions whose denominators are powers of 2 can be stored exactly. Thus 5/8 can be stored precisely, but 1/3 cannot.
2. The system F does not constitute a field. Thus, in general, the result of any of the four basic arithmetic operations does not lie

in F, and consequently the result of a calculation must be mapped onto a nearby number of F. Hence, at each stage of an extended computation, roundoff error is introduced. For some numerical algorithms, roundoff error may grow to the degree that it dominates the final solution.

3. The range of computer-representable floating-point values is restricted by the number of binary digits employed to store the exponent. The number of significant digits is governed by the number of bits used for storing the mantissa. Computations involving larger operands or calculations for which a large number of significant digits are desired cannot be handled directly. On the IBM 360/370, for example, single-precision floating-point operands of 32 bits are stored with 7 bits allocated for the exponent, 24 bits for the normalized mantissa, and one bit for the sign. Thus the range of numbers that can be stored is about 2^{-64} to 2^{63}, each with only roughly 7 decimal digit accuracy.

The introduction of roundoff error in computations can lead to disaster if roundoff error accumulates until it dominates the computed result. The effects of computer roundoff error lead us to choose computational algorithms not only on the basis of their correctness and machine computational speed, but also (or sometimes chiefly) upon the basis of their insensitivity to the consequences of roundoff error.

1.1 Definition. If an algorithm for the computer solution to a numerical problem has the property that the computed solution is "very close" to the exact solution of a slightly perturbed problem,

we call the algorithm numerically stable. If the algorithm does not possess this property, we term it numerically unstable.

A number of familiar computational algorithms fail the test of numerical stability. For instance, application of the quadratic formulas

(1.2) $$x_1 = (-b + [b^2 - 4ac]^{\frac{1}{2}}) / 2a$$

and

(1.3) $$x_2 = (-b - [b^2 - 4ac]^{\frac{1}{2}}) / 2a$$

toward finding roots of the quadratic equation

(1.4) $$ax^2 + bx + c = 0$$

can produce highly inaccurate results when machine floating-point arithmetic is employed. Gregory [6] and Forsythe[4] describe the problem, which arises when $4ac$ is small relative to b^2, thus producing severe cancellation due to subtraction of nearly equal numbers. For certain choices of coefficients a, b, and c, a severe loss of significant digits may be sustained. An alternate computation using the relationship

(1.5) $$x_1 x_2 = c/a$$

to compute one of the roots can lead to significant improvement in the results.

1.6 Example. Compute roots of quadratic equation (1.4) if $a = 5 \times 10^{-8}$, $b = -100$, and $c = 2$.

If we perform the computation on the IBM 370 using single-precision

(32-bit) machine floating-point arithmetic, a is stored inexactly as approximately $5.00000006 \times 10^{-8}$. The output (FORTRAN H results) gives from (1.2) the computed value

$$x_1 = 0.200000051 \times 10^{10}$$

If the second root is computed from (1.3), we get

$$x_2 = 0.0 \quad ,$$

whereas from (1.5) we calculate

$$x_2 = 0.0199999950 \quad .$$

The root x_2 from the customary relationship vanishes due to the severe cancellation. The second result for x_2 is acceptably precise for many purposes.

Almost all numeric values which stem from physical measurement are rational numbers (as opposed to integers) and thus cannot generally be stored exactly as binary-floating-point numbers. Moreover, an error in measurement is usually associated with each. It is therefore of great interest to know to what degree the computed solution to a mathematical or physical problem is affected by perturbations (errors) in the original data.

1.7 Definition. If the solution to a numerical problem is extremely sensitive to small perturbations in the data describing the problem, the problem is called ill-conditioned. Otherwise, the problem is termed well-conditioned.

Rounding errors introduced during computation can be shown to be equivalent to introducing perturbations in the original data. Hence, ill-conditioned problems, in particular, cannot tolerate the effects of

roundoff error. For certain categories of problems, we consequently have great motivation to investigate performing the computation by using exact arithmetic.

Error-Free Computation Using Residue Arithmetic

Our purpose in this thesis is to examine in an exploratory manner numerical computations in an environment totally free of machine roundoff error. We shall pursue exact computation in a finite field of integer residues with respect to a large prime modulus. Our task is fourfold:

1. To develop the necessary software support routines to perform single-modulus residue arithmetic with very large operands.
2. To apply extended-precision residue arithmetic to several common numerical problems in which exact computation is useful.
3. To evaluate extended-precision residue arithmetic in relation to extended-precision rational arithmetic.
4. To suggest goals for additional study in this area.

In Chapter 2 we outline the general principles of single-modulus residue arithmetic. Arithmetic computations are illustrated, and problems associated with residue arithmetic are discussed.

In Chapter 3 we describe assembler language subroutines and algorithms employed to implement extended-precision integer arithmetic. These procedures are then applied to two integer arithmetic computational problems requiring extended-precision.

In Chapter 4 we detail the coding of assembler language support

routines to implement multiple-precision residue arithmetic. The selection and use of large prime moduli is discussed.

In Chapter 5 we treat algorithms and coding for implementing extended-precision rational arithmetic. Binary-to-character conversions and the printing of very large rational operands are also discussed.

In Chapter 6 we present the computational results for four numerical computations in which exact results are useful. Mathematically identical results from parallel multiple-precision residue and rational arithmetic main programs are described, and execution data for the two computational approaches are compared.

Chapter 7 summarizes findings from the current study and suggests directions for additional work.

CHAPTER II

PRINCIPLES OF SINGLE-MODULUS RESIDUE ARITHMETIC

In this chapter we discuss the principles of single-modulus residue arithmetic and illustrate computational techniques by means of numerous examples. The proofs of theorems are omitted here, as they are readily available in the literature (see especially [6], [11], [13], and [14]). The actual implementation and computer coding of residue arithmetic in a multiple-precision environment will be discussed in Chapter 4.

Integer Residue System

Let I denote the set of integers. Let $a, r, p \in I$, where $p > 1$. If p divides a-r evenly, we write

$$a \equiv r \pmod{p} \tag{2.1}$$

and use the terminology "a is congruent to r modulo p." If p does not divide a-r evenly, "a is not congruent to p." Each integer $a \in I$ can be represented by a second integer r, termed its "residue" modulo p, where r satisfies (2.1) and in addition

$$0 \leq r < p \quad . \tag{2.2}$$

Notationally we write

$$r = |a|_p \quad . \tag{2.3}$$

This least non-negative residue we term the standard residue of a modulo p.

2.4 Theorem. Let a, b, c, p $\in$ I, where p > 1. Then

(i) $|a|_p$ is unique.

(ii) $|a|_p = |b|_p \iff a \equiv b \pmod p$.

(iii) $|kp|_p = 0$ for all $k \in I$.

(iv) $|a+b|_p = ||a|_p+|b|_p|_p$

$= ||a|_p+b|_p$

$= |a+|b|_p|_p$

(v) $|ab|_p = ||a|_p|b|_p|_p$

$= ||a|_p\, b|_p$

$= |a\, |b|_p|_p$.

Note that (iv) and (v) describe addition and multiplication modulo p.

2.5 Example. $|827 + 3825|_{997} = |4652|_{997}$

$= 664$.

2.6 Example. $|1048 \cdot 5211|_{997} = |5462088|_{997}$

$= 649$.

Consider the finite set I_p of standard residues modulo p

(2.7) $$I_p = \{0,1,2,\ldots,p-1\},$$

and let $+$ and $\cdot$ denote respectively addition and multiplication modulo p. What we have considered thus far is the mapping $|\cdot|_p : I \longrightarrow I_p$.

2.8 Theorem. The system $(I_p,+,\cdot)$ constitutes a finite commutative ring.

We define subtraction in the ring $(I_p,+,\cdot)$ as the addition of the additive inverse modulo p:

$$(2.9) \qquad |a - b|_p = |a + \underline{b}|_p = |a + |-b|_p|_p \ ,$$

where the additive inverse $\underline{b}$ is given by

$$(2.10) \qquad \underline{b} = |-b|_p \ .$$

2.11 Example. $|418 - 902|_{997} = |418 + |-902|_{997}|_{997}$
$= |418 + 95|_{997}$
$= 513 \ .$

2.12 Theorem. The finite commutative ring $(I_p,+,\cdot)$ is a finite field if and only if p is a prime.

Thus, if p is prime, $(I_p,+,\cdot)$ is isomorphic to the Galois field GF(p), and each non-zero member a of I_p has a multiplicative inverse modulo p, call it b, such that

$$(2.13) \qquad |ab|_p = |ba|_p = 1 \ .$$

We write [1]

$$(2.14) \qquad b = a^{-1} \ (p) \ .$$

2.15 Theorem. If $a \in I$ such that $|a|_p \neq 0$, then $a^{-1} \ (p) \in I_p$ exists (and is unique) if and only if a and p are relatively prime, i. e., gcd(a,p) = 1.

---

[1] When the modulus is clearly understood, we merely write a^{-1} instead of $a^{-1} \ (p)$.

Hence, if p is a prime, the system $(I_p,+,\cdot)$ is always a finite field, and the multiplicative inverse a^{-1} exists for each $a \in I_p$. We thus have strong motivation for restricting ourselves to prime moduli p, for which we can define division modulo p as

$$|a/b|_p = |a \cdot b^{-1}|_p \quad . \tag{2.16}$$

2.17 Example.
$$\begin{aligned} |126/473|_{997} &= |126 \cdot 473^{-1}|_{997} \\ &= |126 \cdot 215|_{997} \\ &= |55990|_{997} \\ &= 605 \quad . \end{aligned}$$

Residue Computations Involving Rational Operands

Thus far we have examined the system $(I_p,+,\cdot)$ in terms of residue operations involving integer operands. More importantly, we can also perform residue operations on rational operands. We shall find that we can map rational operands onto integer residues, carry out computations in the system $(I_p,+,\cdot)$, and, if the prime p is chosen large enough, map the residue result in I_p onto a rational equivalent in the field Q of rational numbers using one of two techniques. Most importantly, the computation produces an exact result (given sufficiently large p).

The definition of integer division modulo p of the previous section provides the basis for this process. Using (2.16), we map rational operands into I_p and perform arithmetic operations on their integer residues. Thus, for $x = a/b \in Q$, we define the mapping $|\cdot|_p : Q \dashrightarrow I_p$ as follows:

$$\begin{aligned} |x|_p &= |a/b|_p \\ &= |a \cdot b^{-1}|_p \quad . \end{aligned} \tag{2.18}$$

$|x|_p$ is the integer representing the rational number x. Two important points must be made:

1. b^{-1} (p) exists only if $|b|_p \neq 0$. Hence there is an infinite subset $\overline{Q}$ of rational numbers for which there is no image in I_p.

2. For any choice of p (at this point, we assume p is a prime number), the number of elements of I_p is finite. An infinite set of rational numbers $(Q - \overline{Q})$ map onto I_p. Hence, we have p (disjoint) generalized residue classes

$$(2.19) \qquad Q_i = \{x \in Q-\overline{Q} : |x|_p = i,\ i = 0,1,2,\ldots,p-1\}.$$

Consequently, in the strict sense, our mapping is $|\cdot|_p : \hat{Q} \dashrightarrow I_p$, where $\hat{Q} = Q-\overline{Q}$.

The following theorem provides the basis for computations in the system I_p.

2.20 Theorem. The mapping $|\cdot|_p : \hat{Q} \dashrightarrow I_p$ is a homomorphism with respect to addition and multiplication.

Thus, we can carry out computations in $(\hat{Q},\div,\cdot)$ by performing corresponding arithmetic in $(I_p,\div,\cdot)$. How are we to retrieve results after computations? The relationship between the systems $(\hat{Q},\div,\cdot)$ and $(I_p,\div,\cdot)$ is not an isomorphism. The mapping from $\hat{Q}$ to I_p is not one-to-one and onto. Hence, the inverse mapping does not exist. Consider the following example.

2.21 Example.
$$\begin{aligned} |34/5|_{997} &= |34 \cdot 5^{-1}|_{997} \\ &= |34 \cdot 399|_{997} \\ &= |13,566|_{997} \\ &= 605 \ . \end{aligned}$$

Thus, $34/5 \in \hat{Q}$ maps onto $605 \in I_{997}$. Earlier (Example 2.17), we saw that $26/473 \in \hat{Q}$ also maps onto $605 \in I_{997}$. In fact, there is an infinite set of rational numbers which map onto $605 \in I_{997}$. $605/1$ is another example. If we perform a residue computation in I_{997} with the residue result 605, which rational number $\in \hat{Q}$ does this result represent? We will now delineate circumstances under which we can answer this question.

As we have noted, each integer i in I_p is the image of an infinite set of rational numbers Q_i. We would like to select a single member of each residue class Q_i to represent that residue class and thus make the mapping one-to-one and onto. The following theorem provides the basis for selecting a single element to represent <u>some</u> of the Q_i.

2.22 <u>Theorem</u>. Let N be the largest integer satisfying the inequality

$$N \leq [(p-1)/2]^{\frac{1}{2}}$$

and let $a/b \in \hat{Q}$. Then if $|a| \leq N$ and $0 < b \leq N$, we can represent a/b uniquely by an integer in I_p.

2.23 <u>Definition</u>. The finite subset of $\hat{Q}$

$$F_N = \{x = a/b : |a| \leq N,\ 0 < b \leq N,\ \gcd(a,b) = 1\}$$

is called the set of order-N Farey fractions.

2.24 <u>Corollary</u>. Let $\hat{I}_p$ denote the set of images of the order-N Farey fractions under the mapping $|\cdot|_p : F_N \rightarrow \hat{I}_p$. Then $|\cdot|_p$ is one-to-one and onto and thus has an inverse.

2.25 <u>Example</u>. Let p = 997. Then

$$N \leq [(997-1)/2]^{\frac{1}{2}}$$
$$= (498)^{\frac{1}{2}} \quad .$$

Hence, N = 22.

The mapping $|\cdot|_{997} : F_{22} \dashrightarrow \hat{I}_{997}$ is shown in Tables 1 and 2. Close scrutiny of these tables shows us that there are 599 distinct (reduced) fractions $\in F_{22}$. The residues corresponding to these Farey fractions constitute the set $\hat{I}_{997}$. Clearly $\hat{I}_{997}$ is a subset of I_{997}, for 398 distinct residues in I_{997} have no inverse element in F_{22}.

In general, we formulate the following strategy for performing exact computations using residue arithmetic.

1. Choose a very large prime p, ensuring that the set of order-N Farey fractions is also quite large.
2. Map rational operands (all Farey fractions) from F_N into I_p.
3. Carry out the necessary arithmetic computations using integer residues in the finite field $(I_p, +, \cdot)$.
4. Map the results back onto rational equivalents in F_N.

2.26 Example. Let p = 997 so that N = 22. Perform the computation

$$\begin{aligned} x &= (15/22) \;/\; (-6/11) \\ &= (15/22)\,(-11/6) \\ &= -5/4 \end{aligned}$$

by using residue arithmetic. We can use the list of residues in Tables 1 and 2 to perform the computation

$$\begin{aligned} |x|_{997} &= |(15/22) \;/\; (-6/11)|_{997} \\ &= |146 \;/\; 362|_{997} \\ &= |146 \cdot 362^{-1}|_{997} \\ &= |(46)\,(829)|_{997} \\ &= |38{,}134|_{997} \\ &= 249. \end{aligned}$$

TABLE 1

RESIDUES MODULO 997 OF ORDER-22 FAREY FRACTIONS a/b
FOR WHICH $0 < |b| \leq 11$

a \ b	1	2	3	4	5	6	7	8	9	10	11
22	22	---	672	---	802	---	288	---	224	---	---
21	21	509	---	753	403	---	---	875	---	700	727
20	20	---	339	---	---	---	715	---	113	---	455
19	19	508	671	254	602	834	430	127	556	301	183
18	18	---	---	---	203	---	145	---	---	---	908
17	17	507	338	752	801	169	857	376	445	899	636
16	16	---	670	---	402	---	572	---	888	---	364
15	15	506	---	253	---	---	287	625	---	---	92
14	14	---	337	---	601	---	---	---	777	---	817
13	13	505	669	751	202	833	714	874	223	101	545
12	12	---	---	---	800	---	429	---	---	---	273
11	11	504	336	252	401	168	144	126	112	699	---
10	10	---	668	---	---	---	856	---	555	---	726
9	9	503	---	750	600	---	571	375	---	300	454
8	8	---	335	---	201	---	286	---	444	---	182
7	7	502	667	251	799	832	---	624	887	898	907
6	6	---	---	---	400	---	713	---	---	---	635
5	5	501	334	749	---	167	428	873	776	---	363
4	4	---	666	---	599	---	143	---	222	---	91
3	3	500	---	250	200	---	855	125	---	100	816
2	2	---	333	---	798	---	570	---	111	---	544
1	1	499	665	748	399	831	285	374	554	698	272
0	0	---	---	---	---	---	---	---	---	---	---
-1	996	498	332	249	598	166	712	623	443	299	725
-2	995	---	664	---	199	---	427	---	886	---	453
-3	994	497	---	747	797	---	142	872	---	897	181
-4	993	---	331	398	---	854	---	775	---	906	---
-5	992	496	663	248	---	830	569	124	221	---	634
-6	991	---	---	---	597	---	284	---	---	---	362
-7	990	495	330	746	198	165	---	373	110	99	90
-8	989	---	662	---	796	---	711	---	553	---	815
-9	988	494	---	247	397	---	426	622	---	697	543
-10	987	---	329	---	---	---	141	---	442	---	271
-11	986	493	661	745	596	829	853	871	885	298	---
-12	985	---	---	---	197	---	568	---	---	---	724
-13	984	492	328	246	795	164	283	123	774	896	452
-14	983	---	660	---	396	---	---	---	220	---	180
-15	982	491	---	744	---	---	710	372	---	---	905
-16	981	---	327	---	595	---	425	---	109	---	633
-17	980	490	659	245	196	828	140	621	552	98	361
-18	979	---	---	---	794	---	852	---	---	---	89
-19	978	489	326	743	395	183	567	870	441	696	814
-20	977	---	658	---	---	---	282	---	884	---	542
-21	976	488	---	244	594	---	---	122	---	297	270
-22	975	---	325	---	195	---	709	---	773	---	---
a \ b	1	2	3	4	5	6	7	8	9	10	11

TABLE 2

RESIDUES MODULO 997 OF ORDER-22 FAREY FRACTIONS a/b
FOR WHICH $12 \leq |b| \leq 22$

a \ b	12	13	14	15	16	17	18	19	20	21	22
22	---	922	---	932	---	981	---	316	---	96	---
21	---	155	---	---	936	705	---	211	350	---	862
20	---	385	---	---	---	529	---	106	---	903	---
19	417	615	215	533	562	353	278	---	649	808	590
18	---	845	---	---	---	177	---	893	---	---	---
17	583	78	927	267	188	---	721	788	948	618	318
16	---	308	---	134	---	822	---	683	---	523	---
15	---	538	642	---	811	646	---	578	---	---	46
14	---	768	---	865	---	470	---	473	---	---	---
13	915	---	357	732	437	294	610	368	549	238	771
12	---	231	---	---	---	118	---	263	---	---	---
11	84	461	72	466	63	939	56	158	848	48	---
10	---	691	---	---	---	763	---	53	---	950	---
9	---	921	784	---	686	587	---	945	150	---	227
8	---	154	---	67	---	411	---	840	---	760	---
7	416	384	---	931	312	235	942	735	449	---	952
6	---	614	---	---	---	59	---	630	---	---	---
5	582	844	214	---	935	880	388	525	---	475	680
4	---	77	---	532	---	704	---	420	---	380	---
3	---	307	926	---	561	528	---	315	50	---	408
2	---	537	---	266	---	352	---	210	---	190	---
1	914	767	641	133	187	176	277	105	349	95	136
0	---	---	---	---	---	---	---	---	---	---	---
-1	83	230	356	864	810	821	720	892	648	902	861
-2	---	460	---	731	---	645	---	787	---	807	---
-3	---	690	71	---	436	469	---	682	947	---	589
-4	---	920	---	465	---	293	---	577	---	617	---
-5	415	153	783	---	62	117	609	472	---	522	317
-6	---	383	---	---	---	938	---	367	---	---	---
-7	581	613	---	66	685	762	55	262	548	---	45
-8	---	843	---	930	---	586	---	157	---	237	---
-9	---	76	213	---	311	410	---	52	847	---	770
-10	---	306	---	---	---	234	---	944	---	47	---
-11	913	536	925	531	934	58	941	839	149	949	---
-12	---	766	---	---	---	879	---	734	---	---	---
-13	82	---	640	265	560	703	387	629	448	759	226
-14	---	229	---	132	---	527	---	524	---	---	---
-15	---	459	355	---	186	351	---	419	---	---	951
-16	---	689	---	863	---	175	---	314	---	474	---
-17	414	919	70	730	809	---	276	209	49	379	679
-18	---	152	---	---	---	820	---	104	---	---	---
-19	580	382	782	464	435	644	719	---	348	189	407
-20	---	612	---	---	---	468	---	891	---	94	---
-21	---	842	---	---	61	292	---	786	647	---	135
-22	---	75	---	65	---	116	---	681	---	901	---
a \ b	12	13	14	15	16	17	18	19	20	21	22

Since $|x|_{997} = 248 \in \hat{I}_{997}$, we perform the inverse mapping $248 \in \hat{I}_{997} \dashrightarrow F_{22}$ to obtain the result $-5/4$.

There are two situations in which the inverse mapping may fail. We shall refer to both cases as "pseudo-overflow."[2] Both result from our using a value of p insufficiently large to guarantee that the final rational result of a computation is a member of F_N. The first situation occurs when the residue result is not a member of $\hat{I}_p$ and hence the inverse mapping fails.

2.27 Example. Let p = 997. Hence N = 22. Compute $3/2 \div 14/17$ using residue arithmetic.

$$\begin{aligned} |x|_{997} &= |3/2 \div 14/17|_{997} \\ &= |500 \div 470|_{997} \\ &= 970 \quad . \end{aligned}$$

970 is not an element of $\hat{I}_{997}$ and hence has no image in F_{22}. Note, however, that the true rational result $3/2 \div 14/17 = 79/34$ reduced modulo 997 gives us 970.

$$\begin{aligned} |79/34|_{997} &= |79 \cdot 34^{-1}|_{997} \\ &= |(79)\ (88)|_{997} \\ &= |6952|_{997} \\ &= 970 \quad . \end{aligned}$$

Observe that both the numerator 79 and the denominator 34 exceed N = 22.

---

[2] The term pseudo-overflow was first suggested by Dr. T. N. Rao. This term is used since the effect on a computation is not necessarily fatal.

The second situation arises when the true result is not a member of F_N, but the residue result is a member of $\hat{I}_p$. We thus incorrectly map the result to an order-N Farey fraction.

2.28 Example. Let p = 997, N =22, and compute (14/5) (4/5) by means of residue arithmetic.

$$\begin{aligned} |x|_{997} &= |(14/5)\ (4/5)|_{997} \\ &= |(601)\ (599)|_{997} \\ &= |359,999|_{997} \\ &= 82\ . \end{aligned}$$

When we map $82 \in \hat{I}_{997}$ to F_{22}, we get the erroneous result -13/12. Notice that the correct result ((14/5)(4/5) = 56/25) reduced modulo 997 is also an element of Q_{82}:

$$\begin{aligned} |56/25|_{997} &= |56 \cdot 25^{-1}|_{997} \\ &= |(56)\ (678)|_{997} \\ &= |37,968|_{997} \\ &= 82\ . \end{aligned}$$

It turns out that pseudo-overflow at an intermediate step of a computation is of no consequence if the final result is a member of F_N.

2.29 Example. Compute (3/2 + 14/17) (2/79) . Let p = 997, so that N = 22.

$$\begin{aligned} |x|_{997} &= |(3/2 + 14/17)\ (2/79)|_{997} \\ &= |(500 + 470)\ ((2)\ (366))|_{997} \\ &= |(970)\ (732)|_{997} \\ &= |710,040|_{997} \\ &= 176 \end{aligned}$$

We map $176 \in \hat{I}_{997}$ onto the correct result 1/17.

Recall that, in Example 2.27, 3/2 + 14/17 gave us pseudo-overflow because the residue result 970 was not a member of $\hat{I}_{997}$. Note also that the operand 2/79 is not a member of F_{22}. The original operands and intermediate results need not be order-N Farey fractions as long as the final result that we wish to retrieve is an order-N Farey fraction.

When p (and N) are large enough to be generally useful, use of a table of Farey fractions and residues is impractical. There are two techniques for determining directly the inverse mapping $\hat{I}_p \dashrightarrow F_N$. The first can be employed under special circumstances. The second is always successful (if the residue result belongs to $\hat{I}_p$).

Inverse Mapping Using Symmetric Residues

How the Inverse Mapping is Performed

In order to describe the first technique, we must first introduce what are termed *symmetric residues*. Consider the set of integers

$$S_p = \{-(p-1)/2,\ldots,-2,-1,0,1,2,\ldots,+(p-1)/2\}\ , \tag{2.30}$$

where p is an odd integer > 2, usually a prime. The set S_p is called the set of symmetric residues modulo p. Analagous to the mapping $|\cdot|_p : I \dashrightarrow I_p$, we can define the one-to-one and onto mapping $/\cdot/_p : I \dashrightarrow S_p$ from any integer a to its symmetric residue:

$$/a/_p = \begin{cases} |a|_p\ , & \text{if } 0 \le |a|_p \le (p-1)/2 \\ |a|_p - p\ , & \text{otherwise.} \end{cases} \tag{2.31}$$

Hence, $/a/_p$ lies between -(p-1)/2 and +(p-1)/2.

The following theorem provides one means for performing the inverse mapping from $\hat{I}_p$ to F_N.

2.32 Theorem. Let $x = a/b$, where b does not evenly divide a, and

$$|a| \leq N, \quad 0 < |b| \leq N,$$

where $|x|_p$ is given and N is the largest integer satisfying

$$N \leq [(p-1)/2]^{\frac{1}{2}} .$$

Then if an integer multiple k of b can be found, $0 < k \leq N$,

$$ka = /kb|x|_p/_p .$$

From this,

$$x = ka/kb .$$

2.33 Example. Let $p = 997 \Longrightarrow N = 22$. Compute $(4/5)\ (15/8)$.

$$\begin{aligned} |(4/5)\ (15/8)|_{997} &= |(599)\ (625)|_{997} \\ &= |374{,}375|_{997} \\ &= 500 . \end{aligned}$$

A common denominator for this computation is 40. Then for the inverse mapping we have

$$\begin{aligned} ka &= /(40)\ (500)/_{997} \\ &= 60 . \end{aligned}$$

Thus,

$$\begin{aligned} x &= ka/kb \\ &= 60/40 \\ &= 3/2 , \end{aligned}$$

which is the correct result. Note that in this instance $N = 20$, which satisfies the condition $0 < k \leq N$.

Difficulties with Implementing this Inverse Mapping

The majority of computational processes to which we might wish to apply residue arithmetic do not provide us with a multiple k of the

denominator for use in Theorem 2.32. Hence, if we wish to develop a set of residue arithmetic software routines for general applicability utilizing this theorem, we are forced to maintain a running common denominator with each residue operand. For the addition or subtraction $A/B \pm C/D$, the least common denominator of the result is given by $B'D$, where $B' = D/\gcd(B,D)$. For the multiplication $(A/B)(C/D)$, the least common denominator is given by the least common multiple of the common denominators of the two operands.

2.34 Example. Perform the computation $(2/9 - 4/3)\ (6/7)$ modulo 997.

From ordinary rational arithmetic we get

$$\begin{aligned}(2/9 - 4/3)\ (6/7) &= ((2 - 12)/9)\ (6/7)\\ &= -60/63\\ &= -20/21\ .\end{aligned}$$

Consequently the result $-20/21$ is a member of F_{22}. The residue calculation is as follows:

$$\begin{aligned}\left|(2/9 - 4/3)\ (6/7)\right|_{997} &= \left|111 \div 331\right|_{997} \cdot \left|713\right|_{997}\\ &= \left|(442)\ (713)\right|_{997}\\ &= \left|315,146\right|_{997}\\ &= 94\ .\end{aligned}$$

To maintain a running common denominator, we perform a parallel computation:

$$\begin{aligned}kb &= \operatorname{lcm}[(9/\gcd(9,3))(3)),7]\\ &= \operatorname{lcm}[9,7]\\ &= 63\ .\end{aligned}$$

Then, to map the residue result 94 onto the rational equivalent x, we have

$$ka = /kb \cdot |x|_p/_p$$
$$= /(63)\ (94)/_{997}$$
$$= /5922/_{997}$$
$$= -60 \quad .$$

Hence,

$$x = ka/kb$$
$$= -60/63$$
$$= -20/21 \quad ,$$

which is the correct answer. In this instance, $k = 3$.

The above example illustrated multiplication within a residue computation. However, suppose we must divide at an intermediate step of the computation. In ordinary rational arithmetic we merely invert the fraction representing the divisor and multiply. When we perform residue arithmetic, we multiply by the multiplicative inverse of the residue of the divisor. If we maintain a running common denominator, we also need to know the numerator of the divisor fraction. Hence we must map the residue representing the divisor back to an order-N Farey fraction and extract the numerator, which will become the denominator of the inverted fraction.

2.35 Example. We have two intermediate residues modulo 997, 331 and 713, the respective residue equivalents modulo 997 of Farey fractions $-4/3$ and $6/7$. The rational computation is given by

$$(-4/3) \ / \ (6/7) = (-4/3)\ (7/6)$$
$$= -28/18$$
$$= -14/9 \quad .$$

To perform the residue division $|331/713|_{997}$, we proceed as

follows:

$$\begin{aligned} |313/713|_{997} &= |(331)\ (713^{-1})|_{997} \\ &= |331 \cdot 832|_{997} \\ &= |275,392|_{997} \\ &= 220. \end{aligned}$$

To maintain a common denominator of the result, we first must map residue 713 to a rational equivalent. If we assume a running common denominator of 7, we get

$$\begin{aligned} ka &= /(7)\ (713)/_{997} \\ &= /4991/_{997} \\ &= 6 \\ a/b &= ka/kb \\ &= 6/7 \end{aligned}$$

Hence a common denominator associated with 713^{-1} (997) is 6. If the common denominator associated with the residue 313 is 3, we now compute the common denominator of the result as follows:

$$\begin{aligned} ka &= lcm(3,6) \\ &= 18 \ . \end{aligned}$$

If at this point, we wish the rational equivalent of the result, we may perform the inverse mapping:

$$\begin{aligned} ka &= /(18)\ (220)/_{997} \\ &= /3960/_{997} \\ &= -28 \ . \end{aligned}$$

Hence,

$$\begin{aligned} x &= ka/kb \\ &= -28/18 \\ &= -14/9 \ , \end{aligned}$$

which is the correct result.

The foregoing examples illustrate clearly difficulties associated with maintaining a running common denominator in parallel with a residue arithmetic computation. Since one must maintain a common denominator with each operand, one has all of the disadvantages of rational arithmetic with none of the advantages of residue arithmetic. Computer memory must be allocated for operand residues and for associated running common denominators. To handle an equivalent magnitude of final results (great enough N), approximately twice the memory is required for residue operand storage as for rational storage. The residue computation is more complex since multiplicative inverses must frequently be computed, and residues of simple fractions, used in residue calculations, may be quite large numbers. For instance, $|1/2|_{997} = 449$, which is the order of magnitude of the modulus.

Furthermore, one of the primary advantages of residue arithmetic has been lost. As we have seen, overflow in intermediate results is unimportant so long as the final result is an order-N Farey fraction. If in performing division we must map an intermediate result to an equivalent Farey fraction, we have the possibility that the intermediate result is not an order-N Farey fraction. The computation thus fails at this point. In maintaining running common denominators, we are computing integer values--not residues. We thus face the added possibility of overflowing the computer memory allotted for the storage of operands. Hence, whenever a running denominator approaches in size the square root of available storage (roughly N if the modulus closely fits operand size), we need to map the residue onto its rational

equivalent and reduce the fraction if possible by eliminating common factors. Here again we face the failure of the computation if the intermediate result is not a member of F_N.

Clearly a better scheme for performing the inverse mapping $\hat{I}_p$ --> F_N is desirable.

An Extended Euclidean Algorithm for the Inverse Mapping

Kornerup and Gregory [11] present a much improved algorithm for the inverse mapping $\hat{I}_p$ --> F_N that overcomes the difficulties described in the previous section. Their algorithm is based on the Euclidean Algorithm for finding the greatest common divisor of two positive integers.

2.36 An Extended Euclidean Algorithm.

Let (a,b) and (c,d) be two integer pairs such that a and c are positive, and $a \geq c$. The matrix

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

will be referred to as the "seed matrix." From the two original integer pairs in the seed matrix ((a,b) and (c,d)), we may generate a finite sequence of additional integer pairs. At each point in the computation, we use the previous two integer pairs generated in the sequence, denoted respectively (U_2,U_1) and (V_2,V_1), to determine the new integer pair, denoted by (T_2,T_1).

Step 1. Initialize $(U_2,U_1) \leftarrow (a,b)$

$$(V_2,V_1) \leftarrow (c,d) .$$

Step 2. Compute the integer quotient

$$Q = U_2 \div V_2 .$$

Step 3. Compute the new integer pair (T_2,T_1) in the sequence by

$$T_2 = U_2 - QV_2$$

$$T_1 = U_1 - QV_1 \quad .$$

Thus, T_2 is the non-negative remainder from the division $U_2 \div V_2$.

Step 4. Update (U_2,U_1) and (V_2,V_1) to contain the last two integer pairs in the current sequence:

$$(U_2,U_1) \leftarrow (V_2,V_1)$$

$$(V_2,V_1) \leftarrow (T_2,T_1)$$

Step 5. If $V_2 \neq 0$, go to Step 2.

Step 6. Record $\gcd(a,c) = U_2$.

Note that at any iteration of the algorithm,

$$\begin{bmatrix} U_2 & U_1 \\ V_2 & V_1 \end{bmatrix}$$

may be considered as the "seed" for computing the next integer pair (T_2,T_1).

2.37 Example. Compute the greatest common divisor of (2805,105).

We may conveniently use b = 0, d = 1 in the seed matrix. We generate the following sequence of integer pairs (Values of (U_2,U_1), (V_2,V_1), and (T_2,T_1) at an intermediate stage of the computation are indicated):

Q				
---	2805	0		
---	105	1	U_2	U_1
26	75	-26	V_2	V_1
1	30	27	T_2	T_1
2	15	-80		
2	0	107		

Hence, gcd(2805,105) =15 .

The following theorem forms the basis for the use of an Extended Euclidean Algorithm in the inverse mapping $\hat{I}_p \dashrightarrow F_N$.

2.38 Theorem. If r/s is an order-N Farey fraction such that

$$|r/s|_p = |k|_p = k \quad ,$$

where $k \in I_p$, and

$$0 < r \leq N$$

$$0 < |s| \leq N \ ,$$

then there exists an i such that

$$(r,s) = (a_i, b_i) \ ,$$

where $\{(a_j,b_j)\}$, $j = 0,1,\ldots,n$ is the sequence of integer pairs generated by an Extended Euclidean Algorithm seeded with the matrix

$$\begin{bmatrix} p & 0 \\ k & 1 \end{bmatrix} .$$

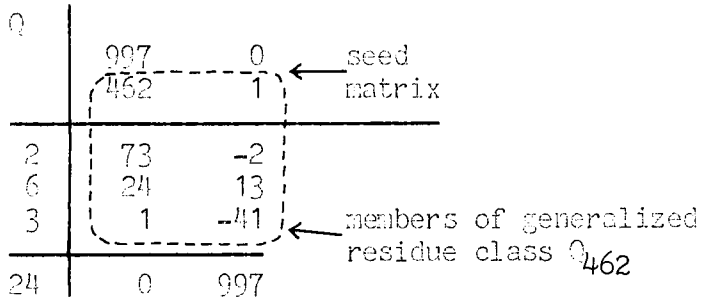
Note that if residue k = 0, we map to the integer 0 represented by r/s = 0/1.

2.39 Example. Attempt to map $696 \in \hat{I}_{997}$ to F_{22}.

0	997	0	seed matrix
	696	1	
1	301	-1	
2	94	3	
3	19	-10	members of generalized residue class 0_{696}
4	18	43	
1	1	-53	
18	0	997	

Hence, $|696|_{997}$ ---> -19/10. In practice, since the order-22 Farey fraction among the elements of $\mathcal{Q}_{696}$ is unique, we can stop the algorithm as soon as an order-22 Farey fraction (-19/10) is encountered.

2.40 Example. Attempt to map $462 \in I_{997}$ --> F_{22}.



Hence, $|462|_{997}$ has no image in F_{22}.

An additional use of an Extended Euclidean Algorithm is to compute directly the residue of rational number r/s without the necessity of computing the multiplicative inverse of s. The following theorem provides the basis for this use.

2.41 Theorem. Given any rational number r/s and an integer p such that gcd(s,p) = 1, the Euclidean Algorithm seeded with the matrix

$$\begin{bmatrix} p & 0 \\ s & r \end{bmatrix}$$

will terminate with $V_2 = 0$. At this point,

$$|r/s|_p = |U_1|_p .$$

2.42 Example. Compute $|-21/5|_{997}$. We use the seed matrix

$$\begin{bmatrix} 997 & 0 \\ 5 & -21 \end{bmatrix} .$$

Q		
	997	0
	5	-21
199	2	4179
2	1	-8379
2	0	20,937

Hence, $|21/5|_{997} = |-8379|_{997}$

$= 594$.

2.43 Example. Compute $|2/47|_{997}$. (Note that r/s is not an order-N Farey fraction.)

Q		
	997	0
	47	2
21	10	-42
4	7	170
1	3	-212
2	1	594
3	0	-1994

Thus, $|2/47|_{997} = |594|_{997}$

$= 594$.

Hence, -21/5 and 2/47 belong to the generalized residue class Q_{594}. Only -21/5 is an order-22 Farey fraction.

We now present an instance in which the true result is not in F_{22}, but we map the final result incorrectly to a member of F_{22}.

2.44 Example. Compute $|9/10 + 13/17|$ modulo 997.

Q		
	997	0
	10	9
99	7	-891
1	3	900
2	1	-2691
3	0	8973

Hence, $|9/10|_{997} = |-2691|_{997}$

$= 300$.

Q		
	997	0
	17	13
58	11	-754
1	6	767
1	5	-1521
1	1	2288
5	0	-12,961

Thus, $|13/17|_{997} = |2288|_{997}$

$= 294$.

$$|9/10 \div 13/17|_{997} = |300 \div 294|_{997}$$
$$= 594$$

We now map $594 \in I_{997}$ to F_{22}:

Q	997	0
	594	1
1	403	-1
1	191	2
2	21	-5
9	2	47
10	1	-475
2	0	997

We recover an order-22 Farey fraction. However, the true result is

$$9/10 \div 13/17 = (153 \div 130) \,/\, 170$$
$$= 283/170 \ ,$$

which is not in F_{22}. Note that

$$|283/170|_{997} = |(283)\ (170^{-1})|_{997}$$
$$= |(283)\ (217)|_{997}$$
$$= 594 \ .$$

2.45 Observation. If we seed the algorithm with r = 1, we obtain the multiplicative inverse (modulo p) of s. For example, suppose r/s = 1/21 .

Q	997	0
	21	1
47	10	-47
2	1	95
10	0	-997

Hence, $21^{-1}\ (997) = |1/21|_{997}$
$$= 95 \ .$$

2.46 Example. Compute -19/10 divided by 9/10 . Using Tables 1 and 2 (pages 14 and 15 , respectively) or the preceding results, we get

$$|(-19/10)/(9/10)|_{997} = |696/300|_{997} \ .$$

Rather than multiply modulo 997 the residue 696 by the multiplicative inverse of 300, we instead use an Extended Euclidean Algorithm to compute $|696/300|_{997}$ directly.

Q	997	0
	300	696
3	97	-2088
3	9	6960
10	7	-71,688
1	2	78,648
3	1	-307,632
2	0	693,912

Thus, $|696/300|_{997} = |-307,632|_{997}$

$= 441$.

Now we perform the inverse mapping from $441 \in \hat{I}_{997}$ to F_{22}.

Q	997	0
	441	1
2	115	-2
3	96	7
1	19	-9
5	1	52
19	0	-997

The unique order-22 Farey fraction that we extract, -19/9, is the correct result.

2.47 Note. The preceding example illustrates that T_1 (or V_1 or U_1) of the generated pairs may be much larger than the prime modulus. If the modulus approaches in size the amount of computer memory allocated for T_1, V_1, U_1, overflow may result. If each time T_1 is

computed, we perform the computation $U_1 \sim (V_1)(Q)$ modulo p, T_1 can be restricted to a value less than p. To illustrate, we re-work the foregoing example, reducing T_1 modulo p at each step. Compute $|696/300|_{997}$.

Q		
	997	0
	300	696
3	97	903
3	9	978
10	7	96
1	2	882
3	1	441
2	0	0

Hence $|696/300|_{997} = 441$, which agrees with the result obtained previously.

CHAPTER III

MULTIPLE-PRECISION INTEGER ARITHMETIC

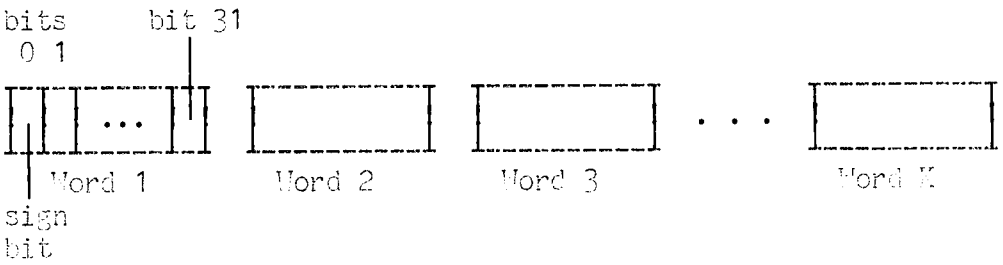
Introduction

A large proportion of exact residue and rational arithmetic computations having practical value cannot be performed using machine integer arithmetic instructions. On the IBM 360/370, for example, the word size of 32 bits is adequate to represent integers in the range $-2^{31} = -2,147,483,648$ to $+2^{31} - 1 = +2,147,483,647$ using a two's complement representation. The largest positive integer that can be stored in 32 bits is $2^{31} - 1$, a Mersenne prime, and hence the largest integer satisfying $[(p-1)/2]^{\frac{1}{2}}$ is $M = 2^{15} = 32,768$). Furthermore, addition, subtraction, multiplication, and division of residues modulo $(2^{31} - 1)$ will frequently fail due to overflow. A double-precision integer of 64 bits allows us to employ the Mersenne prime $2^{61} - 1$, which leads to $M = 2^{30} = 1,073,741,824$. These single- and double-precision constraints severely restrict the size of integer operands we can use. In iterative approximation algorithms, for instance, exact computation is helpful, but rational operand numerators and denominators frequently increase rapidly in magnitude. Consequently, we have strong motivation to develop software routines allowing us to perform multiple-precision integer arithmetic on much larger operands than single- or double-precision.

We have chosen to maintain extended-precision integers in two's complement form and to develop assembler language routines to perform two's complement arithmetic upon these integers. This choice offers us

the advantage of two's complement arithmetic, the computational speed of assembler code that uses algorithms utilizing the machine add, multiply, and divide instructions, and minimization of computer memory required for program code and for fixed storage of integer operands.

Each two's complement integer operand is stored in an array of length K words (K will be designated throughout this thesis as the "precision"). Double-length operands of length 2K words are employed for products. The leftmost bit of the leftmost word represents the sign bit:



A word length of 32 bits allows us to store integers in the range $-2^{(32K-1)}$ to $+2^{(32K-1)}$. Thus, for K = 10, the range of representable integers is -2^{319} to $+2^{319} - 1$, or integers in the approximate decimal interval -1.07×10^{96} to $+1.07 \times 10^{96}$.

In the integer arithmetic subroutines described below, K is restricted to the range 2 to 303. The upper limit of K = 303 arises solely from addressability considerations involving working storage within subroutines. The value of K = 303 stems from the storage necessary to hold the largest Mersenne prime, $2^{9689} - 1$, used by the residue arithmetic routines described in Chapter 4. In 303 words we can store integers from -2^{9695} to $+2^{9695} - 1$, equivalent to the approximate decimal range -3.06×10^{2918} to $+3.06 \times 10^{2918}$. Hence, we can easily

store integers of 2918 or fewer decimal digits.

For compatibility with PL1 error routines, which use register R12, the assembler support routines avoid entirely the use of R12. Save areas of 20 words, rather than the usual 18 words, are also established for PL1 compatibility.

Assembler Macros

The following pages briefly describe assembler macros used to shorten coding of integer arithmetic (and residue and rational arithmetic) support routines. This macro code is listed in Appendix A. Macros MULTPREC and PRECISN are used to obtain variable dimensioning of arrays used to store K-word integer operands.

LOENTRY

Description:

This macro establishes standard entry linkage for a low level subroutine using R15 as a base register.

Parameters: None

LORETURN

Description:

LORETURN provides standard return code for a low level CSECT.

Parameters: None

HIGHTRY

Description:

Establishing a permanent base register and creating a local save area, this macro performs standard entry linkage to a high level CSECT.

Parameters:

&BASEREG --- register other than R1, R12, R13, R14, R15 to be used as a permanent base register

&SAVEAREA -- label for 20 word save area established locally within the CSECT

HIRETURN

Description:

Macro HIRETURN provides standard exit linkage for a high level CSECT.

Parameters: None

PRECISN

Description:

This macro is called to establish the precision K, the number of 32-bit words used in storage of multiple-precision integers. Global symbol "&KWORDS" makes this value available to macro "MULTPREC," which establishes storage for individual operands.

Parameters:

&KWORDS -- value of K to be established

MULTPREC

Description:

This macro sets aside array storage for an individual integer operand in multiples of the precision K. Global symbol &KWORDS, initialized in macro PRECISN, provides the necessary value of K. The established storage may be optionally initialized to a chosen value.

Parameters:

&FACTOR -- integer multiplied by K to determine the number of words of storage reserved

&LEFTVAL -- DC constant (such as F'-1') to which lefthand word(s) of operand are initialized

&RTWORDS -- number of low order words to set to "&RTVALUE"

&RTVALUE -- DC constant to which righthand word(s) of operand are initialized

LDARGTBL

Description:

The macro creates an argument list of argument addresses for a call to a CSECT.

Parameters:

&TABNAME -- optional label for argument table

&ARGLIST -- ordered arguments for subroutine call in the form of implicit addresses, indirect implicit addresses, or address labels

CALL

Description:

This macro makes the call to a subroutine by loading necessary registers and branching. If an argument address table has not been previously created, the macro accomplishes this task.

Parameters:

&SUBNAME -- name of called subroutine

&TABNAME -- label of argument list used in call

&ARGLIST -- arguments for call and to be stored in argument table--absent if argument table was generated previously

ZEROLONG

Description:

This macro zeros a specified memory storage of (generally) more than 256 bytes. IBM 360/370 compatible instructions are used.

Parameters:

&LENREG -- register containing length code (# bytes-1) of area to be zeroed

&STORAGE -- D(B) address of area to be zeroed

SETLONG

Description:

The macro sets each byte of a designated area of memory (usually > 256 bytes) to a specified byte value. IBM 360/370 instructions are used.

Parameters:

&LENREG -- register containing length code (# bytes-1) of area to be set

&STORAGE -- D(B) address of area to be set

&VALUE -- byte value to which area is set

MOVELONG

Description:

The macro copies an area of storage (usually > 256 bytes) into a second area of memory. Overlap of the storage areas is permitted. IBM 360/370 compatible instructions are employed.

Parameters:

&LENREG -- register containing length code (# bytes-1) of area to be copied

&TO -- D(B) address in which copy is made

&FROM -- D(B) address from which copy is made

CMPLONG

Description:

This macro performs a logical compare of (usually) greater than 256 byte areas of memory using IBM 360/370 compatible instructions. The condition code is set appropriately.

Parameters:

&LENREG -- register containing length code (# bytes-1) of operands

&LOCATN1 -- D(B) address of first operand compared

&LOCATN2 -- D(B) address of second operand compared

ERRORCHK

Description:

This code checks a one-word error flag and either aborts execution or makes a stated branch, whichever is specified, if the error flag is set.

Parameters:

&FLAG -- one-word flag, right byte of which is non-zero if error condition is set

&BRANCH -- label or CSECT ("FERROR") at which execution continues if error flag is set

&FLAGSET -- optional integer to which a set error flag is re-set

OVERCHK

Description:

This macro checks for product overflow into the high (left) K words of a 2K product of two K-word integers. Overflow detection results in a program abort.

Parameters:

&LENREG -- register containing length code (#K - 1) for K-word comparisons

&DBLADDR -- D(B) address of 2K word storage checked for overflow

&ERRADR -- integer to which an error flag is set if overflow is detected

BYTEOFF

Description:

This code computes the offset (in bytes) of the low byte or word (whichever is specified) of a multiple-precision integer. The offset is returned in a designated register.

Parameters:

&OFFTYPE -- character value "WORD" or "BYTE" to indicate desired offset

&REG -- register loaded with desired offset

&STORLOC -- optional D(X,B) fullword address at which offset value is to be stored

PUTOUT

Description:

The macro constructs fixed length (133 byte) output records of chosen characters and performs "move mode" QSAM output of the result to a print file.

Parameters:

&LOCATN -- D(X,B) address from which characters are to be printed. First byte is used as carriage control.

&BYTES -- number of bytes printed, including carriage control

Integer Arithmetic Subroutines--Descriptions and Algorithms

The following multiple-precision integer arithmetic routines are given in Appendix B.

ADDITION

Description:

This subroutine performs the addition of two two's complement, K-word integers A and B. The sum C can replace contents of locations A or B if necessary. If the sum C = A + B cannot be contained in K-words, an overflow flag is set.

The employed algorithm is that of Hwang [10] and performs pairwise addition right to left of corresponding words from A and B using the machine add logical instruction. A carry, if any, from the previous addition of words is added to the sum. Written in pseudo-code, the algorithm is as follows:

3.1 Addition of K-word Two's Complement Integers A and B

```
Old_carry = 0
For I = K to 1 do
    C(I) = A(I) + B(I) + Old_carry
    Old_carry = New_carry (from machine ALR instruction)
End
Error_flag = "off"
If (A(1) > 0 and B(1) > 0 and C(1) < 0)
   or (A(1) < 0 and B(1) < 0 and C(1) > 0)
    Then Error_flag = "on"
```

Arguments:

K, number of words to hold integer operands (one word)

A, augend integer (K words)

B, addend integer (K words)

C, integer sum of A + B (K words)

Error_flag, set to 1 upon overflow
(0 otherwise) (K words)

WORDADD

Description:

This procedure adds (at a specified offset) a one-word integer B to a positive K-word operand A; the result is stored in A. The machine ALR instruction is used; a carry (if any) is propagated toward higher words of the sum A. A flag is set if the result overflows, i. e., if the sum is negative or if there is a carry from the lefthost word. The procedure is used by subroutine MULTIPLY.

Arguments:

K, number of words to hold integer operands (one word)

Offset in operand A at which to add word B (one word)

A, multiple-precision integer augend and sum (K words)

B, one-word addend integer (K words)

Error_flag, set to 1 upon overflow
(set to 0 otherwise) (K words)

SUBTRACT

Description:

This subroutine computes the difference A - B of two K-word integers A and B. The difference C can replace the contents of A and B if necessary. If the difference does not fit in K-words, an overflow flag is set.

The algorithm calls TWOSCOMP to compute the two's complement of B.

The sum of A and the two's complement of B is then determined by ADDITION.

Arguments:

K, number of words to hold integer operands (one word)

A, minuend integer (K words)

B, subtrahend integer (K words)

C, difference A - B (K words)

Error_flag, set to 1 if result is incorrect
(set to 0 otherwise) (K words)

MULTIPLY

Description:

This procedure multiplies two signed two's complement K-word integers A and B and places the product in 2K memory location C. The employed algorithm complements negative A or B to provide non-negative operands (the procedure fails with operand overflow if either A or B is $-2^{(32K-1)}$. The relatively fast machine multiply instruction (MR) is used to obtain partial products of each combination of halfwords from the multiplicand and multiplier. This multiplication of halfwords, easily loaded into registers from the K-word operands A and B, gives an unsigned multiply without possibility of overflow. Each of length 32 bits, the partial products are accumulated to a running 2K product total by calls to WORDADD.

The multiplication of halfwords is performed only for halfword pairings in which both halfwords are non-zero. The order of execution time is thus not constant for a given K (i. e., it is not proportional to K^2), but depends on (increases with) the magnitude of the integers

stored in K-word operands A and B.

The multiplication algorithm in pseudo-code is listed below.

3.2 Multiplication of K-word Two's Complement Integers A, B

```
If A < 0
    Then A' = -A
    Else A' = A
If B < 0
    Then B' = -B
    Else B' = B
If (A < 0 and B < 0) or (A ≥ 0 and B ≥ 0)
    Then sign_C = "Positive"
    Else sign_C = "Negative"
Product_C = 0
For i = 1 to 2*K Do
    If Multiplier_halfword B'(i) ≠ 0
        Then For j = 1 to 2*K Do
                 If Multiplicand_halfword A'(j) ≠ 0
                     Then Do
                              Partial_product c(i,j)
                                  = A'(j) * B'(i)
                              C = C + c(i,j)
                          End
             End
End
If sign_C = 'Negative'
    Then C = -C
```

Arguments:

K, number of words to hold multiplicand and multiplier (one word)

A, integer multiplicand (K words)

B, integer multiplier (K words)

C, integer product of A and B (2K words)

Error_flag, set to 1 if either A or B is $-2^{(32K-1)}$ (set to 0 otherwise) (one word)

DIVISION

Description:

This procedure performs the integer division A/B of signed two's complement integers and computes the signed integer quotient and remainder. The quotient or remainder can replace the dividend if necessary. The implemented algorithm features an iterative computation that uses the speed of the machine divide instruction (DR) to perform a trial division at each step. The division fails (an error flag is set) if the divisor is 0 or if either A or B is $-2^{(32K-1)}$.

The signs of the quotient and remainder are determined, and negative operands are complemented. The quotient is initialized to 0, and the remainder is set to the dividend. The leading 56 non-zero bits of the current remainder and leading 31 non-zero bits of the K-word divisor form operands for the trial division at each step. The raw trial quotient from the machine divide is multiplied by an appropriate power of 2 to give a product of the trial quotient and full divisor less than the current remainder. At each iteration, this computed product is subtracted from the current remainder, and the final trial quotient is added to the current quotient.

When the current remainder is less than the divisor, the algorithm stops. Provisional quotient and remainder are (two's) complemented if necessary to provide results of the proper sign. The complete division algorithm is represented by the pseudo-code listed below.

3.3 Division of K-word, Two's Complement Integers A and B

```
If A < 0
    Then Do
            Remainder_sign = 'negative'
            Remainder = -A
         End
    Else Do
            Remainder_sign = 'positive'
            Remainder = +A
         End
If (A < 0 and B > 0) or (A > 0 and B < 0)
    Then Quotient_sign = 'negative'
    Else Quotient_sign = 'positive'
Quotient = 0
If B < 0
    Then B' = -B
    Else B' = B
DVSRZEROS = # consecutive lefthand zero bits in B'
DVSRONES = 32 # K - DVSRZEROS
If DVSRONES > 31
    Then Do
            Trial_divisor = left 31 non-zero bits of Divisor
```

```
            TRDVSRONES = DVSRONES
        End
    Else Do
            Trial_divisor = right (low) word of Divisor
            TRDVSRONES = DVSRONES
        End
If TRDVSRONES ≥ 26
    Then MAX_TR_DVND_BITS = 56
    Else MAX_TR_DVND_BITS = 56 - (26 - TRDVSRONES)
While (Remainder < Divisor) Do
    REMZEROS = # consecutive lefthand zero bits in Remainder
    REMONES = 32*K - REMZEROS
    If REMONES > MAX_TR_DVND_BITS
        Then Do
                Trial_dividend = left "MAX_TR_DVND_BITS" non-zero
                                 bits of Remainder
                TRDVNDONES = MAX_TR_DVND_BITS
            End
        Else Do
                Trial_dividend = right (low) 7 bytes of Remainder
                TRDVNDONES = REMONES
            End
    Trial_quotient = Trial_dividend / Trial_divisor
    SHIFT_FACTOR = (REMONES - DVSRONES) - (TRDVNDONES - TRDVSRONES)
    Trial_quotient = Trial_quotient / 2 ** SHIFT_FACTOR
    If Trial_quotient * Divisor > Remainder
        Then Trial_quotient = Trial_quotient / 2
```

```
        Quotient = Quotient + Trial_quotient
        Remainder = Remainder - (Trial_quotient * Divisor)
    End
    If Quotient_sign = 'negative'
        Then Quotient = - Quotient
    If Remainder_sign = 'negative'
        Then Remainder = - Remainder
```

We now illustrate the algorithm by means of three examples performed on an analagous unsigned decimal machine.

3.4 Example. Using an unsigned decimal machine, a word size of 4 decimal digits, and trial dividend and quotient of 7 and 4 decimal digits respectively, compute 19,910,412 / 1814.

For this computation the trial divisor becomes 1814, and TRDVSRONES = 4. On the first iteration REMONES = 8, and we have

$$\text{Trial_dividend} = 1{,}919{,}041.$$

Hence TRDVNDONES = 7, and the raw trial quotient is

$$1{,}919{,}041 \;/\; 1814 = 1057\ .$$

The shift factor is (8-4) - (7-4) = 1. Thus the adjusted trial quotient is

$$1057 \times 10^{1} = 10{,}570;$$

and this value times the divisor gives 19,173,980 , which is less than 19,190,412. Thus, after the first iteration, we have

$$\text{Quotient} = 10{,}570$$

and

$$\begin{aligned}\text{Remainder} &= 19{,}190{,}412 - 19{,}173{,}980 \\ &= 16{,}432\ .\end{aligned}$$

On the second iteration, the trial dividend = 16,432, giving us the trial quotient

$$16,432 \;/\; 1814 = 9.$$

The shift factor is

$$(\text{RELONES}-\text{DVSRONES}) - (\text{TRDVNDONES}-\text{TRDVSRONES}) = (5-4) - (5-4)$$
$$= 0 .$$

Thus the final trial quotient is

$$9 \times 10^{0} = 9 ,$$

and this value times the divisor gives 16,326, which is $\leq$ 16,432. Consequently, after the second iteration, we have

$$\text{Quotient} = 10,570 + 9$$
$$= 10,579$$
$$\text{Remainder} = 16,432 - 16,326$$
$$= 106 .$$

At this point the remainder is less than the divisor, and the computation stops with the result

$$19,190,412 \;/\; 1814 = 10,579 \text{ remainder } 106 .$$

3.5 Example. Using the decimal machine of 3.4, compute 19,190,412 / 7. We have DVSRONES = 1, TRDVSRONES = 1, and the trial divisor is 7. To prevent fixed point divide overflow, we restrict the trial dividend to 4 digits. On the first iteration the trial dividend is 1919. Thus RELONES = TRDVNDONES = 4, and the raw trial quotient is

$$1919 \;/\; 7 = 274 .$$

The shift factor is

$$(8-1) - (4-1) = 4 .$$

The adjusted trial quotient is accordingly

$$274 \times 10^{4} = 2,740,000,$$

and this value times the divisor gives 19,180,000, which is less than the current remainder of 19,190,412. Consequently, we obtain

$$\text{Quotient} = 2,740,000$$

and

$$\begin{aligned}\text{Remainder} &= 19,180,412 - 19,180,000 \\ &= 10,412 \ .\end{aligned}$$

On the second iteration, the trial dividend is 1041, and we have as the raw trial quotient

$$1041 \ / \ 7 = 148 \ .$$

The shift factor is

$$(5-1) - (4-1) = 1 \ ,$$

and thus the adjusted trial quotient is

$$148 \times 10^{1} = 1480 \ .$$

This value times the divisor 7 gives us 10,360, which is less than the current remainder. Hence we get

$$\begin{aligned}\text{Quotient} &= 2,740,000 + 1480 \\ &= 2,741,480\end{aligned}$$

and

$$\begin{aligned}\text{Remainder} &= 10,412 - 10,360 \\ &= 52 \ .\end{aligned}$$

On the third iteration, the trial dividend is 52, and we get the raw trial quotient

$$52 \ / \ 7 = 7 \ .$$

The shift factor is now $(2-1) - (2-1) = 0$, giving the adjusted trial quotient of 7. This value times the divisor 7 gives 49, which is less than the current remainder. So we now have

$$\text{Quotient} = 2{,}741{,}480 + 7$$

$$= 2{,}741{,}487 ,$$

$$\text{Remainder} = 52 - 49$$

$$= 3 .$$

Since the remainder is now less than the divisor, the computation ceases, having yielded the result

$$19{,}190{,}412 \;/\; 7 = 2{,}741{,}487, \text{ remainder } 3 .$$

3.6 Example. Using the decimal machine of 3.4, compute 1,919,041 / 31,876.

Here we have DVSRONES = 5, TRDVSRONES = 4, and the trial divisor is 3187. On the first iteration, the trial dividend is 1,919,041. Thus the raw trial quotient is

$$1{,}919{,}041 \;/\; 3187 = 602 .$$

Since RENONES = 7, TRDVNDONES = 7, the shift factor is

$$(7-5) - (7-4) = -1 .$$

Thus the adjusted trial quotient is

$$602 \times 10^{-1} = 60 ,$$

and this value times the divisor gives us 1,912,560 , which is less than the original remainder of 1,919,041. Accordingly, we obtain

$$\text{Quotient} = 60$$

and

$$\text{Remainder} = 1{,}919{,}041 - (60)\,(31{,}876)$$

$$= 1{,}919{,}041 - 1{,}912{,}560$$

$$= 6481 .$$

Since this remainder is less than the divisor, the computation is complete, and we have

$$1{,}919{,}041 \;/\; 31{,}876 = 60, \text{ remainder } 6481 .$$

Arguments:

K, number of fullwords in integer operands	(one word)
Integer dividend	(K words)
Integer divisor	(K words)
Integer quotient	(K words)
Integer remainder	(K words)

LEFTZERO

Description:

This subroutine computes the number of consecutive leading 0's in a two's complement multiple-precision integer A. A linear search is made for the first non-zero word, which is examined by a binary search for the first non-zero bit. The subroutine is called by procedure DIVISION.

Arguments:

K, number of fullwords in integer A	(one word)
A, integer	(K words)
Number of consecutive leading 0's	(one word)

POWRMULT

Description:

Procedure POWRMULT multiplies (positive, left shift) or divides (negative, right shift) a K-word integer A by a chosen power of 2. The shift is performed word by word upon portions of the original operand whose bits will appear in the final result. The shifted result A' can replace the contents of A if necessary.

Arguments:

K, number of words in operand A	(one word)
Shift factor, a positive (multiply) or negative (divide) integer	(one word)
A, integer to be shifted	(K words)
A', shifted integer result	(K words)

LSTCOMML

Description:

The procedure LSTCOMML calculates the least common multiple of two positive multiple-precision integers A and B:

(3.7) $$LCM(A,B) = B\ (A/gcd(A,B)) \ .$$

The positive greatest common divisor of A and B is computed by an Extended Euclidean Algorithm.

Arguments:

K, precision (number of 32-bit words to hold operands)	(one word)
A, a multiple-precision integer	(K words)
B, a multiple-precision integer	(K words)
LCM(A,B), the least common multiple of A, B	(2K words)

Computational Results

Choquet Series

The Choquet series is a simple integer series defined as follows:

(3.8) If A(N) is odd,

$$R(N) = 1,$$

$$2A(N+1) = 1 + 3A(N) \ .$$

If A(N) is even,

P(N) = 0,

2A(N+1) = 3A(N) .

A(1) is a positive integer.

Appendix C presents an IBM 360/370 assembler main program that computes terms of the Choquet series using multiple-precision integer arithmetic. The addition of 1 to 3A(N) is performed using WORDADD, and the division by 2 to compute A(N+1) is performed by a right shift of 1 bit position in POWRMULT. Results are reported for N = multiples of 500, using the OUTPUT procedure described in Chapter 5.

At each iteration, calls are made to MULTIPLY, WORDADD, and POWRMULT. Hence, for N = 7000, 7000 multiplications, 7000 WORDADD's, and 7000 POWRMULT's of 128-word operands are performed (an additional 14 calls to OUTPUT for conversion and printing of results occupy a significant fraction of total execution time).

For the initial value A(1) = 4, the following results are obtained from a computation of the first 7000 terms of the Choquet series:

```
A(500)  = 3129387438402899667473812093609033460377016934110000
          8232189494595357167859791296607617326886
P(500)  = 0

A(1000) = 3476072795328998325599150079627116505529894009359999
          3283825434139848063402772564848108021750751367639
          7544440644363415753832176115576673155380187262990420
          60161533073439625480096295 4
P(1000) = 0
```

A(1500) = 3861165265067695704738455968102149912677366935186
279152829300467801101536110422577019544507940507024522679037083452505063073415987632522153313992442320593482544293721294715147640218895169905619636481571178032362061331941822737741321999023500525325699469692905072 83

R(1500) = 1

A(2000) = 428891973267802369452723183807916414987481020418952536535665461512894820203147811686348930893443655338394060684844941335195284714200539799059425683187205388329101926095019331095462954375109445624863149549493713253934635884861532827848044941764421420046227252268261455052621634867948235197471317506504023611682751737723168462814709252628761738131902083 2

R(2000) = 0

A(2500) = 476406245539595236059899615696765452052916339252422612656376416787523983392806388327008393511763281518915013097906336093215366601348362994498639262433394152919701660336465332219857291144707684198436012381330194394610178789990410476284675680012850281971292599516413282403749260581601136483764893583079851326105667273688344334810831930656763299899040487293149990212770347470347732189146852204828371103273014692705621964490619905275677333584321

R(2500) = 1

A(3000) = 529184328305011874764566442921131748474985039154636904545304083040660543864528395140894486902635303557327233750175515952087378345947007502658304317298227432772977434998317325382170964138439635650436355943213242845876836728859481833079527252104609921009585505777547449777630125940017547891219752327565710483756006666961280227419321513541125935376986715708979821089707223005577371606643926578850431717312565743401824340280518627597016203065643091386355972946920211926873641176366672979170426831814141652443415851721900820121042990458

R(3000) = 0

A(3500) = 587809366366403232705203883190968966539377505381179813019192467712280086980425381696064746746205730600029421595578258309238193089366416075720630856919953470807422458992735820804483356505097028205533275129939549236310106219164899136406703828944680234751427201334764233130666314155668436400744158503084102005455975450869254059604475812844355442928516615049441240911202407386936952356063206319839769486079404428037212449466761554410035246805620415904197004205598254191169352145431016508511202088394697

```
          503819771521342525437632991793802719702494035241492
          649299872927404636830035622559167250147063174140
          048733147872364062
R(3500) = 0

A(4000) = 652929107509240325270378010399337132741215199743934
          483227782400557429653806977850458732724420059616107
          074483717747468677011261405316643079573556114159051
          613123656310940246469563502265178509911275439473929
          080560411563947027178310538228711507154179235301261
          855469209997725667944421670989177949566758210333741
          642129980622492963470877097667294540397967062431494
          757519049914617522418164240572199927755243722988392
          267984895301550288599194689294243607021533490195264
          435130272779738966041153835315557162405316077737361
          966016486614211426697926814296871323912677567034737
          857281228873166656343552744435669529835223362427112
          123526683302375277902333589662876914391334851159954
          234961555709243247839230411660393022451635994224292
          92336
R(4000) = 0

A(4500) = 725263059464562577660035496918742674261797562927599
          689289860722479180430593144346903107263476029472779
          927341684026950575661605924940574128188180100901871
          460012280250813443364923282188362180102308237377483
          422682981240292115908628950844288171211827631939211
          043266249968915281250171650086001411109984334099
          843048418197220069154148701409102757866940899796047
          247833417859188517037528431991234605705660252673411
          112766024240056402270200048835442240457239304178169
          864408715675574335508330015107869995747205406329641
          691750918129431062503676164791948547003495866671
          230330945631891319511874321292211667674605557632802
          275091745266821238372435592415290182376612702055411
          912507499140566932765220278893205376285893765308536
          632865733002419704815562798626967107946560954348659
          980284897025035918075201743192325051032994380
R(4500) = 0

A(5000) = 805610439746626070215231818694360386220579823475973
          052820664233150236992653560300672875017230606095695
          448990290518459263130027292602832394804381936019003
          030833932966172916034115470326515982697329420328565
          635217955967181908046156531026691568040177830059209
          990906712355913031915210973352181961027067114531719
          986321209613650402547453344501348641040321902936190
          014277945271153988937031015751956249952480372044330
          054600201537238056357532877237481531116838790019667
          728149142452875064435972056564325757033939943890557
          780520081341143402845645560934008861494593617854531
          594699531499997622380130198036052452575437100531936
          367555862643211925566365404441379557111619387406600
```

```
          4944928810918399121003722171537762960754521235256
          6550227842706062397843410284201398266088334722343 5
          0646194476376563365112318371272190907408443210019 6
          4602157553504166932485791875490977081732865094960 0
          5528377763593053907955649650300
R(5000) = 0

A(5500) = 894859006203753401649494202464267991123071626433 5
          8477132732764391229792020273919018430805440760801 2
          9471987257755416104467679228156590085248100539306
          8270892404331798054119954252423406106417169031900 7
          5954276812022142700669507926885358133984932659060 4
          9372960473272952156757137425859609953727439128530 0
          8170314158469689712381527631929702072559155141447 7
          0153373834784971354223744667146738819322382570008 2
          4105165441300150613769783965912744333087125988040 0
          4042869504244159026501464946237100644948655441667 1
          8742330909288569059330052027209154573565946696864 9
          7536547123770810425926497757344824833071919126064
          0299231585932425931232565369355431376520896224905 9
          0181943290045250496823116614280385502791889791643
          8488833668945269925383998036253395443213917652505 1
          7024531271609726214295283396156678931175249452038 9
          1325091027219102737135423331098674079000190919166 7
          6152366870737570102824051785327168222249178752925 0
          5555691820562198871481579936456083018516849877025 3
          3622628431006859593
R(5500) = 1

A(6000) = 9939948658508220145506468129681699745301390254145 8
          5906968538852602924040854910674910526500432605430 9
          7628979003754311952876981770409950757571580402277 3
          9862284625816229682438956501685592041028046003537 2
          4185145602784199082508921403299627381829184627357 9
          7423262994615604915452707397019703679856831753245 5
          1002290423471771351900884000575500150963419121017
          3033290920762562525022816610595510528606054672187 5
          7303807492554746010819171276639256767598561609189 7
          0947833043022316568782254726474070955181779078650 2
          9011502415845356924257337657152799574587360649513 4
          5316826052888238420501506356396234758150730828612 9
          0284899616060237354064139504749237628238628441410 1
          6046017577939878457901401523753342784272343315578 0
          6804905726781091398771375696771758555573609817854 3
          0136950274298294964439374686490125584547870087024 5
          0551265788445777880905867379579694504397673521588 8
          8103891430978653648411610240456044397634046388730 2
          3641275832966928946228603203208185596699813916579 7
          7905338111401235629517050549378802078029369340835 9
          3
          7910526249278723059351384175194286095529166399360 9
          6035651
R(6000) = 1
```

A(6500) = 1104113370361304392305150798864620802673041196650968851225412272934994004846262761311509359261402519547377537574401688436611607646978453803975852943108457609179433558361683477986904646630956725863652081032824813087954036572527211497647222760037337299622676511991649655747099654722504666297819841294134428720411394579253454523317225021310532847258530771109962312199784888387508581967320377304325948237545650499134334936862276199462796792906957772001405978949474704250679679947687953593164301482171964331504096760915230628094416914403650004202065595300621627125945167845557152701287646801548720876213279720807506020945649030830308376726798120826875452619158040454569743951386006666991353092792869404951012524532551087008231070212832279859163736842736390233191217138854270080772271387768923542315712100680195596132894907469002095754127336060625351178165983927229144748325213346427676334580906226229583964590997207665364920805505328172242831420967799210779538047996099190535933738928060262908149228077316639543594202278363578614574995450603892889804607096085633444467407715542802344932065399576405970941776553176346606920579600955306341905294154225

R(6500) = 1

A(7000) = 1226431218603050067445341713016889319885870577091361589821691638007451474128834553665153791658258102062039610561397502523861644922451876915009564241327265313808402369322919813762563173134406276315209906219899299083271526670873744167571598036805176314777245076375933454718091661616565002578919740850079380311937477870197801287191386715835739966464340338115190851449600258542401220918214997176655485027950882489395369018431933109184514096847079029713446973536007878979451255333800688702749144499011972273919419488114745464955248496120055750276487139736927305169356536570868330131052609052557530631215479944989424051769934078751690666720920804077078833054957907593488802798563690112171598053533233427077702848457334754175136292783939201793025763243508494334296969783398742987731667603080028080800760042039800038103606538980781045698765150696884620739994500635858726507586478107547503159643805377597166673335372745142779527811917474717884334849524681187577889388096517010632412436525237833416333050227058258320177042940728748711386319675433024370100592878891262310199147172311535396678149912116933407756729726662939561232186415864265678584459314690052428970721050309499098509681583237536188870934236683331741036330527083592815628366047036 1

R(7000) = 1

Table 3 presents a summary of execution results for the computation of the Choquet series for a total number of terms = 1000, 2000, ..., 7000. The number of decimal digits in A(N) increases approximately linearly with N. Hence the minimal value of K necessary to compute A(N) successfully (without overflow) is given by the approximate relation

(3.9) $$K \geq 18.3\ (N/1000) \quad .$$

TABLE 3

CHOQUET SERIES COMPUTATION: MULTIPLE-PRECISION INTEGER ARITHMETIC *

Total Iter. N	Digits in A(N)	Minimal K	Exec. Time (sec)	Main Pgm Bytes	Assembler Bytes
1000	177	19	1.89	1888	19,504
2000	353	37	7.84	2248	23,320
3000	529	55	22.93	2608	27,136
4000	705	74	56.49	2992	31,168
5000	881	92	116.16	3352	34,984
6000	1057	110	216.48	3712	38,800
7000	1234	129	372.63	4088	42,824

* A(1) = 4, results printed in multiples of 500 iterations

Amamou[1] presents Choquet series results using the identical starting value (4). He stores large integers in arrays using a sign-magnitude system. The leading word contains the sign of the integer, and subsequent digits contain one radix-10000 digit per 32-bit word. This data structure allows him easily to code multiple-precision integer arithmetic in FORTRAN. For the first 1000 Choquet terms, using

an array length of 51 words and a reporting interval of 100 iterations, he has obtained an execution time of approximately 15.4 seconds on the IRIS 80 CII-HB computer at the University of Lille, France. Using the assembler routines described here, a reporting interval of 500 iterations, and a precision K of only 19 words, we obtain identical results in 1.89 seconds on the IBM 370 at the University of Tennessee (see Table 3).

In general, the execution times reported in Table 3 rise faster than the number of iterations N because the minimal K necessary to perform more iterations also increases, and the actual number of digits in A(N) increases linearly with N. In fact, the increase of execution times is faster than N^2, but slower than N^3.

Direct Computation of the Elements of the Inverse Hilbert Matrix

The infinite Hilbert matrix A is a matrix whose individual coefficients a_{ij} are rational numbers given by the simple relation

$$a_{ij} = 1/(i+j-1) , \quad i,j = 1,2,3,\ldots \quad . \tag{3.10}$$

If we consider a finite N X N segment of the Hilbert matrix, we may compute the coefficients of the inverse matrix $B = A^{-1}$ directly from the relationship (see Gregory and Karney, [5])

$$b_{ij}^{(N)} = [\ (-1)^{i+j}\ (N+i-1)!\ (N+j-1)!\ ]\ /\ (i+j-1)\ [(i-1)!\ (j-1)!\ ]^2\ (N-i)!\ (N-j)!\ , \tag{3.11}$$

where i,j = 1,2,3,...,N. These elements are integers and as a result may be easily computed using the four multiple-precision integer arithmetic operations.

Appendix D presents a FORTRAN program for computing the elements of one designated row of the inverse of an N x N Hilbert matrix using relation (3.11). The computations were performed for a variety of Hilbert matrices ranging from N = 7 to N = 100. In each instance a row was chosen to give the largest (or nearly so) integer coefficients b_{ij} for the given N.

Table 4 summarizes execution data on the IBM 370 under the FORTRAN H optimizing compiler for each multiple-precision integer computation performed. The elements b_{ij} of row 5 of the 7 X 7 inverse matrix can be easily computed using K = 2 (maximum decimal digits in result = 9), while row 72 of the 100 X 100 inverse matrix requires K = 71 (maximum of 150 decimal digits in the result). The implemented algorithm computes all required factorials (from 0! to 2N!), stores them in an array, and is thus inefficient in terms of memory requirements, which grow quite rapidly with N. To compute the factorials, 2N-1 multiplications are required. To compute each b_{ij}, 1 addition, 1 subtraction, 3 multiplications, and 4 divisions are necessary in the implemented code. Thus, for one row of an N X X matrix, we have

(3.12)

	N	additions,
	N	subtractions,
	5N-1	multiplications,
and	4N	divisions

of K-word integers. (These figures exclude the computations required to convert binary representations to EBCDIC for output--a significant fraction of total execution time.) Hence, N = 7 gives 7 additions and 7 subtractions, 34 multiplications, and 28 divisions of 2-word integers;

whereas N =100 requires 100 additions, 100 subtractions, 499 multiplications, and 400 divisions of 71-word integers.

TABLE 4

DIRECT COMPUTATION OF THE INVERSE HILBERT MATRIX:
MULTIPLE-PRECISION INTEGER ARITHMETIC

Matrix Size N	Row	Max. Dec. Digits	Minimal K	Exec. Time (sec.)	Main Pgm Bytes	Asm. Bytes
7X7	5	9	2	0.91	2008	15,904
10X10	7	13	4	0.90	2296	16,328
15X15	11	21	6	1.08	2800	16,752
20X20	15	28	9	1.47	3640	17,384
25X25	18	36	12	1.99	4720	18,024
30X30	22	43	16	3.01	6320	18,872
35X35	26	51	19	4.24	7928	19,552
50X50	36	74	30	12.46	15,096	21,840
75X75	54	112	50	49.41	33,856	26,080
100X100	72	150	71	138.22	61,488	30,528

To compute one row (column) of the inverse Hilbert matrix, we observe from Table 4 that the minimal precision K rises faster than dimension N of the matrix, at approximately N log N. We note that the following approximate relation holds:

$$K \geq 0.150\ N \log_e N \quad . \tag{3.13}$$

Execution time increases much more rapidly, somewhat faster than N^2. We may attribute this increase to a combination of factors: more computed coefficients (N of them), larger operand storage K, and the

greater number of digits in the computed coefficient (which rises slightly faster than N).

The following machine integer results are obtained (for convenience, we label b_{ij} as b(i,j) in the output).

N X N = 7 x 7

b(5,1) = 48510
b(5,2) = -1940400
b(5,3) = 18711000
b(5,4) = -72765000
b(5,5) = 133402500
b(5,6) = -115259760
b(5,7) = 37837800

N X N = 10 X 10

b(7,1) = 9609600
b(7,2) = -832431600
b(7,3) = 1775840800
b(7,4) = -161602721280
b(7,5) = 771285715200
b(7,6) = -2121035716800
b(7,7) = 3480673996800
b(7,8) = -3363975014400
b(7,9) = 1766086882560
b(7,10) = -388375587600

N X N = 15 X 15

b(11,1) = 66927861000
b(11,2) = -13742520792000
b(11,3) = 700868560392000
b(11,4) = -15619356488736000
b(11,5) = 190425987858506400
b(11,6) = -1428194908938798000
b(11,7) = 7056963079462296000
b(11,8) = -23939267045069376000
b(11,9) = 57052628204449878000
b(11,10) = -96355549856404238400
b(11,11) = 114708987924290760000
b(11,12) = -94111355982964320000
b(11,13) = 50636001316921020000
b(11,14) = -16079676946793460000
b(11,15) = 2283970439789438400

N X N = 20 X 20

b(15,1) = 431623806451200
b(15,2) = -161454280100652000
b(15,3) = 15043739981143104000
b(15,4) = -617257652189248224000
b(15,5) = 1403448977609238067200O
b(15,6) = -199991479309316424576000
b(15,7) = 19258438748304544588800O
b(15,8) = -13168270131145899644160000
b(15,9) = 661276173977109308217600O
b(15,10) = -24957732965380920753048000
b(15,11) = 7187827094029705176877824OO
b(15,12) = -15936139218327143550134400O
b(15,13) = 2728162104865881364961280000
b(15,14) = -35958467979815093138764800O
b(15,15) = 3613560329006048768624640000
b(15,16) = -27168620251415848149288960O
b(15,17) = 1478936989492394959739520000
b(15,18) = -55028354031610413672834000O
b(15,19) = 12516737467721336144064000O
b(15,20) = -13124524296038259424320000

N X N = 25 X 25

b(18,1) = -3060399709597230000
b(18,2) = 1809179449378741440000
b(18,3) = -2668313540402471281320OO
b(18,4) = 1739345122632722020416000O
b(18,5) = -631945590151985281792620000
b(18,6) = 1450727267827166212115232000O
b(18,7) = -22746530436637292448524361000O
b(18,8) = 2566922749029174945374700134400
b(18,9) = -21635271367268136183221255640000
b(18,10) = 1399219379143303695361140556800O
b(18,11) = -70835481069129749577657740688000
b(18,12) = 2848761210736845329267329827840000
b(18,13) = -91984653075389480319328666130880O0
b(18,14) = 2401889776811086258385982349056000O
b(18,15) = -50929100991407264270209150606020000
b(18,16) = 8779696871245966096413159969792000O
b(18,17) = -12282900827339399029873971616746000O
b(18,18) = 1387245269911273302197530912008960O0
b(18,19) = -12529688716148203222437404113524000O
b(18,20) = 8915337392997868041180286643328000O
b(18,21) = -48829068287307402265017030464280000
b(18,22) = 1985073815133748221623344828224000O
b(18,23) = -56383888994940503464394489723160O0
b(18,24) = 9982670189523331334006862489600O0
b(18,25) = -82900067835451275835068562920000

N X N = 30 X 30

b(22,1) = -2009718710460279819J
(22,2) = 17281833328471049678314.003
(22,3) = -37098335545117853309503200000
b(22,4) = 3525825810208000778535184128000
b(22,5) = -18730949616730004135968165680000
b(22,6) = 631302375971270509767815954400000
b(22,7) = -14610140701049403226055169230400000
b(22,8) = 2449897485042963969531615569472000000
b(22,9) = -309350597017612596236065040970204000 0
b(22,10) = 30269789600648114255356901858374800000
b(22,11) = -23459086940502288547901598940240470000 0
b(22,12) = 14645332312868546178865290718314912000 00
b(22,13) = -74626583035425753690835635792589956000 00
b(22,14) = 31357043022188870367405391242418102560000
b(22,15) = -10950078515685002350522517576717432640000 0
b(22,16) = 31962391343080547401525186440148181760000 0
b(22,17) = -78289459387636028318620697213800460010000 0
b(22,18) = 16127448035330789685935013289947822558000 00
b(22,19) = -27954243261240035455620689702576225767200 00
b(22,20) = 40719781414251413041225732719920507232000 00
b(22,21) = -49687828511437736151495685759426809420000 00
b(22,22) = 50513207722591186918131826718885726520000 00
b(22,23) = -42429576434152477065718544156058784560000 00
b(22,24) = 29095585201054611872328902113148649408000 00
b(22,25) = -16010478269602062239528811624762504090000 00
b(22,26) = 68947251101435263771928329209615549528000 0
b(22,27) = -22370459183799015563081282356965991908000 0
b(22,28) = 51403020346749060704283521062113843840000
b(22,29) = -74534379502786138021211105540065073568 00
b(22,30) = 51263999331612484475198237705624016000 0

N X N = 35 X 35

b(26,1) = -12829660235542030664904120 0

b(26,2) = 15121892864292206810366989440 0

b(26,3) = -44511021612242962671193619720400

b(26,4) = 58065565664890743027923385603072 00

b(26,5) = -42413266620348632110458787971443904 0

b(26,6) = 19701646430097429109374404735122329600

b(26,7) = -63036717514493154474169373622558502320 0

b(26,8) = 14670363348827497768533963315795433267200

b(26,9) = -25830222012115253768650815097337795715360 0

b(26,10) = 35438882376833682913066740882753118241433 60

b(26,11) = -38761277599661840686166747840511223076568 000

b(26,12) = 34409832855058302350480483050553452087449 6000

b(26,13) = -25151500085960652761552775595266564750396 96000

b(26,14) = 15312993406909863055918521909981136554451 968000

b(26,15) = -78383385001619861517482934026715942738101 011200

b(26,16) = 33987375610458476538745987653860571376954 3680000

b(26,17) = -12558449085080959955430053417970857889090 58860000

b(26,18) = 39727830565116496158997848231050966873415 08160000

b(26,19) = -10796684480810244182716425603364250908927 612980000

b(26,20) = 25266036147890565887974615927208086337567 998464000

b(26,21) = -50978347390784633619079218277043489417511 518640000

b(26,22) = 88699914136211561096081098467858175440398 292480000

b(26,23) = -13296969307945248098713913637827551119416 1128280000

b(26,24) = 17137654821952658017894156352476479603312 1213440000

b(26,25) = -18923422256698488806494863408510573884198 9156548800

b(26,26) = 17810279771010342406112812619774657773363 6853222400

b(26,27) = -14186106716787624517203932650804239957794 9995828800

b(26,28) = 94698880461574762933903727238021315576118 391500800

b(26,29) = -52281673588161067036426016079324267974315 361974400

b(26,30) = 23437742590511466567232296387868728152965 157191680

b(26,31) = -83124930318530250970094701970863792804415 11603200

b(26,32) = 22434944772973168124247039340183841992450 71974400

b(26,33) = -43278078236791754310260359934079720788575 5482400

b(26,34) = 5313185493327938211068056826508861147379557120 0

b(26,35) = -3118508959535472384264426087185348000429956320

N X N = 50 X 50

b(36,1) = -4203361603078651882249470702929862000 0

b(36,2) = 1022030333133426589012679411671302986400 00

b(36,3) = -6209641139838040286085458488596590355264 0000

b(36,4) = 1674621683802818283191673303742381908400 5120000

b(36,5) = -2534853910001278500038695983958546714972 000008000

b(36,6) = 2448297922830503136622740511335571949094907324 80000

b(36,7) = -1635825723254158392024971808314582146728597338496 0000

b(36,8) = 7992177105041745286750576549194101345445432710937 60000

b(36,9) = -2972885537638184429320132785422527442232664011382 4280000

b(36,10) = 8680988891195001314653433556906231298805921171893 00704000

b(36,11) = -2038145217932739439092545269882332565806607579487 9233920000

b(36,12) = 3921969118678589902895262863799950448505016596688 61277440000

b(36,13) = -6283094393077511821866294089037848857984049447114 054828340000

b(36,14) = 8489347487654349024574035901874976850889014908087 1557376320000

b(36,15) = -9779728305777810076309289358959973332224145174116 40340975206400

b(36,16) = 9694501915313515435121408841670561799917180946019 528434503680000

b(36,17) = -8334416896004657154875559746665758523360290386014 5651454202160000

b(36,18) = 625595468673661580211760041257201533556035324319 94

9457399786560000

b(36,19) = -41237239489709672338055476702166517182184751876526
70954675502080000

b(36,20) = 23989668015605164508080765006096427769556443970033
21134680173977600 0

b(36,21) = -12369672570546412949479144456268470568677541422048
3746006946470720000

b(36,22) = 56739973985530636063442068360805972488793222585346
9302207302218240000

b(36,23) = -23226360396926818364641831574084268704161466207350
33644031259351040000

b(36,24) = 85072284382297267687966579151137913867621367270894
10019492022251520000

b(36,25) = -27942897667652941663388929973728505296866050715054
440618421770772672000

b(36,26) = 82454452134057860646065695004444769728457198831308
18543140850391936000 0

b(36,27) = -21889225430053001066338646423397935815678234803318
080606604876360192000 0

b(36,28) = 52332630711143327957108368815336080049493795324622
376527237325192 1920000

b(36,29) = -11275462454002477078258672544420346265128099539529
408246186568781643600 00

b(36,30) = 21900365892548559521836509943773432453054808318102
5235515797151186278400 0

b(36,31) = -38344074963384683337895572965529242005348485944152
566487614316032614400 00

b(36,32) = 60489867059046637137363208852729079277668894517392
529367930429072588800 00

b(36,33) = -85908259610570427961111593087432274068348350135196
203099916360177022800 00

b(36,34) = 10969677929645614806846774907910397562329337510509
31621222870992702384000 0

b(36,35) = -12571478651552344719818864531141604403610645617580
22674564065303747507200 0

b(36,36) = 12900310085093955346292999820688567899077322464720

152193715760662197760000

b(36,37) = -11818095285860035995611612618652922863990974725374
415832128751762338320000

b(36,38) = 962977811339298209066356643447516750213147336658784464254387784411520000

b(36,39) = -69471095231455248503132534373439180991643989938190065864754737359129600000

b(36,40) = 44119245804409440570308477626282488767220166513803353159220001777072128000

b(36,41) = -24490535294388464132366959208668322300882904931593144229007813486450400000

b(36,42) = 11777072080347316157320155820857645443699680420297779378310393030204800000

b(36,43) = -48507924075341815023231573832755137114465106167445473060033039261568000000

b(36,44) = 16862589895993774830388323870444101515005094515479090517914970580224000000

b(36,45) = -48510431005029612044342856299813235220981856355351650123722845375288000000

b(36,46) = 11238508630947265707575238770875799045251093074246991036131492908160000000

b(36,47) = -20146305296289810074716459191832314393441952082932500875318937907200000000

b(36,48) = 26219743818009972784004327817499205662285821998883828799318259712000000000

b(36,49) = -22039454800728868905536045228540275592864673873483079473963988560000000000

b(36,50) = 89805770204993641775945195443340241482844220756981301372214777600000000000

N X N = 75 X 75

b(54,1) = -57363436625843249650823843695649400303754234877749440000000

b(54,2) = 31674629544585620992648360063625345953907174755332716236800000

b(54,3) = -43715937932394498473134838194062918876697199158414

66914369600000

b(54,4) = 26800170791396374642898662909707625214514156620906
00348346163200000

b(54,5) = -92331497890406348342726941970380969614309962885126
3048675427667200000

b(54,6) = 20331708823595919960486583560664229783544322674636
626319307044953600000

b(54,7) = -31038894982822121309677830628299029793303351603167
139654712116802728960000

b(54,8) = 34741962882077513902194095065251667472892503554217
458156174258272067584000000

b(54,9) = -29700613384126049252560945368270503759632227691948
589650190494116202212480000

b(54,10) = 20005746496764410459502779991269613396611535097194
012979190041468888798924800000

b(54,11) = -10880469081033237766705359602283079192227437623368
446980830583881692919821484000000

b(54,12) = 48731220808753263862028774978535643279558401366271
782068397433640554623835176960000

b(54,13) = -18267285754872139677825843347777778780504917784885
118306605231161565859701851136000000

b(54,14) = 58093937394563613966496289730295743422632684577108
465616314733907579438445622067200000

b(54,15) = -15854789217970252577292361881277734543651471028009
577287421346230300621007297349632000000

b(54,16) = 37500023193807901748030629840935163442375653214074
826279813966736015381860738079129600000

b(54,17) = -77523973729917796258243164963742642768001824025331
329344007600061333673992806303194368000000

b(54,18) = 14112166610263119673658066658831888457870286714723
796547869642173063674599051128402534400000

b(54,19) = -22768391029112399754348251913799797199833237659067
687030685207022968742594819411213950080000000

b(54,20) = 32745426885598726543831906277636070038000970434819
284051629563781585918845586139738841088000000

b(54,21) = -42195690708917210378314475851173439237312568067908

599389840562045241246944857072127998182400000

b(54,22) = 489401121299833244942932304402834469129581197639873779698656902480642968646962678199671980032000

b(54,23) = -5129970567235298048577401132832832221472332271639689337954263679153270787191513556079777177600000

b(54,24) = 48776595900325583507766016476606795179519023874425284703614624650316577782047097788414145331200000

b(54,25) = -42207584397162745007711711012419545900835001488452504473999217568504110786279817371116302801920000

b(54,26) = 33338648941556396056724186926265565369773469530119446571867736408539955861567855746299459428352000000

b(54,27) = -24102154059410587504980606669284346076526817965348243612517937816906943570919094173421682603850240000

b(54,28) = 15987284631390775311945679046597389746234523972747379393640918362929281880908382539663338316103680000000

b(54,29) = -97513533426853879512435471836591822185839157769998680239140594658259596509825719727403681617858560000000

b(54,30) = 54801926852647672286541103356349917255187415330683842060730893510736416193639975273876101964824576000000

b(54,31) = -28428499554810979998643197366106519576128471702792243069004151008694515900450737173323227894252748800000

b(54,32) = 13634821376734561424923103741572807508670267649915080884620395746565835418271393777089895245759643648000

b(54,33) = -60551123252637127031091615492971037251638713025379717112120360993660289613856604688785447880949198720000

b(54,34) = 24931394378857258801516433760115290573145515126551329519404815492630489151923631363453181846222685184000000

b(54,35) = -95287442284158135359055461576302545225608443277647614925420242981098734603406767507516981178799530784000000

b(54,36) = 33841135200574207760629465464200271049168162565563

0878230889411440324971689420778527040358747362332 67200000

b(54,37) = -1117828053087485540142582397420485650728271505608 9430760946099393787709655133035577132770615444316 4346880000

b(54,38) = 3436958535986245364758223789787547047696298071974 0048954260299984720091719327960149503017816323705 9788800000

b(54,39) = -9843302504706301639278921500694461435505677343595 8510219772752284463733112613124217463286550000728 1817600000

b(54,40) = 2627385803126797520971109720666374283529516930193 6320040337892115784326089841227888089455986165493 08825600000

b(54,41) = -6539203395980309056100469721997610064376239527225 6070472735914004261754412099851258039133058652913 56556800000

b(54,42) = 1518092683059679926248450948905417179515069061251 4559969178885885146050774115814806976045607960641 575280640000

b(54,43) = -3288151960995815018827360672971953505056156440864 3223402629968869532700855606400534561476336757862 8507952000000

b(54,44) = 6645787543298071184004967491537031762885686245491 0338330330427677987688685298699669166221148761283 72326400000

b(54,45) = -1253416519428328294182223779515501500468393577269 9828752612755063721416241399639989812613658096047 8389350400000

b(54,46) = 2205787993786328580558616033984631147850102277395 4356323340965140196611297996223910242624260263303 0431539200000

b(54,47) = -3621318038513357617811144530538539693207915313075 0367663146225052070137029405511438577955975411763 2677896192000

b(54,48) = 5544564504148927036759104167164772192963187818270 1395254027181681542021205979690229521129856709762 1892300800000

b(54,49) = -7913617283437743632543468643001834208775469010382 5762091082224038349785092128411273132007836615966 87

46329600000

b(54,50) = 10523031327658520592449968566147063221584030250851
34196193665552401841066544936526865703793041551759
371673600000

b(54,51) = -13027310417654178137287821662417638363259075911510
67574614754229535933051131351709941916955327882586
722144000000

b(54,52) = 15001691719706956692350820239628339263988306101559
69857549447085084977575835688059061141739560966522
010828800000

b(54,53) = -16052716656290897316091668253862934694797132908894
11686249941275786354413436016195972240556380645853
568896000000

b(54,54) = 15942320615099688535395569671267157150607909106487
68579961147816122156243426493249112265705084781775
224832000000

b(54,55) = -14673477367698778825431101749042676336009057172858
05031287399626024413755488027779897083378019706408
169472000000

b(54,56) = 12496208794752648704772320482617889196000737748543
60611981201956138161748551181137599821101062805548
302336000000

b(54,57) = -98279045171184727106140866165779103570390658392788
40688517859227221690066781191842879437167771341838
47321600000

b(54,58) = 71224163903351662677805164669068033432760468162176
52007550563548664068267308672384448076048815346933
92384000000

b(54,59) = -47443518661319618794354722222058862267106886612264
32027101485488076932959048339606859707018768144946
90048000000

b(54,60) = 28962578775566367669372829844283761604654959968348
54875401549008857026023928531606765444084582441496
08448000000

b(54,61) = -16148543100192431973548337692783215763121762219194
33886120271569083111549525941142587969645844486294
69184000000

b(54,62) = 81912019236422385624493368688752525166917664027889
80091506997083507328803475051836125733836122146154
7417600000

b(54,63) = -376242649088320600660245463328134496260680356262160679752145216831101212118245797708944565566883760783336000000

b(54,64) = 1556393414760764172431009393816537147157385414657433678049070199438537952100284572029325892485913326387200000

b(54,65) = -576064054113971019884427476761204956399021262964490393619306796188423534049498907458376499777547530243200000

b(54,66) = 1892810536316006414099893971676218165125917411202502372351638063354931486748092476995152856909171140096000000

b(54,67) = -54682279595120991085024312931741158045192438434171465746708221530703160151766948470216373598620837740480000

b(54,68) = 13723699906303366916577555675774859509141046722744218117365432782551019649296757391272138227630797414400000

b(54,69) = -2946544501161740073711622399536145624054598814640475995861610457299339956432711285361252506999953580800000

b(54,70) = 5303752931225705771483320507072300482414658938796817874426542129787737189318689131841177457387532288000000

b(54,71) = -778410414157426451896025520438071750360973026443723229019069938156388189363881012725373600662497024000000

b(54,72) = 8945743868537860573775454725339163479331913935004746891909536781040747013842461006549598042543207219 2000

b(54,73) = -754969703411518904519051204154298615395006239692240888350775914680811500493070349221730158875897600000000

b(54,74) = 4160469464870720609551737220074783373662186064892560269972877349746602045617214342643601864325120000000

b(54,75) = -1123204718369770448527488178749816326067863559537668890224056651885142989140318934410451260024200000 0

N X N = 100 X 100

b(72,1) = -78158824259692315710598183743973702961719092815913
369652256438645478414000000

b(72,2) = 77080446618673656801232231817029452090170552127088
877242479004604331484430400000

b(72,3) = -19002100858519971877403232910385448454666950422822
219741505023476965377986735920000

b(72,4) = 20813184053530377611863765630469503922641089072454
582237579074114546252990289503225600

b(72,5) = -12816539654016074634674003046131220836573723270932
558535772377217904797894020378302080000

b(72,6) = 50473863437452395743097982905309498785488517463345
312342696307370875985888087526186009600000

b(72,7) = -13790940632973549387886602289342876116165521465942
840918113319982743034508883766574301913600000

b(72,8) = 27652350086851313915760907237761753805458853320569
833061584501362866243965477613464501448448000000

b(72,9) = -42393645090963706543541073380542308748187683907932
357505258564136276720709448944967343017481576000

b(72,10) = 51272984686147722018641537187818827675448698269505
3868733009843216252147803108166597866654960832000
0

b(72,11) = -50141227341246168642376400818185265747490623344288
255772804950399403975283330785218881618367450144000
00

b(72,12) = 40444394266794796177913959994019599428652503412819
219018144268256146941640515817811949799505991514240
0000

b(72,13) = -27352394048580480674396626277436958724710915271039
960713752382902120116827993290120229753295533520371
200000

b(72,14) = 15724132554746923259895183826253080586634996640783
768685875094194293352563191208461152948966560921167
15520000

b(72,15) = -77738185946682493055298128202240995349231182372038
121717413093236072748131287147953353099737742513321
292800000

b(72,16) = 33384728488418205765995592068053074196848630120589

449308681132479644728335410748877132200916747506423266816000000

b(72,17) = -1256266384589390480184477595555153430519624268238061642558133836767767500990815133350209801968773531835188716000000

b(72,18) = 417388535339133876548457933159043721788536321161565981788273907164945973246265272297664200477761696494433945280000000

b(72,19) = -1232647395908058592811989806416237063204057244433397829700675963613832421483699543734381665089956013717190377411200000

b(72,20) = 3255103170144591975435540096082767083803085751174749478687508463044156232569249615491197949171068586830841753091200000

b(72,21) = -772733187347368355907741257591822099024558617452788354505817226444395349122960995677475687064088455830278085733824000000

b(72,22) = 16569458500536938495092606702300756465046925019610142077544725619133974912552296650004903043508087253649081910143744000000

b(72,23) = -32230903682682398727129892342746047701971487777507342221123069360440782024595199375097896814375187098717236024033523200000

b(72,24) = 57097473973129817618937096359657536308476697462207882973968997018798242515499883650315202444574779264000375254323790336000000

b(72,25) = -92444709379817099099178970683004555675370764076905153119457037852628925345914235811729952337563130026537644599412259498000000

b(72,26) = 13723750670818208526063682245724387646652979409354991803300838609050067267847061811225889831555763632805588476613778729600000000

b(72,27) = -18735877870587445207116779383733873430876495684766508463162315634443204683800842923263206570121839438007325164120949566560000000

b(72,28) = 23586526064964027161169454272944091915880737645813539228047037131532678168661548606329651016248573798502250677506757141088000000

b(72,29) = -27448940047520585404867897152252324436962922112548393080411311130377947752852166280149837133684870 86

297535393130908814721228800000

b(72,30) = 29597584428631761072177755149748535985936521070958
751630365453508097576372820309504231449866074066968129792050583338828165120000000

b(72,31) = -29633049943742322005924373374873506998555834985096
066038637133245416647542765954755271378618636900817438008577005606445497344000000

b(72,32) = 27601729712204781183009346974146193018512786668305
568404847889875375316840451662844839973692993873108108061705651223133333504000000

b(72,33) = -23962000420077056356638057334842089410888286539283
363268289146945144410050958810609116869290718547130116899233079393969821829720000000

b(72,34) = 19420735596050453903691617304404125505272862145753
309028560371854065250905825153683270281965943127588922170329145069644644332294400000

b(72,35) = -14717702135865682333392725126669516735250444967837
564332197666409154920428418876259514305379002176618809905506178343159079470404800000

b(72,36) = 10444150785180594638493414668374328431274614828788
182692041700866279190720193643145365043790409211964132011143674831882065816820160000000

b(72,37) = -69493956207566535518681376430810803535394693639014
922055139840423454404828650157773823556121643736855075000531667194123375293263360000000

b(72,38) = 43411217114863679341827093825418291590266177606516
831788993835968826221880443494728325155435011135176673827139338639781073731488768000000

b(72,39) = -25488211235325060947434723627524805093332031163779
919034086380886345511099590232282557890493146011527516191063621111064208314203822080000000

b(72,40) = 14080707771129313393312183414275879703668077631564
044922971507951078649506001157220900734393596172202734732509508783386034823329105920000000

b(72,41) = -73263682621657208749577454327404186583147966426731
671239836127307956098210912271164999133641680083492354155088537888555462440134254240000000

b(72,42) = 35936165515528276886296242947236236309372996721739
507447079032826218759100333697592574117170929282944814850100989839501352391602874624000000

b(72,43) = -16631180773364796113903584273788863813090040633840
486094580772030428929513460915021226562676151894567611959916551304824086320756387059200000

b(72,44) = 726781856756479208793698363739950868822943025256117452248678604250616658072333843936041536461909213382038691743988720615540548607964108800000

b(72,45) = -300115961937913472083804197509030723546436148075267651270542974511012692375836916461477397074182516440744959913661913984587704455098316800000

b(72,46) = 11718361354384879308841501028410016771248626383219588654231478557355190111507301104506182970700696252372403183122449578015915812658497536000000

b(72,47) = -43291406184528433289129025466318745595580596333906630367280102779084430586081256380873046943775839618577469175135668992302654477744034816000000

b(72,48) = 15140291161395513694230744362883141517455281971925793925002969185919299674040411486500627772252166592459333040362335769272965736905267404800000

b(72,49) = -50151338297865613908800554200640517205880953980964150374609082941714976507083282006701934783329963170473365966278919970915236447611145890240000

b(72,50) = 15741375792819691523558382824126336763847957399348614882779052307024639450256299267960443579026275834067503883481889964263576750681586430720000000

b(72,51) = -46837044367160229697145024304572624961285315868553665593842590061065115741336365854669188681856869899725441882819066041210478364732917002880000000

b(72,52) = 13215295410226667611443840001876391492578872527950252326628170375728577359914113101265327563495339693408839760785985657782588994639453885568000000

b(72,53) = -35370275289596933240630945696948036172731618741538163413624776676553943832929126532915140884774127348917494011403147717775808910638703728640000000

b(72,54) = 898230218620974063929904284310072273026586563928222590396998088032932057037282314371688272971133726478950423070530562323309487865394507584389120000

b(72,55) = -21648019100661716789958255594654805391301478497355616080860436316141957250269816485295413528968224432172540107914593075350299156944902684313600000000

b(72,56) = 4952264355207014980374780485289874593126909449443 8

688599373187886878009056809079763393958950826378713385402157456211685486068625186510637363200000

b(72,57) = -10754712167759239364467441214128198240486822685509020531567480052777322095990629355664034859366524200579467736546104836018564560371638468811620000000

b(72,58) = 22173663252620260004959238600504798588368030028520496214463945741316616891456621271306498874918402619764541257169439087373455771542879518748160000000

b(72,59) = -43404443852158734692411338261525953084196748142629652568861643574910001806983654759842450660570403942727789618476985534874971582581754218300672000000

b(72,60) = 80664641930543260776116884979738206885939633360674088262198010750681407315535015381917720540323808481250495860781189338065292651033767271805440000000

b(72,61) = -14231741539260831194170117080600276231728069993600411531781736576887898125703820895664944634050395839789649774764088623954953989609997996776448000000

b(72,62) = 23834780197946780813229586596859858087689081875588949306342165202475773202956692588725612872177867706961253934725627652877160528236356989636608000000
1

b(72,63) = -37885517685069445107844164130581124857462027287097358680069414391097260235712151023564380402623753299072624785571501602687400849885188028155904000000

b(72,64) = 57141611674289843213953869560922442219903076995188411684504572506971277863687499786862346667976814243085277588269741060288035509342056951944806400000

b(72,65) = -81758656524435562376582950717410140716465756563514209790429551913912402538476355752600752506470157442534843263432248130694177645274430349871626000000

b(72,66) = 11093724153982929329342763269287595476821527267306018290399234669060044219534285804533776360485753320151490674918963016774881375648691331022187200000 0

b(72,67) = -14269803003912498635014582846223614397186216389595057809066986439972784285180621671371226806380135459060461052531640408589864348907531009208727200000 00

b(72,68) = 17392542347207683749571727719788725396082938015039247296486545678172898995301057312387759476537543986906032041151364567039087481473430124254817600000 0

b(72,69) = -2007665095857606677805234729176996121845421495438 4

404491722877640963595518451324288624763188141607868622326259266765465620206172385938717824938240000

b(72,70) = 219356449820232681599316471219640604665975896065144924832512715377172963677445859796644408513831257624334362393504257849182105224330353465845990400000

b(72,71) = -22670195899070037559365977932458114230425879111877797563532609029566697065313497812556337189177358056379996977822523454077108173061162945228300800000

b(72,72) = 22145474532352667439554000702661240710860225453718506474452518108464739632993132207715330665050826081274185957455312480797474525797361773936204800000

b(72,73) = -20430548274521087632903010379043376712910594225917474155089586268489535922987898440001092631511117691535596411324353033071109739666834765850824400000

b(72,74) = 17784377094235797682160403327523016181006691467783326283760267986173902861588590989196776169660522484196994972946508220493489905634964271676312320000

b(72,75) = -14591937528488946240561030167491204651951691401878816623999445392560258962077213540897391896612252643144612058536924543453869791699068699958816000000

b(72,76) = 11272078725711037836623864573829184545952100236371995910602746176263480468059646439000101676748089343381552172204492583780034386025430703806342400000

b(72,77) = -81875951034550040496312595714075577199092068947578875739064043873705388561139993824346025655319280384433494927549589764044241680930395065006464000000

b(72,78) = 55840777506901778200147279064564061604909283656906337141399173153296657257632924052200564878506520657607478384380774545267508884998217689897472000000

b(72,79) = -35702608177555420941633503153307141049176954268317266294775537089829220647942898375711367957013476549839406002226640209041055395851742330234644480000

b(72,80) = 21361532125110091350554123238647473798644538906028061176633888256679240423924467921642773352633937243274765011927629913557959578297123925104128000000

b(72,81) = -11936810097871963053208492385411972406397174495597095040725926867116401207283450619042980668679901609214804955267125432781770997905341864398236000000

b(72,82) = 62158794030883777509581462331300420608320131234323620527005809725263287521737814708134711640060723333

12875405038176065463792243350543720130057600000

b(72,83) = -3008773381501407444700172400607346355820503675370794254107217024206203314927880106782541433856273344744411504404410185459776363695333712089168000000

b(72,84) = 1349964435530261237701229539468321161829254105748851888590265952729734425579870473151564584692696090379688128611883829192556714714202828688697600000

b(72,85) = -55963988751269042530436459191866561334046059503397371740500007226949278762628504410550936036158632039113764503766100948560552217642108079251200000

b(72,86) = 213579095324747039088673380064435099301421895479873248200140639839079124380947960763692065413819423972250886242305733371228378595352025103104000000

b(72,87) = -747214662590672365635069252495215278676137894288099796632862593397483417844868188270590155806088883691213393834756457156901058084600791043840000000

b(72,88) = 2384799114160225500204814488632523742648948025505147657792246397285739214550065447180340346088996976645284199994368669299479444109405167539200000

b(72,89) = -690403039002558051253767252332806686291926211051135141271234142313390850348630129911364284852391828763050445243720960705385197023356849832920000000

b(72,90) = 1800824029322250626874023362819526176020393425794390360967216833773312175590856653660771357162065599030653637166015492813264934848472003648000000

b(72,91) = -4198080098839174415186433064216529928704758031934949027738695545277062905907964108785474456301446934487707267254108666754898213459439002560000000

b(72,92) = 8661063501596422683845624190025439426981890003335158799814613377342219004635772991216950589396484816097369352197244140815886056615355385600000

b(72,93) = -1562178081896131935045602924587556781702851846237592384382625641208112902675786986105431818235928286254456586555194756737136163653431244800000

b(72,94) = 2425380580254396732194956617043255522839989674474792454922811658511121111695926085618333143264943677606326316536962907477177862311697203200000

b(72,95) = -3175799128453556892022799444968609083572736511093913718880559861448823274394451259114255097543626857277941774027452699049253466347857920000000

b(72,96) = 341037430252099578996393975595514376350621131799923655448649701312383795485422685678598827301034058321319605305517421229201435485859840000000

b(72,97) = -28839175055317159132435225511513042965413491252467502403028899076546343816728813625454429189446528803358117665086072440461261435400040000000

b(72,98) = 18007334114148106313552419410768784324903417535363619363507137705320119568733262734371279703631151037108627498983196627605809403171200000

b(72,99) = -73812555323471094576290645848928762657706122151105842068461161023032295490721076710381919475084594164653072662245599327846925821216000000

b(72,100) = 14899295671542211228690189442832006285969137367961084273988279495344385770703037374227957022180645253349929373263431319880006876800000

CHAPTER IV

MULTIPLE PRECISION RESIDUE ARITHMETIC

In Chapter 2 we delineated by theory and example how arithmetic involving rational operands can be performed using their integer residues with respect to some prime modulus p. If p is chosen suitably large, the integer residue result can be mapped back to an exact rational result. Our purpose in this chapter is to describe a set of IBM 360/370 assembly language subroutines to implement the theory of residue arithmetic. The inverse mapping from $\hat{I}_p$ to F_N utilizes the Kornerup-Gregory algorithm, and, consequently, a common denominator need not be maintained during computations.

Choice of Prime Moduli

In order for residue arithmetic to be practical in numerical computations, namely that it improve computational accuracy in distinction to that offered by machine hardware single- and double-precision integer and real arithmetic, a large prime modulus must be employed. This modulus must be sufficiently large such that the inverse mapping from $\hat{I}_p$ to F_N is always successful, i. e. the true result is an order N Farey fraction. This requirement means that for a given computation we need to know a bound on the magnitudes of numerators and denominators of the rational result. Further, in order to handle efficiently a range of magnitudes of computational results, a selection of large prime moduli should be at hand.

An interesting class of prime moduli exists that offers a range of

values and ease of initialization of the modulus and N. This is the set of what are called Mersenne primes. As of 1971 (see [2]), a total of 24 were known, the largest of these being $2^{19,937} - 1$ and the smallest being 2. They all have the form

$$p = 2^a - 1 \quad , \tag{4.1}$$

where a is a prime. The N associated with a Mersenne prime p must thus satisfy

$$\begin{aligned} N &\leq [(p-1)/2]^{\frac{1}{2}} \\ &= [(2^a - 2)/2]^{\frac{1}{2}} \\ &= [2^{a-1} - 1]^{\frac{1}{2}} \quad . \end{aligned} \tag{4.2}$$

Since a is a prime, N is given by

$$N = 2^{(a-1)/2} \quad . \tag{4.3}$$

The residue arithmetic application subroutines described below are designed to be used with a Mersenne prime. For any Mersenne prime p, the computer storage reserved for operands must be large enough to contain p and each member of I_p. In addition, it is desirable to be able to add two residues modulo p without overflow of operand storage. (Multiplication may be performed by using double operand storage for the product before reduction modulo p.) Thus, for the Mersenne prime $2^{61} - 1$, operand storage of 2 computer words (8 bytes, 64 bits) suffices. For the Mersenne prime $2^{127} - 1$, 5 words are necessary to prevent overflow upon two's complement addition. The number of 32-bit computer words required for a given Mersenne prime to meet these

constraints will be termed here the "precision" and symbolized by "K."

Table 5 presents the list of Mersenne primes for which the residue arithmetic routines are designed. The choice of Mersenne primes ranges from $2^{61} - 1$ to $2^{9689} - 1$, with associated values of precision K ranging from K = 2 to K = 303. Corresponding values of N cover the spread 2^{30} to 2^{4844}. Hence the number of decimal digits in N ranges approximately from 10 to 1459.

TABLE 5

MERSENNE PRIMES USED BY RESIDUE ARITHMETIC SUBROUTINES

Modulus p	N		Precision K	Max. Integer in K (Approx., Dec.)
	Bin.	Approx. Dec		
$2^{61} - 1$	2^{30}	1.07×10^{9}	2	9.22×10^{18}
$2^{89} - 1$	2^{44}	1.76×10^{13}	3	3.96×10^{28}
$2^{107} - 1$	2^{53}	9.01×10^{15}	4	1.70×10^{38}
$2^{127} - 1$	2^{63}	9.22×10^{18}	5	7.31×10^{47}
$2^{521} - 1$	2^{260}	1.85×10^{78}	17	4.63×10^{150}
$2^{607} - 1$	2^{303}	1.63×10^{91}	20	2.28×10^{192}
$2^{1279} - 1$	2^{639}	2.28×10^{192}	41	4.47×10^{394}
$2^{2203} - 1$	2^{1101}	2.72×10^{331}	69	2.36×10^{663}
$2^{2281} - 1$	2^{1140}	1.49×10^{343}	72	1.16×10^{705}
$2^{3217} - 1$	2^{1608}	1.14×10^{484}	101	4.25×10^{972}
$2^{4253} - 1$	2^{2126}	9.77×10^{639}	133	7.63×10^{1280}
$2^{4423} - 1$	2^{2211}	3.78×10^{665}	139	4.79×10^{1336}
$2^{9689} - 1$	2^{4844}	1.55×10^{1458}	303	3.06×10^{2918}

Descriptions of Residue Arithmetic Routines

Initialization and Utilization

The residue arithmetic subroutines (see Appendix E) employ the multiple-precision integer arithmetic subroutines described in Chapter 3 in which multiple precision integers are stored in two's complement form. An application program that utilizes these residue arithmetic routines may be written in assembler or in a higher level language such as FORTRAN or PL1 and compiled and linked to the assembler support routines. The application program must perform several functions in order to interface correctly with the assembler support routines:

1. Determine or estimate the value of N necessary for the result of the computation(s) to be an order-N Farey fraction. The value of the precision K necessary to contain the associated Mersenne prime is then established. (Table 5 may be used.)
2. Storage is established for an integer variable "precision" in the application program and for the Mersenne prime modulus. Storage is also set aside for each operand involved in the computation (arrays of dimension K), and operands are initialized.
3. Subroutine SETUP is called to compute the modulus and N (the latter stored in SETUP) and open a print file FT06F001 for the output of multiple-precision rational numbers. (The value of K in the assembler call to macro PRECISN in subroutine SETUP must be changed at each execution to agree with the precision established in the application program.)
4. Subroutine OPNRES is called to compute the residue modulo p of each

rational number a/b used in the computation.

5. The application program calls RESADD, RESSUB, RESMLT, and RESDIV as necessary to perform residue arithmetic operations needed in the computation.
6. Final residue result(s) are mapped from $\hat{I}_p$ to F_N by calling RATMAP. Each Farey fraction can then be printed by a call to OUTPUT.
7. Subroutine CLSOUT is called to close the output print file FT06F001.

Subroutine Descriptions

SETUP

Description:

This procedure uses the precision K to initialize the Mersenne prime modulus and N, the associated order of Farey fractions. A table lookup and multiple-precision shifts are used to initialize these quantities. The procedure also computes the difference p-N, and (p-1)/2. In addition, the output print file FT06F001 is opened by the procedure.

Arguments:

Precision, established by residue arithmetic main program
(one word)

Modulus, storage established in residue arithmetic main program
(K words)

Fatal Errors:

Execution terminates with an error message if either of the following errors is detected:

1. The caller's PRECISION disagrees with the value of K in SETUP'S

initialization of the number of fullwords of storage in each multiple-precision operand. Hence, the value of K in SETUP must be changed in conjunction with the caller's PRECISION.

2. The specified precision K does not correspond to storage required for one of the Mersenne prime moduli listed in Table 4.

OPNRES

Description:

This procedure reduces a multiple-precision rational operand a/b (either a or b may be negative) to lowest terms with a positive and b positive or negative. The residue modulo p of reduced a/b is then computed by using an Extended Euclidean Algorithm. The procedure enforces the arbitrary size requirement that reduced a/b be an order-N Farey fraction. (Recall from Example 2.29 that in principle the original and intermediate operands in a residue computation need not be Farey fractions as long as the result(s) belong to F_N.)

Arguments:

K, precision (number of 32-bit words to hold operands) (one word)

a, numerator of fraction (K words)

--The computed residue $|a/b|_p$ is stored in this location.

b, denominator of fraction (K words)

p, a prime integer modulus (K words)

Fatal errors:

Execution terminates with an error message if one of the following errors is detected.

1. The denominator b of fraction a/b is 0.

2. Numerator a or denominator b is negative and has no positive complement, i. e., $a = -2^{(32K-1)}$ or $b = -2^{(32K-1)}$.

3. Reduced |a| or |b| exceeds N, the order of Farey fractions associated with the Mersenne prime modulus. This restriction on the size of original operands to order-N Farey fractions is arbitrary and is not required by the theory of residue arithmetic.

RESADD

Description:

This procedure performs the multiple-precision residue addition $|A+B|_p$ by adding A and B and reducing the sum modulo p.

Arguments:

K, precision (number of 32-bit words to hold operands) (one word)

A, residue $\in I_p$ (K words)

B, residue $\in I_p$ (K words)

p, a prime integer modulus (K words)

Result |A+B| modulo p (K words)

Fatal errors:

Execution terminates with an error message if overflow occurs in the K-word addition of residues A and B.

RESSUB

Description:

This procedure carries out the multiple-precision residue subtraction $|A-B|_p$ by adding the additive inverse of B (modulo p) to A and reducing the sum modulo p.

Arguments:

K, precision (number of 32-bit words to hold operands) (one word)

A, residue $\in I_p$ (K words)

B, residue $\in I_p$ (K words)

p, a prime integer modulus (K words)

Result |A-B| modulo p (K words)

RESMLT

Description:

This procedure performs the multiple precision residue multiplication $|A \cdot B|_p$ by reducing the product of A and B modulo p.

Arguments:

K, precision (number of 32-bit words to hold operands) (one word)

A, residue $\in I_p$ (K words)

B, residue $\in I_p$ (K words)

p, a prime integer modulus (K words)

Result |A B| modulo p (K words)

RESDIV

Description:

This subroutine performs the multiple-precision residue division $|A/B|_p$ by using an Extended Euclidean Algorithm to compute the residue of rational number A/B.

Arguments:

K, precision (number of 32-bit words to hold operands) (one word)

A, residue $\in I_p$ (K words)

B, residue $\in I_p$ (K words)

p, a prime integer modulus (K words)

Result |A/B| modulo p (K words)

Fatal Errors:

Execution terminates with an error message if the residue divisor B is 0.

RATMAP

Description:

The procedure RATMAP performs the inverse mapping from multiple-precision residue $|x|_p \in \hat{I}_p$ to the rational equivalent $x = a/b \in F_N$. An Extended Euclidean Algorithm of Kornerup and Gregory is employed; the iterative computation ceases when the (unique) order-N Farey fraction is determined. The inverse mapping fails if no order-N Farey fraction is found.

Arguments:

K, precision (number of 32-bit words to hold operands) (one word)

A, residue $\in I_p$ (K words)

p, a prime integer modulus (K words)

a, reduced numerator of rational equivalent $x = a/b$
of residue A (K words)

b, reduced denominator of rational equivalent $x = a/b$
of residue A (K words)

Fatal Errors:

Execution terminates with an error message if an Extended Euclidean Algorithm fails to produce an order-N Farey fraction (residue

$A = |x|_p \in I_p$ but is not a member of $\hat{I}_p$ and hence has no inverse $\in F_N$).

RESIDUE

Description:

This procedure computes the ordinary residue ($|A|_p$) of multiple-precision integer A modulo p. This residue is a non-negative integer between 0 and p-1:

If $A \geq 0$

Then $|A|_p$ = remainder (A/p)

Else $|A|_p$ = p + remainder (A/p)

Arguments:

K, precision (number of 32-bit words to hold operands) (one word)

A, integer of which ordinary residue is computed (2K words)

p, a prime integer modulus (K words)

Residue result $|A|$ modulo p (K words)

SYMRESD

Description:

This procedure maps a multiple-precision integer A to its symmetric residue $/A/_p$ in S_p, which is given by

$$/A/_p = \begin{cases} |A|_p, & \text{if } 0 \leq |A|_p < p/2 \\ |A|_p - p, & \text{otherwise.} \end{cases}$$

The symmetric residue $/A/_p$ lies between $-(p-1)/2$ and $+(p-1)/2$.

Arguments:

K, precision (number of 32-bit words to hold operands) (one word)

A, integer of which symmetric residue is computed (2K words)

$/A/_p$, the symmetric residue result (K words)

MLINVGCD

Description:

This procedure applies an Extended Euclidean Algorithm described in Chapter 2 to compute one of four values designated by the caller. The seed matrix

$$\begin{bmatrix} U_2 & U_1 \\ V_2 & V_1 \end{bmatrix}$$

is seeded in accordance with the chosen task:

1. a. Compute the ordinary residue modulo p of rational number P/Q.

   b. Compute the residue result of the division $|A/B|_p$, where A and B are ordinary residues modulo p.

   Seed matrices are

   (1a) --> $\begin{bmatrix} p & 0 \\ Q & P \end{bmatrix}$ (1b) --> $\begin{bmatrix} p & 0 \\ B & A \end{bmatrix}$.

2. Compute the greatest common divisor gcd(A,M) of non-negative integer A and positive integer M, where $M \geq A$. The seed matrix is

$$\begin{bmatrix} M & 0 \\ A & 1 \end{bmatrix} .$$

3. Perform the inverse mapping from residue A (modulo p) to the rational equivalent P/Q. The seed matrix is

$$\begin{bmatrix} p & 0 \\ A & 1 \end{bmatrix} .$$

4. Compute the multiplicative inverse $A^{-1}(p)$, where $0 < A < p$. The seed matrix is

$$\begin{bmatrix} p & 0 \\ A & 1 \end{bmatrix} .$$

Remark. The requirement that $A < p$ is imposed by the coding of the procedure. In a more general setting, we could compute $A^{-1}(p)$, where $A > p$ and A is not an integer multiple of p, by substituting $|A|_p$ for A in the seed matrix.

In computations (1), (2), and (4) the algorithm terminates when $V_2 \leq 1$. If the inverse mapping $\hat{I}_p \dashrightarrow F_N$ is designated, the computation ceases when the unique order-N Farey fraction is determined. A one-byte flag informs the caller if the specified computation has failed. The following computations fail:

1. gcd(A,M) where A = 0
2. $A \in I_p \dashrightarrow F_N$ where $A \notin \hat{I}_p$ (i. e., A has no image among the order-N Farey fractions.)
3. multiplicative inverse $A^{-1}(p)$ where A and p are not relatively prime.

Arguments:

K, precision (number of 32-bit words to hold operands) (one word)

U_2, the seed value given by p or the larger of A, M (K words)

A, integer residue (numerator P in computation 1a) (K words)

- In computation 3, the numerator result P is stored here upon

subroutine return.

Computed result (denominator Q in computation 3) (K words)

— In computation 1, the denominator Q or divisor B is stored here upon subroutine entry.

FLAG - A one-byte flag specifying the computation:

X'04' — compute $|P/Q|_p$ or $|A/B|_p$.

X'08' — compute gcd(A,M).

X'0C' — Map $A \in \hat{I}_p$ to F_N.

X'10' — compute A^{-1} (p).

Upon subroutine return, the flag is set to X'00' if the computation has failed.

CHAPTER V

MULTIPLE-PRECISION RATIONAL ARITHMETIC

Principles of Computation in the Field of Rational Numbers

One means of obtaining exact computational results is to maintain operands exactly as fractions and perform all arithmetic operations in the field Q of rational numbers. We may remark that any experimentally determined quantitative value having a finite number of significant digits (or any computer-representable floating-point number, for that matter) can be represented exactly as a rational number a/b if sufficient computer memory is allocated to the storage of a and b.

5.1 Example. The value 6.1987×10^{-5} is represented exactly by the rational number 61,987/1,000,000,000. Hence the numerator 61,987 may be stored in one 32-bit computer word, and the denominator 1,000,000,000 may be stored in a second 32-bit computer word.

In the first part of this chapter, we review the elementary arithmetic operations involving rational numbers; in the second part, we describe subroutines to implement these operations; and in the third part, we describe routines for converting multiple-precision rational numbers to characters for printing. The field Q of rational numbers is defined here as the set of all a/b such that a and b are integers with b not equal to 0. For our purposes throughout this thesis, the numerator is assumed to contain the sign (+ or -), and the denominator is assumed always positive. Furthermore, we assume rational numbers a/b to be maintained in reduced form, i. e., gcd(a,b) = 1.

Rational Addition

The sum P/Q of two rational numbers A/B and C/D is given by

(5.2) $$P/Q = A/B + C/D$$
$$= (AD + BC) / BD \quad .$$

The result P/Q is then reduced to lowest terms, if necessary, by the elimination of common factors in the numerator and denominator. The computation is a simple one, but susceptiple to overflow of whatever computer capacity is designated for holding operands. Elimination of common factors at intermediate points of the computation lessens the likelihood of overflow. Amamou [1] gives the following algorithm for computing P/Q = A/B + C/D (given in the form of machine pseudo-code):

```
(5.3)  GCD1 = gcd(B,D)
       IF GCD = 1
          THEN DO
                  P = A*D + B*C
                  Q = B*D
               END
          ELSE DO
                  TEMP = A * (D/GCD1) + C * (B/GCD1)
                  GCD2 = gcd(TEMP,GCD1)
                  P = TEMP/GCD2
                  Q = (B/GCD1) * (D/GCD2)
               END          ,
```

where gcd is the positive greatest common divisor. The result P/Q is a reduced fraction.

5.4 Example. Suppose we have an unsigned 3-digit decimal machine and wish to compute $P/Q = 19/84 + 31/42$. The customary calculation gives us

$$P/Q = ((19)(42) + (31)(84)) / (84)(42)$$
$$= (798 + 2604) / 3528$$
$$= 3402/3528$$
$$= 27/28 \quad .$$

On our 3-digit machine, the computation quickly fails due to overflow. The second algorithm just presented gives us

$$GCD1 = \gcd(84,42)$$
$$= 42 \quad .$$
$$TEMP = 19(42/42) + 31(84/42)$$
$$= 19 + 62$$
$$= 81 \quad .$$
$$GCD2 = \gcd(81,42)$$
$$= 3 \quad .$$
$$P = 81/3$$
$$= 27 \quad .$$
$$Q = (84/42)\ (42/3)$$
$$= 28 \quad .$$

The second computation is easily handled by a 3 digit machine. In fact, a 2-digit machine is adequate.

Rational Subtraction

The difference P/Q of two rational numbers A/B and C/D may be computed from

(5.5) $$P/Q = A/B - C/D$$

$$= A/B + (-C/D) \quad .$$

Once the numerator C of the subtrahend is complemented, the addition algorithm of the previous section may be employed.

Rational Multiplication

The product P/Q of rational numbers A/B and C/D may be computed from

(5.6) $$P/Q = (A/B)\,(C/D)$$
$$= AB \,/\, CD \quad .$$

To lessen the likelihood of overflow in the intermediate products, one may employ the following algorithm:

(5.7) $$GCD1 = \gcd(A,D)$$
$$GCD2 = \gcd(B,C)$$
$$P = (A/GCD1)\,(C/GCD2)$$
$$Q = (B/GCD2)\,(D/GCD1)$$

The result P/Q is a reduced fraction.

5.8 Example. Compute P/Q = (16/35) (215/44) on a 3-digit machine.

The direct calculation gives us

$$P/Q = (16)(215) \,/\, (35)(44)$$
$$= 3440/1540$$
$$= 172/77 \quad .$$

This calculation would fail on an unsigned 3-digit decimal machine due to overflow. The second algorithm gives

$$GCD1 = \gcd(16,44)$$
$$= 4 \quad .$$

$$GCD2 = \gcd(35,215)$$
$$= 5 \quad .$$
$$P = (16/4)\,(215/5)$$
$$= (4)\,(43)$$
$$= 172 \quad .$$
$$Q = (35/5)\,(44/4)$$
$$= (7)\,(11)$$
$$= 77 \quad .$$

The computation is easily handled.

Rational Division

The quotient P/Q of two rational numbers A/B and C/D is given by

$$P/Q = (A/B) \,/\, (C/D) \tag{5.9}$$
$$= (A/B)\,(D/C) \quad .$$

If fraction C/D is negative, C and D must be complemented to ensure fraction D/C with a positive denominator. The computation may then be completed by the algorithm of the previous section.

Coding and Use of Rational Arithmetic Subroutines

Earlier in this chapter, we outlined arithmetic computations in the field Q of rational numbers and presented several examples of such computations. Our purpose here is to describe a set of IBM 360/370 assembly language subroutines to implement the algorithms already delineated. These subroutines are listed in Appendix F.

Numerator and denominator of each rational operand are maintained as multiple-precision two's complement integers in the forms described

in Chapter 3. By convention, the numerator always contains the sign of the fraction (the denominator is always positive). By employing a sufficiently large precision K, numerator and denominator may be stored exactly throughout a computation without overflow. If we desire, the final computed result may be mapped onto a truncated floating-point equivalent for comparison with other rational numbers (see iterative results in Chapter 6).

Initialization and Utilization of Routines

An application program that uses the rational arithmetic routines described in following pages may be written in IBM assembler or in a higher level language and compiled and linked to the package of assembler support routines. For the application program and the assembler support routines to interface correctly, several requirements must be met:

1. Determine or estimate the value of the precision K necessary to hold the largest reduced numerator and denominator that any operand will assume during the computation. For iterative numerical approximations, these largest values generally obtain at the end of the computations. If the number of decimal digits in the largest numerator/denominator can be estimated, division by 9.6 provides a guideline to the minimum K required. (An additional word is frequently necessary to prevent overflow in computing the truncated floating-point equivalent of any printed rational value.)
2. Establish storage in the main program for an integer variable "precision" and initialize it to the exact value of K in the call to

the PRECISN macro from subroutine SETUP.

3. Set up storage for each operand involved in the numerical computation (associated arrays of dimension K for numerator and denominator) and initialize these operands as necessary.
4. Call subroutine RATSET to open a print file (FT06F001) for the output of multiple-precision rational numbers. A check is made for equality of the application program's precision and K in the call to macro PRECISN from assembler CSECT SETUP.
5. For any rational operand that is unreduced or that does not follow the sign convention, call OPNRAT to reduce the fraction to lowest terms with numerator carrying the sign (positive denominator).
6. The application main program calls RATADD, RATSUB, RATMLT, and RATDIV as necessary to perform multiple-precision arithmetic in the field Q of rational numbers.
7. Each rational result is printed by a call to OUTPUT for the binary-to-EBCDIC character conversion and output.
8. Subroutine CLSOUT is called to close the output print file FT06F001.

Descriptions of Rational Arithmetic Subroutines

RATSET

Description:

The procedure compares the caller's "precision" to the value of "K" used in establishing storage in assembler support routines. Subroutine OPNOUT is called to open the output print file for printing multiple-precision results.

Arguments:

Precision, the number of 32-bit words used by the caller to hold operand numerators and denominators (one word)

Fatal Errors:

Execution terminates with an error message if the caller's "precision" disagrees with assembler's "K."

OPNRAT

Description:

The procedure reduces a multiple-precision rational number P/Q by means of division of P and Q by the positive gcd(P,Q). After reduction P is positive, zero, or negative, and Q is positive. If P is 0, Q is set to 1.

Arguments:

K, the precision (number of 32-bit words to hold P,Q) (one word)

P, numerator of fraction P/Q (K words)

Q, denominator of fraction P/Q (K words)

Fatal errors:

Execution terminates with an error message if one of the following errors is detected:

1. Unreduced numerator P or denominator Q is negative and has no positive complement, i. e., $P = -2^{(32K-1)}$ or $Q = -2^{(32K-1)}$.

2. Denominator Q of fracton P/Q is 0.

RATADD

Description:

This procedure computes the sum P/Q = (A/B) + (C/D) of two multiple-precision rational numbers A/B and C/D by using algorithm (5.3) to minimize the possibility of intermediate overflow. The original operands and the result are all reduced fractions.

Arguments:

K, precision (number of 32-bit words to hold operands)	(one word)
A, numerator of fraction A/B	(K words)
B, denominator of fraction A/B	(K words)
C, numerator of fraction C/D	(K words)
D, denominator of fraction C/D	(K words)
P, numerator of sum P/Q	(K words)
Q, denominator of sum P/Q	(K words)

Fatal Errors:

Execution terminates with an error message if one of the following errors is detected.

1. The intermediate product AD does not fit in K words.
2. The intermediate product BC does not fit in K words.
3. The intermediate sum AD + BC does not fit in K words.
4. The intermediate product BD does not fit in K words.

RATSUB

Description:

The procedure RATSUB computes the difference P/Q = (A/B) - (C/D) of two multiple-precision rational numbers A/B and C/D by using the

algorithm of (5.5). The original operands and the result are all reduced fractions.

Arguments:

K, precision (number of 32-bit words to hold operands) (one word)

A, numerator of fraction A/B (K words)

B, denominator of fraction A/B (K words)

C, numerator of fraction C/D (K words)

D, denominator of fraction C/D (K words)

P, numerator of difference P/Q (K words)

Q, denominator of difference P/Q (K words)

Fatal Errors:

Execution terminates with an error message if numerator C is negative and has no positive complement, i. e., $C = -2^{(32K-1)}$.

RATMLT

Description:

This procedure carries out the multiplication

$$P/Q = (A/B)\ (C/D)$$

of two multiple-precision rational numbers A/B and C/D. The algorithm of (5.7) is employed to minimize the likelihood of intermediate overflow. The original operands and the final product are all reduced fractions.

Arguments:

K, precision (number of 32-bit words to hold operands) (one word)

A, numerator of fraction A/B (K words)

B, denominator of fraction A/B (K words)

C, numerator of fraction C/D (K words)

D, denominator of fraction C/D (K words)

P, numerator of product P/Q (K words)

Q, denominator of product P/Q (K words)

Fatal Errors:

Execution terminates with an error message if one of the following errors is detected:

1. Intermediate product AC of numerators does not fit in K words.
2. Intermediate product BD of denominators does not fit in K words.

RATDIV

Description:

This procedure computes the quotient

$$P/Q = (A/B) / (C/D)$$

of two multiple-precision rational numbers A/B and C/D by using the algorithm of (5.9). If fraction C/D is negative, C and D are (two's) complemented before performing the computation. The original operands and the quotient are all reduced fractions.

Arguments:

K, precision (number of 32-bit words to hold operands) (one word)

A, numerator of fraction A/B (K words)

B, denominator of fraction A/B (K words)

C, numerator of fraction C/D (K words)

D, denominator of fraction C/D (K words)

P, numerator of quotient P/Q (K words)

Q, denominator of quotient P/Q (K words)

Fatal Errors:

Execution terminates with an error message if one of the following errors is detected.

1. The numerator C of divisor C/D is 0 (division by zero).
2. Numerator C of the divisor C/D is negative and has no positive complement, i. e., C = -2 .

REDUCE

Description:

This procedure reduces multiple-precision rational number P/Q to lowest terms by dividing numerator and denominator by the (positive) greatest common divisor, which is computed by the Euclidean Algorithm. If numerator P is 0, the denominator Q is set to 1.

Arguments:

K, precision (number of 32-bit words to hold operands)(one word)

P, numerator of fraction P/Q (K words)

Q, denominator of fraction P/Q (K words)

Rational Number Output and Conversion Subroutines

Introduction

In performing computations with large integer and rational operands, we must not only be concerned with the mechanics of implementing multiple-precision arithmetic operations, but also with the input of multiple-precision data and the output of multiple-precision results. We need the capacity to read in large operands (decimal characters) and convert them to their binary form in multiple-precision

storage. The results must be converted from an internal binary representation used in arithmetic to character format for printing.

The extended-precision integer, residue, and rational computations illustrated in this thesis are all performed on initial operands that can, in fact, fit within a single 32-bit word. A routine for converting decimal character input (EBCDIC in this case) to binary has thus not been implemented. The left K-1 words of initial operands are initialized to an extension of the sign bit (0 or -1).

This section describes routines for printing multiple-precision results (see code in Appendix G). Routines OPNOUT and CLSOUT, in conjunction with the PUTOUT macro (Chapter 3 and Appendix A) allow a PL1, FORTRAN, or other main program to write into a print file (FT06F001) in common with the assembler output routines. Procedure OUTVRT converts a multiple-precision integer to EBCDIC characters for printing. OUTPUT uses OUTVRT to print an extended-precision rational number with appropriate title. FLOAT computes the EBCDIC truncated floating-point equivalent (9K digits) of a supplied rational number. The result can be used for a magnitude comparison of two rational operands or as a basis for an arithmetic comparison of rational operands. Finally, FERROR controls the printing of error messages when fatal error conditions such as overflow arise during execution.

Description of Subroutines

OPNOUT

Description:

This procedure opens the output print file FT06F001.

Arguments: None

CLSOUT

Description:

This routine closes the output print file FT06F001.

Arguments: None

OUTPUT

Description:

This procedure prints a titled multiple-precision rational number P/Q stored as a K-word integer numerator P and K-word denominator Q (numerator contains sign of P/Q). P and Q are printed 100 digits per line. A truncated floating-point equivalent (9K digits) is also printed if denominator Q is not 1.

Arguments:

K, precision (number of 32-bit words to hold P, Q) (one word)

P, numerator of rational number P/Q (K words)

Q, denominator of rational number P/Q (K words)

Title, 64 character title (with leading carriage control) to identify P/Q (64 bytes)

OUTVRT

Description:

This assembler procedure converts a multiple-precision integer in binary form to EBCDIC decimal characters for printing. The conversion proceeds right to left in the character result in series of 9 digits. Successive divisions by 10^9 are employed such that the remainder from each division is converted to EBCDIC characters by the CVD and ED

instructions. The algorithm stops when the final quotient from the division is 0.

Arguments:

K, number of 32-bit words to hold operands (one word)

A, integer (binary) to be converted (K words)

A', signed decimal character output value (3K words)

Address in A' of the leading sign character (one word)

FLOAT

Description:

This subroutine converts a rational number P/Q to an EBCDIC character truncated floating-point equivalent of the form

+0.126971854321067576...E- 712 .

Exactly 9K digits are printed right of the decimal point.

Algorithm Description:

If numerator P is greater than denominator Q, the quotient P/Q is converted directly to characters by OUTVRT. Remaining digits are obtained from successive multiplications of the remainder by 10^9. The quotient of the product divided by Q provides the next 9 decimal characters. The full algorithm in the form of machine pseudo-code follows:

5.10 Algorithm. Convert binary rational number P/Q to truncated floating-point decimal character equivalent.

P' = |P|

Output "+0." or "-0."

Got_exponent = false

```
Exponent = 0
#_output_digits = 0
IF NOT(P' < Q)
   THEN DO
           Convert quotient (P'/Q) to decimal chars. (OUTVRT)
           Remainder = Remainder (P'/Q)
           Exponent = Exponent + # converted characters
           #_output_digits = #_output_digits + # converted chars.
           Got_exponent = true
        END
WHILE (#_output_digits < 9*K) DO
   Remainder = (10**9) * Remainder
   IF Remainder < Q
      THEN DO
              IF Got_exponent
                 THEN DO
                         #_output_digits = #output_digits + 9
                         Output 9 "0"s
                      END
                 ELSE Exponent = Exponent - 9
           END
      ELSE DO
              Convert quotient (Remainder/Q) to 9 chars.
              Remainder = Remainder (Remainder/Q)
```

```
                IF Got_exponent
                   THEN DO
                           #_output_digits = #_output_digits + 9
                           Move 9 chars to output
                        END
                   ELSE DO
                           Exponent = Exponent - # leading 0's in
                                      9 chars.
                           Move (9 - # leading 0's) chars. to
                              output.
                           #_output_digits = 9 - # leading 0's
                           Got_exponent = true
                        END
             END
      END WHILE
      Convert exponent to characters and move to output.
```

Arguments:

K, number of 32-bit words in P and Q	(one word)
P, numerator of rational number P/Q	(K words)
Q, denominator of rational number P/Q	(K words)
Floating-point decimal character output	(3K words)
Number of bytes in floating point output	(one word)

Fatal Errors:

Execution is aborted with an error message if the product of 10^9 times the current remainder used by the conversion algorithm overflows K words. Corrective action is to increase K by 1 or more.

FERROR

Description:

This routine uses a coded error flag (the right byte of a full word) and error message tables to print an error message when a fatal condition is detected during residue or rational arithmetic computations. The value of the error flag provides an offset into the message tables. Program execution ceases after the error message is printed.

Arguments:

Error_flag, a one word flag with values

1 - 10	errors in residue arithmetic
11	overflow in FLOAT routine
50 - 62	errors in rational arithmetic

CHAPTER VI

COMPUTATIONAL RESULTS--RATIONAL VS. RESIDUE ARITHMETIC

Introduction

Our purpose in this chapter is to present and evaluate several applications of multiple-precision residue and rational arithmetic to situations in which ordinary single- or double-precision arithmetic provides limited computational accuracy. A secondary goal is to compare residue arithmetic to rational arithmetic in terms of execution times and computer memory requirements for several specific computational circumstances.

The freedom from roundoff error given us by exact computation allows us much greater leeway to choose computational algorithms based on execution efficiency rather than insensitivity to roundoff error. We can also achieve much greater precision in numerous common numerical approximation techniques than is possible with hardware floating point arithmetic. We now present several circumstances in which extended-precision residue or rational arithmetic is of interest.

Large Systems of Simultaneous Linear Algebraic Equations

Background--Hilbert Matrices

Consider the N by N system of linear algebraic equations

$$AX = B \quad , \tag{6.1}$$

where A is the N by N matrix of coefficients and X is the column vector $X = (x_1, x_2, \ldots, x_N)^T$ of unknowns. If N is large, the accuracy of the

solution vector X may be highly susceptible to the effects of roundoff error.

Consider, for example, the case in which matrix A is a finite segment of the (infinite) Hilbert matrix. In the Hilbert matrix, the element a_{ij} is given by

$$a_{ij} = 1 / (i+j-1) \quad . \tag{6.2}$$

6.3 Example. The 7 X 7 Hilbert matrix is given by

$$A = \begin{bmatrix} 1/1 & 1/2 & 1/3 & 1/4 & 1/5 & 1/6 & 1/7 \\ 1/2 & 1/3 & 1/4 & 1/5 & 1/6 & 1/7 & 1/8 \\ 1/3 & 1/4 & 1/5 & 1/6 & 1/7 & 1/8 & 1/9 \\ 1/4 & 1/5 & 1/6 & 1/7 & 1/8 & 1/9 & 1/10 \\ 1/5 & 1/6 & 1/7 & 1/8 & 1/9 & 1/10 & 1/11 \\ 1/6 & 1/7 & 1/8 & 1/9 & 1/10 & 1/11 & 1/12 \\ 1/7 & 1/8 & 1/9 & 1/10 & 1/11 & 1/12 & 1/13 \end{bmatrix} .$$

We note that the coefficients of a Hilbert matrix are simple rational numbers. Yet, most of these a_{ij} cannot be represented precisely by machine floating point arithmetic. Only those coefficients for which the denominator is a power of 2 can be stored exactly in a binary machine. It is well-known that a system of linear equations in which A is a finite segment of the Hilbert matrix is extremely ill-conditioned. In fact, such a system with N > 7 cannot be solved satisfactorily by single precision machine arithmetic (where word size is 32 bits).

Let us consider the linear system AX = B, where A is an N x N Hilbert matrix, and a single element b_k of column vector B is 1; all other elements of B are 0. Then the elements x_i, i = 0, 1, ... , N, of column solution vector X correspond to the row elements a^{-1}_{ki} (i = 0, 1, ..., N) and to the column elements a^{-1}_{ik} (i = 0, 1, ..., N) of the matrix inverse A^{-1}.

6.4 Example. Consider the system of linear equations

$$\begin{bmatrix} 1/1 & 1/2 & 1/3 & 1/4 \\ 1/2 & 1/3 & 1/4 & 1/5 \\ 1/3 & 1/4 & 1/5 & 1/6 \\ 1/4 & 1/5 & 1/6 & 1/7 \end{bmatrix} \begin{bmatrix} x \\ x \\ x \\ x \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix},$$

where $N = 4$ and $b_3 = 1$.

The solution vector X is given by

$$X = \begin{bmatrix} 240 \\ -2700 \\ 6480 \\ -4200 \end{bmatrix},$$

and we observe that these x_i are elements of row 3 and column 3 of the inverse matrix A^{-1}:

$$A^{-1} = \begin{bmatrix} 16 & -120 & 240 & -140 \\ -120 & 1200 & -2700 & 1680 \\ 240 & -2700 & 6480 & -4200 \\ -140 & 1680 & -4200 & 2800 \end{bmatrix}.$$

Such a system of linear equations may be easily solved by standard (unmodified) Gaussian elimination in which the augmented coefficient matrix [A,B] is reduced to upper triangular form. Back substitution is then employed to obtain the solution vector X. Since the computation is exact, the customary technique of maximal column pivoting to minimize roundoff error is unnecessary. Thus, row interchanges are necessary only to obtain a non zero pivot.

Results -- Hilbert Coefficient Matrices

Appendix H presents PL1 residue and rational arithmetic main programs designed to solve the system AX= B by standard Gaussian elimination and back substitution, where A is an N by N Hilbert matrix, and [$b_i = 0$ if $i \neq k$; $b_i = 1$ if $i = k$]. Table 6 presents residue and

rational arithmetic execution data for linear systems in which N = 7, 10, 15, 20, 25, 30, and 35. Programs were run on the IBM 370 on the PL1X optimizing compiler (execution time optimization) at computing facilities of the University of Tennessee. The computed results of solution vectors X agree in each instance with elements of the corresponding row vector of the matrix inverse A^{-1}, computed directly by means of integer arithmetic and presented earlier in Chapter 3.

A disadvantage of using only Mersenne prime moduli is evident from scrutiny of Table 6. Because of gaps in Mersenne primes, a precision M

TABLE 6

RATIONAL/RESIDUE ARITHMETIC EXECUTION DATA: LINEAR SYSTEMS AX = B IN WHICH A IS A HILBERT MATRIX

Dimension N of N X N Hilbert Matrix A ----> N x N =	7x7	10x10	15x15	20x20	25x25	30x30	35x35
Non-zero element k of B --->	5	7	11	15	18	22	26
Max. decimal digits in result --->	9	13	21	28	36	43	51
Residue:							
Minimal Mersenne prime K -->	2	3	17	17	17	17	17
Exec. time (sec.)	1.90	5.17	175.80	408.45	----	----	----
Main pgm. bytes	8192	8192	8200	8192	----	----	----
Assembler bytes	19728	20008	23760	23768	----	----	----
Rational:							
Minimal K	2	2	3	3	4	5	6
Exec. time (sec.)	1.96	4.39	15.76	41.22	97.91	201.47	383.07
Main pgm. bytes	9896	9896	9896	9896	9896	9896	9896
Assembler bytes	22304	22304	22344	22344	22624	22928	23208
Res/rat exec. time	0.95	1.18	11.15	9.91	----	----	----
Res/rat total memory -->	0.87	0.88	0.99	0.99	----	----	----

of 17 must be used to solve a 15 X 15 Hilbert system with $b_{11} = 1$. In fact, a large prime fitting in K = 6 words would be adequate to solve the system correctly. With K = 5 (and using Mersenne prime $2^{127} - 1$), we get pseudo-overflow (and incorrect results) for many elements of the solution vector X. The Mersenne prime $2^{127} - 1$ is in radix-10 notation

(6.5) $$p = 170141183460469231731687303715884105727 ,$$

giving us a Farey N of

(6.6) $$N = 9223372036854775808 .$$

When we attempt to use residue arithmetic with this p and this N, the following values of x are reported by the Gaussian elimination residue arithmetic program. (Compare the corresponding values of b(11,i), reported on page 63.)

(6.7)
$$x_1 = 66927861000$$
$$x_2 = -13742520792000$$
$$x_3 = 700868560392000$$
$$x_4 = -15619356488736000$$
$$x_5 = 190425987858506400$$
$$x_6 = -1428194908938798000$$
$$x_7 = 7056963079462296000$$
$$x_8 = 3041842781779401727 / 7107201032519170954$$
$$x_9 = -8554174009262481727 / 2982179591284786708$$
$$x_{10} = 1894313830671662654 / 3531528463363563207$$
$$x_{11} = 5935382360121614273 / 1483241954612696547$$

$$x_{12} = -2057167141043414273 / 1807870917206501011$$

We now attempt the inverse mapping $|x_{13}|_p \dashrightarrow F_N$. Because $|x_{13}|_p$ is not a member of $\hat{I}_p$, the inverse mapping fails, and execution ceases with an error message. We obtained correct results for x_1 through x_7 because these elements of solution vector X were indeed order-N Farey fractions. For elements x_8 through x_{12}, which are not order-N Farey fractions, we reported incorrect results, errors that were undetected because in each instance, by chance, $|x_i|_p \in \hat{I}_p$. This example demonstrates that some prior knowledge of the magnitude of the results is mandatory if we are to guarantee that the correct answers are always order-N Farey fractions.

The data of Table 6 show that execution times soar with matrix dimension N. We see that larger Hilbert systems may be solved much more rapidly using rational arithmetic as opposed to residue arithmetic. Even so, rational arithmetic execution times increase more quickly than N^3. The Mersenne prime constraints discussed in the previous paragraph account for some of the extraordinary increase of residue arithmetic execution times.

We note also that operands of much smaller precision may be used in rational arithmetic solutions. As remarked in Chapter 3, the maximum number of decimal digits increases somewhat faster than N. Hence, the minimum K for the rational arithmetic computations increases roughly linearly with N (figures for residue arithmetic are distorted by the Mersenne prime restrictions).

Newton-Raphson Approximation to the Cube Root of a

A well-known numerical technique for approximating a root of $f(x) = 0$ is given by the Newton-Raphson method, in which successive approximations are given by the iterative sequence

$$p_n = p_{n-1} - \frac{f(p_{n-1})}{f'(p_{n-1})} \quad , n \geq 1, \tag{6.8}$$

where $f'(p_{n-1}) \neq 0$.

If we consider the equation

$$f(x) = x^3 - a = 0 \quad , \tag{6.9}$$

we have

$$f'(x) = 3x^2 \quad , \tag{6.10}$$

and for $p_{n-1} \neq 0$, we get

$$p_n = p_{n-1} - \frac{p_{n-1}^3 - a}{3p_{n-1}^2} = p_{n-1} - \frac{p_{n-1}}{3} + \frac{a}{p_{n-1}^2} \tag{6.11}$$

Hence, we have the iteration formula

$$p_n = \frac{1}{3}\left[2p_{n-1} + \frac{a}{p_{n-1}^2} \right] \tag{6.12}$$

This relationship allows us to compute an approximation to the cube root of a by providing an initial value p_0. Appendix I presents

residue and rational arithmetic programs designed to implement such a computation. These FORTRAN programs were run on the IBM 370 under the FORTRAN H optimizing compiler. For a = 1041/617 and initial approximation p_0 = 1/1, the following rational/residue arithmetic results are obtained (truncated floating-point equivalents are from the rational computation with K = 92). For convenience, we denote p_n by p(n) in the output.

```
p(1) =  2275 / 1851
    Truncated floating-point equivalent (9K digits):
        +0.122906537007023230686115613182063749324689357 10
        426796326310102647217720151269584008643976229065 37
        007023230686115613182063749324689357104267963263 10
        102647217720151269584008643976229065370070232306 86
        115613182063749324689357104267963263101026472177 20
        151269584008643976229065370070232306861156131820 63
        749324689357104267963263101026472177201512695840 08
        643976229065370070232306861156131820637493246893 57
        104267963263101026472177201512695840086439762290 65
        370070232306861156131820637493246893571042679632 63
        101026472177201512695840086439762290653700702323 06
        861156131820637493246893571042679632631010264721 77
        201512695840086439762290653700702323068611561318 20
        637493246893571042679632631010264721772015126958 40
        086439762290653700702323068611561318206374932468 93
        571042679632631010264721772015126958400864397622 90
        653700702323068611561318206374 9...E+ 1

p(2) =  34249119473 / 28740245625
    Truncated floating-point equivalent (9K digits):
        +0.119167803643292627566052682265480812152940672 71
        911146027305394680321212460062265038488167061376 67
        186342983803222106248091607950549657141282695631 10
        249166494379951911075568617378509269438437515093 43
        615778509895737886547725007482429962704955135539 13
        824631761476116472821550563905453748187999350127 33
        559405687925466364207525801199550464871853369903 13
        213441786651397138155112757495787755641354230771 29
        668755223103629964192416326991638228213646312565 22
        185690262346183410532351704631612729997327571552 37
        430229860813863521042889486439453490230913779812 20
        715471877599863066584351782136169547785484522977 17
        619106117155879387165119261224128803874827705130 30
        366628956046021962277505761574367198888544641657 00
        741118840037749329430095989306632796044518843599 82
        744580318805121555045860885887957681642117141787 64
```

```
        95618850508693242944405072390 53...E+ 1

p(3) =  12040176296129766910153399229 4259 /
        10113711271873470572080283155 6875
    Truncated floating-point equivalent (9K digits):
        +0.11904805241587084264772913970707311627413527726
        37180381124901719580804468574105891091359148464347
        15626239823492097483888736215481221661946882330290
        06425854156318416982668219639039505231414810592775
        09400147482553345524790029373073124329140670934572
        92095455513737696708805031159533179657121148959409
        67338624478938173025697602586131238238742318403467
        46218237863379297463945903894191902371870749110794
        79827074478842993448035991997500750071145508507990
        37361202467930623187258336590238212089005614719174
        88937836787961366344650634452067807491694273812034
        44544715391472282102671863705343155800981749141002
        29787086487884510502111907615312468524995633433493
        96275224075817618372019327203298480965016212751752
        65587702046451067499452525042986467895475396443126
        16387176746925004165984855234007289435410471371381
        46031756504931943540899538483 22...E+ 1

p(4) =  52362376990532520438028598257658328194631843665221
        01260520774632362051369379175304722938550928833 /
        43984281091785557521031580581674788660641423162441
        55905122094428993837775278361443482182322695625
    Truncated floating-point equivalent (9K digits):
        +0.11904793187653496586176572133933762108907784221
        46330628722001714477870214341798869296052372359954
        48566099779514688507710412080034583358518994367351
        95628478294683690385596279648423676799443742630303
        00143172353614151205963947481337353224314657350018
        51261869415609605316196150864268775010185899181608
        53628201899999836119430553031305010762576318419923
        30300384300600731506558221081351057311098456444241
        18699578638060680975457563896090230112552732161992
        45993262617276720461595171016024786737942859299901
        99934249234223368741637264935718321320106490627927
        80057371151191035578729860374444347097288622027619
        70899574254228540002078988197636605837488730992654
        09028696654645662916936677949445673109329845350287
        39805661720244358310275139454954791029568161146372
        62721218221073777862308680841415463900505479266044
        56872122239139709083240620820 08...E+ 1

p(5) =  43070440559499531960790476647671362233962023359425
        28653968188970054146181830538523527841562596727828
        89668926335573755237263871860145965096857815993177
        84837345930674677581866586816528146096725739471897
        41177179535882655991383240053904642743787995396742
        06950110443191531955583991182838399861699 /
        36179074999985884377068463784124974772070234076835
```

```
        459187713164299459517558024706905352846364031524247216935857404321806533539247305130992993982063499886829358904159218458165079234418748033336268479846236308683810997413110023550713250315544360640032779601370161247890319348705513433825360687 5
    Truncated floating-point equivalent (9K digits):
        +0.119047931876412916355550669003511844176375149132843906852414694469125700771475368917856133301920720640061438684092869118053323357856716671993293249259429200141522704202182935414634443030337791257488858619991190942096053592580853438150572711189394876453700614633593214313277060536318536930214426834697327563458737148010328170652732059005374490807652780963822096051609792211355487971530669262637335728604351455320649263398849305270626120795073483677642114682016047323205714412328534071548678262399552035103506673751881338480539762909073810591237027374748773371236841726190030682201692839893194252854097457880043731430968898006658231906293431085963832678664016399569490576054869355416734682648607765031616566481694272558264762874910201871095791902687100854446144740375944149440259329988541235499518493392990784300440740496896494100...E+ 1

p(6) =  2396951226438104691493554937999581583488456076817754252807295411422626049421899764514441845575913948221318196385570820087999014037497356441642420831329364031189628356598800842693515310930907356550127594836092948330389884840858239758145421608107952077680725626962491187994064110479151376147680362102999095572953459019313094346790711913847791692364228785607385119707001891024524974919129548458326715626772774597716295212674296965369749820517220683750643710440159013053011769094126884882149379152624111499039643669162588964836393645397207627007422120141745632422692288902289347679008217481355263910771656971884842703655307938505261918733070479866737737322350429491569232898010647800194408155843518216230661173000210299733339344969296966735238376250391374287663172034867356471891515762762955317747559819456409802449891080818376427097909427738909103012746359796745619277368770732013433739383602842518200697709314433839270047466336364496127409482411131491781714694504764139528675137364072994097838345044808932386923076818516518728771555587263489405604472342796400495929557761115258728649092175225583084522793537311965699873525893587428308278833373082961073251324895791882439433839446145348307253393344406892216724777242702572552241027813681;13 .42148700952097664838945)4657266834996441271707283999061479405403110992017650430384074236957564428993190170379689745005765306402180448576400342612520429123461722000095964 9
```

```
    265976135804696491732145455645362264789779682047 85
    675696302008986668632409934118286944895556449168 58
    712470392748317402805051453820513814596110187342 44
    285924755131668466340231198157685232531078254350 47
    840628206488428858729649883936240390859044246134 23
    274872428323287062346052551987002165002904776958 03
    066314671542618519456 25
Truncated floating-point equivalent (9K digits):
    +0.119047931876412916355550543876752121778751511 77
    595033429792026884739854893378640981221640496504 01
    254668198201761861379974281204068937809040250054 96
    683659336666515437333556819097135999647315963302 66
    589244698256454136853781099964374801570657789620 05
    056363308392310648704386030953243594970044487675 80
    548197192724962786691900696102579403967926005645 64
    789137525969771147783636526496431581110767136935 54
    657642539445740058581519002935894955168911837002 14
    884828708115905800013555227660043150557125540320 90
    660658428482853802115033294762772925037246456100 05
    336603174534493858950503743602950802231627080963 58
    386640033496297458090082000180381975950891798779 43
    462355605592891986502718392537616489966200650120 63
    763690221483995188263322601689059370939706290662 75
    852797316638259516671080202565391772911254426697 76
    827366270037475284671311477 0016...E+ 1
```

Because the sequence p(n) generated by the Newton-Raphson method is quadratically convergent, the number of correct digits in the truncated, normalized floating-point equivalent approximately doubles with each iteration. However, for a = 1041/617 and p(0) = 1/1, the number of digits in the numerator and denominator has grown to 873, and further computation is impractical. At that point we have approximately 48 correct digits in the floating-point equivalent.

If we use a = -1041/617 and p(0) = -110/100, we obtain the following results (truncated floating-point equivalents are from the rational computation with K = 162):

```
p(1) = -1341727 / 1119855
    Truncated floating-point equivalent (9K digits):
        -0.119812565019578427564282875908041666108558697 33
        135093382625429185028418857798554277116233798125 65
        019578427564282875908041666108558697331350933826 25
```

```
        42918502841885779855427711623379812565019578427564
        28287590804166610855869733135093382625429185028418
        85779855427711623379812565019578427564282875908041
        66610855869733135093382625429185028418857798554277
        11623379812565019578427564282875908041666108558697
        33135093382625429185028418857798554277116233798125
        65019578427564282875908041666108558697331350933826
        25429185028418857798554277116233798125650195784275
        64282875908041666108558697331350933826254291850284
        18857798554277116233798125650195784275642828759080
        41666108558697331350933826254291850284188577985542
        77116233798125650195784275642828759080416661085586
        97331350933826254291850284188577985542771162337981
        25650195784275642828759080416661085586973313509338
        26254291850284188577985542771162337981256501957842
        75642828759080416661085586973313509338262542918502
        84188577985542771162337981256501957842756428287590
        80416661085586973313509338262542918502841885779855
        42771162337981256501957842756428287590804166610855
        86973313509338262542918502841885779855427711623379
        81256501957842756428287590804166610855869733135093
        38262542918502841885779855427711623379812565019578
        42756428287590804166610855869733135093382625429185
        02841885779855427711623379812565019578427564282875
        90804166610855869733135093382625429185028418857798
        55427711623379812565019578427564282875908041666108
        55869733135...E+ 1

p(2) = -7200306530767765541 / 6047994210263439885
    Truncated floating-point equivalent (9K digits):
        -0.11905280131632488745591896375608050815557559584
        38249898592572010268758370349130918864162990143550
        51296507886692781549707875893984170231355849463095
        16913763131676316979422827720881251124550304605052
        25038536041600137203939635490988659673222807030329
        82006385762077148328746954716933659242916303859427
        29628483679003073345680179386806563807137029040476
        32328840064432662228233592649611729472745219558179
        58508351373228187191451887036036934614756094333188
        21500753648124855915086045939202075111224666390832
        67437136578793275083997378061824957564417263271338
        35600543405607821165438775649402654267484916200189
        12260780546292254946268298623348780988915553016276
        76334013564578751513877347714547593062919709583597
        24255474506225346747721103203353633594175578278474
        94731455936064860608112339077265395617236247000989
        73459040855620020877100866155829032892957747638951
        19430878814876555250465423973284730130719094657021
        39399638250803203948531359919104048319520071135728
        02001819575006093281522775451138596147940297897825
        72641301041339517338496118915165276978670013421978
        13119453459672634125970995122398955724466485258064
        83161915282412872978200389219635724809502409380496
```

```
        5858863775364247910016993335956522505918511344944 0
        4807836559497017125892273324532813829032502966988 3
        3442783908260090555398880117974870364929121471194 8
        5489826178200210450093163540256044451085669074946 2
        7102391820874118861877015839198494878052862755163 3
        6263355301403167241075154614504069439576700357787 8
        58994836059...E+ 1

p(3) = -11198412178817080630002310475351729350940895309359
        39946967 /
        9406641496055353749965891840178426899090315438435 2
        5095055
    Truncated floating-point equivalent (9K digits):
        -0.11904793207557766944094486047680218889058752072
        3001667410251316208013510968544884280602871925887 1
        7113825751341903198348813931950099842025121849844 5
        4769027534078552263993661754599119477564588136984 4
        3633881303130733737510123944646290406017239845779 9
        3459099023648518430274133892097530192480170242761 3
        2270799640311075856827689294033011048579161616210 0
        4794961827655442453575334486152510011843492468818 4
        2603580087617038631702145912295544020679388077114 1
        3484678856640694279857113129099777323266564963130 1
        3014917787659189609396532316890423499613822840330 8
        5126791592129500156713461401530364619789553004254 8
        5363210325907520160356777226833748558089070894416 4
        0537991352698922455714601644493248968236951187425 6
        2352927205494761353055953684062312676001145187197 9
        9426721014678649187355447514418429674645872936249 4
        7374194888930095482253991614748302881899564953902 0
        2986001586728486037929041435340110561102197816397 1
        7241703481369883976281821560870938066628605733658 2
        3816056538620352910278469613478926350576241168576 9
        8216266924614324864343908937500212435155794558305 8
        2149060307869046332311057890895129091296051617868 4
        4535009457235665316218148873992149752651890815364 3
        6407301866640569241099053317089353986935542197362 0
        4086670303816120124670780778481180437506331957223 3
        9939824579220077032178975463002229768497265913143 2
        1254982583456019245285537576436575722884479577672 0
        6632614999584451335969730971951594292392133525440 0
        3338409544089751069166032410693737714396158231685 6
        32254008987...E+ 1

p(4) = -42129916604716248023595655964031505073712412018028
        11123791794608546156375810001688718091845783019988
        5817462400338793400419692773142128789231211733214 1
        8160854944019942396501 /
        3538903695399977767525162026047059483252486982327 0
        3280246074651392332934596386028877990203483076522 8
        9790083860944680401573518400261640437044702814781 8
        8564490750785877714685
```

Truncated floating-point equivalent (9K digits):

-0.11904793187641291668874909808820407286141939883
83023872012973866945050141465576473804972907413225
75358298290800865778963603537710672773364516461541
92299886454733205397191613460363064442685943950615
94331270024462200871094242098399130333241897438001
24897041179680032582230878621064726273871599907708
99102928419422328822518022908191688678356235842474
74212954037679050143375760404561560966890860642834
32730103702097891176637759579124336193856919208861
30717647292766937682256559690241559374198251289775
08495388159492424288352578572515145724953134263808
04724083646312261317761853295592944925265359491688
96198925335623378781914289725445898048179710699913
05052253501532736751022229047099676498829644983024
97193298323168236397470707835096910257305444309313
25130593941623712053506120308561445593202494065540
31799608350408801231051792012288237413298604498094
77939159079029622405400442554969122923479624483204
94313219990568875775879159990452213767606273792175
30992264896557472821859977632175645647249220808768
46942083673962743450114372281026548714171520306093
47288874653544455938234864371599029495627485783557
46765123354745772964137768939573371273146306797645
22560901667847952481240349548362625889201150999201
78278422151735719061363410510454545174382444517806
89022107071378020283564063345570316714661744477829
29316973956676049241969013110329493726251853009682
59828463939070585962868354643042977052861740723461
94092289277006213454428349345374723310371317515865
34369093291...E+ 1

p(5) = -22433294260133928286373089099845535954200278454513
54344650044237922669525172941565603356773238206023
79134749394412200831952696557606948345687844779741
14789946472153490782614785103602457474412084038072
38688011371163704616907040284825515646917303187669
44230765420752212001627445334845373547843284860511
23569656554586540784064028268749548551339875063572
10351619828952524294406893447775831082630009338835
62891331398651397565554077934422930424337078601631
24511172801930173759001142296768546752535385130713
5882978443325127 /
18843917661184216596212700719002020779264039408618
23607030123703234554359420131102533225704903712893
59001897339154702512261614473441217109261323334577
51357117634246938457357747079341767333156164241030
50002668287839237745277366910638536830455608828145
81359816330142498790540542810238601077193154851780
05531564558515276325850738512410963879680429470406
93580355206462771340865340449758353066484381440074
39817894184665167812197680066865167636433249581331
84160583146300051602424750813344625553047940824010
4918721162509055

Truncated floating-point equivalent (9K digits):

-0.11904793187641291635555054387675212271132778402
77069575919444762954326773478262894986920692899940
94818524916674522003424488593872784244526166714966
76232871513932568246042217416534229302689590150616
74066997136171574095883704304944712316288970420804
88730329097221232714813788202953694769863252622158
58726392958610653884229193018770161388663560538857
96961748533834059666953342330595230292040166654885
95896804039053625863877979095737436495649889245 32
46585826508509859209758567003814230371724933174698
27779971287965202986408358335334101632988374231163
25494631852323354611467106384090222038443511635715
06268665928425815359720811068990995579089769161353
84772535747388348205157637451299553376692795515023
09056109999342777991041088568025479711845479271517
34689091462936644655778738045801352633604477357236
33635676059610514646758666836638209247980224476064
76694421790110197494248407505663288938717467132364
08101910556813697985369465570878469828865590983765
26773212662741905732228448716409508246589336529024
01387837071492754923068086651640480798742138108374
15976197480557649352243126599624389152735312867647
10265255795144316289591641782264024598521513638945
47575693489095352706652536574168345068697543583411
91620086158733103237651156854673874310112427946017
85900696651610281823863228647462130222111238928670
11515877360976868129369842417176711710085238487848
85479662288804041164622959244676709244526922008729
86942932897720236207416800031671662220561626184852
14664701836...E+ 1

p(6) = -33868847137568021387685268856463899522525363062346
51330122328801469250743632128066260671423650408608
18013872514169091192245594438237893446061602107316
71137738733609869813681491464662992357069210384856
20394607069101187942050683261189330813745684883523
30222909702696801582642608505830612854366907700090
85848340840967943264704735712538563604239401667814
87140437832610942145516637189404007222878135799732
12043807937983161988680948307608731117310115142305
12269957843911705455168164407549115455919833453540
26481917851054993076728621616391213733453515102774
16202928610607818550616427084291920268028445085656
88293766676188715624340574068437777642228549375805
05685820974532648454794223199944725569641425499641
81802723760995469646081235982546371563158584042684
02421586208529236843361624782186313424721449302587
40036012570402058113048528947752135193697570210048
53381180245487232971373129647718469001201448663573
20900987670488570567668962067523974114592109171227
87503269274989516359922614272064802069168432814335
97214857131317158126390877325127808021937132150046

```
    43965791298468903103212341255098647446028353237675
    22173826862335666532001840776976078982217586986374
    18286251306155426392041367742245550453303433942105
    78245565067641874272180032652947742976186854017840
    04885808605641208507917073700854362862074402516901
    38010522502653159710184993109616538318248565621588
    24916768781556456413971314107289406454628840342864
    65643209963925420067782263728307103942588074434086
    39797242348250247577378750158412620195038655936785
    96298501867549337044786549356425773470858067141 /
    28449756836370955327215191612888675275710159101801
    66264599789152505619854757085100451899999641648526
    62589239432584424381772965630071927156874419649818
    97414386068236758698087441591113441314941648298721
    77076084968692891125588083826990854448719725617423
    61629949653849619603346368951144307909331131357751
    41044506910215472712103776618717539964864635017497
    02007426730257650450051482607271174542578954985099
    54908558843821013322230529142925211940258760077010
    07454594154167566660882434584127537715315974469134
    88339765449846442776467807108399679458749838073133
    24100255402610688834592018166926764010018948016607
    06320962545109956898085570225253495823525513511623
    67353343074467919127088483548682876826175208775757
    25177457670657767065525660820092852253858496880165
    87697385438457563869350857482871827351043279600040
    01422319144399885830673585775832520496117129903821
    06580771810695555428598923458868801296640066026151
    48196429726351114660596903242441738286269592508354
    67519717744238678248329215291463049326930489848820
    91781873908988109378900409559039018475968644968104
    36341918191313849299219555621306065241370543020262
    22989606536744875010467754877558278089870407133114
    48147164420392175558800968937978649411770206353804
    79859790100352929993033443798306967794021436372321
    21980243137378748975626225419056599806902090755682
    53107208368420658298518597947214045544085071967534
    07672695543675200806853403979116515125064516070778
    39011682454847819577684464288288290906576077365765
    79523953142215674811919915654994481973276424920180
    11312408272977102426653380276642123451051394285
Truncated floating-point equivalent (9K digits):
    -0.11904793187641291635555054387675212177875151177
    58188183131244951144664773544035360441990088067076
    17177010304842812943965613591467678637566849227705
    55652142194379884300042048653920870634832178659042
    42154492181356695905337547223248682711176703305052
    29279948661567191651397484938569561061956368443448
    71108657217806849906529953096661471858074991244940
    66301120123437130557230027671151171329529395697412
    82078021543016386495279217426162613300233642491565
    03828635665773747745844514165595828316718565114631
    85422315432577778891250922293483282663182146218389
```

```
69347109762105513649989718011489336308389378188672
69189281789364165520739679553908206321693283792234
19045078479836987310637962605740407678893895682340
87363882249169144369662164505784456851890559095931
26521381943627512072572875678817239749843941822408
25902172245146944000130605090708549095154941891628
71244936435282196963346395460605403094050457840788
09860778627662085375010912139768673912592051398519
52558794816644507501701598876087203779497271203190
66039088059278585123050947069667774975586076715701
01482413170426055601696842811634054799885530680051
66749996367753105906489093396773031766430180533822
93500249852165551804456432595141180176487550508194
63522300069979718988814710142845432355699257329862
68324210709209911756449594786702019857713473724557
24971055867328288167572223554377290593145445957734
96170845533685543446118137633298302085659186252975
19071112650784640580566893330487930241204780005631
96862590858...E+ 1
```

In this case there are 1547 digits in the numerator of p(6) and approximately 70 correct digits in the truncated floating-point equivalent. Using K=303, p(6) cannot be computed successfully without pseudo-overflow by residue arithmetic.

Table 7 summarizes the growth in the number of digits in the numerator and denominator and number of correct digits in the floating-point equivalent. With the rapid growth of p(n), a tripling of numerator and denominator digits with each new iteration, only a few iterations are practical. Only because of quadratic convergence and reasonable choice of p(0) can we extract a highly accurate numerical approximation.

Table 8 summarizes execution times and memory requirements for the sample calculations. Residue arithmetic execution times were considerably greater than those obtained from rational arithmetic. Indeed, for the approximation to $(1041/617)^{1/3}$ with p(0) = 1/1, a Mersenne prime of $2^{9689} - 1$ (K = 303) is required, and the residue computation was deemed impractical.

TABLE 7

DIGITS IN P(N) OF THE EXACT NEWTON-RAPHSON COMPUTATION OF $A^{1/3}$

	a = 1041/617; p(0) = 1/1		a = -1041/617; p(0) = -110/100	
	Num/Den Digits	Correct Fl. Pt. Digits	Num/Den Digits	Correct Fl Pt. digits
p(1)	4	1	7	3
p(2)	11	3	19	4
p(3)	33	5	58/57	8
p(4)	97	12	172	18
p(5)	291	24	516	35
p(6)	873	~48	1547	~70

TABLE 8

EXECUTION DATA FOR RESIDUE AND RATIONAL ARITHMETIC:
NEWTON-RAPHSON APPROXIMATIONS TO CUBE ROOT OF A

	a = 1041/617; p(0) = 1/1		a = -1041/617; p(0) = -110/110	
	5 Iterations	6 Iterations	5 Iterations	6 Iterations
Max. digits	291	873	516	1547
Residue:				
Minimal Mersenne K	69	303	133	---
Exec. time(sec)	60.84	---	286.66	---
Main pgm. bytes	3408	---	5488	---
Assembler bytes	37,688	---	54,856	---
Rational:				
Minimal K	32	92	55	162
Exec. time(sec)	8.19	92.69	25.99	390.65
Main pgm. bytes	2208	4128	2944	6384
Assembler bytes	31,096	48,616	37,816	69,056
Res/rat exec. time	7.43	---	11.11	---
Res/rat tot. memory	1.23	---	1.48	---

As we implement the Newton-Raphson computations in programs of Appendix I, to compute each new iteration we require

(6.13)
2 multiplications,
1 division,
1 addition,
1 call to OUTPUT,

plus, for residue arithmetic, one call to RATMAP to extract an order-N Farey fraction. Normally, if we are interested only in final results, we convert and print only final results, thereby realizing significant savings in the execution times reported here.

Milne-Simpson Method of Approximate Solutions to First Order, Linear Differential Equations

In the consideration of multi-step methods for approximating the solution to first order initial-value problems, a method is termed "stable" if small changes or "perturbations" in the initial conditions and starting values produce correspondingly small changes in the computed approximations. Methods which are stable for some categories of initial-value problems, but not all, may be termed "weakly stable." We consider here a weakly stable multi-step predictor-corrector method, the Milne-Simpson method.

We examine the initial-value problem

(6.14) $$y' = f(t,y), \quad y(a) = \alpha \ ,$$

for $a \leq t \leq b$. For this problem, the Milne method provides the predictor difference equation

$$(6.15)\quad w_{i+1} = w_{i-3} + \frac{4h}{3}\,[2f(t_i,w_i) - f(t_{i-1},w_{i-1}) + 2f(t_{i-2},w_{i-2})]\quad ,$$

and the Simpson method gives the corrector difference equation

$$(6.16)\quad w'_{i+1} = w_{i-1} + \frac{h}{3}\,[f(t_{i+1},w_{i+1}) + 4f(t_i,w_i) + f(t_{i-1},w_{i-1})]\quad .$$

We consider in particular a category of equations for which the Milne-Simpson method is mildly unstable. These are "stiff" equations, whose exact solution contains a function of the form $e^{\lambda t}$, where λ is constant. A simple form of stiff equation is given by the initial-value problem

$$(6.17)\quad y' = \lambda y, \quad y(0) = \alpha\quad ,$$

which has the exact analytical solution

$$(6.18)\quad y = \alpha\, e^{\lambda t}\quad .$$

Appendix J presents PL1 multiple-precision residue and rational arithmetic programs for computing the solution to one particular problem,

$$(6.19)\quad y' = -y, \quad y(0) = 1\quad ,$$

using a choice of Euler or Runge-Kutta methods to provide starting values for the Milne-Simpson method. For this problem we get

$$(6.20)\quad w_0 = \alpha$$
$$= 1\quad ,$$

and

$$(6.21)\quad t_0 = a$$
$$= 0\quad .$$

In general, Euler's method gives

$$w_{i+1} = w_i + hf(t_i, w_i) \quad , \tag{6.22}$$

where h is the step size (b-a)/N, with N iterations. For the initial-value problem (6.19), we get

$$\begin{aligned} w_{i+1} &= w_i - hw_i \\ &= w_i (1-h) \quad . \end{aligned} \tag{6.23}$$

The Runge-Kutta Fourth Order method gives the difference equation

$$w_{i+1} = w_i + 1/6\ (k_1 + 2k_2 + 2k_3 + k_4) \tag{6.24}$$

where

$$\begin{aligned} k_1 &= hf(t_i, w_i) \\ k_2 &= hf(t_i + h/2, w_i + k_1/2) \\ k_3 &= hf(t_i + h/2, w_i + k_2/2) \\ k_4 &= hf(t_{i+1}, w_i + k_3) \quad . \end{aligned} \tag{6.25}$$

For initial-value problem (6.19), we have

$$\begin{aligned} k_1 &= -hw_i \\ k_2 &= -h(w_i + k_1/2) \\ &= -hw_i(1 - \frac{h}{2}) \\ k_3 &= -h(w_i + k_2/2) \\ &= -hw_i(1 - \frac{h}{2}(1 - \frac{h}{2})) \\ k_4 &= -h(w_i + k_3) \\ &= -hw_i(1 - h(1 - \frac{h}{2}(1 - \frac{h}{2}))) \end{aligned} \tag{6.26}$$

We may combine terms to get for the Runge-Kutta difference equation

$$(6.27)\qquad w_{i+1} = w_i + \Big[-hw_i - 2hw_i\left(1 - \frac{h}{2}\right) - 2hw_i\left(1 - \frac{h}{2}\left(1 - \frac{h}{2}\right)\right) - hw_i\left(1 - h\left(1 - \frac{h}{2}\left(1 - \frac{h}{2}\right)\right)\right) \Big] / 6$$

$$= w_i - hw_i \Big[1 + (2-h) + \left(2 - h\left(1 - \frac{h}{2}\right)\right) + \left(1 - h\left(1 - \frac{h}{2}\left(1 - \frac{h}{2}\right)\right)\right) \Big] / 6$$

$$= w_i - hw_i \Big[1 + 2 - h + 2 - h + \frac{h^2}{2} + \left(1 - h\left(1 - \frac{h}{2} + \frac{h^2}{4}\right)\right) \Big] / 6$$

$$= w_i - hw_i \Big[5 - 2h + \frac{h^2}{2} + 1 + \left(-h + \frac{h^2}{2} - \frac{h^3}{4}\right) \Big] / 6$$

$$= w_i - hw_i \Big[6 - 3h + h^2 - \frac{h^3}{4}\Big] / 6 \quad .$$

Thus we have the iteration formula

$$(6.28)\qquad w_{i+1} = w_i \Big[1 - h\left(1 - h\left(1 - \frac{h}{2} + \frac{h^2}{6} - \frac{h^3}{24}\right)\right) \Big] \quad .$$

Hence, the initial-value problem (6.19) leads to the following difference equation valid for either the Euler or Runge-Kutta methods:

$$(6.29)\qquad w_{i+1} = \beta\, w_i \quad ,$$

where the "STARTFACTOR" β is constant for a given step size h and starting method.

In like fashion, we may simplify the Milne-Simpson difference equations by combining the predictor-corrector steps to approximate the solution to $y' = -y$, $y(0) = 1$. From 6.15 and 6.16 we have

$$(6.30)\qquad w_{i+1} = w_{i-3} + \frac{4h}{3}\Big[-2w_i + w_{i-1} - 2w_{i-2}\Big]$$

and

$$(6.31)\qquad w'_{i+1} = w_{i-1} + \frac{h}{3}\Big[-w_{i+1} - 4w_i - w_{i-1}\Big] = \frac{-h}{3}\Big[w_{i+1} + 4w_i\Big] + \frac{3-h}{3}\, w_{i-1} \quad .$$

Combining these, we have

$$(6.32)\qquad w'_{i+1} = \frac{-h}{3}\left[w_{i-3} + \frac{4h}{3}(-2w_i + w_{i-1} - 2w_{i-2}) + 4w_i\right] + \frac{3-h}{3} w_{i-1}$$

$$= \frac{-h}{3}\left[w_{i-3} - \frac{8h}{3} w_i + \frac{4h}{3} w_{i-1} - \frac{8h}{3} w_{i-2} + 4w_i\right] + \frac{3-h}{3} w_{i-1}$$

$$= \frac{-h}{3} w_{i-3} + \frac{4h(2h-3)}{9} w_i - \frac{4h^2}{9} w_{i-1} + \frac{8h^2}{9} w_{i-2} + \frac{3-h}{3} w_{i-1} \ .$$

Hence,

$$(6.33)\qquad w'_{i+1} = \frac{-h}{3} w_{i-3} + \frac{8h^2}{9} w_{i-2} - \frac{4h^2+3h-9}{9} w_{i-1} + \frac{h(8h-12)}{9} w_i$$

$$= \delta_{i-3} w_{i-3} + \delta_{i-2} w_{i-2} + \delta_{i-1} w_{i-1} + \delta_i w_i \ ,$$

where δ_{i-3} , δ_{i-2} , δ_{i-1} , and δ_i are constant for given step size h.

Using equations (6.29) and (6.33) and the PL1 programs of Appendix J to implement initial-value problem (6.19), we may analyze the number of operations performed at each step. If results are reported after each iteration, we observe that to get 4 starting values (i = 0 to 3) we require 4 multiplications, 4 calls to procedure RATMAP (residue arithmetic only), and 4 calls to procedure OUTPUT. Each Milne-Simpson iteration (i = 4, 5, 6, ...) then necessitates 4 multiplications, 3 additions, one call to RATMAP (residue arithmetic only), and one call to OUTPUT. Consequently, for N iterations (N > 3), we have

(6.34) 4(N-2) multiplications,

3(N-3) additions,

N calls to RATMAP (residue only),

and

N calls to OUTPUT .

Calls to RATMAP and OUTPUT involve numerous divisions, of course.

Table 9 lists execution data for Euler and Runge-Kutta starting values and choices of step size h = 1/10 and 4/10. The residue arithmetic computations require twice to quadruple the execution times of comparable rational arithmetic computations and roughly 10% additional program memory. A large number of iterations (N = 50 and N = 75) is feasible because numerator and denominator digits grow roughly with N. Minimal K for a particular computation grows correspondingly.

For step size h = 4/10, we get the following results for approximation w_i (truncated floating-point equivalents are derived from the rational arithmetic solutions with K = 19 and K = 20, respectively). For convenience, t_i and w_i are denoted respectively t(i) and w(i) in the output.

Solution using Euler Method starting values (h = 4/10):

```
t(1)  =  0.4
w(1)  =  3 / 5
     Truncated floating-point equivalent (9K digits):
         +0.60000000000000000000000000000000000000000000000
         00000000000000000000000000000000000000000000000000
         00000000000000000000000000000000000000000000000000
         000000000000000000000...E+ 0

t(2)  =  0.8
w(2)  =  9/25
```

Truncated floating-point equivalent (9K digits):
+0.3600000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000
0000000000000000000000000...E+ 0

TABLE 9

MILNE-SIMPSON METHOD: INITIAL-VALUE
PROBLEM Y' = -Y, Y(0) = 1 #

	Euler Start Values				Runge-Kutta Start Values			
Step size --->	h = 1/10		h = 4/10		h = 1/10		h = 4/10	
Iterations -->	50	75	50	75	50	75	50	75
Num. Digits	119	181	109	166	129	192	114	172
Den. Digits	120	183	112	171	132	195	119	176
Terms to oscill. ##	28	28	8	8	---	>75	23	23
Residue arithmetic:								
Minimal Mersenne K	41	41	41	41	41	69	41	41
Exec. (sec)	200.4	346.8	195.6	336.1	210.6	---	200.9	350.2
Main pgm bytes	8280	7800	7808	7800	7800	---	7808	7800
Assembler bytes	30136	30136	30136	30136	30136	---	30136	30136
Rational arithmetic:								
Minimal K	14	20	13	19	15	22	14	20
Exec. (sec)	51.4	156.8	46.0	141.6	63.5	187.8	53.9	155.6
Main pgm bytes	8672	8672	8672	8672	8672	7984	8672	8672
Assembler bytes	25776	27528	25448	27240	26072	28112	25776	27528
Res/Rat exec. time	3.90	2.21	4.25	2.37	3.32	---	3.79	2.25
Tot. Res/Rat memory	1.12	1.05	1.11	1.06	1.09	---	1.10	1.05

\# Results reported at each iteration.
Point at which w_i ceased to be monotone decreasing.

```
t(3)  =  1.2
w(3)  =  27 / 125
     Truncated floating-point equivalent (9K digits):
         +0.216000000000000000000000000000000000000000000000
         000000000000000000000000000000000000000000000000000
         000000000000000000000000000000000000000000000000000
         000000000000000000000...E+ 0

t(4)  =  1.6
w(4)  =  481 / 3125
     Truncated floating-point equivalent (9K digits):
         +0.153920000000000000000000000000000000000000000000
         000000000000000000000000000000000000000000000000000
         000000000000000000000000000000000000000000000000000
         000000000000000000000...E+ 0

t(5)  =  2.0
w(5)  =  58247 / 703125
     Truncated floating-point equivalent (9K digits):
         +0.828401777777777777777777777777777777777777777777
         777777777777777777777777777777777777777777777777777
         777777777777777777777777777777777777777777777777777
         77777777777777777777...E- 1

t(6)  =  2.4
w(6)  =  11512789 / 158203125
     Truncated floating-point equivalent (9K digits):
         +0.727721971358024691358024691358024691358024691358
         024691358024691358024691358024691358024691358024691358
         913580246913580246913580246913580246913580246913580246
         024691358024691358024691...E- 1

t(7)  =  2.8
w(7)  =  1086836243 / 35595703125
     Truncated floating-point equivalent (9K digits):
         +0.305327932189849108367626886145404663923182441700
         960219478737997256515775034293552812071330589849108
         367626886145404663923182441700960219478737997256515
         77503429355281207133058...E- 1

t(8)  =  3.2
w(8)  =  298029408841 / 8009033203125
     Truncated floating-point equivalent (9K digits):
         +0.372116585463415942691662856271909769852156683432
         403597012650510592897424173144337753391251333638164
         913885078494131992074378905654625819234872732815119
         64639536656599756134 73...E- 1

t(9)  =  3.6
w(9)  =  16292367857567 / 1802032470703125
     Truncated floating-point equivalent (9K digits):
         +0.904110670725593076936103913698792528239258920557
         503090653525038527324764178902267608257548815390 6
```

```
        0780030144456299005910345645142170062151772256939 1
        5222950430997984724550 79...E- 2

t(10) =  4.0
w(10) =  8395944917171629 / 405457305908203125
     Truncated floating-point equivalent (9K digits):
         +0.207073464821780735074636695324598591377029623 23
         1929790889299094349137533611445108676221819543467 6
         6621318264868536676696002002103714241091673393659 8
         7945980833243953703233 28...E- 1

t(11) =  4.4
w(11) =  28747282662171323 / 91227893829345703125
     Truncated floating-point equivalent (9K digits):
         +0.315115053691221469196726970214525747501185978 83
         4485441992201914752113174891997000189631168422793 4
         5736926164480681350851322682626627937584374893502 3
         4132815830501933004374 47...E- 3

t(12) =  4.8
w(12) =  260168319541689352801 / 20526276111602783203125
     Truncated floating-point equivalent (9K digits):
         +0.126748913503421761115166646955525369748836386 39
         9791054933080733373396779745430552073873408383416 7
         0669875180504457006144556283392642148051183735922 6
         5018280951062451423419 68...E- 1

t(13) =  5.2
w(13) = -13702992123459647313913 /
         46184121251106262207031 25
     Truncated floating-point equivalent (9K digits):
         -0.296703536892161078075220495382731293801035907 52
         5272882508802986927209211178296767882093091880847 8
         4023903289224334234940269253544822199633329150023 3
         4840375466048208926812 85...E- 2

t(14) =  5.6
w(14) =  88616611721539625936008 69 /
         10391427281498908996582031 25
     Truncated floating-point equivalent (9K digits):
         +0.852785756190723624203583197117234091928014220 80
         2406302838108859601956215129918495927101630967747 3
         5169523319202207248832991437946954380676088151399 2
         9840474240437557813887 90...E- 2

t(15) =  6.0
w(15) = -92006499876156310904850879 7 /
         23380711383372545242309570 3125
     Truncated floating-point equivalent (9K digits):
         -0.393514544393152056693851750687559824901895969 62
         7581863849867046087608736615494059686262265752926 9
         5369099829302070524612921779049286914089937797167 0
         0706388821034025906846 49...E- 2
```

```
t(16) =  6.4
w(16) =  326765883416127109877537994361 /
         52606600612588226795196533203125
      Truncated floating-point equivalent (9K digits):
          +0.62114996903627211950463687291371179875643255951
          16083711404616459145333451070410036842367549967653
          55471022209300511393137203680782894567880562296605
          72588035780312845146867O...E- 2

t(17) =  6.8
w(17) = -46772555127163244128967419554193 /
         11836485137832351028919219970703125
      Truncated floating-point equivalent (9K digits):
          -0.39515577962976967265704991564553963196567960670
          63266761040144576670312133514966671196610447204263
          35692089777587309562420292765336205495502805269224
          29560527740284246967517 9...E- 2

t(18) =  7.2
w(18) =  12757779729109165691976344438924509 /
         26632091560122789815068244934082031 25
      Truncated floating-point equivalent (9K digits):
          +0.47903784426048829353262062619041316638080060395
          81326097909195361357246645488830292366004873186330
          36005546376689555725948945873466071486858937663426
          36072880670928255022549O...E- 2

t(19) =  7.6
w(19) = -21626850814784274265162994668308161 17 /
         59922206010276277083903551101684570312 5
      Truncated floating-point equivalent (9K digits):
          -0.36091546447865098538452718911752169091601947446
          91076484099054357797734601259087825722521144733854
          27873642185993652415966544846046028329508574777471
          92582285214543768144072 6...E- 2

t(20) =  8.0
w(20) =  51670229723036412046038946917045578952 1 /
         13482496352312162343878298997879028320312 5
      Truncated floating-point equivalent (9K digits):
          +0.38323933767781288064894569441047663732079243405
          26066908582398368628168011772040517393485209899405
          67518203442813611732259341239503252783342212376255
          55025372346122346614880 3...E- 2

t(21) =  8.4
w(21) = -95921281579298543952760970921769134791273 /
         30335616792702365273726172745227813720703125
      Truncated floating-point equivalent (9K digits):
          -0.31620020200932153025858769090088407322240538932
          77760625087480878317920097778776519303766728979631
          74702416341683451310194926691588999393207019604318
          18103616916372753022894 9...E- 2
```

```
t(22) =  8.8
w(22) =  213881354261296116746237047244511341152065 49 /
         682551377835803218658838886767625808715820312 5
     Truncated floating-point equivalent (9K digits):
         +0.313355684577854755380593365605689010770030571 79
         780208054022312221607739382523194094136200084227 90
         941664360581302074056985680864598184163130276344 34
         893237350737931292390735...E- 2

t(23) =  9.2
w(23) = -41692977689060084377636364711387203818515826 37 /
         153574060013055724198238749522715806961059570312 5
     Truncated floating-point equivalent (9K digits):
         -0.271484505166534358433232599777324350812565835 04
         054157651930979832845439540183429940244695296433 91
         447181649890086706147336471304773776580773106739 50
         620566917467212890139480...E- 2

t(24) =  9.6
w(24) =  896346018660946722226623230426119132405641554281
         /
         345541635029375379446037186426110565662384033203125
         Truncated floating-point equivalent (9K digits):
         +0.259403188442037108300033352367467939273780090 46
         803198410832029872479996418608768605318498155198 56
         176665206543101837592114966384105385171019746099 80
         676438246231682003193001...E- 2

t(25) =  10.0
w(25) = -17937003745484690615762536346682631323467901763315
         3 /
         777468678816094603753583669458748772740364074707 03
         125
     Truncated floating-point equivalent (9K digits):
         -0.230710306848612967155622008631420741823239077 58
         258137042639581174536665213137913021192621663688 71
         777991418066719023758210336045386391963670440172 62
         215970504873304428233685...E- 2

t(26) =  10.4
w(26) =  378219099093206950709333724084394006177586653623 50
         989 /
         174930452733621285844556325628218473866581916809 08
         203125
     Truncated floating-point equivalent (9K digits):
         +0.216211124582834850507197611876874386050438449 53
         694885728038914529928265115224623756911313885210 78
         744562547320731849783496671320258489168608380360 27
         330811504992147186666338...E- 2

t(27) =  10.8
w(27) = -76756485583150965508913260019068494199064751648895
```

```
          60357 /
          39359351865064789315025173266349156619980931282043
          45703125
     Truncated floating-point equivalent (9K digits):
          -0.19501460757355542109729637387160246301182418000
          15124175997935625764901036306875479032952044373092
          60761436396482332465022037406609922482417209415655
          29558711736977881899603l...E- 2

t(28) =  11.2
w(28) =  16018567929884721147490462660792655841150797285362
          99064641 /
          88558541696395775958806639849285602394957095384597
          7783203125
     Truncated floating-point equivalent (9K digits):
          +0.18088111686392705377497577146414235525116425382
          47261103336765136786977416976258577421512954047580
          72244923744394395463306274372963322332328887956332
          20898987635217191415847l...E- 2

t(29) =  11.6
w(29) = -32753454442950306745660112066792234856242829493328
          3757167833 /
          19925671881689049590731493966089260538865346461534
          5001220703125
     Truncated floating-point equivalent (9K digits):
          -0.16437816821147954167873304432381673570181235486
          30525151731655729021292685265373024330211051001022
          25410115550446868095235895485524276214964731057283
          51660497698770530767067l...E- 2

t(30) =  12.0
w(30) =  67978815806110846676484275509334090583630317324516
          450658501829 /
          44832761733800361579145861423700836212447029538452
          625274658203125
     Truncated floating-point equivalent (9K digits):
          +0.15162754462850810356942114965502152859478382851
          74121422371626577886582601931596555109871040831269
          25288346275716848853695052020063457855380406377078
          33549907971715334178455l...E- 2

t(31) =  12.4
w(31) = -13955668532407928823951876121459456113236172532514
          542039920621277 /
          10087371390105081355307818820332688147800581646151
          840686798095703125
     Truncated floating-point equivalent (9K digits):
          -0.13834792031249432320392960089157538199365958625
          59202696630910938660890449777936574916886461607196
          96173797795902139819537457633516914420644986326063
          79307075702909279256506l...E- 2

t(32) =  12.8
```

```
w(32) =  28879551744881876616615748256080543952275354743284
         96617176994656601 /
         22696585627736433049442592345748548332551308703841
         64154529571533203125
     Truncated floating-point equivalent (9K digits):
         +0.12724183372141019570623760046409369348406855966
         56813039789001700495848697281931146027292367452312
         86708607908651433804373216429400759627916060025189
         32019336148337282957752 9...E- 2

t(33) =  13.2
w(33) = -59415422878752102466254045491692556494731156251298
         1174007910155763313 /
         51067317662406974361245832777934233748240444583643
         693476915359497070 3125
     Truncated floating-point equivalent (9K digits):
         -0.11634725612872860227141706347447576629944731569
         75151956135659934670776231339059143409343111674935
         57661941129295395539141019702129047069915337327986
         96401138425850823424330 2...E- 2

t(34) =  13.6
w(34) =  12276014779956832266119158579558426281308488672881
         188059136822515444306 9 /
         11490146474041569231280312375035202593354100031319
         83103230595588684082031 25
     Truncated floating-point equivalent (9K digits):
         +0.10683949771824658036307933489175813549492895264
         81176787994806361182396678757604500925649020466622
         01595020965716566749294067865669542748566545225001
         55424167512811951774297 7...E- 2

t(35) =  14.0
w(35) = -25285053581799785722767119985925570616840912411338
         39409267812919126495739 7 /
         25852829566593530770380702843829205835046725070469
         61982268840074539184570 3125
     Truncated floating-point equivalent (9K digits):
         -0.97803814923502946310298420403589415288243714870
         71184858651142467307821392622261725837759163741612
         66753774229912159960802393071719442074490571152081
         39079977400919009633334 4...E- 3

t(36) =  14.4
w(36) =  52198513849080759013212658672749346459951899860614
         56715679265041779570826161 /
         58168866524835444233356581398615713128855131408556
         64460104890167713165283203125
Truncated floating-point equivalent (9K digits):
         +0.89736171542545671184111057315709064323792704623
         75458578541743263983001485839079194806475764768187
         01622293897310968195332892606036753207740267829436
         97751700970848080624954 5...E- 3
```

t(37) = 14.8
w(37) = -10757973750751106664431842886858569417868078266369
039809488549035648560058675 93 /
130879949680879749525052308146885354539924045669 25
245035236002877354621887207031 25
Truncated floating-point equivalent (9K digits):
-0.821972638053568791732628430011305318767477200 58
144901582719639684400193485736561544396361669224 41
808119928991098747229178456956268738359958228214 06
384061800370847209968379...E- 3

t(38) = 15.2
w(38) = 221988452475383104942452403623069169537254236189 44
333629651145109203547007199870 9 /
294479886781979436431367693330492047714829102755 81
801329281006474047899246215820 3125
Truncated floating-point equivalent (9K digits):
+0.753832307195003947394714664175280176224045601 66
500056338294506940760430242154808531401962009020 53
977209729674141028402109389319993722257754600239 37
377755057172100801734781...E- 3

t(39) = 15.6
w(39) = -45766186951777066838342781257977077539006244686990
719691050375109913220103766280717 /
662579745259453731970577309993607107358365481200 59
052990882264566607773303985595703125
Truncated floating-point equivalent (9K digits):
-0.690727226107038499639423622032642588431145119 59
904178392657917909456202675561261364981872896607 44
307122920820524126411828117777262035894626091645 94
697710976740353033097843...E- 3

t(40) = 16.0
w(40) = 944149653714420059178885043836714880728966638778 37
908409562629076334124785234758293 21 /
149080442683377089693379894748561599155632233270 13
286922948509527486748993396759033203125
Truncated floating-point equivalent (9K digits):
+0.633315568910431963827759007427436488125990450 74
230237150662498085922472051110257775007687607424 41
833781478875417408170155007325628043791219670471 15
601783259039269224518496...E- 3

t(41) = 16.4
w(41) = -19468434474774424159362695787225936614484045472580
470925209899310768446380175483748086 73 /
335430996037598451810104763184263598100172524857 79
895576634146436845185235142707824707031 25
Truncated floating-point equivalent (9K digits):
-0.580400580290803174149732883748930262586970492 45
701155338831349887633325542676078885314124993691 96
137037616660467000981991198043810271180971143786 68

```
          60549855907264996767812l...E- 3

t(42) =   16.8
w(42) =   40157965705529644276651871139433162703005220151091
          40142867854724337821033613100624064327 49 /
          75471974108459651657273571716459309572538818093004
          76504742682948290166677907109260559082031 25
     Truncated floating-point equivalent (9K digits):
          +0.53209109977458955751129937897384011718083414285
          31990542698603273851752778676278428450017905871915
          16071986501484359667391640129186900838308048323428
          34819410486919140992582 6...E- 3

t(43) =   17.2
w(43) =  -82813741080418905747924018892599507360387537148935
          92875174472476833595649441013170492302323 7 /
          16981194174403421622886553636203344653821234070926
          07213567103663365287502529099583625793457031 25
     Truncated floating-point equivalent (9K digits):
          -0.48767913628387853956500237618710756463117810981
          84962463388731126596756680654123132336685306978818
          55457060691066999479680561775906023467087602247990
          37705302093789325382368 5...E- 3

t(44) =   17.6
w(44) =   17081007123263226428446751810689087063312024632449
          89508923679402807179490040098198021146928208 1 /
          38207686892407698651494745681457525471097776659583
          66230525983242571896880690474063158035278320 3125
     Truncated floating-point equivalent (9K digits):
          +0.44705682318228525239386033527997326051711207090
          60304868377824494496819010625433104800681108280539
          17854409398399332031209582949720481404480368783048
          72923537312274219116423 3...E- 3

t(45) =   18.0
w(45) =  -35226200951137728418484715876771562819743096684979
          68815287357171162156087954773058871811777594553 /
          85967295507917321965863177783279432309969997484063
          24018683462295786767981553566642105579376220703125
     Truncated floating-point equivalent (9K digits):
          -0.40976281437041955434640696978888493052040416245
          43501982137698101112815347673142255752816264484083
          55433713212138816059141145685320750772596365678172
          44603571313195813826187 6...E- 3

t(46) =   18.4
w(46) =   72654257127479524587510020237579977169062758198340
          69275669103832599255476997274921838128731728491 89
          /
          19342641489281397442319215001237872269743249433914
          22904203779016552022795849552494473755359649658203
          125
```

Truncated floating-point equivalent (9K digits):
+0.375617038488463026531403359774819741932159569726108547003711343620537580602652778301020125625482763188091145880095437419595500460410947975206534250467878909904421519845823...E- 3

t(47) = 18.8
w(47) = -1498390309783783580798238622384960402169176404986137242138875498030231742679807012567535075980915369 57 /
435209433508831442452182337527852126069223112263070153445850278724205129066149311256594955921173095703125
Truncated floating-point equivalent (9K digits):
-0.344291781017512721329852183279209638123108700674126109544960722161540493068528602382635318763881767769483397020034949733626551309753078935856724042837184951796200234503743...E- 3

t(48) = 19.2
w(48) = 309037918272027563787538480853270826972142406479230246702887657418966892430806960418746490372292637604441 /
979221225394870745517410259437667283655752002591907845253163127129461540398835950327338650822639465332031 25
Truncated floating-point equivalent (9K digits):
+0.315595608282906746289988268607198517193439259175895140469541185116623662993222265745526467408854778954985552488347040203833691892414950117433760682770099540520231762352998...E- 3

t(49) = 19.6
w(49) = -637355635235570307137717495496386241872148583325373440994236104938951958407967035125158784837440751771323 3 /
220324775713845917741417308373475138822544200583179265181961703604128846589738088823651196435093879699707031 25
Truncated floating-point equivalent (9K digits):
-0.289280056303498521387401741502005400353857249436806073419269938924368808801626060174253148343317146194426416236592265638200404429433969896688644900281207419672888907912494...E- 3

t(50) = 20.0
w(50) = 131451072538511100541980607356639923509398107378278187741480482439066665709694103399101323306791361083659202 9 /
495730745356153314918188943840319062350724451312153346659413831092899048269106998532151919789612293243408203125

```
     Truncated floating-point equivalent (9K digits):
         +0.265166269733928353731119478554158707248270221 46
         475054528619352597082074512914391217114924778305 87
         325946032686364011469270397857013690598076694787 40
         247856821909887361146630...E- 3

t(51) =  20.4
w(51) = -271104976797804514792987611462471748889916281613 40
         800003584922914941710516766227654244869598840891 08
         43081293877 /
         111539417705134495856592512364071789028913001545 23
         450299836811244959022858605490746697341819526627 65
         979766845703125
     Truncated floating-point equivalent (9K digits):
         -0.243057550752593479713369936656970183143577376 99
         659514375937871006362697928451892651962907039272 50
         296777038958050007231123817726877301649011048866 80
         703950198217705683852585...E- 3

t(52) =  20.8
w(52) =  559135910188239621631322844691349219369552346823 02
         948285379132399108023865840532910136618168732977 17
         6301262400401 /
         250963689836552615677333152819161525315054253476 77
         763174632825301157801431862354180069019093934912 23
         45447540283203125
     Truncated floating-point equivalent (9K digits):
         +0.222795540881787762042446388003234028386438084 71
         393809641578264015687811459424766184187190568211 34
         386631142934334866898990768619491154696740134100 67
         174277177421726844002153...E- 3

t(53) =  21.2
w(53) = -115316778652826552659409622416157986952437068174 12
         385453747716349755325130952295755931948657149958 75
         722076227593271 3 /
         564668302132243385273999593843113431958872070322 74
         967142923856927605053221690296905155292961353552 52
         7725696563720703125
     Truncated floating-point equivalent (9K digits):
         -0.204220386052092857632936112209284518391762388 29
         884837140347275749509855674685020041952871520429 49
         737146798137234637932329947372768845859460660983 78
         048781494234569852834379...E- 3

t(54) =  21.6
w(54) =  237832450607537202417501814698587704437338255549 16
         102073925275088457404406190712825198803378362684 86
         642102044967645269 /
         127050367979754761686649908614700522190746215822 61
         867607157867808711136974880316803659940916304549 31
         8738281726837158203125
     Truncated floating-point equivalent (9K digits):
         +0.187195404774770395175710484432658318318244109 45
```

```
          0044511035754536651062976956063227079337734973410 2
          2346695404181069721666235948832092922472816345032 5
          7890056677816357751472 41...E- 3

t(55) =  22.0
w(55) = -4905092113182720987194833150213933808769543974116 3
          2981165667148504863503354408227555247490130669685 2
          4380394963966208599 7 /
          2858633279544482137949622943830761749291789856008 9
          2021161052025696000581934807128082348670616852359 6
          7161133885383605957031 25
     Truncated floating-point equivalent (9K digits):
          -0.1715887150787084600666609075543256284509227071 8
          3795120241251631244387887041192252815771500469420 0
          8458875052476364345904378568969689116748365617562 5
          6420844915944245437629 53...E- 3

t(56) =  22.4
w(56) =  1011637895054719006097061684268840224873712427488 4
          7028400847053601078862798573106198354455423424990 3
          6980357553146415249796 1 /
          6431924878975084810386651623619213935906527176020 0
          7047612367057816001309353316038185284508887917809 2
          6112551242113113403320 3125
     Truncated floating-point equivalent (9K digits):
          +0.1572838480066206262968488415441438615222993549 1
          8891765685535991846652931915704887228036757975771 3
          5611391838587182215154661045178417325833070437161 4
          4886494518002849910743 61...E- 3

t(57) =  22.8
w(57) = -2086419571695577599625954074493314287143281753874 3
          0096879730381122052655004777244172132080147939033 0
          8307681046679768827100993 /
          1447183097769394082336996615314323135578968614604 5
          1585712782588008600294604496108591689014499781507 0
          8375324029475450515747070 3125
     Truncated floating-point equivalent (9K digits):
          -0.1441710848414044048508630295110459803055147762 9
          9533006258946139455839634773137395782069619158293 7
          6940901439173229468344335116652521996863552545234 3
          5476562202567894762594 19...E- 3

t(58) =  23.2
w(58) =  4303077845778944750915135566196883130087331801177 9
          5481633994639741625532162763200739039670427363271 6
          2602217300424982875590329 09 /
          3256161969981136685258242384457227055052679382860 1
          6067853760823019350662860116244331300282624508390 9
          3844479066319763660430908203125
     Truncated floating-point equivalent (9K digits):
          +0.1321518365931862095849218918162472347528890441 3
          9812228576468953392499826573098741067359493774974 1
```

```
          11295715642207735955913716652460702117243557427535
          25086722440167923848136 8...E- 3

t(59) =  23.6
w(59) = -88747479937491074360786250142098915879355375113289
          60952289518452048034533143943490023879167178658688
          69503443121867333320703225317 /
          73263644324575575418310453650287608738685286114353
          61526709618517935389914352615497454256359051438796
          11500778992194682359695434570 3125
     Truncated floating-point equivalent (9K digits):
          -0.12113440541439404919234941602784273204755609008
          44510850040263385987144396901598181473234489432223
          13710793993526906221493163640340248361831555041920
          72981094207942770315345 9...E- 3

t(60) =  24.0
w(60) =  18303468182719781213018549930471054802702246212129
          07404273708276944461892124957646202540303876031223
          20506974300422202502345715109121 /
          16484319973029504469119852071314711966204189375729
          56343509664166535462730729338486927207680786573729
          12587675273243803530931472778320 3125
     Truncated floating-point equivalent (9K digits):
          +0.11103562787343754711330971135502501433558498373
          02501121097329077153683705738097571829088104017128
          36551008707884286174148302801147001324806736344864
          02641828208295292708060 9...E- 3

t(61) =  24.4
w(61) = -37749426555302565591058651627373293510449443357325
          35074867191862151524704165872344998650943269248348
          64465403248314111899661945649460 73 /
          37089719939316385055519667160453101923959426095391
          51772896744374704791144141011595586217281769790890
          53322269364798557944595813751220 7031 25
     Truncated floating-point equivalent (9K digits):
          -0.10177867780362199155181514953233226389682722663
          07501003518303102039427458733359310793301346695090
          62786611372588964470576833201099295584622140220446
          26890457629759998457056 2...E- 3

t(62) =  24.8
w(62) =  77855197898573884743098635936427235563527931866105
          44514928420876863294417642473363270238355066206160
          93757071636428334881764629584921 8949 /
          83451869863461866374919251111030729328908708714630
          91489017674843085780074317276090068988883982029503
          69975106070796755375340580940246582031 25
     Truncated floating-point equivalent (9K digits):
          +0.93293533177812712569508148551183699203920185885
          88434364969042818695130267683303136882930548173803
          50677132552889912036360657201659654999610562709440
```

```
          5116670368197915081905 24...E- 4

t(63) =  25.2
w(63) = -160570093236330527860100541750193434218213604710 65
          710816826034304269147267688214102830163675036656 29
          042168706551657449509857246384864743837 /
          187766707192789199343568314999819140990044594607 91
          955850289768396943005167213871202655224988959566 38
          332443988659292699594516307115554809570312 5
     Truncated floating-point equivalent (9K digits):
          -0.855157421871734764910899359759802593213851010 79
          824684740080622231574422656985473997263500387038 17
          397247157200334009749193923344207724620077260893 19
          462083368544321677734398...E- 4

t(64) =  25.6
w(64) =  331163037884249276698341391247254620273599537519 24
          104710854711950017144469876222975530871286313845 07
          839673688276680817914044572170378106498 81 /
          422475091183775698523028708749593067227600337867 81
          900663151978893121761626231210205974256225159024 36
          247998974483408574087661691009998321533203125
     Truncated floating-point equivalent (9K digits):
          +0.783864054461364948360022081371678315689670284 64
          472725314982774417136059501363519185572467083621 29
          991553931499399069220994964436899166091816215415 46
          445674608811222730755414...E- 4

t(65) =  26.0
w(65) = -682997234694822885311579829871561324123968679143 40
          258144987225664494627472507734765192787277756178 30
          655945485106667215899724528498385431487595 3 /
          950568955163495321676814594686584401262100760202 59
          276492091952509523963659020222963442076506607804 81
          557997692587669291697238804772496223449707031 25
     Truncated floating-point equivalent (9K digits):
          -0.718514139331795435053092338496150635785189726 70
          628517283768500089151425910496242656258230476167 19
          345722091449401069214166041072139257641474915235 92
          417278091010257142167414...E- 4

t(66) =  26.4
w(66) =  140862734149836969907283995591096864092026543069 99
          658728269097217537701406179074833468535412754226 34
          069531900858190662533082329019157083088850738 9 /
          213878014911786447377283283804481490283972671045 58
          337210720689314642891823279550166774467213986756 08
          350549480832225590631878731073811650276184082031 25
     Truncated floating-point equivalent (9K digits):
          +0.658612500251302212413843796780101696504570616 84
          555587249690008317750042335729836351070598724593 61
          373318439954053624704543589071291829845152239727 62
          259375769280442037256600...E- 4
```

```
t(67) =  26.8
w(67) = -29051811863984574518532365521768240260833343811376
         54781742467594199363837998979599514621009762824469
         83358939168914651765595475178490371321396059355 7
         /
         48122553355151950659888738856008335313893850985256
         25872412155095794650660237898787524255123147020118
         78873633187250757892172714491607621312141418457031
         25
     Truncated floating-point equivalent (9K digits):
         -0.60370470472706780565366264582565390755318309302
         33568230882406577610898835453269422448094215427172
         80928505138403875491461289167946490262514208979186
         895083366441005111157079...E- 4

t(68) =  27.2
w(68) =  59917043481896303712243382910523741014626830977164
         05063128718971576944345041547576564226979941394178
         81399673363973471666690896943228558628750100496241
         /
         10827574504909188898474966242601875445626116471682
         65821292734896553796398553527227192957402708079526
         72746567467131420525738860760611714795231819152832
         03125
     Truncated floating-point equivalent (9K digits):
         +0.55337456652669626909822548970341590099192032358
         87420377040424018581298284052325536901565641569857
         66325049067648090205829366517394985664894698250184
         62432205231707850657215 4...E- 4

t(69) =  27.6
w(69) = -12357411148465588123117662981553252813767086127528
         05036624323182938063890521434183247043084454494335
         68489953500019413395394102916802688996321372714778
         633 /
         24362042636045675021568674045854219752658762061285
         98097908653517246041896745436261184154156093178935
         13679776801045696182912436711376358289271593093872
         0703125
     Truncated floating-point equivalent (9K digits):
         -0.50724035472221722241694971131042166834491915816
         33059002031975840636669899275180753659315705210712
         27960095357894003752833410694882794964763844179797
         93929731488693761785881 4...E- 4

t(70) =  28.0
w(70) =  25486173807491492590396852389614331094856270744364
         46533657145660047267476629235099772339118352063128
         57105943957354741869432429619804719701748928004894
         42229 /
         54814595931102768798529516603171994443482214637893
         45720294470413803594267677231587664346851209652604
         05779497802352816411552982600596806150861084461212
```

```
        158203125
     Truncated floating-point equivalent (9K digits):
        +0.464952324733460051501961860810687009578078428260629553125218994606791312995505798602599653703154147213354179167091498899576565223223168797590524837413279423423476170547852...E- 4

t(71) =  28.4
w(71) = -525631964715744896202674579489197501749192668098155464444733847852386885309585854340208277785989545884465083824709104227021783474398200197382559314731384647 /
        123332840844981229796691412357136987497834982935260278706625584310580871022737710722447804152217183591300387005529383692599421085134281383943744003772735595703125
     Truncated floating-point equivalent (9K digits):
        -0.426189781338466900284682431247231152970108834900563141832066944357783003223568870142672124221530008041668944501669210355399326793909726199375382168927431119816226982488521...E- 4

t(72) =  28.8
w(72) =  10840739464181403198003990346014929445531789329099234549420391514513151899796710169737661908561513138252101937516134751272778030861946845795316375347846645842 01 /
        27749889190120776704255567780355822187012871160433562708990756469880695980115984912550755934248866308042587076244111330834869744155213311387342400848865509033203125
     Truncated floating-point equivalent (9K digits):
        +0.390658837947389321127173651474814870040224859382285318013388872854752492355697971111137678946557247886145518688913077994440925220986641337910116967510239020847714888970705...E- 4

t(73) =  29.2
w(73) = -223581588597029135177153262093253950476955219940322683883873449090537169646341009982253364554754758911503220341867350434578087761389473900349589447897958243822113 /
        624372506777717475845750275058005999207789601109755160952292020572315659552609660532392008520599491930958209215492504943784569243492299506215204019099473953247070312 5
     Truncated floating-point equivalent (9K digits):
        -0.358090060292527098175902958390148669281525264993789278520067860008595756402837872875810177550301509794708232178915995410393377697465222591049162278415588192380196329340319...E- 4

t(74) =  29.6
```

```
w(74) =  46111916704355364473442555592542954251694003386079
         65016633527457426755083300000392924206523642292758
         30303724068025404906967210440992493754156014630625
         72673601207469 /
         14048381402498643206529381188805134982175266024969
         49112142657046287710233993371736197882019171348856
         84465597073485813612351528079785767388898420904297
         3816394805908203125
     Truncated floating-point equivalent (9K digits):
         +0.32823650912662369229811044590263527756261616704
         10916502301327460626858986760012970636508558652297
         58043645958568861728261153188612585520378624485525
         14145940944166530467393 8...E- 4

t(75) =  30.0
w(75) = -95102143781041486623535690472969221714237245892407
         63684897164953597720063399467063682874294814068437
         74355392855746913012697789475463951761731817654893
         6853444623894597 /
         31608858155621947214691107674811553709894348556181
         35502320978354147348026485086406445234543135534927
         90047593415343080627790938179517976625021447034669
         108688831329345703125
     Truncated floating-point equivalent (9K digits):
         -0.30087181040459897871604586225181113687339015371
         66858739731261170354196872333630764734876784209264
         74611052857693291113150923581310510828125877198071
         20305836535692446151780 2...E- 4


Solution using Runge-Kutta starting values (h = 4/10):


t(1)  =  0.4
w(1)  =  419 / 625
     Truncated floating-point equivalent (9K digits):
         +0.67040000000000000000000000000000000000000000000
         00000000000000000000000000000000000000000000000000
         00000000000000000000000000000000000000000000000000
         00000000000000000000000000000000...E+ 0

t(2)  =  0.8
w(2)  =  175561 / 390625
     Truncated floating-point equivalent (9K digits):
         +0.44943616000000000000000000000000000000000000000
         00000000000000000000000000000000000000000000000000
         00000000000000000000000000000000000000000000000000
         00000000000000000000000000000000...E+ 0

t(3)  =  1.2
w(3)  =  73560059 / 244140625
     Truncated floating-point equivalent (9K digits):
         +0.30130200166400000000000000000000000000000000000
```

```
        0000000000000000000000000000000000000000000000000000
        0000000000000000000000000000000000000000000000000000
        0000000000000000000000000000000...E+ 0

t(4)  =  1.6
w(4)  =  11080882933 / 54931640625
     Truncated floating-point equivalent (9K digits):
        +0.201721317749191111111111111111111111111111111111
        1111111111111111111111111111111111111111111111111111
        1111111111111111111111111111111111111111111111111111
        1111111111111111111111111111111...E+ 0

t(5)  =  2.0
w(5)  =  1672753021871 / 12359619140625
     Truncated floating-point equivalent (9K digits):
        +0.135340175359676365432098765432098765432098765432
        0987654320987654320987654320987654320987654320987
        6543209876543209876543209876543209876543209876543
        2098765432098765432098765432098765...E+ 0

t(6)  =  2.4
w(6)  =  251601796813177 / 2780914306640625
     Truncated floating-point equivalent (9K digits):
        +0.904744875497852832482853223593964334705075445816
        1865569272976680384087791495198902606310013717421
        1248285322359396433470507544581618655692729766803
        8408779149519890260631001371742112...E- 1

t(7)  =  2.8
w(7)  =  38043348776348699 / 625705718994140625
     Truncated floating-point equivalent (9K digits):
        +0.608007049024670224091205608901082152110958695320
        8352385307117817405883249504648681603414113702179
        5458009449778997104099984758420972412741960067062947
        7213839353757049230300259106843...E- 1

t(8)  =  3.2
w(8)  =  5708762105001944413 / 140783786773681640625
     Truncated floating-point equivalent (9K digits):
        +0.405498547512372359552635133533167369472810716523
        5651746854307439584074243424952158376941184440041
        3216142525698995749292960083998035529814222086741519
        7547799285339294484241900794255...E- 1

t(9)  =  3.6
865808645872138702631 / 3167635202407836914062 5
     Truncated floating-point equivalent (9K digits):
        +0.273329657788247037695624825634454247978609102421
        5293889632151076036662583428828411808648561176123
        0315312518228740349352044723685225641228283101981217
        1710594402765311671474349927837 70...E- 1

t(10) =  4.0
```

```
w(10) =  1293824064551371698132 97 /
         7127179205417633056640625
      Truncated floating-point equivalent (9K digits):
          +0.181533819658679001062843053438314151732950809 42
          402093762263564743990604998694325637485837771476 25
          251177668096949823425575202348164915975830075419 68
          179179083117620038933975946739357...E- 1

t(11) =  4.4
w(11) =  197328604648835582675471 39 /
         160361532121896743774414062 5
      Truncated floating-point equivalent (9K digits):
          +0.123052331839059006386376938186775805757655734 10
          341754021171554507020406966653743498836996202335 59
          717591497851834057232837781070479212574634894955 18
          137142199518518495287945393099743...E- 1

t(12) =  4.8
w(12) =  29262036548151278925687646 93 /
         360813447274267673492431640625
      Truncated floating-point equivalent (9K digits):
          +0.811001828485292579907941174192421204651362669 67
          026620009247884667245877127577524687910865757025 04
          985481033761226810077151082218576209581935662674 77
          402311574698363462134807227503138...E- 2

t(13) =  5.2
w(13) =  45097148385015637248726383899 1 /
         811830256367102265357971191406 25
      Truncated floating-point equivalent (9K digits):
          +0.555499724620057970805058779712532802034868344 01
          494157653827360919181765032795900706451508528473 53
          896373015235363271891613581874573724181347987966 9
          422669302574983040943115656665460...E- 2

t(14) =  5.6
w(14) =  65922201366135474740616561140617 /
         182661807682598009705543518066406 25
      Truncated floating-point equivalent (9K digits):
          +0.360897563658655342655395563351175650560400425 26
          022276738730629393770512945205501712668442992472 56
          816872260382442588285260345556740120081520093708 57
          315336231800215373027049145362719...E- 2

t(15) =  6.0
w(15) =  10359084300661204883973318632491979 /
         410989067285845521837472915649414062 5
      Truncated floating-point equivalent (9K digits):
          +0.252052551399290521162541102718954005050991472 68
          532531115427858556122119021238843627120203941259 11
          074773131281262749564948538028472182202306490921 29
          860620121321311527574967071561668...E- 2

t(16) =  6.4
```

```
w(16) =  14740523902313165542793148195083817 73 /
         92472540139315242413431406021118164062 5
      Truncated floating-point equivalent (9K digits):
          +0.15940433646686585840682593122498178902844148197
          17395218986742274641162449131070955266753621787639
          87004268680178187439805881182924445487615193446227
          24962971225516307018391190626978 0...E- 2

t(17) =  6.8
w(17) =  24018406433499301609436234287447061695 1 /
         20806321531345929543022066354751586914062 5
      Truncated floating-point equivalent (9K digits):
          +0.11543802395495128706273364477669336100671727326
          68401512434583463505262139035159823214052550318743
          88919713749318440841545127318978435867679866967558
          21553007723741250652580871559751 6...E- 2

t(18) =  7.2
w(18) =  32486126674073933960312228865113452127137 /
         46814223445528341471799649298191070556640625
      Truncated floating-point equivalent (9K digits):
          +0.69393710464670719070403241490639924943642753749
          25852464780479180914855652965130267497404333624423
          30554956567314924063170480564189511499116081046755
          14652571666921763023273494 1280044...E- 3

t(19) =  7.6
w(19) =  56627055775814992461801997502960303338672 19 /
         10533200275243876831154921092092990875244140625
      Truncated floating-point equivalent (9K digits):
          +0.53760542186694439734115706139922645877965390343
          84685453200640013135201531096808963933944553092772
          20780548767001925700427561995575023471966284066526
          73155576317038204180923828722106 5...E- 3

t(20) =  8.0
w(20) =  69544750496948673887794982399415618050840365 3 /
         23699700619298722870098572457209229469299316406 25
      Truncated floating-point equivalent (9K digits):
          +0.29344147259108504225789763478293722706723586027
          75756398209304347365318119538528826241417709388061
          78194118799029541257878075344778161440807755134512
          32414180961875296518014134366197 9...E- 3

t(21) =  8.4
w(21) =  13741821367730642741585838933903516412150013251 1
         /
         53324326393422126457721788028720766305923461914062
         5
      Truncated floating-point equivalent (9K digits):
          +0.25770267150389692172924918210471034586485240991
          45328703213760331203332527694553286358366622923622
          96624356712610179498849689552124376010482830663555
```

```
        113128009436619274478693860497879...E- 3

t(22) =  8.8
w(22) =  139888098953189394540253700375086273990787435248 57
         /
         119979734385199784529874023064621724188327789306640
         0625
     Truncated floating-point equivalent (9K digits):
         +0.116593106052454659759048009415620099585708025340
         426925775807141067794642682951997756384071237929780
         886317010570720266212443127691602720671623899084930
         587022808945377693529039189974905...E- 3

t(23) =  9.2
w(23) =  34950755715268402636711026487779434206469135818968
         59 /
         269954402366699515192216551895398879423737525939940
         140625
     Truncated floating-point equivalent (9K digits):
         +0.129469108148835237167810627729981247329014246970
         837356986437279849352791143985604956607770165208050
         243331799870834281638394064769324881151688711961660
         764523428954001096589413896408356...E- 3

t(24) =  9.6
w(24) =  240802722308052959170555919479434856049692154125918
         8333 /
         607397405325073909182487241764647478703409433364860
         81640625
     Truncated floating-point equivalent (9K digits):
         +0.396450034519290375720057921289789436290963738280
         733802872382650051896284114594345154378532603122860
         253293556082306930227427564904117748888188913580470
         586438358519099487544158015450909...E- 4

t(25) =  10.0
w(25) =  952770209055094573843529787441569578310452260515900
         241671 /
         136664416198141629566059629397045682708267122507090
         53369140625
     Truncated floating-point equivalent (9K digits):
         +0.697160413485928604760606821976667957481340170030
         470775518089163883817383204937377429824002906730060
         884544275903779115396197699690972968634814781034740
         376885592486739055897108230461705...E- 4

t(26) =  10.4
w(26) =  219573559572946306636303190106379361068843222866960
         3205777 /
         307494936445818666523634166143352786093601025640960
         4508056640625
     Truncated floating-point equivalent (9K digits):
         +0.714072114848095022264087361148155471435410027560
```

```
        3466209547748963985780220296752368003782294068616202092102429561193850654662061042673680779257342669393208383011236186727086681923781...E- 5

t(27) =  10.8
w(27) =  28398248392265589938198103929219042524310782322250423142448 99 /
        691863607003091999678176873822543768710602307692170143127441406 25
     Truncated floating-point equivalent (9K digits):
        +0.41046021361459868857299252230221637334726598989038636219204453750002972214639937929124222645608070829118853615938394347610694642075987583942346895073871682698239229860876843566061 4...E- 4

t(28) =  11.2
w(28) =  89409366130712717637283784805862144572895751015275418388706187 /
        15566931157569569992758979661007234795988551923073828220367431640625
     Truncated floating-point equivalent (9K digits):
        -0.57435447761479020511815544593259406067518196997677544079651001855130987619420536707251074289289373517844191323146837838351182638234476538133334507246228722286874352767884656979782 4...E- 5

t(29) =  11.6
w(29) =  93241116796879552905122578862339022704564276262835720321766160431 /
        35025595104531532483707704237266278290974241826916113495826721191406 25
     Truncated floating-point equivalent (9K digits):
        +0.26620851556870828883569913027856718821002364672594063078449470175718279519192395765349046907598427931891083054322276331333274223954800704294139898280288950507611977162291924562567 0...E- 4

t(30) =  12.0
w(30) = -79559882825960072457130408438500902046391236477058832853179366381 03 /
        78807588985195948088342334533849126154692044110561255365610122680664 0625
     Truncated floating-point equivalent (9K digits):
        -0.10095459568101422720716892140658172762560691813628929305684837679403076583542693915909644094258080169207349984991192224607425291994622638035969337274786624187398312912430629919587 4...E- 4

t(31) =  12.4
w(31) =  33401483804515673221347963938477334344660454761709290089565324324153 39 /
        17731707521669088319877025270116053384805709924876282457262277603149414 0625
```

```
     Truncated floating-point equivalent (9K digits):
         +0.188371502088545539476680605811519673449195909152
         244248188052018039131264206514898005602657598492717
         793141370508751418074472473283590580518122453007197
         864393896759485361051582832013601...E- 4

t(32) =  12.8
w(32) = -432757019517867753982540656548545503727689359351351
         1188305577879915021907 /
         398963419237554487197233068577611201158128473309717
         635528840124607086181640625
     Truncated floating-point equivalent (9K digits):
         -0.108470350576224525715263518828799681350073669985
         511242380074788803859218005133488714790833056126834
         428390433094803074647832349908331329447113059856644
         012690641685491682714756272017082...E- 4

t(33) =  13.2
w(33) =  12785615484202001979209002012583740935481594108722
         864568069660965954206479l /
         897667693284497596193774404299625202605789064946867
         179939890280365943908691406 25
     Truncated floating-point equivalent (9K digits):
         +0.142431498647572026193638394239238944781744088510
         038942777655527701029602849639002985043951114778420
         899260118507974381156564781893731660931267870373770
         230332795982311530784295555221777...E- 4

t(34) =  13.6
w(34) = -205508798399049921204963733796736360099049725581650
         390130494129562552600294783 /
         201975230989011959143599240967415670586302539613040
         390486475313082337379455566406 25
     Truncated floating-point equivalent (9K digits):
         -0.101749505319407308127126673489964838770285289240
         125580130128284115320499206988307506554899774078640
         759682809095273230883034444530172496393292372902170
         050863639802770668833686853112278...E- 4

t(35) =  14.0
w(35) =  511542636117261582141764059916058275503904887269720
         144639391381229409208721521 79 /
         454444269725276908073098292176685258819180714129340
         878594569454435259103775024414062 5
     Truncated floating-point equivalent (9K digits):
         +0.112564437533011932429598084706993319649062500360
         663709363767443012155866390847262292703609084059640
         241408504173400967172299189698039231250912827140740
         579411792652254614174816274638328...E- 4

t(36) =  14.4
w(36) = -922836046727919946318336663008682085735610897750310
         101545077002414993747637770082 7 /
```

```
          1022499606881873043164471157397541832343156606791O
          34768377812724793329834938049316406 25
     Truncated floating-point equivalent (9K digits):
          -0.902529488047551893284709859039976959799135372 20
          28008120076787960726052307124503859849392140447163
          09329486267617426679024225539906560761180888400952
          14850935804869494675882813100717 4...E- 5

t(37) =  14.8
w(37) =  21025009805521644161007671503600920143991938693727
          83361172155155532082336168186907 51 /
          23006241154842143471200601041444691227721023652798
          28228850078630784992128610610961914062 5
     Truncated floating-point equivalent (9K digits):
          +0.913882874825751018663391046366558145747250085 77
          69252636388260964438438341670312712442743147520696
          67313171521338363771740627207481466792772987526198
          49262101699662596189168687808345 5...E- 5

t(38) =  15.2
w(38) = -40359618735433851042785312164367986361291660942244
          20288535853349570324322452253733226 3 /
          51764042598394822810201352343250555262372303218796
          13514912676919266232289373874664306640625
     Truncated floating-point equivalent (9K digits):
          -0.779684443283519412109920389756380794271189565 70
          20895919784468044408636605891164849088178249820424
          51457692130552300050906240709596502943055201202252
          98339092887496356857889518623857 6...E- 5

t(39) =  15.6
w(39) =  87764376503358359518501824009711830205712298354938
          69740902956041955730496662728297839419 /
          11646909584638835132295304277231374934033768224229
          13040855352306834902265109121799468994140625
     Truncated floating-point equivalent (9K digits):
          +0.753542181001484001299688225319102930561472733 84
          61011209329117313042359195217806622003208392774682
          53495715217966146003215869086068835257822441021933
          11174100680070319406589931103202 2...E- 5

t(40) =  16.0
w(40) = -17418546186068948849467011137845517686936546766683
          40811586318113845368619108776547773734947 /
          26205546565437379047664434623770593601575978504515
          54341924542690378530096495524048805236816406 25
     Truncated floating-point equivalent (9K digits):
          -0.664689291733466566605682241509084669360295417 45
          86013458202907172502497624384446080575185334879781
          62153972239282364961461274354467678974627082543406
          58507125147434435853387801634949 8...E- 5

t(41) =  16.4
```

```
w(41) =  369525249758473710740320186039667544588707949439920
         905140126984889672354129689945941236934311 /
         589624797722341028572449779034838356035459516351590
         972693302210533516927171149291098117828369140625
     Truncated floating-point equivalent (9K digits):
         +0.626712531742069077095666869874301980822729099640
         152863229785343938032251222751992837816272602761987
         463294610022061050680063069635311835361973966612249
         95897828075805438889773249488 7...E- 5

t(42) =  16.8
w(42) = -74661949284845287508352474848422627075220214557422
         885653976236549460262247795260057906919398543 /
         132665579487526731428801200282838630107978391179109
         938559929973700413086135085904970765113830566406251
     Truncated floating-point equivalent (9K digits):
         -0.562783124102397883564214092782942277874623226841
         267418830429196086133717074281264330439323898003947
         852165802764440119694749455713440717458343713609811
         763128972190843094717919315237...E- 5

t(43) =  17.2
w(43) =  156320031856200453455164529577459562278350460231956
         454767745057783925001967665607649583618609010597
         /
         298497553846935145714802700636386917742951380152997
         361759842440825929443803943286184221506118774414062
         5
     Truncated floating-point equivalent (9K digits):
         +0.523689490388115237259567695042378567483829855215
         824971915434181908233200389892047356433577995463974
         999165225618310933218503284285543474108172317677445
         37924542693705361461635340541 38...E- 5

t(44) =  17.6
w(44) = -31887710002209056719396445621915392315619865042059
         920686337815770411858078442364977891772000316362677
         /
         671619496155604077858306076431870564921640605344244
         063959645491858341248558872393914498388767242431640625
     Truncated floating-point equivalent (9K digits):
         -0.474788331558814020318242100847225680937770627339
         273482227747473705857475903668642088796076059625813
         339371033510095556409079083918327989643731204704527
         504494277831909326025913883868...E- 5

t(45) =  18.0
w(45) =  662964712037934221315789066518386539839231998773633
         877054071727676488946910882739105729349478347451471 /
         151114386635010917518118867197170877107369136202 45
```

```
        491439092023566812678092574628863076213747262954711
        9140625
    Truncated floating-point equivalent (9K digits):
        +0.438717137924930918535746956541592938602042413511
        717152493083200735463090785001758549736497160718283
        174312115918462695144215983815131654987889145254552
        529662468156149712800691767220164...E- 5

t(46) =  18.4
w(46) = -13593141355034998836267714653972438937205824009810
        481436553947360648306143033369102761786815748814502
        1623 /
        340007369928774564415767451193634473491580556455523
        557379570530253285257082929149419214809313416481018
        06640625
    Truncated floating-point equivalent (9K digits):
        -0.399789609204133361639459907566529746618308544493
        762781539881087719593485123792226721526937348045085
        092641025447829218237207498381864147987577246226845
        15266341194446242800330413954 3...E- 5

t(47) =  18.8
w(47) =  28155310560814632680453568949618719621905268609188
        733933566948208274318557383680815094539666893365862
        201099 /
        765016582339742769935476765185677565356056252024928
        004104033693069891828436590586193233320955187082290
        6494140625
    Truncated floating-point equivalent (9K digits):
        +0.368035297675559397663112884481854110656236392016
        414787330566757422083217579700914613280971691296662
        626380518087806260344105749107361785313341212169770
        386182166010229960096461414701 39...E- 5

t(48) =  19.2
w(48) = -57886393363588424085720324414576708673624308438123
        517516949474947137905253865505897854521882512176596
        22636787 /
        172128731026442123235482272166777452205112656705608
        800923407580940725661398232881893477497214917093515
        3961181640625
    Truncated floating-point equivalent (9K digits):
        -0.336297101700563949889449296240285519640122470707
        711264851604240403745174958160829809129993742629134
        012342353168286929757408554229664632871885569575996
        402842667687259704026016854572 9...E- 5

t(49) =  19.6
w(49) =  11965990317931076473218145718254855579067066989507
        057126268191981886089792883004062605254800930526260
        29556258231 /
        387289644809494777279835112375249267461503477587619
        802077667057116632738146023984260324368733563460 4
```

```
        09641265869140625
     Truncated floating-point equivalent (9K digits):
        +0.308967473783531398415260527346756762159777536 72
        821602330836456047192324955771606723467501014623 47
        238569528754220231769740311184494297102404345938 22
        852499569090138625082456424027150...E- 5

t(50) =  20.0
w(50) = -246376248236967485941414926032846653810592529861 79
        456600432145169632572387620535685323831485197517 5
        78529132409503 /
        871401700821363248879629002844310851788382824572 14
        455467475087851242366082855396458572982965051778 59
        2169284820556640625
     Truncated floating-point equivalent (9K digits):
        -0.282735560424932482294893070680491018427082331 59
        696896664325823455617174053298979202543684656716 42
        658858123402842521519609841262206038662200631887 43
        884403056095561440588062043701529...E- 5

t(51) =  20.4
w(51) =  508753425947158492955701598014979877290996759535 55
        371872818535431840309896285203631242121509681886 08
        5938238558443539 /
        196065382684806730997916525639969941652386135528 73
        252480181894766529532368642464203178921167136650 18
        3238089084625244140625
     Truncated floating-point equivalent (9K digits):
        +0.259481515288717113030784269336924044888944059 69
        672839495277165028247298810547459891522329851424 99
        920233553038590477310943563939301467551172281516 89
        432966380674904923004139778262342...E- 5

t(52) =  20.8
w(52) = -104832565163526526663350963912930718318670151068 87
        503541027122644796691147443286674318183739463619 95
        7245771748043888507 /
        441147111040815144745312182689932368717868804939 64
        818080409263224691447829445544457152572626057462 91
        228570044040679931640625
     Truncated floating-point equivalent (9K digits):
        -0.237636295330578209028991375960677450090481152 50
        068111814950307179638785793507455281264109936581 40
        067755132341621622886530131520002660707308791567 05
        312037109019140076336804425264085...E- 5

t(53) =  21.2
w(53) =  216350114890177593393323316504992005482969445027 73
        742491244566826304159047272692210341914934686687 12
        820535849648406330591 /
        992580999841834075676952411052347829615204811114 20
        840680920842255555757616252475028593288408629291 55
        264282599091529846191406 25
```

```
     Truncated floating-point equivalent (9K digits):
         +0.21796721368296650773852102761631379639388163285
         69992598305208684386204586016860377399251457053603
         68641649701534591382646144561319899859395760644835
         49370827940768970259594725161304 5...E- 5

t(54) =  21.6
w(54) = -44599181871879345937740398505000787525047725394583
         33247520643041192282275914739830893689115133106458
         37022663491288980250183 /
         22333072496441266702731429248677826166342108250069
         68915320718950750004546365680688143348989194159059
         93446358479559421539306640625
     Truncated floating-point equivalent (9K digits):
         -0.19970016162794502227265519105778034515377346724
         81487883380464285707899515198303793267818607070288
         53030934678746906663946227320941404453943560485686
         20406202327928429661758673799457O...E- 5

t(55) =  22.0
w(55) =  92014354785493496218439044708223306368380097966742
         82499304050406626226403471114454825755024505429790
         08377180425987677525723 79 /
         50249413116992850081145715809525108874269743562656
         80059471617639187510229322781548322535225686857884
         85254306579008698463439941406 25
     Truncated floating-point equivalent (9K digits):
         +0.18311528250342289229038963842469084523765848323
         69613922401958815945954521171566687944097363926422
         09226935403090928972227558969054189999386531113780
         82825360512752819881831224900416 1...E- 5

t(56) =  22.4
w(56) = -18972386546104574202351547484869668257260508445735
         74536953497231409510303260263363285594026362428727
         93254603189030644604936634 27 /
         11306117951323391268257786057143149496710692301597
         78013381113968817189801597625848372570425779543024
         09182218980276957154273986816406 25
     Truncated floating-point equivalent (9K digits):
         -0.16780637375080488439936512541410271633831444593
         70081145359368371136369364224588220504229876691765
         64909819713305449997769479374235712620352539331620
         65890507202930710141060026312824 1...E- 5

t(57) =  22.8
w(57) =  39136325004922289408894272278918093439833253870123
         41401322441760218319587225313724642340318258189245
         57176040250120554497135520455 1 /
         25438765390477630353580018628572086367599057678595
         00530107506429838677053594658158838283458003971804
         20659992705623153597116470336914 0625
```

```
     Truncated floating-point equivalent (9K digits):
         +0.153845221669334629343303463876099179549223732463209174526330634240038862788721185706585741401710039144205205147631693807927241260981254102870114054946221371241061261968753433612500...E- 5

t(58) =  23.2
w(58) = -807045506632365706477484650563975976394174828881167080143050794338805009680392623004801427996011738928670638245003099124827076511663 /57237222128574668295555041914287194327097879776838761927418894671370233705879808573861377805089365594648498358765209559351205825805664062 5
     Truncated floating-point equivalent (9K digits):
         -0.14100011786376728208099972492843376276742977243759836292632057150361634902970519425769196733492298789936805988155649611629199369414531469102800082382482449999999969308904068955504 4...E- 5

t(59) =  23.6
w(59) =  16646325703692793689563610661859546170178390461929994576411259689688053727644634567306627400662411113781653169730406395275703620817161 9 /12878374978929300366499884430714618723597022949788721433669251301058302583822956929118810006145107258795912130722172150854021310806274414062 5
     Truncated floating-point equivalent (9K digits):
         +0.12925796718086211736039076738458627547950689474813331886832683815915977611005229762982183617416179372459923587778122557512250037169745930820898872464900271552616118077579497514334 6...E- 5

t(60) =  24.0
w(60) = -34329219533019103925245981707620494991993648342286144571865144796909995324477901865100127472583666068921815749278147903632115212258553547 /28976343702590925824624739969107892128093301637024623225755815427381180813601653090517322513826491332290802294124887339421547949314117431640625
     Truncated floating-point equivalent (9K digits):
         -0.11847326179372157844109632749144435796374196972103512001468203843583571143150661937523444613932193103041092227624731786883997314017159558643724247491426483744759535798173388210453 4...E- 5

t(61) =  24.4
w(61) =  70805036692959425010053064898897404733119226177828718299088721500714752152335457125011523592208190668160091660462843383140156217088085761111 /65196773330829583105405664930492757288209928683305402257950584711607656830603719453663975656109605497654305161780996513698482885956764221191406255
     Truncated floating-point equivalent (9K digits):
         +0.10860205662889433779304590696486751562936250134
```

```
        87445857301793159890055710091544088591778877274064
        63201215512496861531824203040741452701931452830539
        97264785172150851032599100934098 4...E- 5

t(62) =  24.8
w(62) = -14602405814710353741464654337276792838136565220604
        64968358353910300929885844334773684219689705996032
        35034048490183319220788841367513053524194 3 /
        14669273999436656198716274609360870389847233953743
        71550803888156011172278688583687707439452262466123
        69722186614007242155821586493402719497680664062 5
     Truncated floating-point equivalent (9K digits):
        -0.99544161594303369105696827243187145566784883971
        94041304581239431486879683836641553728694311702012
        16888102348561617359873757464649772232948737236738
        83260422034556308488537394788799 3...E- 6

t(63) =  25.2
w(63) =  30117150469126653035792353587342670472280327840139
        61226171130839087604057247426629482649216354595167
        53315647379106699932150939457113750327865259 /
        33005866498732476447111617871061958377156276395923
        35989308748351025137627049313297341738767590548778
        31874919881516294850598569610156118869781494140625
     Truncated floating-point equivalent (9K digits):
        +0.91247870951314795179243256190435317281368922921
        95072794317314616452851237024490224076415793697943
        11863188925115870934416215814666738420340741463512
        00463672591480407185130435252467 1...E- 6

t(64) =  25.6
w(64) = -62112927900019105594162419674971512581391060765034
        23956106923978965598954269316026736271010011975114
        94704284242140986781889963556346448475326708 67 /
        74263199622148072006001140209889406348601621890827
        55975944683789806559660860954919018912227078734751
        21718569733411663413846781622851267457008361816406
        25
     Truncated floating-point equivalent (9K digits):
        -0.83638906236265506147034576771865403280724890281
        41385622397091635607455158849333383827059077752945
        32148527578314916729340589390594070521750280602889
        07518854905420986586026414254954 0...E- 6

t(65) =  26.0
w(65) =  12810489401417140644818982258968903447730633262155
        79793791111681619530908404759832145709915260246730
        04971006817735145258379890100465866433277699012 71
        /
        16709219914983316201350256547225116428435364925436
        20094587553852706475923693714856779255251092715319
        02386678190017624268115525865141535177826881408691
        40625
```

Truncated floating-point equivalent (9K digits):
+0.766671901297429997015551552590402806891148276454629727118491453557297583331506369595381219892225434452432513708548241685570427139537160538764330156680039447502265889968699797829176...E- 6

t(66) = 26.4
w(66) = -26420318146985707061100132781521595056991814487233529848123124661357444365257037946351642545701748511685771034520909374979309703262121577263726413690233759574480871246145303807723125651196397957108223145212821996168589570828310858427753324314958609467803700259275396546032599331965684541501104831695556640625

Truncated floating-point equivalent (9K digits):
-0.702747565752788224673304617314218105239311472911697163430115836103581651851499265407410183854907133333935029618515329434506971092395987028202866828369669865872277073157466248924026...E- 6

t(67) = 26.8
w(67) = 5449023670187998758744758758096612407225302708903972752483549466910221608213641298370991287209285788276663396371563425518370548295942329170935919357172992 /
8459042581960303826933567377032715191895403493502076728849491379326534363699431462449797086568713025583255833696422285733484969227902183774858713150024414062 5

Truncated floating-point equivalent (9K digits):
+0.644165532611048589849824412503136762045921374822090751828611229691417966932801057835331102711275227180991788003137870580303933222647063505286418582807612773122165061685274269241828...E- 6

t(68) = 27.2
w(68) = -1123810947235089026811727212409921181253713442551940649757459813558191489406534761137226119610142138277356796620427155877458923602149904929501021085888473872 /
1903284580941068361060052659832360918176465786037967263991135560348470231832372079050120434447796043075623256258169501429003411807627799134934321045875549316406 25

Truncated floating-point equivalent (9K digits):
-0.590458704120550917220880231489193622636193878686957645688557873132133371681549589417343038572491319963413243812752582069722482501531193230065072648352106272151952520305263450860718...E- 6

t(69) = 27.6
w(69) = 2317780320239875036316775726302788449087070867258 3

```
        69456650163119261885587062191584097737306834759169
        73901631079540501823740274320055061690741683507990
        65996031 /
        42823903071174038123851184846228120658970480185854
        26343980055010784058021622837177862770977507541096
        92015232658088137821525767656716254805360222235321
        99859619140625
    Truncated floating-point equivalent (9K digits):
        +0.54123518736432913421574993723263758055027809773
        68591837982046781704800552433537139571195213314387
        13430751107329044303393505096079314590980562549671
        86363297424559344350412161764260 6...E- 6

t(70) =  28.0
w(70) = -47802216280509280800478119684222019266373120535419
        29225069145529879295720499919610577101403100129296
        15880717460626431136328310123404031688334165731505
        0783500903 /
        96353781910141585778665165904013271482683580418172
        09273955123774264130548651383650191234699391967468
        07034273480698310098432977227611573312060500029474
        4968414306640625
    Truncated floating-point equivalent (9K digits):
        -0.49611146893112167279168904974893709332290821646
        28942568492214957174279728667461845376800932410423
        63491321244367923285100017057008254142006470671003
        01374675436770724441934264876959 4...E- 6

t(71) =  28.4
w(71) =  98588479806330139949330801539236190894764753380075
        57413099557211761026932912564311791467745310627944
        77165442487806912311548953529018619966978954886959
        523403631739 /
        21679600929781856800199662328402986083603805594088
        72086639902849209429373446561321293027807363192680
        31582711533157119772147419876212603995213612506631
        7617893218994140625
    Truncated floating-point equivalent (9K digits):
        +0.45475228130650904526099989048800129931191047635
        99427612698575119319057093290682686276681431461390
        16164807734551884269567105225680867849749197513587
        98801881276352013953694784064 4235...E- 6

t(72) =  28.8
w(72) = -20333051989888661815292297478005658234184593883657
        75226862726000809093008402933945566381819918153088
        85567901876795082910876568717938476043157787845202
        276258421835107 /
        48779102092009177800449240238906718688108562586699
        62194939781410721216090254762972909312566567183530
        71061100949603519487331694721478358989230628139921
        464025974273681640625
```

Truncated floating-point equivalent (9K digits):
-0.41683940699719340267177744591948285683357323190
37547187358169906133686517594300806325840775432268
35816725113926720393707219891176932708839145233494
80980836095374134768455088362695 6...E- 6

t(73) = 29.2
w(73) = 41935347017589284796709014659503078104431776535496
87475539171043498037072450221891380540538329668816
93780011389460694193381288400300339364036357874750
94513219708636391 /
10975297970702065005101079053754011704824426582007
41493861450817412273620307321668904595327477616294
40988747713660791884649631312332630772576891331482
32940584421157836914062 5
Truncated floating-point equivalent (9K digits):
+0.38208846019063275991568376857476270922375561696
46301234797118293409576290475403460271139903258059
04047678139869750242994406609394674903152565322300
18119830169539959309694012419815 5...E- 6

t(74) = 29.6
w(74) = -86488224941524392611863688682831491198945585387133
79276362874144193167767477594407246366808619526089
29673594228744449254392794107078523016021454088582
80556329411547255 83 /
24694420434079646261477427870946526335854959809516
68361188264339177615645691473755035339486824636662
42224682355736781740461670452748419238298005495835
24116314947605133056640625
Truncated floating-point equivalent (9K digits):
-0.35023387235347272141655851762703286847658542696
11313309058390979931263917181731766729013228933369
05535004386065435451902010058734964226255396909302
58165396675756001613467879437759 7...E- 6

t(75) = 30.0
w(75) = 17837517175444330036560169582918532587397165997676
81419365811018731113784882958341817440064284764997
18494787920180975857562663755963405431099860488900
64132388944307617525 79 /
55562445976679204088324212709629684255673659571412
53812673594763149635202805815948829513845355432490
45005535300407758916038758518683943286170512365629
29261708632111549377441406 25
Truncated floating-point equivalent (9K digits):
+0.32103549190277068118327852951999078657315606798
90218461552906178570736441534895593199821595874021
12357934832569920144789917666498825677330913545915
04918050286540658343195534507248 3...E- 6

The above results are illustrated graphically in Figure 1 in comparison to the analytical solution $y = e^{-x}$. The Runge-Kutta Milne-Simpson results are virtually indistinguishable from the graph of $y = e^{-x}$ on the main plot. The values w_i from the Milne-Simpson numerical approximation oscillate around the true solution.

Figure 2 illustrates the approximate absolute error $|e^{-x} - w_i|$ for the two Milne-Simpson approximations for step size h = 4/10. In each instance, the absolute error quickly reaches a maximum and then gradually declines over the computation interval (maximum $x = t_i = 30$). These results contrast markedly to those obtained by Young and Gregory ([14], Figures 8.9 and 8.13) for the identical problem using ordinary machine floating-point arithmetic. When roundoff error is a factor, solutions "blow up" and become unusable by x = 5 for Euler starting values and by x = 25 for Runge-Kutta starting values. Hence, in the absence of roundoff error, much of the instability of the Milne-Simpson method vanishes (or at least is delayed beyond x = 30).

If we use step size h = 1/10, we get the results listed below as a consequence of the first 75 iterations (floating-point equivalents are from the rational arithmetic solutions using K = 20 and K = 22, respectively) :

Solution using Euler Method starting values (h = 1/10):

```
t(1)  =  0.1
w(1)  =  9 / 10
     Truncated floating-point equivalent (9K digits):
         +0.900000000000000000000000000000000000000000000000
         000000000000000000000000000000000000000000000000000
         000000000000000000000000000000000000000000000000000
         000000000000000000000000000000000...E- 0

t(2)  =  0.2
```

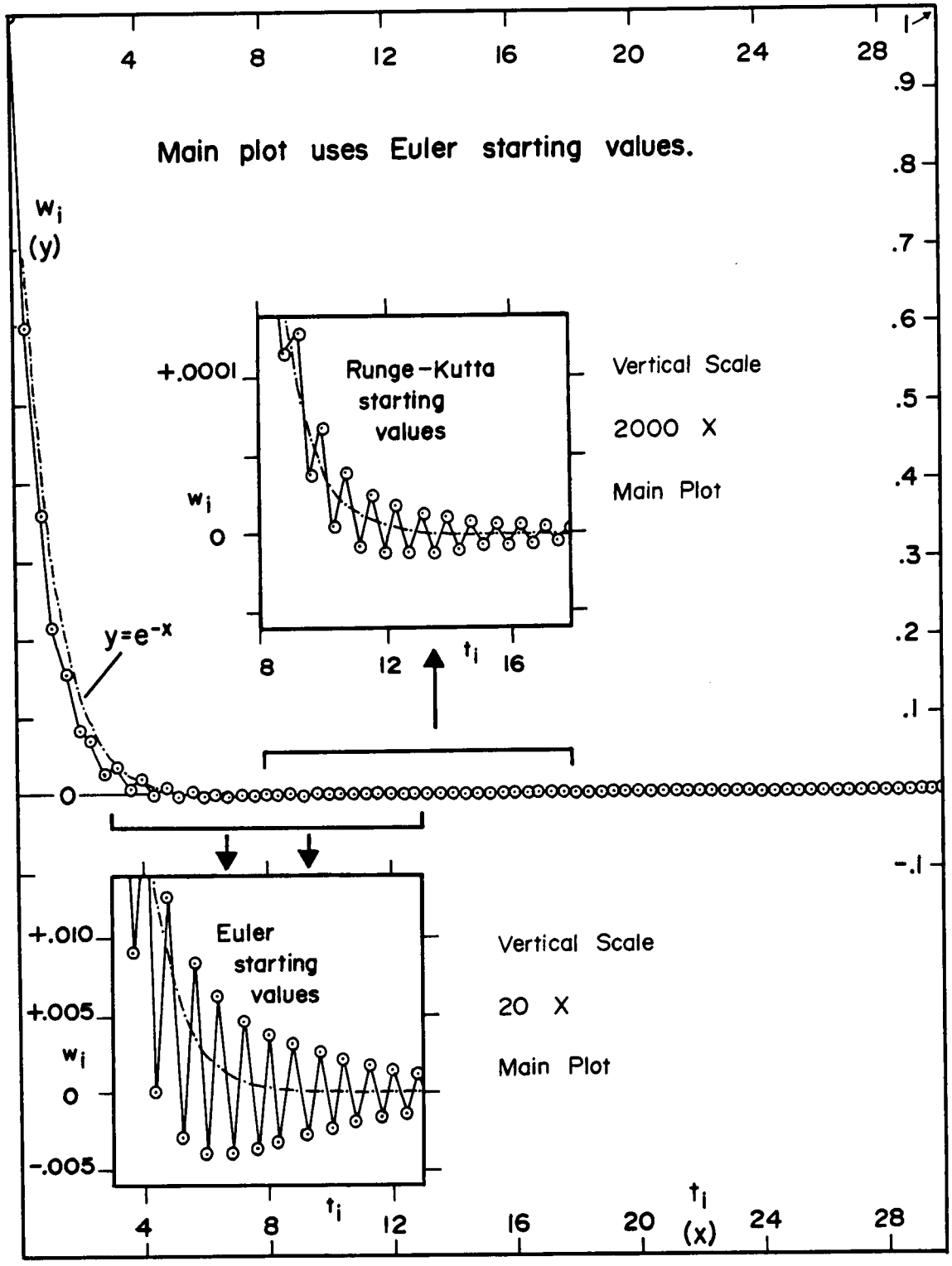


Figure 1. Milne-Simpson approximation to the initial-value problem $y' = -y$, $y(0) = 1$, where step size $h = 0.4$.

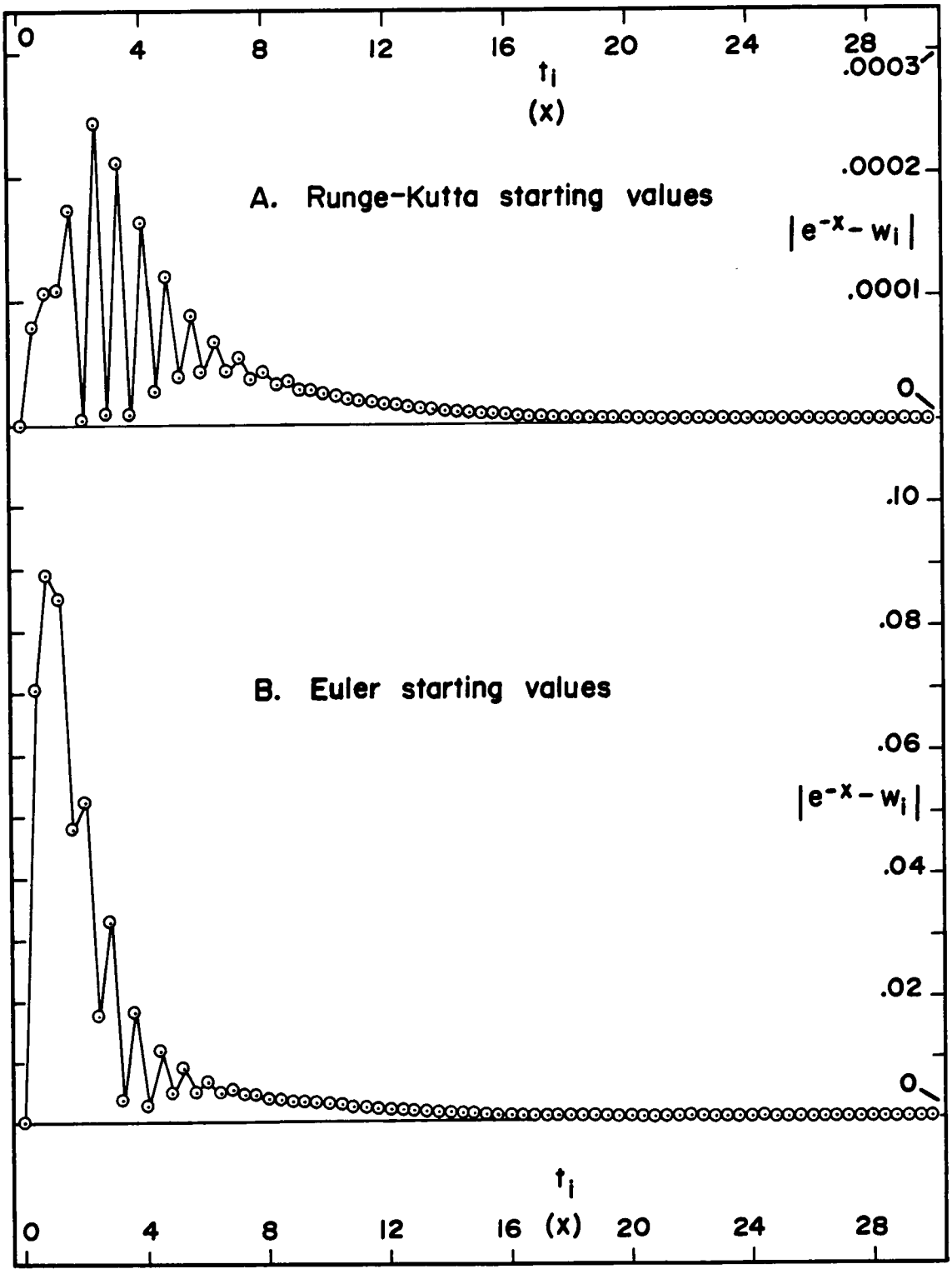


Figure 2. Absolute error in the Milne-Simpson approximation to the initial-value problem $y' = -y$, $y(0) = 1$, where step size $h = 0.4$.

```
w(2)  =  81 / 100
     Truncated floating-point equivalent (9K digits):
         +0.810000000000000000000000000000000000000000000000
         000000000000000000000000000000000000000000000000000
         000000000000000000000000000000000000000000000000000
         000000000000000000000000000000...E+ 0

t(3)  =  0.3
w(3)  =  729 / 1000
     Truncated floating-point equivalent (9K digits):
         +0.729000000000000000000000000000000000000000000000
         000000000000000000000000000000000000000000000000000
         000000000000000000000000000000000000000000000000000
         000000000000000000000000000000...E+ 0

t(4)  =  0.4
w(4)  =  49751 / 75000
     Truncated floating-point equivalent (9K digits):
         +0.663346666666666666666666666666666666666666666666
         666666666666666666666666666666666666666666666666666
         666666666666666666666666666666666666666666666666666
         666666666666666666666666666666...E+ 0

t(5)  =  0.5
w(5)  =  20118719 / 33750000
     Truncated floating-point equivalent (9K digits):
         +0.596110192592592592592592592592592592592592592592
         592592592592592592592592592592592592592592592592592
         592592592592592592592592592592592592592592592592592
         592592592592592592592592592592...E+ 0

t(6)  =  0.6
w(6)  =  4127843293 / 7593750000
     Truncated floating-point equivalent (9K digits):
         +0.543584301958847736625514403292181069958847736625
         551440329218106995884773662551440329218106995884773
         366255144032921810699588477366255144032921810699588
         847736625514403292181069958847736...E+ 0

t(7)  =  0.7
w(7)  =  1666018472917 / 3417187500000
     Truncated floating-point equivalent (9K digits):
         +0.487540842554586191129401005944215820759030635573
         384545038866026520347508001828989483310470964791952
         446273433927754915409236396890717878372199359853680
         841335162322816643804298125285 7...E+ 0

t(8)  =  0.8
w(8)  =  342579806645099 / 768867187500000
     Truncated floating-point equivalent (9K digits):
         +0.445564347412210252502159223695574861555657166082
         406137275821775135904079662653050856068687161509932
         428999644363156022963979068231468780165625158 76...
```

```
          64482040339379159680942945689173 3...E+ 0

t(9)  =  0.9
w(9)  =  137924213577196931 / 345990234375000000
     Truncated floating-point equivalent (9K digits):
          +0.398636145977774552614495306158162430072199924 35
          660778901138602403653462943205360519794295133420 26
          678408327547178332119652040395829452375710568059 29
          538744658391053757613733227206783...E+ 0

t(10) =  1.0
w(10) =  284407669727231047 57 / 77847802734375000000
     Truncated floating-point equivalent (9K digits):
          +0.365338082434593984671462432643824369340466141 42
          805441557325585844273713670316491702120586606352 66
          504967939369876743922028346828089414754726614368 60
          661233639607532476167000037006302...E+ 0

t(11) =  1.1
w(11) =  11414018628935076137533 / 35031511230468750000000
     Truncated floating-point equivalent (9K digits):
          +0.325821474096375925074538206986776065940074738 79
          369348480689156323892823334906275439655438564066 23
          863406460157557645331452771977684432688845220057 52
          772667632454513782826803463148795...E+ 0

t(12) =  1.2
w(12) =  23621129106547438037210 51 /
          78820900268554687500000000
     Truncated floating-point equivalent (9K digits):
          +0.299681036705577973292210414694923375619403546 82
          686903531320461474096172543935847440427963529827 66
          484505676208570373261802372047493853635603665762 75
          732283224700591403156274473650772...E+ 0

t(13) =  1.3
w(13) =  94411966269295423335220241 9 /
          35469405120849609375000000 00
     Truncated floating-point equivalent (9K digits):
          +0.266178600818422592671954845645691530096988064 39
          050603060214849228929999449082656372707570799987 64
          167925391610698558932781000473450265386798717641 73
          353602197671412606506036735019797...E+ 0

t(14) =  1.4
w(14) =  19628609519853482600783106469 3 /
          79806161521911621093750000000 0
     Truncated floating-point equivalent (9K digits):
          +0.245953559794555980391386343433406299292052691 11
          982202729450224664655618512578824246801104886171 21
          316147205818208737210266837678980045048866979448 97
          300997953692137865758068896060392...E+ 0

t(15) =  1.5
```

```
w(15) =  78045108357201879119534201859517 /
         3591277268486022949218750000000000
      Truncated floating-point equivalent (9K digits):
          +0.217318526313907823871310181080648293966705276l8
          618033686975025857653261033112278466063103463171 95
          108876392986852674549321440452235091808793400559 81
          387984410215436882233077986736803...E+ 0

t(16) =  1.6
w(16) =  16321915751223125306542001919090299 /
         8080373854093551635742187500000000 0
      Truncated floating-point equivalent (9K digits):
          +0.201994561711453160319959066062905641372044930 76
          359072788116440681270172461621413819221678782282 31
          473775063411855271275750285490787972493066914564 84
          735633660881795577437492210160458...E+ 0

t(17) =  1.7
w(17) =  64463898772014467061591161695273957 31 /
         3636168234342098236083984375000000000 0
      Truncated floating-point equivalent (9K digits):
          +0.177285248144405768421879607549880550258805018 36
          550378389644985952083121892890873007390001433978 92
          316350627914123709896141155715439632206642849733 19
          521559385474668844524144286771099...E+ 0

t(18) =  1.8
w(18) =  13583932208499815331642414384590963983 57 /
         8181378527269721031188964843750000000000
      Truncated floating-point equivalent (9K digits):
          +0.166034760074021748836736584789480734635752010 51
          136436491645436300298019147386555656603072606634 53
          820271699508481520773909449097935138616506985182 27
          696353373134923798399207389880752...E+ 0

t(19) =  1.9
w(19) =  53191041242655976560912555499804687703593 3 /
         3681620337271374464035034179687500000000000
      Truncated floating-point equivalent (9K digits):
          +0.144477258298932611277837344153176122840995690 98
          641864855626610798434632181595338608257863818050 86
          574014580162633849307896379578845818879538509809 17
          377693763345047791210783691951452...E+ 0

t(20) =  2.0
w(20) =  113175857295385793038623027907972313577045851 /
         8283645758860592544078826904296875000000000000
      Truncated floating-point equivalent (9K digits):
          +0.136625660476037814852524849734496072885865351 04
          042106646009371215346628420658404271084277530807 97
          229314679907874819342908676763312195789164301589 04
          947604205441713489145749126808359...E+ 0

t(21) =  2.1
```

```
w(21) =  438308208857102914086173749368585117162931336l9 /
         372764059148726664483547210693359375000000000000
     Truncated floating-point equivalent (9K digits):
         +0.117583280388688230045735401464673768429341349567
         759687956514899207961621758377986901578883912016961
         605389300331145273372661487101220059600296867847O
         367639081631309779395204561714816...E+ 0

t(22) =  2.2
w(22) =  9442418337282121452846441791540824467411204631093
         /
         838719133084634995087981224060058593750000000000000
     Truncated floating-point equivalent (9K digits):
         +0.112581410925429415913649729017301248887476186735
         553801253788996176196683398203344005500133417643521
         787322353658397101844923325539097121420482287532483
         346200681869477616317406534159076...E+ 0

t(23) =  2.3
w(23) =  360551475886076911540154596595289428952141192330111
         17 /
         377423609888085747789591550827026367187500000000000
         000
     Truncated floating-point equivalent (9K digits):
         +0.955296559197735964641812961019449205358548087665
         543832637964575423671888489710768934135964785950571
         149563220370229870477193892770310028469098351892804
         650109896279450011726677875015291...E- 1

t(24) =  2.4
w(24) =  7891746220058078673576493601009702182233302999932454499 /
         849203122248192932526580989360809326171875000000000000000
     Truncated floating-point equivalent (9K digits):
         +0.929311964747062253022472198724972871082193437738346140500878389004944088134067451531210608827616140167690270059985874188774671791383339210075453930456829487832596725316992689457901...E- 1

t(25) =  2.5
w(25) =  2959198626517456360420815919011758082948665534497145845 31 /
         382141405011686819636961445212364196777343750000000000000000
     Truncated floating-point equivalent (9K digits):
         +0.774372676634440285889627217630697799623880743289043111157957047686130330992564909886428135395225421039782780098810400063611659641478137772618355328254376604934127570296509196639245...E- 1

t(26) =  2.6
w(26) =  661030467421431344737946473617564708608066043121 87
```

```
          099471957 /
          859818161276295344183163251727819442749023437500000000000000
     Truncated floating-point equivalent (9K digits):
          +0.768802634315390735600958067449059524195478225015218924526377414491931871263891080685532125567106435920826743383502048082677453260918667206891269829723315600073985279294962245665818...E- 1

t(27) =  2.7
w(27) =  2421576572669420180099352479400161809780525791918767667975433 /
          3869181725743329048824234632775187492370605468750000000000000
     Truncated floating-point equivalent (9K digits):
          +0.625862713182384382531150284081228816191431408176167148942726874842256092462405244285417847359538946077745725951564280701951490608553508767001357162632203443198322831514552334072924...E- 1

t(28) =  2.8
w(28) =  5552295895329288559806524605075063551296743029500323562028410651 /
          8705658882922490359854527923744171857833862304687500000000000000
     Truncated floating-point equivalent (9K digits):
          +0.637780088790405534314870784858982574227076742179567587369438595198912264625176426940954587394643478566684436225031645750917511887986553050158189883375460724700393570602966373577564...E- 1

t(29) =  2.9
w(29) =  1973942774149784216508260538370435534263924887809632400654914824819 /
          3917546497315120661934537565684877336025238037109375000000000000000
     Truncated floating-point equivalent (9K digits):
          +0.503872200496565969707588295689788498570216008730999599178329620929407941638619292314084838853156385066702524845505363496155888191550944283424486149657324230328746163677260566147349...E- 1

t(30) =  3.0
w(30) =  4679770719820008315081535386927009801635346007017296349422477389174937 /
          8814479618959021489352709522790974006056785583496093750000000000000000
     Truncated floating-point equivalent (9K digits):
          +0.530918547903192396779609634244222935827168189182873716677457109402382693670733260583785055271189839780305780549209174790955993493981685817358580836278763148152320317870648052680307...E- 1

t(31) =  3.1
```

```
w(31) =  160078357383740258323899071198303587340420735002931
         794229250908762422717 /
         396651582853155967020871928525593830272555351257324
         2187500000000000000000
     Truncated floating-point equivalent (9K digits):
         +0.403574230644133660140044047542709852285819886738
         057258094581328665347114567894604005064293948314571
         873174582533994393550670302586600495122143629861792
         0200654370814308508470913659442 70...E- 1

t(32) =  3.2
w(32) =  396128720298849797554092214427134113696693387759533
         063147501848185683606 99 /
         892466061419600925796961839182586118113249540328979
         492187500000000000000000
     Truncated floating-point equivalent (9K digits):
         +0.443858581769202230448523991944958210813405833691
         232754165655830831936155552407454122410988590804474
         972580400699505020141843192464098218159771461946340
         395921609341446858262304396933...E- 1

t(33) =  3.3
w(33) =  128923105912360495624288206380270475891809880667352
         475793990730953071720133 31 /
         401609727638820416608632827632163753150962293148040
         771484375000000000000000000
     Truncated floating-point equivalent (9K digits):
         +0.321015894386664066615614095863790601108810481993
         020875989545134928962206707005470152660544607565464
         878893611516750442651512615717985124979895072141476
         847447484084767249090034580581 14...E- 1

t(34) =  3.4
w(34) =  337079868826079282059139914105910683356414502388321
         345013236479877160026382555 7 /
         903621887187345937369423862172368444589665159583091
         735839843750000000000000000
     Truncated floating-point equivalent (9K digits):
         +0.373031987832088948316894512934004726648549101568
         679551270968541555076924347233125684223918868132015
         351403706335709526784657484572892568848738412143933
         748425493357930066782286280422...E- 1

t(35) =  3.5
w(35) =  102861006979344615606833034770872875631459732522919
         95322137392584519526668827927 33 /
         406629849234305671816240737977565800065349321812391
         281127929687500000000000000000
     Truncated floating-point equivalent (9K digits):
         +0.252959804040540808352240546754125001437250031219
         596281632831620268182233728252783896701058267947745
         670658801010337931591402157448159764144628347834923
         879189918796179614634151190852...E- 1
```

```
t(36) =  3.6
w(36) =  28867321203035429028680190287116233167850805956907
         30772197457138669980263555748154 51 /
         91491716077718776158654166044952305014703597407788
         03825378417968750000000000000000000
      Truncated floating-point equivalent (9K digits):
         +0.31551841456896189369605433330327519066793314145
         23209913785870018568153547713593279818403887921428
         52498950623160794949771370161286391117215394562975
         41861618793925596190502065946447 4...E- 1

t(37) =  3.7
w(37) =  81006265384823328779824470664465941609181231043329
         68816514861801234509951556279027601 9 /
         41171272234973449271394374720228537256616618833504
         61721420288085937500000000000000000000
      Truncated floating-point equivalent (9K digits):
         +0.19675434104271266391905440154073749963308232616
         25213766556798146987269376309359633571899643644898
         22719280525762363085909436324101193486025018903671
         89373214374685931722908335068861 9...E- 1

t(38) =  3.8
w(38) =  24912240676161268575298959928795169708593281510282
         53433862042803540290756864574423992389 3 /
         92635362528690260860637343120514208827387392375385
         38873195648193359375000000000000000000000
Truncated floating-point equivalent (9K digits):
         +0.26892797735255426622128894294898508152396267463
         35599645148375567524864781741570873743504350947471
         90693727755544305753921490204552310597912693831937
         57978022156699672335000462271135 9...E- 1

t(39) =  3.9
w(39) =  62623672308364460750798373081297934472261746292153
         82015152193441762164496747196636233224317 /
         41685913137910617387286804404231393972324326568923
         42492938041687011718750000000000000000000000
      Truncated floating-point equivalent (9K digits):
         +0.15022742119426506670679774265310533003754604343
         64616396002229561995867758435647128759477241903724
         57296745852938022677343104278142912342160921074968
         45185467275025188667998533854740 2...E- 1

t(40) =  4.0
w(40) =  21694874849785201947902370939980345424849615848266
         50282763091399466799314799597510323552435899 /
         93793304560298889121395309909520636437729734780077
         70609110593795776367187500000000000000000000000
      Truncated floating-point equivalent (9K digits):
         +0.23130515500536350175216186958877880879174798192
         61219632294579794613165845136083307683151552598677
         61033687140315094776919685804160569566491231124890
```

```
          655637022267542385326252991524781...E- 1

t(41) =  4.1
w(41) =  47102792602981884484791530791694426496457009471709
         27865209262217364702400570359758844387516821 31 /
         42206987052134500104627889459284286396978380651034
         96774099767208099365234375000000000000000000000000
Truncated floating-point equivalent (9K digits):
         +0.11159951442352432198227043915668416099004259028
         04921760317134359802586379016312067808872625662007
         12919089283556133974464680363021289974518767498311
         41077897507542620108694949547008...E- 1

t(42) =  4.2
w(42) =  19092868591588479915593544585281238465679435969598
         20608526511151636657575126219245228585461334591 57
         /
         94965720867302625235412751283389644393201356464828
         67741724476218223571777343750000000000000000000000
     Truncated floating-point equivalent (9K digits):
         +0.20105010963131951222485955947898790286372274482
         78793229139059581452592162281399159640873553203461
         83495330219901955644970559815370269720261345275540
         05555094090026231618192944037 5815...E- 1

t(43) =  4.3
w(43) =  33936563411419461078247121773982527608554847415115
         36886058153811761721979022090806840458341362215113
         3 /
         42734574390286181355935738077525339976940610409172
         90483776014298200607299804687500000000000000000000
         000
     Truncated floating-point equivalent (9K digits):
         +0.79412428684754703866972547027954343184275042571
         66540754992815932926430845118097749348741678090979
         24509127294055256306053053923821823283678713326969
         63558158922358325790147321957 1042...E- 2

t(44) =  4.4
w(44) =  17005032053348866265844317920427276149489719724337 3
         05133843078924826694416679838039410988793473160826
         0251 /
         96152792378143908050855410674432014948116373420639
         03588496032170951366424560546875000000000000000000
         00000
     Truncated floating-point equivalent (9K digits):
         +0.17685427154806384475213417851918082940834777057
         48636072324673044587993891942685527520296694118794
         12249773160889634384894263849936891544189485139550
         44422858016988907930440266795210 3...E- 1

t(45) =  4.5
w(45) =  22703549390679356699312243317294660056347009924114
```

```
        90120291408097513183956669684283541659971234840767
        487219 /
        43268756570164758622884934803494406726652368039287
        56614823214476928114891052246093750000000000000000
        00000000
    Truncated floating-point equivalent (9K digits):
        +0.52471000302176943749877734481323985218950823125
        42865358951254718390186013526759080602095658190459
        35232286587756419676909566513804016688265355286009
        98676447691860050832246037823125 5...E- 2

t(46) =  4.6
w(46) =  15347735517041047793485861186309180604775068226472
        97295615736103991021463596084510570295712176366882
        563650293 /
        97354702282870706901491103307862415134967828088397
        02383352232573088258504867553710937500000000000000
        0000000000
    Truncated floating-point equivalent (9K digits):
        +0.15764760363035324902172431769728478842518799458
        16897980380816965633328896531687187266762502757234
        99298277047539707791814189997520769397087360848936
        38543569481111846011696049952746 0...E- 1

t(47) =  4.7
w(47) =  13053228899662406187186339798948176919652763523587
        29072541468342436433480795270510627240386908704232
        36799705917 /
        43809616027291818105670996488538086810735522639778
        66072508504657889716327190399169921875000000000000
        0000000000000
    Truncated floating-point equivalent (9K digits):
        +0.29795351074363932265520990551161656928587653689
        91211473290512166884243677333950693093113691212898
        62157553250891221972201934771123681502583056598034
        27035655284437665942618104991838 7...E- 2

t(48) =  4.8
w(48) =  14051921776104788347450341237671591109895109374026
        29394845515924488184522688492530072722345519166601
        78366753471099 /
        98571636061406590737759742099210695324154925939501
        98663144135480251861736178398132324218750000000000
        000000000000000
    Truncated floating-point equivalent (9K digits):
        +0.14255542808836961494056656078265965355523573284
        69230325140393964896035141403157442054718033054632
        87018706623558801517574585117921486146748502212581
        35624652499001086211471161455079 6...E- 1

t(49) =  4.9
w(49) =  46937924974829590467209672694665136953108508968217
        34893281409660347400222155560409833112479686740857
```

```
        780178405590931 /
        443572362276329658319918839446448128958697166727758939841486096611333778128027915954589843750000000000000000000000000
    Truncated floating-point equivalent (9K digits):
        +0.105817965605325401647502061297629693920038725327768561041585556837461821946266351964514487782671823393759181173805068090209315676205275552925272379078182875613054981171420899497434...E+ 2

t(50) =  5.0
w(50) =  13060630489340996511656335542623446282630105294117866443253334335985066418295162506913928868172888526391394052845237275 7
        998037815121741731219817388754508290157068625137457614643343717375501000788062810897827148437500000000000000000000000000
    Truncated floating-point equivalent (9K digits):
        +0.130863082454925284475056729938600618899558635904258823611347943555408692938118784259536867762656232274408404830330268089240239639677074763839747616827430063147139042584724971089698...E+ 1

t(51) =  5.1
w(51) = -261797620773159255361566492678286174304176982736194953378754980228241779963690564744958997946206476778158402779166170467 /
        449117016804783779048917824939528730570680881311855926589504672818975450354628264904022216796875000000000000000000000000000
    Truncated floating-point equivalent (9K digits):
        -0.582916280117157014108196941669123366966619284238011155716109926922249798444774456917038031509201554619114461720805450435448302315123160475416931676069586114446727201561107887685 83...E+ 3

t(52) =  5.2
w(52) =  123269470034215226207415038495904234574545071703859752189246410810383653370013021335586075013627922198997831335171912074505 1
        101051328781076350286006510611393964378403198295167583482638551384269476329791359603404998779296875000000000000000000000000000000
    Truncated floating-point equivalent (9K digits):
        +0.121986985743921871204779237252284298096109198941974802633644011090645648994649136292601214703390629524343142267726378781072701836351133730495268858869832643201145069384936063700193...E+ 1

t(53) =  5.3
w(53) = -908509379798017184542789017462389163834779948830158208699491003038491875704250248117033501716085573771602923639688499055415 81 /
```

```
          45473097951404357628702929775127283970281439232825
          41256718734812292126434840611182153224945068359375
          00000000000000000000000000000
     Truncated floating-point equivalent (9K digits):
          -0.19979051807011559162504791590549482083611677411
          55614516672472617881788548354191858475415500368164
          96277917726252381611145710520726526920703724877902
          52791806372556851256112468327702 5...E- 2

t(54) =  5.4
w(54) =  11812303110103257384836562987755618145828261783173
          13874030172173144029356721376393286520791230383934
          86108456422962497733366096693 /
          10231447039083980466458159199403638893313323827385
          71782761715332765728447839137515984475612640380859
          37500000000000000000000000000
     Truncated floating-point equivalent (9K digits):
          -0.11545095297840500443795444273903132856472340235
          82697761575807952782767737966574724304929721737965
          21495249594085860048425511363163120388993657361889
          25706254749176472481431665513475 3...E- 1

t(55) =  5.5
w(55) = -14877339864977856875483481069942152336644419731334
          75792723241087452714820479284572310415025997449040
          82994375096384256413998741324 83 /
          46041511675877912099061716397316375019909957223235
          73022427718997445778015276118821930140256881713867
          18750000000000000000000000000000
     Truncated floating-point equivalent (9K digits):
          -0.32312883142740942953232615692964832098664623924
          96766432032447395054494978673600231395998917564560
          65581136789545323752834601763393627245950821936657
          04964401009632577773189490222945 1...E- 2

t(56) =  5.6
w(56) =  11485069614164023808397916406315455111846370263556
          95968904110473908982880660885675297127704672091881
          85582377563076357992344183794662 99 /
          10359340127072530222288886189396184379479740375228
          03930046236774425300053437126734934281557798385620
          11718750000000000000000000000000000
Truncated floating-point equivalent (9K digits):
          -0.11086680609390218982731909435480029659172081659
          36736792399979907236653421447911043850159588989432
          58251824724524698590350990700863999029416309603465
          07494985032659143723133162732670 9...E- 1

t(57) =  5.7
w(57) = -20137034381799494784776252634362718444624582457986
          96759210794602708048673509742052364335007458101270
          78973359522220143085983761031426026 9 /
          46617030571826386000299987852282829707658831688526
```

```
        176852080654849138502404670703072042670100927352905273437500000000000000000000000000000000
     Truncated floating-point equivalent (9K digits):
        -0.431967333285479056086448938535738252496557313736673409022426458384796600896116270644717439947666531363722946554969487771051387487968296577674913081963614777710899144428103703891947 6...E- 2

t(58) =  5.8
w(58) =  11319390875722546140685089157601143886857134455016180584433425251504040586564568114261616283951417411483870938855070605897814933421645663577 /
        10488831878660936850067497266763636842232371299183899171814734105616304105090819120960077270865440368652343750000000000000000000000000000000000
     Truncated floating-point equivalent (9K digits):
        -0.107918508053802871268735632905714074180734055579804793115110462945168050461368873819131318505866295177812790206213974501425200045852328027392587177701644259837354318279628653916669...E- 1

t(59) =  5.9
w(59) = -24983832719300600465694886363141813602406945728322102505443074775439306470916505865706502574116082379527231306931002153371039012963474617206 7 /
        47199743453974215825303737700436365079004567084632754062731663034752733684729086860443203477188944816589355468750000000000000000000000000000000000
     Truncated floating-point equivalent (9K digits):
        -0.529321366834610689244103752633417351541670803822618163185608722240850400650184155567514562507378111258393683443586253317537700806923684468350020613236955819628007043233443100931017...E- 2

t(60) =  6.0
w(60) =  11294220018968452162585135468679239765137467759865123713288700553279188893256783346143343594982975771278511852641546027038459939079326870426985 1 /
        10619942277144198560693340982598182142776027594042369664114624182819365079064044543599720782367512583732604980468750000000000000000000000000000000000
     Truncated floating-point equivalent (9K digits):
        -0.106349165788550530331744764604855625642259483095694139067718441901328065157518756801841813563596017219371294977960594060769459292135493425174237020166578244963886202491120220634806...E- 1

t(61) =  6.1
w(61) = -29518706685168625485204646782878576323854329222232598923513470468978035458107432337920861983193204241377595688240827618322775686060935417580810381 /
        47789740247148893523120034421691819642492124173190663488515808822687142855788200446198743520653806620
```

```
        67967224121093750000000000000000000000000000000
    Truncated floating-point equivalent (9K digits):
        -0.617678743021200943858728519277164948494513437021
        200199283000197336562819861874987016287126270797426
        755920076249239286810688901173254431300324869349827
        28378777841276174932683298929252...E- 2

t(62) =  6.2
w(62) =  113925205693268209356400429143385857169495289767162
        512440476855072271305395598122550343472962758364552
        500698060715459949337665661784715611961032630933
        /
        107526915556085010427020077448806594195607279389678
        992849160569851046071425523451003947172921471064910
        292626542724609375000000000000000000000000000000000
        0
    Truncated floating-point equivalent (9K digits):
        -0.105950407955156039054501526193812292358533227272
        284809267346635589524317971670033006610527950139100
        830334164057737851368010312531231864609297855738178
        537162221916343703978388799450...E- 1

t(63) =  6.3
w(63) = -338272511873754805837956857288270651128239991788866
        658409583225451992719462269111186401394647937641258
        024748533201766110090898879009864324107692429088
        3 /
        48387112000238254692159034851962967388023275725355546
        782122256432970732141485552951776227814661979209631
        681442260742187500000000000000000000000000000000000
        000
    Truncated floating-point equivalent (9K digits):
        -0.699096304553347131132042125870781824711305520269
        982536597245184149332089885052389458150734104841488
        650332817416625756428711611870488306511589439670777
        727154570155961048740690721558...E- 2

t(64) =  6.4
w(64) =  116006061421650147100427126738724224449203865979297
        238085013854476284373789607390689835494348518394827
        690227597499562410266809996560377148183187038421499 /
        108871002000536073057357828416916676623052370382049
        980259775076974184147318342494141496512582989453221
        671283245086669921875000000000000000000000000000000
        000000
Truncated floating-point equivalent (9K digits):
        -0.106553682146765712249819853044474223212436668224
        816294337725976742323873955070858502208579243189075
        037741376354143309450474295339215106040873619763878
        293946937972992663012696864686...E- 1

t(65) =  6.5
```

```
w(65) = -3798243227084790732449121664136393318323115726956641096210635518800277559456118223444695412084592605797467354558817746870840881294093251810493134542558714691 /
489919509002412328758110227876125044803735666719224911168987846383828662932541223636734306623452539497520774602890014648437500000000000000000000000000000000000000
Truncated floating-point equivalent (9K digits):
-0.77527903202280855760002740748898695372711287519242259514621076295273944292949430949134965384781049149372691776371356104225997180229734455229037150098677716679477024589609432064609 4...E- 2

t(66) =  6.6
w(66) =  1190759466510360891900260603341410903002698778401269170267409281615297260119533315194248890014299317000806441091278398747293913175220612946723287848334140399571 /
1102318895255427739705748012721281350808405250118256050130222654363614491598217753182652189902768213869421742856502532958984375000000000000000000000000000000000000000
Truncated floating-point equivalent (9K digits):
-0.10802313846161911530769291234431148550354308627829057072756673465435329043871506314161212146878796696842609136219584635733419198056962834099333294716408462858452491480027434184388 2...E- 1

t(67) =  6.7
w(67) = -4204687250440539304919590413000411748592385796625955006762118022967024223724835463538607444532957027879472078434515319896852889822721546477532882925846358185366 7 /
4960435028649424828675866057245766078637823625532152225586001944636265212191979889321934854562456962412397842854261398315429687500000000000000000000000000000000000000
Truncated floating-point equivalent (9K digits):
-0.84764485900047105456921445996823352565923767498026514107395314429564685583927300263799240879405115330630604275356117148212459846197291827286491316604969831118532935475313523119316 0...E- 2

t(68) =  6.8
w(68) =  1230495764423777970833893870784256486169324139458114733294849927522777691792260345850873259050441829318668545040179497640011186893523134991939262966135993255083465 1 /
1116097881446120586452069862880297367693510315744734250756850437543159672743195475097435342276552816542789514642208814620971679687500000000000000000000000000000000000000
```

```
      Truncated floating-point equivalent (9K digits):
          +0.110249807376162457171180718289436161860107760 36
          653352991539146612603254059147931234169141674454 42
          574081782221929624643707812988003656708567647471 88
          429684076997665843348517153626342...E- 1

t(69) =  6.9
w(69) = -460747533064980310867752114370284682132477135970 20
          330446277065567639560836057061048468360468240734 98
          540262067016565693744797773346091394587668410211 73
          7939420210319181 /
          502244046650754263903431438296133815462079642085 13
          041284058269689442185273443796379384590402444876 74
          442552815889939665794372558593750000000000000000 00
          000000000000000000
      Truncated floating-point equivalent (9K digits):
          -0.917377789020106777040050965782715785613470761 09
          487915795854467040128019566253912863458483715177 83
          499534830522653498445099338026799595644222548376 35
          597283131592528739117388383675429...E- 2

t(70) =  7.0
w(70) =  127861483242125934443252530724410848315561271238 23
          767753456584099367509764558794435621055280773303 87
          718247304318862792445651425193529970299695728490 81
          952165323903114949 3 /
          113004910496419709378272073616630108478967919469 15
          434288913110680124491686524854185361532840550097 26
          749574383575236424803733825683593750000000000000 00
          000000000000000000
      Truncated floating-point equivalent (9K digits):
          +0.113146838204147703900227682130126890648989455 85
          927528249019838379918244682867628048028151056197 37
          383369594700326763200841134865699395009988488259 253
          056704093608145479211244296108118...E- 1

t(71) =  7.1
w(71) = -501134002144616509963221731589818843333151355508 17
          054849769711486111159838554680051792403960814660 88
          050593898205070568048913305877207824731655072469 78
          445585800730286769283 /
          508522097233888692202224331274835488155355637611 19
          454300108998060560212589361843834126897782475437 70
          373084726088563911616802215576171875000000000000 00
          000000000000000000000
      Truncated floating-point equivalent (9K digits):
          -0.985471437466611965637964721212635654610609020 65
          758114506185773644841886774375531607665630577718 39
          586725194340448295683840928180236542029902804584 28
          660989001449866307424196136309013...E- 2

t(72) =  7.2
w(72) =  133462953609241989996092841799601793642113802924 93
```

```
        764456426818423974764563540784506672123891756871149791261580091406221447950007912054467043722024789851860128528493973833669 9 /
        1144174718776249557455004745368379848349550184625187721752452456362604783260641486267855200105697348333944063369926880113780498504638671875000000000000000000000000000000000000000000000000000
    Truncated floating-point equivalent (9K digits):
        -0.116645606146574451617749905867735721277077422650694882485973363664653655442321453069804364888128852594145539557776256168903036583641774623268030401985267192295209432609265702045067...E+ 1

t(73) =  7.3
w(73) = -542046058432807174824095069131572301327880464125919820845032939768010983648854678065368564439605485338608222016516310041275459456254584272770022928283678726128003021906672426699 /
        514878623449312300854752135415770931757297583081334474788603605363172152467288668820534840047563806750274828516467096051201224327087402343750000000000000000000000000000000000000000000000000000
    Truncated floating-point equivalent (9K digits):
        -0.105276473667034925381237511066457206656586495507394602966540840280855368828135394292339092798038192833648427796619124653076320417965978172329294644590512121321990318045900272372 41...E+ 1

t(74) =  7.4
w(74) =  139819509212915239050039568979128461777977073684648056702689872053778959088375466234987308155317021449211202685889341055022086392889668724919370819129721065173098706566595447935577 /
        115847690276095267692319230468548459645391956193300256827435811206713734305139950484620339010701856518811836416205096611520275473594665527343750000000000000000000000000000000000000000000000000000
    Truncated floating-point equivalent (9K digits):
        -0.120692530752826295615725352755587140586942033192089182661469725157334930272078002341394452745529616432466777777933576423504228465108763763259255870893361344752281558643846105810342...E+ 1

t(75) =  7.5
w(75) = -583857359486685056044155552027691539734824062943114557144982064860496590360071806397627570755117008389271484027229386947083919064642590040973132508597786221822803849442272333721526 7 /
        521314606242428704615436537108468068404263802869851155723461150430211804373129777180791525548158354334653263872922934751841239631175994873046875000000000000000000000000000000000000000000000000000000
```

```
     Truncated floating-point equivalent (9K digits):
         -0.111997122753773732199174478535108786639631904205
         586228731101520651920122364913269315018260034152539
         798256216781552434199998726446550532649391686310619
         399699684195587357475265570458801...E+ 1


Solution using Runge-Kutta Method starting values (h = 1/10):


t(1)  =  0.1
w(1)  =  72387 / 80000
     Truncated floating-point equivalent (9K digits):
         +0.904837500000000000000000000000000000000000000000
         000000000000000000000000000000000000000000000000000
         000000000000000000000000000000000000000000000000000
         000000000000000000000000000000000000000000000000000
         0...E+ 0

t(2)  =  0.2
w(2)  =  5239877769 / 6400000000
     Truncated floating-point equivalent (9K digits):
         +0.818730901406250000000000000000000000000000000000
         000000000000000000000000000000000000000000000000000
         000000000000000000000000000000000000000000000000000
         000000000000000000000000000000000000000000000000000
         0...E+ 0

t(3)  =  0.3
w(3)  =  379299032064603 / 512000000000000
     Truncated floating-point equivalent (9K digits):
         +0.740818422001177734375000000000000000000000000000
         000000000000000000000000000000000000000000000000000
         000000000000000000000000000000000000000000000000000
         000000000000000000000000000000000000000000000000000
         0...E+ 0

t(4)  =  0.4
w(4)  =  6435071971772593 / 9600000000000000
     Truncated floating-point equivalent (9K digits):
         +0.670319997059645104166666666666666666663[illegible]666666
         666666666666666666666666666666666663[illegible]666
         666666666666666666666666666666666666666666666666666666666
         666666666666666666666666666666666666666666666666666666666
         6...E+ 0

t(5)  =  0.5
w(5)  =  1048085061107692159 3 / 1728000000000000000
     Truncated floating-point equivalent (9K digits):
         +0.606530706659544073668981481481481481481481481481
         148148148148148148148148148148148148148148148148148
         814814814814814814814814814814814814814814814814814
         481481481481481481481481481481481481481481481481481
```

```
          1...E+ 0

t(6)  =  0.6
w(6)  =  533444722532934097499 / 972000000000000000000
     Truncated floating-point equivalent (9K digits):
          +0.548811442935117384258230452674897119341563786008
          230452674897119341563786008230452674897119341563786
          008230452674897119341563786008230452674897119341563
          786008230452674897119341563786008230452674897119341
          56378600823045267489711934156378600823045267489711
          9...E+ 0

t(7)  =  0.7
w(7)  =  868825539159954763549499 /
          174960000000000000000000
     Truncated floating-point equivalent (9K digits):
          +0.496585241860970943958332761774119798811156835848
          193872885230909922267946959304983996342021033379058
          070416095107453132144490169181527206218564243255601
          280292638317329675354366712391403749428440786465...
          E+ 0

t(8)  =  0.8
w(8)  =  44220681630483024495420457 /
          984150000000000000000000000
     Truncated floating-point equivalent (9K digits):
          +0.449328675816522120565162393944012599705329472133
          313011227963216989280089417263628511913834273230706
          701214245795864451557181323985164863079815068041131
          941269115480363765686125082558553066097647716303...
          E+ 0

t(9)  =  0.9
w(9)  =  720225727481798067891761689 57 /
          177147000000000000000000000000
     Truncated floating-point equivalent (9K digits):
          +0.406569531226494418698460425279570074570836649788
          029150931147578000191930995162209916058414762880545
          535628602234302584858902493409428327805880088288257
          774616561386870790925050946389157027779189035095...
          E+ 0

t(10) =  1.0
w(10) =  366573812775447165021706182115 1 /
          996451875000000000000000000000
     Truncated floating-point equivalent (9K digits):
          +0.367879093785083363932358682264610119781248843553
          031600246625056528695879065910734524936289572439210
          875086165099543818912478838980557891970447644548814
          763382325915137647766481446984080390234601144184...
          E+ 0

t(11) =  1.1
w(11) =  597041733640598273461276719163165 1 /
```

```
        17936133750000000000000000000000000
    Truncated floating-point equivalent (9K digits):
        +0.33287091965435319830912652464088873110683622104
        45715482022428607279983067699860344763541919952509
        27474824389063222724908594083159086611962848459468
        02777382277270317523139567355199946588266270037153
        3...E+ 0

t(12) =  1.2
w(12) =  30387672243609750307201546184719399 3 /
        10089075234375000000000000000000000000
    Truncated floating-point equivalent (9K digits):
        +0.30119383132434547409714806629476556494666961199
        35979947617070918518199485314262767346824550624115
        28782103209827913876800666574967851016263623579288
        80730931089191826403378474905595403344813805136671
        5...E+ 0

t(13) =  1.3
w(13) =  49492655540933720295363022635543192049 3 /
        18160335421875000000000000000000000000
    Truncated floating-point equivalent (9K digits):
        +0.27253161569536556776369782623747297398402006904
        01104040042996624614072316999562027121733701939892
        32545037991868657553971889528365479228263875670585
        23157595140336868795117682413883209293386993100752
        9...E+ 0

t(14) =  1.4
w(14) =  25190304722283393332832854530854760375199 /
        10215188674804687500000000000000000000000
    Truncated floating-point equivalent (9K digits):
        +0.24659656834742729071899813400206534712558095635
        08101362148621894866239728444473635428159260790115
        78859781260868003598329355571378979180728148114330
        93441405087514232763904884369323719096983153753264
        3...E+ 0

t(15) =  1.5
w(15) =  41027668144134692933586073897193564558489699 /
        18387339614648437500000000000000000000000000
    Truncated floating-point equivalent (9K digits):
        +0.22312998510915432551815323922727971019829398104
        67294579152059037578348756430397733461484319419857
        14397466104164186297771017797416699503562430307524
        46168406669028399325052674110719297264801716827628
        0...E+ 0

t(16) =  1.6
w(16) =  20881870436821542471877755819943179963238240 57 /
        10342878533239746093750000000000000000000000000
    Truncated floating-point equivalent (9K digits):
        +0.20189611982497701069312706874821119990224028279
```

```
        453424673861197713186168200991980392551875657400 13
        280855017509837934388118033894237932549442070783 93
        185696978747812055637725750839744255759247785070 42
        7...E+ 0

t(17) =  1.7
w(17) =  340104928095133455756193991449494525616117737255 7
         /
         186171813598315429687500000000000000000000000000 00
     Truncated floating-point equivalent (9K digits):
         +0.182683361955609635548812804173241009111284134 99
         231449321048186539729696528168702056120418625364 87
         728635072893632053009891751018527382023658026314 81
         375053748887701184340119955255731880453210744632 67
         3...E+ 0

t(18) =  1.8
w(18) =  173103304022053576143644274656161713057531324875 95
         1 /
         104721645149052429199218750000000000000000000000 00
         00
     Truncated floating-point equivalent (9K digits):
         +0.165298495622058031068173654847004648180772427 70
         975218445882198313017657369872500130792120445093 55
         843452422121672883408139128878571805563592982602 14
         280520683030500788503344237618485536587597966324 84
         6...E+ 0

t(19) =  1.9
w(19) =  281935025632880330993841008899963999621688881891 33
         6451 /
         188498961268294372558593750000000000000000000000 00
         00000
     Truncated floating-point equivalent (9K digits):
         +0.149568477054680701361818731604810546208216279 81
         200721855260216891977693651830213876726149721276 38
         789398780174541602834031458755081738020769160234 66
         981125870962896793304172589346328436934781110950 87
         4...E+ 0

t(20) =  2.0
w(20) =  143496496684333075041580319170884926480650450053 07
         900393 /
         106030665713415584564208984375000000000000000000 00
         0000000
     Truncated floating-point equivalent (9K digits):
         +0.135334901199415092351234863920785373583390848 50
         422563534108210008993380132777990404161171795498 86
         799356182484082195451636172466216788932723754526 85
         911638149539151655332686629875192463465569563128 63
         0...E+ 0

t(21) =  2.1
```

```
w(21) =  233714233900949433036543957124378327108758426530950
         62326893 /
         190855198284148052215576171875000000000000000000000
         0000000000
     Truncated floating-point equivalent (9K digits):
         +0.122456310334807969131197468804685308985647976285
         148403097453781086196383230308324792683542470325342
         427755575981222492934062613851285841943688928141765
         072712050034406759799730167457497759516456804496321
         2...E+ 0

t(22) =  2.2
w(22) =  118953497121782882675074274863068198435857582829375
         1786550399 /
         107356049034833279371261596679687500000000000000000
         000000000000
     Truncated floating-point equivalent (9K digits):
         +0.110802789587745202347027350951893026490073843186
         584745859020481170757328121386146215728496485019779
         927271899896045374769998934231469406144886141355042
         330721130342364999647441314009733169388039285440...E+ 0

t(23) =  2.3
w(23) =  193740903814958232671629753103663187245489936023977
         97717781764899 /
         193240888262699902868270874023437500000000000000000
         000000000000000
     Truncated floating-point equivalent (9K digits):
         +0.100258752460079042558561401823852694685670572688
         571793413705858069608232714361039959574894444707249
         514518925165724171083107822883508395465672724625076
         016835370897322180258260481402113843149802273239...E+ 0

t(24) =  2.4
w(24) =  986082152653385093652380308818623511528570669862456
         88223078757657 /
         108697999647768695363402366638183593750000000000000
         00000000000000000
     Truncated floating-point equivalent (9K digits):
         +0.907175988379494482760881586143130893333247804595
         741857956931642623014234612455951382684517548142886
         839191766934951059798361939598643024289954802092142
         817156568766263926277158241395738643370474625854...E- 1

t(25) =  2.5
w(25) =  160604428389076108374812520761628782683669875651665
         0544076124826061 57 /
         195656399365983651654124259948730468750000000000000
         00000000000000000000
```

```
     Truncated floating-point equivalent (9K digits):
         +0.8208493507470648354848066106458661392756610221 5
         2989890678890990085368191455988815282790152114974 1
         7447917654065940627204668745348452997799241544941 5
         8772392988507379721414148714747798100240857039290 0
         4...E- 1

t(26) =  2.6
w(26) =  8174269302612551677183409702209126711481907817860 7
         1414918259327622075 1 /
         1100567246433658040554448962211608886718750000000 0
         0000000000000000000
     Truncated floating-point equivalent (9K digits):
         +0.7427323799705041368857798000755051707283889523 9
         7653223277841484351456166568943495464199708633205 8
         6493783582406250937396866821162599705699304255982 3
         0120122073420422310609377576402029112212186717579 6
         5...E- 1

t(27) =  2.7
w(27) =  1331354642599519908219112687500153273078677109192 3
         4487457303953373555812 51 /
         1981021043580584472998008131980895996093750000000 0
         0000000000000000000000
     Truncated floating-point equivalent (9K digits):
         +0.6720547703991932525479486881698363735912307701 6
         2384979067050052096297119119728864628994346857357 2
         7385143014277978570984599449296358657345869870391 4
         6518681691374569860588588766255190235336486127562 8
         9...E- 1

t(28) =  2.8
w(28) =  6776176849520560378135782526108594975659628831072 7
         5334370906570288151326793 /
         1114324337014078766061379574239253997802734375000 0
         0000000000000000000000000
     Truncated floating-point equivalent (9K digits):
         +0.6080973576937095805097411767680431181364164895 0
         8369152322026998061742534889521851349257073909786 4
         1239495922996002648526281992166234300469315697944 5
         3890297070095070061328032312876821742252203294514 9
         9...E- 1

t(29) =  2.9
w(29) =  1103646670879974450420560664850757748532040863159 2
         0425826452516706422651045329 3 /
         2005783806625341778910483233630657196044921875000 0
         0000000000000000000000000000
     Truncated floating-point equivalent (9K digits):
         +0.5502321173570644130221939913758728083180938678 7
         8088417642060359449698605376589592136469116895768 6
         0294692273362721038356393791307639514800641263654 2
         5696949177568746265826032013003969998399580000050 8
```

```
          0...E- 1

t(30) =  3.0
w(30) =  561720734598581123911423441205278598866965199440059033228205084087186239846855999 /
         112825339122675475063714681891724467277526855468750000000000000000000000000000000000
     Truncated floating-point equivalent (9K digits):
         +0.497867534869821893751955919500134579601542604779190295854628639943699425964486016669197831588235146496284125999442641931276572766623962628260046381739942749571446108989331975592540404165109042281934...E- 1

t(31) =  3.1
w(31) =  914884846623375775014977353224994954093223785623937225577103806298591236271320000990 /
         203085610420815855114686427405104041099548339843750000000000000000000000000000000000000
     Truncated floating-point equivalent (9K digits):
         +0.450492206083746231827378224796155592028278807398167400432078420034832981392920513105729249199345385364520038628348063174211090176319401325525014188037820448230448676534473167345905736980781910579718...E- 1

t(32) =  3.2
w(32) =  465646230862390186457749091883363274868171749534363954572939872432737033220722297 1257 /
         114235655861708918502011115415371023118495941162109375000000000000000000000000000000000000
     Truncated floating-point equivalent (9K digits):
         +0.407618993693257122126014201931014231722467695075508533481323657867875246406183171293664096625508344618536015288176925941778639523210499700967766985006045776732035406046196916435745842485666567497469...E- 1

t(33) =  3.3
w(33) =  75840800592866883432620585307590648631837930360858383951956870328040896829451466171197 57 /
         205624180551076053303620007747667841613292694091796875000000000000000000000000000000000000000
     Truncated floating-point equivalent (9K digits):
         +0.368832111036806757604161977413246712367602877955829869846496533783357746150216499482145177922676658013289900821169818479910746284371292046922192954774337861819384353862351408940532496783427345933802...E- 1

t(34) =  3.4
w(34) =  3860038338110967712709802929978181238324696447213586289047058788995602276785294471266055 51 /
```

```
        1156636015599802799832862543580631609074771404266357421875000000000000000000000000000000000000
    Truncated floating-point equivalent (9K digits):
        +0.333729737449792915501691672905933222482961615466830083222040493435346196545710180003425274218109054922434840974037448270317433090736160403932443689036237870249857577940021108781119215248595965249336...E- 1

t(35) =  3.5
w(35) =  628694279324082487036382072773965745163167836916167188961099724099987613897157001071138866051 /
        208194482807964503969915257844513689633458852767944335937500000000000000000000000000000000000000
    Truncated floating-point equivalent (9K digits):
        +0.301974514811701683968001746606633975650204187462358604690095221837252274091588664853701698517370529871343376539439824566261166874443545450262419017668257444038384027408721484744582871930074676325928...E- 1

t(36) =  3.6
w(36) =  319983071272538401639121327740941677160962791506550803410528358687967053437907934515940334731937117109396579480033483077332537538950418820604681968688964843750000000000000000000000000000000000000
    Truncated floating-point equivalent (9K digits):
        +0.273234326722340907352335180975523013494098535744302301443698516670520173430381469195998817515059005714209280954470245133621747945798325738580499820905056101662968515639771530768342048276459070999963...E- 1

t(37) =  3.7
w(37) =  521166232196584798399631122097051603185085740297094937778566527314070963344166889799748495572996932107969138430640602695391985675701107538770884275436401367187500000000000000000000000000000000000000
    Truncated floating-point equivalent (9K digits):
        +0.247236177558361796987462940392151677567303774360111598300344062419938052363302495187685389122689825475080784799346028501313057577259387110033369611364893811398228248879942245494208262781074667878043...E- 1

t(38) =  3.8
w(38) =  265254160964074197018785929548452202135662861338135894075220167687502212017942231783065642868778079911857326403672353390161579919425818729905586224049329757690429687500000000000000000000000000000000000000
```

```
Truncated floating-point equivalent (9K digits):
        -0.223704865611122811679132979096174463254515660291
        030289312737697130305600691535418646045630830995034
        540659879102376524713322152905201958094190112766556
        854889564822795056175280849506638771017679040838...E- 1

t(39) =  3.9
w(39) =  432029333089907610578489969350529625596691272839980
        775212889601449998942568087729111178677793706719529
        9 /
        213431875266102361022908438549664737138300552032887
        935638427734375000000000000000000000000000000000000
        00000
     Truncated floating-point equivalent (9K digits):
        -0.202420248873914708914287379744929975252473308117
        049902217686741617020799148737095280150417679752556
        824932695698022285709251295237618989081267085424027
        581131701953041985105423162943180282629016231148...E- 1

t(40) =  4.0
w(40) =  219885795539809691053444177442261316505790874729120
        807925090820555536214176445412956693523252268200464857 /
        120055429837182578075385996684186414640294060518499463796615600585937500000000000000000000000000000000000000
        0000000
     Truncated floating-point equivalent (9K digits):
        -0.183153561515722860745620690510807167739832250472749405821713030725535799900228993068577986785599068866881512482489150307002990412817438063543973389567683016726460633709069308735918352931112983827226...E- 1

t(41) =  4.1
w(41) =  358138049628700449644014294200961693262286703196999013683053964074198374955599908801776360424326395024913357 /
        216099773706928640535694794031535546352529308933299034833908081054687500000000000000000000000000000000000000
        0000000000
     Truncated floating-point equivalent (9K digits):
        -0.165728100259097006777858365714337556845330760919935751371084931670582561898459873091630367292939871121526489516985717702771635828295917258569870634853125318838999420308596298625173048692453305848344...E- 1

t(42) =  4.2
w(42) =  182276989297723994361039269312604310212135335710019346832369543880681525409960274383847715590243560969680303 51 /
```

```
        12155612271014736030132832164273874482329773627498
        07070940732955932617187500000000000000000000000000
        000000000000
    Truncated floating-point equivalent (9K digits):
        +0.14995294785139418440836410076380822977822444482
        50818575848903725335952823926829293488730775955863
        30546097875751606762586031680288306279777659653165
        17508717682424212689076313431011584273768841721998
        5...E- 1

t(43) =  4.3
w(43) =  29688484676913446869363708855242806795097138297216
        47210506730561134656091627516525098261070476403557
        3690833190851 /
        21880102087826524854239097895692974068193592529496
        52727693319320678710937500000000000000000000000000
        0000000000000000
    Truncated floating-point equivalent (9K digits):
        +0.13568713965658911203824989950534262286359973321
        03618786865499234447283215564727567738024719431916
        39129184498962721020835714637223542145380993303297
        42101199165890163731733146149073993021779277057745
        7...E- 1

t(44) =  4.4
w(44) =  15110058150625155043754382154249237961477385789498
        54505763441620072657986936631855852052907552220143
        351810410339593 /
        12307557424402420230509492566327297913358957978 41
        79659327492117881774902343750000000000000000000000
        00000000000000000
    Truncated floating-point equivalent (9K digits):
        +0.12277056794930052597912618747472226516642986407
        91570689910430663554250322621325302779265685276842
        72859353181176364348109239421260975958001049166110
        13585184901434986095450715750280142380894623993311
        3...E- 1

t(45) =  4.5
w(45) =  24610817026939628935773590269990045413101391469484
        71013025009095611853739368857096980447310881808025
        233192489658866093 /
        22153603363924356414917086619389136244046012436115
        23386789485812187194824218750000000000000000000000
        00000000000000000000
    Truncated floating-point equivalent (9K digits):
        +0.11109171100813639487488160201015350759344788546
        80426417056688725571982313963713680189191540329775
        72707054675700518755138999072246461959249207638797
        23020879504047287543974717791821417264792718734430
        9...E- 1

t(46) =  4.6
```

```
w(46) =  1252564026567184138866909539285351857314939374768810446656098905055724838469873028671435718528457164053250861085038359997 /
         1246140189220745048339086122340638913727588199531481905069085769355297088623046875000000000
```

```
          2...E- 2

t(50) =  5.0
w(50) =  860727089164466371711018788304643264723128225479557
         542637595998290527005363712009478161694274902631960
         353056818728478185109924951 /
         127748840335582941596136625760577061140104784017590
         457467434799582406412810087203979492187500000000000
         000000000000000000000000000000000
      Truncated floating-point equivalent (9K digits):
         -0.673765090080994600788044221074062960629874511430
         775093691228066011097022080661110881338624033589590
         376760164884479515026793364207517287418639170250530
         144941992231411557790023455520924764324950718664788
         2...E- 2

t(51) =  5.1
w(51) =  140198713826572941122104309529596450879160791897290
         787533364244346785463503102928923771750268737130410
         297442444374795590413380630555651 /
         229947912604049294873045926369038710052188611231670
         023441382639248331543058156967163085937500000000000
         000000000000000000000000000000000
      Truncated floating-point equivalent (9K digits):
         -0.609697701705095170950823334824382579998572124500
         378479885531126548226764497206220479375211070822960
         322639533772745187766030739347527097010528551224260
         210922331008849110660199571212585769552182342376307
         7...E- 2

t(52) =  5.2
w(52) =  713504646706404351008633651644652683097150586579760
         514238705824432535140203098246911110429539191701020
         414710367535148499226793537925993 /
         129345700839777728366088333582584274404356093817810
         450685777734577186492970213294029235839843750000000
         000000000000000000000000000000000
      Truncated floating-point equivalent (9K digits):
         -0.551626101272768409514781873509227650178745835680
         351984250943093541941730111613409987386484123117400
         465783765838583150770064451057886727596333118006100
         618594286082613868148267920974070039073692111493899
         7...E- 2

t(53) =  5.3
w(53) =  116221205043707127072873887510021367347751173910730
         811925452273945204019842919216445460460860714759110
         209963712560646468844120512910711524930 /
         232822261511599911058959000448651693927840968872060
         611234399922238935687346383929252624511718750000000
         000000000000000000000000000000000
      Truncated floating-point equivalent (9K digits):
         -0.499184245909906833450655447933357896521721885060
```

```
        3859441603855783118599548790108365187229976474521 3
        7957068855027083788956520157088504324720945318892 6
        9530753550420822316873202083910040554443622893708 26
        6...E- 2

t(54) = 5.4
w(54) = 5914619599105477058676436743799495490726241790270 2
        7657408000218077264580312565049942064015868618121 9
        4833628870476527084645564291144408511 99 /
        1309625221002749499706644377523665778344105449905 3
        7188193499562594013241323409602046012878417968750 0
        0000000000000000000000000000000000000000000000000
    Truncated floating-point equivalent (9K digits):
        +0.4516268856350208895969921692631424805171704269 1
        9637418263343791871623630930731286626384027574043 6
        5416976459595080513132080376058815064612654954710 7
        5469970342932876066877789434753612698385752548036 1
        0...E- 2

t(55) = 5.5
w(55) = 9634478571545735276553924491398510586605086083697 4
        9368547797955091484108854782467079861671514463839 9
        1270560805887937410436994500947235244656 99 /
        2357325397804949099471959879542598401019389809829 6
        6938748299212669223834382137283682823181152343750 0
        0000000000000000000000000000000000000000000000000
    Truncated floating-point equivalent (9K digits):
        +0.4087038039176514114233311665512870270792898528 8
        9619444842067458899969635164176829783516115496939 0
        3205101229015631821663721542444922518970965972562 9
        7719118599822115981306313500319822633543570896419 9
        2...E- 2

t(56) = 5.6
w(56) = 4902923713481982176316154658152764898945379643096 4
        9944743650813347325297353882101474323891670629919 2
        3043240991727276642117228851063366443129205 7 /
        1325995536265283868452977432242711600573406768029 1
        8903045918307126438406839952222071588039398193359 3
        7500000000000000000000000000000000000000000000000
    Truncated floating-point equivalent (9K digits):
        +0.3697541642780525941041599058959191941797783624 4
        3082118501435651740439714004958148008862952798139 5
        3617835219386922983807419576502716155720754032224 1
        8870134195975973446778181976488741971731803462113 0
        3...E- 2

t(57) = 5.7
w(57) = 7986803024903684365744064330092680287352361729392 9
        4541046117536718802351088590271684146200198343191 7
        9952164796675298687801972706214117232643634055 7 /
        2386791965277510963215359378036880881032132182452 5
        4025482652952827589132311913999728858470916748046 8
```

```
        75000000000000000000000000000000000000000000000000
    Truncated floating-point equivalent (9K digits):
        +0.33462501722369688207355017187686224294557282541
        35850645100674480598438429637382459476960869259047
        57278265175357221264563476683164600919593308452237
        04768209263638686142232637365232833164094974912400
        3...E+ 2

t(58) =  5.8
w(58) =  40642581222718310406470245534820251575553982508510
         84862410577970288571190915272207670726483660087136
         87887028029445209696062867611130915947389919999951
         /
         13425704804685999168086396501457454955805743526295
         53893339922859655188869254516248474828898906707763
         67187500000000000000000000000000000000000000000000
         00
Truncated floating-point equivalent (9K digits):
         +0.30272214244225564478382415945574830305655901271
         65726893421688290439659222997183874100110705380463
         42385113845214967388816241949530039875201279115336
         37397623427478115177344631978984366846651711887792
         0...E+ 2

t(59) =  5.9
w(59) =  66209481943448930288240517405953864470718986808722
         47055976859235434140571564178451630945612980307884
         06613289343286170698042861104649414286842063229604
         51 /
         24166268648434798502555513702623418920450338347331
         97008011861147379339964658129247254692018032073974
         60937500000000000000000000000000000000000000000000
         00000
    Truncated floating-point equivalent (9K digits):
        +0.27397478239875971785559943026327515021065665410
        75162202912721806820277926290480243237211017056103
        05315268155601712844188920902955467185679668521698
        50149567282405551534068028538520596667729699500832
        3...E+ 2

t(60) =  6.0
w(60) =  33690281960611216717529557776002161393768860244942
         25727370215714996108695308424439849869867825436397
         59243652638846845278547726564621615794830210847223
         2393 /
         13593526114744574157687476457725673142753315320374
         23317006671895400878730120197701580764260143041610
         71777343750000000000000000000000000000000000000000
         0000000
    Truncated floating-point equivalent (9K digits):
        +0.24784063881753366357380910010814281749061976831
        39181658430368797199477058451821181112538931182031
        44758971235683945327933320364085004630854117486001
```

```
        897185508532032704742991540038459943735643054571110...E-2

t(61) = 6.1
w(61) = 5488713409386993852531991663579129829197403286045268149677475176080629716479086984837276847739433027035419099915579836337774547753281653961691877506850158893 /
        2446834700654023348383745762390621165695596757667361970612009411721581714216355862845375668257474899291992187500000000000000000000000000000000000000000000000000000000
    Truncated floating-point equivalent (9K digits):
        -0.2243189295917658711541628616835826707328494841614169353080632279250648193906453239519236044492744446786115553949828305696650403146848395074892896025215242475612789519170353933422552809362786889361633...E-2

t(62) = 6.2
w(62) = 279270101106732151364438946964971705120157609180894447960120577881024538422968225752500989499406186699450410125344168372336894451502811001818912797975906826399 /
        1376344519117888133465856991344724405703773176187891108469255294093389714246700172850523813394829630851745605468750000000000000000000000000000000000000000000000000000
    Truncated floating-point equivalent (9K digits):
        -0.2029071189862542038722027898261379699275400666501526488307021876488941090691291895266887382661651974758148064293591756171966332445687216935843523852192717922402512102913830692315111749884739272127334...E-2

t(63) = 6.3
w(63) = 4550142055313759521386000627564396146838287622725774070810048353740144473732378484690856424671683379547663226171699836189945107337632639448534629856591484625408992 /
        2477420134412198640238542584420503930266791717138203995244659529368101485644060311130942864110693335533142089843750000000000000000000000000000000000000000000000000000
    Truncated floating-point equivalent (9K digits):
        -0.1836645303762069459541564618338022536181871381580620311397224888708097124938293305895056676233760921922230429931897878000392290349043962249684626108801574615632670331693474515468369137963833261912...E-2

t(64) = 6.4
w(64) = 2314939617408120332422536568960916065726997700871 4
```

```
        00524525034473187096988541684256920624882623030482
        65144696208228824934562728939112854702461476286124
        77309572625657 /
        13935488256068617351341802037365334607750703408902
        39747325120985269557085674783925011155361062265001
        23739242553710937500000000000000000000000000000000
        00000000000000000
      Truncated floating-point equivalent (9K digits):
        +0.16611029990241008852053458017141556265906326785
        46700410076806390913485533714741023915173162296989
        33225762755176603136964099512027303248377067546801
        95698693026119800868209731493638705337879985065249
        1...E- 2

t(65) =  6.5
w(65) =  37721115509329056872142103767368250864317147487173
        68424010361448221521734732144517171765738546386568
        68497178297672404811109719527721258831254363050345
        74637943327974157 /
        25083878860923511232415243667257602293951266136024
        31545185217773485202754214611065020079649912077002
        22730636596679687500000000000000000000000000000000
        00000000000000000000
    Truncated floating-point equivalent (9K digits):
        +0.15037991420095815864261373270775092472552201294
        83526527839368493859215348143164162447975227964094
        48396252509703351191968168721937645290536860777498
        76571949654800932136098954753851684192017849280522
        0...E- 2

t(66) =  6.6
w(66) =  19188853779870176159856112827074530516186474507415
        11627932392950195050847064532263787780203341640577
        16298140324777292144192768197494939479703327853829
        42443995923585447 51 /
        14109681859269475068233574562832401290347587201513
        67744166684997585426549245718724073794803075543313
        75285983085632324218750000000000000000000000000000
        00000000000000000000
    Truncated floating-point equivalent (9K digits):
        +0.13599777777600205815808597373896005847178846624
        18763057611635784482047663870776589006133961052483
        00851548443486307510746888236605021336022871702943
        51679563354984834316006603477209937356929285368382
        6...E- 2

t(67) =  6.7
w(67) =  31271644900828780930280503844314970975780588178680
        81960106582057463793132096388232696698768941955038
        59029906171646928376884052335976881683960354247913
        70244589437953024052 51 /
        25397427346685055122820434213098322322625656962724
        61939500032995653767788642293703332830645535977964
```

```
        75514769554130183593750000000000000000000000000000
        00000000000000000000000000
    Truncated floating-point equivalent (9K digits):
        +0.12312918341672289688926302916945114572909849665
        80875005105652902540173470733358128584143587544878
        68128689645161187718636772565674835716409668263967
        60027488456710315648293143159859004549861669331486
        4...E+ 2

t(68) =  6.8
w(68) =  15905630526860301689190388384823002120255489679900
        68754153221630148243892382631996607698534920163691
        82932645328790389057393402955033904861271291817216
        81294983303926906925879 /
        14286052882510343506586494244867306306476932041532
        59840968768560055244381111290208124717238113987605
        17477057874202728271484375000000000000000000000000
        00000000000000000000000000
    Truncated floating-point equivalent (9K digits):
        +0.11133677480875575614092197427497824759215784245
        30064452489171356111581116626288331400018372690252
        16552694276914908883607010055510054379589896565556
        27303375516355693995661540300645832838795552673532
        8...E+ 2

t(69) =  6.9
w(69) =  25925396993804275231434071947746578445075141707513
        89612961791620010608837744592090215267410766827803
        20008826392698590746904448624210950419620529482844
        36163101764147573885188529 3 /
        25714895188518618311855639640762051351658477674758
        67713743783408099439886000322374624491028605177609
        31458704173564910888671875000000000000000000000000
        00000000000000000000000000
    Truncated floating-point equivalent (9K digits):
        +0.10081859872942295080478387861180697910787363010
        29460938187133614639303401621794593417417893246726
        36111631428835454104113766622102213835409757168391
        36045783292083495927058726275519659594811113940181
        1...E+ 2

t(70) =  7.0
w(70) =  13183876340778794181990312792770706392585730445213
        76783088738091092147943078362428356385564773995053
        82583725250168348731800349689449125860212181125999
        36045755171188824620709761599 /
        14464628543541722800418825422928653885307893692051
        75588980878167055934935875181335726276203590412450
        23945521097630262374877929687500000000000000000000
        00000000000000000000000000
    Truncated floating-point equivalent (9K digits):
        +0.91145626734156484175860185265328431296006643309
        75781101012884213765413598516314087646158184165661
```

```
        80106817640393783444106848664705635431217209088170
        39303516393942410579094039982929591994101318306439
        7...E-3

t(71) = 7.1
w(71) = 21493690194818993636471769729040647803710389834963
        56181602913235054200516043858031646256121450975377
        18400565936416048041349319143654003508586284473560
        74889554635198042041010817576099 /
        26036331378375101040753885761271576993554208645693
        16060165580700700682884575326404307297166462742410
        43101937975734472274780273437500000000000000000000
        00000000000000000000000000000000000000
    Truncated floating-point equivalent (9K digits):
        +0.82552683334914564790773287830249279123477919398
        15540995616191009512795092356677365683586258268725
        26044859162884183545536423257440549201996067673831
        46599081086240983965150019271117892277633380540093
        2...E-3

t(72) = 7.2
w(72) = 10927555315487490448824870543471858884517007941290
        99677419288608476312782902093778308606800873528501
        88391172903445667872647626924446049342528751554296
        06830381846158969264041713232392 57 /
        14645436400335994335424060740715262058874242363202
        40283843139144144134122573621102422854656135292605
        86744840111350640654563903808593750000000000000000
        00000000000000000000000000000000000000
    Truncated floating-point equivalent (9K digits):
        +0.74614064182046438846598804448257345161789422112
        10025979195959575992316736792263558807110273571142
        61087335646189279520826055391018777295295950105212
        07011642359950120925405413865221828497492251495601
        1...E-3

t(73) = 7.3
w(73) = 17820114652956167687869932546683637565890124243040
        07844140326405013503562186225669496785708024983402
        83402183367846236073987295954267615133078714353523
        85778073310478611925807016622688877 57 /
        26361785520604789803763309333287471705973636253764
        32510917650459459441420632517984361138381043526690
        56140712200431153178215026855468750000000000000000
        00000000000000000000000000000000000000
    Truncated floating-point equivalent (9K digits):
        +0.67598284035152639230282184762843856822159230184
        04115066539596590050303737513045794180636940577839
        43244923950174464089356387933151548316313620795491
        39188770466551965640766271970100876420245390827345
        1...E-3

t(74) = 7.4
```

```
w(74) =  90570567726854766294636729629974106856255296307719
         11925456733547530248765094201818311034848328705053
         79058719942027151051876806679634345576509061352586
         66395992122370949421829488324084129551 /
         14828504355340194264616861499974202834610170392742
         43287391178383445935799105791366203140339336983763
         44079150612742523662745952606201171875000000000000
         00000000000000000000000000000000000000000000
     Truncated floating-point equivalent (9K digits):
         -0.61078693815966380489348156337149433600150477582
         62080561628676862652058031382225677048989004565634
         92514839479190706724520593829141825618865132811960
         17418474609260711248870716557013178553509815019617
         1...E- 3

t(75) =  7.5
w(75) =  14775012685045816652007073186489797132250094457496
         15683404462593372926781945647945810513817214720699
         51944629481350139962056490404885389050692128320057
         31476053385300795610875814628701656089 0051 /
         26691307839612349676310350699953565102298306706936
         37917304121090202684483904244591656526108065707 74
         19342471102936542592942714691162109375000000000000
         00000000000000000000000000000000000000000000000000
     Truncated floating-point equivalent (9K digits):
         -0.55355146978292095028518855883934026480403352938
         97582891873289760765652464772362410479482351632850
         24439772254960794975596490207389211238796873671758
         62049696415654637814976460005011245379544828633384
         1...E- 3
```

Decreasing h from 4/10 to 1/10 improves the accuracy of the numerical approximation considerably over the interval x = 0 to x = 7.5. With Runge-Kutta starting values, the graphs of $y = e^{-x}$ and w_i are indistinguishable at any reasonable plotting scale (see Figure 3). However, beyond x = 3.5, the curves e^{-x} and w_i using Euler starting values begin to diverge (Cf. Figure 8.13 in Young and Gregory, [14]).

Figure 4 depicts the approximate absolute error $|e^{-x} - w_i|$ for each starting method and h = 1/10. From initial maxima in the Milne-Simpson portion of the approximation, the average absolute error declines rapidly and then (beyond x = 1.6, Euler; beyond x = 3.0,

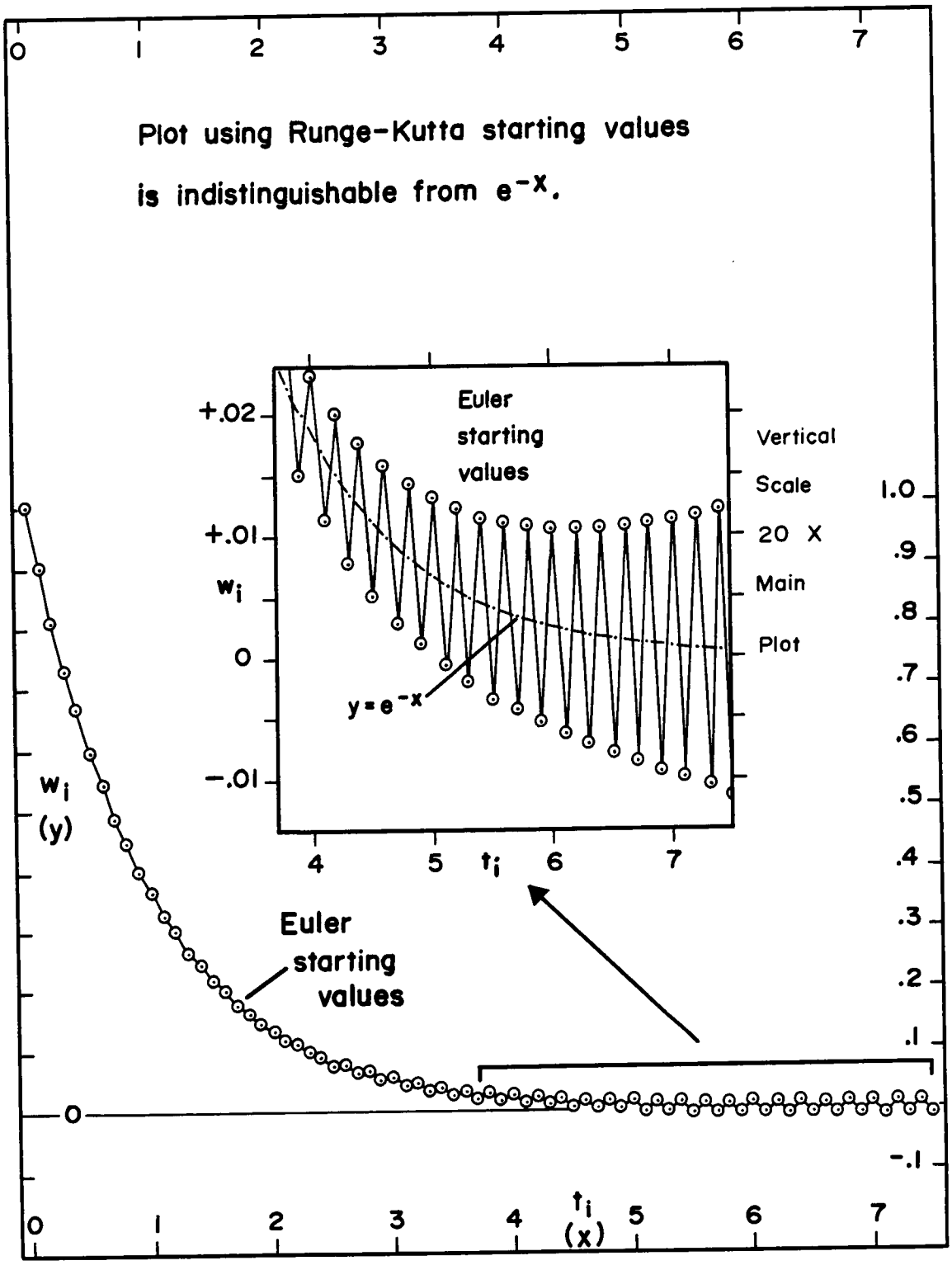


Figure 3. Milne-Simpson approximation to the initial-value problem $y' = -y$, $y(0) = 1$, where step size $h = 0.1$.

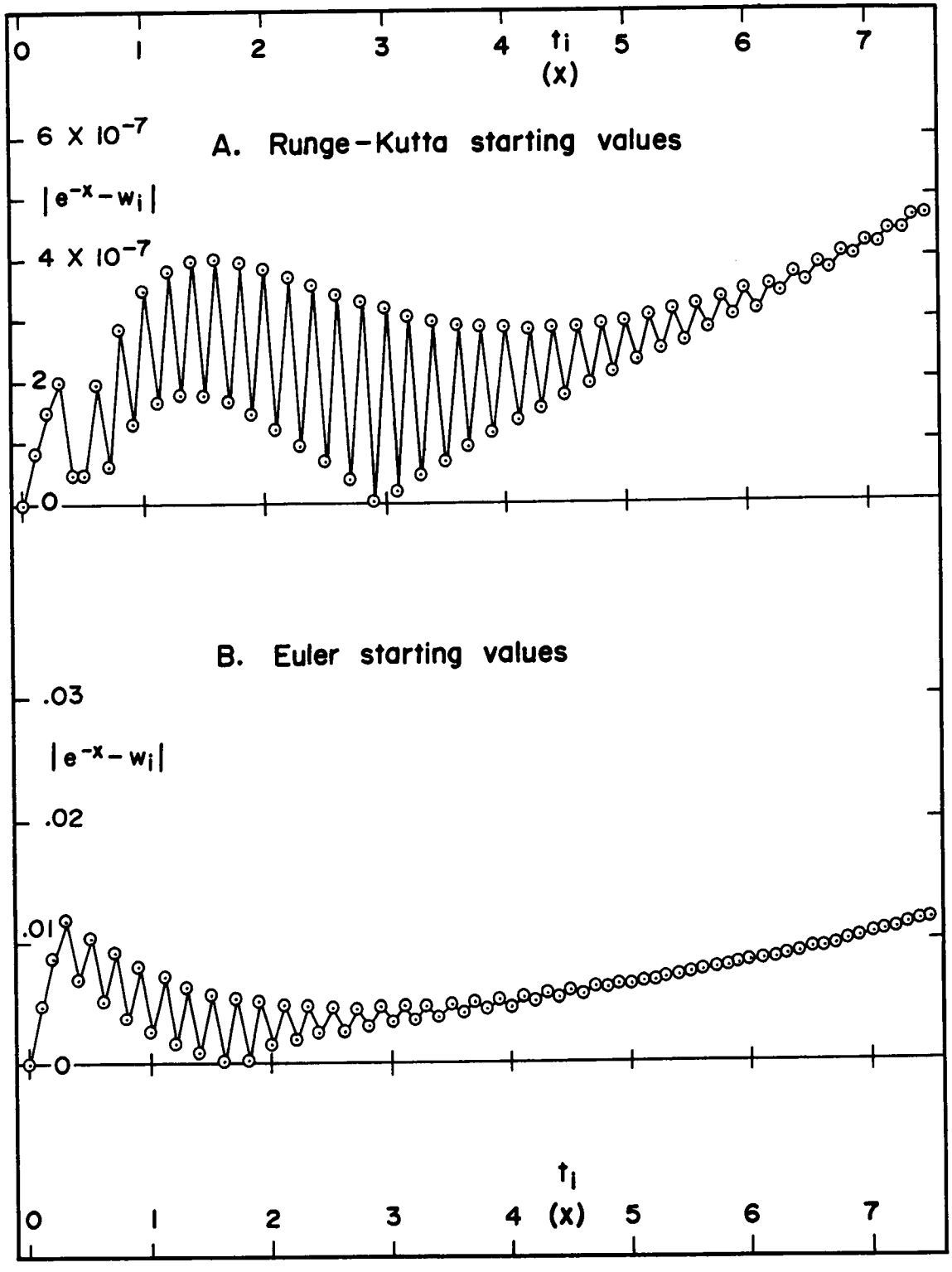


Figure 4. Absolute error in the Milne-Simpson approximation to the initial-value problem $y' = -y$, $y(0) = 1$, where step size $h = 0.1$.

Runge-Kutta) enters a longer period of gradual increase continuing at least through $x = 7.5$. Note the oscillatory behavior of the absolute error. In this instance ($h = 1/10$), the inherent instability of the Milne-Simpson remains; freedom from roundoff error appears to delay, but does not prevent, the onset of instability in the Milne-Simpson numerical solution.

Maclaurin Series Approximation to e^x

The following theorem forms the basis for numerous numerical approximation techniques and will be used here to compute an approximation to e^x.

6.35 Taylor's Theorem. Suppose the first n derivatives of function $f(x)$ are continuous on $[a,b]$, and let the $n+1$ derivative of f, $f^{(n+1)}(x)$, exist on $[a,b]$. Let $x_0 \in [a,b]$. For every $x \in [a,b]$, there exists a point $\xi(x)$ between x and x_0 such that

$$f(x) = P_n(x) + R_n(x) ,$$

where the nth-degree Taylor polynomial $P_n(x)$ is given by

$$P_n(x) = f(x_0) + f'(x_0)(x-x_0) + \frac{f''(x_0)}{2!}(x-x_0)^2 + \ldots + \frac{f^{(n)}(x_0)}{n!}(x-x_0)^n ,$$

and the remainder term (truncation error) $R_n(x)$ associated with $P_n(x)$ is

$$R_n(x) = \frac{f^{(n+1)}(\xi(x))}{(n+1)!}(x-x_0)^{n+1} .$$

If we consider $f(x) = e^x$ and let $x_0 = 0$, we get the Maclaurin series expansion of e^x with the approximating Taylor polynomial

$$(6.36) \qquad P_n(x) = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \ldots + \frac{x^n}{n!} ,$$

with truncation error

$$(6.37) \qquad R_n(x) = e^{\xi(x)} \frac{x^{n+1}}{(n+1)!} ,$$

where $\xi(x)$ lies between 0 and x. $R_n(x)$ can be used to compute a bound on the error of the $P_n(x)$ approximation.

Appendix K lists assembler programs designed to compute approximation $P_n(x)$ and bound on error term $R_n(x)$ for a specified number of iterations and designated value of x. For positive x, $\xi(x) \leq x$, and hence

$$(6.38) \qquad R_n(x) \leq \frac{e^x x^{n+1}}{(n+1)!} .$$

For negative x, $\xi(x) \leq 0$ and consequently

$$(6.39) \qquad |R_n(x)| \leq \left| \frac{x^{n+1}}{(n+1)!} \right| .$$

Table 10 presents execution statistics for Maclaurin approximations to e^x where x = -326/917, +326/917, and +1/1, with "results" reported for n = multiples of 5. Reported results include x^n, n!, $x^n/n!$, $P_n(x)$, and $x^{n+1}/(n+1)!$. At each iteration of the calculation we perform 2 multiplications, 1 division, and 2 additions, plus, if results are reported, 5 calls to procedure RATMAP (residue arithmetic only) and 5 calls to OUTPUT.

Using Mersenne prime moduli, residue arithmetic execution times are greater than rational arithmetic execution times by a factor ranging from 5 to 9. Furthermore, the growth of execution time with n is much more dramatic in residue arithmetic calculations. Computer storage

TABLE 10

MACLAURIN SERIES COMPUTATION OF E^X *

	x = -326/917			x = +326/917			x = +1/1			
# Iterations	25	50	75	25	50	75	25	50	75	100
Num digits	91	197	311	93	199	308	23	64	107	157
Den digits	91	197	311	93	199	308	23	63	107	157
Rn bound (sign. digits)	38	89	145	38	88	145	26	65	110	159
Residue arithmetic:										
Minimal K	41	69	----	41	69	----	17	17	41	41
Exec. (sec.)	67.7	412.2	----	66.9	412.1	----	7.1	31.8	185.1	353.3
Main bytes	3896	5128	----	3896	5128	----	2840	2840	3896	3896
Asm. bytes	30136	37696	----	30136	37696	----	23704	23704	30136	30136
Rational arithmetic:										
Minimal K	12	23	34	12	23	34	4	8	13	18
Exec. (sec.)	8.7	57.3	196.7	8.6	58.2	195.8	1.4	6.4	20.7	50.3
Main bytes	2632	3248	3864	2632	3248	3864	2184	2408	2688	2968
Asm. bytes	25192	28408	31616	25192	28408	31616	22856	24024	25488	26944
Res/rat exec time	7.80	7.19	----	7.74	7.08	----	5.09	5.00	8.95	7.02
Res/rat tot. memory	1.22	1.35	----	1.22	1.35	----	1.06	1.00	1.21	1.14

* Reporting interval is multiples of 5 iterations.

requirements are also approximately 10 to 25 percent larger for residue arithmetic.

The overhead associated with printing intermediate results (converting binary numbers to character format, computing floating-point equivalents, and the actual printing) is illustrated by the computation of the first 75 terms of the Maclaurin series approximation to $e^{-326/917}$. If we report only the final results (iteration 75),

execution time drops from 196.7 seconds to 147.9 seconds, a saving of 24.8 percent.

When x is negative, we may compute $P_n(x)$ or we may approximate e^x by computing $P_n(-x)$ and inverting the final result. The two approaches are illustrated for x = -326/917 and x = +326/917. For x = -326/917 we get the machine results listed below for $P_n(x)$ and term $x^{n+1}/(n+1)!$ (the absolute value of the latter being a bound on $R_n(x)$). Truncated floating-point equivalents derive from the rational computation with K = 34. For convenience, the p_n term is denoted p(n) in the output.

```
    P(5)   =  6816187328520113 / 9726082238465355
        Truncated floating-point equivalent (9K digits):
              +0.700815308918837160660873545566730297597838820 81
              970091969701974449975788666555948656895867850132 63
              049653863411382798796379656692977958846212178404 68
              777369991291256752758567124694307668642178352806 46
              113457525849050092420009752979623773709877115557 37
              311614635749428214034891942423036295361309658378 41
              373842651...E+ 0
Error term =  75021478312036 / 267564522380181916 05
        Truncated floating-point equivalent (9K digits):
              +0.280386493861985654170580693578709805084353353 90
              283361273875081085913934674822567283159546378530 56
              153671005953282228184759882710850111816452767102 15
              056497206304553834329584934374087654447871261726 38
              836950562017648557281469542427649395909594786133 36
              196204939486872580840170126867476108658200015705 84
              729273192...E- 5

    P(10)  =  139219606788344255896876863512653 7 /
              198653018989721626352285624079652 5
        Truncated floating-point equivalent (9K digits):
              +0.700817976471565855830697038321813197267600471 59
              386842680623043027605989727040819745494686928096 61
              378976401177541418392948700042861037730593029410 19
              322060087829458028020145585007433074694086639376 02
              934945207388186698992164082669865696310437457490 81
              479539921445256945388482257292037517403035619409 79
              028757132...E+ 0
Error term = -17264485300991680723259096 /
              601143900764796613504651527027436430 25
        Truncated floating-point equivalent (9K digits):
              -0.287193886173130752334241761079861014377119377 35
```

```
            2020567599236625625125379080369065083158901692011 3
            2892927615127814235902485767938489175135368920012 4
            6994702981875888733097301155099668004644866112324 9
            2230913964295932960645878689271864024040282229240 2
            8711268046934518388563145621346386805574770795291 3
            649087734...E- 12

    P(15) = 1219873865557082983711859537869038691543375052434 3
            5743 /
            1740642943691759258404513121707364557688943813743 9
            7375
        Truncated floating-point equivalent (9K digits):
            +0.7008179764712869433277738892579367604625333463 0
            1819625864219174422498516343326078234193468883720 2
            1813944492722663316981939148788552053626333867745 2
            9671445755671781083264478320329226102821612539557 4
            5005495661798540839263393751880416121483608475063 3
            5946388105173703261936520660866393942706654767274
            642095709...E+ 0
Error term = 4966285312794523347175024712525932 82 /
            1596169579365343239956938532605653299400761477203 2
            2392875
        Truncated floating-point equivalent (9K digits):
            +0.3111376997154136837679476937305564225840209210 8
            4246660747806685907983373633690269710755891190109 9
            3779251104463997723847478274327517138401054936949 0
            4539976748741144103503810518319844177791112098845 6
            3465862000706311748341104990353936733437008769648 9
            5312606332390266408623689236816428603711921190576 0
            608956955...E- 20

    P(20) = 1149679113731697179097224501070921091439632210297 4
            7217761498657465208999 9 /
            1640481768918780617341860807819586921335677329450 1
            6457815044789428938812 5
        Truncated floating-point equivalent (9K digits):
            +0.7008179764712869433308214620294533495302642577 7
            1666512854209643470009455626717181836954151905345 6
            2326669246108856700090390221560041508777561881786 1
            6128997790863182017236270806419261574559981072065 3
            2265806167722928041528537686268478635554320084382 8
            3602155471766950272987077991984582276044679378607 3
            518252121...E+ 0
Error term = -228575500543065801026827831766458039682730020504
            /
            3159075742406895834815221357618178534416113833322 1
            8192814431751003307471231 25
        Truncated floating-point equivalent (9K digits):
            -0.7235518207895655412607813980689414892260896907 3
            6048100572219179915713104942887809591269317955499 3
            0405080074450488198695442238154537891522369604318 9
            7411616642207517040261842412499934221083436729251 1
            4692200858468869877300040748240917824009497927302 4
```

```
            2766837317495038190524076214540611946135246700805 8
            314341220...E- 29

    P(25) = 582444063522611714336141149326464848634560147771 48
            702993167773223722984990824150893736266 31 /
            831091785709173942510159692711040791560937895692 20
            569412358872174317129894885762743442656 25
        Truncated floating-point equivalent (9K digits):
            +0.700817976471286943330821462022332972200449596 19
            980096732206697128238466986223168536010946953392 24
            865465279488128977860121213789383560041054318799 77
            148669775140004906227878668522869310403056793713 87
            672874676789311297844158664351155066867700315934 47
            410294145038728592652010427388548718419857522047 53
            334397840...E+ 0
Error term = 263007238598993618585009538034697580905821443000 31
            793849672 /
            505279704049392191001844298537224181086094973381 88
            598806201235874691759779323592060893575162968 75
        Truncated floating-point equivalent (9K digits):
            +0.520518114009352903828371093302682716829565010 62
            636000609445025195664146746534984325981498608015 63
            872124706885449377390562470770189777240103066256 49
            841812806795731725526144408092820803300117123506 86
            875594761424999951429407591902574274677280031990 19
            308413909831516437185246702966193325210158525731 91
            856266897...E- 38

    P(30) = 205856929661454827085417942605955268020877885558 02
            182042284530404049857095570851522737985979037934 65
            31833222681539 /
            293738084028569357535631339198901130213248095226 32
            913554024742879037789686968739075370143794332682 66
            87421541328125
        Truncated floating-point equivalent (9K digits):
            +0.700817976471286943330821462022332972205587100 73
            029149692139226480662485687973555152081172617836 44
            177840163939576698604847465386525495700918755542 85
            191519502425013465935493260909885664069962994794 44
            584683362747887056774885969690937813523764138648 31
            509725968508709590913098847870106697074315569215 42
            843437572...E+ 0
Error term = -121050256726766118296265380099707141182663790266 90
            494480971926876639584 /
            835009251468014112666539207940716242857200360299 88
            583360026136582240724743146034569554707764149517 02
            243332155334609375
        Truncated floating-point equivalent (9K digits):
            -0.144968761141208832171842678994507259704253788 00
            112713554125905238714314699610019672761754449449 97
            419186899179777435849435672849482088959969825990 39
            946131277271838525996347678698815452955746963220 47
            823738102937744221987391339017567067140664418492 40
```

```
                3813416798509391315043553262068204990416626765773583395961
                5...E- 47

     P(35)  =   8124652013618652951570490702815122004700039819660605820480079
                7660362872920988835707812571502100460584366097042359399652659
                978766226929 /
                1159309875943447100378641054386007474714226445769507606887301
                5110142619837050261772221091655918420809281972800484746559914
                7205648828125
          Truncated floating-point equivalent (9K digits):
                +0.7008179764712869433308214620223329722055871007288577430471
                2542985636298793828249016216496456624637570760923397506454701
                4958511667188555024262205811076894368954480477293941601074356
                1225115605041358396989537453136794335710722820283117460582990
                9583301907420282135558077513839327347589542463850558816038125
                3799...E+ 0
Error term =    1741060047870681415433639414111441956186795059236752665808573
                5557093202451385376495677844061612689194249246218477196888165
                1085693574627964089937040070415151758104061406694362947269390
                0041215224006133589761888219778515625 /
                Truncated floating-point equivalent (9K digits):
                +0.1819710785654313724747200615545642428940465026271979536988
                9956769400657108217139768631901086980564405814109641081824092
                7269498323996589843408145400504368519251680993723556198577398
                2142968887597061149959517868662446671820187657310990383551314
                0484559914244577328979984545530051856654533577830880935695717
                1957...E- 57

     P(40)  =   2166519679400224897980431188869359054214340738085354086388100
                4218838071646222292470514323181519090393578756441861717911448
                3753948451955010101212985873129 23 /
                3091415677304603061235637708908459921242858758689828598750703
                6619757527212563775470260167627595372849125700198366335837974
                2532430185498928396389625527343 75
          Truncated floating-point equivalent (9K digits):
                +0.7008179764712869433308214620223329722055871007288577430473
                0566871186387471981477475162663011199891518786976375671755344
                4365170871567408704875985942581792220731526837624846016905981
                9635823806956159358133261470710015460996013487454390321892459
                8266449150611954144853736239610750868186525958076956236292767
                7475...E+ 0
Error term = -4006653331941186679458269506501396947424003425390844172190141
                3512046810182493515498000199704 /
                3486838656588634838798288128254941029769032822513844574816906
                1673790712518322807990661145668841096990257368510739373855009
                5399753330526096327571822528 5462890625
```

Truncated floating-point equivalent (9K digits):

-0.114907907320871422174389582304440757024938608402486710258565428366709086072181067840364918869645715529566195771364291292650958281151813793016938068328828680371077951813349498700760026896537598736697178677360547341817252139316011462923789592732733803404786957511184029739134976986786691836553809418307369159...E- 67

P(45) = 772337953947669181220133242674009214978830149906056710632573539463889146569505283072438568257228430435769635787578138471214797277216537731678505418776077165732882515755407633597 /
1102052144604644665390594679919942417712351151823808222619676879603153363281162762563001381090876473216715996445250055168437851447512506765308674709579634942921904468091884765625

Truncated floating-point equivalent (9K digits):

+0.700817976471286943330821462022332972205587100728857743047305668711852480394669161936686862847148119043580325582004195966713114005048057920259697210888006662325055947207660841495649066849902435423670766800095852596283023728702523876218695121375524447665991772897359225307036471417309023917044535402043742 14...E+ 0

Error term = 922040049221081327768318234391122674205849470970184632854749550030107966785629362018536082215715591054 4 /
2324338178185656063775303239419150553197119814311593922327160506771010758496300382521626212858767569661375708102676891355752272487948628018712525829974408058116588713652594159179687 5

Truncated floating-point equivalent (9K digits):

+0.396689284663736894759127462991590960584791370076306112544393271516209941590409660308791623956397089506990369674919511655212677897288685406301314231370513500144723716137761639051099330785661085095736446735797126032422498826682048299764693244966992630428711535305038606873845966308454966584280837948279537015...E- 78

P(50) = 641764070437046364874608149071496780182433321356160392474501653338589644830447372747492527386779479429870386717228433686350351141308847370961978420684888774416246831173215736194996345509586276980577 /
9157357430647185134801086671201117064413142208462397549301513695210660403803372679478555816036164325852836328117484750634284844772420150065086876868196441029532426958226629473128991799432641601562 5

Truncated floating-point equivalent (9K digits):

+0.700817976471286943330821462022332972205587100728857743047305668711852480394669555647484650114543735201910324421933088026612391340797282678517707952

```
                0715891377385789278310459169610193613838129371011876958794772821697009920246025181511815425053902996249022356573401193225312335833153989069653336391284468560 31...E+ 0
Error term = -10609326312188337234069173115013405166365591767821747026580245806834091221016502058703724523154666168528787818013921327612618373138412317651503091394189429369207155798933183370062050142560824484223313638221530546220784194749326918268932032157939962496790091145708534323168647243335821692819766453034400605684318847656625
        Truncated floating-point equivalent (9K digits):
                -0.799128161736594996642462485565751508716924245526832004389927274435425711693437383505055330180596097872713082139820313383155854623098741819220707322882501583985199499440734015611398482937681079583258856316561284455595721632144938758909899603980243467730702372014554273690986872042751386561994029843695055734...E- 89

    P(55)   =   5218070143051721349512894911701354604182589200762655709600592882846046783989974130674969480119517965177043137384387283631852145717554750683927101365228590932920654308883517126283521917587868618861696154441203573239394 9 /
                7445685353742505968688969744799053292512382252547151519528501373872410629369604210012118719924005646101683367557494400376543938073205330398259671530630656797826546334159517594290889375089891583361710821680859973144531 25
        Truncated floating-point equivalent (9K digits):
                +0.700817976471286943330821462022332972205587100728857743047305668711852480394669555647484642177531842262350188592763522067559671786133571490270822852317950552753752815280313878251264591433717383682306488567338323132878588770718039199823572651243463101588614334912672936547454363962331236052551137015827143422...E+ 0
Error term =   6103737299349311493959365939387869229370440776523574350768266727131822122459855330330532317650181471370522701064786338616192 8 /
                6165407202851835809878870086150600877918170636603921362897095091136423494060130135704744918151792799260145014189350795726197584465455747041609319270610245992097069518547122703470165237932559815494248007603657781622302246093 75
        Truncated floating-point equivalent (9K digits):
                +0.989997432209506188836345971004546040676629241962565345250736111892369914938922323075921573014962777940459924839309108625312714035238884737318151289150903317649814105075214407942637278781260063386000346091274110487332152600497301952263674174443377 7
```

```
            6331614941200247057462213378117599535472409606599 3
            860665261...E- 100

    P(60) = 1441779253126714911924688294050248975247090262425 9
            1356525196115096086160667405054712561513105307544 8
            6076674095727025933581383151393239615211710806127 9
            4211864064166251059090793259490533867420449026834 1
            3689310225700976285014389364309658999666 01 /
            2057280637101045776958853948643928996560570265563 8
            0129407464715165080177693783704703912128226094093 5
            6174012271329633204355132345868232745851483047030 9
            7357259101766320125351029433971207376529864075073 0
            8846338266961152194202824050789123535156 25
        Truncated floating-point equivalent (9K digits):
            +0.7008179764712869433308214620223329722055871007 2
            8857743047305668711852480394669555647484642177531 8
            4236073623573358970332543976157975605115434907886 6
            0405306676403629579730693803656145796436035668280 3
            3908680642626002750867953694017117877116224478076 68
            5494652644311899936239527520290031607844528088150 6
            647046626...E+ 0
Error term = -2809272364573616923306634761011294939392175701976 0
            8041583334878211029983391006497148133853283374284 1
            8449283697362261743966069235914155616 /
            3567421316923157126398169813084159144698867187290 1
            1294258625966937861786889372551790704714214073784 6
            5845477457062336469112404178955771388245526623254 9
            1863063073640582114404576536889470237649481209788 2
            6383066332809185078900824436798727298767089843 75
        Truncated floating-point equivalent (9K digits):
            -0.7874798390778717032162885492329219315747301053 4
            9446827978257375740249740582368189722113950437035 6
            2367229154600361957135863467738550479794104424658 9
            4456015742125775470774019598481329699548819432496 9
            7117354364920763402119982333326862187553274954473 4
            3324052100423137747002881772320368683146615749035 0
            992087003...E- 111

    P(65) = 2244153601470254009730441218238085145855659869245 2
            9137429320930831805052524739541000367134930521028 7
            2555793889871349430953751175223202710047811271026 8
            9503470649157590816741959908625647967793922656942 9
            6156247545580306581866064727973776492347399208243 1
            891499676044133 /
            3202191834133408142291701799101868278540041358118 7
            7224231044435679923627127306557969314696451530493 4
            1031978170341063889208507290006490667114745816376 8
            9234102784091833775091253671345017316034623741706
            2675766397131496831923643796015411077530757048593 2
            121185302734375
        Truncated floating-point equivalent (9K digits):
            +0.7008179764712869433308214620223329722055871007 2
            8857743047305668711852480394669555647484642177531 8
```

```
                4236073623573280671354469047379281378626550212468 2
                2034304824240788524413855456129333345092797059195 2
                1125234837135130614595148623066468877565543275411 2
                8533124799372443332257376653938705010275841055857 1
                780804209...E+ 0
Error term =    4040562994084771720559231542525943353796815002034 3
                1769142025797466237486843667776477221816703776584 9
                2798419190886929931850513831840625297776647043 6 /
                9690152709271106379388918814262163597690019153803 2
                1668245563566811016888049942375070943202931976426 1
                0896869141269093435133863910288641407755932314938 1
                1391318434940298194120364273485715690005237490477 7
                3363136694359622563084138491122235461715823904747 9
                1919188446044921875
        Truncated floating-point equivalent (9K digits):
                +0.4169761938033175751849590371260425153175352856 3
                5179989712653652735721441870499113310323230739471 2
                8901217508658244962756179139017892044222599509256 3
                '6558827457068146785624097089052179826838074749465
                7898139276038434198828249953944409463953149620465 6
                9701837119575032285989391995561272177452874480518 3
                343900747...E- 122

    P(70)   =   8930707846352442869963849004593168697660067877936 7
                9619382331160772402625553422184525107300140525037 4
                2189410853152118840849870342351734685014372894849 7
                4943887558293009281589925525354034731562984319710 0
                2590844003958542267518237151141878444466966648510 5
                5417528295063780871349167961419963 /
                1274326308140633276830694960634256928557506178176 7
                3652948092595453850343812938934303026178422810191 7
                5582443587552077706586285130583873792481363816899 8
                3425063596211417623616147294224175141378565388699 4
                2583113825830669618851769854271222743733423177262 6
                6039795847651767275548110961914062 5
        Truncated floating-point equivalent (9K digits):
                +0.7008179764712869433308214620223329722055871007 2
                8857743047305668711852480394669555647484642177531 8
                4236073623573280671354469462154475625013925441803 6
                7082433539914309769004908046049169093230040400514 0
                8843335024379211027642266489715301121734804013074 8
                1330733381568202898919724100831515669590078018036 6
                031212036...E+ 0
Error term = -1859687171957950590130014512030285259814351125708 7
                7598715752164147527319633082159368377498919072358 1
                5993575516703982484769686447478736836292119819093 4
                740962992 /
                1227090255943419597060771480101795428885908674642 3
                5569347648761614642609809163134743916784644908027 6
                5722116627343798692759729204094509968194774638057 8
                1415154554110758787195284462879939356050769657510 7
                7152512531585106047675070440751119822816766825642 1
                7205153696262810821614749625276641845703125
```

Truncated floating-point equivalent (9K digits):

-0.151552598755514823448386752283497834533784907696625940340476625626473916312742229148583255294316948709052949799988227156685665129769169339432073442766024209680592205316436619025935691753832959922176233863341698459705950761920298905957158357218403184297287303446181372131268299270993050930894213842312950285...E- 133

P(75) = 1108632077486568960032719759076698489774236511265146226585985842310609015938071297629241308259228534091863511535720967956228751934284714615604117574635345377991504582895262037090731937504188904720081166102446747873856358436681816933040264412018525964237681679125207305509702337288040521133514773070859104712361 3 /

158191158718370928814374636589593153305305640274508329623792787626878443363617046810926053991376112076306643450721114983541597345324110680153822305526396456457442064866074241607549925031781625121987247174119267378578927167325372430787367909682988530764634611856260829379109487586079273683961818533744812011718 75

Truncated floating-point equivalent (9K digits):

+0.700817976471286943330821462022332972205587100728857743047305668711852480394669555647484642177531842360736235732806713544694621544756235058462450603598761537318075506668092186809050568305744066922743121647638929805256065814741490075062792191471230020191381353433224890868698883145483196849540784388078247590...E+ 0

Error term = 1069911662869746576009179226961426297766631276020452760652836781358477903082061042965924164994114413406089193901163945110259390723940254108246929471420971418505826173632 8 /

275616455835017669273284929330048151003834017050275862703534173882310311872429980658676463869174600070549064884191398635824525054758198038032004602918640546085801309616161151152834234382873125450038380751467999553698064803630996386160831109040670917151222884237163243027222460221225918539566676431343585968017578125

Truncated floating-point equivalent (9K digits):

+0.388188600578402755769723210300825355403654404058110556165495373207115875992727421368881682708427126185109920943059297120390177101819593106180899360009146669385751264015493227833533940883920787081815487627320232429890495702526290183065352412634935761534184158894653552016761281770430097492953216093796551579...E- 145

For x = +326/917, we get the following machine results for $P_n(x)$

(values of term $x^{n+1}/(n+1)!$ are omitted; they are identical in magnitude to those presented above, being, however, always positive):

P(5) = 13878157292959157 / 9726082238465355

Truncated floating-point equivalent (9K digits):

+0.142690108439273731811030752311954467792139434369502907088104756643331089523966932844124114012691183140664710311035308946275040102668950794144260905558569416066435683513125958260969957232790809716360803893137194559706245085696539461868540174143052057469315016272677223747984958231108518389885579684800809933...E+ 1

P(10) = 85037638442120952364830591844774317595950569691648790568568722389575

Truncated floating-point equivalent (9K digits):

+0.142690403724363279910850074008525140695287587879532837620550541908373907466680862228175098867818586975094449499442717304274701744225180355229804985335635496264172823483383335415830579662977289634859135183729389342687084419196965983926331906080298544021345633486063467452217049784196753774321528351112135029...E+ 1

P(15) = 248373044375392780950319580066613022680730638123103867 /17406429436917592584045131217073645576889438137439 7375

Truncated floating-point equivalent (9K digits):

+0.142690403724392874000519811603480373621079423005756715385335708089173671390175032255351829380728973682950436908525000281985195460346798344828002520804067810089129637384769358383906255906726666560067170700307023099342137801513452571089363236951722334107125404215984977180585713395148879368488691169146628586...E+ 1

P(20) = 212800914463206348991047561206424736195808499076735530570583896964158769 /14913470626534369248562370980178062921233430267728768892276799038994437 5

Truncated floating-point equivalent (9K digits):

+0.142690403724392874000837586825805217416588370056982607199108562529476043564144394836964528154562614953808359559152497634880272586455696443940045070115633264361587226993487643012641719883629604096039674756609415937871383929672557804224452837977277774206598423094719738617572475572111310580418979646652795935...E+ 1

P(25) = 6048029944178300530973743773034299659448934640729 4

```
          4579883428370733013816024618054002672879769 /
          4238568107116787106801814432826308036960783268030 2
          4904003030248089017362463917389991557546875
     Truncated floating-point equivalent (9K digits):
          +0.1426904037243928740008375868265406448491743411 1
          8864309701604266193217782513530438274223026115838 9
          1615902293634531082845114861547833502452252275626 4
          9900405437595767720798425903559401237097437470991 9
          5901227156912416724042538009613121333456511627230 0
          7660989582838116902981610096407596004615973156750 7
          052578867...E+ 1

P(30) =   4191360579926620003316520881292606131783514349924 6
          8305629806595625954918689988917111413037420401854 0
          02473680133679 /
          2937380840285693575356313391989011302132480952263 2
          9135540247428790377896869687390753701437943326826 6
          87421541328125
     Truncated floating-point equivalent (9K digits):
          +0.1426904037243928740008375868265406448497018009 5
          5710816258365576194406403249880511631008703240633 1
          4505565302190561109848102878090726889622866691955 4
          5602477246954787795315349492060678576700880022897 4
          7066666893576501066355286528643512463793645963699 1
          8766031198451941436524076735587539871714322833984 3
          705404020...E+ 1

P(35) =   2808529613583128775117790727213978034889606947401 7
          2711483990489225384939902751453662611073445223807 9
          4481594755176475963251611115856 9 /
          1968268040651013752765095168736854795779671385007 6
          5298283073261632302543922754868798320741187069962 8
          0614554839532676721811894140625
     Truncated floating-point equivalent (9K digits):
          +0.1426904037243928740008375868265406448497018009 5
          5857413097510697912223581728056826260229040695819 0
          0247362724112465955372324054618384043336881547041 8
          6846836397192597144323520230459903147070098602794 9
          2712781830681321712542447050438229038829407620315 3
          0060671169437433270769619478072104353424823575007 3
          936036670...E+ 1

P(40) =   1323346053223533755330022409951493964383816526496 4
          4508037547875372859729899656549871867019786150058 2
          4224237769620638322631014705587365490595578760209 8
          351589 /
          9274247031913809183706913126725379763728576276069 4
          8579625211098592725816376913264107805028827861185 4
          7377100595099007513922759729055649678518916887658 2
          03125
     Truncated floating-point equivalent (9K digits):
          +0.1426904037243928740008375868265406448497018009 5
          5857413097529071514126374236917612259601819722582 5
```

```
          71403277433929332300707963385395942389746985133247
          44524362205774453704949690026420932747384561461126
          18398715320767607717340811684775465445215395707482
          87296420849223173607841399243226487249393321382872
          297097320...E+ 1

P(45)  =  14295660494451794845939028596088683658428415235356
          87020038408764415805470426984096585896103706506377
          32864918864438545580644229427512683926205440605887
          32333276830570647917428424 3 /
          10018655860042224230823587999272203797385010471125
          52929654251708730139421164693420511819437355342248
          37883272404113641378948895586137500615028061337234
          51226753835495164471679687 5
    Truncated floating-point equivalent (9K digits):
          +0.14269040372439287400083758682654064484970180095
          58574130975290715141275331233866352846710969697460
          30850551875192174119458005371395543138675603254153
          24360726294835460422496841610125447946435007191446
          17554144101079608731244983406586524240661321194536
          36990190095846619294377542529056923106535197591105
          364206210...E+ 1

P(50)  =  40506677893656091819334314095012686122112672317212
          40865505759729962830603860124311103249855307228078
          61148675196928799869568415622500071014148938096579
          99187423828480136492872662093627533239757502684 67
          /
          28387808035006273917883368680723462899680740846233
          43240283469245515304725179045530638352302971210941
          01437926171642027269662830187945024652017693182914
          08967191550523570502551366699874578241188964843 75
    Truncated floating-point equivalent (9K digits):
          +0.14269040372439287400083758682654064484970180095
          58574130975290715141275331233866752558931336601998
          00401893525837788952117415725956490706646970956922
          62997722793964174272434158975521184238654452575880
          59378418280471669071862418286283749076529261777122
          29138276225668457959154175801337988166723796723607
          225247388...E+ 1

P(55)  =  13705319253781091198952387538948648613434565352715
          19606916290972138079574809028871754748998328569208
          93045945790681350008911015440940697996800881095651
          51135680115870117978753335443753412227254271039379
          59886871579841038355 69 /
          96049341063278326996087709707907787473409731057858
          25460191766772295409711886789430915633148701967283
          47117154414916777648574168011443487621375497627451
          35472691962447710657776966352472938659601425366069
          59968309365356445312 5
    Truncated floating-point equivalent (9K digits):
          +0.14269040372439287400083758682654064484970180095
```

```
          585741309752907151412753312338667525589313446482 82
          309107496023201460628722666021348914412530263340 32
          054965504899652976475636538239661447112228906833 53
          129220904544497530199597045489890169782656838153 46
          327554431144424753069676998457974814990296168945 00
          615479587...E+ 1

P(60)  =  636376247912731231896492247755173065875938058043 83
          919133593786770697406974431693648497368683351045 74
          011184513327424472083159355310119729116507119386 17
          716430039411990063148013105792083365673495158276 87
          704627976533767808352720497919160155161 57 /
          445983914336590343256814492363369223030613134492 85
          202878541301888933465094456607313435496328733684 61
          827932730148382023321741977076330175674097723482 23
          902622602480810656544628758413338662044935568722 13
          805849554376193832309703115905334472656 25
    Truncated floating-point equivalent (9K digits):
          +0.142690403724392874000837586826540644849701800 95
          585741309752907151412753312338667525589313446482 82
          309207117032108071428721168362255873641820232549 51
          806188671304411400993640458276435110693700351673 89
          614461800984007396626345482889436316138809784596 49
          351774301387800583426581537291155752767645735754 87
          255296018...E+ 1

P(65)  =  456922045615450109416997210575683603629513787744 71
          453639328422588541464252026339041929998264512402 55
          002171573118651773437343657278860561956936307379 09
          848752346457369118854672297761369233441292666560 50
          836182524231033108883913173281475114051367809081 23
          809244838033977 /
          320219183413340814229170179910186827854004135811 87
          722423104443567992362712730655796931469645153049 34
          103197817034106388920850729000649066711474581637 68
          923410278409183377750912536713450173160346237417 06
          267576639713149683192364379601541107753075704859 32
          121185302734375
    Truncated floating-point equivalent (9K digits):
          +0.142690403724392874000837586826540644849701800 95
          585741309752907151412753312338667525589313446482 82
          309207117032108863449580628796000866012311692428 57
          609310897891319235184559021232597371860764764758 19
          279027110840356815908389099635838503789644535192 93
          723348255220352353505428572924197585824665850338 44
          465646820...E+ 1

P(70)  =  268932686234713816738562974676702155501474431574 04
          023205577833985328602333845019977227968634030681 49
          288310309512523454267553452694103900819678529323 89
          142156024116195820305239966876875413993107084408 62
          360208188290960689557696310078621328748691548599 74
          244551011833460010670468227894483990 57 /
```

```
        188472860973999661643259784677806599733655163752 33
        933271022894867624465849933668383417571788733627 36
        068643406598952292804111570813354933907993708519 48
        548566905879668666532828184815755503409889820988 64
        508042534840356036628176761446713843798173287917 14
        747285805867696380053565611267089843 75
    Truncated floating-point equivalent (9K digits):
        +0.142690403724392874000837586826540644849701800 95
        585741309752907151412753312338667525589313446482 82
        309207117032108863449580632988004147446631207680 40
        281552423616861885495679586558128264646523441900 79
        033246261879600808271929609722373391702894175587 05
        982443666963116183124277235450716004562682770543 80
        181086093...E+ 1

P(75) = 477216919728411404683526461743330992228362326601 75
        769495472301175580275427465322335367284250523731 89
        191184851227540516570194428164429327782052713045 97
        674198462937712039405174685295303222980285705267 88
        296651356426090874130369058004601730855732271128 34
        923697126158517008196959271233579108815433389070 83
        54030169 /
        334442196021925853730178935707385102125381903328 77
        025290442449815407704728037430615417770399868099 80
        471599227287972832633014743030108048206211888066 09
        717666929052991260199242299259883788121829010543 86
        294739802828452204475121643220039612665894923579 44
        284029848969573559812039906682437075770862971115 11
        23046875
    Truncated floating-point equivalent (9K digits):
        +0.142690403724392874000837586826540644849701800 95
        585741309752907151412753312338667525589313446482 82
        309207117032108863449580632988004147461861664324 62
        440359751695588011624454349960764866059124810125 87
        867005813773551456707966643919496804877785373490 92
        566824021500406703663319400539671313514312867262 73
        750919968...E+ 1
```

The two computations are performed in virtually identical computer execution time and provide comparable bounds on the truncation error $R_n(x)$. After 75 iterations, we have 145 significant digits.

For $x = +1/1$ and 100 iterations, we get the machine results for $P_n(x)$ given below (truncated floating-point equivalents are from the rational computation with K = 18):

```
P(5)  =  163 / 60
```

```
        Truncated floating-point equivalent (9K digits):
              +0.27166666666666666666666666666666666666666666666666
              66666666666666666666666666666666666666666666666666666
              66666666666666666666666666666666666666666666666666666
              666666666666666...E+ 1
Error term = 1 / 720
        Truncated floating-point equivalent (9K digits):
              +0.13888888888888888888888888888888888888888888888888
              88888888888888888888888888888888888888888888888888888
              88888888888888888888888888888888888888888888888888888
              888888888888888...E- 2

   P(10)  =  9864101 / 3628800
        Truncated floating-point equivalent (9K digits):
              +0.27182818011463844797178130511463844797178130511
              46384479717813051146384479717813051146384479717813
              05114638447971781305114638447971781305114638447971
              781305114638447...E+ 1
Error term = 1 / 39916800
        Truncated floating-point equivalent (9K digits):
              +0.25052108385441718775052108385441718775052108385
              44171877505210838544171877505210838544171877505210
              83854417187750521083854417187750521083854417187750
              521083854417187...E- 7

   P(15)  =  888656808019 / 326918592000
        Truncated floating-point equivalent (9K digits):
              +0.27182818284589944642854695764748674801584854494
              90740496031501322506613511904517195522486527777533
              06853835954365054894155423255952356481457010557539
              65806875859785 9...E+ 1
Error term = 1 / 20922789888000
        Truncated floating-point equivalent (9K digits):
              +0.47794773323873852974382074911175440275969376498
              47702757755667808577861487914397967308020218073128
              12603817894823185828476833767839058844349849640854
              931860222865513...E- 13

   P(20)  =  6613313319248080001 / 2432902008176640000
        Truncated floating-point equivalent (9K digits):
              +0.27182818284590452353397844906664158861464034345
              40261720776551790178813437759641322287953390036848
              50145591687172582038650623329459126285979184383510
              376415660686614...E+ 1
Error term = 1 / 51090942171709440000
        Truncated floating-point equivalent (9K digits):
              +0.19572941063391261230847574373505430355287473790
              06217651053969813659091146131012976603281167818700
              39725055242199938501677737549690836095283080921607
              503997011673593...E- 19

   P(25)  =  853519036400770422159 79 / 31399210614030336000000
```

```
      Truncated floating-point equivalent (9K digits):
            +0.271828182845904523536028746877832451509386078 4
            391696193080756839191240891000263619440385718826 80
            567756879819574196869132583893555649434869104758 02
            141166650096926...E+ 1
Error term = 1 / 403291461126605635584000000
      Truncated floating-point equivalent (9K digits):
            +0.247959626322479746007494354584795661742265554 24
            726584208142923554006931515797772582893498122766 55
            008171876484746357830112211787914716926156391527 40
            833076177756148...E- 26

   P(30)  =  554640022134056545398181834379 77 /
            204040661393993122027929600000000
      Truncated floating-point equivalent (9K digits):
            +0.271828182845904523536028747135266237222570213 09
            834818158153785765237928007911999934379359859554 17
            917022688752925346939281769130641698777185823448 60
            078969837134158...E+ 1
Error term = 1 / 8222838654177922817725562880000000
      Truncated floating-point equivalent (9K digits):
            +0.121612504155351794962997468569229214972478510 43
            941918714377391474559686892842808187273287251740 88
            693576772783372058425740638622531166770719372459 40
            930387172184333...E- 33

   P(35)  =  175552552173787468334560001126914165381 1 /
            645821747899134058104165708595200000000
      Truncated floating-point equivalent (9K digits):
            +0.271828182845904523536028747135266249775724433 08
            628473549695391873715351872669489090752280783019 29
            787920113770707280037121426730066889096720514460 51
            338026778732550...E+ 1
Error term = 1 / 37199332678990121746799944815083520000000 0
      Truncated floating-point equivalent (9K digits):
            +0.268822026628663638669161566136746524622698590 4
            081781386999793705966543261843770750381279646387 02
            973309718295021420493760784098272568937624168106 59
            400347956509099...E- 41

   P(40)  =  221788768801477525370663341865315589377446975120 1
            /
            815915283247897734345611269596115894272000000000
      Truncated floating-point equivalent (9K digits):
            +0.271828182845904523536028747135266249775724709 36
            999289531811847777417343778495364051097564458849 51
            070874209930559126023277991092973398349699701304 74
            901832418570756...E+ 1
Error term = 1 /
            334525266131638071081700620534407516651520000000 00
      Truncated floating-point equivalent (9K digits):
            +0.298931082714240451078912191448721200908673649 87
            276210801870071828728463003908660407399538970376 92
```

```
            22463643962111523793516127177889515866693907210253
            430473478388763...E- 49

   P(45)  =  52446275686862783210194117527876021128167788169502
            18873 /
            19293906591206482993865026047686728236190924800000
            00000
       Truncated floating-point equivalent (9K digits):
            +0.27182818284590452353602874713526624977572470936
            99959574966781947323134279551697480772253613389970
            02625718501556173571277581893272273856336784097478
            926767710473271...E+ 1
Error term = 1 /
            55026221598120889498503054288002548929616517529600
            00000000
       Truncated floating-point equivalent (9K digits):
            +0.18173154015614791268097229179993954720581619670
            19380744823048229218399934385786971221649968458777
            81172286664925242964526656605226950439134171904813
            841186115080193...E- 57
   P(50)  =  26669057057831373733063413228807023646124027886883
            46977445977371 /
            98109978070043154979395510213112157562508521190195
            2000000000000
       Truncated floating-point equivalent (9K digits):
            +0.27182818284590452353602874713526624977572470936
            99959574966967627723419298053548538722835117660645
            04308549791388095014223536288443731116412678523219
            063776004717953...E+ 1
Error term = 1 /
            15511187532873822802242430164693032110632597200169
            86112000000000000
       Truncated floating-point equivalent (9K digits):
            +0.64469596404571726804541778342521245095845500151
            10231130831936424280350205279938114329596975168712
            20730443471449725154340660396168930122170044985419
            136399648796376...E- 66

   P(55)  =  86281006307589430592675033594549893859179084519496
            44236234459487161204269 /
            31741008384145689814912752118916292398950802628623
            591905689600000000000000
       Truncated floating-point equivalent (9K digits):
            +0.27182818284590452353602874713526624977572470936
            99959574966967627724076630352116014296678736937414
            63393397558263178586157341597727185588905394527327
            879674369025147...E+ 1
Error term = 1 /
            71099858780486345185404564746372494973649797888116
            84586874470400000000000000
       Truncated floating-point equivalent (9K digits):
            +0.14064725544496499047018184982060489540125645701
            06306850651082064397942685651162716199243841232298
```

```
            4945052402861152761335900743128485118178686377071 1
            105892633047502...E- 74

   P(60)  =  9834255679742094600105637733781649667700164457438 1
            3677744528736750847228861625287 /
            3617820483800604410554930949227549730762683741896 1
            9389236925117235200000000000000
        Truncated floating-point equivalent (9K digits):
            +0.2718281828459045235360287471352662497757247093 6
            9995957496696762772407663035354759456937975785627 4
            9426128891712033335069418782248477715025400671217 0
            198972615600990...E+ 1
Error term = 1 /
            5075802138772247988008568121766252272260045289880 3
            6003099405939480985600000000000000
        Truncated floating-point equivalent (9K digits):
            +0.1970131956802168311835039121583381272996734616 0
            5266947044548580437658077375233053887003162138524 2
            9803595376588095684608091354845332053736284065977 7
            367932793625011...E- 83

   P(65)  =  1120971936596863269895427510724621349404079948800 2
            7291860323019701140051814002014686594989 13 /
            4123825296041235333361585153392748126093129275672 7
            1874646106156719447788748800000000000000
        Truncated floating-point equivalent (9K digits):
            +0.2718281828459045235360287471352662497757247093 6
            9995957496696762772407663035354759457138217852330 1
            5288711313368073694503439765554064975813976977846 2
            186294854371998...E+ 1
Error term = 1 /
            5443449390774430640037292402478427526442930643887 9
            8874532860126869671081148416000000000000000
        Truncated floating-point equivalent (9K digits):
            +0.1837070445983758169693060003004184130718384266 6
            1886264725183512385965734324696597726313376492948 0
            4101584064632440144801520308810562993734427969336 6
            452779591996462...E- 92

   P(70)  =  3256113370137347642791233047588458160768753106587 7
            5672104218132471641727135742027147215543785080465 0
            1 /
            1197857166996989179607278372168909873645893814254 6
            4258575553628646280095827898453196800000000000000 0
        Truncated floating-point equivalent (9K digits):
            +0.2718281828459045235360287471352662497757247093 6
            9995957496696762772407663035354759457138217852516 6
            4274262740259311722891072923263666020002761414744 6
            187302270494771...E+ 1
Error term = 1 /
            8504785885678623175211676442399260102885846081207 9
            6235886430763388588680378079017697280000000000000
```

```
          00
      Truncated floating-point equivalent (9K digits):
          +0.11758085546679308394123312760251653389169772401
          49977652974785975756856508013334282433114397426203
          47177439723480659150546036258846375153456832720161
          134753701681314...E- 101

P(75) =   49880352549776482129450810428299786001825984923351
          77142626172834378457256428421230110396879510751414
          4894163 /
          18349956221446300363865737512273693536188050211960
          15002795554327859159961358939897148211200000000000
          0000000
      Truncated floating-point equivalent (9K digits):
          +0.27182818284590452353602874713526624977572470936
          99959574966967627724076630353547594571382178525166
          42742746639193146571784043710378027990442637864039
          217890445449779...E+ 1
Error term = 1 /
          18854947016660502549879322608611465582303945353793
          29335672487982961844043495537923117729972224000000
          000000000000
      Truncated floating-point equivalent (9K digits):
          +0.53036478920698403856827775685976411654849570196
          99844279096986206451839853612328453332182206657827
          04015477822273693802607745898594950183857400660218
          323738760756584...E- 111

P(80) =   19454595456153906732658104250624381123142389194687
          34805980310746153993455597653185852254935492569210
          92507319165187339201 /
          71569457046263802294811533723186532165584657342365
          75257710944505822703925548014884266894486728081408
          0000000000000000000
      Truncated floating-point equivalent (9K digits):
          +0.27182818284590452353602874713526624977572470936
          99959574966967627724076630353547594571382178525166
          42742746639193200305992164278502766477418166569580
          047899701325633...E+ 1
Error term = 1 /
          57971260207473679858797342315781091054123572447316
          25958745865049716390179693892056256184534249745940
          480000000000000000000
      Truncated floating-point equivalent (9K digits):
          +0.17249926884823517582858543656549345994049177042
          22330086663670433349741319006932164221991416491588
          78840424897247389078406014044621755212496455678840
          330548612440258...E- 120

P(85) =   38288414614989307578461955705877673881514050355409
          20142411291876315174057100509968150213985293604740
          75012244133279098899206672363 /
          14085520571902751384747397211303057974002831716528
```

```
            710320255095637628001307989796672552014322617046200913756160000000000000000000000
        Truncated floating-point equivalent (9K digits):
            +0.271828182845904523536028747135266249775724709369995957496696762772407663035354759457138217852516642742742746639193200305992181741359658728758286306676365147312793900...E+ 1
Error term = 1 /
            242270953836727323817655232034412597152848705524293817508387644967201622497424502767894646349013194655716605952000000000000000000000
        Truncated floating-point equivalent (9K digits):
            +0.412760995143448323981745742450379074462389016645697382843880092252722601968796496639711995187714922188276208709988498553990517046941543627131008723259766455729011...E- 130

   P(90) = 119434520557937811818095103133650347785541235146222106722019050802813084714212116549676475595484769535230469149643156910441362602709077 /
            439375046794333018075051896274897229244896378539460750907816179458094768513340769007784319465694708938412811878400000000000000000000000
        Truncated floating-point equivalent (9K digits):
            +0.271828182845904523536028747135266249775724709369995957496696762772407663035354759457138217852516642742742746639193200305992181741359662904357289285658404382300333776...E+ 1
Error term = 1 /
            135200152767840296255166568759495142147586866476906677791741734597153670771559994765685283954750449427751168336768008192000000000000000000000
        Truncated floating-point equivalent (9K digits):
            +0.739644134660968639432635580962974693849761343750631001949462014037311222164257343241716731737336741258003352262824097699032275298104294336561254556247357432071371...E- 140

   P(95) = 539996015633712959995004918827542179695533686176694694656120114142920107263943473051582286252448361383312488228733173577588157824168886193592867363 /
            198653432466135755049994271142206290296110506540718709023312510985274884376821213939841641834335085933909273785850511679815680000000000000000000000
        Truncated floating-point equivalent (9K digits):
            +0.271828182845904523536028747135266249775724709369995957496696762772407663035354759457138217852516642742742746639193200305992181741359662904357290033429525957673588037...E+ 1
Error term = 1 /
            991677934870949689209571401541893801158183648651267795444376054838492222809091499987689476037000748982075094738965754305639874560000000000000000000000
```

```
        Truncated floating-point equivalent (9K digits):
              +0.100839190309314816860649675410295402621592883580
              547344107314539550973904233562489022560837869208754
              026619716726572164455970548044998051604324124368356
              70228450878 9...E- 149

   P(100) =   429977890779876775280119912224203763466351828078471
              427513178281334659752387095672066000822754494999649
              605775817505090667134768643813040977474177102242650
              8339 /
              158180026176176529968981760773333390662230454685392
              578760327057449521355920728670523629599959587319129
              243555798012243658052856289689600000000000000000000
              0000
        Truncated floating-point equivalent (9K digits):
              +0.271828182845904523536028747135266249775724709369
              995957496696762772407663035354759457138217852516642
              747427466391932003059921817413596629043572900334295
              26059563073802...E+ 1
Error term = 1 /
              942594775983835942085162312448293674956231279470254
              376832788935341697759931622147650308786159180834691
              162349000354959958336970630260326400000000000000000
              0000000000
        Truncated floating-point equivalent (9K digits):
              +0.106090127537174942889460075199919952674409965276
              505921270976813177818838304577868919398607091099130
              980444685249686513418826153649446631602276706071992
              326820004522...E- 159
```

Here, after 100 iterations, we have a value of e to 159 significant decimal digits. We observe that for the three illustrated computations, the number of significant digits grows slightly faster than O(n), where n is the number of terms in the Taylor approximation. Minimal rational arithmetic K increases correspondingly.

CHAPTER VII

CONCLUSIONS: SUGGESTIONS FOR ADDITIONAL RESEARCH

We present in this chapter some overall conclusions from the implementation of extended-precision residue arithmetic and exploratory applications of exact computation. We also suggest avenues which future investigations might profitably explore.

Residue vs. Rational Arithmetic

In relation mainly to the numerical results of Chapter 6, we may draw several conclusions regarding the relative efficiencies and uses of equivalent forms of exact computation, residue and rational arithmetic.

Advantages of Residue Arithmetic

1. Smaller operand storage for a given precision K is necessary for residue arithmetic computations. Only a single integer residue is needed to represent an operand at an intermediate step of a computation. (Keep in mind that denominator storage must be available for original operands and final results.) In contrast, rational computations require, at each step, memory to store operand denominators.
2. Intermediate "pseudo-overflow" in a residue computation is of no consequence as long as final results are order-N Farey fractions. A rational computation fails at the point at which overflow takes place, prompting us to institute algorithms for addition, subtraction, multiplication, and division that minimize the possibility of overflow and which maintain operands as reduced

fractions. Thus, residue arithmetic is ideally suited to computations for which intermediate results are significantly larger than the final results.

Disadvantages of Residue Arithmetic

1. Total memory required by residue arithmetic is frequently larger than that needed by rational arithmetic---as much as 50% larger. Why?

In terms of a final rational result, the precision K must be large enough for K-word storage to hold numerator or denominator. Approximately 9.6 decimal digits may be stored per 32-bit word. For the result of a residue computation to be valid, the numerator and denominator of the final result may not exceed the N of Farey fractions, $N \leq [(p-1)/2]^{\frac{1}{2}} \doteq (p/2)^{\frac{1}{2}}$. If the precision K is barely sufficient to hold the modulus p, we can store only the equivalent of about 4.7 decimal digits per 32-bit word in a residue computation.

In the computations illustrated in this thesis, intermediate results were not significantly larger than the final results (many orders of magnitude difference). Thus, the magnitude of the final result was the principal determinant of minimal K. If one has a computation with intermediate results greater than the square of final results, residue arithmetic begins to be preferable to rational arithmetic in terms of memory requirements.

The uneven distribution of Mersenne primes contributes to the use of a larger K than would otherwise be true. Mersenne primes and associated values of N can be computed readily and were employed

here for that reason. Filling the larger gaps in Mersenne primes with a choice of other large primes would necessitate storage tables of available p and associated N, but overall memory requirements would actually decrease for many calculations, thus reducing execution times also.

[Note: The residue Gaussian elimination computations involving Hilbert coefficient matrices required marginally less computer memory than comparable rational calculations principally because Hilbert coefficients were computed for temporary use by the program. As soon as each coefficient had been determined, the residue equivalent was computed. Hence, only K-word storage, re-used for each coefficient denominator, was necessary. The rational computation required K-word storage for each of K^2 denominators.]

2. Residue arithmetic execution times are almost always greater than execution times of comparable rational arithmetic computations, usually by factors of 3 to 10. Execution time is required to convert a rational number P/Q to its residue equivalent modulo p. The inverse mapping onto a rational result by an Extended Euclidean Algorithm employs a finite number of successive divisions that varies with the residue being mapped. This conversion adds execution time.

As noted in (1), larger residue operand memory is required for given magnitude of final results. Larger operand storage increases integer arithmetic execution times. For a large p, the residues are generally also large numbers, even when P/Q is a simple ("small")

fraction. For example, $|1/2|_p$ for any odd p is given by $(p+1)/2$, which is the order of magnitude of p. Hence, computations in residue arithmetic with "small" numbers may involve arithmetic with "large" integer operands. For a given K, execution times of extended-precision integer arithmetic routines implemented here for multiplication and division increase with the magnitudes of the integer operands. Consequently, for small operands, rational arithmetic computations are relatively fast regardless of precision K.

3. The final rational result reported by a residue computation is not guaranteed correct. One must know independently bounds for the magnitude of the numerator and denominator in order to choose a large enough modulus p to ensure the result is an order-N Farey fraction. This we cannot always do. If the true result is not a Farey fraction, but the residue result $\in \hat{I}_p$, then the reported result, a Farey fraction, will be incorrect. In the worst possible case (a finite possibility), the reported Farey fraction may lie "very close" to the true result among the real numbers. For instance, if the reported Farey fraction is 600/1000, but the correct result is really 5999/10000, do we know from prior information enough significant digits of the result to detect the error?

4. We cannot easily "peek" at the intermediate results of a residue computation. An intermediate residue must be mapped to F_N to determine the sign and magnitude of its rational equivalent. If we have pseudo-overflow, the "peeking" itself causes the computation to

fail. Additional research is needed to search for methods of determining the sign and relative magnitude of the rational equivalent of a residue without performing the inverse mapping. We would also like the ability to know by direct inspection which of two residues maps to the larger Farey fraction.

With existent hardware constraints and multiple-precision software routines developed here, rational arithmetic is the superior form of exact computation--in most instances. The general advantages of exact arithmetic accrue alike to residue or rational computation in the forms described and implemented in this thesis.

Additional Software for Exact Computation

A number of added software routines would extend the versatility of those developed and described in this thesis. Those discussed below would be useful in either residue or in rational arithmetic.

The computations illustrated in this thesis utilize initial operands that can fit into single 32-bit words. Compilers and assemblers are generally not designed to allow easy initialization of arithmetic operands larger than single- or double-precision. The availability of an input conversion routine to convert multiple-precision decimal character input to a corresponding binary number in storage for use in arithmetic would be helpful in this regard.

We often wish to compare one rational number to another to determine which is larger. The FLOAT routine, which converts a rational number to a decimal character truncated floating-point equivalent, could be utilized to provide such an "arithmetic compare." We could then stop

a numerical approximation when successive iterations are sufficiently "close" or when the approximation error is less than a designated limit.

Another useful software addition would be the development of a small interpreter or preprocessor to allow user-controlled execution of multiple-precision exact arithmetic assignment statements with ease comparable to that with which we execute ordinary assignment statements in a higher level language. At the moment, if we wish to execute the assignment statement

(7.1) Y = A*(X**2) + B*X + C

using residue arithmetic, we must program separate calls to procedures RESMLT (3 calls) and RESADD (2 calls), plus any subroutine calls to compute residues of operands or to extract the rational value of Y if we wish it at this point. Complex assignment statements result in a multiple-precision residue or rational arithmetic application program significantly longer than a comparable program to use machine floating-point or integer arithmetic.

Hardware and Micro-programmed Features Useful for Exact Computation

Most "real world" computations do not require that results be computed exactly, or, indeed, that computations be performed with tens of significant decimal digits. Computer architecture is consequently not generally designed with such computations in mind. If we wish to handle computations exactly or with extended-precision, we must develop software routines such as those of this thesis to overcome limitations of fixed-word-length operands and the supplied instruction sets.

Machines supplied with hardware or that can be micro-programmed to give alternate specialized extended-precision instruction sets would facilitate exact computation significantly and lower execution times dramatically. We might for instance "choose" 4 or 5 large precisions "K" with corresponding prime moduli and implement instructions for these precisions. Integer addition, subtraction, multiplication, and division operations are used by both rational and residue arithmetic; the overall execution efficiency rests solidly on the efficiency of these processes.

In residue arithmetic, we might desire machine instructions to perform residue addition, subtraction, multiplication, and division directly for a selected large modulus or for selected moduli. We also compute frequently the residue of a large integer I modulo p. We might even wish instructions to compute directly the residue of r/s modulo p for selected p or to perform the inverse mapping $\hat{I}_p \dashrightarrow F_N$.

In rational arithmetic, the processes of computing the greatest common divisor (and reducing a rational number to lowest terms) are repeated every time an addition, subtraction, multiplication, or division is performed. We also need machine instructions for performing directly rational addition, subtraction, multiplication, and division.

New Avenues for Applying Numerical Approximation Techniques

The use with a computer of extended-precision exact computation in either form (residue or rational) allows us to select numerical algorithms based on their mathematical characteristics (stability) in an environment free of roundoff error. We can also study experimentally the exact behavior of such algorithms in a way that is impossible for

exact hand calculations. Further, when we have a computationally stable, rapidly convergent numerical algorithm, we can extract many more significant digits than is possible using machine floating-point arithmetic.

We extract our exact results at enormous price in terms of computer memory and execution time. Some iterative computations can be sustained only for a very few iterations.

The Newton-Raphson approximation to the cube root of -1041/917, described in Chapter 6, is a perfect example. Our initial approximation of -11/10 gives us approximately 70 significant floating-point digits in the p_6 term. Because the method is quadratically convergent, the number of correct digits approximately doubles at each step. However, the number of numerator and denominator digits triples with each iteration (see Tables 6 and 7, pages 117 and 131, respectively).

Our presumed goal in such a computation is to gain the greatest number of correct digits in the approximation while using as little computer memory and execution time as feasible. An obvious approach in such a numerical situation is to compute exactly iteration p_{n+1} from p_n. At this point, however, we then "truncate" the number of numerator and denominator digits to prevent the overall number of digits from growing faster than the number of significant or "meaningful" digits, plus some generous bound (e. g., 10 extra decimal digits). If operand sizes can be controlled, the computation can proceed through more iterations than is otherwise possible, thus, hopefully, in the end allowing more significant digits.

The truncation itself can be performed by making simple

(identical) right bit shifts of both numerator and denominator if both are significantly greater than 1 ("many" digits). Or we might merely divide numerator and denominator by some appropriate power of 10.

In general, we try to formulate an ideal amount of truncation based on the current accuracy of the approximation, the growth of digits in the approximation, and how rapidly the method is converging. The degree of truncation at each step, if any, can and must be taylored to the particular numerical algorithm and to how each iteration is computed from previous iterations, as well as to the data employed in a given application of the algorithm. This topic merits further study.

LIST OF REFERENCES

LIST OF REFERENCES

1. Amanou, F. [1982], "Conception et realisation de logiciels d'arithmetique", Ph. D. dissertation, University of Sciences and Technology, Lille, France.

2. Andrews, G. E. [1971], Number Theory, W. B. Saunders Co., Philadelphia, Pa.

3. Burden, R. L., Faires, J. D., and Reynolds, A. C. [1978], Numerical Analysis, Prindle, Weber, and Schmidt, Boston, Mass.

4. Forsythe, G. E. [1969], "Solving a quadratic equation on a computer," An essay in The Mathematical Sciences, A Collection of Essays, edited by the National Research Council's COSRIMS, M.I.T. Press, Cambridge, Mass.

5. Gregory, R. T. and Karney, D. L. [1978], A Collection of Matrices for Testing Computational Algorithms, Krieger Pub. Co., Melbourne, Fla.

6. Gregory, R. T. [1980], Error-Free Computation, Robert E. Krieger Pub. Co., Melbourne, Fla.

7. Gregory, R. T. [1981], "Error-free computation with rational numbers", BIT, 21, 194-202.

8. Howell, J. A. and Gregory, R. T. [1970], "Solving linear equations using residue arithmetic -- algorithm II", BIT, 10, 23-27.

9. Howell, J. A. [1969], "Solving systems of linear algebraic equations using residue arithmetic," Master's thesis, University of Texas, Austin, Texas.

10. Kwang, S. F. [1981], "Computation in a finite field using rational operands", Master's thesis, Department of Computer Science, University of Tennessee, Knoxville, Tenn.

11. Kornerup, P. and Gregory, R. T. [1982], "Mapping integers and Hensel codes onto Farey fractions", Technical Report DAIMI PB-149, Computer Science Dept., Aarhus Univ., Denmark (submitted to BIT).

12. Rao, T. R. [1975], "Finite field computational techniques for exact solution of numerical problems", Ph. D. dissertation, Department of Applied Mathematics, Indian Institute of Science, Bangalore, India.

13. Szabo', N. S. and Tanaka, R. I. [1967], Residue Arithmetic and Its Applications to Computer Technology, Negraw-Hill, New York.

14. Young, D. M. and Gregory, R. T. [1973], A Survey of Numerical Mathematics, Vol. II, Addison Wesley, Reading, Mass.

APPENDICES

APPENDIX A

ASSEMBLER MACROS

```
         PRINT   OFF
         TITLE   '- - - -    ASSEMBLER MACROS    - -'
*********************************************************************
*                  - - -     L O E N T R Y     - - -
*
*        THIS MACRO SETS UP ENTRY TO A LOW LEVEL ROUTINE THAT
*    CALLS NO OTHER ROUTINES.
*
*        ADDRESSABILITY IS ESTABLISHED USING R15 AS A BASE
*    REGISTER, THE CSECT NAME IS RECORDED IN STORAGE, AND
*    REGISTERS ARE STORED IN THE CALLING PROGRAM'S SAVE AREA.
*
*    SAMPLE CALL:    LEFTZERO LOENTRY
*********************************************************************
*
         MACRO
&NAME    LOENTRY
&NAME    CSECT
         USING   &NAME,R15          ESTABLISH ADDRESSABILITY.
         B       *+12               AVOID EXECUTION OF CONSTANT.
         DC      AL1(7),CL7'&NAME'  INSERT CSECT NAME INTO STORAGE.
         STM     R14,R12,12(R13)    SAVE CALLING PGM'S REGISTERS.
         MEND
         SPACE   3
*********************************************************************
*                  - - -     L O R E T U R N     - - -
*
*        THIS MACRO SETS UP THE RETURN FROM A LOW LEVEL ROUTINE
*    THAT CALLS NO OTHER ROUTINES.
*
*        THE CALLING PROGRAM'S REGISTERS ARE RELOADED, X'FF' IS
*    MOVED INTO THE LEFT BYTE OF THE R14 POSITION OF THE CALLING
*    PROGRAM'S SAVE AREA, AND A RETURN IS EFFECTED.
*********************************************************************
*
         MACRO
&LABEL   LORETURN
&LABEL   LM      R14,R12,12(R13)   RELOAD REGS OF CALLING PROGRAM.
         MVI     12(R13),X'FF'     SET FLAG IN CALLING PGM'S SAVE
*                                     AREA TO FLAG SUCCESSFUL RETURN.
         BR      R14               RETURN TO CALLER.
         LTORG
         MEND
         SPACE   3
*********************************************************************
*                  - - -     H I E N T R Y     - - -
*
```

```
*       THIS MACRO SETS UP ENTRY TO A HIGH LEVEL ROUTINE THAT CALLS
*   OTHER ROUTINES.
*
*       MACRO "LOENTRY" IS CALLED TO ESTABLISH TEMPORARY ADDRESSA-
*   BILITY AND TO STORE REGISTERS.  STANDARD LINKAGE OF CALLING
*   AND CALLED PROGRAM SAVE AREAS IS EFFECTED.  A PERMANENT
*   BASE REGISTER IS ESTABLISHED, AND R15 IS DROPPED AS BASE
*   REGISTER.  A 20 WORD SAVE AREA IS ESTABLISHED FOR LINKAGE
*   TO HIGH LEVEL ROUTINES IN ASSEMBLER, PL1, OR FORTRAN.
*
*   PARAMETERS.
*       &BASEREG -  REGISTER USED IN SETTING UP STANDARD LINKAGE
*                   AND WHICH WILL BECOME PERMANENT BASE REGISTER.
*                   R13,R14,R15 MAY NOT BE CHOSEN AS &BASEREG.
*       &SAVAREA --- 20 WORD SAVE AREA FOR CURRENT CSECT
*
*   SAMPLE CALL:  DIVISION HIENTRY  R11,DIVSAVE
*********************************************************************
*
         MACRO
&NAME    HIENTRY  &BASEREG,&SAVAREA
****** STORE CSECT NAME, ESTABLISH SAVE AREA, AND STORE CALLING
****** PROGRAM'S REGISTERS.
&NAME    CSECT
         USING    &NAME,R15          ESTABLISH TEMPORARY ADDRESSABILITY
         B        *+92               BRANCH AROUND CSECT NAME, SAVE AREA
         DC       AL1(7),CL7'&NAME'  INSERT CSECT NAME INTO STORAGE.
&SAVAREA DC       20F'0'             SAVE AREA FOR CURRENT CSECT.
         STM      R14,R12,12(R13)    SAVE CALLING PGM'S REGISTERS.
*
******* SET UP STANDARD LINKAGE AND ESTABLISH NEW BASE REGISTER.
         LA       &BASEREG,&SAVAREA BASE REG. GETS @ LOCAL SAVE AREA.
         ST       R13,4(&BASEREG)    STORE @ OF CALLER'S SAVE AREA
*                                       IN LOCAL SAVE AREA.
         ST       &BASEREG,8(R13)    STORE @ OF LOCAL SAVE AREA
*                                       IN CALLER'S SAVE AREA.
         LR       R13,&BASEREG       POINT R13 TO LOCAL SAVE AREA.
         LR       &BASEREG,R15       SET UP FOR NEW BASE REGISTER,
         USING    &NAME,&BASEREG        AND NOTIFY ASSEMBLER.
         DROP     R15                DROP R15 AS BASE REGISTER.
*
         MEND
         SPACE    3
*********************************************************************
*                 -----   H I R E T U R N   -----
*
*       THIS MACRO SETS UP THE RETURN FROM A HIGH LEVEL ROUTINE
*   THAT CALLS OTHER ROUTINES.
*
*      R13 IS POINTED TO THE SAVE AREA OF THE CALLING PROGRAM, AND
*   MACRO "LORETURN" COMPLETES THE RETURN CODE BY RESTORING REGIS-
*   TERS AND RETURNING TO THE CALLER.
*********************************************************************
```

```
*
         MACRO
&LABEL   HIRETURN
&LABEL   L        R13,4(R13)        POINT R13 TO SAVE AREA OF CALLING
*                                     PROGRAM.
         LORETURN                   RESTORE REGS.; RETURN TO CALLER.
         MEND
         SPACE    3
*********************************************************************
*                  -----   P R E C I S N   -----
*
*        THIS MACRO ESTABLISHES THE PRECISION OF (NUMBER OF FULLWORDS
*    IN) MULTIPLE PRECISION OPERANDS USED IN RESIDUE OR RATIONAL
*    ARITHMETIC.  ARGUMENT "&NOWORDS" IS THE NUMBER OF FULLWORDS (K)
*    EACH OPERAND WILL HAVE, AN INTEGER VALUE IN THE RANGE 1 TO
*    303.  (VALUES FOR RESIDUE ARITHMETIC ARE RESTRICTED TO A
*    CHOICE FROM 2,3,4,5,17,20,41,69,72,101,133,139,303.)  GLOBAL
*    SYMBOL "&KWORDS" MAKES THIS VALUE AVAILABLE TO MACRO
*    "MULTPREC" FOR ESTABLISHING MULTIPLE PRECISION STORAGE.
*
*        THIS MACRO SHOULD BE CALLED ONLY ONCE BY ASSEMBLER SUPPORT
*    ROUTINES.
*
*    SAMPLE CALL:  K    PRECISN  17
*********************************************************************
*
         MACRO
&LABEL   PRECISN  &NOWORDS
         GBLA     &KWORDS           GLOBAL K FOR USE IN "MULTPREC"
&LABEL   DC       F'&NOWORDS'       STORAGE FOR # FULLWORDS IN OPERANDS
&KWORDS  SETA     &NOWORDS          SAVE VALUE OF K FOR MULTPREC.
         MEND
         SPACE    3
*********************************************************************
*                  -----   M U L T P R E C   -----
*
*       THIS MACRO ESTABLISHES STORAGE (DS OR DC) FOR MULTIPLE PRE-
*    CISION (K-WORD) VARIABLES.
*
*    PARAMETERS:
*         &FACTOR  -- AN INTEGER MULTIPLIED BY K TO DETERMINE TOTAL
*                     NUMBER OF WORDS
*         &LEFTVAL -- VALUE (OF FORM F'-1') TO WHICH EACH LEFT WORD
*                     OF VARIABLE WILL BE SET.  IF OMITTED, A "DS" IS
*                     GENERATED.
*         &RTWORDS -- # OF LOW ORDER WORDS TO SET TO &RTVALUE
*         &RTVALUE -- VALUE (OF FORM F'-1') TO WHICH EACH WORD OF
*                     &RTWORDS WILL BE SET.  MUST BE PRESENT IF
*                     &RTWORDS IS INCLUDED.
*
*         &RTWORDS AND &RTVALUE ARE NOT INCLUDED IN A MACRO CALL
*    IF A DS IS DESIRED OR IF ALL WORDS WILL BE SET TO SAME VALUE.
*    WHEN &RTWORDS IS ABSENT BUT &RTVALUE IS PRESENT, ALL BUT THE
```

```
*   LEFTMOST WORD WILL BE SET TO &RTVALUE.
*
*   SAMPLE CALLS:
*        ONE     MULTPREC 1,F'0',1,F'1'        K-WORD CONSTANT OF 1
*        MAXNEG  MULTPREC 1,X'80000000',,F'0   MAXIMUM NEGATIVE INTEGER
*                                              IN K WORDS
*        PROD    MULTPREC 2                    2-K WORD STORAGE
*********************************************************************
*
         MACRO
&LABEL   MULTPREC &FACTOR,&LEFTVAL,&RTWORDS,&RTVALUE
         GBLA     &KWORDS           K -- # FULLWORDS IN OPERANDS,
.*                                    SET BY PREVIOUS CALL TO PRECISN
         LCLC     &RGHTWDS          # WORDS TO BE SET TO &RTVALUE
.******* DETERMINE WHICH PARAMETERS HAVE BEEN SUPPLIED IN MACRO
.******* CALL AND ESTABLISH DS OR DC ACCORDINGLY.
         AIF      (T'&LEFTVAL EQ '0').NOCONS
         AIF      ('&LEFTVAL'(1,1) EQ 'F').RTCHECK
         DS       0F                FORCE FULLWORD BOUNDARY.
.RTCHECK AIF      (T'&RTVALUE EQ '0').NORIGHT
.*
.***** IF ONLY OMITTED PARAMETER IS &RTWORDS, RIGHT K-1 WORDS
.***** ARE SET TO &RTVALUE.  IF BOTH &RTWORDS AND &RTVALUE ARE
.***** PRESENT, THE SPECIFIED # OF LOW WORDS OF THE K-WORD
.***** CONSTANT ARE SET TO THE SUPPLIED VALUE.
&RGHTWDS SETC     '&RTWORDS'        INITIALIZE &RGHTWDS TO SUPPLIED
.*                                    VALUE.
         AIF      (T'&RTWORDS NE '0').GENER
&RGHTWDS SETC     '&KWORDS'.'*'.'&FACTOR'.'-1' SET &RGHTWDS WHEN
.*                                     NO VALUE IS SUPPLIED.
.GENER   ANOP
&LABEL   DC       (&FACTOR*&KWORDS-(&RGHTWDS))&LEFTVAL LEFT WORDS CONST
         DC       (&RGHTWDS)&RTVALUE RIGHT WORDS CONSTANT
         AGO      .EXIT
.*
.***** IF &RTWORDS AND &RTVALUE ARE OMITTED, ALL WORDS WILL BE SET
.***** TO THE SAME VALUE.
.NORIGHT ANOP
&LABEL   DC       (&FACTOR*&KWORDS)&LEFTVAL MULT. PRECISION CONST.
         AGO      .EXIT
.***** IF &LEFTVAL IS OMITTED, ONLY DS WILL BE GENERATED.
.NOCONS  ANOP
&LABEL   DS       (&FACTOR*&KWORDS)F MULTIPLE PRECISION STORAGE
.EXIT    MEND
         SPACE    3
*********************************************************************
*                  -- -     L D A R G T B L     -- --
*
*         THIS MACRO CREATES AN ARGUMENT TABLE OF ARGUMENT ADDRESSES
*    FOR A CALL TO A SUBROUTINE.
*
*    PARAMETERS:
*         &TABNAME -- OPTIONAL NAME FOR THE ARGUMENT TABLE;  IF NONE
```

```
*                       IS GIVEN, &SYSNDX IS USED TO CREATE A DEFAULT
*                       TABLE LABEL.
*          &ARGLIST --- ARGUMENTS FOR THE SUBROUTINE CALL, SEPARATED BY
*                       COMMAS.  ARGUMENT ADDRESSES ARE INSERTED IN THE
*                       TABLE IN THE ORDER SUPPLIED AND MAY BE OF THE
*                       FORM ...
*                LASTNUM  --- AN IMPLICT ADDRESS WHERE THE ARGUMENT
*
*                             IS FOUND (MAY NOT BEGIN WITH
*                             CHARACTERS R0 ---> R9)
*                @LASTNUM -- AN IMPLICIT @ WHERE THE @ OF THE ARGUMENT
*                             MAY BE FOUND
*                @4(R9)   -- A D(B) @ WHERE THE @ OF THE ARGUMENT MAY
*                             BE FOUND
*                R12      --- A REGISTER CONTAINING THE @ OF THE ARGUMENT
*
*   SAMPLE CALL:
*       LDARGTBL RESTBL,(K,@4(R1),@8(R1),R7,R8,MODULUS,RESULT,@RSLTDEN)
*********************************************************************
*
         MACRO
&LABEL  LDARGTBL &TABNAME,&ARGLIST
.***** ESTABLISH GLOBAL AND LOCAL VARIABLES.
         GBLC     &DFARGNM           DEFAULT NAME FOR ARGUMENT TABLE
.*                                     --- AVAILABLE TO "CALL" MACRO
         LCLC     &OPERADD           LABEL LOCATION WITH @ OF ARGUMENT
         LCLA     &OFFSET            OFFSET OF ARG. ADDR IN TABLE
         LCLC     &ARGSTRT           NAME USED FOR ARGUMENT TABLE
         LCLA     &ARGNO             NUMBER OF ARGUMENTS IN CALL
         LCLA     &ARGNDX            COUNTER IN ARGUMENT LIST
&DFARGNM SETC     'ARGS'.'&SYSNDX'   STORE DEFAULT ARG. TABLE NAME
.*                                     FOR "CALL" MACRO.
&ARGNO   SETA     N'&ARGLIST         ESTABLISH # OF ARGUMENTS.
&ARGNDX  SETA     1                  INITIALIZE ARGUMENT COUNTER.
.*
&LABEL   B        GEAD&SYSNDX        AVOID EXECUTION OF ARGUMENT LIST.
.***** GENERATE A HEADING FOR THE ARGUMENT TABLE.
         AIF      (T'&TABNAME EQ '0').NOTABNM  NO SUPPLIED NAME?
&TABNAME DS       0F                 HEAD OF ARGUMENT LIST
&ARGSTRT SETC     '&TABNAME'         RECORD ARG. LIST NAME.
         AGO      .ARGLOOP
.NOTABNM ANOP
ARGS&SYSNDX DS    0F                 HEAD OF ARGUMENT LIST
&ARGSTRT SETC     '&DFARGNM'         RECORD ARG. LIST NAME.
.*
.***** GENERATE A FULL WORD OF STORAGE FOR EACH ARGUMENT'S ADDRESS.
.***** DC'S ARE GENERATED FOR ARGUMENTS THAT ARE IMPLICIT ADDRESSES.
.ARGLOOP AIF      ('&ARGLIST(&ARGNDX)'(1,1) EQ '@'                        1
               OR ('&ARGLIST(&ARGNDX)'(1,2) GE 'R0' AND                   2
               '&ARGLIST(&ARGNDX)'(1,2) LE 'R9')).ADDR  --- DS??
         AIF      (&ARGNDX EQ &ARGNO).LASTDC LAST ARGUMENT (A DC)?
         DC       A(&ARGLIST(&ARGNDX)) GENERATE NEXT ARGUMENT @.
         AGO      .INCRMNT
```

```
.ADDR    DS       F                  STORAGE FOR NEXT ARGUMENT @
         AIF      (&ARGNDX EQ &ARGNO).ARGMOV  LAST DS?
.INCRMNT ANOP
&ARGNDX  SETA     &ARGNDX+1          INCREMENT ARGUMENT COUNTER.
         AGO      .ARGLOOP           GO CREATE NEXT ARGUMENT @.
.*
.LASTDC  DC       X'80',AL3(&ARGLIST(&ARGNDX)) @ OF LAST ARGUMENT;
*                                                LEFT BYTE IS MARKED.
.*
.******** WE NOW MOVE ARGUMENT ADDRESSES FROM REGISTERS/STORAGE INTO
.******** ARGUMENT TABLE DS'S.
.ARGMOV  ANOP
GEAD&SYSNDX DS    0H
&ARGNDX  SETA     1                  RE-INITIALIZE ARGUMENT COUNTER.
.**** IF "ARG" IS REALLY @ OF THE ARGUMENT, DETERMINE TYPE OF ADDRESS
.**** SPECIFICATION AND MOVE @ INTO ARGUMENT TABLE.
.*
.*
.ADRLOOP AIF      ('&ARGLIST(&ARGNDX)'(1,1) EQ '@'                    1
               OR ('&ARGLIST(&ARGNDX)'(1,2) GE 'R0' AND               2
                '&ARGLIST(&ARGNDX)'(1,2) LE 'R9')).ADDRGEN --DS??
         AIF      (&ARGNDX EQ &ARGNO).EXIT   IF LAST ARG. IS A
.*                                             LABEL, QUIT.
         AGO      .ADDNDX
.ADDRGEN ANOP
&OFFSET  SETA     4*(&ARGNDX-1)      SET BYTE OFFSET OF ARG. @ IN TABLE
         AIF      ('&ARGLIST(&ARGNDX)'(1,1) NE '@').REG  START WITH @?
.*
.**** ADDRESS IS FOUND AT AN ADDRESS.  ARGUMENT IN CALL IS OF THE FORM
.**** @4(R6) OR @ARGADDR.
&OPERADD SETC     '&ARGLIST(&ARGNDX)'(2,8)  STRIP "@" FROM ARG.
         MVC    &ARGSTRT+&OFFSET.(4),&OPERADD  MOVE @ INTO ARG. LIST.
         AGO      .LASTCHK           GO INCREMENT ARG. COUNTER
.*
.**** ADDRESS IS IN REGISTER (ARGUMENT IN CALL IS OF FORM  "R4").
.REG     ST       &ARGLIST(&ARGNDX),&ARGSTRT+&OFFSET STORE  REG. @ IN
*                                                    IN ARGUMENT TABLE.
.LASTCHK ANOP
         AIF      (&ARGNDX NE &ARGNO).ADDNDX  IF NOT LAST ARGUMENT,
.*                                            INCREMENT ARG. COUNTER.
         MVI      &ARGSTRT+&OFFSET,X'80' MARK LAST ARGUMENT LEFT BYTE.
         AGO      .EXIT
.ADDNDX  ANOP
&ARGNDX  SETA     &ARGNDX+1          INCREMENT ARGUMENT POINTER.
         AGO      .ADRLOOP           TRY NEXT ARGUMENT.
.*
.*
.EXIT    MEND
         SPACE    3
*********************************************************************
*                 - - - -    C A L L    - - - -
*
*       THIS MACRO SETS UP FOR, AND MAKES, A CALL TO A
```

```
*    SUBROUTINE.
*
*    PARAMETERS:
*        &SUBNAME -- NAME OF CSECT CALLED
*        &TABNAME -- NAME OF ARGUMENT LIST GENERATED FOR THE CALL
*        &ARGLIST -- ARGUMENTS FOR THE SUBROUTINE CALL, SEPARATED
*                    BY COMMAS.  PERMISSIBLE FORMS ARE DESCRIBED
*                    IN PRECEDING DESCRIPTION OF MACRO LDARGTBL.
*
*        IF &ARGLIST IS PRESENT, A CALL WILL BE MADE TO LDARGTBL
*    TO GENERATE A TABLE OF ARGUMENT ADDRESSES.
*        IF &TABNAME IS PRESENT AND &ARGLIST IS MISSING, ONLY LA,
*    L, AND BALR WILL BE GENERATED.  A PREVIOUS CALL TO LDARGTBL WITH
*    "&TABNAME" SHOULD HAVE BEEN MADE TO SET UP THE ARGUMENT LIST.
*        IF BOTH &TABNAME AND &ARGLIST ARE MISSING, ONLY L AND BALR
*    WILL BE MADE (THERE ARE NO ARGUMENTS IN THE SUBROUTINE
*    CALL).
*
*    SAMPLE CALLS:  CALL OUTVRT,OUTTBL,(K,R3,R4)
*                   CALL OUTVRT,OUTTBL,
*                   CALL OUTVRT,,(K,R3,R4)
*********************************************************************
*
         MACRO
&LABEL   CALL     &SUBNAME,&TABNAME,&ARGLIST
         GBLC     &DFARGNM            DEFAULT ARG. TABLE NAME SUPPLIED
.*                                        BY LDARGTBL
&LABEL   DS       0H                  HANG OPTIONAL LABEL HERE.
.*
.****** DETERMINE KIND OF MACRO CALL AND TYPE OF SUBROUTINE
.****** CALL AND ACT ACCORDINGLY.
.*** (1)  CHECK FOR SIMPLE SUBROUTINE CALL WITH NO ARGUMENTS.
.*** (2)  CHECK FOR ARGUMENT TABLE CONSTRUCTED PREVIOUSLY.
.*** (3)  CHECK FOR CALL WITH ARGUMENTS, NO SUPPLIED ARGUMENT
.***       TABLE NAME.
         AIF      (T'&TABNAME EQ 'O' AND T'&ARGLIST EQ 'O').NOTBARG
         AIF      (T'&TABNAME NE 'O' AND T'&ARGLIST EQ 'O').TBNOARG
         AIF      (T'&TABNAME EQ 'O').DFTABNM
.*
.*** SUBROUTINE CALL WITH ARGUMENTS AND A SUPPLIED ARGUMENT
.*** LIST NAME.
         LDARGTBL &TABNAME,&ARGLIST BUILD ARGUMENT LIST WITH
.*                                      SUPPLIED NAME.
         AGO      .TBNOARG
.*** SIMPLE SUBROUTINE CALL WITH NO ARGUMENTS.
.NOTBARG SR       R1,R1               ZERO R1 TO INDICATE NO ARGUMENTS.
         AGO      .DOCALL
.*** SUBROUTINE CALL WITH ARGUMENTS, BUT NO SUPPLIED ARGUMENT
.*** TABLE NAME.
.DFTABNM LDARGTBL ,&ARGLIST           BUILD ARGUMENT LIST (DEFAULT NAME)
         LA       R1,&DFARGNM         LOAD @ OF DEFAULT-NAMED ARG. LIST.
         AGO      .DOCALL
.*
```

```
***** LOAD ARGUMENT TABLE ADDRESS AND MAKE CALL TO SUBROUTINE.
.TBNOARG LA       R1,&TABNAME       LOAD @ OF ARGUMENT LIST.
.DOCALL  L        R15,=V(&SUBNAME)  PREPARE R15 FOR CALL TO SUB-
         BALR     R14,R15              ROUTINE USING BALR.
         MEND
         SPACE    3
*********************************************************************
*                  - - -   Z E R O L O N G   - - -
*
*        THIS MACRO ZEROS A D(B) STORAGE AREA OF 1 OR MORE BYTES
*   (NORMALLY > 256).
*
*   PARAMETERS:
*        &LENREG  -- CONTAINS THE LENGTH CODE (# BYTES - 1) OF THE
*                    AREA TO BE ZEROED
*        &STORAGE -- AN IMPLICIT ADDRESS OR AN EXPLICIT D(B)
*                    ADDRESS OF THE AREA TO BE ZEROED
*
*        BECAUSE R4 AND R5 ARE USED AS WORK REGISTERS WITHIN THE
*   GENERATED MACRO CODE, THEY SHOULD NOT BE USED AS "&LENREG" OR
*   AS THE D(B) ADDRESS REGISTER FOR "&STORAGE".  A CSECT WHICH
*   CALLS THIS MACRO SHOULD NOT USE AS ITS BASE REGISTER R4 OR R5.
*   R4 AND R5 ARE NOT PERMANENTLY ALTERED BY THE MACRO'S CODE.
*
*   SAMPLE CALLS:  ZEROLONG R3,14(R6)
*                  ZEROLONG R2,TABLE+12
*********************************************************************
*
         MACRO
&LABEL   ZEROLONG &LENREG,&STORAGE
*
****** STORE WORK REGISTERS AND SET UP FOR "ZERO" LOOP.
&LABEL   STM      R4,R5,SAV&SYSNDX  STORE R4,R5 AS WORK REGISTERS.
         ST       &LENREG,SAV&SYSNDX+8 SAVE LENGTH REGISTER.
         LA       R4,256            LOAD MAXIMUM MOVE (256 BYTES).
         LA       R5,&STORAGE       GET BEGINNING @ OF AREA TO BE
*                                     ZEROED.
*
****** THIS LOOP ZEROS ALL REQUESTED BYTES IN EXCESS OF 256.
DEC&SYSNDX CR     &LENREG,R4        <= 256 BYTES TO BE ZEROED?
         BL       L256&SYSNDX       IF SO, ZERO THEM AND QUIT.
         XC       0(256,R5),0(R5)   ZERO NEXT 256 BYTES.
         SR       &LENREG,R4        SUBTRACT 256 FROM REMAINING
*                                     BYTES TO BE ZEROED.
         LA       R5,256(R5)        INCREMENT @ TO BE ZEROED.
         B        DEC&SYSNDX        LOOP FOR NEXT ZERO.
*
****** ZERO REMAINING BYTES (A NUMBER <= 256).
L256&SYSNDX  EX   &LENREG,ZERO&SYSNDX ZERO LAST BYTES.
         B        SAV&SYSNDX+12     ZERO REMAINING BYTES (<= 256).
ZERO&SYSNDX XC    0(*-*,R5),0(R5)   INSTR. COPY FOR ZEROING
SAV&SYSNDX DS     3F                SAVE WORDS FOR R4,R5, LENGTH REG.
         LM       R4,R5,SAV&SYSNDX  RESTORE TRASHED REGISTERS.
```

```
         L         &LENREG,SAV&SYSNDX+8 RESTORE LENGTH REGISTER.
*
.EXIT    MEND
         SPACE    3
*********************************************************************
*                  ----    S E T L O N G    ----
*
*        THIS MACRO SETS A D(B) STORAGE AREA OF 1 OR MORE BYTES
*    (NORMALLY > 256) TO A SPECIFIED BYTE VALUE.
*
*    PARAMETERS.
*         &LENREG  --- CONTAINS THE LENGTH CODE (# BYTES - 1) OF
*                      THE AREA TO BE SET TO SOME BYTE VALUE
*         &STORAGE --- AN IMPLICIT ADDRESS OR AN EXPLICIT D(B)
*                      ADDRESS OF THE AREA TO BE SET
*         &VALUE   --- A VALUE ( IN THE FORM OF AN IMMEDIATE BYTE)
*                      TO WHICH EACH BYTE OF &STORAGE IS TO BE SET
*
*        BECAUSE R4 AND R5 ARE USED AS WORK REGISTERS WITHIN THE
*    GENERATED MACRO CODE, THEY SHOULD NOT BE USED AS "&LENREG" OR
*    AS THE D(B) ADDRESS REGISTER FOR "&STORAGE".  A CSECT WHICH
*    CALLS THIS MACRO SHOULD NOT USE AS ITS BASE REGISTER R4 OR
*    R5.  R4 AND R5 ARE NOT PERMANENTLY ALTERED BY THE MACRO'S
*    CODE.
*
*    SAMPLE CALLS:  SETLONG R3,14(R6),C'*'
*                   SETLONG R2,TABLE+12,X'F5'
*********************************************************************
*
         MACRO
&LABEL   SETLONG  &LENREG,&STORAGE,&VALUE
*
****** STORE WORK REGISTERS AND SET UP FOR "SET" LOOP.
&LABEL   STM      R4,R5,SV&SYSNDX   STORE R4,R5 AS WORK REGISTERS.
         ST       &LENREG,SV&SYSNDX+8 SAVE LENGTH REGISTER.
         LA       R4,256            LOAD MAXIMUM MOVE (256 BYTES).
         LA       R5,&STORAGE       GET BEGINNING @ OF AREA TO BE SET.
         MVI      0(R5),&VALUE      MOVE VALUE BYTE INTO FIRST BYTE
*                                       OF STORAGE.
         BCTR     &LENREG,R0        ADJUST LENGTH CODE FOR # BYTES TO
*                                     MOVE.
         LTR      &LENREG,&LENREG   CHECK # BYTES TO MOVE.
         BM       SV&SYSNDX+12      IF ONLY ONE BYTE, QUIT.
*
****** THIS LOOP SETS ALL REQUESTED BYTES IN EXCESS OF 256.
NXT&SYSNDX CR     &LENREG,R4        < = 256 BYTES TO BE SET?
         BL       BLOW&SYSNDX       IF SO, SET THEM AND QUIT.
         MVC      1(256,R5),0(R5)   SET NEXT 256 BYTES.
         SR       &LENREG,R4        SUBTRACT 256 FROM REMAINING BYTES
*                                     TO BE SET.
         LA       R5,256(R5)        INCREMENT ADDRESS TO BE SET.
         B        NXT&SYSNDX        LOOP FOR NEXT BYTES.
*
```

```
****** SET REMAINING BYTES (A NUMBER <= 256).
BLOW&SYSNDX EX    &LENREG,SET&SYSNDX SET LAST BYTES
         B        SV&SYSNDX+12          AND QUIT.
SET&SYSNDX MVC    1(*-*,R5),0(R5)    INSTRUCTION COPY FOR BYTE SET
SV&SYSNDX DS      3F                 SAVE WORDS FOR R4,R5, LENGTH REG.
         LM       R4,R5,SV&SYSNDX    RESTORE TRASHED REGISTERS.
         L        &LENREG,SV&SYSNDX+8 RESTORE LENGTH REGISTER.
*
.EXIT    MEND
         SPACE    3
*********************************************************************
*                 -----    M O V E L O N G    -----
*
*        THIS MACRO MOVES 1 OR MORE CONSECUTIVE BYTES (NORMALLY
*   > 256) FROM ONE D(B) STORAGE LOCATION TO ANOTHER.
*
*   PARAMETERS:
*        &LENREG  -- CONTAINS THE LENGTH CODE (# BYTES - 1) OF THE
*                    NUMBER OF BYTES TO BE MOVED
*        &TO      -- AN IMPLICIT ADDRESS OR AN EXPLICIT D(B)
*                    ADDRESS OF THE AREA TO WHICH MOVE IS MADE
*        &FROM    -- AN IMPLICIT ADDRESS OR AN EXPLICIT D(B)
*                    ADDRESS FROM WHICH BYTES WILL BE MOVED
*
*        BECAUSE R4, R5, AND R6 ARE USED AS WORK REGISTERS WITHIN
*   THE GENERATED MACRO CODE, THEY SHOULD NOT BE USED AS "&LENREG"
*   OR AS THE D(B) ADDRESS REGISTER FOR "&TO" OR "&FROM".  A CSECT
*   WHICH CALLS THIS MACRO SHOULD NOT USE AS ITS BASE REGISTER
*   R4, R5, OR R6.  HOWEVER, THESE REGISTERS ARE NOT PERMANENTLY
*   ALTERED BY THE MACRO'S CODE.
*
*   SAMPLE CALL:  MOVELONG R3,14(R6),TABLE+12
*********************************************************************
*
         MACRO
&LABEL   MOVELONG &LENREG,&TO,&FROM
*
****** STORE WORK REGISTERS AND SET UP FOR "MOVE" LOOP.
&LABEL   STM      R4,R6,KEEP&SYSNDX STORE R4-R6 FOR WORK REGISTERS.
         ST       &LENREG,KEEP&SYSNDX+12 STORE LENGTH REGISTER.
         LA       R4,256            LOAD MAXIMUM MOVE (256 BYTES).
         LA       R5,&TO            LOAD BEGINNING ADDRESS TO WHICH
*                                      TO MOVE BYTES.
         LA       R6,&FROM          LOAD BEGINNING @ OF AREA FROM
*                                      WHICH BYTES ARE MOVED.
*
****** THIS LOOP MOVES ALL REQUESTED BYTES IN EXCESS OF 256.
AGN&SYSNDX CR     &LENREG,R4        <= 256 BYTES TO BE MOVED?
         BL       FEW&SYSNDX        IF SO, MOVE THEM AND QUIT.
         MVC      0(256,R5),0(R6)   MOVE NEXT 256 BYTES.
         SR       &LENREG,R4        SUBTRACT 256 FROM REMAINING
*                                      BYTES TO BE MOVED.
         LA       R5,256(R5)        INCREMENT ADDRESSES OF
```

```
         LA        R6,256(R6)            STORAGE LOCATIONS.
         B         AGN&SYSNDX          LOOP FOR NEXT MOVE.
*
****** MOVE REMAINING BYTES (A NUMBER <= 256).
FEW&SYSNDX EX      &LENREG,MV&SYSNDX MOVE LAST BYTES
         B         KEEP&SYSNDX+16        AND QUIT.
MV&SYSNDX MVC      0(*-*,R5),0(R6)    INSTR. COPY FOR BYTE MOVE.
KEEP&SYSNDX DS     4F                 SAVE WORDS FOR R4-R6, LENGTH REG.
         LM        R4,R6,KEEP&SYSNDX RESTORE TRASHED REGISTERS.
         L         &LENREG,KEEP&SYSNDX+12 RESTORE LENGTH REGISTER.
*
         MEND
         SPACE     3
*********************************************************************
*                  - - - -   C M P L O N G   - - - -
*
*        THIS MACRO PERFORMS A LOGICAL COMPARE BETWEEN TWO AREAS
*    OF LENGTH 1 OR MORE BYTES (NORMALLY > 256).  THE CONDITION
*    CODE IS SET AS IN THE MACHINE CLC INSTRUCTION.
*
*    PARAMETERS:
*         &LENREG    -- CONTAINS THE LENGTH CODE (# BYTES - 1) OF
*                       THE AREAS TO BE COMPARED
*         &LOCATN1   -- AN IMPLICIT ADDRESS OR AN EXPLICIT D(B)
*                       ADDRESS OF THE FIRST OPERAND
*         &LOCATN2   -- AN IMPLICIT ADDRESS OR AN EXPLICIT D(B)
*                       ADDRESS OF THE SECOND OPERAND
*
*         BECAUSE R4, R5, AND R6 ARE USED AS WORK REGISTERS WITHIN
*    THE GENERATED MACRO CODE, THEY SHOULD NOT BE USED AS "&LENREG"
*    OR AS THE D(B) ADDRESS REGISTER FOR "&LOCATN1" OR "&LOCATN2".
*    A CSECT WHICH CALLS THIS MACRO SHOULD NOT USE AS ITS BASE
*    REGISTER R4, R5, OR R6.  HOWEVER, THESE REGISTERS ARE NOT PER-
*    MANENTLY ALTERED BY THE MACRO'S CODE.
*
*    SAMPLE CALL:  CMPLONG R3,14(R6),TABLE+12
*********************************************************************
*
         MACRO
&LABEL   CMPLONG &LENREG,&LOCATN1,&LOCATN2
*
****** STORE WORK REGISTERS AND SET UP FOR COMPARE LOOP.
&LABEL   STM       R4,R6,STO&SYSNDX  STORE R4-R6 FOR WORK REGISTERS.
         ST        &LENREG,STO&SYSNDX+12 STORE LENGTH REGISTER.
         LA        R4,256            LOAD MAXIMUM COMPARE (256 BYTES).
         LA        R5,&LOCATN1       LOAD BEGINNING ADDRESS AT WHICH
*                                       TO COMPARE BYTES.
         LA        R6,&LOCATN2       LOAD BEGINNING C OF 2ND AREA AT
*                                       WHICH BYTES ARE COMPARED.
*
****** THIS LOOP COMPARES ALL REQUESTED BYTES IN EXCESS OF 256.
MORE&SYSNDX CR     &LENREG,R4        <= 256 BYTES TO BE COMPARED?
         BL        NO&SYSNDX         IF SO, COMPARE THEM AND QUIT.
```

```
         CLC       0(256,R5),0(R6)   COMPARE NEXT 256 BYTES.
         BNE       STO&SYSNDX+16     IF DIFFERENT, QUIT.
         SR        &LENREG,R4        SUBTRACT 256 FROM REMAINING
*                                        BYTES TO BE COMPARED.
         LA        R5,256(R5)        INCREMENT ADDRESSES OF STORAGE
         LA        R6,256(R6)           LOCATIONS.
         B         MORE&SYSNDX       LOOP FOR NEXT COMPARE.
*
****** COMPARE REMAINING BYTES (A NUMBER <=  256).
NO&SYSNDX EX       &LENREG,CMP&SYSNDX COMPARE LAST BYTES
         B         STO&SYSNDX+16        AND QUIT.
CMP&SYSNDX CLC     0(*-*,R5),0(R6)   INSTR. COPY FOR BYTE COMPARE
STO&SYSNDX DS   4F                   SAVE WORDS FOR R4-R6, LENGTH REG.
         LM        R4,R6,STO&SYSNDX RESTORE TRASHED REGISTERS.
         L         &LENREG,STO&SYSNDX+12 RESTORE LENGTH REGISTER.
*
         MEND
         SPACE     3
*********************************************************************
*                  ----   E R R O R C H K   ----
*
*        THIS MACRO CHECKS AN ERROR FLAG AFTER THE RETURN FROM A SUB-
*    ROUTINE AND ACTS APPROPRIATELY.  IF THE ERROR FLAG IS SET, A
*    BRANCH IS MADE TO A DESIGNATED LABEL, NORMALLY AN EXIT FROM THE
*    CURRENT SUBROUTINE.  IF &BRANCH IS "FERROR", THE ERROR IS CONSID-
*    ERED FATAL, AND PROCEDURE "FERROR" IS CALLED TO ABORT EXECUTION.
*    IF THE FLAG IS SET AND IF PARAMETER "&FLAGSET" IS PRESENT, THE
*    ERROR FLAG IS RE-SET TO THE SUPPLIED VALUE BEFORE APPROPRIATE
*    ACTION IS TAKEN.
*
*    PARAMETERS:
*        &FLAG     -- A ONE-WORD ERROR FLAG.  THE FLAG IS SET TO AN
*                     ERROR CONDITION IF THE RIGHTMOST BYTE IS NON-
*                     ZERO.
*                     LEGAL FORMS OF &FLAG:
*                         -- AN IMPLICIT LABEL SUCH AS "OVFLOFLG"
*                         -- A REGISTER CONTAINING THE ADDRESS OF
*                            THE ERROR FLAG
*        &BRANCH   -- AN IMPLICIT LABEL AT WHICH EXECUTION RESUMES IF
*                     THE ERROR FLAG IS SET.  THE VALUE "FERROR" IS
*                     USED IF EXECUTION IS TO CEASE.
*        &FLAGSET  -- A DECIMAL INTEGER TO WHICH &FLAG IS RE-SET (IF
*                     ALREADY TURNED ON)
*
*    SAMPLE CALLS:  ERRORCHK  ERFLAG,EXIT
*                   ERRORCHK  R5,FERROR,58
*
*    ----  THE CONTENTS OF R1 ARE DESTROYED BY THE MACRO.  ----
*********************************************************************
*
         MACRO
&LABEL   ERRORCHK &FLAG,&BRANCH,&FLAGSET
.*
```

```
.***** INITIALIZE LCLC CHARACTER VARIABLES.
         LCLC     &FLAGADD          D(X,B) LOCATION OF ERROR FLAG
         LCLC     &FLGADD2          LOCATION OF LOW BYTE OF ERROR FLAG
         LCLC     &LABELAD          "-3" EXTENDER FOR LOW BYTE LABEL
&FLAGADD SETC     '&FLAG'           INITIALIZE &FLAGADD,
&FLGADD2 SETC     '&FLAG'               &FLGADD2.
&LABELAD SETC     '-3'              SET LABEL EXTENDER DEFAULT.
.*
.***** IF "&FLAG" IS A REGISTER CONTAINING THE ERROR FLAG ADDRESS,
.***** CONVERT THE REGISTER INTO A D(B) ADDRESS.
         AIF      (NOT('&FLAG'(1,1) EQ 'R'                              1
               AND '&FLAG'(2,1) GE '0'                                  2
               AND '&FLAG'(2,1) LE '9')).CHKFLAG
&FLAGADD SETC     '0('.'&FLAG'.')'
&FLGADD2 SETC     '3('.'&FLAG'.')'
&LABELAD SETC     ''                BLANK LABEL EXTENDER FOR REGISTER.
.*
.***** TEST WHETHER ERROR FLAG IS SET.  IF IT IS, RESET FLAG AS
.***** SPECIFIED BY CALLER.
.CHKFLAG TM       &FLGADD2&LABELAD,X'FF' TEST IF ERROR FLAG IS SET.
         AIF    (T'&FLAGSET EQ 'O' AND '&BRANCH' NE 'FERROR').BRONLY
         BZ       GO&SYSNDX         IF ERROR FLAG NOT SET, WE ARE DONE
         AIF      (T'&FLAGSET EQ 'O').NORESET  IS &FLAGSET OMITTED?
         LA       R1,&FLAGSET       LOAD ERROR RECODE VALUE
         ST       R1,&FLAGADD          AND STORE IN ERROR FLAG.
.*
.***** TEST FOR FATAL ERROR (ABORT EXECUTION?).  IF ERROR IS NOT
.***** FATAL, TAKE SUPPLIED BRANCH.
.NORESET AIF      ('&BRANCH' EQ 'FERROR').CALLF  IS ERROR FATAL?
         B        &BRANCH           ERROR NOT FATAL
         AGO      .OUT
.BRONLY  BNZ      &BRANCH           UPON ERROR, BRANCH AS DIRECTED.
         AGO      .OUT2
.CALLF   CALL     FERROR,,(&FLAG)   FATAL ERROR--TERMINATE EXECUTION.
.*
.OUT     ANOP
GO&SYSNDX DS      0H                LABEL FOR BRANCH
.OUT2    MEND
         SPACE    3
*********************************************************************
*                  - - -    O V E R C H K    - - - -
*
*        THIS MACRO CHECKS A 2-K WORD 2'S COMPLEMENT PRODUCT FROM
*    THE MULTIPLICATION OF TWO K-WORD INTEGERS FOR OVERFLOW INTO THE
*    LEFT (HIGH) K-WORDS OR INTO THE SIGN BIT OF THE LOW K-WORDS.
*    OVERFLOW OCCURS IF  (1) THE SIGN BIT OF THE RIGHT K WORDS IS 1,
*                            BUT THE LEFT K-WORDS ARE NOT ALL SET
*                            TO "-1".
*                        (2) THE SIGN BIT OF THE RIGHT K WORDS IS 0,
*                            BUT THE LEFT K-WORDS ARE NOT ALL "0'S".
*    MACRO "CMPLONG" IS EMPLOYED TO MAKE THE NECESSARY COMPARISONS.
*    IF OVERFLOW IS DETECTED, EXECUTION ABORTS WITH FATAL ERROR
*    NUMBER "&ERRNBR".
```

```
*
*   PARAMETERS:
*       &LENREG  -- A REGISTER CONTAINING THE LENGTH CODE (4K - 1)
*                   FOR K-WORD COMPARISONS
*       &DBLADDR -- THE ADDRESS OF THE 2-K WORD AREA CHECKED FOR
*                   OVERFLOW.  MAY BE AN IMPLICIT ADDRESS OR AN EX-
*                   PLICIT D(B) ADDRESS.
*       &ERRNBR  -- A DECIMAL INTEGER TO WHICH THE ERROR FLAG IS SET
*                   IF OVERFLOW IS DETECTED.
*
*         BECAUSE R4, R5, AND R6 ARE USED AS WORK REGISTERS WITHIN
*   GENERATED CODE (INCLUDING THAT FROM MACRO "CMPLONG"), THEY
*   SHOULD NOT BE USED AS "&LENREG" OR AS THE D(B) ADDRESS REGISTER
*   FOR "&DBLADDR".  A CSECT WHICH CALLS THIS MACRO SHOULD NOT USE
*   AS ITS BASE REGISTER R4, R5, OR R6.  HOWEVER, THESE REGISTERS
*   ARE NOT PERMANENTLY ALTERED BY THE MACRO'S CODE.
*
*   SAMPLE CALL:  OVERCHK  R2,0(R14),11
*********************************************************************
*
         MACRO
&LABEL   OVERCHK  &LENREG,&DBLADDR,&ERRNBR
         LCLC     &OPNDCHR          OPERAND FOR LA OF D(B) ADDRESS
*
****** SAVE WORK REGISTER AND BRANCH AROUND STORAGE.
&LABEL   ST       R5,RGSV&SYSNDX    SAVE R5 DURING OVERCHK EXECUTION.
         B        PAST&SYSNDX       BRANCH AROUND STORAGE.
MIN&SYSNDX DC     A(MINUS1)         @ OF K WORDS OF BINARY 1'S
ZERO&SYSNDX DC    A(ZERO)           @ OF K WORDS OF BINARY 0'S
RGSV&SYSNDX DS    F                 SAVE WORD FOR R5
****** ESTABLISH ADDRESS OF SECOND K WORDS OF THE PRODUCT AND LOAD
****** INTO R5.
         AIF      ('&DBLADDR'(K'&DBLADDR,1) EQ ')').REGADD  D(B) @?
PAST&SYSNDX LA    R5,&DBLADDR+1(&LENREG) LOAD IMPLICIT @ OF 2ND K WORDS
         AGO      .TEST
.REGADD  ANOP
&OPNDCHR SETC     '&DBLADDR'(1,K'&DBLADDR-1)
PAST&SYSNDX LA    R5,&OPNDCHR,&LENREG)  @ IS OF FORM D(B,&LENREG).
         LA       R5,1(R5)          POINT R5 TO SECOND K WORDS.
*
****** TEST FOR OVERFLOW OF 2K-WORD PRODUCT INTO LEFT K-WORDS --
****** RIGHT K-WORD SIGN BIT = 1 BUT LEFT K-WORDS NOT ALL "-1'S", OR
****** RIGHT K-WORD SIGN BIT = 0 BUT LEFT K-WORDS NOT ALL "0'S".
.TEST    TM       0(R5),X'80'       TEST SIGN BIT OF SECOND K WORDS.
         BO       NEG&SYSNDX        IF SET, CHECK LEFT K WORDS FOR 1'S
         L        R5,ZERO&SYSNDX    LOAD ADDRESS OF K WORDS OF 0'S.
         CMPLONG  &LENREG,0(R5),&DBLADDR  ARE LEFT K WORDS 0'S?
         BNE      OVER&SYSNDX       IF NOT, WE HAVE OVERFLOW.
         B        OK&SYSNDX         OTHERWISE, CORRECT RESULT IS IN
*                                      RIGHT K WORDS.
NEG&SYSNDX L      R5,MIN&SYSNDX     LOAD ADDRESS OF K WORDS OF 1'S.
         CMPLONG  &LENREG,0(R5),&DBLADDR  ARE LEFT K WORDS "-1"?
         BE       OK&SYSNDX         IF SO, NO OVERFLOW
```

```
*
****** IF OVERFLOW HAS OCCURRED, STORE ERROR FLAG VALUE SPECIFIED
****** BY THE CALLER AND CALL "FERROR" TO PRINT AN ERROR MESSAGE
****** AND ABORT EXECUTION.
OVER&SYSNDX LA    R5,&ERRNBR        LOAD FATAL ERROR NUMBER
         ST       R5,RGSV&SYSNDX       AND STORE AS ERROR FLAG.
         CALL     FERROR,,(RGSV&SYSNDX)
OK&SYSNDX L       R5,RGSV&SYSNDX    RESTORE R5 TO ORIGINAL VALUE.
         MEND
         SPACE    3
*********************************************************************
*              ------    B Y T E O F F    ---
*
*        THIS MACRO COMPUTES THE BYTE OFFSET OF THE LOW WORD OR
*   BYTE IN A MULTIPLE-PRECISION INTEGER (K WORDS).  AT THE TIME OF
*   MACRO CALL, R1 IS ASSUMED TO CONTAIN AN ADDRESS WHERE THE ADDRESS
*   OF K IS FOUND.  THE COMPUTED OFFSET IS LOADED INTO A DESIGNATED
*   REGISTER "&REG" AND, IF REQUESTED, STORED AT A SPECIFIED LOCATION
*   "&STORLOC".
*
*   PARAMETERS:
*       &OFFTYPE --- CHARACTER VALUE "WORD" OR "BYTE" TO INDICATE
*                    DESIRED OFFSET:
*                       IF &OFFTYPE  IS  'WORD', 4K - 4  IS COMPUTED.
*                       IF &OFFTYPE  IS  'BYTE', 4K - 1  IS COMPUTED.
*       &REG      --- REGISTER TO BE LOADED WITH THE COMPUTED OFFSET
*       &STORLOC  --- OPTIONAL D(X,B) WORD ADDRESS.  IF PRESENT, THE
*                     COMPUTED OFFSET IS STORED AT THIS LOCATION.
*
*   SAMPLE CALL:  BYTEOFF  WORD,R2,WORDOFFS
*********************************************************************
*
         MACRO
&LABEL   BYTEOFF &OFFTYPE,&REG,&STORLOC
****** COMPUTE OFFSET OF LOW WORD OF A K-WORD INTEGER.
&LABEL   L        &REG,0(R1)        LOAD @  OF  # OF FULLWORDS.
         L        &REG,0(&REG)      LOAD # OF FULLWORDS IN INTEGER.
         BCTR     &REG,0            SUBTRACT 1 AND
         SLL      &REG,2               MULTIPLY BY 4 TO GET OFFSET
*                                      OF LOW WORD.
.*
.***** IF BYTE OFFSET OF LAST BYTE IS DESIRED, ADD 3 TO WORD OFFSET.
.***** STORE OFFSET AT DESIGNATED ADDRESS (IF ANY).
         AIF    ('&OFFTYPE' EQ   WORD').WORD  IF WORD OFFSET, QUIT.
         LA       &REG,3(&REG)      ADD 3 TO GET OFFSET OF LAST
*                                      BYTE OF INTEGER.
.WORD    AIF      (T'&STORLOC EQ 'O').EXIT   NO STORAGE REQUESTED?
         ST       &REG,&STORLOC     STORE OFFSET AT D(X,B) ADDRESS.
.EXIT    MEND
         SPACE    3
*********************************************************************
*              ------    P U T O U T    ---
*
```

```
*         THIS MACRO PERFORMS "MOVE MODE" QSAM OUTPUT BY CONSTRUCTING
*    FIXED LENGTH (133 BYTE) OUTPUT RECORDS, THE FIRST CHARACTER OF
*    WHICH IS USED AS A CARRIAGE CONTROL.  PRINT REQUESTS OF LESS THAN
*    133 BYTES ARE PADDED WITH BLANKS.  REQUESTS OF MORE THAN 133
*    BYTES ARE TRUNCATED.  ANY CSECT THAT CALLS THE "PUTOUT" MACRO
*    MUST ESTABLISH ADDRESSABILITY TO THE "PRINTOUT" DCB AND TO
*    "RECORD", WHICH ARE STORED IN "OPNOUT" CSECT.
*
*    PARAMETERS.
*        &LOCATN -- ANY D(X,B) ADDRESS FROM WHICH EBCDIC CHARACTERS ARE
*                   TO BE PRINTED.  THE FIRST BYTE IS USED AS A CAR-
*                   RIAGE CONTROL.
*        &BYTES  -- # OF BYTES TO BE PRINTED (1 TO 133).  THIS VALUE
*                   INCLUDES THE CARRIAGE CONTROL BYTE.
*        IF THE MACRO IS CALLED WITH PARAMETER "&LOCATN" = "SKIP",
*    PARAMETER "&BYTES" DESIGNATES A NUMBER OF BLANK LINES TO BE
*    PRINTED.
*
*    SAMPLE CALLS:  PUTOUT  4(R10),18   -- PRINT 18 BYTES OF CHARACTERS
*                   PUTOUT  =C'ODENOMINATOR IS : ',18 -- PRINT 18 BYTES
*                   PUTOUT  SKIP,4                -- PRINT 4 BLANK LINES
*************************************************************************
*
         MACRO
&LABEL   PUTOUT   &LOCATN,&BYTES
*
****** SAVE REGISTERS AND INITIALIZE OUTPUT RECORD TO BLANKS.
&LABEL   STM      R14,R2,SAV&SYSNDX SAVE TRASHED REGISTERS.
         B        BYND&SYSNDX       BRANCH AROUND STORAGE.
SAV&SYSNDX DS     5F                SAVE AREA FOR REGISTERS R14-->R2
BYND&SYSNDX MVI   RECORD,C' '       MOVE BLANK INTO CARRIAGE CONTROL.
         MVC      RECORD+1(132),RECORD AND INITIALIZE RECORD BY PROPA-
*                                      GATING BLANKS THROUGHOUT.
*
****** MOVE OUTPUT CHARACTERS TO RECORD OR SET UP TO PRINT REQUESTED
****** NUMBER OF BLANK LINES.
         AIF      ('&LOCATN' EQ 'SKIP').SKIPLIN A SKIP LINE REQUEST?
         LA       R1,&LOCATN        LOAD @ OF CHARS. TO PRINT.
         LA       R2,&BYTES         LOAD # BYTES TO PRINT.
         BCTR     R2,R0             GET LENGTH CODE (BYTES-1) FOR MVC.
         EX       R2,MOV&SYSNDX     MOVE PRINT CHARACTERS TO OUTPUT
*                                      RECORD.
         B        LOAD&SYSNDX       BRANCH AROUND MVC.
MOV&SYSNDX MVC    RECORD(*-*),0(R1) MVC COPY FOR EXECUTE INSTRUCTION
LOAD&SYSNDX LA    R2,1              LOAD COUNTER FOR OUTPUT LOOP.
         AGO      .OUT
*
.SKIPLIN LA       R2,&BYTES         LOAD # BLANK LINES -- LOOP COUNT.
.OUT     ANOP
*
****** THIS LOOP PRINTS THE REQUESTED NUMBER OF BLANK LINES FROM A
****** "PUTOUT  SKIP" REQUEST.  FOR ANY OTHER PUTOUT REQUEST, THE
****** LOOP EXECUTES ONCE AND PRINTS A SINGLE LINE OF THE REQUESTED
```

```
****** EBCDIC CHARACTERS.  THE PUT MACRO TRASHES R14, R15, R0, R1.
LOOP&SYSNDX PUT   PRINTOUT,RECORD   PRINT NEXT BLANK LINE/RECORD.
         BCT      R2,LOOP&SYSNDX    LOOP UNTIL DONE.
         LM       R14,R2,SAV&SYSNDX RESTORE TRASHED REGISTERS.
*
         MEND
         SPACE    3
**********          SYMBOL    EQUATES            **************
R0       EQU      0
R1       EQU      1
R2       EQU      2
R3       EQU      3
R4       EQU      4
R5       EQU      5
R6       EQU      6
R7       EQU      7
R8       EQU      8
R9       EQU      9
R10      EQU      10
R11      EQU      11
R12      EQU      12
R13      EQU      13
R14      EQU      14
R15      EQU      15
NOCARRY  EQU      8+4               NO CARRY FOR LOGICAL ADD
         SPACE    3
```

APPENDIX B

MULTIPLE-PRECISION INTEGER ARITHMETIC ROUTINES (ASSEMBLER)

```
         PRINT    OFF
         TITLE    '------  SUPPORT ROUTINES.  MULTIPLE PRECISION INTEGERX
                  ARITHMETIC  ------'
*********************************************************************
*                 -------   A D D I T I O N   ------
*
*        THIS SUBROUTINE PERFORMS THE MULTIPLE PRECISION ADDITION
*    A + B OF SIGNED, TWO'S COMPLEMENT INTEGERS.  AN OVERFLOW FLAG
*    IS SET IF THE SUM CANNOT BE CONTAINED IN K-WORDS.
*
*    UPON ENTRY, R1 POINTS TO AN ARGUMENT LIST CONSISTING OF ...
*        R1 --->  @ K              --- # FULLWORDS TO HOLD OPERANDS
*                 @ A  (K WORDS)   --- AUGEND INTEGER
*                 @ B  (K WORDS)   --- ADDEND INTEGER
*                 @ A+B  (K WORDS) --- CAN BE OPTIONALLY PLACED IN
*                                      EITHER A OR B
*                 @ OVERFLOW ERROR FLAG (1 => ERROR, 0 => NO OVERFLOW)
*********************************************************************
         SPACE
ADDITION LOENTRY
****** LOAD OPERAND ADDRESSES AND SET UP FOR ADDITION LOOP.
         BYTEOFF  WORD,R2,          GET BYTE  OFFSET OF LAST  WORD IN
*                                   OPERANDS FOR INDEX REG. (4K - 4).
         LM       R7,R10,4(R1)      LOAD @ OF OPERAND A INTO R7;
*                                         @ OF OPERAND B INTO R8;
*                                         @ OF SUM A+B INTO R9;
*                                         @ OF ERROR FLAG INTO R10.
         L        R3,=F'-4'         LOAD DECREMENT FOR INDEX REG R2.
         SR       R4,R4             ZERO INITIAL CARRY BIT.
         L        R5,0(R7)          LOAD FIRST WORD OF OPERAND A
         ST       R5,ASAVE             AND SAVE FOR OVERFLOW CHECK.
         SPACE    2
****** THIS LOOP ADDS TWO MULTIPLE PRECISION, SIGNED INTEGERS, ONE
****** WORD AT A TIME, AND PROPAGATES ANY CARRY.
ADDLOOP  L        R5,0(R2,R7)       LOAD NEXT WORD OF 1ST OPERAND.
         L        R6,0(R2,R8)       LOAD NEXT WORD OF 2ND OPERAND.
         ALR      R5,R4             ADD OLD CARRY, IF ANY.
         IC       R4,=X'00'         ZAP OLD CARRY.
         BC       NOCARRY,*+8       BRANCH IF NO NEW CARRY.
         LA       R4,1              SET CARRY IF THERE WAS.
         ALR      R5,R6             ADD SECOND OPERAND.
         BC       NOCARRY,*+8       IF NEW CARRY, LEAVE IT SET.
         LA       R4,1              SET CARRY FOR NEXT ADD.
         ST       R5,0(R2,R9)       STORE SUM WORD.
         BXH      R2,R3,ADDLOOP     CONTINUE LOOP UNTIL ALL FULLWORDS
*                                      ARE ADDED.
         SPACE    2
```

```
****** DETERMINE IF OVERFLOW OCCURRED, I. E., WHETHER THE OPERANDS
****** ARE OF THE SAME SIGN, WHICH DIFFERS FROM THE SIGN OF THE
****** RESULT.
          SR        R2,R2               ZERO ERROR FLAG.
          X         R6,ASAVE            EXCL. OR-- 1ST WORDS OF OPERANDS
          LTR       R6,R6               OPERANDS OF OPPOSITE SIGN?
          BM        STOREFLG            IF SO, THERE IS NO OVERFLOW.
          X         R5,ASAVE            EXCLUSIVE OR -- 1ST WORDS OF
*                                           1ST OPERAND AND SUM
          LTR       R5,R5               IS SIGN OF RESULT DIFFERENT
*                                           FROM THAT OF THE OPERANDS?
          BNM       STOREFLG            IF SO, WE HAVE OVERFLOW.
          LA        R2,1                TURN ON OVERFLOW FLAG.
STOREFLG  ST        R2,0(R10)           STORE ERROR FLAG.
          LORETURN
ASAVE     DS        F                   FIRST WORD OF OPERAND A
          SPACE     3
*****************************************************************************
*                    - - - -   W O R D A D D   - - - -
*
*         THIS SUBROUTINE ADDS (AT A SPECIFIED OFFSET) A ONE-WORD
*    OPERAND "B" TO A POSITIVE MULTIPLE PRECISION OPERAND "A".
*
*    UPON ENTRY, R1 POINTS TO AN ARGUMENT LIST CONSISTING OF ...
*        R1 --> @K              -- # FULLWORDS IN MULTIPLE PRECISION SUM
*               @ OFFSET (1 WORD) IN OPERAND A AT WHICH TO ADD WORD
*               @ A (K WORDS) -- MULTIPLE PRECISION SUM
*               @ B (1 WORD)  -- WORD TO BE ADDED
*               @ OVERFLOW ERROR FLAG (SET TO 1 IF OVERFLOW)
*****************************************************************************
          SPACE
WORDADD   LOENTRY
****** LOAD ADDRESSES OF OPERANDS AND SET UP FOR ADD.
          LM        R6,R9,4(R1)         LOAD @ OF OFFSET AT WHICH TO ADD
*                                           INTO R6;
*                                         @ OPERAND A INTO R7;
*                                         @ WORD TO ADD INTO R8;
*                                         @ OVERFLOW ERROR FLAG INTO R9.
          SR        R10,R10             ZERO OVERFLOW ERROR FLAG.
          L         R2,0(R6)            LOAD OFFSET FOR ONE WORD ADD.
          L         R3,=F'-4'           LOAD BYTE DECREMENT FOR ADD. LOOP.
          LA        R4,1                LOAD BIT FOR PROPAGATION OF CARRY.
          SPACE
****** ADD WORD B TO K-WORD INTEGER A.
          L         R5,0(R2,R7)         LOAD WORD OF SUM.
          AL        R5,0(R8)            ADD ONE-WORD TO SUM.
          ST        R5,0(R2,R7)         STORE SUM WORD.
          BC        NOCARRY,SGNTEST     IF NO CARRY, SUMMATION IS COMPLETE.
          AR        R2,R3               SET OFFSET TO NEXT HIGH WORD.
          BM        OVERLOAD            GO LOAD ERROR FLAG -- OPERAND A
*                                           IS ENTIRELY PROCESSED.
          SPACE
****** THIS LOOP PROPAGATES A CARRY FROM THE ADDITION.
```

```
ADDLP    L        R5,0(R2,R7)        LOAD NEXT WORD OF K-WORD SUM.
         ALR      R5,R4              ADD CARRY.
         ST       R5,0(R2,R7)        STORE SUM WORD.
         BC       NOCARRY,SGNTEST    IF NO CARRY, WE ARE DONE.
         BXH      R2,R3,ADDLP        CONTINUE LOOP UNTIL PROPAGATION
*                                      OF CARRY HAS ENDED.
         SPACE
****** DETERMINE IF OVERFLOW OCCURRED -- CARRY FROM LAST ADD
****** OR NEGATIVE RESULT.  SET THE ERROR FLAG ACCORDINGLY.
         B        OVERLOAD           GO LOAD ERROR FLAG -- CARRY
*                                      ON LAST ADD.
SGNTEST  TM       0(R7),X'80'        TEST SIGN OF RESULT.
         BNO      STORFLG            IF NON-NEGATIVE, STORE FLAG.
OVERLOAD LA       R10,1              SET ERROR FLAG.
STORFLG  ST       R10,0(R9)          STORE ERROR FLAG FOR CALLER.
         SPACE
         LORETURN
         SPACE    3
*********************************************************************
*                  -----    S U B T R A C T    -------
*
*         THIS SUBROUTINE PERFORMS THE MULTIPLE PRECISION SUBTRACTION
*    A - B OF TWO SIGNED (2'S COMPLEMENT) INTEGERS BY ADDING THE
*    2'S COMPLEMENT OF B TO A.
*
*    UPON ENTRY, R1 POINTS TO AN ARGUMENT LIST CONSISTING OF ...
*        R1  ---> @ K              -- # FULLWORDS IN OPERANDS
*                 @ A  (K WORDS)   -- MINUEND INTEGER
*                 @ B  (K WORDS)   -- SUBTRAHEND INTEGER
*                 @ A-B  (K WORDS) -- CAN OPTIONALLY BE PLACED IN
*                                     EITHER A OR B
*                 @ OVERFLOW ERROR FLAG -- SET TO 1 IF B IS MAXIMUM
*                                          NEGATIVE INTEGER
*                                       -- SET TO 2 IF A-(-B) RESULTS
*                                          IN OVERFLOW
*
*    SUBROUTINES CALLED:
*        TWOSCOMP -- COMPUTES 2'S COMPLEMENT OF A K-WORD INTEGER
*        ADDITION -- ADDS TWO K-WORD INTEGERS (RESULT IN K WORDS)
*********************************************************************
         SPACE
SUBTRACT HIENTRY  R11,SUBTRSAV
         LM       R2,R6,0(R1)        LOAD @ K INTO R2;
*                                         @ A INTO R3;
*                                         @ B INTO R4;
*                                         @ A-B INTO R5;
*                                         @ ERROR FLAG INTO R6.
         SPACE
***** COMPLEMENT B AND PUT RESULT IN BBBCMP.  SET ERROR FLAG
***** IF B IS MAXIMUM NEGATIVE INTEGER.
         CALL     TWOSCOMP,,(R2,R4,BBBCMP,R6)
         ERRORCHK R6,GETOUT,         CHECK FOR ATTEMPT TO COMPLEMENT
*                                      MAXIMUM NEGATIVE INTEGER.
```

```
***** ADD 2'S COMPLEMENT OF B TO A AND STORE IN A-B.  SET ERROR FLAG
***** UPON ADDITION OVERFLOW.
         CALL     ADDITION,,(R2,R3,BBBCMP,R5,R6)
         ERRORCHK R6,GETOUT,2        CHECK FOR ADDITION OVERFLOW AND
*                                       AND RE-SET ERROR FLAG IF
*                                       OVERFLOW IS PRESENT.
GETOUT   HIRETURN
BBBCMP   MULTPREC 2                  2'S COMPLEMENT OF SUBTRAHEND B
         SPACE    3
*********************************************************************
*                  -----  M U L T I P L Y  -----
*
*       THIS PROCEDURE PERFORMS THE MULTIPLE PRECISION MULTIPLICA-
*    TION A * B OF TWO SIGNED, TWO'S COMPLEMENT INTEGERS.  IF EITHER
*    OPERAND IS THE MAXIMUM NEGATIVE INTEGER THAT CAN BE CONTAINED
*    IN K WORDS, NO PRODUCT IS COMPUTED.  INSTEAD, AN ERROR FLAG IS
*    SET.
*       THE ALGORITHM EMPLOYED MAKES USE OF THE MACHINE MULTIPLY
*    INSTRUCTION.  NEGATIVE OPERANDS ARE COMPLEMENTED TO OBTAIN
*    POSITIVE OPERANDS FOR MULTIPLICATION.  EACH NON-ZERO HALFWORD
*    OF THE MULTIPLICAND IS THEN MULTIPLIED BY EACH NON-ZERO HALFWORD
*    OF THE MULTIPLIER, AND EACH OF THESE PARTIAL PRODUCTS IS
*    ACCUMULATED IN A RUNNING PRODUCT TOTAL.  USE OF HALFWORDS
*    ENSURES EASY LOADING FROM THE K-WORD OPERANDS AND MAKES POSSI-
*    BLE AN UNSIGNED MACHINE MULTIPLY WITHOUT POSSIBILITY OF OVERFLOW.
*
*    UPON ENTRY, R1 POINTS TO AN ARGUMENT LIST CONSISTING OF ...
*        R1 --> @ K                  --- # FULL WORDS TO HOLD OPERANDS
*               @ A  (K WORDS)       --- MULTIPLICAND INTEGER
*               @ B  (K WORDS)       --- MULTIPLIER INTEGER
*               @ A*B  (2K WORDS)    --- PRODUCT
*               @ ERROR FLAG --- SET TO 1 IF A OR B IS MAX. NEG. INTEGER
*                                CONTAINED IN K WORDS
*
*    SUBROUTINES CALLED:
*        TWOSCOMP -  COMPUTES 2'S COMPLEMENT OF A K-WORD INTEGER
*        WORDADD  --- ADDS AT A SPECIFIED OFFSET A WORD TO A POSITIVE
*                     K-WORD INTEGER
*********************************************************************
         SPACE
MULTIPLY HIENTRY  R11,MULTSAVE
****** LOAD ARGUMENT ADDRESSES AND COMPUTE USEFUL CONSTANTS.
         ST       R1,R1SAVE          SAVE @ OF INPUT ARGUMENT LIST.
         LM       R6,R10,0(R1)       LOAD @ OF # WORDS IN A,B INTO R6;
*                                          @ OF A(MULTIPLICAND) INTO R7;
*                                          @ OF B (MULTIPLIER) INTO R8;
*                                          @ OF PRODUCT A*B INTO R9;
*                                          @ OF ERROR FLAG INTO R10.
         L        R2,0(R6)           LOAD # FULLWORDS IN OPERANDS.
         AR       R2,R2              MULTIPLY BY 2 FOR # HALFWORDS.
         ST       R2,TWOXK           STORE TWICE THE PRECISION (2*K).
         BCTR     R2,0               SUBTRACT 1,
         AR       R2,R2                 DOUBLE FOR OFFSET IN BYTES OF
```

```
*                                          LEAST SIGNIF. HALFWORD OF B
*                                          (4K-2).
          SPACE
****** DETERMINE SIGN OF PRODUCT AND STORE.  COMPLEMENT ANY NEGATIVE
****** OPERANDS.
          SR        R5,R5            ZERO PRODUCT SIGN.
          TM        0(R7),X'80'      CHECK SIGN OF OPERAND A.
          BNO       SCNDOPND         IF OPERAND POS. OR 0, GO CHECK 2ND.
          LA        R5,1             SET SIGN FLAG TO NEGATIVE.
***  GET 2'S COMPLEMENT OF OPERAND A.
          CALL      TWOSCOMP,,(R6,R7,ACMPL,R10)
          ERRORCHK  R10,EXIT         CHECK ERROR FLAG; IF SET, RETURN.
          LA        R7,ACMPL         1ST OPERAND WILL BE 2'S COMPL. OF A
SCNDOPND  TM        0(R8),X'80'      CHECK SIGN OF OPERAND B.
          BNO       STORSIGN         IF OPERAND IS POS. OR 0, USE
*                                        ORIGINAL OPERAND.
          X         R5,=F'1'         BOTH OPERANDS NEG. => PROD. = POS.
***  GET 2'S COMPLEMENT OF  OPERAND B.
          CALL      TWOSCOMP,,(R6,R8,BCMPL,R10)
          ERRORCHK  R10,EXIT         CHECK ERROR FLAG;  IF SET, RETURN.
          LA        R8,BCMPL         2ND OPERAND WILL BE 2'S COMPL. OF B
STORSIGN  ST        R5,PRODSIGN      STORE SIGN OF PRODUCT A * B.
          SPACE
****** SET UP ARGUMENT TABLE FOR CALLS TO WORDADD AND LOAD REGISTERS
****** WITH VALUES NEEDED IN THE MULTIPLY LOOPS.  SUM OF PARTIAL
****** PRODUCTS WILL BE ACCUMULATED IN FINAL PRODUCT.
          LDARGTBL  ADDTABL,(TWOXK,PRODOFFS,R9,PRODWORD,R10)
          LA        R1,ADDTABL       LOAD @ OF ARGUMENT TABLE FOR
*                                        CALL TO "WORDADD."
          L         R15,=V(WORDADD)  GET @ OF "WORDADD" INTO R15.
          LA        R10,3(R2,R2)     GET LENGTH (8K-1) FOR ZERO MACRO.
          ZEROLONG  R10,0(R9)        ZERO ACCUMULATED PROD. (2K WORDS).
          L         R10,=X'0000FFFF' LOAD "AND" VALUE INTO R10.
          L         R9,=F'-2'        LOAD BYTE DECREMENT FOR LOOPS.
          LR        R6,R2            COPY BYTE OFFSET OF LOW HALFWORD
*                                        OF A,B (4K-2).
          SPACE     3
****** MULTIPLY EACH NON-ZERO HALFWORD IN OPERAND A BY EACH NON-ZERO
****** HALFWORD IN OPERAND B, ACCUMULATING PARTIAL PRODUCTS AS WE GO.
*------ R0 CONTAINS CURRENT HALFWORD OF MULTIPLIER.
*------ R1 POINTS TO THE ARGUMENT LIST ("ADDTABL") FOR THE CALL
*------    TO "WORDADD."
*------ R2 IS INDEX REGISTER FOR MULTIPLIER LOOP.
*------ R3 IS INDEX REGISTER FOR MULTIPLICAND LOOP.
*------ R6 CONTAINS BYTE OFFSET OF LOW HALFWORD OF A,B.
*------ R7 CONTAINS ADDRESS OF MULTIPLICAND.
*------ R8 CONTAINS ADDRESS OF MULTIPLIER.
*------ R9 CONTAINS BYTE DECREMENT (-2) FOR LOOPS.
*------ R10 CONTAINS X'0000FFFF'.
*------ R15 CONTAINS THE ADDRESS OF "WORDADD."
          SPACE
OUTLOOP   LH        R0,0(R2,R8)      LOAD NEXT HALFWORD OF MULTIPLIER.
          NR        R0,R10           ZERO LEFT HALFWORD OF MULTIPLIER.
```

```
         BZ        OUTFOOT             IF RESULT = 0, DO NEXT HALFWORD.
         LR        R3,R6               COPY BYTE OFFSET OF LOW ORDER
*                                        HALFWORD OF A.
****** MULTIPLY EACH NON-ZERO HALFWORD OF THE MULTIPLICAND BY THE
****** CURRENT HALFWORD OF THE MULTIPLIER.
INLOOP   LH        R5,0(R3,R7)         LOAD NEXT HALFWORD OF MULTIPLICAND.
         NR        R5,R10              ZERO LEFT HALFWORD OF MULTIPLICAND.
         BZ        INFOOT              IF RESULT = 0, DO NEXT HALFWORD.
         MR        R4,R0               PUT PRODUCT OF HALFWORDS INTO R5.
         LA        R4,0(R2,R3)         ADD BYTE OFFSETS OF HALFWORD
*                                        MULTIPLIER & MULTIPLICAND TO GET
*                                        BYTE OFFSET IN PARTIAL SUM AT
*                                        WHICH TO ADD PARTIAL PRODUCT.
         ST        R4,PRODOFFS         STORE TENTATIVE OFFSET IN PRODUCT
*                                        OF PRODUCT WORD.
         SPACE
****** TEST BOUNDARY ALIGNMENT OF PRODUCT WORD.  IF PRODUCT WORD
****** IS HALFWORD (NOT FULLWORD) ALIGNED WITH RESPECT TO THE
****** 2-K PRODUCT, WORDADD MUST BE CALLED TWICE TO ACCUMULATE
****** EACH HALFWORD TO THE RUNNING PRODUCT.
         TM        PRODOFFS+3,X'02'    TEST FOR HALFWORD BOUNDARY.
         BO        HLFWORD             IF SO, PROCESS HALFWORD AT A TIME.
         ST        R5,PRODWORD         STORE ENTIRE PARTIAL PRODUCT
         B         LASTBALR               AND GO ACCUMULATE IT TO TOTAL.
*** PRODUCT WORD IS HALFWORD ALIGNED WITH RESPECT TO THE 2-K PRODUCT.
HLFWORD  SR        R14,R14             ZERO R14
         ST        R14,PRODWORD           AND PRODUCT WORD.
         STH       R5,PRODWORD         STORE RIGHTHAND HALFWORD,
         LA        R4,2(R4)               INCREMENT OFFSET TO FULLWORD,
         ST        R4,PRODOFFS            STORE FULLWORD OFFSET,
*** ACCUMULATE LOW HALFWORD TO PRODUCT TOTAL.
         BALR      R14,R15                AND CALL WORDADD TO ACCUMULATE
*                                        LOW HALFWORD TO PRODUCT TOTAL.
         SRL       R5,16               SHOVE LEFT HALFWORD TO RIGHT
         ST        R5,PRODWORD            AND STORE IN PARTIAL PRODUCT.
         S         R4,=F'4'            SUBTRACT 4 FOR HIGH WORD BOUNDARY
         ST        R4,PRODOFFS            AND STORE FOR "WORDADD."
*** ACCUMULATE HIGH HALFWORD (OR ENTIRE WORD IF FULLWORD ALIGNED) OF
*** THE PARTIAL PRODUCT TO RUNNING TOTAL.
LASTBALR BALR      R14,R15             CALL "WORDADD" SUBROUTINE.
         SPACE
INFOOT   BXH       R3,R9,INLOOP        LOOP THRU ALL MULTIPLICAND HALF-
*                                        WORDS AT CURRENT MULTIPLIER H'.
OUTFOOT  BXH       R2,R9,OUTLOOP       LOOP THRU ALL MULTIPLIER HALFWDS.
         SPACE   3
****** TEST SIGN OF THE PRODUCT AND GET 2'S COMPLEMENT OF THE
****** TENTATIVE RESULT IF THE PRODUCT IS NEGATIVE.
         TM        PRODSIGN+3,B'00000001'   TEST FOR PRODUCT SIGN.
         BNO       EXIT                IF NON-NEGATIVE, NO ADJUSTMENT IS
*                                        NECESSARY.
         L         R1,R1SAVE           RESTORE R1 OF INPUT ARG. LIST.
         CALL      TWOSCOMP,(TWOXM,C12(R1),C12(R1),C16(R1))
         SPACE
```

```
EXIT     DIRETURN
         SPACE   2
*       ******   DATA STORAGE FOR MULTIPLY CSECT   ******
PRODSIGN DS      F                 SIGN OF PRODUCT:  1 -> NEGATIVE
TWOXK    DS      F                 TWO TIMES THE PRECISION K
PRODWORD DS      F                 PRODUCT OF 2 UNSIGNED HALFWORDS
PRODOFFS DS      F                 OFFSET AT WHICH TO ADD PARTIAL
*                                     PRODUCT TO TOTAL.
R1SAVE   DS      F                 SAVE WORD FOR POINTER TO INPUT
*                                     ARGUMENT LIST.
ACMPL    MULTPREC 2                TWO'S COMPLEMENT OF OPERAND A
BCMPL    MULTPREC 2                TWO'S COMPLEMENT OF OPERAND B
         SPACE   3
*********************************************************************
*
*                 - - -   D I V I S I O N   - - -
*
*
*        THIS PROCEDURE PERFORMS MULTIPLE PRECISION DIVISION OF TWO
*    2'S COMPLEMENT K-WORD INTEGERS.  THE INTEGER QUOTIENT AND REMAIN-
*    DER ARE COMPUTED.  THE DIVISION FAILS IF THE DIVISOR IS 0 OR IF
*    IF EITHER OPERAND IS THE MAXIMUM NEGATIVE INTEGER THAT CAN BE
*    CONTAINED IN K-WORDS.
*
*    ALGORITHM DESCRIPTION:
*        NEGATIVE OPERANDS ARE COMPLEMENTED, AND THE SIGNS OF THE QUO-
*    TIENT AND REMAINDER ARE DETERMINED.  THE QUOTIENT IS SET TO ZERO,
*    AND THE REMAINDER IS INITIALIZED TO THE VALUE OF THE DIVIDEND.
*    AN ITERATIVE TRIAL DIVISION IS EMPLOYED USING A 56-BIT TRIAL DIVI-
*    DEND AND A 31-BIT TRIAL DIVISOR FORMED FROM THE LEADING NON-ZERO
*    BITS OF THE CURRENT REMAINDER AND THE FULL DIVISOR.
*         THE TRIAL QUOTIENT FROM THE MACHINE DIVIDE INSTRUCTION (DR)
*    IS SHIFTED BY AN APPROPRIATE POWER OF TWO TO ENSURE A PRODUCT OF
*    THE SHIFTED TRIAL QUOTIENT AND THE FULL DIVISOR LESS THAN THE
*    CURRENT REMAINDER.  THIS PRODUCT IS SUBTRACTED FROM THE REMAINDER
*    AND ADDED TO THE CURRENT QUOTIENT.  THE PROCESS CONTINUES UNTIL
*    THE REMAINDER IS LESS THAN THE DIVISOR.  THE SIGNS OF THE FINAL
*    QUOTIENT AND REMAINDER ARE SET AS NECESSARY BY TAKING THE 2'S
*    COMPLEMENT OF THE OPERAND.
*
*    UPON ENTRY, R1 POINTS TO AN ARGUMENT LIST CONSISTING OF ...
*        R1 --> C K  --- # FULLWORDS IN DIVISOR, DIVIDEND
*                    --- IN CONSTRUCTING TRIAL DIVIDENDS, K IS ASSUMED
*                        TO BE 2 OR GREATER.
*               C DIVIDEND  (K WORDS)
*               C DIVISOR   (K WORDS)
*               C QUOTIENT  (K WORDS)  --- CAN BE PLACED IN DIVIDEND
*                                          IF DESIRED (NOT IN DIVISOR)
*               C REMAINDER (K WORDS)  --- CAN BE PLACED IN DIVIDEND
*                                          IF DESIRED (NOT IN DIVISOR)
*               C OVERFLOW ERROR FLAG  --- 0 IF NO OVERFLOW ERROR
*                                      --- 1 IF DIVISION BY 0
*                                      --- 2 IF DIVIDEND IS MAXIMUM NEG.
*                                          INTEGER IN K WORDS
*                                      --- 3 IF DIVISOR IS MAX. NEG. INT.
```

```
*
*    SUBROUTINES CALLED:
*         TWOSCOMP -- COMPUTES 2'S COMPLEMENT OF A K-WORD INTEGER
*         LEFTZERO -- DETERMINES # OF CONSECUTIVE LEADING BINARY
*                     0'S IN A K-WORD INTEGER
*         POWRMULT -- MULTIPLIES (DIVIDES) A K-WORD INTEGER BY A
*                     SPECIFIED POWER OF 2 (RESULT IN K WORDS)
*         MULTIPLY -- MULTIPLIES TWO K-WORD INTEGERS (RESULT IN
*                     2K WORDS)
*         SUBTRACT -- COMPUTES THE DIFFERENCE OF TWO 2'S COMPLEMENT
*                     INTEGERS (RESULT IN K WORDS)
*         ADDITION -- ADDS TWO K-WORD INTEGERS (RESULT IN K WORDS)
*********************************************************************
          SPACE
DIVISION HIENTRY  R11,DIVSAVE
****** LOAD OPERAND ADDRESSES FROM ARGUMENT LIST AND COMPUTE
****** USEFUL CONSTANTS.
          LM        R5,R10,0(R1)       LOAD @ # WORDS IN OPNDS. INTO R5;
*                                              @ DIVIDEND INTO R6;
*                                              @ DIVISOR  INTO R7;
*                                              @ QUOTIENT INTO R8;
*                                              @ REMAINDER "   R9;
*                                              @ OVERFLOW FLAG INTO R10.
          BYTEOFF   WORD,R2,           GET BYTE OFFSET (4K-4) OF LOW
*                                        WORD IN A K-WORD INTEGER.
          LA        R3,3(R2)           GET BYTE OFFSET (4K-1) OF LOW
*                                        BYTE OF OPERANDS.
          ST        R1,SAVER1          SAVE POINTER TO ARGUMENT LIST.
          SPACE     2
********** DETERMINE SIGN OF QUOTIENT AND REMAINDER.  COMPLEMENT
********** NEGATIVE DIVISORS AND DIVIDENDS FOR USE IN POSITIVE
********** OPERAND ALGORITHM.  IF DIVIDEND OR DIVISOR IS MAXIMUM
********** NEGATIVE INTEGER, SET ERROR FLAG AND RETURN TO CALLER.
          MVI       QTRMSIGN,X'00'     ZERO QUOTIENT AND REMAINDER SIGNS.
          TM        0(R6),B'10000000'  TEST SIGN OF DIVIDEND.
          BNO       DIVSRCHK           IF NON-NEGATIVE, CHECK DIVISOR.
          MVI      QTRMSIGN,X'FF'      SET QUOT. & REMAIND. FLAGS TO NEG.
***   GET 2'S COMPLEMENT OF DIVIDEND.
          CALL      TWOSCOMP,,(R5,R6,DVDNDCMP,R10) COMPLEMENT DIVIDEND.
          ERRORCHK  R10,DPART,2        IF DIVIDEND WAS MAX. NEG. #, RE-SET
*                                        ERROR FLAG TO 2 AND QUIT.
          L         R6,=A(DVDNDCMP)    USE 2'S COMPLEMENT OF DIVIDEND IN
*                                       DIVISION--ENSURE ADDRESSA-
*                                       BILITY.
DIVSRCHK  TM        0(R7),B'10000000'  TEST SIGN OF DIVISOR.
          BNO       INITREM            IF NON-NEGATIVE, INITIALIZE REM.
          XI        QTRMSIGN,X'F0'     IF BOTH OPERANDS ARE NEGATIVE,
*                                        QUOTIENT IS NON-NEGATIVE.
***   GET 2'S COMPLEMENT  OF  DIVISOR.
          CALL      TWOSCOMP,,(R5,R7,DIVSRCMP,R10) COMPLEMENT DIVISOR.
          ERRORCHK  R10,DPART,3        IF DIVISOR WAS MAX. NEG. #, RE-SET
*                                        ERROR FLAG TO 3 AND QUIT.
          L         R7,=A(DIVSRCMP)    USE 2'S COMPLEMENT OF DIVISOR IN
```

```
*                                       DIVISION-- ENSURE ADDRESSA-
*                                       BILITY.
         SPACE
****** INITIALIZE REMAINDER TO DIVIDEND AND ZERO QUOTIENT.
INITREM  CR       R6,R9                 IS REMAINDER TO BE PLACED AT
*                                          DIVIDEND ADDRESS?
         BE       ZEROQT                IF SO, BRANCH AROUND MOVE.
         MOVELONG R3,0(R9),0(R6)        INITIALIZE REMAINDER TO DIVIDEND.
ZEROQT   ZEROLONG R3,0(R8)              ZERO QUOTIENT.
         SPACE
************ LOAD ARGUMENT TABLES FOR SUBROUTINE CALLS WITHIN DIVISION
************ LOOP TO POWMULT, MULTIPLY, SUBTRACT, ADDITION, AND
************ LEFTZERO.
         LDARGTBL POWRTABL,(R5,SHFTFACT,TRIALQT,TRIALQT)
         LDARGTBL MLTTABLE,(R5,R7,TRIALQT,TRIALPRD,R10)
         LA       R15,TRIALPRD+1(R3) LOAD 0 OF LOW K WORDS OF TRIALPRD
         LDARGTBL SUBTABLE,(R5,R9,R15,R9,R10)
         LDARGTBL ADDTABLE,(R5,R8,TRIALQT,R8,R10)
         LDARGTBL LEFTTBL2,(R5,R9,RENZEROS)
         SPACE
****** DETERMINE NUMBER OF LEFT ZEROS IN DIVISOR AND CHECK FOR
****** DIVISION BY 0.
         CALL     LEFTZERO,LEFTTABL,(R5,R7,DIVZEROS)
         SR       R0,R0                 ZERO OVERFLOW FLAG.
         ST       R0,0(R10)             STORE OVERFLOW FLAG INITIALIZATION.
         LA       R6,1(R3)              GET # OF BYTES IN OPERANDS.
         SLL      R6,3                  MULTIPLY BY 8 FOR # BITS IN K WORDS
         ST       R6,TOTBITS            STORE TOTAL BITS IN K WORDS.
         L        R5,DIVZEROS           LOAD LEFT ZERO BITS OF DIVISOR.
         LR       R8,R6                 COPY TOTAL BITS IN R8.
         SR       R8,R5                 GET # NON-ZERO DIVISOR BITS IN R8.
         BZ       OVERFLO               IF ZERO, WE HAVE DIVISION BY 0.
         SPACE
****** DETERMINE SIZES OF TRIAL DIVISOR AND TRIAL DIVIDEND.  IF THERE
****** ARE > 31 NON-ZERO BITS IN THE FULL DIVISOR, THE LEFT 31 WILL
****** BE USED AS A TRIAL DIVISOR.  OTHERWISE, THE RIGHT WORD OF THE
****** DIVISOR BECOMES THE TRIAL DIVISOR.  IF THERE ARE >= 26 NON-
****** ZERO BITS IN THE DIVISOR, THE TRIAL DIVIDEND WILL BE 56 BITS.
****** IF NOT, THE LENGTH OF THE TRIAL DIVIDEND IS DECREASED CORRES-
****** PONDINGLY TO PREVENT A FIXED-POINT DIVIDE ERROR.
         C        R8,=F'31'             COMPARE BITS IN DIVISOR TO 31.
         BH       TRIALDV               IF > 31, USE LEFT 31 BITS.
**** GET LOW 31 BITS OF DIVISOR FOR A TRIAL DIVISOR.
         L        R0,0(R7,R2)           LOAD LOW WORD OF DIVISOR.
         S        R6,=F'31'             GET # ZERO BITS LEFT OF TRIAL
*                                          DIVISOR.
         C        R8,=F'26'             COMPARE DIVISOR BITS TO 26.
         BNL      FULLDVND              IF >= 26, TRIAL DIVIDEND
*                                          WILL BE 56 BITS.
**** TRIAL DIVIDEND IS < 56 BITS TO PREVENT FIXED-POINT DIVIDE ERROR.
         LA       R8,30(R8)             GET # BITS IN TRIAL DIVIDEND FOR
*                                          OVERFLOW PREVENTION (A # < 56).
         B        LOOPSET
```

```
         SPACE
****** ESTABLISH AS TRIAL DIVISOR LEFT 31 NON-ZERO BITS OF COMPLETE
****** DIVISOR.
TRIALDV  SR       R4,R4              ZERO LEFT HALF OF DIVIDEND.
         ST       R4,DVWORDS-4       ZERO RIGHT HALF OF TRIAL DIVISOR
*                                      WORK AREA.
         LR       R6,R5              SAVE # ZERO BITS LEFT OF
*                                      TRIAL DIVISOR.
         D        R4,=F'8'           DIVIDE DIVISOR LEFT ZEROS BY 8:
*                                      BYTE OFFSET GOES IN R5.
         LA       R5,0(R5,R7)        GET ADDRESS FOR MVC AND MOVE
         MVC      DVWORDS(5),0(R5)      5 BYTES OF DIVISOR INTO DVWORDS.
         LM       R0,R1,DVWORDS      LOAD PORTION OF COMPLETE DIVISOR &
         SLDL     R0,0(R4)              SHIFT LEFT FOR TRIAL DIVISOR.
         SRL      R0,1               SHIFT RIGHT 1 BIT TO GET 31 BIT
*                                      TRIAL DIVISOR.
         SPACE
**** SET UP FOR LOOP.
FULLDVND LA       R8,56              TRIAL DIVIDEND WILL BE 56 BITS.
*
LOOPSET  L        R10,TOTBITS        LOAD TOTAL # BITS IN K WORDS.
         SR       R10,R8             SUBTRACT # DIVIDEND BITS.
         S        R2,=F'4'           GET 8 OF OPERANDS' LOW DOUBLEWORD.
         SPACE    3
********** THIS LOOP ADDS AN ADJUSTED TRIAL QUOTIENT TO A CURRENT
********** QUOTIENT AND SUBTRACTS THE PRODUCT OF THIS TRIAL QUOTIENT
********** AND THE DIVISOR FROM THE CURRENT REMAINDER UNTIL THE
********** REMAINDER IS LESS THAN THE DIVISOR.
*------ R0 CONTAINS THE ONE-WORD TRIAL DIVISOR.
*------ R1 IS UPDATED WITH # OF ZERO BITS LEFT OF TRIAL DIVIDEND.
*------ R2 CONTAINS (4K-8) OFFSET OF LOW DOUBLEWORD OF OPERANDS.
*------ R3 CONTAINS  (4K-1) OFFSET OF LOW BYTE OF OPERANDS.
*------ R4, R5 ARE UPDATED WITH A 56-BIT TRIAL DIVIDEND AND LATER
*------    R5 RECEIVES THE ONE-WORD TRIAL QUOTIENT.
*------ R6 CONTAINS # ZERO BITS LEFT OF TRIAL DIVISOR.
*------ R7 CONTAINS THE ADDRESS OF THE K-WORD DIVISOR.
*------ R8 CONTAINS THE # OF BITS IN THE TRIAL DIVIDEND (A # <= 56).
*------ R9 POINTS TO THE CURRENT K-WORD REMAINDER.
*------ R10 CONTAINS THE TOTAL # OF BITS IN K WORDS MINUS THE # OF BITS
*------    IN THE TRIAL DIVIDEND.
         SPACE
DIVLOOP  CLCLONG  R3,0(R9),0(R7)     IS REMAINDER < DIVISOR?
         BL       SIGNADJ            IF DONE, ADJUST SIGNS AND QUIT.
         CALL     LEFTZERO,LEFTTBL2, GET # LEADING ZEROS IN REMAINDER.
         SPACE
****** ESTABLISH AS TRIAL DIVIDEND LEFTMOST 56 NON-ZERO BITS OF
****** CURRENT REMAINDER.
         C        R10,REMZEROS       COMPARE TOTAL BITS IN K WORDS
*                                      MINUS TRIAL DIVIDEND BITS TO
*                                      REMAINDER LEFT ZEROS.
         BH       TRIALREM           IF GREATER, GET FULL 7 BYTES.
**** GET LOW <= 7 BYTES OF REMAINDER AS TRIAL DIVIDEND.
         LA       R14,0(R2,R9)       GET C OF REMAINDER'S LOW DOUBLEWD.
```

```
         LM        R4,R5,0(R14)      LOAD LOW DOUBLEWORD OF REMAINDER.
         LR        R1,R10            COPY (# BITS - DIVIDEND BITS).
         B         TRDIVSN
         SPACE
****** GET LEADING 7 NON-ZERO BYTES OF REMAINDER AS TRIAL QUOTIENT,
****** SHIFTING LEFT SO HIGH BIT OF 7 BYTES IS A "1".
TRIALREM SR        R14,R14           ZERO LEFT HALF OF DIVIDEND.
         L         R1,REMZEROS       LOAD # ZERO BITS LEFT OF
*                                      TRIAL DIVIDEND.
         LR        R15,R1            COPY # OF ZERO BITS LEFT OF TRIAL
*                                      DIVIDEND.
         D         R14,=F'8'         DIVIDE REMAINDER LEFT ZEROS BY 8:
*                                      BYTE OFFSET GOES INTO R15.
         LA        R5,0(R15,R9)      GET ADDRESS FOR MVC AND MOVE 8
         MVC       DVWORDS(8),0(R5)    BYTES OF REMAINDER INTO DVWORDS
         LM        R4,R5,DVWORDS     LOAD RAW TRIAL DIVIDEND.
         LA        R15,64            LOAD 64 INTO R15.
         LA        R14,0(R14,R8)     ADD TRIAL DIVIDEND BITS TO R14.
         SR        R15,R14           SUBTRACT FOR FINAL BIT SHIFT.
         SRDL      R4,0(R15)         SHIFT RIGHT FOR <= 7 BYTE
*                                      FINAL TRIAL DIVIDEND.
         SPACE
***** USE MACHINE "DIVIDE" TO GET A ONE-WORD TRIAL QUOTIENT FROM
***** A 56-BIT TRIAL DIVIDEND AND A 31-BIT TRIAL DIVISOR.  A POWER
***** OF 2 RELATIVE MAGNITUDE CORRECTION PRODUCES A K-WORD TRIAL
***** QUOTIENT WHICH IS THEN MULTIPLIED BY THE ENTIRE DIVISOR.  IF
***** THE RESULT IS LESS THAN THE CURRENT REMAINDER, THE RESULT IS
***** ADDED TO A CUMULATIVE QUOTIENT AND SUBTRACTED FROM THE
***** REMAINDER.
TRDIVSN  DR        R4,R6             DIVIDE FOR ONE WORD TRIAL QUOTIENT
         LA        R4,31(R6)         ADD 31 TO # BITS LEFT OF TRIAL
*                                      DIVISOR.
         LA        R1,0(R1,R8)       ADD # BITS LEFT OF TRIAL
*                                      DIVIDEND TO ACTUAL # BITS IN
*                                      TRIAL DIVIDEND.
         SR        R4,R1             SUBTRACT FOR # BITS TO SHIFT
*                                      FOR TRUE TRIAL QUOTIENT.
NEXTZERO ZEROLONG  R3,TRIALQT        ZERO K WORDS OF TRIAL QUOTIENT.
         ST        R5,TRIALQT+4(R2)  STORE QUOTIENT AT LOW WORD OF
*                                      K-WORD TRIAL QUOTIENT.
NEXTSHFT ST        R4,SHFTFACT       STORE BITS TO SHIFT.
         CALL      POWRMULT,POWRTABL, MULTIPLY/DIVIDE RAW TRIAL
*                                      QUOTIENT BY SHIFT FACTOR.
         CALL      MULTIPLY,MLTTABLE, GET PRODUCT OF DIVISOR &
*                                      ADJUSTED TRIAL QUOTIENT.
         SPACE
****** IF ADJUSTED TRIAL QUOTIENT IS LESS THAN THE CURRENT REMAINDER,
****** SUBTRACT IT AND ADD TO CURRENT QUOTIENT.  IF NOT, TRY AN
****** ADJUSTED TRIAL QUOTIENT THAT IS SMALLER BY A FACTOR OF 2.
         LA        R14,TRIALPRD+4(R3) GET 8 OF LOW K-WORDS OF TRIALPRD.
         CMPLONG   R3,0(R14),0(R9)   IS PRODUCT <= REMAINDER?
         BNH       SUBTRCT           IF SO, SUBTRACT FROM REMAINDER.
         L         R4,=F'-1'         LOAD SHIFT FOR DIVIDE BY 2 AND
```

```
         B         NEXTSHFT              TRY TRIAL QUOTIENT SMALLER BY A
*                                        FACTOR OF 2.
SUBTRCT  CALL      SUBTRACT,SUBTABLE,    SUBTRACT PRODUCT OF DIVISOR
*                                          AND ADJUSTED TRIAL QUOTIENT.
         CALL      ADDITION,ADDTABLE,    ADD ADJUSTED TRIAL QUOTIENT TO
*                                          CURRENT QUOTIENT.
         B         DIVLOOP             LOOP UNTIL REMAINDER < DIVISOR.
         SPACE     3
OVERFLO  LA        R0,1                SET OVERFLOW FLAG.
         ST        R0,0(R10)           STORE OVERFLOW FLAG
         B         DPART                  AND RETURN.
         SPACE
****** OBTAIN OUTPUT VALUES OF THE QUOTIENT AND REMAINDER BY COM-
****** PLEMENTING TENTATIVE VALUES IF NECESSARY.
SIGNADJ  TM        QTRNSIGN,X'F0'      TEST SIGN OF QUOTIENT.
         BNO       REMADJ              IF NON-NEGATIVE, CHECK REMAINDER.
***  COMPLEMENT QUOTIENT TO GET NEGATIVE VALUE.
         L         R1,SAVER1           RESTORE POINTER TO ORIG. ARG. LIST
         CALL      TWOSCOMP,,(@LEFTTABL,@12(R1),@12(R1),@20(R1))
REMADJ   TM        QTRNSIGN,X'0F'      TEST SIGN OF REMAINDER.
         BNO       DPART               IF NON-NEGATIVE, WE ARE DONE.
***  COMPLEMENT REMAINDER TO GET NEGATIVE VALUE.
         L         R1,SAVER1           RESTORE POINTER TO ORIG. ARG. LIST
         CALL      TWOSCOMP,,(@LEFTTABL,R9,R9,@20(R1))
         SPACE
DPART    HIRETURN
         SPACE     2
*        ******    DATA STORAGE FOR DIVISION CSECT    ******
SAVER1   DS        F                   SAVE WORD FOR ARG. LIST POINTER
QTRNSIGN DC        X'00'               FLAG BYTE:  LEFT DIGIT SET TO F
*                                        IF QUOTIENT IS NEGATIVE; SET
*                                        TO 0 OTHERWISE.  RIGHT DIGIT
*                                        SET TO F IF REMAINDER IS NEG.
DIVZEROS DS        F                   # LEADING ZEROS IN DIVISOR
REMZEROS DS        F                   # LEADING ZEROS IN REMAINDER
SHFTFACT DS        F                   # BITS TO SHIFT RAW TRIAL QUOTIENT
*                                        FOR ADJUSTED TRIAL QUOTIENT
DVWORDS  DS        2F                  DIVISOR/REMAINDER TO BE SHIFTED
*                                        LEFT FOR TRIAL DIVISOR/DIVIDEND
TOTBITS  DS        F                   TOTAL # BITS IN K WORDS
TRIALQT  MULTPREC  2                   CURRENT TRIAL QUOTIENT
TRIALPRD MULTPREC  4                   PRODUCT OF TRIAL QUOT., DIVISOR
DIVSRCMP MULTPREC  2                   2'S COMPLEMENT OF DIVISOR
DVDNDCMP MULTPREC  2                   2'S COMPLEMENT OF DIVIDEND
         SPACE     3
*****************************************************************
*                  ------   L E F T Z E R O   ------
*
*        THIS SUBROUTINE FINDS THE NUMBER OF LEADING BINARY ZEROS
*    IN A SIGNED 2'S COMPLEMENT MULTIPLE PRECISION INTEGER "A".
*    THE ALGORITHM PERFORMS A LINEAR SEARCH OF OPERAND A TO FIND
*    THE FIRST FULLWORD WITH A NON-ZERO BIT.  A BINARY SEARCH
*    IS PERFORMED ON THE WORD TO FIND THE BIT POSITION OF THE
```

```
*    FIRST "1" AND THUS PRODUCE A FINAL LEADING ZERO TALLY.
*
*    UPON ENTRY, R1 POINTS TO AN ARGUMENT LIST CONSISTING OF ...
*        R1 --> @ K -- NUMBER OF FULLWORDS IN INTEGER A
*                  @ A  (K WORDS)
*                  @ # LEADING ZEROS IN A  (1 WORD)
*********************************************************************
          SPACE
LEFTZERO  LOENTRY
****** LOAD OPERAND ADDRESSES AND SET UP FOR 1'S SEARCH.
          BYTEOFF WORD,R9,          PUT BYTE OFFSET OF LAST WORD OF
*                                      OPERAND A INTO R9. LIMIT REG.
          LM        R6,R7,4(R1)     LOAD @ A INTO R6 ;
*                                      @ # LEADING ZEROS INTO R7.
          LA        R8,4            LOAD INCREMENT REGISTER.
          SR        R4,R4           ZERO LEFTZEROS COUNT.
          SR        R5,R5           ZERO INDEX REGISTER.
          SPACE
******* PERFORM LINEAR SEARCH OF INTEGER "A" FOR FIRST NON-ZERO WORD.
WORDSRCH  L         R2,0(R5,R6)     LOAD NEXT WORD OF OPERAND A
          LTR       R2,R2              AND TEST.
          BM        STOZEROS        IF NEGATIVE, ALL LEADING
*                                      ZEROS ARE FOUND.
          BP        BITSRCH         IF POSITIVE, SEARCH CURRENT WORD.
          LA        R4,32(R4)       INCREASE LEADING 0 COUNT BY 32.
          BXLE      R5,R8,WORDSRCH  BRANCH UNTIL 1 IS FOUND OR A IS
*                                      FULLY SCANNED.
          SPACE
          B         STOZEROS        NO 1'S WERE FOUND.  STORE ZEROS.
BITSRCH   LA        R5,32           INITIALIZE # BITS TO SHIFT.
          LA        R6,1            LOAD 1 FOR SCAN-DONE COMPARISON.
          SR        R3,R3           ZERO RIGHT REG. FOR DOUBLE SHIFTS
          SPACE
****** MAKE A BINARY SEARCH THROUGH THE FIRST WORD WITH A NON-ZERO
****** BIT.  FIND IT AND MAKE A FINAL TALLY OF LEADING ZEROS.
BITLOOP   CR        R2,R6           IS SEARCH FOR 1 COMPLETED?
          BE        LDDONE          WE FIND THE FIRST 1.
          BH        RTSHIFT         NOT ENOUGH RIGHT SHIFT; TRY MORE
          AR        R4,R5           TOO MUCH RT. SHIFT; ACCUMULATE
*                                      LEADING ZEROS.
          SRL       R5,1            DIVIDE # BITS TO SHIFT BY 2.
          SLDL      R2,0(R5)        SHIFT BITS BACK LEFT BY R5.
          B         BITLOOP         CONTINUE SEARCH FOR 1.
*
RTSHIFT   SRL       R5,1            DIVIDE BITS TO SHIFT BY 2.
          SRDL      R2,0(R5)        SHIFT BITS RIGHT R5 POSITIONS.
          B         BITLOOP         CONTINUE SEARCH FOR 1.
          SPACE
LDDONE    AR        R4,R5           ACCUMULATE LEADING ZEROS
          SR        R4,R6              AND ADJUST FOR FINAL TALLY.
STOZEROS  ST        R4,0(R7)        STORE # OF A'S LEADING ZEROS.
          LORETURN
          SPACE     3
```

```
*********************************************************************
*                 -------   P O W R M U L T   -------
*
*          THIS SUBROUTINE PERFORMS A MULTIPLE-PRECISION MULTIPLI-
*    CATION OR DIVISION OF A K-WORD OPERAND BY A POWER OF 2.  THE
*    RESULT IS ASSUMED TO FIT IN K WORDS.
*
*    UPON ENTRY, R1 POINTS TO AN ARGUMENT LIST CONSISTING OF ...
*        R1  - -> C K  --- # FULLWORDS IN OPERANDS
*                  C SHIFT FACTOR (1 WORD) --- # BITS TO SHIFT LEFT
*                         OR RIGHT.  IF VALUE IS NEGATIVE, A RIGHT
*                         SHIFT IS PERFORMED (DIVIDE).  OTHERWISE,
*                         A LEFT SHIFT IS MADE (MULTIPLY).
*                  C A (K WORDS)              --- OPERAND TO BE SHIFTED
*                  C A * (2 ** SHIFT FACTOR) --- RESULT OF SHIFT
*         IF SO DESIGNATED BY THE CALLING PROGRAM, THE SHIFT RESULT
*    CAN BE STORED BACK AT OPERAND A'S ADDRESS.
*********************************************************************
         SPACE
POWRMULT LOENTRY
****** LOAD OPERAND ADDRESSES AND GET OFFSETS.  ZERO WORK AREA.
         LM       R8,R10,4(R1)      COPY C SHIFT FACTOR INTO R8;
*                                        C OPERAND A INTO R9;
*                                        C A * (2**P) INTO R10.
         BYTEOFF  WORD,R3,          GET OFFSET (4K-4) OF LOW WORD
*                                      OF OPERAND INTO R3.
         LA       R11,3(R3)         GET OFFSET (4K-1) OF A'S LOW BYTE.
         LA       R14,RESULT        LOAD ADDRESS OF RESULT WORK AREA.
         ZEROLONG R11,0(R14)        ZERO RESULT WORK AREA.
         SPACE
******* CHECK FOR LEFT, RIGHT, OR ZERO SHIFT.
         L        R4,0(R8)          LOAD # SHIFT BITS INTO R4
         SRDA     R4,32                AND SHIFT RIGHT INTO R5.
         D        R4,=F'32'         GET TOTAL WORD SHIFT IN R5 AND
*                                      REMAINING BIT SHIFT IN R4.
         LTR      R5,R5             IF NEGATIVE SHIFT, SHIFT RIGHT.
         BP       MLSHIFT           IF POSITIVE SHIFT, SHIFT LEFT.
         LTR      R9,R10            IS RESULT TO BE PLACED AT
*                                      ORIGINAL VALUE?
         BE       GOAWAY            IF SO, QUIT.
         MOVELONG R11,0(R10),0(R9)  MOVE A INTO RESULT WHEN SHIFT
*                                      = 0 BITS.
         B        GOAWAY
         SPACE
****** PERFORM LEFT SHIFT (MULTIPLY) OF OPERAND A.
MLSHIFT  LA       R3,0(R3,R9)       GET C OF LOW WORD IN OPERAND A.
         SLA      R5,2              MULTIPLY BY 4 FOR BYTE SHIFT.
         AR       R9,R5             GET C OF BEGINNING HIGH WORD
*                                      TO SHIFT.
         LA       R2,4              LOAD INCREMENT FOR LOOP.
*
*** SHIFT LEFT ONE WORD AT A TIME.
SHFTLOOP L        R6,0(R9)          LOAD LEFT REG. FROM OPERAND A.
```

```
         SR        R7,R7              ZERO RIGHT SHIFT REGISTER.
         CR        R9,R3              IS THIS LOW WORD OF OPERAND A?
         BE        SHIFTP             IF SO, LEAVE RIGHT REG.= 0.
         L         R7,4(R9)           LOAD RIGHT REG. FROM OPERAND A.
SHIFTP   SLDL      R6,0(R4)           SHIFT R6,R7 LEFT "POWER" BITS.
         ST        R6,0(R14)          STORE LEFT REG. IN PRODUCT.
         LA        R14,4(R14)         INCREMENT PRODUCT REGISTER.
         BXLE      R9,R2,SHFTLOOP     LOOP UNTIL SHIFT IS COMPLETE.
         B         RSLTMOV
         SPACE
****** PERFORM RIGHT SHIFT (DIVIDE) OF OPERAND A.
DVSHIFT  LR        R2,R3              COPY OFFSET OF LOW WORD OF A.
         SLA       R5,2               MULTIPLY BY 4 FOR BYTE SHIFT.
         AR        R2,R5              GET OFFSET OF BEGINNING LOW WORD
*                                       TO SHIFT.
         LA        R14,0(R3,R14)      GET @ OF LOW WORD OF OPERAND A.
         L         R3,=F'-4'          LOAD INCREMENT & LIMIT FOR LOOP.
         AR        R9,R3              POINT R9 4 BYTES BEFORE OPERAND A.
         LCR       R4,R4              GET POSITIVE BIT SHIFT.
*
*** SHIFT RIGHT ONE WORD AT A TIME.
RTSHFT   L         R7,4(R2,R9)        LOAD RIGHT REG. FROM OPERAND A.
         SR        R6,R6              ZERO LEFT SHIFT REGISTER.
         LTR       R2,R2              IS THIS HIGH WORD OF OPERAND A?
         BZ        SHFTP              IF SO, LEAVE LEFT REGISTER = 0.
         L         R6,0(R2,R9)        LOAD LEFT REG. FROM OPERAND A.
SHFTP    SRDL      R6,0(R4)           SHIFT R6,R7 RIGHT "SHFTFACT" BITS.
         ST        R7,0(R14)          STORE RIGHT REG. IN RESULT.
         AR        R14,R3             DECREMENT RESULT POINTER.
         BXH       R2,R3,RTSHFT       LOOP UNTIL RIGHT SHIFT IS COMPLETE
         SPACE
RSLTMOV  MOVELONG  R11,0(R10),RESULT  MOVE RESULT INTO OUTPUT ADDRESS.
GOAWAY   LORETURN
RESULT   MULTPREC  2                  WORK AREA FOR A*(2**SHIFT FACTOR)
.        SPACE     3
*********************************************************************
*                 ------   T W O S C O M P   ---
*
*        THIS SUBROUTINE COMPUTES THE TWO'S COMPLEMENT OF A MULTIPLE-
*   PRECISION INTEGER.
*
*   UPON ENTRY, R1 POINTS TO AN ARGUMENT LIST CONSISTING OF ...
*       R1 ---> @ K              --- # FULLWORDS TO HOLD OPERANDS
*               @ A  (K WORDS)   --- INTEGER OPERAND TO BE COMPLEMENTED
*               @ -A  (K WORDS) --- 2'S COMPLEMENT RESULT
*               @ OVERFLOW ERROR FLAG (K WORDS) --- SET TO 1 IF A IS
*                                                   MAX. NEGATIVE INT.
*        IF SO DESIRED BY THE CALLING PROGRAM, -A CAN OPTIONALLY
*   PLACED IN STORAGE LOCATION A.
*********************************************************************
         SPACE
TWOSCOMP LOENTRY
         BYTEOFF   WORD,R2,           GET BYTE  OFFSET  OF LOW  ORDER
```

```
*                                          WORD (4K - 4) FOR INDEX, R2.
         LM       R8,R10,4(R1)       LOAD @ A INTO R8;
*                                          @ -A INTO R9;
*                                          @ ERROR FLAG INTO R10.
         SPACE
****** IF INTEGER TO BE COMPLEMENTED IS THE MAXIMUM NEGATIVE INTEGER
****** IN K-WORDS, SET AN OVERFLOW ERROR FLAG AND RETURN TO CALLER.
****** OTHERWISE, SET UP FOR COMPLEMENTATION LOOP.
         LA       R3,3(R2)           GET LENGTH FOR COMPARE (4K-1).
         CMPLONG  R3,0(R8),MAXNEG    COMPARE A TO MAX. NEG. INTEGER.
         BNE      CMPL               IF NO OVERFLOW, FIND 2'S COMPL.
         LA       R6,1               SET OVERFLOW FLAG
         B        STFLAG                AND RETURN.
*
CMPL     L        R7,=F'-4'          LOAD INCREMENT FOR INDEX REGISTER.
         LA       R0,1               LOAD INITIAL CARRY FOR ADD
         SR       R6,R6              ZERO R6 FOR ZERO CARRY.
         LCR      R14,R0             LOAD -1 FOR COMPLEMENTING A WORD.
         SPACE
****** THIS LOOP COMPUTES THE TWO'S COMPLEMENT OF OPERAND A.  THE
****** ONE'S COMPLEMENT OF EACH WORD IS TAKEN, AND THE TWO'S
****** COMPLEMENT OF OPERAND A IS OBTAINED BY PROPAGATING (LEFT)
****** AN ADD OF 1 TO THE RIGHTHAND WORD OF A.
COMPLOOP L        R11,0(R2,R8)       LOAD NEXT WORD TO COMPLEMENT.
         XR       R11,R14            GET 1'S COMPLEMENT OF THE WORD.
         LTR      R0,R0              CHECK CARRY.
         BZ       STORTWOS           IF NO FURTHER CARRY, STORE
*                                       COMPLEMENTED WORD.
         ALR      R11,R0             ADD CARRY.
         LR       R0,R6              ZERO NEXT CARRY.
         BC       NOCARRY,STORTWOS   IF NO MORE CARRY, STORE WORD.
         LA       R0,1               ADDITIONAL CARRY;  LOAD IT.
STORTWOS ST       R11,0(R2,R9)       STORE CURRENT COMPLEMENTED WORD.
         BXH      R2,R7,COMPLOOP     CONTINUE UNTIL ALL WORDS COMPL.
         SPACE
STFLAG   ST       R6,0(R10)          STORE ERROR FLAG (1 = SET).
         LORETURN
MAXNEG   MULTPREC 2,X'80000000',,F'0' MAXIMUM NEGATIVE INTEGER FOR
*                                       COMPARISON TO OPERAND A
         SPACE    3
*********************************************************************
*                 ------  L S T C O M M L  ------
*
*        THIS PROCEDURE COMPUTES THE LEAST COMMON MULTIPLE OF TWO
*    POSITIVE MULTIPLE PRECISION INTEGERS A AND B.
*
*    UPON ENTRY, R1 POINTS TO AN ARGUMENT LIST CONSISTING OF ...
*        R1 --> @ K  -- # FULLWORDS IN A AND B
*               @ A  (K WORDS)
*               @ B  (K WORDS)
*               @ LEAST COMMON MULTIPLE (2K WORDS)
*
*    SUBROUTINES CALLED:
```

```
*        REDUCE    --- REDUCES A RATIONAL NUMBER (K-WORD NUMERATOR
*                     AND DENOMINATOR) TO LOWEST TERMS
*        MULTIPLY  --- MULTIPLIES TWO K-WORD INTEGERS (RESULT IN
*                     2K WORDS)
*********************************************************************
          SPACE
LSTCOMML  HIENTRY   R11,LSTSAVE
          BYTEOFF   BYTE,R2,            GET # BYTES-1 (4K-1) IN OPERANDS.
          LK        R7,R10,0(R1)        LOAD @ K INTO R7;
*                                              @ A INTO R8;
*                                              @ B INTO R9;
*                                              @ LEAST COM. MULT. INTO R10.
          MOVELONG  R2,APRIME,0(R8)     MOVE A INTO APRIME.
          MOVELONG  R2,BBPRIME,0(R9)    MOVE B INTO BBPRIME.
****** GET REDUCED FRACTION A'/B' AND MULTIPLY REDUCED A' BY B TO GET
****** LEAST COMMON MULTIPLE.
          CALL      REDUCE,,(R7,APRIME,BBPRIME,ERRFLAG)
          CALL      MULTIPLY,,(R7,APRIME,R9,R10,ERRFLAG)
          HIRETURN
          SPACE     2
*         ******    DATA STORAGE FOR LSTCOMML CSECT    ******
ERRFLAG   DS        F                   ERROR FLAG FOR CALLS TO
*                                          REDUCE, MULTIPLY
APRIME    MULTPREC  1                   STORAGE FOR REDUCED A
BBPRIME   MULTPREC  1                   STORAGE FOR REDUCED B
          SPACE     3
```

APPENDIX C

CHOQUET SERIES PROGRAM (ASSEMBLER)

```
         PRINT    ON
         TITLE    'TEST PROGRAM TO CALCULATE CHOQUET SERIES'
*********************************************************************
*                          --- MAIN PROGRAM ---                      *
*                - - -   C H O Q U E T   S E R I E S   - - - -        *
*                                                                    *
*     THIS PROGRAM USES MULTIPLE PRECISION, 2'S COMPLEMENT INTEGER   *
*   ARITHMETIC TO CALCULATE A SPECIFIED NUMBER OF TERMS OF THE       *
*   CHOQUET SERIES.  THE CHOQUET SERIES IS DEFINED AS FOLLOWS:       *
*                                                                    *
*       A(1) IS A POSITIVE INTEGER.                                  *
*       IF A(N) IS ODD, R(N) = 1,                                    *
*                       2A(N+1) = 1 + 3A(N)  .                       *
*       IF A(N) IS EVEN, R(N) = 0,                                   *
*                        2A(N+1) = 3A(N)   .                         *
*                                                                    *
*       VALUES OF N, A(N), AND R(N) ARE REPORTED FOR VALUES OF N     *
*   THAT ARE MULTIPLES OF A SPECIFIED NUMBER OF ITERATIONS.          *
*                                                                    *
*   SUBROUTINES CALLED:                                              *
*       RATSET   --- VERIFIES THAT PRECISION AND K AGREE AND OPENS   *
*                     OUTPUT PRINT FILE FT06F001                     *
*       MULTIPLY --- PERFORMS MULTIPLICATION OF TWO K-WORD INTEGERS  *
*                     (RESULT IN 2K WORDS)                           *
*       WORDADD  --- ADDS AT A SPECIFIED OFFSET A ONE-WORD INTEGER   *
*                     TO A K-WORD 2'S COMPLEMENT INTEGER (RESULT IN K *
*                     WORDS)                                         *
*       POWRMULT --- MULTIPLIES OR DIVIDES A K-WORD 2'S COMPLEMENT   *
*                     INTEGER BY A SPECIFIED POWER OF 2              *
*       OUTPUT   --- CONVERTS TO EBCDIC CHARACTERS AND PRINTS MULTI- *
*                     PLE PRECISON 2'S COMPLEMENT RATIONAL NUMBERS    *
*       CLSOUT   --- CLOSES OUTPUT PRINT FILE FT06F001               *
*                                                                    *
*   FATAL ERRORS:                                                    *
*       1.  "PRECSION" AND "K" OF SUPPORT ROUTINES DISAGREE ---      *
*           ERROR #62                                                *
*       2.  OVERFLOW (INTO MORE THAN K WORDS) IN MULTIPLICATION      *
*           3 * A(N)                                                 *
*       3.  OVERFLOW IN ADDITION 3*A(N) + 1                          *
*                                                                    *
*   REQUIRED PROGRAM INPUT:   NONE                                   *
*                                                                    *
*   DATA INITIALIZATIONS/PROGRAM MODIFICATIONS:                      *
*       THE FOLLOWING VARIABLES MAY BE MODIFIED AS NECESSARY FOR     *
*   EACH PROGRAM EXECUTION BY CHANGING THEIR DC STORAGE INITIALI-    *
*   ZATIONS ...                                                      *
*       ITERNO   --- TOTAL NUMBER OF TERMS OF CHOQUET SERIES TO      *
```

```
*                    COMPUTE                                          *
*       PRTINTVL --- NUMBER OF ITERATIONS AFTER WHICH N, A(N), AND    *
*                    R(N) ARE PRINTED                                 *
*       PRECSION --- NUMBER OF 32-BIT WORDS TO HOLD A(N), R(N)  (AN   *
*                    INTEGER FROM 2 TO 303)                           *
*       AN       --- INITIAL TERM A(1) OF CHOQUET SERIES (A POSITIVE  *
*                    INTEGER --- SET TO 4 IN THE SAMPLE PROGRAM)      *
*************************************************************************
         SPACE
MAIN     MIENTRY  R11,MAINSAVE
         SPACE
****** OPEN OUTPUT PRINT FILE AND VERIFY THAT "PRECSION" OF MAIN
****** PROGRAM AGREES WITH "K" OF CSECT SETUP.  CONVERT PRECISION
****** TO CHARACTERS AND PRINT.
         CALL     RATSET,,(PRECSION) OPEN OUTPUT PRINT FILE AND
*                                      VERIFY THAT PRECSION = K.
         L        R12,=A(PRINTOUT)  ESTABLISH ADDRESSABILITY TO
         USING    PRINTOUT,R12         PRINTOUT DCB USING R12.
         PUTOUT   =CL86'   *-*-*  MULTIPLE PRECISION INTEGER ARITHMETIX
               C COMPUTATION OF CHOQUET SERIES  *-*-*',86
         L        R5,PRECSION       LOAD PRECISION INTO R5
         CVD      R5,PACKK             AND CONVERT TO PACKED DECIMAL.
         MVC      OUTK(4),=X'40202120' MOVE EDIT PATTERN TO OUTK.
         ED       OUTK(4),PACKK+6   CONVERT K TO CHARACTERS
         PUTOUT   OUTKMSG,78           AND PRINT PRECISION MESSAGE.
         SPACE
****** INITIALIZE VALUES AND REGISTERS WITHIN CHOQUET SERIES LOOP.
         LA       R1,KADD           LOAD C OF C OF PRECISION K.
         BYTEOFF  WORD,R10,         LOAD OFFSET OF LOW WORD OF
*                                      A(N) INTO R10.
         ST       R10,ADDOFF        STORE OFFSET FOR "WORDADD."
         L        R2,ADDONE         LOAD C OF K-WORD CONSTANT OF 1.
         LA       R3,2              LOAD INDEX REGISTER WITH N = 2.
         LA       R4,1              LOAD INCREMENT = 1 INTO R4.
         L        R5,ITERNO         LOAD LIMIT REGISTER WITH NUMBER
*                                      OF ITERATIONS TO PERFORM.
         LA       R9,ANPRIME-4(R10) LOAD C OF SECOND K WORDS OF
*                                      PRODUCT 3 * A(N).
         LA       R6,AN+3(R10)      LOAD C OF LAST BYTE OF A(N).
         LA       R7,RN+3(R10)      LOAD C OF LAST BYTE OF R(N).
         L        R8,PRTINTVL       LOAD DIVISOR FOR PRINT CHECK.
         SPACE
****** LOAD ARGUMENT LISTS FOR CALLS WITHIN CHOQUET SERIES LOOP.
         LDARGTBL MULTS,(PRECSION,AN,THREE,ANPRIME,ERRORS)
         LDARGTBL ADDIT,(PRECSION,ADDOFF,R9,ADDWORD,ERRORS)
         LDARGTBL POWRS,(PRECSION,SHIFT,R9,AN)
         LDARGTBL OUTS,(PRECSION,AN,R2,TITLE)
         SPACE    2
****** THIS LOOP COMPUTES A SPECIFIED NUMBER OF TERMS ("ITERNO") IN
****** THE CHOQULT SERIES.  THE VALUES OF A(N) AND R(N) ARE REPORTED
****** FOR N = MULTIPLES OF A SPECIFIED NUMBER OF ITERATIONS
****** ("PRTINTVL").
         SPACE
```

```
*** MULTIPLY CURRENT A(N) BY 3.  CHECK FOR OVERFLOW INTO MORE THAN
*** K WORDS OR INTO SIGN BIT.  IF OVERFLOW OCCURS, ABORT EXECUTION.
CHKLOOP  CALL     MULTIPLY,MULTS,
         TM       0(R9),X'80'        TEST SIGN BIT OF 2ND K WORDS OF
*                                      PRODUCT.
         BO       NEGATIV1           IF SET, CHECK LEFT K WORDS FOR 1'S
         L        R14,=A(ZERO)       LOAD C OF K-WORD CONSTANT OF 0.
         CLCLONG  R10,0(R14),ANPRIME  ARE LEFT K WORDS 0?
         BNE      OVER1              IF NOT, WE HAVE OVERFLOW.
         B        ANCHK              RESULT OK -- CONTINUE CALCULATION.
NEGATIV1 L        R14,=A(MINUS1)     LOAD C OF K WORDS OF 1'S.
         CLCLONG  R10,0(R14),ANPRIME  ARE LEFT K WORDS "-1"?
         BE       ANCHK              IF SO, NO OVERFLOW.
OVER1    PUTOUT   =CL77'0******  ERROR -- OVERFLOW IN MULTIPLICATION OFX
               A(N) BY 3.  EXECUTION CEASES.',77
         B        GETOUT2            CEASE EXECUTION--RETURN TO OP. SYS.
         SPACE
*** IF A(N) IS ODD, ADD 1 TO 3*A(N).  ABORT EXECUTION IF OVERFLOW
*** HAS OCCURRED.
ANCHK    TM       0(R6),X'01'        IS A(N) ODD?
         BNO      EVEN               IF NOT, BRANCH.
         CALL     WORDADD,ADDIT      ADD 1 TO 3*A(N).
         TM       ERRORS+3,X'FF'     ADDITION OVERFLOW, 3*A(N) + 1?
         BNM      EVEN               IF NOT, DIVIDE PRODUCT BY 2.
         PUTOUT   =CL75'0******  ERROR -- OVERFLOW IN ADDITION OF 1 TO X
               3 * A(N).  EXECUTION CEASES.',75
         B        GETOUT2            CEASE EXECUTION--RETURN TO OP. SYS.
         SPACE
*** DIVIDE (3 * A(N)) OR (3 * A(N) + 1) BY 2 TO GET A(N+1).
*** THE RESULT IS STORED IN "AN" FOR THE NEXT ITERATION.  IF
*** A(N+1) IS ODD, SET R(N+1) TO 1.  OTHERWISE, SET R(N+1) TO 0.
EVEN     CALL     POWRMULT,POWRS,
         NI       0(R7),X'00'        SET R(N) TO 0 (EVEN) BY DEFAULT.
         TM       0(R6),X'01'        IS NEW A(N) ODD?
         BNO      EVEN2              IF NOT, BRANCH.
         OI       0(R7),X'01'        SET NEW R(N) = 1.
         SPACE
****** IF N IS A MULTIPLE OF THE SPECIFIED ITERATION INTERVAL,
****** CONVERT N, R(N), AND A(N) TO EBCDIC CHARACTERS AND PRINT.
EVEN2    LR       R14,R3             COPY N INTO R14
         SRDA     R14,32               AND SHIFT RIGHT INTO R15.
         DR       R14,R8             DIVIDE N BY ITERATION INTERVAL
*                                      FOR PRINTING RESULTS.
         LTR      R14,R14            TEST REMAINDER IN R14.
         BNZ      NEXTN              IF NOT 0, GO DO NEXT ITERATION.
         CVD      R3,PACKN           CONVERT N TO PACKED DECIMAL.
         MVC      TITLE+38(6),PATTRN MOVE EDIT PATTERN INTO "TITLE."
         ED       TITLE+38(6),PACKN+5 EDIT N FOR PRINTING.
         MVC      TITLE+53(1),0(R7) MOVE R(N) INTO TITLE
         OI       TITLE+53,X'F0'       AND CONVERT TO CHARACTER FOR
*                                      PRINTING.
CHARVERT CALL     OUTPUT,OUTS,       CONVERT BINARY A(N) AND R(N) TO
*                                      DECIMAL CHARACTERS AND PRINT.
```

```
         SPACE
NEXTN    BXLE     R3,R4,CHOLOOP     COMPUTE NEXT TERMS OF CHOQUET
*                                     SERIES.
         SPACE    2
****** CLOSE OUTPUT PRINT FILE FT06F001 AND RETURN TO CALLER.
         CALL     CLSOUT            CLOSE OUTPUT PRINT FILE.
GETOUT2  HIRETURN
         SPACE    2
*        ******   DATA STORAGE FOR MAIN PROGRAM    ******
PACKK    DS       D                 PACKED DECIMAL VALUE OF PRECS. K
OUTKMSG  DC       CL68'0            --- SOLUTION USING K-WORD RATIONAL AR
               ITHMETIC, WHERE K ='
OUTK     DS       4C                EBCDIC CHARS. FOR PRECISION K
         DC       C'  --- '
PACKN    DS       D                 PACKED DECIMAL VALUE OF ITER. #
PATTRN   DC       X'4020202021 20'  PATTERN FOR EDITING N
ERRORS   DS       F                 ERROR FLAG FOR CALLS TO MULTIPLY,
*                                     WORDADD.
ADDOFF   DS       F                 OFFSET AT WHICH TO ADD 1 TO
*                                     3 * A(N)
ADDWORD  DC       F'1'              VALUE OF 1 FOR ADDITION TO 3 * A(N)
SHIFT    DC       F'-1'             SHIFT FOR DIVISION BY 2
KADD     DC       A(K)              ADDRESS OF K
ADDONE   DC       A(ONE111)         ADDRESS OF K-WORD CONSTANT OF 1
TITLE    DC       CL64' *  CHOQUET SERIES WHERE A(1)=4  --- K =      , R(X
               N) =  , A(N) = :'
         SPACE
*---- FOLLOWING DATA CAN BE MODIFIED FOR EACH PROGRAM EXECUTION ----  *
*-------------------------------------------------------------------------*
ITERNO   DC       F'7000'           # TERMS IN SERIES TO COMPUTE
PRTINTVL DC       F'500'            ITERATION FREQUENCY FOR PRINTING
*                                     OF RESULTS
PRECSION DC       F'129'            NUMBER OF FULLWORDS IN OPERANDS
AN       MULTPREC 1,F'0',1,F'4'     STORAGE FOR TERM A(N) --- A(1) = 4
*-------------------------------------------------------------------------*
         SPACE
RN       MULTPREC 1,F'0',1,F'0'     STORAGE FOR TERM R(N) --- R(1) = 0
AMPRIME  MULTPREC 2                 STORAGE FOR PRODUCT 3 * A(N)
THREE    MULTPREC 1,F'0',1,F'3'     MULTIPLE PRECISION VALUE OF 3
         SPACE    3
```

APPENDIX D

INVERSE HILBERT MATRIX PROGRAM (FORTRAN)

```
C=======================================================================
C=       COMPUTATION OF ELEMENTS OF INVERSE OF HILBERT MATRIX
C=
C=       LET A BE A FINITE N X N SEGMENT OF THE INFINITE HILBERT
C=    MATRIX, IN WHICH INDIVIDUAL ELEMENTS ARE GIVEN BY
C=
C=       A(I,J) = 1/(I+J-1)    ,        I,J = 1, 2, 3, ..., N    .
C=
C=    THIS PROGRAM COMPUTES THE COEFFICIENTS OF THE INVERSE MATRIX
C=    B = A(-1) DIRECTLY FROM THE RELATIONSHIP
C=
C=    B(I,J) = (((-1)**(I+J) * (N+I-1) FACTORIAL * (N+J-1) FACTORIAL) /
C=             ((I+J-1) * ((I-1) FACTORIAL * (J-1) FACTORIAL)**2
C=             *(N-I) FACTORIAL * (N-J) FACTORIAL)    .
C=
C=    THE PROGRAM COMPUTES THE INVERSE B(I,J) ELEMENTS FOR A SPECIFIED
C=    ROW I OF THE MATRIX.  (BECAUSE OF MATRIX SYMMETRY, THESE ARE
C=    ALSO THE ELEMENTS OF COLUMN I OF A(-1)).  SINCE THE B(I,J)
C=    ELEMENTS EVALUATE TO INTEGERS, MULTIPLE PRECISION, 2'S COMPLEMENT
C=    INTEGER ARITHMETIC IS EMPLOYED.
C=
C=    REQUIRED PROGRAM INPUT:   NONE
C=
C=    SUBROUTINES CALLED:
C=       RATSET --  VERIFIES THAT "PRECSN" OF MAIN PROGRAM AGREES WITH
C=                  ASSEMBLER'S "K" (CSECT SETUP) AND OPENS OUTPUT PRINT
C=                  FILE FT06F001
C=       CLSOUT --- CLOSES OUTPUT PRINT FILE FT06F001
C=       ADDITION -- ADDS TWO K-WORD 2'S COMPLEMENT INTEGERS (RESULT
C=                   IN K WORDS)
C=       SUBTRACT -- PERFORMS SUBTRACTION OF TWO 2'S COMPLEMENT, K-WORD
C=                   INTEGERS (RESULT IN K WORDS)
C=       MULTIPLY -- MULTIPLIES TWO 2'S COMPLEMENT, K-WORD INTEGERS
C=                   (RESULT IN 2K WORDS)
C=       DIVISION --- PERFORMS INTEGER DIVISION OF 2'S COMPLEMENT,
C=                    K-WORD INTEGERS (QUOTIENT AND REMAINDER IN K WORDS)
C=       TWOSCOMP -- COMPUTES THE 2'S COMPLEMENT OF A K-WORD INTEGER
C=                   (RESULT IN K WORDS)
C=       OUTPUT ---- CONVERTS TO EBCDIC CHARACTERS AND PRINTS A MULTIPLE
C=                   PRECISION RATIONAL NUMBER
C=
C=    VARIABLES:
C=       N       --- DIMENSION N OF THE N X N HILBERT MATRIX
C=       N2PLS1 --  2 * N PLUS 1 (1 MORE THAN NUMBER OF FACTORIALS TO BE
C=                  COMPUTED)
C=       PRECSN ---- NUMBER OF FULLWORDS IN MULTIPLE PRECISION INTEGERS
C=       ROW     -- ROW NUMBER FOR WHICH ELEMENTS OF INVERSE MATRIX
```

```
C=                  ARE COMPUTED.
C=       MLTROW  -- "PRECSN"-WORD INTEGER CONTAINING "ROW"
C=       COL     -- COLUMN NUMBER WITHIN CURRENT "ROW"
C=       COLUMN  -- "PRECSN"-WORD INTEGER CONTAINING "COL"
C=       RCSUM   -- SINGLE-PRECISION REAL QUANTITY (ROW+COL)/2
C=       NFACT   -- ARRAY OF FACTORIALS CONTAINING 0 FACTORIAL TO (2N)
C=                  FACTORIAL -- N FACTORIAL IS STORED IN ARRAY POSITION
C=                  NFACT(N+1).
C=       MULTN   -- MULTIPLE PRECISION N FOR COMPUTING N FACTORIAL
C=       INVELE  -- A PARTICULAR ELEMENT OF THE INVERSE OF THE HILBERT
C=                  MATRIX
C=       PRODCT  -- STORAGE FOR 2-K WORD PRODUCTS
C=       WORK1   -- K-WORD WORKING STORAGE
C=       WORK2   -- K-WORD WORKING STORAGE
C=       ERRFLG  -- ERROR FLAG FOR CALLS TO MULTIPLE PRECISION ARITHMETIC
C=                  ROUTINES
C=       TITLE   -- 64 CHARACTER TITLE FOR OUTPUT
C=       ONE     -- MULTIPLE PRECISION CONSTANT = 1
C=
C=    PROGRAM MODIFICATIONS/DATA INITIALIZATIONS:
C=       THIS PROGRAM MAY BE RE-RUN WITH ANY DESIRED PRECSN, N, AND ROW.
C=    "DIMENSION" STATEMENTS FOR ARRAYS INVELE, MULTN, ONE, WORK1, WORK2,
C=    MLTROW, COLUMN, PRODCT, AND NFACT MUST BE CHANGED TO AGREE WITH
C=    THE CHOSEN PRECISION;  AND "DATA" INITIALIZATIONS FOR PRECSN,
C=    N, N2PLS1, ROW, MULTN, MLTROW, COLUMN, ONE, AND NFACT MUST BE
C=    ALTERED CORRESPONDINGLY.
C=================================================================
      IMPLICIT INTEGER (A-Z)
      REAL RCSUM
      DIMENSION INVELE(9), MULTN(9), ONE(9), WORK1(9), WORK2(9)
      DIMENSION MLTROW(9), COLUMN(9)
      DIMENSION PRODCT(9,2), NFACT(9,31)
      DIMENSION TITLE(16)
      DATA TITLE/' ***      ROW ELEMENT OF INVERSE OF N X N HILBERT MA
     1TRIX     ***'/
      DATA PRECSN/9/, N/15/, N2PLS1/31/, ROW/11/
      DATA MULTN/9*0/, MLTROW/9*0/, COLUMN/9*0/, NFACT/279*0/
      DATA ONE/9*0/
C
C---------- VERIFY THAT "PRECSN" AGREES WITH "K", THE NUMBER OF FULL-
C---------- WORDS PER OPERAND ALLOCATED BY ASSEMBLER SUPPORT ROUTINES.
C---------- OPEN OUTPUT PRINT FILE FT06F001.
      CALL RATSET(PRECSN)
C
C---------- INITIALIZE 0 FACTORIAL AND "ONE" TO 1.
C---------- COMPUTE N FACTORIAL FOR N = 0 TO N = 2N;  STORE IN ARRAY
C---------- NFACT.
      ONE(PRECSN) = 1
      NFACT(PRECSN,1) = 1
      DO 50 I = 2,N2PLS1
         MULTN(PRECSN) = I - 1
         CALL MULTIP(PRECSN,NFACT(1,I-1),MULTN,PRODCT,ERRFLG)
         DO 25 J = 1,PRECSN
```

```
                NFACT(J,I) = PRODCT(J,2)
   25         CONTINUE
   50      CONTINUE
C
C
        WRITE(6,100) N, PRECSN
  100   FORMAT(' -*-*-* MULTIPLE PRECISION COMPUTATION OF ONE ROW ',
     1          'OF THE COEFFICIENTS OF THE INVERSE OF AN N X N HILBERT',
     2          ' MATRIX, WHERE N = ',I2,' *-*-*-*',/,T40,
     3          '-- - PRECISION OF OPERANDS IS K = ',I3,' -*-*-*')
        MLTROW(PRECSN) = ROW
C
C
C
C------------ USE THE KNOWN RELATIONSHIP TO COMPUTE DIRECTLY THE N ELEMENTS
C------------ OF THE DESIGNATED "ROW" OF THE INVERSE OF THE N X N HILBERT
C------------ MATRIX.
        DO 1000 COL = 1,N
C
C           ---  MULTIPLY (N+ROW-1) FACTORIAL BY (N-COL-1) FACTORIAL.
C           ----- IF PRODUCT OVERFLOWS INTO LEFT "PRECSN" WORDS OR INTO
C           ---  SIGN BIT OF RIGHT "PRECSN" WORDS, ABORT EXECUTION.
            CALL MULTIP(PRECSN,NFACT(1,N+ROW),NFACT(1,N-COL),PRODCT,
     1                  ERRFLG)
            IF (PRODCT(1,2) .LT. 0) GO TO 150
               DO 125 I = 1,PRECSN
                  IF (PRODCT(I,1) .NE. 0) GO TO 170
  125          CONTINUE
               GO TO 190
  150       DO 160 I = 1,PRECSN
               IF (PRODCT(I,1) .NE. -1) GO TO 170
  160       CONTINUE
            GO TO 190
  170       WRITE(6,175)
  175       FORMAT ('0***  ERROR --- OVERFLOW IN PRODUCT (N-ROW-1) FACTORI
     1AL * (N-COL-1) FACTORIAL.  EXECUTION CEASES.')
            STOP
C
C           ---- DIVIDE PRODUCT BY (ROW - COL - 1) AND PLACE RESULT IN
C           ----- "INVELE".
190         COLUMN(PRECSN) = COL
            CALL ADDITI(PRECSN,MLTROW,COLUMN,WORK2,ERRFLG)
            CALL SUBTRA(PRECSN,WORK2,ONE,WORK2,ERRFLG)
            CALL DIVISI(PRECSN,PRODCT(1,2),WORK2,INVELE,WORK1,ERRFLG)
C
C           ----- DIVIDE "INVELE" BY (N-ROW) FACTORIAL AND BY
C           ------ (N-COL) FACTORIAL.
            CALL DIVISI(PRECSN,INVELE,NFACT(1,N-ROW+1),INVELE,WORK1,
     1                  ERRFLG)
            CALL DIVISI(PRECSN,INVELE,NFACT(1,N-COL+1),INVELE,WORK1,
     1                  ERRFLG)
C
C           ---  COMPUTE THE SQUARE OF THE PRODUCT OF (ROW-1)
```

```
C           ---- FACTORIAL AND (COL-1) FACTORIAL AND DIVIDE "INVELE"
C           -- -- BY THE RESULT TO GET THE MAGNITUDE OF THE INVERSE
C           -  - MATRIX ELEMENT.
            CALL MULTIP(PRECSN,NFACT(1,ROW),NFACT(1,COL),PRODCT,ERRFLG)
C
            DO 400 J = 1,PRECSN
               WORK1(J) = PRODCT(J,2)
  400       CONTINUE
C
            CALL MULTIP(PRECSN,WORK1,WORK1,PRODCT,ERRFLG)
            CALL DIVISI(PRECSN,INVELE,PRODCT(1,2),INVELE,WORK1,ERFLG)
C
C           ---- IF (ROW+COL)/2 IS ODD, THE INVERSE MATRIX ELEMENT IS
C           --   NEGATIVE.  COMPLEMENT THE PREVIOUSLY COMPUTED POSITIVE
C           -- -- RESULT.  PRINT THE SIGNED INVERSE MATRIX ELEMENT.
            RCSUM = (ROW+COL) / 2.
            IF (RCSUM - ((ROW+COL)/2) .EQ. 0) GO TO 800
               CALL TWOSCO(PRECSN,INVELE,INVELE,ERRFLG)
  800       CONTINUE
            WRITE (6,900) ROW, COL
  900       FORMAT(////,' ********************************',/,'  ROW I = ',
     1             I2,',  COLUMN J = ',I3,/,
     2             ' ********************************')
            CALL OUTPUT(PRECSN,INVELE,ONE,TITLE)
C
 1000    CONTINUE
C
C
C
C---------- CLOSE OUTPUT PRINT FILE FT06F001.
           CALL CLSOUT
C
           STOP
           END
```

APPENDIX E

MULTIPLE-PRECISION RESIDUE ARITHMETIC SUBROUTINES (ASSEMBLER)

```
         PRINT    OFF
         TITLE    '--- - MULTIPLE PRECISION RESIDUE ARITHMETIC SUBROUTN
               LNES ---- - '
         PRINT    NOGEN
*********************************************************************
*                 ---- -  O P N R E S  ------ -
*
*       THIS PROCEDURE PREPARES A RATIONAL OPERAND P/Q FOR USE IN
*    RESIDUE ADDITION, SUBTRACTION, MULTIPLICATION, OR DIVISION BY
*    COMPUTING THE RESIDUE EQUIVALENT WITH RESPECT TO A PRIME
*    MODULUS M.  THE ORIGINAL P/Q IS REDUCED TO A RATIONAL NUMBER
*    WITH A POSITIVE DENOMINATOR AND POSITIVE OR NEGATIVE NUMERATOR.
*    THE RESIDUE IS THEN COMPUTED BY MLINVGCD, WHICH APPLIES THE
*    EXTENDED EUCLIDEAN ALGORITHM.  THE RESIDUE "A" IS STORED BACK
*    AT P.
*
*    UPON ENTRY, R1 POINTS TO AN ARGUMENT LIST CONSISTING OF ...
*        R1 --->  C K              --- # FULLWORDS IN P, Q
*                 C P  (K WORDS) --- NUMERATOR OF FRACTION P/Q
*                                    (RESIDUE A = !P/Q! MOD M IS
*                                     STORED HERE UPON RETURN.)
*                 C Q  (K WORDS) --- DENOMINATOR OF FRACTION P/Q
*                                    (UNCHANGED BY THE PROCEDURE)
*                 C M  (K WORDS) --- A MERSENNE PRIME MODULUS
*
*    FATAL ERRORS:
*        Q IS 0 (DIVISION BY ZERO) --- ERROR #1
*        P OR Q IS THE GREATEST NEGATIVE INTEGER THAT CAN BE
*            CONTAINED IN K WORDS --- ERRORS #2 AND #3
*        REDUCED P OR Q IS GREATER IN ABSOLUTE VALUE THAN THE ORDER
*            M OF FAREY FRACTIONS --- ERRORS #4 AND #5.
*
*    SUBROUTINES CALLED:
*        FERROR   --- UPON FATAL ERROR, ABORTS EXECUTION WITH ERROR
*                     MESSAGE
*        TWOSCOMP --- COMPUTES 2'S COMPLEMENT OF A K-WORD INTEGER
*        REDUCE   --- REDUCES A RATIONAL NUMBER (K-WORD NUMERATOR
*                     AND DENOMINATOR) TO LOWEST TERMS.
*        MLINVGCD --- USES THE EXTENDED EUCLIDEAN ALGORITHM OF GREGORY
*                     AND KORNERUP TO COMPUTE THE RESIDUE OF RATIONAL
*                     NUMBER P/Q.
*********************************************************************
         SPACE
OPNRES   HIENTRY  R11,OPNDSAV
******** LOAD ARGUMENT ADDRESSES AND COMPUTE LENGTH CODE OF OPERANDS.
         LM       R7,R10,0(R1)       LOAD C K INTO R7;
*                                          C P INTO R8;
```

```
*                                          @ Q INTO R9;
*                                          @ MODULUS INTO R10.
         BYTEOFF  BYTE,R2,          GET # BYTES-1 (4K-1) OF OPERANDS
         SPACE
****** CHECK FOR ZERO DENOMINATOR.  IF DENOMINATOR IS ZERO, CEASE
****** EXECUTION WITH ERROR #1.
         L        R3,ADDR0          LOAD @ OF K-WORD CONSTANT OF 0.
         CMPLONG  R2,0(R9),0(R3)    CHECK FOR ZERO DENOMINATOR.
         BNE      QSIGN             IF NOT 0, CHECK SIGN OF Q.
         LA       R14,1             LOAD ERROR #1 INTO R14
         ST       R14,ERRORFLG         AND STORE IN ERROR FLAG.
         CALL     FERROR,,(ERRORFLG) ABORT EXECUTION.
QSIGN    MOVELONG R2,QCOPY,0(R9)    USE A COPY OF THE DENOMINATOR
*                                      IN COMPUTATIONS.
         TM       0(R9),B'10000000' TEST SIGN OF Q.
         BNO      REDUC             IF POSITIVE, REDUCE P/Q.
         SPACE
****** COMPLEMENT NUMERATOR AND DENOMINATOR TO GET DENOMINATOR
****** WITH A POSITIVE VALUE.  IF EITHER IS MAXIMUM NEGATIVE
****** INTEGER, ABORT EXECUTION.  REDUCE THE NUMERATOR AND
****** DENOMINATOR TO LOWEST TERMS AND STORE BACK AT P, Q.
         CALL     TWOSCOMP,,(R7,R8,R8,ERRORFLG)  COMPLEMENT NUM.
         ERRORCHK ERRORFLG,FERROR,2  ABORT UPON 2'S COMPL. OVERFLOW.
         CALL     TWOSCOMP,,(R7,QCOPY,QCOPY,ERRORFLG)  COMPLEMENT DEN.
         ERRORCHK ERRORFLG,FERROR,3  ABORT UPON 2'S COMPL. OVERFLOW.
REDUC    CALL     REDUCE,,(R7,R8,QCOPY,ERRORFLG) REDUCE FRACTION.
         ERRORCHK ERRORFLG,FERROR,2  ABORT IF P IS MAX. NEG. INTEGER.
         SPACE
****** CHECK NUMERATOR AND DENOMINATOR TO DETERMINE IF EITHER EXCEEDS
****** IN ABSOLUTE VALUE N, THE ORDER OF FAREY FRACTIONS ASSOCIATED
****** WITH THE MODULUS. IF SO, ABORT EXECUTION WITH RESPECTIVE ERRORS
****** #4 AND #5.
         LM       R5,R6,=A(N,NEGN)  LOAD @ OF N AND -N, WHERE N IS
*                                      ORDER-N OF FAREY FRACTIONS.
         CMPLONG  R2,0(R5),QCOPY    IS Q LESS THAN OR EQUAL N?
         BNL      PLESSN            IF SO, CHECK NUMERATOR P.
         LA       R6,5              LOAD ERROR #5
         ST       R6,ERRORFLG          AND STORE IN ERROR FLAG.
         CALL     FERROR,,(ERRORFLG)  ABORT EXECUTION WITH MESSAGE.
PLESSN   TM       0(R8),X'80'       IS REDUCED P POSITIVE?
         BNO      TESTP             IF SO, CHECK POSITIVE P.
         CMPLONG  R2,0(R8),0(R6)    CHECK NEGATIVE P AGAINST -N.
         BNL      COMPRES           IF !P! <= N, COMPUTE RESIDUE.
         B        SETERR            IF NOT, STOP WITH ERROR MESSAGE.
TESTP    CMPLONG  R2,0(R5),0(R8)    COMPARE POSITIVE P TO -N.
         BNL      COMPRES           IF P <= N, COMPUTE RESIDUE.
SETERR   LA       R6,4              LOAD ERROR #4
         ST       R6,ERRORFLG          AND STORE IN ERROR FLAG.
         CALL     FERROR,,(ERRORFLG) ABORT EXECUTION WITH MESSAGE.
         SPACE
******** GET THE RESIDUE REPRESENTATION OF P/Q BY CALLING MLINVGCD
******** TO APPLY THE EXTENDED EUCLIDEAN ALGORITHM OF GREGORY AND
******** KORNERUP.
```

```
COMPRES  CALL     MLINVGCD,(R7,R10,R8,QCOPY,RESFLAG)
         MOVELONG R2,0(R8),QCOPY    MOVE RESIDUE INTO POSITION FOR
*                                      RETURN.
         MINRETURN
         SPACE    2
*        ********   DATA STORAGE FOR OPNRES CSECT      ********
RESFLAG  DC       X'04'             "COMPUTE RESIDUE" FLAG FOR CALL
*                                      TO MLINVGCD
ERRORFLG DS       F                 ERROR FLAG INVOKED IF DENOMINATOR
*                                      IS 0, OR MAX. NEGATIVE INTEGER
*                                      IS COMPLEMENTED, OR REDUCED
*                                      P/Q IS NOT ORDINARY FAREY
*                                      FRACTION.
ADDR0    DC       A(ZERO)           ADDRESS OF K-WORD CONSTANT OF 0
QCOPY    MULTPREC 1                 COPY OF DEN. Q FOR USE IN
*                                      REDUCE, MLINVGCD.
         SPACE    3
*************************************************************************
*                                                                       *
*                        R E S A D D                                    *
*                                                                       *
*        THIS PROCEDURE PERFORMS THE RESIDUE ADDITION !A+B! MOD M.      *
*    THE SUM OF A AND B IS REDUCED MODULO M TO OBTAIN THE RESULT.       *
*                                                                       *
*    UPON ENTRY, R1 POINTS TO AN ARGUMENT LIST CONSISTING OF ...        *
*        R1 --> C K                 -- # OF FULLWORDS IN OPERANDS       *
*               C A  (K WORDS)      -- RESIDUE A                        *
*               C B  (K WORDS)      -- RESIDUE B                        *
*               C M  (K WORDS)      -- A MERSENNE PRIME MODULUS         *
*               C !A+B! MOD M (K WORDS) -- RESIDUE RESULT A+B REDUCED   *
*                                          MODULO M.  CAN BE OPTION-    *
*                                          ALLY STORED IN A OR B.       *
*                                                                       *
*    FATAL ERRORS:                                                      *
*        OVERFLOW IN ADDITION OF RESIDUE A + RESIDUE B --- ERROR #6     *
*                                                                       *
*    SUBROUTINES CALLED:                                                *
*        ADDITION -- ADDS TWO K-WORD INTEGERS (RESULT IN K WORDS)       *
*        FERROR   -- UPON FATAL ERROR, ABORTS EXECUTION WITH ERROR      *
*                    MESSAGE                                            *
*        RESIDUE  -- COMPUTES THE RESIDUE OF A K-WORD INTEGER WITH      *
*                    RESPECT TO A PRIME MODULUS                         *
*************************************************************************
         SPACE
RESADD   MIENTRY  R11,RESADSAV
******** LOAD ARGUMENT ADDRESSES AND COMPUTE LENGTH CODE OF OPERANDS.
         LM       R3,R7,0(R1)       LOAD C K INTO R3;
*                                        C A INTO R4;
*                                        C B INTO R5;
*                                        C MODULUS INTO R6;
*                                        C !A+B! MOD M INTO R7.
         BYTEOFF  BYTE,R2,          GET # BYTES-1 (4K-1) IN OPERANDS.
         SPACE
******** ADD A AND B AND PUT SUM IN SUM2.  IF OVERFLOW OCCURS, CALL
```

```
****** FERROR TO CEASE EXECUTION WITH ERROR #6.
         CALL     ADDITION,,(R3,R4,R5,SUM2,RSADDERR) ADD A AND B.
         ERRORCHK RSADDERR,FERROR,6 CHECK FOR OVERFLOW IN ADDITION.
         SPACE
*********  COMPUTE THE RESIDUE !A+B! MODULO M.
         CALL     RESIDUE,,(R3,SUM1,R6,R7) GET RESIDUE OF A+B.
         SPACE
LV       MIRETURN
         SPACE    2
*        ******       DATA STORAGE FOR RESADD CSECT       ******
RSADDERR DS       F                  IF ADDITION OVERFLOW, SET TO 1
SUM1     MULTPREC 1,F'0'             2K WORD SUM OF A+B FOR CALL
SUM2     MULTPREC 1                     TO RESIDUE.
         SPACE    3
*************************************************************************
*                 ------    R E S S U B    ------
*
*        THIS PROCEDURE PERFORMS THE RESIDUE SUBTRACTION !A-B! MODULO M.
*    THE RESULT IS DETERMINED BY REDUCING (MODULO M) THE SUM OF A AND
*    THE ADDITIVE INVERSE (MODULO M) OF B.
*
*    UPON ENTRY, R1 POINTS TO AN ARGUMENT LIST CONSISTING OF ...
*        R1 - >   @ K                  --- # FULLWORDS IN OPERANDS
*                 @ A  (K WORDS)       --- RESIDUE A
*                 @ B  (K WORDS)       --- RESIDUE B
*                 @ M  (K WORDS)       --- A MERSENNE PRIME MODULUS
*                 @ !A-B! MOD M  (K WORDS) --- RESIDUE RESULT A-B
*                                              REDUCED MODULO M
*                    ---   CAN BE OPTIONALLY STORED IN A OR B
*
*    SUBROUTINES CALLED:
*        SUBTRACT --- PERFORMS SUBTRACTION OF TWO K-WORD INTEGERS
*                     (RESULT IN K WORDS)
*        RESADD   --- PERFORMS RESIDUE ADDITION !A+B! MOD M, WHERE
*                     OPERANDS ARE OF LENGTH K WORDS
*************************************************************************
         SPACE
RESSUB   MIENTRY  R11,RESSUBSV
****** LOAD ADDRESSES OF ARGUMENTS FROM ARGUMENT LIST.
         LM       R3,R7,0(R1)        LOAD @ K INTO R3;
*                                         @ A INTO R4;
*                                         @ B INTO R5;
*                                         @ MODULUS INTO R6;
*                                         @ !A-B!  MOD  M INTO R7.
         SPACE
****** GET ADDITIVE INVERSE OF OPERAND B AND COMPUTE THE RESIDUE
****** SUM OF A + ADDITIVE INVERSE OF B, REDUCED MODULO M.
         CALL     SUBTRACT,,(R3,R6,R5,BADDINV,SUBERROR)
         CALL     RESADD,,(R3,R4,BADDINV,R6,R7)
         MIRETURN
SUBERROR DS       F                  ERROR FLAG FOR CALL TO SUBSTRACT
BADDINV  MULTPREC 1                  TWO'S COMPLEMENT OF B
         SPACE    3
```

```
*********************************************************************
*                  --  --    R E S M L T   ----  --                  *
*                                                                   *
*        THIS PROCEDURE PERFORMS THE RESIDUE MULTIPLICATION !A*B!   *
*    MODULO M.  THE PRODUCT OF A AND B IS REDUCED MODULO M TO OBTAIN *
*    THE RESULT.                                                    *
*                                                                   *
*    UPON ENTRY, R1 POINTS TO AN ARGUMENT LIST CONSISTING OF ...    *
*        R1 --->  C K                  --  # OF FULLWORDS IN OPERANDS*
*                 C A  (K WORDS)       --  RESIDUE A               *
*                 C B  (K WORDS)       --  RESIDUE B               *
*                 C M  (K WORDS)       --  A MERSENNE PRIME MODULUS *
*                 C !A*B! MOD M  (K WORDS) --  RESIDUE PRODUCT REDUCED*
*                                              MODULO M             *
*                    --  --  CAN BE OPTIONALLY STORED IN A OR B     *
*                                                                   *
*    SUBROUTINES CALLED:                                            *
*        MULTIPLY -  MULTIPLIES TWO K-WORD INTEGERS (RESULT IN      *
*                    2K WORDS)                                      *
*        RESIDUE  --  COMPUTES THE RESIDUE OF A K-WORD INTEGER      *
*                     WITH RESPECT TO A PRIME MODULUS               *
*********************************************************************
          SPACE
RESMLT    MIENTRY  R11,RESMLTSV
****** LOAD ARGUMENT ADDRESSES AND COMPUTE LENGTH CODE OF OPERANDS.
          LM       R3,R7,0(R1)        LOAD C K INTO R3;
*                                            C A INTO R4;
*                                            C B INTO R5;
*                                            C MODULUS INTO R6;
*                                            C !A*B! MOD  M INTO R7.
          BYTEOFF  BTYE,R2,           GET # BYTES-1 (4K-1) INTO R2.
          SPACE
******* MULTIPLY A BY B AND REDUCE PRODUCT MODULO M.
          CALL     MULTIPLY,,(R3,R4,R5,ATIMESB,MERROR) MULTIPLY A*B.
          CALL     RESIDUE,,(R3,ATIMESB,R6,R7) GET RESIDUE OF A * B.
          SPACE
LEAV      MIRETURN
          SPACE    2
*         ******      DATA STORAGE FOR RESMLT CSECT      ******
MERROR    DS       F                  ERROR FLAG FOR CALL TO MULTIPLY
ATIMESB   MULTPREC 1                  K WORD STORAGE FOR PRODUCT A*B
ATIMESB2  MULTPREC 1                     OR ADENOM * BDENOM.
          SPACE    3
*********************************************************************
*                  --  --  -   R E S D I V   ----  --                *
*                                                                   *
*        THIS PROCEDURE PERFORMS THE RESIDUE DIVISION !A/B! MODULO M.*
*    THE RESIDUE RESULT IS COMPUTED BY THE EXTENDED EUCLIDEAN       *
*    ALGORITHM OF GREGORY AND KORNERUP (SUBROUTINE "MLINVGCD").     *
*                                                                   *
*    UPON ENTRY, R1 POINTS TO AN ARGUMENT LIST CONSISTING OF ...    *
*        R1 - >   C K                  --  # FULLWORDS IN OPERANDS *
*                 C A  (K WORDS)       --  RESIDUE A               *
```

```
*                C B  (K WORDS)      -- RESIDUE B
*                C M  (K WORDS       -- A MERSENNE PRIME MODULUS
*                C !A/B! MOD M  (K WORDS) -- RESIDUE QUOTIENT REDUCED
*                                            MODULO M.
*                   ---- CAN BE OPTIONALLY STORED IN A OR B
*
*   FATAL ERRORS.
*       RESIDUE OF DIVISOR B IS ZERO -- ERROR #7.
*
*   SUBROUTINES CALLED:
*       FERROR   -- UPON FATAL ERROR, ABORTS EXECUTION WITH ERROR
*                   MESSAGE
*       MLINVGCD -- COMPUTES THE K-WORD RESIDUE !A/B! MODULO M BY
*                   MEANS OF THE EXTENDED EUCLIDEAN ALGORITHM.
*****************************************************************
         SPACE
RESDIV   MIENTRY  R11,RESDIVSV
****** LOAD ARGUMENT ADDRESSES AND COMPUTE LENGTH CODE OF OPERANDS.
         LM       R6,R10,0(R1)       LOAD C K INTO R6;
*                                         C A INTO R7;
*                                         C B INTO R8;
*                                         C MODULUS INTO R9;
*                                         C (A/B) MOD  M INTO R10.
         BYTEOFF  BYTE,R2,           GET # BYTES-1 (4K-1) IN B.
         SPACE
******** CHECK FOR RESIDUE B = 0.  IF IT IS ZERO, ABORT EXECUTION
******** WITH ERROR #7.
         L        R3,ZEROGG          LOAD C OF K-WORD CONSTANT OF 0.
         CMPLONG  R2,0(R8),0(R3)     IS B = 0?
         BNE      MLINV              IF NOT, CALL MLINVGCD.
         CALL     FERROR,,(BZEROERR)  B IS ZERO. ABORT EXECUTION.
         SPACE
******* COMPUTE THE RESIDUE RESULT !A/B! MOD M BY CALLING "MLINVGCD"
******* TO USE THE EXTENDED EUCLIDEAN ALGORITHM OF GREGORY AND
******* KORNERUP.  B IS SENT TO MLINVGCD IN "BTEMP", AND THE RESIDUE
******* RESULT RETURNS IN "BTEMP".
MLINV    MOVELONG R2,BTEMP,0(R8)     COPY B INTO BTEMP FOR MLINVGCD.
         CALL     MLINVGCD,,(R6,R9,R7,BTEMP,RESFLAG2)
         MOVELONG R2,0(R10),BTEMP    COPY RESULT INTO DESIGNATED OUTPUT
*                                       AREA.
         MIRETURN
         SPACE    2
*        ******   DATA STORAGE FOR RESDIV CSECT   ******
RESFLAG2 DC       X'04'              "COMPUTE RESIDUE" FLAG FOR CALL
*                                       TO MLINVGCD
BZEROERR DC       F'7'               ERROR FLAG INVOKED WHEN RESIDUE
*                                       OF B IS 0.
ZEROGG   DC       A(ZERO)            ADDRESS OF K-WORD CONSTANT OF 0
BTEMP    MULTPREC 1                  RESIDUE B/ RESIDUE RESULT
*                                       !A/B! MOD M
         SPACE    3
*****************************************************************
*              ----   R A T M A P   ----
```

```
*
*       THIS PROCEDURE USES THE EXTENDED EUCLIDEAN ALGORITHM OF
*   GREGORY AND KORNERUP TO PERFORM THE INVERSE MAPPING FROM RESIDUE A
*   (MODULO M) TO THE REDUCED RATIONAL EQUIVALENT P/Q.  IF THE
*   INTEGER RESIDUE HAS NO IMAGE AMONG THE ORDER-N FAREY FRACTIONS,
*   THE INVERSE MAPPING FAILS AND EXECUTION CEASES.
*
*   UPON ENTRY, R1 POINTS TO AN ARGUMENT LIST CONSISTING OF ...
*       R1 --->  C K            -- NUMBER OF FULLWORDS IN OPERANDS
*                C A (K WORDS)  -- INTEGER IN RESIDUE SYSTEM
*                C M (K WORDS)  -- A MERSENNE PRIME MODULUS
*                C P (K WORDS)  -- REDUCED NUMERATOR OF RATIONAL
*                                  EQUIVALENT OF "A"
*                C Q (K WORDS)  -- REDUCED DENOMINATOR OF RATIONAL
*                                  EQUIVALENT OF "A"
*       IF SO DESIRED BY THE CALLING PROGRAM, P OR Q CAN BE STORED
*    IN A.
*
*   FATAL ERRORS.
*       RESIDUE A DOES NOT MAP TO AN ORDER-N FAREY FRACTION
*           -- ERROR #8
*              ( P/Q IS ORDER N FAREY FRACTION  <==>
*                !P! <= N AND 0 < !Q! <= N. )
*
*   SUBROUTINES CALLED:
*       MLINVGCD -- COMPUTES RATIONAL EQUIVALENT P/Q (AN ORDER-N
*                   FAREY FRACTION) OF RESIDUE A MOD M
*       FERROR   -- UPON FATAL ERROR, ABORTS EXECUTION WITH ERROR
*                   MESSAGE
*************************************************************************
          SPACE
RATMAP    HIENTRY  R11,RATNSAV
****** COMPUTE LENGTH CODE OF OPERANDS AND LOAD ARGUMENT ADDRESSES.
          BYTEOFF  BYTE,R2,           GET # BYTES-1 (4K-1) IN OPERANDS.
          LM       R5,R9,0(R1)        LOAD C K (# FULLWORDS) INTO R5;
*                                          C A (RESIDUE) INTO R6;
*                                          C MODULUS M INTO R7;
*                                          C P (RATIONAL NUM.) INTO R8;
*                                          C Q (RATIONAL DEN.) INTO R9.
          SPACE
****** CALL MLINVGCD TO DETERMINE THE RATIONAL EQUIVALENT OF
****** RESIDUE A BY MEANS OF THE EXTENDED EUCLIDEAN ALGORITHM
****** OF GREGORY AND KORNERUP.  IF THE INVERSE MAPPING FAILS,
****** (NO ORDER-N FAREY FRACTION IS FOUND), ABORT EXECUTION
****** WITH ERROR #8.
          MOVELONG R2,ATEMP,0(R6)     COPY RESIDUE INTO ATEMP FOR
*                                        MLINVGCD.
          CALL     MLINVGCD,,(R5,R7,ATEMP,R9,RATNFLG) GET RATN. EQUIV.
          CLI      RATNFLG,X'00'      DID INVERSE MAPPING FAIL?
          BNE      STORENUM           IF MAPPING SUCCEEDED, SET NUMER.
          LA       R10,8              RESIDUE DOES NOT MAP TO ORDER-N
          ST       R10,NRERROR           FAREY FRACTION;  SET ERROR FLAG,
          CALL     FERROR,,(NRERROR)     REPORT ERROR, & ABORT EXECUTION.
```

```
STORERUN MOVELONG R2,0(R8),ATEMP     MOVE NUMER. INTO OUTPUT POSITION.
         SPACE
DEPART   DIRETURN                    RETURN RATIONAL EQUIVALENT P/Q.
         SPACE    2
*        *******      DATA STORAGE FOR RATMAP CSECT      *******
MREMROR  DS       F                  ERROR FLAG FOR CALL TO FERROR
RATNFLG  DC       X'0C'              "COMPUTE RATIONAL EQUIVALENT" FLAG
*                                      FOR CALL TO MLINVGCD
ATEMP    MULTPREC 1                  USED TO HOLD RESIDUE A FOR CALL
*                                      TO MLINVGCD
         SPACE    3
*************************************************************************
*                 - - - -   R E D U C E   - - - -
*
*
*         THIS PROCEDURE REDUCES THE RATIONAL NUMBER P/Q BY A FACTOR
*   EQUAL TO THE GREATEST COMMON DIVISOR OF P AND Q.
*
*   UPON ENTRY, R1 POINTS TO AN ARGUMENT LIST CONSISTING OF ...
*       R1 --> C K            --- # FULLWORDS IN P AND Q
*              C P  (K WORDS) --- POSITIVE, ZERO, OR NEGATIVE
*              C Q  (K WORDS) --- POSITIVE ONLY
*              C ERROR FLAG --- SET TO 1 IF P IS MAXIMUM NEGATIVE
*                               INTEGER
*       REDUCED FRACTION P'/Q' IS STORED BACK AT P AND Q.
*
*   SUBROUTINES CALLED:
*       TWOSCOMP -- COMPUTES 2'S COMPLEMENT OF A K-WORD INTEGER
*       MLINVGCD -- DETERMINES THE GREATEST COMMON DIVISOR OF
*                   TWO POSITIVE K-WORD INTEGERS
*       DIVISION -- COMPUTES THE QUOTIENT AND REMAINDER FROM THE
*                   DIVISION OF TWO K-WORD INTEGERS
*************************************************************************
         SPACE
REDUCE   DIENTRY  R11,REDSAVE
******** GET LENGTH CODE OF OPERANDS AND LOAD ARGUMENT ADDRESSES.
         BYTEOFF  BYTE,R2,            GET # BYTES-1 (4K-1) IN P, Q.
         LM       R6,R9,0(R1)         LOAD C K INTO R6,
*                                         C P INTO R7;
*                                         C Q INTO R8;
*                                         C ERROR FLAG INTO R9.
         SPACE
******** IF NUMERATOR P = 0, SET DENOMINATOR Q TO 1 AND RETURN TO
******** CALLER.
         L        R3,=A(ONE111)       LOAD C OF K-WORD CONSTANT OF 1.
         L        R10,=A(ZERO)        LOAD C OF K-WORD CONSTANT OF 0.
         CMPLONG  R2,0(R7),0(R10)     IS P = 0?
         BNE      PSIGN               IF NOT, CHECK FOR P NEGATIVE.
         MOVELONG R2,0(R8),0(R3)      SET Q TO 1 (P = 0).
         B        DONE                RETURN TO CALLER.
         SPACE
******* DETERMINE SIGN OF NUMERATOR.  IF P IS NEGATIVE, COMPLEMENT IT.
PSIGN    SR       R10,R10             INITIALIZE SIGN OF P TO 0.
         TM       0(R7),B'10000000'   CHECK SIGN OF P.
```

```
         BNO       SIZECMP             IF POSITIVE, COMPARE P AND Q.
         CALL      TWOSCOMP,,(R6,R7,R7,R9) GET POSITIVE P.
         ERRORCHK  R9,DONE,            IF P WAS MAXIMUM NEGATIVE
#                                        INTEGER, RETURN TO CALLER.
         LA        R10,1               SET SIGN OF P TO NEGATIVE.
         SPACE
******* COMPARE RELATIVE MAGNITUDES OF P AND Q.  THE SMALLER WILL
******* BE THE SECOND OPERAND FOR THE CALL TO MLINVGCD.  THE
******* LARGER BECOMES THE FIRST OPERAND.
SIZECMP  CMPLONG   R2,0(R7),0(R8)      IS P < Q?
         BE        PEQUALQ             P = Q = GCD(P,Q)
         BL        PLOW
PHIGH    CALL      MLINVGCD,,(R6,R7,R8,GCD,GCDFLAG)
         B         FACTOR              GO FACTOR OUT GCD.
***** P = Q  =>  REDUCED FRACTION = 1/1.
PEQUALQ  MOVELONG  R2,0(R7),0(R3)      SET P TO 1.
         MOVELONG  R2,0(R8),0(R3)      SET Q TO 1.
         B         SGNCHECK            CHECK FOR NEGATIVE SIGN.
PLOW     CALL      MLINVGCD,,(R6,R8,R7,GCD,GCDFLAG)
         SPACE
****** COMPUTE REDUCED P', Q' BY DIVIDING P AND Q BY THE GREATEST
****** COMMON DIVISOR.  REMAINDER (0) GOES INTO WORKAREA.
FACTOR   CMPLONG   R2,0(R9),0(R3)      IS GREATEST COMMON DIVISOR = 1?
         BE        SGNCHECK            IF SO, NEW P,Q  =  OLD P,Q.
         CALL      DIVISION,,(R6,R7,GCD,R7,WORKAREA,DIVERROR)
         CALL      DIVISION,,(R6,R8,GCD,R8,WORKAREA,DIVERROR)
****** RESTORE NEGATIVE NUMERATOR IF NECESSARY.
SGNCHECK LTR       R10,R10             TEST SIGN OF P.
         BZ        DONE                IF POSITIVE, QUIT.
         CALL      TWOSCOMP,,(R6,R7,R7,R9)  CONVERT P' TO NEGATIVE.
DONE     HIRETURN
         SPACE     2
#          ******   DATA STORAGE FOR REDUCE CSECT   ******
DIVERROR DS        F                   ERROR FLAG FOR CALLS TO DIVISION,
#                                        TWOSCOMP
GCDFLAG  DC        X'08'               "COMPUTE G. C. D." FLAG FOR CALL
#                                        TO MLINVGCD.
WORKAREA MULTPREC  1                   REMAINDER (0) IN REDUCING P AND Q
GCD      MULTPREC  1                   GREATEST COMMON DIVISOR OF P, Q
         SPACE     3
*************************************************************************
#                  - - -   R E S I D U E   - - -
#
#         THIS PROCEDURE COMPUTES THE ORDINARY RESIDUE OF A MULTIPLE
#    PRECISION INTEGER A MODULO M.  THE RESIDUE IS A NON-NEGATIVE
#    K-WORD INTEGER BETWEEN 0 AND M - 1:
#        THE POSITIVE REMAINDER FROM THE DIVISION A/M, OR
#        M PLUS THE NEGATIVE REMAINDER FROM THE DIVISION A/M.
#
#    UPON ENTRY, R1 POINTS TO AN ARGUMENT LIST CONSISTING OF ...
#        R1  -->  K              --  # FULLWORDS IN MODULUS AND RESIDUE
#                 A (2K WORDS)   --  INTEGER OF WHICH RESIDUE IS COMPUTED
#                 M (K WORDS     --  A MERSENNE PRIME MODULUS
```

```
*                    C A MODULO M  (K WORDS) --  RESIDUE RESULT
*          IF SO DESIGNATED BY THE CALLING PROGRAM, THE RESIDUE CAN
*     BE STORED IN A.
*
*     SUBROUTINES CALLED:
*          DIVISION --  COMPUTES THE QUOTIENT AND REMAINDER FROM THE
*                       DIVISION OF TWO K-WORD INTEGERS
*          ADDITION --  ADDS TWO K-WORD INTEGERS (RESULT IN K WORDS)
*********************************************************************
          SPACE
RESIDUE   HLENTRY  R11,RESDSAVE
******    COMPUTE LENGTH CODE OF OPERANDS AND LOAD ARGUMENT ADDRESSES.
          BYTEOFF  BYTE,R9,          GET # BYTES-1 (4K-1) IN MODULUS.
          LM       R5,R8,0(R1)       LOAD C # FULLWORDS INTO R5;
*                                          C A INTO R6;
*                                          C MODULUS INTO R7;
*                                          C A MODULO M INTO R8.
          SPACE
******    GET THE REMAINDER (AMODM1,AMODM2) FROM THE DIVISION A/MODULUS.
          L        R10,0(R5)         LOAD K INTO R10,
          AR       R10,R10              DOUBLE,
          ST       R10,TWOK             AND STORE IN TWOK.
          MOVELONG R9,MODULUS2,0(R7) MOVE MODULUS INTO "MODULUS2" FOR
*                                       FOR CALL TO DIVISION.
          CALL     DIVISION,,(TWOK,R6,MODULUS1,QUOTIENT,AMODM1,OVERERR)
          MOVELONG R9,0(R8),AMODM2   STORE TENTATIVE RESIDUE (REMAINDER
*                                       FROM DIVISION)
          SPACE
******    IF THE REMAINDER IS NEGATIVE, ADD THE MODULUS TO GET A
******    RESIDUE BETWEEN 0 AND M - 1.
          TM       0(R6),B'10000000' IS A NEGATIVE?
          BZ       LEAVE             IF A IS POSITIVE, WE QUIT.
          CALL     ADDITION,,(R5,R8,R7,R8,OVERERR)
LEAVE     HLRETURN
          SPACE    2
*         ******   DATA STORAGE FOR RESIDUE CSECT    ******
TWOK      DS       F                 TWO TIMES PRECISION K
OVERERR   DS       F                 OVERFLOW ERROR FLAG FOR
*                                       CALLS TO DIVISION,ADDITION
MODULUS1  MULTPREC 1,F'0'            ZEROED LEFT K-WORDS OF MODULUS
*                                       FOR CALL TO DIVISION
MODULUS2  MULTPREC 1                 MERSENNE PRIME MODULUS M
AMODM1    MULTPREC 1                 2K WORDS TO RECEIVE A MOD M
AMODM2    MULTPREC 1                    FROM DIVISION
QUOTIENT  MULTPREC 2                 2K WORD QUOTIENT FROM A/MODULUS
          SPACE    3
*********************************************************************
*                  ------  S Y M R E S D  ------
*
*       THIS PROCEDURE COMPUTES THE SYMMETRIC RESIDUE OF A MULTIPLE-
*    PRECISION INTEGER A MODULO M.  IF THE ORDINARY RESIDUE <=
*    (MODULUS - 1)/2,
*       THE SYMMETRIC RESIDUE = THE ORDINARY RESIDUE.
```

```
*    OTHERWISE,
*        THE SYMMETRIC RESIDUE = (THE ORDINARY RESIDUE) - (MODULUS).
*
*    UPON ENTRY, R1 POINTS TO AN ARGUMENT LIST CONSISTING OF ...
*        R1 --->  ( K              --- # FULLWORDS IN MODULUS AND RESIDUE
*                 ( A  (2K WORDS) --- INTEGER OF WHICH SYMMETRIC RESIDUE
*                                     IS COMPUTED
*                 ( M  (K WORDS)  --- A MERSENNE PRIME MODULUS
*                 ( SYMMETRIC RESIDUE A MOD M  (K WORDS)
*        IF SO DESIGNATED BY THE CALLING PROGRAM, THE SYMMETRIC
*    RESIDUE CAN BE STORED IN A.
*
*    SUBROUTINES CALLED:
*        RESIDUE  --- COMPUTES THE RESIDUE OF A K-WORD INTEGER WITH
*                     RESPECT TO A PRIME MODULUS
*        SUBTRACT --- PERFORMS SUBTRACTION OF TWO K WORD INTEGERS
*                     (RESULT IN K WORDS)
*********************************************************************
         SPACE
SYMRESD  ENTRY   R11,SYMRESSV
****** COMPUTE LENGTH CODE OF OPERANDS AND LOAD ARGUMENT ADDRESSES.
         BYTEOFF  BYTE,R2,            DETERMINE LENGTH (4K-1) FOR
*                                        COMPARISON.
         LM       R6,R9,0(R1)         LOAD ( # FULLWORDS K INTO R6;
*                                         ( A INTO R7;
*                                         ( MODULUS INTO R8;
*                                         ( SYM. RESD. A MOD M INTO R9.
         SPACE
****** GET THE ORDINARY RESIDUE A MODULO M.
         CALL     RESIDUE,,(R6,R7,R8,R9)
         L        R10,HLFMODAD        LOAD ADDRESS OF (MODULUS-1)/2.
         CMPLONG  R2,0(R9),0(R10)     COMPARE RESIDUE TO (MODULUS-1)/2
         BNH      OUT                 IF <= , WE ARE DONE.
****** SUBTRACT MODULUS TO GET SYMMETRIC RESIDUE <= (MODULUS - 1)/2.
         CALL     SUBTRACT,,(R6,R9,R8,R9,SUBTRERR)
OUT      MRETURN
         SPACE    2
HLFMODAD DC       A(HALFMOD)          ADDRESS OF (MODULUS-1)/2
SUBTRERR DS       F                   ERROR FLAG FOR SUBTRACT
         SPACE    3
*********************************************************************
*                 --- --- M L I N V G C D --- ---
*
*        THIS PROCEDURE USES THE EXTENDED EUCLIDEAN ALGORITHM OF
*    GREGORY AND KORNERUP TO PERFORM ONE OF 4 TASKS SPECIFIED BY
*    THE CALLER:
*    (1)  COMPUTE THE RESIDUE EQUIVALENT MODULO M OF RATIONAL
*         NUMBER P/Q OR PERFORM RESIDUE DIVISION A/B WHERE A AND B
*         ARE RESIDUES MODULO M.
*    (2)  COMPUTE THE GREATEST COMMON DIVISOR OF NON-NEGATIVE
*         INTEGER A AND POSITIVE INTEGER M (M >= A).  THE GREATEST
*         COMMON DIVISOR DOES NOT EXIST IF A IS 0.
*    (3)  PERFORM THE INVERSE MAPPING FROM RESIDUE A (MODULO M) TO
```

```
C          THE RATIONAL EQUIVALENT P/Q.  THE MAPPING FAILS IF AN
C          ORDER-N FAREY FRACTION IS NOT FOUND.
C     (4)  COMPUTE THE MULTIPLICATIVE INVERSE OF INTEGER A MODULO M
C          WHERE 0 < A < M.  IF A AND M ARE NOT RELATIVELY PRIME, THE
C          MULTIPLICATIVE INVERSE DOES NOT EXIST.
C
C           THE CALLER SPECIFIES THE TASK BY MEANS OF A ONE-BYTE
C     "COMPUTE" FLAG.  SEED VALUES OF THE VARIABLES U1, U2, V1, AND
C     V2 ARE LOADED IN ACCORDANCE WITH THE SELECTED TASK, AND THE
C     SAME BASIC ALGORITHM IS PERFORMED IN EACH INSTANCE.  THE
C     DESIRED QUANTITY IS EXTRACTED AT AN APPROPRIATE POINT.  IF
C     THE COMPUTATION FAILS, THE COMPUTE FLAG IS ZEROED TO INFORM
C     THE CALLER THAT THE SPECIFIED VALUE DOES NOT EXIST.
C
C     UPON ENTRY, R1 POINTS TO AN ARGUMENT LIST CONSISTING OF ...
C         C K --  NUMBER OF FULLWORDS IN MULTIPLE PRECISION INTEGERS
C                 FLAG VALUE --> X'04'      X'08'      X'0C'      X'10'
C         ---------------------------------------------------------------
C         C M, REPRESENTING ... MODULUS    LARGER     MODULUS    MODULUS
C                                          INTEGER
C         C A, REPRESENTING ... P OR       SMALLER    RESIDUE    INTEGER A
C                               RESIDUE    INTEGER      A
C                                  A                  (NUM. P
C                                                     RESULT ON
C                                                     RETURN)
C         C Q, REPRESENTING ... Q OR       (GCD       (DEN. Q    (MULT.
C                               RESIDUE    RESULT     RESULT     INVERSE
C                                  B       UPON       UPON       UPON
C                               (RESIDUE   RETURN)    RETURN)    RETURN)
C                                RESULT
C                                UPON
C                                RETURN)
C         ---------------------------------------------------------------
C         C "COMPUTE FLAG" --- A ONE BYTE COMPUTE FLAG.
C                 X'04' --- COMPUTE RESIDUE P/Q OR A/B MODULO M
C                 X'08' --- COMPUTE GREATEST COMMON DIVISOR (A,M)
C                 X'0C' --- COMPUTE RATIONAL EQUIV. P/Q OF RESIDUE A
C                 X'10' --- COMPUTE MULTIPLICATIVE INVERSE OF A MOD M,
C                           WHERE 0 < A < M
C         IF SO SPECIFIED BY THE CALLING PROGRAM, THE MULTIPLICATIVE
C     INVERSE AND THE GREATEST COMMON DIVISOR MAY BE STORED IN "A".
C
C     SUBROUTINES CALLED:
C         SUBTRACT --- PERFORMS SUBTRACTION OF TWO K-WORD INTEGERS
C                      (RESULT IN K WORDS)
C         DIVISION --- COMPUTES THE QUOTIENT AND REMAINDER FROM
C                      THE DIVISION OF TWO K-WORD INTEGERS
C         MULTIPLY --- MULTIPLIES TWO K-WORD INTEGERS (RESULT IN
C                      K WORDS)
C         RESIDUE  --- COMPUTES THE RESIDUE OF A K-WORD INTEGER
C                      WITH RESPECT TO A PRIME MODULUS
C         RESSUB   --- PERFORMS THE RESIDUE SUBTRACTION !A - B! MOD M
C         TWOSCOMP --- COMPUTE ... (COMPLEMENT OF K-WORD INTEGER
```

```
*                    (RESULT IN K WORDS)
*********************************************************************
         SPACE
LLINVGCD REGINFO  R11,LINVSAV
****** GET LENGTH CODE (4K - 1) OF OPERANDS AND LOAD ADDRESSES OF
****** ARGUMENTS FROM ARGUMENT LIST.
         BYTEOFF  BYTE,R9,          GET # BYTES-1 (4K-1) IN OPERANDS.
         LM       R0,R4,0(R1)       LOAD @ M (OPERAND SIZE) INTO R0;
*                                          @ N INTO R1;
*                                          @ A (OR RESULT NUM. P) --> R2;
*                                          @ Q (OR GCD/MLINV/RESIDUE
*                                            RESULT) INTO R3;
*                                          @ ONE BYTE COMPUTE FLAG (R4)--
*                                            X'04': COMPUTE RESIDUE.
*                                            X'08': COMPUTE GCD.
*                                            X'0C': COMPUTE RATN EQUIV.
*                                            X'10': COMPUTE MLV. INV.
*  FLAG IS SET TO X'00' IF ORDER N FAREY FRACTION IS NOT FOUND,
*  OR IF GREATEST COMMON DIVISOR, OR MULTIPLICATIVE INVERSE
*  DOES NOT EXIST RESPECTIVELY IN THE LAST 3 CASES.
         SPACE
****** ESTABLISH ADDRESSABILITY TO OPERANDS FOR INSTANCES IN WHICH
****** OPERANDS ARE VERY LARGE.
         LM       R6,R8,=A(ONE111,V1,V2) LOAD @ OF M-WORD CONSTANT OF
*                                            1 INTO R6;
*                                        @ V1 INTO R7;
*                                        @ V2 INTO R8.
*                                       --- ADDRESSABILITY FOR LARGE
*                                       OPERANDS.
         LR       R5,R1             SAVE @ OF N IN R5.
         SPACE
******* INITIALIZE VARIABLES ACCORDING TO SPECIFIED COMPUTATION.
******* ZERO U1 AND SET U2 = M.  SET V2 = A AND V1= 1 UNLESS THE
******* RESIDUE OF P/Q IS REQUESTED.  IN THE LATTER CASE, SET
******* V1 = P, V2 = Q.
         ZEROLONG R9,U1             ZERO U1.
         L        R15,=A(U2)        LOAD U2 @-- ENSURE ADDRESSABILITY.
         MOVELONG R9,0(R15),0(R1)   INITIALIZE U2 TO M.
         SR       R15,R15           ZERO R15 FOR FLAG INSERTION.
         IC       R15,0(R4)         GET COMPUTE FLAG INTO R4.
BRANCHES B        BRANCHES(R15)     BRANCH ACCORDING TO FLAG OFFSET.
         B        RESIDUAL          COMPUTE RESIDUE OF P/Q.
         B        GCDCHK            COMPUTE GREATEST COMMON DIVISOR.
         B        RATNEQ            COMPUTE RATIONAL EQUIV. OF RESIDUE
         B        V1LOAD1           COMPUTE MULTIPLICATIVE INVERSE
         SPACE
**** RESIDUE OF P/Q IS TO BE COMPUTED.  SET V1 = |P| MOD N AND V2 = Q.
RESIDUAL MOVELONG R9,0(R7),0(R2)    MOVE NUMERATOR P INTO V1.
         L        R15,=A(PREV1)     LOAD @ OF PREV1 INTO R15 ---
*                                     ENSURE ADDRESSABILITY.
         TM       0(R7),X'80'       TEST SIGN OF V1.
         BO       NEG3              IF NEGATIVE, LOAD 1'S INTO PREV1.
         ZEROLONG R9,0(R15)         ZERO PREV1.
```

```
          B         GETRES1             GET RESIDUE OF NUM. P OR RES. A.
NEG3      SETLONG   R9,0(R15),X'FF'     EXTEND NEG. SIGN THROUGH PREV1.
GETRES1   CALL      RESIDUE,,(R0,PREV1,R5,V1) GET RESIDUE OF P OR A.
          MOVELONG  R9,0(R6),0(R3)      MOVE DENOMINATOR Q IN TO V2.
          B         LOADARGS            GO LOAD ARG. TABLES FOR CALLS.
          SPACE
**** RATIONAL EQUIVALENT OF RESIDUE A IS TO BE COMPUTED.  IF RESIDUE
**** IS <= 0 OR >= (MODULUS - 1), DENOMINATOR OF RATIONAL EQUIVALENT
**** IS 1;  NUMERATOR IS EITHER A OR A - MODULUS, RESPECTIVELY.
RATNEQ    LM        R14,R15,=A(1,MODLESS1) LOAD C'S OF 1 AND MODULUS - 1
*                                          INTO R14 AND R15.
          CMPLONG   R9,0(R2),0(R14)     COMPARE RESIDUE TO 1.
          BNH       MOVEOUT             IF RESIDUE < 1, SET Q = 1.
          CMPLONG   R9,0(R2),0(R15)     IS RESIDUE >= MODULUS - 1?
          BL        V1LOAD1             IF NOT, DENOMINATOR IS NOT 1.
          CALL      SUBTRACT,,(R0,R2,R5,R2,ERROR) SUBTRACT MODULUS FROM
*                                          RESIDUE FOR NUMERATOR (NEG.).
MOVEOUT   MOVELONG  R9,0(R3),0(R6)      IF RESIDUE <= 0 OR >= MODULUS - 1,
*                                          DENOMINATOR OF RAT. EQ. IS 1.
          B         GONE                RETURN TO CALLER.
          SPACE
**** SET THE GREATEST COMMON DIVISOR FLAG TO X'00' IF A = 0 AND IF THE
**** "COMPUTE" FLAG SPECIFIES THE GCD --- GREATEST COMMON DIVISOR DOES
**** NOT EXIST.
GCDCHK    CMPLONG   R9,0(R2),U1         IS A = 0?
          BNE       V1LOAD1             IF NOT, SET UP V1, V2.
          MVI       0(R4),X'00'         SET FLAG TO INDICATE GREATEST
*                                          COMMON DIVISOR DOES NOT EXIST
*         B         GONE                   AND RETURN TO CALLER.
          SPACE
**** SET V1 = 1 AND V2 = A --- ALL DEFAULT CASES.
V1LOAD1   MOVELONG  R9,0(R7),0(R5)      INITIALIZE V1 TO 1.
          MOVELONG  R9,0(R8),0(R2)      INITIALIZE V2 TO A.
          SPACE
****** LOAD ARGUMENT TABLES FOR CALLS TO DIVISION, MULTIPLY, SUBTRACT,
****** AND RESSUB WITHIN THE EXTENDED EUCLIDEAN ALGORITHM LOOP.
LOADARGS  LDARGTBL  DIVTBL,(R0,U2,V2,Q,T2,ERROR)
          LDARGTBL  MULTLIST,(R0,V1,Q,PRET1,ERROR)
          LDARGTBL  SUBLIST,(R0,U1,T1,T1,ERROR)
          LDARGTBL  RSSUBLST,(R0,U1,T1,R5,T1)
          LA        R10,1(R9,R9)        GET LENGTH (8K-1) FOR DOUBLE
*                                          OPERAND MOVES IN INVLOOP.
          SPACE     2
********** PERFORM THE EXTENDED EUCLIDEAN ALGORITHM OF GREGORY AND
********** KORNERUP USING THE INITIALIZED VALUES OF U1, U2, V1, V2.
********** LOOP UNTIL V2 <= 1, OR UNTIL A FAREY FRACTION IS FOUND IF
********** WE ARE COMPUTING THE RATIONAL EQUIVALENT TO RESIDUE "A".
INVLOOP   CMPLONG   R9,0(R8),0(R6)      IS V2 <= 1?
          BE        INVSTORE            IF V2 = 1, RECORD INVERSE OR
*                                          RESIDUE OR NO FAREY FRACTION.
          BL        NOINVERS            IF V2 = 0, NO MULTIPLICATIVE
*                                          INVERSE EXISTS.
          SPACE
```

```
******  COMPUTE U2/V2.  SET Q TO QUOTIENT, T2 TO REMAINDER.
******  MULTIPLY V1 BY Q AND PLACE RESULT TEMPORARILY IN T1.
******  SUBTRACT V1*Q FROM U1 AND SET T1 TO RESULTANT VALUE.
          CALL     DIVISION,DIVTBL,  COMPUTE U2/V2.
          CALL     MULTIPLY,MULTLIST,  MULTIPLY V1 * Q.
          CLI      0(R4),X'04'       ARE WE COMPUTING RESIDUE?
          BH       GETT1             IF NOT, GO SUBTRACT FOR T1.
******  IF WE ARE COMPUTING RESIDUE OF P/Q OR A/B, COMPUTE
******  T1 = U1 - (V1*Q) USING RESIDUE ARITHMETIC TO PREVENT POSSIBLE
******  OVERFLOW.
          CALL     RESIDUE,,(R0,PRET1,R5,T1)  GET RESIDUE OF PRODUCT
*                                    TO PREVENT POSSIBLE OVERFLOW.
          CALL     RESSUB,RSSUBLST,  GET RESIDUE DIFFERENCE T1 =
*                                    !U1 - (V1*Q)! MOD M.
          B        NEXTITER          GO SET UP FOR NEXT ITERATION.
GETT1     CALL     SUBTRACT,SUBLIST, GET T1 = U1 - (V1 * Q).
          SPACE
******  IF WE ARE COMPUTING THE RATIONAL EQUIVALENT OF RESIDUE A,
******  DETERMINE IF NEW T1, T2 CONSTITUTE AN ORDER-N FAREY
******  FRACTION.  T2 IS THE POTENTIAL NUMERATOR (POSITIVE);  T1 IS THE
******  POTENTIAL DENOMINATOR, POSSIBLY NEGATIVE.
FAREYCHK  CLI      0(R4),X'0C'       RATIONAL EQUIV. OF RESIDUE DESIRED?
          BNE      NEXTITER          IF NOT, SET UP FOR NEXT ITERATION.
          LM       R14,R15,=A(T1,NEGN) LOAD ADDRESS OF T1 INTO R14;
*                                       @ NEGN (-N OF FAREY FRAC-
*                                             TIONS) INTO R15 --
*                                    ENSURE ADDRESSABILITY.
          TM       0(R14),X'80'      IS T1 NEGATIVE?
          BNO      T1CHECK           IF NOT, CHECK T1?
          CMPLONG  R9,0(R15),0(R14)  IS !T1! <= N?
          BH       NEXTITER          IF NOT, CONTINUE FAREY SEARCH.
          L        R15,=A(N)         LOAD @ OF FAREY N FOR T2CHECK.
          B        T2CHECK           CHECK SIZE OF T2.
*                                    --- ENSURE ADDRESSABILITY.
T1CHECK   L        R15,=A(N)         LOAD @ OF ORDER N OF FAREY FRAC-
*                                       TIONS ASSOCIATED WITH MODULUS.
          CMPLONG  R9,0(R14),0(R15)  IS POSITIVE T1 <= N?
          BH       NEXTITER          IF NOT, CONTINUE FAREY SEARCH.
T2CHECK   L        R14,=A(T2)        LOAD @ OF T2 FOR CMPLONG --
*                                       ENSURE ADDRESSABILITY.
          CMPLONG  R9,0(R14),0(R15)  IS T2 <= N?
          BNH      OUTMAP            IF SO, GO RECORD RATIONAL EQUIV.
          SPACE
********  SET UP U1,U2,V1,V2 FOR NEXT LOOP EXECUTION.
NEXTITER  MOVELONG R10,U1,0(R7)      SET NEW U1,U2 TO OLD V1,V2.
          L        R1,=A(T1)         LOAD @ OF T1 FOR MOVELONG ---
*                                       ENSURE ADDRESSABILITY.
          MOVELONG R10,0(R7),0(R1)   SET NEW V1,V2 TO OLD T1, T2.
          B        INVLOOP           LOOP UNTIL V2 IS 0 OR 1.
          SPACE    2
******** NO MULTIPLICATIVE INVERSE CAN BE COMPUTED.  SET FLAG TO
******** ERROR IF MULTIPLICATIVE INVERSE WAS DESIRED.  IF GCD IS
******** COMPUTED, SET GCD TO NEXT TO LAST VALUE OF V2, WHICH IS NOW
```

```
******* STORED IN U2.
NOINVERS CLI     0(R4),X'08'       IS GCD DESIRED?
         BE      GCDEQU2           IF SO, SET GCD = U2.
         MVI     0(R4),X'00'       ZERO FLAG TO INDICATE NO MULTI-
*                                     PLICATIVE INVERSE EXISTS.
         B       GONE              EXIT.
GCDEQU2  L       R15,=A(U2)        LOAD U2 (--ENSURE ADDRESSABILITY.
         MOVELONG R9,0(R3),0(R15)   COPY LAST V2 (NOW IN U2) INTO GCD.
         B       GONE              RETURN TO CALLER.
         SPACE
******* COMPUTE MULTIPLICATIVE INVERSE OF A MOD M OR RESIDUE OF
******* P/Q BY COMPUTING V1 MOD M FOR FINAL V1.  IF A RATIONAL
******* EQUIVALENT P/Q OF RESIDUE A WAS DESIRED, SET FLAG TO
******* ERROR TO INDICATE NO FAREY FRACTION WAS FOUND.  IF
******* GCD IS TO BE COMPUTED, SET GCD TO 1.
INVSTORE CLI     0(R4),X'0C'       WAS RATIONAL EQUIV. DESIRED?
         BNE     CHKGCD            IF NOT, CHECK GCD FLAG?
         MVI     0(R4),X'00'       ZERO FLAG TO INDICATE NO FAREY
*                                     FRACTION WAS FOUND.
         B       GONE              RETURN TO CALLER.
         SPACE
CHKGCD   CLI     0(R4),X'08'       WAS GCD DESIRED?
         BNE     RESMLI            IF NOT, COMPUTE RESIDUE OR ML. INV.
         MOVELONG R9,0(R3),0(R8)    IF INVERSE EXISTS, GCD IS 1.
         B       GONE              RETURN TO CALLER.
         SPACE
**** GET MULTIPLICATIVE INVERSE OF A MODULO M OR RESIDUE OF P/Q.
RESMLI   L       R8,=A(PREV1)      LOAD @ OF PREV1 INTO R8 --
*                                     ENSURE ADDRESSABILITY.
         TM      0(R7),X'80'       TEST SIGN OF V1.
         BO      NEG               IF NEGATIVE, LOAD 1'S INTO PREV1.
         ZEROLONG R9,0(R8)          ZERO PREV1.
         B       GETAMODM          GET MULTIPLIC. INVERSE OR RESIDUE.
NEG      SETLONG R9,0(R8),X'FF'     EXTEND NEG. SIGN THROUGH PREV1.
GETAMODM CALL    RESIDUE,,(R0,PREV1,R5,R3) GET INVERSE OR RESIDUE.
         B       GONE              RETURN TO CALLER.
         SPACE
****** STORE RATIONAL EQUIVALENT P/Q OF RESIDUE A FOR CALLER.
****** P/Q IS AN ORDER-N FAREY FRACTION.
OUTMAP   LM      R7,R8,=A(T1,T2)   LOAD @ T1 INTO R7;
*                                        @ T2 INTO R8;
*                                     -- ENSURE ADDRESSABILITY FOR
*                                     LARGE OPERANDS.
         TM      0(R7),X'80'       IS T1 NEGATIVE?
         BNO     POSITIVQ          IF NOT, P AND Q ARE POSITIVE.
         CALL    TWOSCOMP,,(R0,T1,R3,ERROR)  SET Q = -T1.
         CALL    TWOSCOMP,,(R0,T2,R2,ERROR) SET P = -T2.
         B       GONE              RETURN TO CALLER.
POSITIVQ MOVELONG R9,0(R3),0(R7)    MOVE T1 INTO DENOMINATOR Q.
         MOVELONG R9,0(R2),0(R8)    MOVE T2 INTO NUMERATOR P.
         SPACE
GONE     MIRETURN
         SPACE   2
```

```
*        ******        DATA STORAGE FOR MLINVGCD CSECT        ******
******  VARIABLES USED IN IMPLEMENTING THE EXTENDED EUCLIDEAN    ***
******  ALGORITHM OF GREGORY AND KORNERUP                        ***
U1        MULTPREC 1                  INITIALIZED TO 1 (OR TO P IF
*                                        RESIDUE OF P/Q IS COMPUTED)
*                                        --- RECEIVES OLD V1
U2        MULTPREC 1                  INITIALIZED TO A (OR TO DEN. Q IF
*                                        RESIDUE OF P/Q IS COMPUTED)
*                                        --- RECEIVES OLD V2
PREV1     MULTPREC 1                  EXTRA K WORDS FOR FINAL CALL TO
*                                        RESIDUE.
V1        MULTPREC 1                  INITIALIZED TO 1 (OR TO P IF
*                                        RESIDUE OF P/Q IS COMPUTED)
*                                        --- RECEIVES OLD T1
V2        MULTPREC 1                  INITIALIZED TO A (OR TO Q IF
*                                        RESIDUE OF P/Q IS COMPUTED)
*                                        --- RECEIVES OLD T2
Q         MULTPREC 1                  INTEGER QUOTIENT OF U2/V2
PRET1     MULTPREC 1                  EXTRA K WORDS FOR CALL TO MULTIPLY
T1        MULTPREC 1                  U1 MINUS PRODUCT OF V1 AND Q
T2        MULTPREC 1                  REMAINDER OF U2/V2
ERROR     DS       F                  OVERFLOW ERROR: DIVISION, MULTIPLY,
*                                                      SUBTRACT
```

APPENDIX F

MULTIPLE PRECISION RATIONAL ARITHMETIC ROUTINES (ASSEMBLER)

```
         PRINT    OFF
         TITLE    '--- --   MULTIPLE PRECISION RATIONAL ARITHMETIC SUBROUT
               TINES  -   - '
*********************************************************************
*                          R A T S E T                              *
*                                                                   *
*       THIS PROCEDURE IS CALLED BY A RATIONAL ARITHMETIC APPLI-    *
*   CATION PROGRAM TO OPEN THE OUTPUT PRINT FILE "FT06F001" AND TO  *
*   VERIFY THAT THE OPERAND PRECISION USED IN THAT PROGRAM AGREES   *
*   WITH THE PRECISION K USED IN ESTABLISHING STORAGE FOR OPERANDS  *
*   IN THE ASSEMBLER SUPPORT ROUTINES.  IF THE MAIN PROGRAM'S PRE-  *
*   CISION DIFFERS FROM ASSEMBLER'S "K" (CALL TO PRECISN MACRO,     *
*   SETUP CSECT), EXECUTION IS ABORTED.                             *
*                                                                   *
*   UPON ENTRY, R1 POINTS TO AN ARGUMENT LIST CONSISTING OF ...     *
*       R1 --> C PRECISION USED IN MAIN APPLICATION PROGRAM         *
*                                                                   *
*   FATAL ERRORS:                                                   *
*       THE PRECISION USED IN THE MAIN RATIONAL ARITHMETIC APPLICA- *
*           TION PROGRAM DIFFERS FROM THE VALUE OF K USED IN ASSEM- *
*           BLER SUPPORT ROUTINES -- ERROR # 62                     *
*                                                                   *
*   SUBROUTINES CALLED:                                             *
*       OPNOUT -- OPENS THE OUTPUT PRINT FILE "FT06F001"            *
*********************************************************************
         SPACE
RATSET   HIENTRY  R11,RATSAVE
         L        R2,0(R1)           LOAD C OF MAIN PGM'S PRECISION.
****** OPEN THE OUTPUT PRINT FILE "FT06F001" FOR THE PRINTING OF
****** MULTIPLE PRECISION RESULTS.
         CALL     OPNOUT             OPEN OUTPUT PRINT FILE FT06F001.
         SPACE
****** COMPARE PRECISION USED IN THE MAIN PROGRAM TO THE VALUE OF
****** "K" EMPLOYED TO ESTABLISH STORAGE IN ASSEMBLER SUPPORT
****** ROUTINES.  IF THEY DIFFER, ABORT EXECUTION WITH FATAL
****** ERROR #62.
         L        R2,0(R2)           LOAD PRECISION FROM MAIN PROGRAM.
         L        R3,KADDR           LOAD C OF PRECSN MACRO'S "K" --
*                                       SETUP CSECT
         L        R3,0(R3)           LOAD "K" USED IN PRECSN MACRO.
         CR       R2,R3              COMPARE MAIN PGM'S PRECISION AND
*                                       "K" USED IN ASSEMBLER ROUTINES.
         BE       EXIT19             IF THEY AGREE, PROCEED WITH COM-
*                                       PUTATIONS.
         CALL     FERROR,,(KPRCERR)  CALL FERROR TO ABORT EXECUTION.
         SPACE
EXIT19   HIRETURN
```

```
         SPACE   2
KADDR    DC      A(K)                ADDRESS OF ASSEMBLER'S "K"
KPRCERR  DC      F'62'               ERROR FLAG INVOKED WHEN APPLICAT-
*                                       ION PROGRAM'S PRECISION AND
*                                       ASSEMBLER'S K DISAGREE.
         SPACE   3
*********************************************************************
*                  - - -   O P E R A T   - - -
*
*
*        THIS PROCEDURE PREPARES AN UNREDUCED RATIONAL OPERAND P/Q FOR
*    USE IN RATIONAL ADDITION, SUBTRACTION, MULTIPLICATION, OR DIVI-
*    SION BY DIVIDING NUMERATOR AND DENOMINATOR BY THE GREATEST COMMON
*    DIVISOR.  THE NUMERATOR P OF THE REDUCED FRACTION IS A POSITIVE
*    OR NEGATIVE INTEGER, AND DENOMINATOR Q IS A POSITIVE INTEGER.
*
*    UPON ENTRY, R1 POINTS TO AN ARGUMENT LIST CONSISTING OF ...
*        R1 --->  C K              --- # FULLWORDS IN P,Q
*                 C P  (K WORDS) --- NUMERATOR
*                 C Q  (K WORDS) --- DENOMINATOR
*
*    FATAL ERRORS:
*        P OR Q IS THE GREATEST NEGATIVE INTEGER THAT CAN BE
*            CONTAINED IN K WORDS--ERRORS #58 AND #59
*        Q IS 0 (DIVISION BY ZERO)--ERROR #57
*
*    SUBROUTINES CALLED.
*        FERROR   --- UPON FATAL ERROR, ABORTS EXECUTION WITH ERROR
*                     MESSAGE
*        TWOSCOMP --- COMPUTES TWO'S COMPLEMENT OF A K-WORD INTEGER
*        REDUCE   --- REDUCES A RATIONAL NUMBER (K-WORD NUMERATOR AND
*                     DENOMINATOR) TO LOWEST TERMS
*********************************************************************
         SPACE
OPRAT    HIENTRY  R11,OPRATSV
******* LOAD ARGUMENT ADDRESSES AND COMPUTE LENGTH CODE OF OPERANDS.
         LM       R6,R8,0(R1)        LOAD C K INTO R6;
*                                         C P INTO R7;
*                                         C Q INTO R8.
         BYTEOFF  BYTE,R2,           GET # BYTES-1 (4K-1) OF OPERANDS.
         SPACE
******* IF DENOMINATOR IS 0, CEASE EXECUTION WITH FATAL ERROR #57.
         L        R3,OPNDO           LOAD C OF K WORDS OF 0'S.
         CHPLONG  R2,0(R8),0(R3)     CHECK FOR ZERO DENOMINATOR.
         BNE      Q2SIGN             IF NOT 0, CHECK SIGN OF Q.
         LA       R5,57              LOAD ERROR #57 INTO R5
         ST       R5,ERRSFLAG           AND STORE IN ERROR FLAG.
         CALL     FERROR,,(ERRSFLAG) CEASE EXECUTION WITH MESSAGE.
         SPACE
******* TEST DENOMINATOR Q.  IF IT IS NEGATIVE, COMPLEMENT
******* NUMERATOR P AND DENOMINATOR Q TO GET DENOMINATOR WITH
******* A POSITIVE VALUE.  IF EITHER P OR Q IS THE MAXIMUM
******* NEGATIVE INTEGER, ABORT EXECUTION WITH ERRORS #58, #59.
Q2SIGN   TM       0(R8),B'10000000'  TEST SIGN OF Q.
```

```
         BNO       REDUCEIT          IF Q IS POSITIVE, REDUCE P/Q.
         CALL      TWOSCOMP,,(R6,R7,R7,ERRSFLAG)  COMPLEMENT NUM.
         ERRORCHK ERRSFLAG,FERROR,58  ABORT UPON 2'S COMPL. OVERFLOW.
         CALL      TWOSCOMP,,(R8,R8,R8,ERRSFLAG)  COMPLEMENT DEN.
         ERRORCHK ERRSFLAG,FERROR,59  ABORT UPON 2'S COMPL. OVERFLOW.
         SPACE
****** REDUCE NUMERATOR AND DENOMINATOR TO LOWEST TERMS AND STORE
****** BACK AT P, Q.
REDUCEIT CALL  REDUCE,,(R6,R7,R8,ERRSFLAG) REDUCE FRACTION P/Q.
         ERRORCHK ERRSFLAG,FERROR,58  ABORT IF P IS MAX. NEG. INTEGER
         MIRETURN
         SPACE    2
OPNDO    DC       A(ZERO)           @ OF K WORD OPERAND OF ZERO
ERRSFLAG DS       F'3'              ERROR FLAG INVOKED IF DENOMINATOR
*                                  IS 0.  ALSO USED IN CALLS TO
*                                  TWOSCOMP.
         SPACE    3
*********************************************************************
*                 .-.-.     R A T A D D     .-.-.
*
*        THIS PROCEDURE PERFORMS THE RATIONAL ADDITION
*
*   (C_NUMER/C_DENOM) = (A_NUMER/A_DENOM) + (B_NUMER/B_DENOM)
*                     = (A_NUMER*B_DENOM) + (B_NUMER*A_DENOM)   ,
*
*   WHERE THE ORIGINAL FRACTIONS A AND B ARE REDUCED AND THE
*   DENOMINATORS ARE POSITIVE.  TO PERFORM THE ADDITION WITH
*   MINIMAL CHANCE OF INTERMEDIATE OVERFLOW DURING COMPUTATION,
*   THE FOLLOWING ALGORITHM IS EMPLOYED TO COMPUTE P/Q = A/B + C/D:
*
*   GCD1 = GCD(B,D)
*   IF GCD1 = 1
*      THEN DO;
*              P = A*D + B*C;
*              Q = B*D;
*           END;
*      ELSE DO;
*              P = A*(D/GCD1) + C*(B/GCD1);
*              GCD2 = GCD(P,GCD1);
*              IF GCD2 EXISTS (P IS NOT 0)
*                 THEN DO;
*                         P = P/GCD2;
*                         Q = (B/GCD1) * (D/GCD2);
*                      END;
*                 ELSE Q = 1;
*           END;
*
*   ----  THE RESULT C_NUMER/C_DENOM IS A REDUCED FRACTION.
*
*   UPON ENTRY, R1 POINTS TO AN ARGUMENT LIST CONSISTING OF ...
*       R1 --->  @ K --- NUMBER OF WORDS IN MULTIPLE PRECISION INTEGERS
*                @ NUMERATOR OF A (K WORDS)
*                @ DENOMINATOR OF A (K WORDS)
```

```
*                C NUMERATOR OF B (K WORDS)
*                C DENOMINATOR OF B (K WORDS)
*                C NUMERATOR OF SUM C (K WORDS)
*                C DENOMINATOR OF SUM C (K WORDS)
*        IF SO DESIGNATED BY THE CALLING PROGRAM. SUM C CAN BE
*        STORED IN LOCATION A OR LOCATION B.
*
*    FATAL ERRORS:
*        OVERFLOW OF INTERMEDIATE 2K PRODUCT OF NUMERATOR OF A AND
*            DENOMINATOR OF B INTO LEFT K WORDS OR INTO SIGN BIT OF
*            K WORDS -- ERROR #50
*        OVERFLOW OF INTERMEDIATE 2K PRODUCT OF NUMERATOR OF B AND
*            DENOMINATOR OF A INTO LEFT K WORDS OR INTO SIGN BIT OF
*            K WORDS -- ERROR #51
*        OVERFLOW IN K-WORD ADDITION OF ABOVE PRODUCTS -- ERROR #52
*        OVERFLOW OF INTERMEDIATE 2K PRODUCT OF DENOMINATORS OF A
*            AND B INTO LEFT K WORDS OR INTO SIGN BIT OF K WORDS
*            -- ERROR #53
*
*    SUBROUTINES CALLED:
*        MLINVGCD -- COMPUTES THE GREATEST COMMON DIVISOR OF TWO
*                    POSITIVE K-WORD INTEGERS (RESULT IN K WORDS)
*        DIVISION -- PERFORMS DIVISION OF 2 K-WORD INTEGERS (QUOTIENT
*                    AND REMAINDER IN K-WORDS)
*        MULTIPLY -- MULTIPLIES TWO K-WORD INTEGERS (RESULT IN
*                    2K WORDS)
*        FERROR   -- UPON FATAL ERROR, ABORTS EXECUTION WITH MESSAGE
*        ADDITION -- ADDS TWO K-WORD INTEGERS (RESULT IN K WORDS)
*        TWOSCOMP -- COMPUTES 2'S COMPLEMENT OF K-WORD INTEGER
*********************************************************************
         SPACE
RATADD   HIENTRY  R11,RATADDSV
******* COMPUTE LENGTH CODE OF OPERANDS AND LOAD ARGUMENT ADDRESSES.
******* LOAD ADDRESSES OF STORAGE TO PREVENT ADDRESSABILITY PROBLEMS
******* FOR LARGE OPERANDS.
         BYTEOFF BYTE,R10           GET 4K - 1 INTO R10.
         L       R0,0(R1)           LOAD C # OPERAND WORDS INTO R0.
         L       R9,4(R1)           LOAD C NUMERATOR OF A INTO R9.
         LM      R14,R15,8(R1)      LOAD C DENOMINATOR OF A INTO R14;
*                                        C NUMERATOR OF B INTO R15;
         LM      R6,R8,16(R1)       LOAD C DENOMINATOR OF B INTO R6;
*                                        C NUMERATOR OF RESULT INTO R7;
*                                        C DENOM. OF RESULT INTO R8.
         LM      R1,R3,=A(ADENTEMP,BNUMTEMP,BDENTEMP)
*                                   LOAD C ADENTEMP INTO R1;
*                                        C BNUMTEMP INTO R2;
*                                        C BDENTEMP INTO R3.
         SPACE
******** MINIMIZE POSSIBILITY OF OVERFLOW IN COMPUTING NUMERATOR
******** AND DENOMINATOR.  REMOVE COMMON FACTORS FROM DENOMINATORS
******** OF A AND B BEFORE COMPUTING TENTATIVE NUMERATOR.
         SPACE
**** COPY DENOMINATORS OF A AND B AND NUMERATOR OF B INTO WORKING
```

```
**** STORAGE AREAS FOR COMPUTATIONS.
         MOVELONG R10,0(R1),0(R14)  MOVE DENOM. OF A INTO ADENTEMP.
         MOVELONG R10,0(R2),0(R15)  COPY NUMERATOR OF B INTO BNUMTEMP.
         MOVELONG R10,0(R3),0(R6)   COPY DENOM. OF B INTO BDENTEMP.
         SPACE
**** SET UP FOR CALLS TO MLINVGCD WITH THE LARGEST OPERAND FIRST.
**** GET GREATEST COMMON DIVISOR OF THE DENOMINATORS OF A AND B.
         L        R2,=A(ONE111)     LOAD R2 WITH @ OF K-WORD CONST 1.
         SR       R4,R4             ZERO FLAG:  DEFAULT -- GCD NOT 1.
         L        R5,=A(ABDENGCD)   LOAD @ OF GCD OF DENOMINATORS.
         CMPLONG  R10,0(R14),0(R6)  COMPARE DENOM. OF A TO DENOM. OF B.
         BE       EQDENS            IF DENOMS. ARE EQUAL, GCD = EITHER.
         BL       ADENLOW           IF A_DENOM LOW, B_DENOM IS 1ST ARG.
ADENHI   CALL     MLINVGCD,,(R0,R14,R6,ABDENGCD,ABGCDFLG)
         B        GCDLOOK           GO TEST GCD(A_DENOM,B_DENOM).
EQDENS   MOVELONG R10,0(R5),0(R6)   SET GCD = A_DENOM = B_DENOM.
         MOVELONG R10,0(R1),0(R2)   SET ADENTEMP = 1.
         MOVELONG R10,0(R3),0(R2)   SET BDENTEMP = 1.
         B        MULTIPLI          GO MULTIPLY FOR TENTATIVE P.
ADENLOW  CALL     MLINVGCD,,(R0,R6,R14,ABDENGCD,ABGCDFLG)
         SPACE
**** IF THEIR GREATEST COMMON DIVISOR IS NOT 1, DIVIDE DENOMINATORS
**** OF A AND B BY THE GCD.
GCDLOOK  CMPLONG  R10,0(R5),0(R2)   COMPARE GCD TO 1.
         BNE      DIVID             FACTOR ABDENGCD FROM DENOMINATORS.
         LA       R4,1              SET "GCD = 1" FLAG.
         B        MULTIPLI          GO FETCH NUMERATOR.
DIVID    CALL     DIVISION,,(R0,ADENTEMP,ABDENGCD,ADENTEMP,CROSPRD2,ERRX
               ATUM)
         CALL     DIVISION,,(R0,BDENTEMP,ABDENGCD,BDENTEMP,CROSPRD2,ERRX
               ATUM)
         SPACE
******** GET PROVISIONAL NUMERATOR OF THE RESULT, GIVEN BY THE SUM
******** OF THE FOLLOWING 2K-WORD PRODUCTS:  (1) NUMERATOR OF A
******** AND ADJUSTED DENOMINATOR OF B AND (2) NUMERATOR OF B AND
******** ADJUSTED DENOMINATOR OF A.  IF EITHER PRODUCT OVERFLOWS
******** INTO THE LEFT K WORDS, ABORT EXECUTION WITH ERROR #50 OR
******** #51.  IF THE SUM DOES NOT FIT IN K-WORDS, CEASE EXECUTION
******** WITH ERROR #52.
         SPACE
*** MULTIPLY NUMERATOR OF A BY ADJUSTED DENOMINATOR OF B.
MULTIPLI CALL     MULTIPLY,,(R0,R9,BDENTEMP,CROSSPRD,ERRATUM)
         OVERCHK  R10,CROSSPRD,50    CHECK FOR PRODUCT OVERFLOW.
*** MULTIPLY NUMERATOR OF B BY ADJUSTED DENOMINATOR OF A.
         CALL     MULTIPLY,,(R0,BNUMTEMP,ADENTEMP,CROSPRD2,ERRATUM)
         L        R14,=A(CROSPRD2)  LOAD @ OF CROSPRD2 FOR
*                                      ADDRESSABILITY OF LARGE
*                                      OPERANDS.
         OVERCHK  R10,0(R14),51     CHECK FOR PRODUCT OVERFLOW.
         LA       R14,1(R10,R14)    GET @ OF 2ND K WORDS OF CROSPRD2.
*** FETCH NUMERATOR OF RESULT.  CHECK FOR OVERFLOW OF SUM.
         LA       R3,CROSSPRD-1(R10) GET @ OF 2ND K WORDS OF CROSSPRD.
         CALL     ADDITION,,(R0,R3,R14,R7,ERRATUM) PUT TENTATIVE NUM.
```

```
*                                          IN DESIGNATED OUTPUT AREA.
         ERRORCHK ERRATUM,FERROR,52 CHECK FOR SUM OVERFLOW.
         SPACE
******** IF THE G. C. D. OF THE DENOMINATORS OF A AND B IS NOT 1,
******** PERFORM COMPUTATIONS SO AS TO MINIMIZE POSSIBILTIY OF OVER-
******** FLOW.  SET UP FOR CALLS TO HLIMVGCD WITH LARGEST OPERAND
******** FIRST.  GET GREATEST COMMON DIVISOR OF PROVISIONAL NUMERATOR
******** AND THE GREATEST COMMON DIVISOR OF THE ORIGINAL DENOMINATORS
******** OF A AND B;  PLACE IN GCD22.
         LTR      R4,R4              ORIGINAL GCD = 1?
         BNE      GETDEN             IF IT IS (R4 = 1), GET DENOM.
         LR       R15,R7             SET R15 TO DEFAULT @ OF TENTATIVE
*                                       NUMERATOR.
         TM       0(R7),X'80'        IS PROVISIONAL NUM. NEGATIVE?
         BNO      COMPARE            IF NOT,SET UP FOR GCD.
         CALL     TWOSCOMP,,(R0,R7,CROSSPRD,ERRATUM)  2'S COMPL. NUM
         LA       R15,CROSSPRD       RESET R15 TO @ OF NUM. COMPLEMENT.
COMPARE  CMPLONG  R10,0(R5),0(R15)   COMPARE GCD OF DENS TO TENTATIVE P
         BNL      TEMPLOW            TENTATIVE P < ABDENGCD.
TEMPEQHI CALL     HLIMVGCD,,(R0,R15,ABDENGCD,GCD22,ABGCDFLG)
         B        REDUCNUM           REDUCE TENTATIVE NUMERATOR.
TEMPLOW  CALL     HLIMVGCD,,(R0,ABDENGCD,R15,GCD22,ABGCDFLG)
         SPACE
****** GET FINAL REDUCED NUMERATOR BY FACTORING GCD22 FROM TENTATIVE
****** NUMERATOR.  FACTOR GCD22 FROM DENOMINATOR OF B.
REDUCNUM CLI      ABGCDFLG,X'00'     IF PROV. NUM. IS 0, GCD DOES NOT
*                                       EXIST.
         BNE      DIV77              IF GCD OK, REDUCE PROVISIONAL NUM.
         MVI      ABGCDFLG,X'08'     RESET FLAG TO COMPUTE GCD.
         MOVELONG R10,0(R8),0(R2)    SET DENOMINATOR OF RESULT TO 1.
         B        HIRET              NO GCD => WE ARE DONE.
DIV77    CALL     DIVISION,,(R0,R7,GCD22,R7,CROSPRD2,ERRATUM)
         CALL     DIVISION,,(R0,R6,GCD22,BDENTEMP,CROSPRD2,ERRATUM)
         SPACE
****** GET REDUCED DENOMINATOR OF RESULT BY MULTIPLYING ADJUSTED
****** DENOMINATORS OF A AND B.  IF PRODUCT OVERFLOWS INTO LEFT
****** K WORDS, ABORT EXECUTION WITH ERROR #53.
GETDEN   CALL     MULTIPLY,,(R0,ADENTEMP,BDENTEMP,CROSSPRD,ERRATUM)
         OVERCHK  R10,CROSSPRD,53    CHECK FOR PRODUCT OVERFLOW.
         MOVELONG R10,0(R8),0(R3)    MOVE DENOMINATOR OF RESULT INTO
*                                       DESIGNATED OUTPUT AREA.
         SPACE
HIRET    HIRETURN
         SPACE    2
*          ******      DATA STORAGE FOR RATADD CSECT      ******
ABGCDFLG DC       X'08'              "COMPUTE GCD" FLAG FOR CALL TO
*                                       HLIMVGCD
ERRATUM  DS       F                  ERROR FLAG FOR CALLS TO MULTIPLY,
*                                       DIVISION
CROSSPRD MULTPREC 2                  STORAGE FOR 2K WORD PRODUCTS
CROSPRD2 MULTPREC 2                  STORAGE FOR 2K WORD PRODUCTS
ABDENGCD MULTPREC 1                  GREATEST COMMON DIVISOR OF
*                                       DENOMINATORS OF A AND B
```

```
GCD22    MULTPREC 1                    GREATEST COMMON DIVISOR OF PRO-
*                                        VISIONAL NUMERATOR AND ADDEND
ADENTEMP MULTPREC 1                    WORKING STORAGE FOR A DENOMINATOR
BNUMTEMP MULTPREC 1                    WORKING STORAGE FOR B NUMERATOR
BDENTEMP MULTPREC 1                    WORKING STORAGE FOR B DENOMINATOR
         SPACE  3
*****************************************************************************
*                                                                           *
*                  ------  R A T S U B  -----                               *
*                                                                           *
*        THIS PROCEDURE PERFORMS THE RATIONAL SUBTRACTION                   *
*                                                                           *
*   (C_NUMER/C_DENOM) = (A_NUMER/A_DENOM) - (B_NUMER/B_DENOM)               *
*                     = (A_NUMER/A_DENOM) + (-B_NUMER/B_DENOM)   ,          *
*                                                                           *
*   WHERE THE ORIGINAL FRACTIONS A AND B ARE REDUCED AND THE                *
*   DENOMINATORS ARE POSITIVE.                                              *
*   ----- THE RESULT C IS ALSO A REDUCED FRACTION.                          *
*                                                                           *
*   UPON ENTRY, R1 POINTS TO AN ARGUMENT LIST CONSISTING OF ...             *
*       R1 --> ; K -- NUMBER OF WORDS IN MULTIPLE PRECISION INTEGERS        *
*              ; NUMERATOR OF A (K WORDS)                                   *
*              ; DENOMINATOR OF A (K WORDS)                                 *
*              ; NUMERATOR OF B (K WORDS)                                   *
*              ; DENOMINATOR OF B (K WORDS)                                 *
*              ; NUMERATOR OF DIFFERENCE C (K WORDS)                        *
*              ; DENOMINATOR OF DIFFERENCE C (K WORDS)                      *
*       IF SO DESIGNATED BY THE CALLING PROGRAM, DIFFERENCE C CAN           *
*       BE STORED IN LOCATION A OR LOCATION B.                              *
*                                                                           *
*   FATAL ERRORS:                                                           *
*       NUMERATOR OF B IS MAXIMUM NEGATIVE INTEGER THAT CAN BE STORED       *
*           IN K BYTES -- ERROR #54                                         *
*                                                                           *
*   SUBROUTINES CALLED.                                                     *
*       TWOSCOMP -- COMPUTES TWO'S COMPLEMENT OF A K-WORD INTEGER           *
*       PERROR --- UPON FATAL ERROR, ABORTS EXECUTION WITH MESSAGE          *
*       RATADD -- ADDS TWO RATIONAL NUMBERS (K-WORD NUMERATORS AND          *
*                 DENOMINATORS)                                             *
*****************************************************************************
         SPACE
RATSUB   HIENTRY  R11,RATSUBSV
****** LOAD ARGUMENT ADDRESSES.
         LM       R2,R6,0(R1)          LOAD ; # OPERAND WORDS INTO R2;
*                                           ; NUMERATOR OF A INTO R3;
*                                           ; DENOMINATOR OF A INTO R4,
*                                           ; NUMERATOR OF B INTO R5;
*                                           ; DENOMINATOR OF B INTO R6,
*                                           ; NUMERATOR OF RESULT INTO R7;
*                                           ; DENOM. OF RESULT INTO R8.
         SPACE
****** COMPLEMENT NUMERATOR OF SUBTRAHEND B AND ADD FRACTION (-B) TO A.
****** IF NUMERATOR OF B IS MAXIMUM NEGATIVE INTEGER, ABORT EXECUTION
****** WITH ERROR #54.
```

```
         CALL     TWOSCOMP,(R2,R5,ENUMCHP,NTSUBFLG)
         ERRONCHK NTSUBFLG,FERROR,54 CHECK FOR OVERFLOW OF NUMERATOR
*                                    OF D AND ABORT IF ERROR FOUND.
         CALL     RATADD,(R4,R3,R7,ENUMCHP,R6,R7,R8)
         MEXRETURN
         SPACE
NTSUBFLG DS       F                  OVERFLOW ERROR FLAG FOR CALL TO
*                                      TWOSCOMP
ENUMCHP  MULTPREC 1                  TWO'S COMPLEMENT OF NUMERATOR OF N
         SPACE    3
*********************************************************************
*                 - - -   R A T M L T   - - -
*
*         THIS PROCEDURE PERFORMS THE RATIONAL MULTIPLICATION
*
*   (C_NUMER/C_DENOM) = (A_NUMER/A_DENOM) * (B_NUMER/B_DENOM)
*                     = (A_NUMER*B_NUMER) / (A_DENOM*B_DENOM)   ,
*
*   WHERE THE ORIGINAL FRACTIONS A AND B ARE REDUCED AND THE
*   DENOMINATORS ARE POSITIVE.  TO MINIMIZE THE POSSIBILITY OF
*   OVERFLOW IN THE MULTIPLICATION, A_NUMER AND B_DENOM ARE FIRST
*   REDUCED BY THE GCD(A_NUMER, B_DENOM), AND B_NUMER AND A_DENOM
*   ARE REDUCED BY GCD(B_NUMER, A_DENOM).
*   - - - - THE RESULT C IS ALSO A REDUCED FRACTION.
*
*   UPON ENTRY, R1 POINTS TO AN ARGUMENT LIST CONSISTING OF ...
*       R1 - >  C K -- NUMBER OF WORDS IN MULTIPLE PRECISION INTEGERS
*               C NUMERATOR OF A (K WORDS)
*               C DENOMINATOR OF A (K WORDS)
*               C NUMERATOR OF B (K WORDS)
*               C DENOMINATOR OF B (K WORDS)
*               C NUMERATOR OF PRODUCT C (K WORDS)
*               C DENOMINATOR OF PRODUCT C (K WORDS)
*       IF SO DESIRED BY THE CALLING PROGRAM, PRODUCT C CAN BE
*       STORED IN LOCATION A OR LOCATION B.
*
*   FATAL ERRORS.
*       OVERFLOW OF INTERMEDIATE 2K PRODUCT OF NUMERATORS INTO
*           LEFT K WORDS OR INTO THE SIGN BIT OF K WORDS -  ERROR #55
*       OVERFLOW OF INTERMEDIATE 2K PRODUCT OF DENOMINATORS INTO
*           LEFT K WORDS OR INTO THE SIGN BIT OF K WORDS -  ERROR #56
*
*   SUBROUTINES CALLED.
*       REDUCE -- REDUCES A RATIONAL NUMBER (K-WORD NUMERATOR AND
*                 DENOMINATOR) TO LOWEST TERMS
*       MULTIPLY -- MULTIPLIES TWO K-WORD INTEGERS (RESULT IS 2K
*                   WORDS)
*       FERROR -- UPON FATAL ERROR, ABORTS EXECUTION WITH MESSAGE
*********************************************************************
         SPACE
RATMLT   HIENTRY  R11,RATMLTSV
******** LOAD ARGUMENT ADDRESSES AND COMPUTE LENGTH CODE OF OPERANDS.
         LH       R2,R8,0(R1)        LOAD C # OPERAND WORDS INTO R2;
```

```
*                                          C NUMERATOR OF A INTO R3;
*                                          C DENOMINATOR OF A INTO R4,
*                                          C NUMERATOR OF B INTO R5;
*                                          C DENOMINATOR OF B INTO R6,
*                                          C NUMERATOR OF RESULT INTO R7;
*                                          C DENOM. OF RESULT INTO R8.
         BYTEOFF  BYTE,R9,          GET # BYTES - 1 IN K WORDS ---> R9
         SPACE
******** MOVE NUMERATORS AND DENOMINATORS OF A AND B INTO TEMPORARY
******** STORAGE.  MINIMIZE POSSIBILITY OF OVERFLOW IN MULTIPLICATION
******** BY REDUCING A_NUMER AND B_DENOM BY THE GCD(A_NUMER,B_DENOM)
******** AND B_NUMER AND A_DENOM BY THE GCD(B_NUMER,ADENOM).
         LM       R14,R15,=A(ANUMPRIM,ADENPRIM) LOAD C'S OF A'S TEMPO-
*                                          RARY STORAGE -- ENSURE ADDRESSING.
         LR       R10,R4                GET INTO R10 C OF A_DENOM FOR
*                                          MOVELONG.
         MOVELONG R9,0(R14),0(R3)       MOVE A_NUMER INTO WORKING STORAGE.
         MOVELONG R9,0(R15),0(R10)      MOVE A_DENOM INTO WORKING STORAGE.
         LM       R14,R15,=A(BNUMPRIM,BDENPRIM) LOAD C'S OF B'S TEMPO-
*                                          RARY STORAGE-- ENSURE ADDRESSING.
         LR       R10,R5                GET INTO R10 C OF B_NUMER FOR
*                                          MOVELONG.
         MOVELONG R9,0(R14),0(R10)      MOVE B_NUMER INTO WORKING STORAGE.
         MOVELONG R9,0(R15),0(R6)       MOVE B_DENOM INTO WORKING STORAGE.
         CALL     REDUCE,,(R2,ANUMPRIM,BDENPRIM,RTNULFLG)
         CALL     REDUCE,,(R2,BNUMPRIM,ADENPRIM,RTNULFLG)
         SPACE
******** GET NUMERATOR OF RESULT BY MULTIPLYING ADJUSTED NUMERATORS
******** OF FRACTIONS A AND B.  DENOMINATOR OF RESULT IS GIVEN BY THE
******** PRODUCT OF THE ADJUSTED DENOMINATORS OF A AND B.  IF EITHER
******** PRODUCT OVERFLOWS INTO LEFT K WORDS, ABORT EXECUTION WITH
******** RESPECTIVE ERRORS #55 OR #56.
         SPACE
***** GET NUMERATOR AND MOVE K-WORD RESULT INTO OUTPUT AREA FOR CALLER.
         CALL     MULTIPLY,,(R2,ANUMPRIM,BNUMPRIM,RATPRODS,RTNULFLG)
         OVERCHK  R9,RATPRODS,55        CHECK FOR PRODUCT OVERFLOW.
         LA       R10,RATPRODS+1(R9) GET C OF 2ND K WORDS OF RATPRODS.
         MOVELONG R9,0(R7),0(R10)       MOVE RIGHT K WORDS INTO NUMERATOR.
         SPACE
***** GET DENOMINATOR AND MOVE K-WORD RESULT INTO OUTPUT FOR CALLER.
         CALL     MULTIPLY,,(R2,ADENPRIM,BDENPRIM,RATPRODS,RTNULFLG)
         OVERCHK  R9,RATPRODS,56        CHECK FOR PRODUCT OVERFLOW.
         MOVELONG R9,0(R8),0(R10)       MOVE RIGHT K WORDS INTO DENOMINA-
*                                          TOR.
         HIRETURN
         SPACE    2
*        ******   DATA STORAGE FOR RATMLT CSECT   ******
RTNULFLG DS       F                     ERROR FLAG FOR CALLS TO MULTIPLY
RATPRODS MULTPREC 2                     2K WORDS FOR PRODUCTS OF NUMERA-
*                                          TORS, DENOMINATORS
ANUMPRIM MULTPREC 1                     WORKING STORAGE FOR NUMERATOR OF A
ADENPRIM MULTPREC 1                     WORKING STORAGE FOR DENOM. OF A
BNUMPRIM MULTPREC 1                     WORKING STORAGE FOR NUMERATOR OF B
```

```
EDELPATH MULTPREC 1                    WORKING STORAGE FOR DENOM. OF D.
         SPACE   3
*********************************************************************
*                  - - -   R A T D I V   - - - -
*
*        THIS PROCEDURE PERFORMS THE RATIONAL DIVISION
*
*   (C_NUMER/C_DENOM) = (A_NUMER/A_DENOM) / (D_NUMER/D_DENOM)
*                     = (A_NUMER/A_DENOM) * (D_DENOM/D_NUMER)   ,
*
*   WHERE THE ORIGINAL FRACTIONS A AND B ARE REDUCED AND THE
*   DENOMINATORS ARE POSITIVE.  EXECUTION ABORTS IF NUMERATOR
*   OF DIVISOR D IS 0.
*   - - -  THE RESULT C IS REDUCED TO LOWEST TERMS.
*
*   UPON ENTRY, R1 POINTS TO AN ARGUMENT LIST CONSISTING OF ...
*        R1 - >   F K -- NUMBER OF WORDS IN MULTIPLE PRECISION INTEGERS
*                 F NUMERATOR OF A (K WORDS)
*                 F DENOMINATOR OF A (K WORDS)
*                 F NUMERATOR OF B (K WORDS)
*                 F DENOMINATOR OF B (K WORDS)
*                 F NUMERATOR OF QUOTIENT C (K WORDS)
*                 F DENOMINATOR OF QUOTIENT C (K WORDS)
*        IF SO DESIRED BY THE CALLING PROGRAM, QUOTIENT C CAN BE
*        STORED IN LOCATION A OR LOCATION B.
*
*   FATAL ERRORS:
*        NUMERATOR OF DIVISOR IS 0 ( DIVISION BY 0) - - ERROR #60
*        NUMERATOR OF DIVISOR IS MAXIMUM NEGATIVE INTEGER IN
*            K WORDS - - ERROR #61
*
*   SUBROUTINES CALLED:
*        FERROR   - - UPON FATAL ERROR, ABORTS EXECUTION WITH MESSAGE
*        TWOSCOMP - - COMPUTES 2'S COMPLEMENT OF A K-WORD INTEGER
*                     (RESULT IN K WORDS)
*        RATMLT   - - MULTIPLIES TWO RATIONAL NUMBERS (K-WORD NUMERA-
*                     TORS AND DENOMINATORS)
*********************************************************************
         SPACE
RATDIV   HIENTRY  R11,RATDIVSV
******** LOAD ARGUMENT ADDRESSES AND COMPUTE LENGTH CODE OF OPERANDS.
         LM       R2,R8,0(R1)        LOAD F # OPERAND WORDS INTO R2;
*                                            F NUMERATOR OF A INTO R3;
*                                            F DENOMINATOR OF A INTO R4;
*                                            F NUMERATOR OF B INTO R5;
*                                            F DENOMINATOR OF B INTO R6;
*                                            F NUMERATOR OF RESULT INTO R7;
*                                            F DENOM. OF RESULT INTO R8.
         BYTEOFF  BYTE,R9,           GET # BYTES - 1 IN OPERANDS - - > R9.
         SPACE
******** TEST FOR DIVISION BY 0 (NUMERATOR OF OPERAND B = 0).  IF
******** DIVISION BY 0 IS ATTEMPTED, ABORT EXECUTION WITH ERROR #60.
         L        R10,=A(ZERO)       LOAD F OF K-WORD CONSTANT = 1.
```

```
         CLFLONG  R9,0(R5),0(R10)   IS NUMERATOR OF B = 0?
         BNE      CHKSIGN           IF NOT, TEST SIGN.
         LA       R10,60            LOAD ERROR #60,
         ST       R10,ERRORDIV         STORE IN ERROR FLAG,
         CALL     FERROR,,(ERRORDIV)   AND ABORT EXECUTION WITH MSG.
         SPACE
****** IF B IS NEGATIVE, COMPLEMENT WORKING NUMERATOR AND DENOMINATOR
****** TO GET POSITIVE DENOMINATOR.  MULTIPLY A BY B', INVERSE OF B,
****** AND MOVE QUOTIENT INTO OUTPUT FOR RETURN TO CALLER.  IF NUMERA-
****** TOR OF B IS THE LARGEST NEGATIVE INTEGER THAT CAN BE STORED IN
****** K WORDS, ABORT EXECUTION WITH ERROR #61.
CHKSIGN  TM       0(R5),X'80'       IS B NEGATIVE?
         BNO      POSCALL           IF NOT, CALL RATMLT WITH B INVERSE
         CALL     TWOSCOMP,,(R2,R5,NUMERA,ERRORDIV) GET 2'S COMP, NUM
         ERRORCHK ERRORDIV,FERROR,61 IF NUM. OF B IS MAX. NEG. INT.,
*                                      ABORT EXEC. WITH ERROR #61.
         CALL     TWOSCOMP,,(R2,R6,DENOMIN,ERRORDIV) GET 2'S COMP, DEN
         CALL     RATMLT,,(R2,R3,R4,DENOMIN,NUMERA,NUMERA,DENOMIN)
         B        SETRSLT           MOVE RESULTS INTO CALLER'S AREA.
POSCALL  CALL     RATMLT,,(R2,R3,R4,R6,R5,NUMERA,DENOMIN)
SETRSLT  MOVELONG R9,0(R7),NUMERA   MOVE NUMERATOR FOR OUTPUT.
         MOVELONG R9,0(R8),DENOMIN  MOVE DENOMINATOR FOR OUTPUT.
         HERETURN
         SPACE    2
*        ******   DATA STORAGE FOR RESDIV CSECT    ******
NUMERA   MULTPREC 1                 NUMERATOR OF RESULT
DENOMIN  MULTPREC 1                 DENOMINATOR OF RESULT
ERRORDIV DS       F                 ERROR FLAG FOR CALL TO TWOSCOMP,
*                                      FERROR
         SPACE    3
```

APPENDIX G

OUTPUT AND CONVERSION SUBROUTINES (ASSEMBLER)

```
         PRINT    OFF
         PRINT    NOGEN
         TITLE    '--- SETUP CSECT - INITIALIZE PRECISION K, MODULUS, HALF
                  MOD, AND N ---'
*********************************************************************
*                  ---------   S E T U P   ---------
*
*       THIS IS AN INITIALIZATION SUBROUTINE CALLED BY ANY RESIDUE
*   ARITHMETIC APPLICATION PROGRAM (POSSIBLY HIGHER LEVEL LANGUAGE)
*   TO INITIALIZE THE MODULUS, "HALFMOD" ((MODULUS-1)/2), AND THE
*   ORDER N OF FAREY FRACTIONS ASSOCIATED WITH THE MODULUS.  THE
*   SUBROUTINE COMPARES THE CALLER'S PRECISION (THE NUMBER OF FULL-
*   WORDS IN MULTIPLE PRECISION OPERANDS) TO K, WHICH IS INITIAL-
*   IZED LOCALLY BY A CALL TO THE "PRECISN" MACRO.  IF THEY DISAGREE,
*   EXECUTION CEASES WITH AN ERROR MESSAGE.  OTHERWISE, A TABLE OF
*   PRECISION VALUES IS SCANNED.  IF THE SUPPLIED VALUE IS NOT FOUND
*   IN THE TABLE, EXECUTION TERMINATES WITH AN ERROR.
*       WHEN K IS LOCATED IN THE TABLE, THE POWER OF 2 OF THE COR-
*   RESPONDING LARGEST MERSENNE PRIME IS EXTRACTED FROM THE TABLE.
*   THIS POWER IS USED TO INITIALIZE THE MODULUS, "HALFMOD"
*   ((MODULUS-1)/2), AND N, THE LARGEST ORDER OF FAREY FRACTIONS FOR
*   WHICH THERE IS A UNIQUE  RESIDUE <--> RATIONAL MAPPING.  THE
*   THE NEGATIVE OF N ("NEGN") AND THE DIFFERENCE OF THE MODULUS AND
*   N ("MODLESSN") ARE ALSO COMPUTED.
*
*        CSECT "OPNOUT" IS CALLED TO OPEN THE OUTPUT PRINT FILE
*   "FT06F001."
*
*   UPON ENTRY, R1 POINTS TO AN ARGUMENT LIST CONSISTING OF ...
*       R1 --->  0 OF PRECISION ESTABLISHED IN MAIN PROGRAM (ONE WORD)
*                0 MODULUS (K WORDS) --- A MERSENNE PRIME TO BE COMPUTED
*
*   FATAL ERRORS:
*       CALLER'S PRECISION AND SETUP'S "K" DIFFER -- ERROR #9
*       VALUE OF K IS NOT IN TABLE OF MERSENNE MODULI --- ERROR #10
*
*   SUBROUTINES CALLED:
*       OPNOUT   --- OPENS THE OUTPUT PRINT FILE "FT06F001"
*       FERROR   --- UPON FATAL ERROR, ABORTS EXECUTION WITH ERROR
*                    MESSAGE
*       POWRMULT --- DIVIDES (MULTIPLIES) A K-WORD INTEGER BY A SPECI-
*                    FIED POWER OF 2 (RESULT IN K WORDS)
*       TWOSCOMP --- COMPUTES 2'S COMPLEMENT OF K-WORD
*                    INTEGER (RESULT IN K WORDS)
*       SUBTRACT --- COMPUTES DIFFERENCE OF TWO K-WORD INTEGERS
*                    (RESULT IN K WORDS)
*********************************************************************
```

```
         SPACE
SETUP    MIENTRY  R11,SETSAVE
         LM       R2,R3,0(R1)          LOAD C CALLER'S PRECISION INTO R2;
*                                        C  MODULUS INTO R3.
         SPACE
****** OPEN THE QSAM OUTPUT PRINT FILE.
         CALL     OPNOUT               OPEN OUTPUT PRINT FILE.
         SPACE
********** COMPARE CALLER'S PRECISION TO VALUE OF K USED IN
********** GENERATING MULTIPLE PRECISION STORAGE.  IF THEY DIFFER,
********** ABORT EXECUTION.
         L        R8,0(R2)             LOAD CALLER'S PRECISION.
         C        R8,K                 COMPARE CALLER'S "PRECISION" TO
*                                        ASSEMBLER "K" OF THIS CSECT.
         BE       GETMODU              IF EQUAL, PROCEED.
         LA       R6,9                 LOAD ERROR # 9
         ST       R6,ERFLAG               AND STORE IN ERROR FLAG.
         CALL     FERROR,,(ERFLAG)     ABORT EXECUTION.
         SPACE
****** SET UP FOR SEARCH OF MODULUS TABLE.
GETMODU  LR       R7,R8                COPY K INTO R7.
         SLL      R8,5                 MULTIPLY K BY 32 FOR TOTAL BITS.
         LA       R4,4                 LOAD INCREMENT FOR SCAN OF
*                                        MODULUS TABLE.
         LA       R5,MODTABL+48        LOAD LIMIT FOR SCAN LOOP.
         LA       R6,MODTABL           LOAD INDEX REGISTER FOR MODTABL
*                                        SCAN.
         SPACE
********** PERFORM LINEAR SEARCH OF TABLE FOR SPECIFIED PRECISION.
********** IF IT IS NOT FOUND, ABORT EXECUTION WITH ERROR #10.
MODLOOP  CH       R7,0(R6)             COMPARE K TO TABLE ENTRY.
         BE       GETMOD               IF K IS FOUND, GET MODULUS.
         BXLE     R6,R4,MODLOOP        CONTINUE TABLE SEARCH FOR K.
*
         LA       R6,10                LOAD AND STORE FATAL ERROR #10:
         ST       R6,ERFLAG               K IS NOT IN THE TABLE.
         CALL     FERROR,,(ERFLAG)     ABORT PROGRAM EXECUTION.
         SPACE
********** EXTRACT THE MERSENNE POWER OF 2 FROM THE TABLE AND USE
********** IT TO SHIFT INITIAL VALUES OF THE MODULUS, HALFMOD
********** ((MODULUS - 1) / 2), AND N TO THEIR PROPER VALUES.
         SPACE
**** GET MODULUS BY SHIFTING K-WORDS OF 1'S TO RIGHT.
GETMOD   LH       R9,2(R6)             LOAD # BITS IN POWER OF 2.
         SR       R8,R9                SUBTRACT POWER OF 2 IN MODULUS
*                                        FOR # LEFT 0'S IN MODULUS.
         LCR      R10,R8               GET NEGATIVE SHIFT VALUE.
         ST       R10,SHIFTPOW         STORE SHIFT (DIVIDE) FOR POWRMULT.
         SLL      R7,2                 MULTIPLY K BY 4
         BCTR     R7,R0                   AND SUBTRACT 1 FOR 4K - 1.
         SETLONG  R7,0(R3),X'FF'       SET MODULUS BITS TO ALL 1'S.
         CALL     POWRMULT,,(K,SHIFTPOW,R3,R3) SHIFT RT. FOR MODULUS.
**** GET HALFMOD BY SHIFTING K-WORDS OF 1'S TO RIGHT.
```

```
          BCTR     R10,R0              SUBTRACT 1 FROM NEGATIVE SHIFT.
          ST       R10,SHIFTPOW        STORE NEGATIVE SHIFT FOR POWRMULT.
          CALL     POWRMULT,,(K,SHIFTPOW,HALFMOD,HALFMOD) GET HALFMOD.
***** GET N BY SHIFTING LEFT A K-WORD CONSTANT OF 1.
          BCTR     R9,R0               SUBTRACT 1 FROM MODULUS POWER OF 2
          SRL      R9,1                DIVIDE BY 2 FOR POWER OF 2 FOR N.
          ST       R9,SHIFTPOW         STORE LEFT SHIFT TO GET N.
          CALL     POWRMULT,,(K,SHIFTPOW,N,N) SHIFT LEFT FOR N.
          SPACE
****** COMPUTE TWO'S COMPLEMENT OF N ("NEGN") AND THE DIFFERENCE OF
****** THE MODULUS AND N ("MODLESSN" = MODULUS - N).
          CALL     TWOSCOMP,,(K,N,NEGN,ERFLAG) COMPUTE -N.
          CALL     SUBTRACT,,(K,R3,N,MODLESSN,ERFLAG) GET MOD. - N
          SPACE
EXIT10    MIRETURN
          SPACE    2
*         ******   DATA STORAGE FOR SETUP CSECT     ******
SHIFTPOW  DS       F                   SHIFT FACTOR FOR CALLS TO POWRMULT
ERFLAG    DS       F                   ERROR FLAG FOR CALL TO FERROR
K         PRECISN  303                 ESTABLISH K = 303 WORDS IN
*                                        MULTIPLE PRECISION OPERANDS.
          SPACE
***** TABLE OF LEGAL RESIDUE ARITHMETIC PRECISIONS K AND
***** CORRESPONDING MERSENNE PRIME POWERS OF 2
MODTABL   DS       0F                  TABLE OF LEGAL VALUES OF THE
          DC       H'2',H'61'          PRECISION K IN RESIDUE ARITH-
          DC       H'3',H'89'          METIC AND CORRESPONDING VALUES
          DC       H'4',H'107'         OF A, THE INTEGER POWER OF 2 IN
          DC       H'5',H'127'         THE EMPLOYED MERSENNE PRIME:
          DC       H'17',H'521'        MODULUS = (2 ** A) - 1
          DC       H'20',H'607'
          DC       H'41',H'1279'
          DC       H'69',H'2203'
          DC       H'72',H'2281'
          DC       H'101',H'3217'
          DC       H'133',H'4253'
          DC       H'139',H'4423'
          DC       H'303',H'9689'
          SPACE
HALFMOD   MULTPREC 1,F'-1'             (MODULUS - 1) / 2--VALUE TO SHIFT
N         MULTPREC 1,F'0',1,F'1'       ORDER N OF ASSOCIATED FAREY
*                                        FRACTIONS -- VALUE TO SHIFT
NEGN      MULTPREC 1                   2'S COMPLEMENT OF N
MODLESSN  MULTPREC 1                   MODULUS - N
***** K-WORD CONSTANTS FOR USE BY SUPPORT ROUTINES
MINUS1    MULTPREC 1,F'-1'             K WORD OPERAND OF -1
ZERO      MULTPREC 1,F'0'              K WORD OPERAND OF 0
ONE111    MULTPREC 1,F'0',1,F'1'       K WORD OPERAND OF +1
          SPACE    3
          TITLE    '---- ROUTINES TO CONVERT TO DECIMAL CHARACTERS AND PX
                RINT K-WORD 2'S COMPLEMENT RATIONAL NUMBERS ------'
*************************************************************************
*                 ----    O P N O U T    ----
```

```
*
*         THIS SUBROUTINE IS CALLED BY AN INTEGER OR RATIONAL ARITH-
*    METIC MAIN PROGRAM TO OPEN THE OUTPUT PRINT FILE "FT06F001."  IT
*    IS ALSO CALLED BY RESIDUE ARITHMETIC SUBROUTINE "SETUP."
*
*    PARAMETERS:   NONE
*********************************************************************
          SPACE
OPNOUT    LOENTRY
          OPEN      (PRINTOUT,(OUTPUT)) OPEN OUTPUT PRINT FILE.
          LORETURN
          SPACE     2
PRINTOUT  DCB       DSORG=PS,DDNAME=FT06F001,RECFM=FA,LRECL=133,BLKSIZE=1X
               33,MACRF=PM
RECORD    DS        CL133               RECORD FOR OUTPUT TO PRINT FILE
          DS        CL123               IGNORED BYTES OF A "PUTOUT" MACRO
*                                        REQUEST OF MORE THAN 133 BYTES
          SPACE     3
*********************************************************************
*                 - - -    C L S O U T    - - -
*
*         THIS SUBROUTINE IS CALLED BY ANY MAIN PROGRAM TO CLOSE THE
*    OUTPUT PRINT FILE "FT06F001" BEFORE CEASING EXECUTION.
*
*    PARAMETERS:  NONE
*********************************************************************
          SPACE
CLSOUT    LOENTRY
          L         R10,=A(PRINTOUT)  ESTABLISH ADDRESSABILITY TO
          USING     PRINTOUT,R10         PRINTOUT DCB USING R10.
          CLOSE     (PRINTOUT)        CLOSE OUTPUT PRINT FILE.
          DROP      R10
          LORETURN
          SPACE     3
*********************************************************************
*                 - - - -    O U T P U T    - - -
*
*        THIS SUBROUTINE PRINTS A MULTIPLE PRECISION RATIONAL NUMBER
*    ("K" WORDS IN NUMERATOR AND DENOMINATOR).  THE PRINTED RATIONAL
*    NUMBER IS IDENTIFIED BY A 64 CHARACTER TITLE PROVIDED BY THE
*    CALLER.  DIGITS ARE PRINTED 100 PER LINE.  NUMERATOR AND DENOMI-
*    NATOR ARE IDENTIFIED BY SEPARATE OUTPUT MESSAGES WHICH INDICATE
*    THE SIGN (THE DENOMINATOR IS ALWAYS ASSUMED TO BE POSITIVE).  THE
*    TRUNCATED NORMALIZED FLOATING POINT EQUIVALENT, 9 * K SIGNIFICANT
*    DIGITS, IS ALSO PRINTED WHENEVER THE DENOMINATOR IS NOT 1.
*
*    UPON ENTRY, R1 POINTS TO AN ARGUMENT LIST CONSISTING OF ...
*        R1 --> C K --  NUMBER OF FULLWORDS IN NUMERATOR, DENOMINATOR
*               C NUMERATOR   (K WORDS)
*               C DENOMINATOR (K WORDS)
*               C TITLE --  A 64 CHARACTER TITLE TO IDENTIFY OUTPUT
*                           (1ST CHARACTER IS USED AS CARRIAGE CONTROL)
*
```

```
*    SUBROUTINES CALLED:
*         OUTVRT --- CONVERTS A K-WORD INTEGER TO DECIMAL CHARACTERS (3K
*                    WORDS) FOR PRINTING
*         FLOAT  --- CONVERTS A K-WORD RATIONAL NUMBER TO A TRUNCATED
*                    FLOATING POINT EQUIVALENT (9*K EBCDIC CHARACTERS)
*********************************************************************
          SPACE
OUTPUT    NIENTRY  R11,OUTPSAV
****** ESTABLISH ADDRESSABILITY TO PRINT DCB FOR THE PUTOUT MACRO
****** AND LOAD ARGUMENT ADDRESSES.
          L        R10,=A(PRINTOUT)  LOAD @ OF PRINT DCB INTO R10
          USING    PRINTOUT,R10         AND ESTABLISH ADDRESSABILITY
*                                       TO PRINT DCB.
          SPACE
          LM       R6,R9,0(R1)       LOAD @ K (# FULLWORDS) INTO R6;
*                                          @ NUMERATOR INTO R7;
*                                          @ DENOMINATOR INTO R8;
*                                          @ OUTPUT TITLE INTO R9.
          SPACE
****** PRINT TITLE IDENTIFYING PRINTED RATIONAL NUMBER.
          PUTOUT   SKIP,3            LEAVE FOUR BLANK LINES.
          PUTOUT   =64C'*',64        SET OFF TITLE BY ASTERISKS.
          PUTOUT   0(R9),64          PRINT THE TITLE.
          PUTOUT   =64C'*',64        SET OFF TITLE BY ASTERISKS.
          PUTOUT   SKIP,1            LEAVE ONE BLANK LINE.
          SPACE
****** CONVERT NUMERATOR TO CHARACTERS AND PRINT IDENTIFYING HEADING.
          CALL     OUTVRT,,(R6,R7,OUTVAL,SIGNLOC)
          L        R3,SIGNLOC        LOAD @ OF SIGN IN EBCDIC OUTPUT.
          CLI      0(R3),C'-'        IS SIGN A "-"?
          BE       POSNUM            IF SO, PUT "POSITIVE" INTO OUTPUT.
          MVC      NUMMSG+12(4),=C'NEGA' MOVE "NEGATIVE" INTO OUTPUT.
          B        PRINT             GO PRINT NUMERATOR HEADING.
POSNUM    MVC      NUMMSG+12(4),=C'POSI' MOVE "POSITIVE" INTO OUTPUT.
PRINT     PUTOUT   NUMMSG,22         PRINT NUMERATOR HEADING
          PUTOUT   SKIP,1               AND LEAVE BLANK LINE.
          SPACE
****** SET UP REGISTERS FOR BXLE PRINT LOOPS.
          LA       R3,2(R3)          ADVANCE OUTVAL POINTER TO FIRST
*                                        DIGIT --> INDEX REGISTER.
          BYTEOFF  BYTE,R2,          GET OFFSET 4K-1 OF LOW BYTE.
          LA       R5,1(R2)          ADD 1 FOR 4K IN R5.
          M        R4,=F'3'          MULTIPLY BY 3 FOR LENGTH OF OUT-
*                                        PUT AREA --> R5.
          LA       R5,OUTVAL-1(R5)   GET LIMIT @ FOR FIRST PRINT LOOPS.
          LA       R4,100            LOAD INCREMENT FOR PRINT LOOPS.
          SPACE
****** PRINT 100 DIGITS OF NUMERATOR PER LINE.
NUMLOOP   MVC      RECORD-1(100),0(R3) MOVE NEXT 100 DIGITS TO RECORD
          PUT      PRINTOUT,RECORD      AND OUTPUT TO PRINT FILE.
          BXLE     R3,R4,NUMLOOP     PRINT REMAINING DIGITS.
          SPACE
****** CONVERT DENOMINATOR TO CHARACTERS AND PRINT IDENTIFYING
```

```
****** HEADING.
PRINT2   PUTOUT   DENMSG,24          PRINT DENOMINATOR HEADING
         PUTOUT   SKIP,1                AND LEAVE BLANK LINE.
         L        R9,ONE@             LOAD @ OF K WORD CONSTANT OF 1.
         CMPLONG  R2,0(R9),0(R8)      COMPARE DENOMINATOR TO 1.
         BNE      CNVRT               IF DEN. NOT 1, CONVERT TO CHARS.
         PUTOUT   =C' 1',2            PRINT DENOMINATOR = 1 MESSAGE
         B        EXIT5                  AND EXIT.
CNVRT    CALL     OUTVRT,,(R6,R8,OUTVAL,SIGNLOC)
         L        R3,SIGNLOC          LOAD @ OF SIGN IN EBCDIC OUTPUT.
         LA       R3,2(R3)            ADVANCE OUTVAL POINTER TO FIRST
*                                         DIGIT ---> INDEX REGISTER.
         SPACE
****** PRINT 100 DIGITS OF DENOMINATOR PER LINE.
DENLOOP  MVC      RECORD+1(100),0(R3) MOVE NEXT 100 DIGITS TO RECORD
         PUT      PRINTOUT,RECORD        AND OUTPUT TO PRINT FILE.
         BXLE     R3,R4,DENLOOP       PRINT REMAINING DIGITS.
         PUTOUT   SKIP,1              LEAVE BLANK LINE AFTER OUTPUT.
         SPACE
****** IF DENOMINATOR IS NOT 1, COMPUTE TRUNCATED NORMALIZED
****** FLOATING POINT EQUIVALENT OF THE RATIONAL NUMBER -- 9 * K
****** SIGNIFICANT DIGITS.
         CALL     FLOAT,,(R6,R7,R8,OUTVAL,FLTLEN)
         PUTOUT   =C' TRUNCATED FLOATING POINT EQUIVALENT (9K DIGITS):'X
               ,49
         PUTOUT   SKIP,1              LEAVE BLANK LINE AFTER TITLE.
         L        R5,FLTLEN           LOAD # BYTES IN FLOATING PT. NO.
         LA       R3,OUTVAL           LOAD STARTING @ INTO INDEX REG.
         LA       R5,OUTVAL-1(R5)     LOAD LIMIT @ (LAST  BYTE OF OUTVAL)
         SPACE
****** PRINT FLOATING POINT EQUIVALENT 100 DIGITS PER LINE.
FLTLOOP  MVC      RECORD+1(100),0(R3) MOVE NEXT 100 CHARS TO RECORD
         PUT      PRINTOUT,RECORD        AND OUTPUT TO PRINT FILE.
         BXLE     R3,R4,FLTLOOP       PRINT NEXT DIGITS.
         PUTOUT   SKIP,1              LEAVE BLANK LINE AFTER OUTPUT.
         L        R15,=A(BLNKS)       LOAD @ OF BLANKS INTO R15 (ADDR.)
         MVC      0(6,R15),=C'        ' GUARANTEE BLANKS (RESTORE BLANKS
*                                         WHEN K = 2 OR 3).
         SPACE
         DROP     R10
EXIT5    HIRETURN                     RETURN TO CALLER.
         SPACE    2
*       ******    DATA STORAGE FOR OUTPUT CSECT    ******
ONE@     DC       A(ONE111)           ADDRESS OF K-WORD CONSTANT OF 1
SIGNLOC  DS       F                   ADDRESS IN OUTVAL OF SIGN (- OR -)
NUMMSG   DC       CL22'0NUMERATOR (POSITIVE):' SET TO POS. OR NEG.
DENMSG   DC       CL24'0DENOMINATOR (POSITIVE):' ALWAYS POSITIVE
FLTLEN   DS       F                   # OF CHARACTERS IN FLOATING
*                                         POINT RESULT
OUTVAL   MULTPREC 3                   OUTPUT AREA FOR EBCDIC CHARACTERS
*                                         OF NUMERATOR OR DENOMINATOR
BLNKS    DC       CL110' '            BLANKS AFTER VALUE
         SPACE    3
```

```
*********************************************************************
*                  ----    O U T V R T    ----                      *
*                                                                   *
*        THIS PROCEDURE CONVERTS A MULTIPLE PRECISION INTEGER TO     *
*    DECIMAL CHARACTERS PRECEDED BY A "+" OR "-" SIGN.  THE CHARACTERS*
*    ARE CONVERTED RIGHT TO LEFT IN GROUPS OF 9 DIGITS BY MEANS OF  *
*    SUCCESSIVE DIVISIONS OF THE INTEGER BY 10**9.  THE REMAINDER   *
*    FROM EACH DIVISION IS CONVERTED TO EBCDIC CHARACTERS AND PLACED*
*    IN AN OUTPUT FIELD PROVIDED BY THE CALLER.                     *
*                                                                   *
*    UPON ENTRY, R1 POINTS TO AN ARGUMENT LIST CONSISTING OF ...    *
*        R1 --> @ K   # FULLWORDS IN MULTIPLE PRECISION INTEGER TO BE*
*                     CONVERTED (1 WORD)                            *
*                @ VALUE TO BE CONVERTED (K WORDS)                  *
*                @ SIGNED DECIMAL CHARACTER OUTPUT VALUE (3K WORDS) *
*                @ OF THE ADDRESS OF THE LEADING SIGN MOVED INTO THE*
*                  OUTPUT LINE BY OUTVRT (1 WORD)                   *
*                                                                   *
*    SUBROUTINES CALLED:                                            *
*        TWOSCOMP --- COMPUTES 2'S COMPLEMENT OF A K-WORD INTEGER   *
*        DIVISION --- COMPUTES THE QUOTIENT AND REMAINDER FROM THE  *
*                     THE DIVISION OF TWO K-WORD INTEGERS           *
*********************************************************************
          SPACE
OUTVRT    HIENTRY  R11,OUTSAVE
****** LOAD ARGUMENT ADDRESSES AND INITIALIZE REGISTERS FOR
****** CONVERSION LOOP.
          LM       R2,R5,0(R1)        LOAD @ K INTO R2;
*                                            @ VALUE TO CONVERT --> R3;
*                                            @ DECIMAL OUTPUT INTO R4;
*                                            @ SIGN IN RESULT --> R5.
          BYTEOFF  WORD,R9,           GET OFFSET OF LOW  WORD INTO R9.
          LR       R7,R9              COPY OFFSET OF LOW WORD INTO R7.
          M        R6,=F'3'           MULTIPLY BY 3 FOR OFFSET OF FIRST
*                                         EDIT (12K - 12).
          LA       R8,0(R7,R4)        LOAD @ FOR FIRST EDIT.
          LA       R10,9              LOAD 9 CHAR. FIELD WIDTH.
          SPACE
****** SET SIGN FLAG FOR LATER USE.  IF INTEGER IS NEGATIVE, USE
****** 2'S COMPLEMENT IN CONVERSION ALGORITHM.
          SR       R0,R0              ZERO SIGN FLAG  (POSITIVE).
          TM       0(R3),X'80'        TEST SIGN OF VALUE TO CONVERT.
          BNO      PREDIV             IF ZERO OR POSITIVE, START
*                                         CONVERSION.
          LA       R0,1               SET SIGN FLAG TO NEGATIVE.
          CALL     TWOSCOMP,,(R2,R3,COMPLEM,ERRORFLG)
          L        R3,=A(COMPLEM)     USE POSITIVE VALUE FOR CONVERSION
*                                         ---ENSURE ADDRESSABILITY FOR
*                                         LARGE OPERANDS.
          SPACE
****** BLANK 3-K WORD OUTPUT AREA.  PERFORM INITIAL DIVISION OF
****** INTEGER BY 10**9 (AVOID MOVELONG OF INTEGER INTO QUOT).
PREDIV    LA       R14,11(R7)         GET # BYTES - 1 IN 3K WORDS.
```

```
         LA        R7,3(R9)           GET  # BYTES - 1 IN K WORDS.
         LR        R15,R4             COPY @ OF OUTPUT VALUE INTO R15.
         SETLONG   R14,0(R15),C' '    BLANK OUTPUT VALUE.
         LDARGTBL DIGITDV,(R2,QUOT,TENSBASE,QUOT,REM,ERRONFLG)
         CALL      DIVISION,,(R2,R3,TENSBASE,QUOT,REM,ERRONFLG)
         B         LOADREM            AVOID FIRST LOOP DIVISION.
         SPACE     2
****** BEGINNING AT THE RIGHT AND PROCEEDING LEFT TO HIGHER POWERS
****** OF 10, CONVERT 2'S COMPLEMENT POSITIVE INTEGER TO DECIMAL
****** EBCDIC CHARACTERS.  GET NEXT LEFT SERIES OF 9 DIGITS BY DIVIDING
****** CURRENT REMAINDER BY 1,000,000,000.  REMAINDER FROM DIVISION
****** IS CONVERTED TO DECIMAL CHARACTERS.
DIVIDE   CALL      DIVISION,DIGITDV, DIVIDE CURRENT QUOTIENT BY 10**9.
LOADREM  L         R15,REM(R9)       LOAD LAST WORD OF REMAINDER.
*                                       (0 ---> 999,999,999) INTO R15.
         CVD       R15,OUTPACK       CONVERT NEXT 9 DECIMAL DIGITS
*                                       TO PACKED DECIMAL.
         MVC       0(12,R8),PATTERN  COPY EDIT PATTERN INTO OUTPUT AREA
         ED        0(12,R8),OUTPACK+2 EDIT INTO CHARACTER FORM FOR
*                                       OUTPUT WITH LEADING 0'S.
         SR        R8,R10            DECREMENT POINTER IN OUTPUT LINE
*                                       BY 9 TO GET @ FOR NEXT DIGITS.
         CMPLONG   R7,QUOT,ZROS      COMPARE QUOTIENT TO ZERO.
         BNE       DIVIDE            IF NOT ZERO, CONTINUE DROPPING
*                                       OFF DIGITS IN BASE 10**9.
         SPACE     2
****** APPEND LEFT SIGN TO RESULT, LEAVING ONE BLANK BETWEEN SIGN AND
****** FIRST NON-ZERO DIGIT.  STORE ADDRESS OF SIGN FOR CALLER.
         LA        R1,20(R8)         LOAD DEFAULT POSITION IN CASE
*                                       INTEGER WAS 0.
         TRT       11(10,R8),NONZERO FIND @ OF FIRST NON-ZERO DIGIT.
         S         R1,=F'2'          SUBTRACT 2 TO GET SIGN POSITION.
         MVI       1(R1),C' '        BLANK LEADING ZERO.
         ST        R1,0(R5)          STORE @ OF SIGN FOR CALLER.
         LTR       R0,R0             TEST SIGN OF RESULT.
         BNZ       NEGATIV           IF NEGATIVE, SET SIGN TO "-".
         MVI       0(R1),C'+'        SET SIGN TO "+".
         B         RTURN             RETURN TO CALLER.
NEGATIV  MVI       0(R1),C'-'        SET SIGN TO NEGATIVE ("-").
         SPACE
RTURN    HIRETURN
         SPACE     2
*        ******    DATA STORAGE FOR OUTVRT CSECT    ******
ERRONFLG DS        F                 ERROR FLAG FOR CALLS TO TWOS-
*                                       COMP, DIVISION
PATTERN  DC        X'402021'         PATTERN FOR EDITING CHAR. OUTPUT
         DC        9X'20'
NONZERO  DC        256X'00'          SCAN TABLE TO FIND NON-ZERO DIGIT
         ORG       NONZERO+C'1'      RESET LOCATION COUNTER TO "1".
         DC        9X'04'            SEARCH FOR NON-ZERO DIGITS.
         ORG       ,                 RESET LOC. COUNTER PAST TABLE.
OUTPACK  DS        D                 FIELD TO RECEIVE PACKED DECIMAL
QUOT     MULTPREC  1                 STORAGE FOR QUOTIENT OF REMAINING
```

```
*                                          VALUE/(10**9)
REM      MULTPREC 1                        REMAINDER FROM VALUE /(10**9)
ZROS     MULTPREC 1,F'0'                   K WORDS OF ZEROS
TENSBASE MULTPREC 1,F'00000',1,F'1000000000' BASE OF DIGIT CONVERS.
COMPLEM  MULTPREC 1                        STORAGE FOR 2'S COMPLEMENT OF
*                                             A NEGATIVE VALUE FOR CONVERS.
         SPACE    3
*********************************************************************************
*                    - - - -    F L O A T    - - - -
*
*        THIS SUBROUTINE CONVERTS A RATIONAL NUMBER TO A TRUNCATED
*    NORMALIZED FLOATING POINT DECIMAL RESULT OF THE FORM
*
*              -0.18765491281205431 1...E-1054    .
*
*    THE NUMBER OF DIGITS PRINTED RIGHT OF THE DECIMAL POINT IS
*    EXACTLY 9 TIMES THE PRECISION K.
*
*         THE TECHNIQUE EMPLOYED IS AS FOLLOWS:  IF THE NUMERATOR P
*    IS LARGER THAN THE DENOMINATOR Q, THE QUOTIENT P/Q IS CONVERTED
*    (BY "OUTVRT") TO EBCDIC CHARACTERS.  REMAINING DIGITS ARE EX-
*    TRACTED 9 AT A TIME BY MEANS OF SUCCESSIVE MULTIPLICATIONS OF THE
*    REMAINDER BY 10**9.  THE QUOTIENT OF THE PRODUCT DIVIDED BY THE
*    DENOMINATOR PROVIDES THE NEXT 9 DECIMAL CHARACTERS.  THE RUNNING
*    EXPONENT IS SET OR ADJUSTED AS NECESSARY AT VARIOUS STAGES OF THE
*    COMPUTATION.
*
*    UPON ENTRY, R1 POINTS TO AN ARGUMENT LIST CONSISTING OF ...
*         R1 --->  C K              --- NUMBER OF FULLWORDS IN NUMERATOR
*                                       AND DENOMINATOR
*                  C P (K WORDS) --- NUMERATOR
*                  C Q (K WORDS) --- DENOMINATOR
*                  C FLOATING POINT OUTPUT AREA (3K WORDS)
*                  C NUMBER OF BYTES IN OUTPUT (1 WORD)
*
*    FATAL ERRORS:
*         OVERFLOW OF PRODUCT OF 10**9 AND CURRENT REMAINDER INTO
*            LEFT K WORDS OR INTO SIGN BIT OF K WORDS --- ERROR #11
*
*    SUBROUTINES CALLED:
*        TWOSCOMP --- COMPUTES 2'S COMPLEMENT OF A K-WORD INTEGER
*        DIVISION --- PERFORMS DIVISION OF TWO K-WORD INTEGERS
*                     (QUOTIENT AND REMAINDER IN K WORDS)
*        OUTVRT   --- CONVERTS A K-WORD INTEGER TO EBCDIC
*                     CHARACTERS FOR PRINTING (RESULT IN 3K WORDS)
*        MULTIPLY --- MULTIPLIES TWO K-WORD INTEGERS (RESULT IN
*                     2K WORDS)
*        FERROR   --- UPON FATAL ERROR, ABORTS EXECUTION WITH ERROR
*                     MESSAGE
*********************************************************************************
         SPACE
FLOAT    HIENTRY  R11,FLTSAVE
****** LOAD ARGUMENT ADDRESSES AND COMPUTE NUMBER OF BYTES - 1 IN
```

```
****** NUMERATOR AND DENOMINATOR.
         LM        R6,R10,0(R1)       LOAD @ K INTO  R6;
*                                         @ P (NUMERATOR) INTO R7;
*                                         @ Q (DENOMINATOR) INTO R8;
*                                         @ FLOATING PT. OUTPUT INTO R9;
*                                         @ # BYTES IN FL. PT. OUTPUT
*                                           INTO R10.
         BYTEOFF   BYTE,R2,           GET 4K - 1 INTO R2.
         SPACE
****** BLANK LAST 4K BYTES OF OUTPUT AREA.  MOVE SIGN OF FRACTION
****** INTO OUTPUT AND COMPLEMENT A NEGATIVE NUMERATOR.
         LA        R3,2(R2,R2)        GET 8 * K INTO R3.
         LA        R3,0(R3,R9)        GET @ OF LOW 4K BYTES OF OUTPUT.
         SETLONG   R2,0(R3),C' '      BLANK LAST 4K BYTES OF OUTPUT.
         TM        0(R7),X'80'        TEST SIGN OF NUMERATOR.
         BNO       POSNBR             IF NON-NEGATIVE, BRANCH.
         MVC       0(3,R9),=C'-0.'    MOVE "-0." INTO OUTPUT.
         CALL      TWOSCOMP,,(R6,R7,NUMCOMP,CALLERR) GET 2'S CMPL., NUM.
         L         R7,=A(NUMCOMP)        AND RE-SET NUMERATOR POINTER
*                                        (ENSURE ADDRESSABILITY).
         B         SETUPS             GO INITIALIZE FOR OUTPUT COM-
*                                        PUTATIONS.
POSNBR   MVC       0(3,R9),=C'+0.'    MOVE "+0." INTO OUTPUT.
         SPACE
****** INITIALIZE REGISTERS FOR OUTPUT COMPUTATIONS AND LOAD ARGUMENT
****** TABLES FOR SUBROUTINE CALLS WITHIN DIGIT LOOP.
SETUPS   L         R5,0(R6)           LOAD K INTO R5.
         M         R4,=F'9'           MULTIPLY BY 9 FOR # OF DIGITS
*                                        TO PRINT --> R5
         LA        R3,1               SET EXPONENT FLAG TO 1 --- EXPONENT
*                                        NOT YET ESTABLISHED.
         SR        R0,R0              ZERO RUNNING EXPONENT.
         SR        R4,R4              ZERO # DIGITS IN PRINT LINE.
         LDARGTBL MULT22,(R6,REM1,MULTFACT,WORKFLT,CALLERR)
         LDARGTBL DIV22,(R6,REM1,R8,QUOT1,REM1,CALLERR)
         SPACE
********** COMPARE P TO Q.  IF P IS >= Q, CALL "OUTVRT" TO GET DIGITS
********** IN WHOLE NUMBER PART OF FLOATING POINT RESULT.
         CMPLONG   R2,0(R7),0(R8)     COMPARE P TO Q.
         BL        RTDIG              IF LOW, GO FETCH FRACTIONAL DIGITS.
         CALL      DIVISION,,(R6,R7,R8,QUOT1,REM1,CALLERR) DIVIDE P BY Q
         CALL      OUTVRT,,(R6,QUOT1,WORKFLT,SIGNPOS) CONVERT QUOTIENT
*                                                      TO CHARACTERS.
         SPACE
****** DETERMINE NUMBER OF DIGITS CONVERTED AND SET UP TO GET REMAIN-
****** ING DIGITS.  SET EXPONENT AND RECORD NUMBER OF DIGITS OBTAINED
****** SO FAR.
         L         R1,SIGNPOS         LOAD @ OF SIGN IN CONVERTED OUTPUT
*                                        (2 BYTES BEFORE FIRST DIGIT).
         LA        R15,1(R2)          LOAD 4K INTO R15 AND
         M         R14,=F'3'             AND MULTIPLY BY 3 FOR #
*                                        BYTES IN OUTPUT.
         L         R14,=A(WORKFLT-2)  LOAD @ OF WORKFLT-2 INTO R14
```

```
*                                          (ENSURE ADDRESSABILITY)
         LA       R15,0(R15,R14)           AND GET C OF SECOND CHARACTER
*                                          BEFORE END OF OUTPUT VALUE.
         SR       R15,R1                 GET # DIGITS LEFT OF DECIMAL POINT.
         LR       R4,R15                 SET # PRINT DIGITS TO # QUOTIENT
*                                          DIGITS.
         LR       R0,R15                 SET EXPONENT TO # QUOTIENT DIGITS.
         SR       R3,R3                  TURN OFF EXPONENT FLAG.
         SPACE
****** MOVE PRINT DIGITS TO OUTPUT FIELD AND ADJUST OUTPUT POINTER.
         BCTR     R15,R0                 GET # DIGITS FOR MOVE.
         MOVELONG R15,3(R9),2(R1)        MOVE DIGITS TO OUTPUT POSITION.
         LA       R6,4(R15,R9)           SET NEW POINTER (R6) IN OUTPUT
*                                          FIELD.
         B        RTLOOP                 COMPUTE FRACTIONAL DIGITS.
         SPACE
RTDIG    LA       R6,3(R9)               INITIALIZE POINTER IN OUTPUT FIELD.
         MOVELONG R2,REM1,0(R7)          MOVE NUM. INTO CURRENT REMAINDER.
         SPACE    2
********** COMPUTE DIGITS IN FRACTIONAL PART OF FLOATING POINT RESULT,
********** 9 DIGITS AT A TIME.
*        R0 CONTAINS THE RUNNING EXPONENT.
*        R2 CONTAINS 4K - 1.
*        R3 CONTAINS THE EXPONENT FLAG, 1 IF EXPONENT NOT YET COMPUTED.
*        R4 CONTAINS THE NUMBER OF PRINT DIGITS ACCUMULATED SO FAR.
*        R5 CONTAINS 9 # K, THE NUMBER OF DIGITS TO BE PRINTED.
*        R6 POINTS TO THE NEXT C IN THE OUTPUT PRINT AREA TO WHICH TO
*           MOVE CHARACTERS.
*        R7 CONTAINS THE ADDRESS OF NUMERATOR P (OR 2'S COMPLEMENT OF P).
*        R8 CONTAINS THE ADDRESS OF DENOMINATOR Q.
*        R9 CONTAINS THE ADDRESS OF THE FLOATING POINT RESULT (EBCDIC).
*        R10 CONTAINS THE C OF THE TOTAL # OF PRINT CHARACTERS IN 0(R9).
         SPACE
RTLOOP   CR       R4,R5                  DO WE HAVE ENOUGH PRINT DIGITS?
         BNL      EXIT14                 IF SO, OUTPUT EXPONENT.
****** MULTIPLY CURRENT REMAINDER BY 10**9.  ABORT EXECUTION
****** WITH ERROR #11 IF PRODUCT DOES NOT FIT IN K WORDS.
         CALL     MULTIPLY,MULT22,     MULTIPLY REMAINDER BY 10**9--
*                                          RESULT IN FIRST 2K WORDS OF
*                                          WORKFLT
         L        R15,=A(WORKFLT)      LOAD C OF PRODUCT INTO R15
*                                          (ENSURE ADDRESSABILITY).
         OVERCHK  R2,0(R15),11         IF PRODUCT > K WORDS, ABORT.
         LA       R15,1(R2,R15)        LOAD C OF LOW K WORDS OF PRODUCT.
         MOVELONG R2,REM1,0(R15)       MOVE NEW REMAINDER INTO OLD
*                                          REMAINDER FIELD.
         CMPLONG  R2,REM1,0(R8)        COMPARE NEW REMAINDER TO DENOM.
         BL       EXPADJ               IF NEW REMAINDER < DENOMINATOR,
*                                          ADJUST EXPONENT OR OUTPUT 0'S.
         SPACE
****** DIVIDE REMAINDER BY DENOMINATOR AND CONVERT QUOTIENT TO NEXT
****** 9 DECIMAL CHARACTERS OF OUTPUT.
         CALL     DIVISION,DIV22,      DIVIDE REMAINDER BY DIVISOR.
```

```
         L         R15,QUOT1-3(R2)   LOAD LOW WORD OF QUOT1.
         CVD       R15,OUTDIG        CONVERT QUOTIENT WORD TO PACKED
*                                      DECIMAL.
         MVC       OUTCHARS(10),=X'F0202020202020202020' MOVE EDIT
*                                      PATTERN INTO OUTCHARS.
         ED        OUTCHARS(10),OUTDIG+3 EDIT 9 PACKED DECIMAL DIGITS.
         LTR       R3,R3             EXPONENT NOT YET DETERMINED?
         BNZ       SETEXP            IF NOT YET SET, ESTABLISH EXP.
         LA        R4,9(R4)          INCREMENT # PRINT DIGITS BY 9.
         MVC       0(9,R6),OUTCHARS+1 MOVE NEXT 9  DIGITS INTO OUTPUT.
         LA        R6,9(R6)          ADVANCE POINTER IN OUTPUT FIELD.
         B         RTLOOP            GET NEXT DIGITS.
         SPACE
****** SET EXPONENT (NEGATIVE) AND MOVE NON-ZERO DIGITS TO OUTPUT.
SETEXP   LR        R14,R2            SAVE 4K - 1 IN R14 DURING TRT.
         TRT       OUTCHARS(10),FRSTDIG PUT IN R1 C OF FIRST
*                                      NON-ZERO DIGIT.
         LR        R2,R14            RESTORE 4K - 1 TO R2.
         LA        R15,OUTCHARS+10   POINT R15 TO FIRST BYTE AFTER
*                                      OUTCHARS.
         SR        R15,R1            GET # NON-ZERO DIGITS TO MOVE.
         LR        R4,R15            SET PRINT DIGITS TO # NON-ZERO
*                                      DIGITS FOUND.
         SR        R3,R3             TURN OFF EXPONENT FLAG.
         BCTR      R15,R0            GET LENGTH CODE FOR MOVE.
         EX        R15,DIGMOVE       MOVE NON-ZERO DIGITS TO OUTPUT.
         LA        R6,1(R15,R6)      INCREMENT POINTER IN OUTPUT FIELD.
         S         R15,=F'8'         GET 2'S COMPL. OF # LEADING 0'S.
         AR        R0,R15            GET FINAL VALUE OF EXPONENT.
         B         RTLOOP            GET NEXT DIGITS.
         SPACE
****** PRODUCT OF 10**9 AND REMAINDER IS LESS THAN THE DENOMINATOR.
****** ADJUST THE EXPONENT IF IT IS NOT YET ESTABLISHED.  OTHERWISE,
****** OUTPUT 9 0'S.
EXPADJ   LTR       R3,R3             EXPONENT NOT YET DETERMINED?
         BZ        MOREOS            IF ALREADY SET, OUTPUT 9 0'S.
         S         R0,=F'9'          SUBTRACT 9 FROM RUNNING EXPONENT.
         B         RTLOOP            GET NEXT DIGITS.
         SPACE
MOREOS   MVC       0(9,R6),=C'000000000' MOVE 9 ZEROS TO OUTPUT.
         LA        R6,9(R6)          INCREMENT POINTER IN PRINT FIELD.
         LA        R4,9(R4)          INCREMENT # DIGITS ACCUMULATED.
         B         RTLOOP            GET NEXT DIGITS.
         SPACE     2
****** CONVERT EXPONENT TO DECIMAL CHARACTERS AND ADD TO END OF
****** OUTPUT CHARACTERS.  STORE TOTAL NUMBER OF OUTPUT CHARACTERS.
EXIT14   LA        R15,3(R5,R9)      GET END C + 1 OF VALUE PRINTED.
         CVD       R0,OUTDIG         CONVERT EXPONENT INTO PACKED
*                                      DECIMAL.
         MVC       3(6,R15),=X'402020202120' MOVE EDIT PATTERN INTO
*                                      OUTPUT.
         ED        3(6,R15),OUTDIG+5 EDIT 5 DECIMAL CHARACTERS OF
*                                      EXPONENT.
```

```
         MVC       0(4,R15),=C'...E' MOVE "...E" INTO PRINT LINE.
         LTR       R0,R0             TEST SIGN OF EXPONENT.
         BM        MIN5              IF MINUS, SET SIGN TO "-".
         MVI       4(R15),C'+'       OVERWRITE LEFT ZERO DIGIT WITH "+".
         B         OUT22             GO ESTABLISH # OF EBCDIC CHARS.
*                                      TO PRINT.
MIN5     MVI       4(R15),C'-'       OVERWRITE LEFT ZERO DIGIT WITH "-".
OUT22    SR        R15,R9            SUBTRACT # OF OUTPUT CHARS FROM R15
         LA        R15,9(R15)           AND ADD 9 TO GET # BYTES IN
*                                      OUTPUT.
         ST        R15,0(R10)        STORE # OF OUTPUT CHARACTERS.
         SPACE
         MIRETURN
         SPACE     2
*        ******    DATA STORAGE FOR FLOAT CSECT     ******
CALLERR  DS        F                 ERROR FLAG FOR CALLS TO TMOSCOMP,
*                                      MULTIPLY, AND DIVISION
SIGNPOS  DS        F                 ADDRESS OF SIGN IN WORKFLT RESULT
FRSTDIG  DC        256X'00'          TRT TABLE TO SCAN FOR FIRST
*                                      NON-ZERO DIGIT
         ORG       FRSTDIG+C'1'      RESET LOCATION COUNTER TO "1".
         DC        9X'04'            LOOK FOR A NON-ZERO DIGIT.
         ORG       ,                 RESET LOCATION COUNTER PAST TABLE.
OUTDIG   DS        D                 DOUBLE WORD FOR PACKED DECIMAL OF
*                                      NEXT 9 DIGITS OF OUTPUT/EXPONENT
OUTCHARS DS        CL10              OUTPUT FIELD FOR NEXT 9 DIGITS OF
*                                      OUTPUT LINE PLUS LEADING
*                                      FILL CHARACTER
DIGMOVE  MVC       0(*-*,R6),0(R1)   MOVE NON-ZERO DIGITS TO OUTPUT.
         SPACE
***** MULTIPLE PRECISION VARIABLES
QUOT1    MULTPREC  1                 QUOTIENT OF REM1 AND DENOMINATOR
*                                      --- CONVERTED TO DEC. DIGITS
REM1     MULTPREC  1                 CURRENT REMAINDER IN DIVISION BY
*                                      DENOMINATOR TO GET NEXT GROUP
*                                      OF OUTPUT DIGITS
NUMCOMP  MULTPREC  1                 2'S COMPLEMENT OF NEGATIVE NUM.
WORKFLT  MULTPREC  3                 STORAGE FOR CONSTRUCTING WHOLE
*                                      # DIGITS OF FLOATING POINT
*                                      RESULT AND FOR PRODUCT OF
*                                      REM1 AND 10**9
MULTFACT MULTPREC  1,F'0',1,F'1000000000' MULTIPLICATION FACTOR OF
*                                      ONE BILLION
         TITLE     '------  FERROR CSECT --- PRINT RESIDUE OR RATIONAL ARITH
               METIC FATAL ERROR MESSAGE AND CRASH.  ------ '
*********************************************************************
*                  ------     F E R R O R     ------
*
*        THIS SUBROUTINE USES A CODED VALUE OF AN ERROR FLAG TO
*    PRINT AN ERROR MESSAGE WHEN A COMPUTATION ERROR IS DETECTED
*    IN A RESIDUE OR RATIONAL ARITHMETIC OPERATION.  PROGRAM EXE-
*    CUTION THEN CEASES (AN OC4 CRASH IS FORCED).
*
```

```
*    UPON ENTRY, R1 POINTS TO AN ARGUMENT LIST CONSISTING OF ...
*        R1 --> A ERROR FLAG (ONE WORD);  LEGAL VALUES ARE:
*                    1-11 ERROR IN RESIDUE ARITHMETIC
*                   50-62 ERROR IN RATIONAL ARITHMETIC
*
*    SUBROUTINES CALLED:
*        CLSOUT -- CLOSES PRINT FILE "FT06F001"
*********************************************************************
          SPACE
FERROR    HIENTRY  R11,FERRSAV
****** ESTABLISH ADDRESSABILITY TO PRINT DCB AND LOAD ERROR FLAG.
          L        R8,=A(PRINTOUT)   LOAD A OF PRINT DCB INTO R8
          USING    PRINTOUT,R8          AND ESTABLISH ADDRESSABILITY.
          L        R7,0(R1)          LOAD A OF ERROR FLAG.
          L        R7,0(R7)          LOAD ERROR FLAG.
          PUTOUT   SKIP,4            LEAVE FOUR BLANK LINES.
          PUTOUT   =40CL2'*-*',80    PRINT SEPARATOR FROM NORMAL OUTPUT
          SPACE
****** DETERMINE WHETHER A RATIONAL OR RESIDUE ARITHMETIC ERROR HAS
****** OCCURRED AND PERFORM A TABLE LOOKUP TO PRINT THE APPROPRIATE
****** MESSAGE.
          C        R7,=F'50'         IS THIS RESIDUE OR RATIONAL
*                                     ARITHMETIC ERROR?
          BNL      RATN              IF ERROR #50 OR LARGER --- RATIONAL
*** RESIDUE ARITHMETIC ERROR
          PUTOUT   =CL72'0FATAL ERROR DETECTED DURING RESIDUE ARITHMETICX
                .  ALL EXECUTION CEASES.',72
          LA       R6,RESERR         LOAD A OF RESIDUE ERROR TABLE.
          BCTR     R7,R0             SUBTRACT 1 FROM ERROR NUMBER.
          B        GETERR            GO GET AND PRINT ERROR MESSAGE.
*** RATIONAL ARITHMETIC ERROR
RATN      PUTOUT   =CL73'0FATAL ERROR DETECTED DURING RATIONAL ARITHMETICX
              C.  ALL EXECUTION CEASES.',73
          LA       R6,RATERR         LOAD A OF RATIONAL ERROR TABLE.
          S        R7,=F'50'         SUBTRACT 50 FROM ERROR NUMBER.
*** PRINT MESSAGE.
GETERR    SLL      R7,6              MULTIPLY BY 64 FOR OFFSET IN
*                                     ERROR TABLE.
          PUTOUT   0(R6,R7),64       PRINT ERROR MESSAGE.
          PUTOUT   SKIP,3            LEAVE 3 BLANK LINES AFTER MSG.
          SPACE
****** CLOSE OUTPUT PRINT FILE AND CEASE EXECUTION (CRASH).
CRASH     CALL     CLSOUT            CLOSE OUTPUT PRINT FILE.
          LA       R1,1              LOAD 1 INTO R1
          ST       R4,0(R1)             AND FORCE CRASH.
          SPACE
          DROP     R8
          HIRETURN
          SPACE    2
*                  DATA STORAGE FOR FERROR CSECT
********   TABLE OF ERRORS IN RESIDUE ARITHMETIC   ********
RESERR DC CL64'0*** ERROR #1: ZERO DENOM. OF RATIONAL # IN OPNRES'
 DC CL64'0*** ERROR #2: 2'S COMPLEMENT OVERFLOW OF NUMERATOR -- OPNRES'
```

```
 DC CL64'0*** ERROR #3: 2''S COMPL. OVERFLOW OF DENOMINATOR --- OPNRES'
 DC CL64'0*** ERROR #4: !P! > ORDER N OF FAREY FRACTIONS --- OPNRES.'
 DC CL64'0*** ERROR #5: !Q! > ORDER N OF FAREY FRACTIONS --- OPNRES.'
 DC CL64'0*** ERROR #6: OVERFLOW IN CALL TO ADDITION FROM RESADD'
 DC CL64'0*** ERROR #7. RESIDUE OF DIVISOR B IS ZERO --- RESDIV.'
 DC CL64'0*** ERROR #8. RESIDUE DOESN''T MAP TO FAREY FRACTION---RATMAP'
 DC CL64'0*** ERROR #9: PRECISION (MAIN PGM) && "K" (SETUP) DISAGREE.'
 DC CL64'0*** ERROR #10: INVALID CHOICE OF PRECISION "K" ---MAIN, SETUP'
 DC CL64'0*** ERROR #11: OVERFLOW, PRODUCT (10**9)*REMAINDER --- FLOAT'
         SPACE
******** TABLE OF ERRORS IN RATIONAL ARITHMETIC ********
RATERR   DC       CL64'0*** ERROR#50: OVERFLOW, PRODUCT OF A NUM., B DEN
               OM.---RATADD'
 DC CL64'0*** ERROR #51: OVERFLOW, PRODUCT OF A NUM., B DEN. --- RATADD'
 DC CL64'0*** ERROR #52: OVERFLOW, ADDITION TO GET NUM. OF C --- RATADD'
 DC CL64'0*** ERROR #53: OVERFLOW, PRODUCT OF DENOMS. OF A, B ---RATADD'
 DC CL64'0*** ERROR #54: 2''S COMPL. OVERFLOW OF NUM. OF B --- RATSUB'
 DC CL64'0*** ERROR #55: OVERFLOW, PRODUCT OF NUMS. OF A, B --- RATMLT'
 DC CL64'0*** ERROR #56: OVERFLOW, PRODUCT OF DENOMS. OF A,B --- RATMLT'
 DC CL64'0*** ERROR #57: ZERO DENOMINATOR OF RATIONAL NUMBER IN OPNRAT'
 DC CL64'0*** ERROR #58: 2''S COMPL. OVERFLOW IN NUMERATOR --- OPNRAT'
 DC CL64'0*** ERROR #59: 2''S COMPL. OVERFLOW IN DENOMINATOR --- OPNRAT'
 DC CL64'0*** ERROR #60: NUMERATOR OF DIVISOR B IS ZERO --- RATDIV'
 DC CL64'0*** ERROR #61: 2''S COMPL. OVERFLOW OF NUM. OF B --- RATDIV'
 DC CL64'0*** ERROR #62: PRECISION (MAIN PGM) && "K" (SETUP) DISAGREE.'
```

APPENDIX H

GAUSSIAN ELIMINATION PROGRAMS FOR HILBERT MATRICES (PL1)

```
LIN_EQ: PROC OPTIONS(MAIN);
/*****************************************************************/
/*                                                               */
/*                        L I N _ E Q                            */
/*                (MULTIPLE PRECISION RATIONAL ARITHMETIC)       */
/*                                                               */
/*    THIS PROGRAM USES MULTIPLE PRECISION RATIONAL ARITH-       */
/*  METIC TO SOLVE THE N BY N SYSTEM OF LINEAR EQUATIONS         */
/*  AX = B.  THE MATRIX A IS A FINITE SEGMENT OF THE INFINITE    */
/*  HILBERT MATRIX (EACH I,J ELEMENT IS GIVEN BY 1/(I+J-1) ).    */
/*  A SINGLE SPECIFIED ELEMENT OF B ("NON_ZERO_B_POSITION")      */
/*  IS EQUAL TO 1;  ALL OTHER ELEMENTS OF B ARE ZERO.            */
/*      THE SYSTEM IS SOLVED BY USING GAUSSIAN ELIMINATION       */
/*  TO PRODUCE A TRIANGULARIZED AUGMENTED MATRIX.  BACKWARD      */
/*  SUBSTITUTION THEN GENERATES THE SOLUTION VECTOR X.  THE      */
/*  SYSTEM OF EQUATIONS HAS NO UNIQUE SOLUTION IF AT ANY STEP    */
/*  OF THE TRIANGULARIZATION A NON-ZERO PIVOT CANNOT BE          */
/*  FOUND, OR IF THE VALUE OF A(N,N) IN THE TRIANGULARIZED       */
/*  MATRIX IS 0.                                                 */
/*                                                               */
/*  MODIFICATIONS:                                               */
/*    THE PARTICULAR PROGRAM LISTED HERE IS SET UP TO SOLVE      */
/*  SYSTEMS UP TO N = 30 BY 30 WITH A MAXIMUM PRECISION K OF     */
/*  20. THE "DCL" INITIALIZATIONS OF "PRECISION", "MATRIX_       */
/*  SIZE", AND "NON_ZERO_B_POSITION" CAN BE ALTERED AS NEC-      */
/*  ESSARY FOR THE GIVEN EXECUTION.  IF PROGRAM EXECUTION        */
/*  ENDS WITH A RATIONAL ARITHMETIC OVERFLOW ERROR, THE VALUE    */
/*  OF PRECISION MUST BE INCREASED.  LARGER MATRICES AND         */
/*  PRECISIONS CAN BE EMPLOYED BY CHANGING THE INITIALIZA        */
/*  TIONS IN THE DECLARATIONS OF "MATRIX_SIZE", "PRECISION",     */
/*  AND THE ARRAYS A, ADENOM, B, BDENOM, X, AND XDENOM.          */
/*                                                               */
/*  REQUIRED DATA:  NONE                                         */
/*                                                               */
/*  PROGRAM OUTPUT:                                              */
/*      THE HILBERT MATRIX IS PRINTED, AND THE NON-ZERO          */
/*  ELEMENT OF B IS REPORTED.  THE SOLUTION VECTOR X IS          */
/*  PRINTED.  IF THE SYSTEM HAS NO UNIQUE SOLUTION OR IF THE     */
/*  VALUE OF K IS TOO SMALL TO PREVENT OVERFLOW, AN APPRO-       */
/*  PRIATE ERROR MESSAGE IS PRODUCED.                            */
/*                                                               */
/*  PROCEDURES CALLED:                                           */
/*    --- INTERNAL, PL1 ---                                      */
/*    GAUSS    -- TRIANGULARIZES AUGMENTED MATRIX A BY PER-      */
/*                FORMING STANDARD GAUSSIAN ELIMINATION          */
/*    BCKSOLV  -- USES BACKWARD SUBSTITUTION TO OBTAIN           */
/*                SOLUTION VECTOR X FROM TRIANGULARIZED          */
/*                MATRIX                                         */
/*                                                               */
```

```
/*    --- EXTERNAL, ASSEMBLER ---                                    */
/*      RATSET   --- VERIFIES THAT "PRECISION" AGREES WITH "K"       */
/*                   OF ASSEMBLER SUPPORT ROUTINES AND OPENS         */
/*                   OUTPUT PRINT FILE "FT06F001"                    */
/*      CLSOUT   --- CLOSES OUTPUT PRINT FILE "FT06F001"             */
/*      OUTPUT   --- PRINTS A TITLED MULTIPLE PRECISION              */
/*                   RATIONAL NUMBER                                 */
/*      RATADD   --- PERFORMS ADDITION OF TWO MULTIPLE PRECI-        */
/*                   SION RATIONAL NUMBERS                           */
/*      RATSUB   --- PERFORMS SUBTRACTION OF MULTIPLE PRECISION      */
/*                   RATIONAL NUMBERS                                */
/*      RATMLT   --- PERFORMS MULTIPLICATION OF TWO MULTIPLE         */
/*                   PRECISION RATIONAL NUMBERS                      */
/*      RATDIV   --- PERFORMS DIVISION OF MULTIPLE PRECISION         */
/*                   RATIONAL NUMBERS                                */
/*******************************************************************/

DEFAULT RANGE(*) VALUE(FIXED(31,0) BINARY);
/******* PROCEDURE DECLARATIONS *******/
DCL RATSET ENTRY OPTIONS(ASSEMBLER,INTER);
DCL CLSOUT ENTRY OPTIONS(ASSEMBLER,INTER);
DCL OUTPUT ENTRY OPTIONS(ASSEMBLER,INTER);
DCL RATADD ENTRY OPTIONS(ASSEMBLER,INTER);
DCL RATSUB ENTRY OPTIONS(ASSEMBLER,INTER);
DCL RATMLT ENTRY OPTIONS(ASSEMBLER,INTER);
DCL RATDIV ENTRY OPTIONS(ASSEMBLER,INTER);

/**** DECLARATIONS OF THE PRECISION AND ARRAY DIMENSIONS ****/
DCL PRECISION FIXED(31,0) BIN    /* # OF FULLWORDS (K) IN          */
      INIT(3);                   /* MULTIPLE PRECISION OPERANDS    */
DCL MATRIX_SIZE FIXED(31,0) BIN  /* ACTUAL DIMENSIONS OF MATRIX    */
      INIT(10);                  /* A AND VECTOR B FOR CURRENT     */
                                 /* EXECUTION                      */


   /*****     ARRAY DECLARATIONS OF A, B, AND X     *****/
DCL A(MATRIX_SIZE,MATRIX_SIZE,   /* MATRIX OF COEFFICIENTS         */
   PRECISION) FIXED(31,0) BIN    /* OF LINEAR EQUATIONS            */
   INIT((18000)0);
DCL ADENOM(MATRIX_SIZE,MATRIX_SIZE, /* DENOMINATORS OF ELE-        */
   PRECISION) FIXED(31,0) BIN    /* MENTS OF A                     */
   INIT((18000)0);
DCL B(MATRIX_SIZE,PRECISION)     /* VECTOR B                       */
   FIXED(31,0) BIN INIT((600)0);
DCL BDENOM(MATRIX_SIZE,PRECISION)  /* DENOMINATORS OF              */
   FIXED(31,0) BIN INIT((600)0);  /* ELEMENTS OF B                */
DCL NON_ZERO_B_POSITION FIXED(31,0) /* POSITION OF NON-ZERO        */
   BIN INIT(7);                  /* ELEMENT OF VECTOR B            */
DCL X(MATRIX_SIZE,PRECISION)     /* VECTOR OF SOLUTIONS            */
   FIXED(31,0) BIN INIT((600)0);
DCL XDENOM(MATRIX_SIZE,PRECISION)  /* DENOMINATORS OF ELE-         */
   FIXED(31,0) BIN INIT((600)0); /* MENTS OF X                     */
```

```
DCL TITLE CHAR(64) ;                    /* TITLE FOR CALL TO OUTPUT    */
DCL FT06F001 FILE PRINT EXT;            /* PRINT FILE ESTABLISHED IN   */
                                        /* ASSEMBLER SUPPORT ROUTINES  */



/****** VERIFY THAT "PRECISION" AGREES WITH ASSEMBLER'S "K"      */
/****** AND OPEN OUTPUT PRINT FILE "FT06F001."  PRINT HEADINGS. */
CALL RATSET(PRECISION);
PUT FILE(FT06F001) EDIT ('*_*_*_  GAUSSIAN ELIMINATION/BACK SUBSTI',
       'TUTION SOLUTION TO SYSTEM OF LINEAR EQUATIONS AX = B  _*_*_*',
       'MATRIX OF COEFFICIENTS A IS AN N X N HILBERT MATRIX, N = ',
       MATRIX_SIZE,':') (A,A,SKIP(2),A,F(2),A);
PUT FILE(FT06F001) SKIP(2);

/****** INITIALIZE VECTOR B.  ALL ELEMENTS ARE ZERO EXCEPT  */
/****** "NON_ZERO_B_POSITION."                              */
B(NON_ZERO_B_POSITION,PRECISION) = 1;
BDENOM(*,PRECISION) = 1;   /* SET ALL DENOMINATORS OF B TO 1. */

/****** INITIALIZE A TO MATRIX_SIZE BY MATRIX_SIZE HILBERT MATRIX. */
A(*,*,PRECISION) = 1;    /* SET ALL NUMERATORS OF A TO 1. */
DO I = 1 TO MATRIX_SIZE;
   DO J = 1 TO MATRIX_SIZE;
      ADENOM(I,J,PRECISION) = I + J - 1;
      PUT FILE(FT06F001) EDIT ('1/',ADENOM(I,J,PRECISION))
               (X(2),A,F(2));
   END;
   PUT FILE(FT06F001) SKIP;
END;

/****** REPORT B AND PRINT HEADING THAT INDICATES THE PRECISION. */
PUT FILE(FT06F001) SKIP(2) EDIT ('ALL ELEMENTS OF VECTOR B ARE ',
               'ZERO EXCEPT FOR B(',NON_ZERO_B_POSITION,
               '), WHICH IS 1.') (A,A,F(2),A);
PUT FILE(FT06F001) SKIP(3) EDIT('---- SOLUTION USING K-WORD RATIONAL ',
       'ARITHMETIC, WHERE K = ', PRECISION,' ----') (A,A,F(3),A);
PUT FILE(FT06F001) SKIP EDIT(' ') (A);  /* FLUSH OUTPUT BUFFER. */
TITLE = '              ------  SOLUTION VECTOR X(  ):  ------';


/****** TRIANGULARIZE MATRIX, PERFORMING GAUSSIAN ELIMINATION. */
CALL GAUSS;


/****** BACK SOLVE TRIANGULARIZED MATRIX TO GET SOLUTIONS X(I). */
CALL BCKSOLV;


/****** PRINT SOLUTIONS X(I). */
PUT FILE(FT06F001) SKIP;
DO I = 1 TO MATRIX_SIZE;
```

```
   PUT STRING(SUBSTR(TITLE,40,2)) EDIT(I) (F(2));
   CALL OUTPUT(PRECISION,X(I,1),XDENOM(I,1),TITLE);
END;

CALL CLSOUT;     /* CLOSE OUTPUT PRINT FILE. */



GAUSS: PROC;
/*****************************************************************************/
/*                        G A U S S                                          */
/*                                                                           */
/*        THIS INTERNAL PROCEDURE USES RATIONAL ARITHMETIC TO                */
/*   PERFORM STANDARD GAUSSIAN ELIMINATION TO TRIANGULARIZE THE              */
/*   AUGMENTED MATRIX.  IF A ZERO PIVOT IS ENCOUNTERED IN POSI-              */
/*   TION A(I,I) DURING THIS TRIANGULARIZATION, ROW-SWITCHING IS             */
/*   PERFORMED TO OBTAIN A NON-ZERO PIVOT.  IF A NON-ZERO PIVOT              */
/*   CANNOT BE FOUND, THE SYSTEM OF LINEAR EQUATIONS HAS NO                  */
/*   UNIQUE SOLUTION, AND EXECUTION CEASES.                                  */
/*****************************************************************************/

      /****    LOCAL VARIABLE DECLARATIONS    ****/
DCL M_JI(PRECISION) FIXED(31,0) BIN;     /* RATIO OF MATRIX ELE-    */
                                         /* ELEMENT A(J,I) TO       */
                                         /* PIVOT ELEMENT A(I,I)    */
DCL M_JI_DENOM(PRECISION) FIXED(31,0)    /* DENOMINATOR OF M_JI     */
   BINARY;
DCL (TEMP1(PRECISION),TEMP2(PRECISION))  /* TEMPORARY STORAGE FOR   */
      FIXED(31,0) BINARY;                /* RESULTS FROM RATMLT     */
DCL TEMP  FIXED(31,0) BINARY;            /* TEMPORARY STORAGE FOR   */
                                         /* WORD OF MATRIX A OR     */
                                         /* VECTOR B                */
DCL TEMPDENOM FIXED(31,0) BINARY;        /* TEMPORARY STORAGE FOR   */
                                         /* DENOMINATOR WORD OF     */
                                         /* MATRIX A OR VECTOR B    */
DCL ZERO_PIVOT BIT(1);                   /* SET TO TRUE IF CANDI-   */
                                         /* DATE FOR PIVOT IS ZERO  */



/********* TRIANGULARIZE ALL ROWS OF THE AUGMENTED MATRIX.  */
DO I = 1 TO MATRIX_SIZE-1;


   /***** CHECK FOR ZERO PIVOT A(I,I).  IF PIVOT IS ZERO, SEARCH   */
   /***** ROWS J = I+1 TO N FOR A NON-ZERO PIVOT.                  */
   DO K = 1 TO PRECISION WHILE (A(I,I,K) = 0);
   END;
   IF K > PRECISION   /* IS THE PIVOT ZERO? */
      THEN DO,   /* CHECK ROWS I+1 TO N FOR A NON-ZERO PIVOT. */
              ZERO_PIVOT = '1'B;
              DO J = I+1 TO MATRIX_SIZE WHILE(ZERO_PIVOT);
                 /*** CHECK NEXT PIVOT CANDIDATE.  */
```

```
      DO K = 1 TO PRECISION WHILE (A(J,I,K) = 0);
      END;
      IF K <= PRECISION
         THEN ZERO_PIVOT = '0'B;
         ELSE ZERO_PIVOT = '1'B;
   END;

   /**** IF A NON-ZERO PIVOT HAS BEEN FOUND IN ROW J,  */
   /**** SWAP ROWS I AND J.                            */
   IF ^(ZERO_PIVOT)
      THEN DO;
              PUT FILE(FT06F001) SKIP LIST
                 ('SWITCHING ROWS TO FIND NON-ZERO PIVOT');
              J = J - 1;  /* RESET J TO ROW OF FIRST */
                          /* NON-ZERO PIVOT.         */
              /*** SWAP ROWS I AND J OF MATRIX A. */
              DO K = 1 TO MATRIX_SIZE;  /* COLUMN POINTER */
                 /*** LOOP THROUGH ALL WORDS OF ELEMENT. */
                 DO L = 1 TO PRECISION;
                    TEMP  = A(I,K,L); /* SAVE I-TH ROW  */
                                      /* ELEMENT WORD.  */
                    TEMPDENOM = ADENOM(I,K,L);
                    A(I,K,L) = A(J,K,L);
                    ADENOM(I,K,L) = ADENOM(J,K,L);
                    A(J,K,L) = TEMP ;
                    ADENOM(J,K,L) = TEMPDENOM;
                 END;
              END;
              /*** SWAP ELEMENTS I AND J OF VECTOR B ---  */
              /*** LOOP THOUGH ALL WORDS OF ELEMENT.      */
              DO K = 1 TO PRECISION;
                 TEMP = B(I,K);   /* SAVE I-TH ELEMENT WORD. */
                 TEMPDENOM = BDENOM(I,K);
                 B(I,K) = B(J,K);
                 BDENOM(I,K) = BDENOM(J,K);
                 B(J,K) = TEMP;
                 BDENOM(J,K) = TEMPDENOM;
              END;
           END;

      /**** A NON-ZERO PIVOT CANNOT BE FOUND.  **/
      /**** ABORT EXECUTION.                   **/
      ELSE DO;
              PUT FILE(FT06F001) SKIP(3) EDIT ('*******  ER',
                 'ROR -- NON-ZERO PIVOT COULD NOT BE ',
                 'FOUND DURING GAUSSIAN ELIMINATION.',
                 '*******  SYSTEM OF LINEAR EQUATIONS ',
                 'HAS NO UNIQUE SOLUTION.') (A,A,A,SKIP,
                 A,A);
              STOP;
           END;
END;
```

```
      /********* REDUCE ROWS J =  I+1 TO N.  COMPUTE FACTOR  */
      /********* M(J,I) = A(J,I)/A(I,I) AND SUBTRACT PRODUCT */
      /********* OF M(J,I) AND ROW I FROM ROW J.             */
      DO J = I+1 TO MATRIX_SIZE;
         CALL RATDIV(PRECISION,A(J,I,1),ADENOM(J,I,1),A(I,I,1),
                     ADENOM(I,I,1),M_JI,M_JI_DENOM);

         /*****  MULTIPLY EACH MATRIX COLUMN K OF PIVOTAL ROW I BY   */
         /*****  FACTOR M(J,I) AND SUBTRACT FROM COLUMN K OF ROW J.  */
         DO K = I TO MATRIX_SIZE;
            CALL RATMLT(PRECISION,A(I,K,1),ADENOM(I,K,1),M_JI,
                        M_JI_DENOM,TEMP1,TEMP2);
            CALL RATSUB(PRECISION,A(J,K,1),ADENOM(J,K,1),TEMP1,
                        TEMP2,A(J,K,1),ADENOM(J,K,1));
         END;

         /***** REDUCE AUGMENTED ELEMENT B(I).  */
         CALL RATMLT(PRECISION,B(I,1),BDENOM(I,1),M_JI,
                     M_JI_DENOM,TEMP1,TEMP2);
         CALL RATSUB(PRECISION,B(J,1),BDENOM(J,1),TEMP1,TEMP2,
                     B(J,1),BDENOM(J,1));
      END;

   END;


   RETURN;
   END GAUSS;



   BCKSOLV: PROC;
   /*****************************************************************/
   /*                 - - - -    B C K S O L V    - - - -           */
   /*                                                               */
   /*        THIS INTERNAL PROCEDURE USES RATIONAL ARITHMETIC AND   */
   /*    BACKWARD SUBSTITUTION TO SOLVE A TRIANGULARIZED AUGMENTED  */
   /*    MATRIX A FOR THE SOLUTIONS X TO THE SYSTEM OF LINEAR EQUA- */
   /*    TIONS AX = B.  IF A(N,N) IS 0, THE SYSTEM DOES NOT HAVE A  */
   /*    UNIQUE SOLUTION AND EXECUTION CEASES.                      */
   /*****************************************************************/

         /****    LOCAL VARIABLE DECLARATIONS    ****/
   DCL (SUMPROD(PRECISION),SUMDENOM        /* TEMPORARY VARIABLES TO   */
        (PRECISION)) FIXED(31,0) BIN;      /* HOLD SUM OF PRODUCTS     */
                                           /* OF COEFFICIENTS AND PRE- */
                                           /* VIOUSLY COMPUTED ROOTS   */
   DCL (PRODRAT(PRECISION),PRODDENOM       /* TEMPORARY VARIABLES TO   */
        (PRECISION)) FIXED(31,0) BIN;      /* HOLD PRODUCT OF COEFFI-  */
                                           /* CIENT AND ROOT           */
```

```
/****** CHECK FOR A(N,N) = 0.  IF A(N,N) = 0, THE SYSTEM OF  */
/****** EQUATIONS HAS NO UNIQUE SOLUTION -- ABORT EXECUTION. */
DO J = 1 TO PRECISION WHILE (A(MATRIX_SIZE,MATRIX_SIZE,J) = 0);
END;
IF J > PRECISION
   THEN DO;
           PUT FILE(FT06F001) SKIP(3) EDIT ('*******  ERROR -- CAUS',
              'SIAN ELIMINATION PRODUCES A(N,N) = 0.','******* SYS',
              'TEM OF LINEAR EQUATIONS HAS NO UNIQUE SOLUTION.')
              (A,A,SKIP,A,A);
           STOP;
        END;

/******* DETERMINE ROOT X(N).  */
CALL RATDIV(PRECISION,B(MATRIX_SIZE,1),BDENOM(MATRIX_SIZE,1),
           A(MATRIX_SIZE,MATRIX_SIZE,1),ADENOM(MATRIX_SIZE,MATRIX_SIZE,1),
           X(MATRIX_SIZE,1),XDENOM(MATRIX_SIZE,1));


/******* EXTRACT EACH ADDITIONAL ROOT X(I).    */
DO I = MATRIX_SIZE - 1 TO 1 BY -1;

   /***** INITIALIZE ROOT X(I) TO TRIANGULARIZED ELEMENT B(I).  */
   DO K = 1 TO PRECISION;
      X(I,K) = B(I,K);
      XDENOM(I,K) = BDENOM(I,K);
   END;

   /***** SUBTRACT FROM X(I) THE PRODUCT OF EACH COEFFICIENT AND  */
   /***** PREVIOUSLY COMPUTED ROOT.                               */
   SUMPROD, SUMDENOM = 0;  /* INITIALIZE SUM OF PRODUCTS TO 0. */
   SUMDENOM(PRECISION) = 1;
   /** COMPUTE SUM OF PRODUCTS OF COEFFICIENTS AND ROOTS.  */
   DO J = I+1 TO MATRIX_SIZE;  /* COMPUTE SUM OF PRODUCTS. */
      CALL RATMLT(PRECISION,A(I,J,1),ADENOM(I,J,1),X(J,1),XDENOM(J,1),
                  PRODRAT,PRODDENOM);
      CALL RATADD(PRECISION,SUMPROD,SUMDENOM,PRODRAT,PRODDENOM,
                  SUMPROD,SUMDENOM);
   END;
   /** SUBTRACT SUM OF PRODUCTS FROM ORIGINAL X(I).  */
   CALL RATSUB(PRECISION,X(I,1),XDENOM(I,1),SUMPROD,SUMDENOM,
               X(I,1),XDENOM(I,1));

   /***** DIVIDE BY A(I,I) FOR ROOT X(I).  */
   CALL RATDIV(PRECISION,X(I,1),XDENOM(I,1),A(I,I,1),ADENOM(I,I,1),
               X(I,1),XDENOM(I,1));
END;


RETURN;
END BCKSOLV;
```

```
END LIN_EQ;


LIN_EQ: PROC OPTIONS(MAIN);
/*****************************************************************/
/*                  - - - -   L I N _ E Q   - - - -              */
/*          (MULTIPLE PRECISION RESIDUE ARITHMETIC)              */
/*                                                               */
/*       THIS PROGRAM USES MULTIPLE PRECISION RESIDUE ARITH-     */
/*   METIC TO SOLVE THE N BY N SYSTEM OF LINEAR EQUATIONS        */
/*   AX = B.  THE MATRIX A IS A FINITE SEGMENT OF THE INFINITE   */
/*   HILBERT MATRIX (EACH I,J ELEMENT IS GIVEN BY 1/(I+J-1) ).   */
/*   A SINGLE SPECIFIED ELEMENT OF B ("NON_ZERO_B_POSITION")     */
/*   IS EQUAL TO 1;  ALL OTHER ELEMENTS OF B ARE ZERO.           */
/*        THE SYSTEM IS SOLVED BY USING GAUSSIAN ELIMINATION     */
/*   TO PRODUCE A TRIANGULARIZED AUGMENTED MATRIX.  BACKWARD     */
/*   SUBSTITUTION THEN GENERATES THE SOLUTION VECTOR X.  THE     */
/*   SYSTEM OF EQUATIONS HAS NO UNIQUE SOLUTION IF AT ANY STEP   */
/*   OF THE TRIANGULARIZATION A NON-ZERO PIVOT CANNOT BE         */
/*   FOUND, OR IF THE VALUE OF A(N,N) IN THE TRIANGULARIZED      */
/*   MATRIX IS 0.                                                */
/*                                                               */
/*   MODIFICATIONS:                                              */
/*      THE PARTICULAR PROGRAM LISTED HERE IS SET UP TO SOLVE    */
/*   SYSTEMS UP TO N = 30 BY 30 WITH A MAXIMUM PRECISION K OF    */
/*   20. THE "DCL" INITIALIZATIONS OF "PRECISION", "MATRIX_      */
/*   SIZE", AND "NON_ZERO_B_POSITION" CAN BE ALTERED AS NEC-     */
/*   ESSARY FOR THE GIVEN EXECUTION.  IF PROGRAM EXECUTION       */
/*   ENDS WITH A RESIDUE ARITHMETIC OVERFLOW ERROR, THE VALUE    */
/*   OF PRECISION MUST BE INCREASED.  LARGER MATRICES AND        */
/*   PRECISIONS CAN BE EMPLOYED BY CHANGING THE INITIALIZA-      */
/*   TIONS IN THE DECLARATIONS OF "MATRIX_SIZE", "PRECISION",    */
/*   AND THE ARRAYS A, ADENOM, B, BDENOM, X, AND XDENOM.         */
/*                                                               */
/*   REQUIRED DATA:  NONE                                        */
/*                                                               */
/*   PROGRAM OUTPUT:                                             */
/*        THE HILBERT MATRIX IS PRINTED, AND THE NON-ZERO        */
/*   ELEMENT OF B IS REPORTED.  THE SOLUTION VECTOR X IS         */
/*   PRINTED.  IF THE SYSTEM HAS NO UNIQUE SOLUTION OR IF THE    */
/*   VALUE OF K IS TOO SMALL TO PREVENT OVERFLOW, AN APPRO-      */
/*   PRIATE ERROR MESSAGE IS PRODUCED.                           */
/*                                                               */
/*   PROCEDURES CALLED:                                          */
/*     - - -  INTERNAL, PL1 - - - -                              */
/*     GAUSS    -  TRIANGULARIZES AUGMENTED MATRIX A BY PER-     */
/*                 FORMING STANDARD GAUSSIAN ELIMINATION         */
/*     BCKSOLV  -  USES BACKWARD SUBSTITUTION TO OBTAIN SOLU-    */
/*                 TION VECTOR X FROM TRIANGULARIZED MATRIX      */
```

```
/*    ---- EXTERNAL, ASSEMBLER ---                                    */
/*      SETUP    --  COMPUTES MODULUS AND ASSOCIATED ORDER-N        */
/*                   OF FAREY FRACTIONS AND OPENS OUTPUT PRINT      */
/*                   FILE "FT06F001"                                */
/*      CLSOUT   --  CLOSES OUTPUT PRINT FILE "FT06F001"            */
/*      OUTPUT   --  PRINTS A TITLED MULTIPLE PRECISION             */
/*                   RATIONAL NUMBER                                */
/*      OPNRES   --  COMPUTES RESIDUE EQUIVALENT OF MULTIPLE        */
/*                   PRECISION RATIONAL NUMBER                      */
/*      RATMAP   --  MAPS RESIDUE RESULT TO AN ORDER-N FAREY        */
/*                   FRACTION                                       */
/*      RESADD   --  PERFORMS THE MULTIPLE PRECISION RESIDUE        */
/*                   ADDITION !A-B! MOD M                           */
/*      RESSUB   --  PERFORMS THE MULTIPLE PRECISION RESIDUE        */
/*                   SUBTRACTION !A-B! MOD M                        */
/*      RESMLT   --  PERFORMS THE MULTIPLE PRECISION RESIDUE        */
/*                   MULTIPLICATION !A*B! MOD M                     */
/*      RESDIV   --  PERFORMS THE MULTIPLE PRECISION RESIDUE        */
/*                   DIVISION !A/B! MOD M                           */
/*******************************************************************/

DEFAULT RANGE(*) VALUE(FIXED(31,0) BINARY);
/*******  PROCEDURE DECLARATIONS  *******/
DCL SETUP ENTRY OPTIONS(ASSEMBLER,INTER);
DCL CLSOUT ENTRY OPTIONS(ASSEMBLER,INTER);
DCL OUTPUT ENTRY OPTIONS(ASSEMBLER,INTER);
DCL OPNRES ENTRY OPTIONS(ASSEMBLER,INTER);
DCL RATMAP ENTRY OPTIONS(ASSEMBLER,INTER);
DCL RESADD ENTRY OPTIONS(ASSEMBLER,INTER);
DCL RESSUB ENTRY OPTIONS(ASSEMBLER,INTER);
DCL RESMLT ENTRY OPTIONS(ASSEMBLER,INTER);
DCL RESDIV ENTRY OPTIONS(ASSEMBLER,INTER);

/*** DECLARATIONS OF THE PRECISION, MODULUS, AND ARRAY DIMENSIONS ***/
DCL PRECISION FIXED(31,0) BIN    /* # OF FULLWORDS (K) IN          */
              INIT(17);          /* MULTIPLE PRECISION OPERANDS    */
DCL MATRIX_SIZE FIXED(31,0) BIN  /* ACTUAL DIMENSIONS OF MATRIX    */
              INIT(20);          /* A AND VECTOR B FOR CURRENT     */
                                 /* EXECUTION                      */
DCL MODULUS(PRECISION)           /* K-WORD MERSENNE PRIME          */
      FIXED(31,0) BIN;           /* MODULUS                        */

   /****     ARRAY DECLARATIONS OF A, B, AND X     ****/
DCL A(MATRIX_SIZE,MATRIX_SIZE,       /* MATRIX OF COEFFICIENTS     */
   PRECISION) FIXED(31,0) BINARY     /* OF LINEAR EQUATIONS        */
   INIT((18000)0);
DCL ADENOM(PRECISION) FIXED(31,0)    /* DENOMINATOR OF AN ELE-     */
   BIN INIT((20)0);                  /* MENT OF A -- RESET AS      */
                                     /* NECESSARY                  */
DCL B(MATRIX_SIZE,PRECISION)         /* VECTOR B                   */
   FIXED(31,0) BIN INIT((600)0);
DCL BDENOM(PRECISION) FIXED(31,0)    /* DENOMINATOR Q OF ANY       */
   BIN INIT((20)0);                  /* RATIONAL ELEMENT OF B      */
```

```
DCL NON_ZERO_B_POSITION FIXED(31,0)    /* POSITION OF NON-ZERO    */
   BIN INIT(15);                       /* ELEMENT OF VECTOR B     */
DCL X(MATRIX_SIZE,PRECISION)           /* VECTOR OF SOLUTIONS     */
   FIXED(31,0) BIN INIT((600)0);
DCL XDENOM(PRECISION) FIXED(31,0)      /* DENOMINATOR Q OF ANY    */
   BIN INIT((20)0);                    /* RATIONAL ELEMENT OF X   */

DCL TITLE CHAR(64) ;                /* TITLE FOR CALL TO OUTPUT    */
DCL FT06F001 FILE PRINT EXT;        /* PRINT FILE ESTABLISHED IN   */
                                    /* ASSEMBLER SUPPORT ROUTINES  */


/****** DETERMINE THE LARGEST MERSENNE PRIME THAT WILL FIT IN    */
/****** "PRECISION" WORDS.  THIS PRIME BECOMES THE MODULUS.      */
/****** OPEN OUTPUT PRINT FILE "FT06F001" AND PRINT HEADINGS.    */
CALL SETUP(PRECISION,MODULUS);
PUT FILE(FT06F001) EDIT ('*_*_*_  GAUSSIAN ELIMINATION/BACK SUBSTI',
      'TUTION SOLUTION TO SYSTEM OF LINEAR EQUATIONS AX = B  _*_*_*',
      'MATRIX OF COEFFICIENTS A IS AN N X N HILBERT MATRIX, N = ',
      MATRIX_SIZE,':') (A,A,SKIP(2),A,F(2),A),
PUT FILE(FT06F001) SKIP(2);

/****** INITIALIZE VECTOR B.  ALL ELEMENTS ARE ZERO EXCEPT       */
/****** "NON_ZERO_B_POSITION."                                   */
B(NON_ZERO_B_POSITION,PRECISION) = 1,
BDENOM(PRECISION) = 1;

/****** INITIALIZE A TO "MATRIX_SIZE" BY "MATRIX_SIZE" HILBERT   */
/****** MATRIX.  PRINT MATRIX AND GET RESIDUE REPRESENTATION     */

/****** OF EACH ELEMENT OF A AND B.                              */
A(*,*,PRECISION) = 1;    /* SET ALL NUMERATORS OF A TO 1. */
DO I = 1 TO MATRIX_SIZE;
   DO J = 1 TO MATRIX_SIZE;
      ADENOM(PRECISION) = I + J - 1;
      PUT FILE(FT06F001) EDIT ('1/',ADENOM(PRECISION))
                (X(2),A,F(2));
      CALL OPNRES(PRECISION,A(I,J,1),ADENOM,MODULUS);
   END;
   PUT FILE(FT06F001) SKIP;
   CALL OPNRES(PRECISION,B(I,1),BDENOM,MODULUS);
END,

/****** REPORT B AND PRINT HEADING THAT INDICATES THE PRECISION. */
PUT FILE(FT06F001) SKIP(2) EDIT ('ALL ELEMENTS OF VECTOR B ARE ',
               'ZERO EXCEPT FOR B(',NON_ZERO_B_POSITION,
               '), WHICH IS 1.') (A,A,F(2),A);
PUT FILE(FT06F001) SKIP(3) EDIT('---  SOLUTION USING K-WORD RESIDUE ',
      'ARITHMETIC, WHERE K = ', PRECISION,'  ---') (A,A,F(3),A);
PUT FILE(FT06F001) SKIP EDIT(' ') (A);  /* FLUSH OUTPUT BUFFER.  */
TITLE = '                ---   SOLUTION VECTOR X(  ):  --- ';
```

```
/****** TRIANGULARIZE MATRIX, PERFORMING GAUSSIAN ELIMINATION. */
CALL GAUSS;


/****** BACK SOLVE TRIANGULARIZED MATRIX TO GET SOLUTIONS X(I). */
CALL BCKSOLV;


/****** PRINT SOLUTIONS X(I). */
DO I = 1 TO MATRIX_SIZE;
   CALL RATMAP(PRECISION,X(I,1),MODULUS,X(I,1),XDENOM);
   PUT STRING(SUBSTR(TITLE,40,2)) EDIT(I) (F(2));
   CALL OUTPUT(PRECISION,X(I,1),XDENOM,TITLE);
END;

CALL CLSOUT;    /* CLOSE OUTPUT PRINT FILE. */



GAUSS. PROC;
/*************************************************************************/
/*                    G A U S S                                          */
/*                                                                       */
/*       THIS INTERNAL PROCEDURE USES RESIDUE ARITHMETIC TO              */
/*   PERFORM STANDARD GAUSSIAN ELIMINATION TO TRIANGULARIZE THE          */
/*   AUGMENTED MATRIX.  IF A ZERO PIVOT IS ENCOUNTERED IN POSI-          */
/*   TION A(I,I) DURING THIS TRIANGULARIZATION, ROW SWITCHING IS         */
/*   PERFORMED TO OBTAIN A NON-ZERO PIVOT.  IF A NON-ZERO PIVOT          */
/*   CANNOT BE FOUND, THE SYSTEM OF LINEAR EQUATIONS HAS NO              */
/*   UNIQUE SOLUTION, AND EXECUTION CEASES.                              */
/*************************************************************************/

      /****    LOCAL VARIABLE DECLARATIONS    ****/
DCL M_JI(PRECISION) FIXED(31,0) BIN;     /* RATIO OF MATRIX ELE-   */
                                         /* ELEMENT A(J,I) TO      */
                                         /* PIVOT ELEMENT A(I,I)   */
DCL TEMP1(PRECISION) FIXED(31,0)         /* TEMPORARY STORAGE FOR  */
      BINARY;                            /* RESULTS FROM RESULT    */
DCL TEMP FIXED(31,0) BINARY;             /* TEMPORARY STORAGE FOR  */
                                         /* WORD OF MATRIX A OR    */
                                         /* VECTOR B               */
DCL ZERO_PIVOT BIT(1);                   /* SET TO TRUE IF CANDI-  */
                                         /* DATE FOR PIVOT IS ZERO */


/******** TRIANGULARIZE ALL ROWS OF THE AUGMENTED MATRIX.  */
DO I = 1 TO MATRIX_SIZE-1;


   /**** CHECK FOR ZERO PIVOT A(I,I).  IF PIVOT IS ZERO, SEARCH   */
   /**** ROWS J = I+1 TO N FOR A NON-ZERO PIVOT.                  */
   DO K = 1 TO PRECISION WHILE (A(I,I,K) = 0);
```

```
END;
IF K > PRECISION   /* IS THE PIVOT ZERO? */
   THEN DO;   /* CHECK ROWS I+1 TO N FOR A NON-ZERO PIVOT. */
            ZERO_PIVOT = '1'B;
            DO J = I+1 TO MATRIX_SIZE WHILE(ZERO_PIVOT);
               /*** CHECK NEXT PIVOT CANDIDATE.  */
               DO K = 1 TO PRECISION WHILE (A(J,I,K) = 0);
               END;
               IF K <= PRECISION
                  THEN ZERO_PIVOT = '0'B;
                  ELSE ZERO_PIVOT = '1'B;
            END;

            /***** IF A NON-ZERO PIVOT WAS BEEN FOUND IN ROW J,  */
            /***** SWAP ROWS I AND J.                            */
            IF ^(ZERO_PIVOT)
               THEN DO;
                       PUT FILE(FT06F001) SKIP LIST
                          ('SWITCHING ROWS TO FIND NON-ZERO PIVOT'),
                       J = J - 1;  /* RESET J TO ROW OF FIRST */
                                   /* NON-ZERO PIVOT.         */
                       /*** SWAP ROWS I AND J OF MATRIX A. */
                       DO K = 1 TO MATRIX_SIZE;  /* COLUMN POINTER */
                          /*** LOOP THROUGH ALL WORDS OF ELEMENT. */
                          DO L = 1 TO PRECISION;
                             TEMP = A(I,K,L); /* SAVE I-TH ROW  */
                                              /* ELEMENT WORD.  */
                             A(I,K,L) = A(J,K,L);
                             A(J,K,L) = TEMP;
                          END;
                       END;
                       /*** SWAP ELEMENTS I AND J OF VECTOR B -   */
                       /*** LOOP THOUGH ALL WORDS OF ELEMENT.     */
                       DO K = 1 TO PRECISION;
                          TEMP = B(I,K);   /* SAVE I-TH ELEMENT WORD. */
                          B(I,K) = B(J,K);
                          B(J,K) = TEMP;
                       END;
                    END;

               /***** A NON-ZERO PIVOT CANNOT BE FOUND.  **/
               /***** ABORT EXECUTION.                   **/
               ELSE DO;
                       PUT FILE(FT06F001) SKIP(3) EDIT ('********  ER',
                          'ROR --  NON-ZERO PIVOT COULD NOT BE ',
                          'FOUND DURING GAUSSIAN ELIMINATION.',
                          '********  SYSTEM OF LINEAR EQUATIONS ',
                          'HAS NO UNIQUE SOLUTION.') (A,A,A,SKIP,
                          A,A);
                       STOP;
                    END;
         END;
```

```
      /********* REDUCE ROWS J = I+1 TO N.  COMPUTE FACTOR  */
      /********* M(J,I) = A(J,I)/A(I,I) AND SUBTRACT PRODUCT */
      /********* OF M(J,I) AND ROW I FROM ROW J.             */
      DO J = I+1 TO MATRIX_SIZE;
         CALL RESDIV(PRECISION,A(J,I,1),A(I,I,1),MODULUS,M_JI);

         /*****  MULTIPLY EACH MATRIX COLUMN K OF PIVOTAL ROW I BY   */
         /*****  FACTOR M(J,I) AND SUBTRACT FROM COLUMN K OF ROW J.  */
         DO K = I TO MATRIX_SIZE;
            CALL RESMLT(PRECISION,A(I,K,1),M_JI,MODULUS,TEMP1);
            CALL RESSUB(PRECISION,A(J,K,1),TEMP1,MODULUS,A(J,K,1));
         END;

         /**** REDUCE AUGMENTED ELEMENT B(I).  */
         CALL RESMLT(PRECISION,B(I,1),M_JI,MODULUS,TEMP1);
         CALL RESSUB(PRECISION,B(J,1),TEMP1,MODULUS,B(J,1));
      END;

   END;


   RETURN;
   END GAUSS;



   BCKSOLV: PROC;
   /*****************************************************************/
   /*                  B C K S O L V                                */
   /*                                                               */
   /*        THIS INTERNAL PROCEDURE USES RESIDUE ARITHMETIC AND    */
   /*   BACKWARD SUBSTITUTION TO SOLVE A TRIANGULARIZED AUGMENTED   */
   /*   MATRIX A FOR THE SOLUTIONS X TO THE SYSTEM OF LINEAR EQUA-  */
   /*   TIONS AX = B.  IF A(N,N) IS 0, THE SYSTEM DOES NOT HAVE A   */
   /*   UNIQUE SOLUTION AND EXECUTION CEASES.                       */
   /*****************************************************************/

         /****    LOCAL VARIABLE DECLARATIONS    ****/
   DCL SUMPROD(PRECISION) FIXED(31,0) BIN; /* TEMPORARY VARIABLE TO    */
                                           /* HOLD SUM OF PRODUCTS     */
                                           /* OF COEFFICIENTS AND PRE- */
                                           /* VIOUSLY COMPUTED ROOTS   */
   DCL PRODRES(PRECISION) FIXED(31,0) BIN; /* TEMPORARY VARIABLE TO    */
                                           /* HOLD PRODUCT OF COEFFI-  */
                                           /* CIENT AND ROOT           */


   /****** CHECK FOR A(N,N) = 0.  IF A(N,N) = 0, THE SYSTEM OF  */
   /****** EQUATIONS HAS NO UNIQUE SOLUTION -- ABORT EXECUTION. */
   DO J = 1 TO PRECISION WHILE (A(MATRIX_SIZE,MATRIX_SIZE,J) = 0);
   END;
```

```
IF J > PRECISION
   THEN DO;
           PUT FILE(FTO6P001) SKIP(3) EDIT ('******* ERROR -- GAUS',
              'SIAN ELIMINATION PRODUCES A(N,N) = 0.','******* SYS',
              'TEM OF LINEAR EQUATIONS HAS NO UNIQUE SOLUTION.')
              (A,A,SKIP,A,A);
           STOP;
        END;

/******* DETERMINE ROOT X(N).  */
CALL RESDIV(PRECISION,D(MATRIX_SIZE,1),A(MATRIX_SIZE,MATRIX_SIZE,1),
           MODULUS,X(MATRIX_SIZE,1));


/******* EXTRACT EACH ADDITIONAL ROOT X(I).    */
DO I = MATRIX_SIZE - 1 TO 1 BY -1;

   /***** INITIALIZE ROOT X(I) TO TRIANGULARIZED ELEMENT D(I).  */
   DO K = 1 TO PRECISION;
      X(I,K) = D(I,K);
   END;

   /***** SUBTRACT FROM X(I) THE PRODUCT OF EACH COEFFICIENT AND  */
   /***** PREVIOUSLY COMPUTED ROOT.                               */
   SUMPROD = 0;  /* INITIALIZE SUM OF PRODUCTS TO 0. */
   /** COMPUTE SUM OF PRODUCTS OF COEFFICIENTS AND ROOTS.  */
   DO J = I+1 TO MATRIX_SIZE;  /* COMPUTE SUM OF PRODUCTS. */
      CALL RESMLT(PRECISION,A(I,J,1),X(J,1),MODULUS,PRODRES),
      CALL RESADD(PRECISION,SUMPROD,PRODRES,MODULUS,SUMPROD);
   END;
   /** SUBTRACT SUM OF PRODUCTS FROM ORIGINAL X(I).  */
   CALL RESSUB(PRECISION,X(I,1),SUMPROD,MODULUS,X(I,1)),

   /***** DIVIDE BY A(I,I) FOR ROOT X(I).  */
   CALL RESDIV(PRECISION,X(I,1),A(I,I,1),MODULUS,X(I,1));
END;


RETURN;
END BCKSOLV;



END LIN_EQ;
```

APPENDIX I

NEWTON-RAPHSON APPROXIMATION PROGRAMS FOR CUBE ROOT OF A (FORTRAN)

```
C=======================================================================
C=        THIS PROCEDURE USES MULTIPLE PRECISION RATIONAL ARITHMETIC
C=    AND THE NEWTON-RAPHSON METHOD TO FIND AN APPROXIMATION TO THE
C=    CUBE ROOT OF RATIONAL NUMBER A.  A FIXED NUMBER OF ITERATIONS
C=    (SPECIFIED BY THE USER) ARE PERFORMED.
C=
C=                  F(X) = X**3 - A = 0
C=                  F'(X) = 3 X**2
C=                  G(X) = X - (F(X)/F'(X))
C=                       = (2X + (A/X**2)) / 3
C=                       = (2/3)X + (A/3)/X**2
C=        THUS THE ITERATION FORMULA IS
C=                  P(K+1) = (2/3)P(K) + ((A/3)/P(K)**2) .
C=        THE INITIAL APPROXIMATION P(0) IS PROVIDED BY THE USER.
C=    NUMERATORS AND DENOMINATORS OF RATIONAL NUMBERS A AND P(0) ARE
C=    RESTRICTED TO VALUES WHICH FIT INTO A SINGLE 32-BIT WORD.
C=
C=    VARIABLES:
C=       PRECSN --- NUMBER OF FULLWORDS IN OPERANDS
C=       A      --- RATIONAL NUMBER OF WHICH CUBE ROOT IS DESIRED
C=       ADENOM --- DENOMINATOR OF ABOVE
C=       PK     --- VALUE OF THE KTH ITERATION OF NEWTON-RAPHSON METHOD
C=       PKDEN  --- DENOMINATOR OF ABOVE
C=       PKPLS1 --- VALUE OF K+1 ITERATION OF NEWTON-RAPHSON METHOD
C=       PK1DEN --- DENOMINATOR OF ABOVE
C=       ITERNO --- NUMBER OF ITERATIONS DESIRED BY THE USER
C=    CONSTANTS:
C=       TWOTHR, TTDEN --- CONSTANT = 2/3
C=       TITLE  --- A 64 CHARACTER TITLE FOR EACH NEWTON-RAPHSON ITERATION
C=
C=    SUBROUTINES CALLED:
C=       RATSET --- VERIFIES THAT "PRECSN" AND "K" OF ASSEMBLER SUPPORT
C=                  ROUTINES AGREE AND OPENS PRINT FILE"FT06F001"
C=       OPNRAT --- REDUCES A RATIONAL NUMBER FOR COMPUTATIONAL USE
C=       RATMLT --- MULTIPLIES TWO MULTIPLE PRECISION RATIONAL NUMBERS
C=       RATDIV --- DIVIDES TWO MULTIPLE PRECISION RATIONAL NUMBERS
C=       RATADD --- ADDS TWO MULTIPLE PRECISION RATIONAL NUMBERS
C=       OUTPUT --- CONVERTS TO EBCDIC CHARACTERS AND PRINTS A MULTIPLE
C=                  PRECISION RATIONAL NUMBER
C=       CLSOUT --- CLOSES OUTPUT PRINT FILE "FT06F001"
C=
C=    REQUIRED PROGRAM INPUT (FORMATTED):
C=       COLUMNS  1-10 --- NUMERATOR OF RATIONAL NUMBER "A" OF WHICH CUBE
C=                         ROOT IS TO BE TAKEN (POSITIVE OR NEGATIVE)
C=       COLUMNS 11-20 --- DENOMINATOR OF RATIONAL NUMBER "A" (POSITIVE)
C=       COLUMNS 21-30 --- NUMERATOR OF INITIAL APPROXIMATION P(0) TO
C=                         CUBE ROOT OF A (POSITIVE OR NEGATIVE)
```

```
C=       COLUMNS 31-40 =  DENOMINATOR OF P(0) (POSITIVE)
C=       COLUMNS 41-43 =  NUMBER OF ITERATIONS OF NEWTON-RAPHSON
C=                        METHOD TO BE PERFORMED
C=
C=    PROGRAM OUTPUT:
C=       RATIONAL NUMBER A AND THE INITIAL APPROXIMATION P(0) TO ITS
C=    CUBE ROOT ARE PRINTED.  THE TERM P(K) IS PRINTED FOR EACH
C=    ITERATION K = 1, 2, ...
C=
C=    PROGRAM MODIFICATIONS:
C=       THE PROGRAM MAY BE RE-RUN WITH A DIFFERENT PRECISION BY
C=    CHANGING "DATA" INITIALIZATIONS OF PRECSN, A, ADENOM, PK,
C=    PKDEN, TWOTHR, AND TTDEN AND BY RE-DIMENSIONING ARRAYS
C=    CORRESPONDINGLY.
C=================================================================

      IMPLICIT INTEGER (A-Z)
      DATA PRECSN/92/
      DIMENSION A(92),ADENOM(92)
      DIMENSION PK(92),PKDEN(92),PKPLS1(92),PK1DEN(92)
      DIMENSION TWOTHR(92),TTDEN(92)
      DIMENSION TITLE(16)
      DATA A/92*0/, ADENOM/92*0/, PK/92*0/, PKDEN/92*0/,
     1     TWOTHR/92*0/, TTDEN/92*0/
      DATA TITLE/' * TERM P(K) OF NEWTON-RAPHSON APPROXIMATION TO CUBE
     1 ROOT OF A */
C
C---------- VERIFY THAT "PRECSN" AGREES WITH ASSEMBLER ROUTINES' "K".
C---------- OPEN OUTPUT PRINT FILE "FT06F001" AND READ IN VALUES OF A
C---------- AND INITIAL TERM P(0).  INITIALIZE MULTIPLE PRECISION
C---------- VARIABLES.
      CALL RATSET(PRECSN)
      READ (5,50) A(PRECSN), ADENOM(PRECSN), PK(PRECSN),
     1            PKDEN(PRECSN), ITERNO
   50 FORMAT(4I10,I3)
      WRITE (6,100) A(PRECSN), ADENOM(PRECSN), PK(PRECSN),
     1              PKDEN(PRECSN), PRECSN
  100 FORMAT (' #-#.#-#  MULTIPLE PRECISION RATIONAL ARITHMETIC COM',
     1        'PUTATION OF CUBE ROOT OF A =',I10,'/',I10,'  #.#-#',
     2        /,T42,'P(0) =',I10,'/',I10,'.',//,T20,'**** SOLUTION ',
     3        'USING PRECISION-WORD ARITHMETIC, WHERE PRECISION = ',
     4        I3,' ****')
C
C---------- IF "A" OR INITIAL TERM P(0) IS NEGATIVE, EXTEND NEGATIVE
C---------- SIGN INTO LEFT WORDS OF NUMERATOR.  INITIALIZE MULTIPLE
C---------- PRECISION OPERANDS.
      PRMIN1 = PRECSN - 1
      IF (A(PRECSN) .GE. 0) GOTO 150
      DO 125 I = 1,PRMIN1
         A(I) = -1
  125 CONTINUE
C
  150 IF (PK(PRECSN) .GE. 0) GOTO 190
```

```
      DO 175 I = 1,PRCIN1
         PK(I) = -1
175   CONTINUE
C
190   TWOTHR(PRECSN) = 2
      TTDEN(PRECSN) = 3
      ADENOM(PRECSN) = 3 * ADENOM(PRECSN)
      CALL OPNRAT(PRECSN,A,ADENOM)
      CALL OPNRAT(PRECSN,PK,PKDEN)
C
C
C------------*** THIS LOOP PERFORMS A SPECIFIED NUMBER OF ITERATIONS OF THE
C------------*** NEWTON-RAPHSON METHOD, APPLIED TO FINDING THE CUBE ROOT OF
C------------*** A.
C
      DO 300 N = 1,ITERNO
C---- COMPUTE (2/3) * P(K) AND STORE IN PKPLS1, PK1DEN.
C---- COMPUTE (A/3) / (P(K) SQUARED) AND PLACE RESULT IN PK,PKDEN.
         CALL RATMLT(PRECSN,TWOTHR,TTDEN,PK,PKDEN,PKPLS1,
     1                PK1DEN)
         CALL RATMLT(PRECSN,PK,PKDEN,PK,PKDEN,PK,PKDEN)
         CALL RATDIV(PRECSN,A,ADENOM,PK,PKDEN,PK,PKDEN)
C---- COMPUTE P(K+1) = (2/3) P(K) + ((A/3) / P(K)**2) .
         CALL RATADD(PRECSN,PKPLS1,PK1DEN,PK,PKDEN,
     1               PKPLS1,PK1DEN)
C
C---- PRINT ITERATION P(K+1) AND COPY P(K+1) INTO P(K) FOR
C---- NEXT COMPUTATIONS.
         WRITE (6,200) N
200      FORMAT (////' *****  ITERATION K = ',I3,'  *****')
         CALL OUTPUT(PRECSN,PKPLS1,PK1DEN,TITLE)
C
         DO 275 I = 1, PRECSN
            PK(I) = PKPLS1(I)
            PKDEN(I) = PK1DEN(I)
275      CONTINUE
300   CONTINUE
C
C
C-------------  CLOSE OUTPUT PRINT FILE "FT06F001".
      CALL CLSOUT
      STOP
      END
```

```
C=============================================================
C=       THIS PROCEDURE USES MULTIPLE PRECISION SINGLE MODULUS RESI-
C=    DUE ARITHMETIC AND THE NEWTON-RAPHSON METHOD TO FIND AN APPROXI-
C=    MATION TO THE CUBE ROOT OF RATIONAL NUMBER "A."  A FIXED NUMBER
C=    OF ITERATIONS (SPECIFIED BY THE USER) ARE PERFORMED.
C=
```

```
C=                F(X) = X**3 - A = 0
C=                F'(X) = 3 X**2
C=                G(X) = X - (F(X)/F'(X))
C=                     = (2X + (A/X**2)) / 3
C=                     = (2/3)X + (A/3)/X**2
C=      THUS THE ITERATION FORMULA IS
C=                P(K+1) = (2/3)P(K) + ((A/3)/P(K)**2) .
C=     THE INITIAL APPROXIMATION P(0) IS PROVIDED BY THE USER.
C=  NUMERATORS AND DENOMINATORS OF RATIONAL NUMBERS A AND P(0) ARE
C=  RESTRICTED TO VALUES WHICH FIT INTO A SINGLE 32 BIT WORD.
C=
C=  VARIABLES:
C=     PRECSN -- NUMBER OF FULLWORDS IN OPERANDS
C=     MODULU -- MERSENNE PRIME USED IN RESIDUE ARITHMETIC WITH
C=               OPERANDS OF "PRECSN" WORDS
C=     A      -- RATIONAL NUMBER OF WHICH CUBE ROOT IS DESIRED
C=     ADENOM -- DENOMINATOR OF ABOVE
C=     PK     -- VALUE OF THE KTH ITERATION OF NEWTON-RAPHSON METHOD
C=     PKDEN  -- DENOMINATOR OF ABOVE
C=     PKPLS1 -- VALUE OF K+1 ITERATION OF NEWTON-RAPHSON METHOD
C=     ITERNO -- NUMBER OF ITERATIONS DESIRED BY THE USER
C=  CONSTANTS:
C=     TWOTHR, TTDEN -- CONSTANT = 2/3
C=     TITLE  -- A 64 CHARACTER TITLE FOR EACH NEWTON-RAPHSON ITERATION
C=
C=  SUBROUTINES CALLED:
C=     SETUP  -- DETERMINES MODULUS, ORDER-N OF ASSOCIATED FAREY
C=               FRACTIONS, AND OPENS OUTPUT PRINT FILE "FT06F001"
C=     OPNRES -- COMPUTES RESIDUE (MODULO "MODULU") OF RATIONAL NUMBER
C=     RESMLT -- PERFORMS THE RESIDUE MULTIPLICATION !A*B! MOD M
C=     RESDIV -- PERFORMS THE RESIDUE DIVISION !A/B! MOD M
C=     RESADD -- PERFORMS THE RESIDUE ADDITION !A+B! MOD M
C=     RATMAP -- PERFORMS INVERSE MAPPING FROM RESIDUE TO ORDER-N
C=               FAREY FRACTION
C=     OUTPUT -- CONVERTS TO EBCDIC CHARACTERS AND PRINTS A MULTIPLE
C=               PRECISION RATIONAL NUMBER
C=     CLSOUT -- CLOSES OUTPUT PRINT FILE "FT06F001"
C=
C=  REQUIRED PROGRAM INPUT (FORMATTED):
C=     COLUMNS  1-10 -  NUMERATOR OF RATIONAL NUMBER "A" OF WHICH CUBE
C=                      ROOT IS TO BE TAKEN (POSITIVE OR NEGATIVE)
C=     COLUMNS 11-20 -- DENOMINATOR OF RATIONAL NUMBER "A" (POSITIVE)
C=     COLUMNS 21-30 -  NUMERATOR OF INITIAL APPROXIMATION P(0) TO
C=                      CUBE ROOT OF A (POSITIVE OR NEGATIVE)
C=     COLUMNS 31-40 -- DENOMINATOR OF P(0) (POSITIVE)
C=     COLUMNS 41-43 -- NUMBER OF ITERATIONS OF NEWTON-RAPHSON
C=                      METHOD TO BE PERFORMED
C=
C=  PROGRAM OUTPUT:
C=     RATIONAL NUMBER A AND THE INITIAL APPROXIMATION P(0) TO ITS
C=  CUBE ROOT ARE PRINTED.  THE TERM P(K) IS PRINTED FOR EACH
C=  ITERATION K = 1, 2, ...
C=
```

```
C=    PROGRAM MODIFICATIONS:
C=       THE PROGRAM MAY BE RE-RUN WITH A DIFFERENT PRECISION BY
C=    CHANGING "DATA" INITIALIZATIONS OF PRECSN, A, ADENOM, PK,
C=    PKDEN, TWOTHR, AND TTDEN AND BY RE-DIMENSIONING ARRAYS
C=    CORRESPONDINGLY.
C=============================================================

      IMPLICIT INTEGER (A-Z)
      DATA PRECSN/133/
      DIMENSION MODULU(133), A(133),ADENOM(133)
      DIMENSION PK(133),PKDEN(133),PKPLS1(133)
      DIMENSION TWOTHR(133),TTDEN(133)
      DIMENSION TITLE(16)
      DATA A/133*0/, ADENOM/133*0/, PK/133*0/, PKDEN/133*0/,
     1     TWOTHR/133*0/, TTDEN/133*0/
      DATA TITLE/' * TERM P(K) OF NEWTON-RAPHSON APPROXIMATION TO CUBE
     1 ROOT OF A *'/
C
C---------- COMPUTE MODULUS AND ASSOCIATED ORDER N OF FAREY FRACTIONS.
C---------- OPEN OUTPUT PRINT FILE "FTO6F001" AND READ IN VALUES OF A
C---------- AND INITIAL TERM P(0).
      CALL SETUP(PRECSN,MODULU)
      READ (5,50) A(PRECSN), ADENOM(PRECSN), PK(PRECSN),
     1            PKDEN(PRECSN), ETERNO
   50 FORMAT(4I10,I3)
      WRITE (6,100) A(PRECSN), ADENOM(PRECSN), PK(PRECSN),
     1              PKDEN(PRECSN), PRECSN
  100 FORMAT (' *-*-*-*  MULTIPLE PRECISION RESIDUE ARITHMETIC COM',
     1        'PUTATION OF CUBE ROOT OF A =',I10,'/',I10,'  *-*-*-*',
     2        /,T42,'P(0) =',I10,'/',I10,'.',//,T20,'**** SOLUTION ',
     3        'USING PRECISION-WORD ARITHMETIC, WHERE PRECISION = ',
     4        I3,' ****')
C
C---------- IF "A" OR INITIAL TERM P(0) IS NEGATIVE, EXTEND NEGATIVE
C---------- SIGN INTO LEFT WORDS OF NUMERATOR.  INITIALIZE MULTIPLE
C---------- PRECISION OPERANDS.
      PRMIN1 = PRECSN - 1
      IF (A(PRECSN) .GE. 0) GOTO 150
      DO 125 I = 1,PRMIN1
         A(I) = -1
  125 CONTINUE
C
  150 IF (PK(PRECSN) .GE. 0) GOTO 190
      DO 175 I = 1,PRMIN1
         PK(I) = -1
  175 CONTINUE
C
  190 TWOTHR(PRECSN) = 2
      TTDEN(PRECSN) = 3
      ADENOM(PRECSN) = 3 * ADENOM(PRECSN)
      CALL OPNRES(PRECSN,TWOTHR,TTDEN,MODULU)
      CALL OPNRES(PRECSN,A,ADENOM,MODULU)
      CALL OPNRES(PRECSN,PK,PKDEN,MODULU)
```

```
C
C
C------------ *** THIS LOOP PERFORMS A SPECIFIED NUMBER OF ITERATIONS OF THE
C------------ *** NEWTON-RAPHSON METHOD, APPLIED TO FINDING THE CUBE ROOT OF
C------------ *** A.
C
      DO 300 N = 1,ITERNO
C---- COMPUTE (2/3) * P(K) AND STORE IN PKPLS1.
C---- COMPUTE (A/3) / (P(K) SQUARED) AND PLACE RESULT IN PK.
         CALL RESMLT(PRECSN,TWOTHR,PK,MODULU,PKPLS1)
         CALL RESMLT(PRECSN,PK,PK,MODULU,PK)
         CALL RESDIV(PRECSN,A,PK,MODULU,PK)
C---- COMPUTE P(K+1) = (2/3) P(K) + ((A/3) / P(K)**2) .
         CALL RESADD(PRECSN,PKPLS1,PK,MODULU,PKPLS1)
C
C---- PRINT ITERATION P(K+1) AND COPY P(K+1) INTO P(K) FOR
C---- NEXT COMPUTATIONS.
         WRITE (6,200) N
200      FORMAT (////' *****  ITERATION K = ',I3,'  *****')
         CALL RATMAP(PRECSN,PKPLS1,MODULU,PK,PKDEN)
         CALL OUTPUT(PRECSN,PK,PKDEN,TITLE)
C
         DO 275 I = 1, PRECSN
            PK(I) = PKPLS1(I)
275      CONTINUE
300   CONTINUE
C
C
C------------ CLOSE OUTPUT PRINT FILE "FT06F001".
      CALL CLSOUT
      STOP
      END
```

APPENDIX J

MILNE SIMPSON PROGRAMS FOR SOLVING Y' = -Y, Y(0) = 1 (PL1)

```
MLN_SMP: PROC OPTIONS(MAIN);
/*****************************************************************/
                    M L N _ S M P
```

THIS PROGRAM USES MULTIPLE PRECISION RATIONAL ARITHMETIC AND THE MILNE-SIMPSON METHOD TO SOLVE THE INITIAL VALUE PROBLEM

(1) $Y' = -Y, \qquad Y(0) = 1.$

FOR THIS INITIAL VALUE PROBLEM, THE MILNE SIMPSON METHOD PREDICTOR-CORRECTOR EQUATIONS BECOME

(2) $W(I+1) = W(I-3) + (4H/3)(-2W(I) + W(I-1) - 2W(I-2))$ PREDICTOR

(3) $W(I+1)' = W(I-1) + (H/3)(-W(I+1) - 4W(I) - W(I-1))$ CORRECTOR

ONE MAY COMBINE THE PREDICTOR-CORRECTOR EQUATIONS TO OBTAIN

(4) $W(I+1) = (-H/3)W(I-3) + (8H**2/9)W(I-2)$
$- (14H**2 + 3H - 9)/9)W(I-1) + (14(3H - 12)/9)W(I)$.

STARTING VALUES FOR THE MILNE-SIMPSON METHOD ARE PROVIDED BY THE EULER METHOD,

(5) $W(I+1) = W(I)(1 - H)$,

OR BY THE RUNGE KUTTA FOURTH-ORDER METHOD, WHICH FOR THE INITIAL VALUE PROBLEM (1) BECOMES

(6) $W(I+1) = W(I) + (K1 + 2K2 + 2K3 + K4)/6$
$= W(I)(1 - H(1 - H/2 + H**2/6 - H**3/24))$.

FOR EACH ITERATION I, WE HAVE
(7) $T(I+1) = (I+1)H$.

TO SHORTEN RATIONAL COMPUTATIONS, VALUES OF COEFFICIENTS ("W_I_FACTOR") IN (4) AND THE MULTIPLIER ("START_FACTOR") IN (5) AND (6) ARE PRE-COMPUTED FROM THE CHOSEN STEP SIZE H AND READ AS INPUT DATA. NUMERATORS AND DENOMINATORS OF EACH OF THESE QUANTITIES ARE THUS PRESUMED TO FIT IN SINGLE 32-BIT INTEGER WORDS.

REQUIRED INPUT DATA, IN ORDER, UNFORMATTED

START_TYPE -- A 5-CHARACTER VALUE INDICATING METHOD OF OBTAINING STARTING VALUES:
'EULER' -- EULER METHOD
'RUNGE' -- RUNGE-KUTTA FOURTH ORDER METHOD

H -- NUMERATOR OF STEP SIZE

H_DENOM -- DENOMINATOR OF STEP SIZE

```
        START_FACTOR          -- NUMERATOR OF MULTIPLIER FOR COMPUTING
                                 STARTING VALUES
        START_FACTOR_DENOM    -- DENOMINATOR OF MULTIPLIER FOR COMPUTING
                                 STARTING VALUES
        W_I_FACTOR(0)         -- NUMERATOR OF COEFFICIENT OF W(I) IN
                                 EQUATION (4)
        W_I_FACTOR_DENOM.(0)  -- DENOMINATOR OF COEFFICIENT OF W(I) IN
                                 EQUATION (4)
        W_I_FACTOR(1)         -- NUMERATOR OF COEFFICIENT OF W(I-1) IN
                                 EQUATION (4)
        W_I_FACTOR_DENOM(1)   -- DENOMINATOR OF COEFFICIENT OF W(I-1) IN
                                 EQUATION (4)
        W_I_FACTOR(2)         -- NUMERATOR OF COEFFICIENT OF W(I-2) IN
                                 EQUATION (4)
        W_I_FACTOR_DENOM.(2)  -- DENOMINATOR OF COEFFICIENT OF W(I-2) IN
                                 EQUATION (4)
        W_I_FACTOR(3)         -- NUMERATOR OF COEFFICIENT OF W(I-3) IN
                                 EQUATION (4)
        W_I_FACTOR_DENOM(3)   -- DENOMINATOR OF COEFFICIENT OF W(I-3) IN
                                 EQUATION (4)

PROGRAM MODIFICATIONS:
        IN ADDITION TO PROVIDING THE DATA LISTED ABOVE FOR EACH
PROGRAM EXECUTION, THE PRECISION AND NUMBER OF ITERATIONS ("ITERATION_
NO") MAY BE CHANGED BY MODIFYING THE INITIALIZATIONS IN DECLARATIONS
OF THESE VARIABLES.  A MAXIMUM PRECISION OF 101 IS ASSUMED IN INITIAL-
IZATIONS OF W_I, W_I_DENOM, W_I_FACTOR, W_I_FACTOR_DENOM, H, H_DENOM,
START_FACTOR, AND START_FACTOR_DENOM.

PROGRAM OUTPUT:
        THE VALUE OF H, THE PRECISION, AND METHOD OF OBTAINING STARTING
VALUES ARE REPORTED.  THE VALUES OF I, T(I), AND W(I) ARE REPORTED
FOR EACH ITERATION.

SUBROUTINES CALLED:
        RATSET -- VERIFIES THAT "PRECISION" AGREES WITH "K" OF ASSEMBLER
                  SUPPORT ROUTINES AND OPENS PRINT FILE "FT06F001"
        CLSOUT -- CLOSES OUTPUT PRINT FILE FT06F001
        OPNRAT -- REDUCES RATIONAL NUMBER P/Q TO STANDARD FORM. (REDUCED
                  WITH NUMERATOR CONTAINING SIGN)
        OUTPUT -- CONVERTS TO EBCDIC CHARACTERS AND PRINTS A MULTIPLE
                  PRECISION RATIONAL NUMBER
        RATADD -- ADDS TWO MULTIPLE PRECISION RATIONAL NUMBERS (REDUCED
                  RESULT)
        RATMLT -- MULTIPLIES TWO RATIONAL NUMBERS (NUMERATOR AND
                  DENOMINATOR OF OPERANDS AND PRODUCT IN H-WORDS)
*******************************************************************/

DEFAULT RANGE(*) VALUE(FIXED(31,0) BINARY);
/*******  PROCEDURE DECLARATIONS  *******/
DCL RATSET ENTRY OPTIONS(ASSEMBLER,INTER);
DCL CLSOUT ENTRY OPTIONS(ASSEMBLER,INTER);
DCL OUTPUT ENTRY OPTIONS(ASSEMBLER,INTER);
```

```
DCL OPNRAT ENTRY OPTIONS(ASSEMBLER,INTER);
DCL RATADD ENTRY OPTIONS(ASSEMBLER,INTER);
DCL RATMLT ENTRY OPTIONS(ASSEMBLER,INTER);

/*** DECLARATIONS OF PRECISION, # OF ITERATIONS, AND PRINT FILE ***/
DCL PRECISION FIXED(31,0) BIN     /* # OF FULLWORDS (K) IN          */
                INIT(22),         /* MULTIPLE PRECISION OPERANDS    */
DCL ITERATION_NO FIXED(31,0)      /* NUMBER OF ITERATIONS OF        */
   BIN INIT(75);                  /* MILNE-SIMPSON METHOD           */
DCL TITLE CHAR(64) ;              /* TITLE FOR CALL TO OUTPUT       */
DCL FT06F001 FILE PRINT EXT,      /* PRINT FILE ESTABLISHED IN      */
                                  /* ASSEMBLER SUPPORT ROUTINES     */

/*** ARRAY DECLARATIONS OF H, START_FACTOR, W_I, AND W_I_FACTOR ***/
DCL H(PRECISION) FIXED(31,0)      /* STEP SIZE H = (B - A) / N      */
   BIN INIT((101)0),
DCL H_DENOM(PRECISION) FIXED(31,0) /* DENOMINATOR OF H              */
   BIN INIT((101)0),
DCL START_FACTOR(PRECISION)       /* MULTIPLICATIVE FACTOR FOR      */
   FIXED(31,0) BIN INIT((101)0),  /* OBTAINING STARTING VALUES      */
DCL START_FACTOR_DENOM(PRECISION) /* FOR MILNE-SIMPSON METHOD       */
   FIXED(31,0) BIN INIT((101)0);  /* FROM EULER OR RUNGE-KUTTA      */
                                  /* METHOD:                        */
                                  /* W(I) = W(I-1) * START_FACTOR   */
DCL W_I(0:3,PRECISION) FIXED(31,0) /* ARRAY CONTAINING CURRENT      */
       BIN INIT((404)0);          /* ITERATION VALUE (W_I(0)) AND   */
                                  /* THREE PREVIOUS VALUES          */
                                  /* (W(I-3) IS STORED IN W_I(3).   */
DCL W_I_DENOM(0:3,PRECISION)      /* DENOMINATORS OF W_I            */
   FIXED(31,0) BIN INIT((404)0),
DCL W_I_FACTOR(0:3,PRECISION)     /* FACTORS BY WHICH TO MULTIPLY   */
   FIXED(31,0) BIN INIT((404)0);  /* RESPECTIVE W_I TO GET W(I-1)   */
DCL W_I_FACTOR_DENOM(0:3,PRECISION) /* DENOMINATORS OF W_I_FACTOR   */
   FIXED(31,0) BIN INIT((404)0);

/*** STORAGE FOR RATIONAL SUMS & PRODUCTS IN MILN.-SIMPS. COMPUT. ***/
DCL PRODUCT(PRECISION)            /* PRODUCT IN MILNE-SIMPSON       */
   FIXED(31,0) BIN;               /* COMPUTATION OF W(I-1)          */
DCL PRODUCT_DENOM(PRECISION)      /* ASSOCIATED DENOMINATOR         */
   FIXED(31,0) BIN;
DCL SUM(PRECISION)                /* SUM IN MILNE-SIMPSON COMPU-    */
   FIXED(31,0) BIN;               /* TATION OF W(I-1)               */
DCL SUM_DENOM(PRECISION)          /* ASSOCIATED DENOMINATOR         */
   FIXED(31,0) BIN;

DCL H_FLOAT FLOAT;                /* SINGLE PRECISION FLOATING      */
                                  /* POINT REPRESENTATION OF H      */
DCL T_I FLOAT INIT(0.0);          /* PRODUCT OF ITERATION NUMBER    */
                                  /* I AND STEP SIZE H              */
DCL START_TYPE CHAR(5);           /* METHOD USED TO GET STARTING    */
                                  /* VALUES.  "EULER" OR "RUNGE"    */
```

```
/******* VERIFY THAT "PRECISION" AGREES WITH "K" OF ASSEMBLER SUB-  */
/******* PORT ROUTINES AND OPEN OUTPUT PRINT FILE "FT06F001."       */
/******* READ IN CHOICE OF STARTING METHOD, VALUE OF H, AND THE     */
/******* MULTIPLICATIVE FACTOR "START_FACTOR" FOR USE IN THE RUNGE- */
/******* KUTTA METHOD.                                              */
CALL RATSET(PRECISION);
PUT FILE(FT06F001) EDIT ('***** MILNE-SIMPSON SOLUTION TO THE ',
       'INITIAL VALUE PROBLEM  Y'' = -Y,  Y(0) = 1.  *****')
       (A,A);
GET LIST(START_TYPE,H(PRECISION),H_DENOM(PRECISION),START_FACTOR
         (PRECISION),START_FACTOR_DENOM(PRECISION));

/*******  PRINT APPROPRIATE HEADINGS.  ****/
IF START_TYPE = 'EULER'
   THEN PUT FILE(FT06F001) EDIT('EULER METHOD IS USED TO GET STARTING',
        ' VALUES -- W(I+1) = W(I) (1 - H).') (COL(9),A,A);
   ELSE PUT FILE(FT06F001) EDIT('RUNGE KUTTA FOURTH ORDER METHOD IS ',
        'USED TO GET STARTING VALUES -- W(I+1) = W(I) (1 - H (1 - H/2',
        ' + H**2/6 - H**3/24)).') (COL(1),A,A,A);
PUT FILE(FT06F001) SKIP(2) EDIT('--- SOLUTION USING K WORD RATIONAL ',
       'ARITHMETIC, WHERE K = ',PRECISION,', AND STEP SIZE H = ',
       H(PRECISION),'/',H_DENOM(PRECISION),' --- ') (A,A,F(3),A,F(9),
                                                    A,F(9),A);
PUT FILE(FT06F001) SKIP(2);

/*******  COMPLETE INITIALIZATIONS OF H AND START_FACTOR (EXTEND  */
/*******  A NEGATIVE SIGN INTO LEFT WORDS), AND COMPUTE REDUCED   */
/*******  STANDARD RATIONAL FORMS OF H AND START_FACTOR.          */
H_FLOAT = H(PRECISION)/FLOAT(H_DENOM(PRECISION));
IF H(PRECISION) < 0
   THEN DO I = 1 TO PRECISION-1;
           H(I) = -1;
        END;
IF START_FACTOR(PRECISION) < 0
   THEN DO I = 1 TO PRECISION-1;
           START_FACTOR(I) = -1;
        END;
CALL OPNRAT(PRECISION,H,H_DENOM);
CALL OPNRAT(PRECISION,START_FACTOR,START_FACTOR_DENOM);

/***** INITIALIZE W(I) TO INITIAL VALUE PROBLEM, Y' = -Y,Y(0) = 1. */
/***** INITIAL VALUE W(0) IS STORED IN W_I(3).                     */
W_I(3,*) = 0;
W_I(3,PRECISION) = 1;
W_I_DENOM(3,*) = 0;
W_I_DENOM(*,PRECISION) = 1;  /* INITIALIZE ALL DENOMINATORS TO 1. */
TITLE = '     ITERATION I =    .  T(I) =          .  VALUE OF W(I) IS.  ';


/******** USE EULER'S METHOD OR THE RUNGE KUTTA FOURTH ORDER    */
/******** METHOD (WHICHEVER IS SPECIFIED) TO GET 3 STARTING     */
/******** VALUES FOR USE BY THE MILNE-SIMPSON METHOD.  VALUES   */
```

```
/********** FOR THE INITIAL VALUE PROBLEM Y' = -Y, Y(0) = 1          */
/********** ARE GIVEN BY                                             */
/**********  W(I-1) = W(I) (1 - H)                  (EULER'S METHOD)  */
/**********  W(I-1) = W(I) (1 - H(1 - H/2 + H**2/6   H**3/24)) )      */
/**********                                        (RUNGE-KUTTA METHOD)*/
/**********                                                          */
/**********     FIRST  ITERATION W(1) IS STORED IN W_I(2).           */
/**********     SECOND ITERATION W(2) IS STORED IN W_I(1).           */
/**********     THIRD  ITERATION W(3) IS STORED IN W_I(0).           */
DO I = 2 TO 0 BY -1;
   T_I = (3-I) * H_FLOAT;
   CALL RATMLT(PRECISION,W_I(I+1,1),W_I_DENOM(I+1,1),START_FACTOR,
               START_FACTOR_DENOM,W_I(I,1),W_I_DENOM(I,1));
   PUT STRING (SUBSTR(TITLE,21,2)) EDIT(3-I)(F(2));
   PUT STRING (SUBSTR(TITLE,33,8)) EDIT(T_I) (F(8,4));
   CALL OUTPUT(PRECISION,W_I(I,1),W_I_DENOM(I,1),TITLE);
   PUT FILE(FT06F001) EDIT ('COMPARISON VALUE OF E**(-',T,') ',
        'SINGLE PRECISION, IS ',EXP(-T_I)) (A,A,E(15,8,0));
   PUT FILE(FT06F001) SKIP;  /* FLUSH OUTPUT BUFFER. */
END;


/**********  READ IN COEFFICIENTS FOR THE MILNE-SIMPSON METHOD **/
/**********  AND COMPUTE RESIDUE EQUIVALENTS.                  **/
DO I = 0 TO 3;
   GET LIST(W_I_FACTOR(I,PRECISION),W_I_FACTOR_DENOM(I,PRECISION));
   /***** EXTEND NEGATIVE NUMERATOR SIGN INTO LEFT K-1 WORDS. */
   IF W_I_FACTOR(I,PRECISION) < 0
      THEN DO J = 1  TO PRECISION - 1;
              W_I_FACTOR(I,J) = -1;
           END;
   CALL OPNRAT(PRECISION,W_I_FACTOR(I,1),W_I_FACTOR_DENOM(I,1));
END;
PUT FILE(FT06F001) SKIP;    /* FLUSH OUTPUT BUFFER.  */



/********** USE THE MILNE-SIMPSON METHOD TO FIND W(I) APPROXI-   */
/********** MATIONS TO THE SOLUTION OF THE INITIAL VALUE         */
/********** PROBLEM  Y' = -Y,  Y(0) = 1.  TO SIMPLIFY THE COM-   */
/********** PUTATION, THE PREDICTOR-CORRECTOR STEPS ARE COM-     */
/********** BINED, AND THE MULTIPLICATIVE FACTORS FOR THE 4      */
/********** PREVIOUS ITERATIONS ARE PRE-DETERMINED.  W(I 1) IS   */
/********** COMPUTED BY THE RELATION                             */
/*                                                               */
/*   W(I-1) =  ((8H**2 - 12H)/9)      * W(I)                     */
/*             - ((4H**2 + 3H - 9)/9) * W(I-1)                   */
/*             - ((8H**2) / 9)        * W(I-2)                   */
/*             - (H/3)                * W(I-3)                   */
/*                                                               */

DO I = 4 TO ITERATION_NO;
   /***** COMPUTE W(I-1). *****/
```

```
      T_I = I * M_FLOAT;
      CALL RATMLT(PRECISION,W_I(0,1),W_I_DENOM(0,1),W_I_FACTOR(0,1),
                  W_I_FACTOR_DENOM(0,1),SUM,SUM_DENOM);
      DO J = 1 TO 3;
         CALL RATMLT(PRECISION,W_I(J,1),W_I_DENOM(J,1),W_I_FACTOR(J,1),
                     W_I_FACTOR_DENOM(J,1),PRODUCT,PRODUCT_DENOM);
         CALL RATADD(PRECISION,SUM,SUM_DENOM,PRODUCT,PRODUCT_DENOM,
                     SUM,SUM_DENOM);
      END;

      /****** PRINT NEWLY COMPUTED W(I+1) (STORED IN SUM, SUM_DENOM). */
      PUT STRING (SUBSTR(TITLE,21,2)) EDIT(I)(F(2));
      PUT STRING (SUBSTR(TITLE,33,8)) EDIT(T_I) (F(8,4));
      CALL OUTPUT(PRECISION,SUM,SUM_DENOM,TITLE);
      PUT FILE(FT06F001) SKIP(2) EDIT ('COMPARISON VALUE OF EXP(-T), ',
          'SINGLE PRECISION, IS ',EXP(-T_I)) (COL(1),A,A,E(15,8,0));
      PUT FILE(FT06F001) SKIP;  /* FLUSH OUTPUT BUFFER. */

      /****** MOVE CURRENT ITERATION RESULTS DOWN ONE POSITION TO SET  */
      /****** UP FOR NEXT ITERATION.  LAST FOUR VALUES OF W(I) ARE     */
      /****** SAVED FOR USE IN COMPUTING NEXT W(I).                    */

      /**** MOVE W(2) THROUGH W(0) DOWN ONE POSITION.   */
      DO J = 3 TO 1 BY -1;
         DO K = 1 TO PRECISION;
            W_I(J,K) = W_I(J-1,K);
            W_I_DENOM(J,K) = W_I_DENOM(J-1,K);
         END;
      END;
      /**** COPY COMPUTED W(I+1) INTO W(0) POSITION. */
      DO K = 1 TO PRECISION;
         W_I(0,K) = SUM(K);
         W_I_DENOM(0,K) = SUM_DENOM(K);
      END;
   END;   /*  TO MILNE-SIMPSON LOOP  */



CALL CLSOUT;  /* CLOSE OUTPUT PRINT FILE. */

END MLN_SMP;



MLN_SMP: PROC OPTIONS(MAIN);
/*****************************************************************
                  - - - -    M L N _ S M P    - - - -

      THIS PROGRAM USES MULTIPLE PRECISION RESIDUE ARITHMETIC
AND THE MILNE-SIMPSON METHOD TO SOLVE THE INITIAL VALUE PROBLEM

(1)    Y' = Y,          Y(0) = 1.
```

FOR THIS INITIAL VALUE PROBLEM, THE MILNE-SIMPSON METHOD PREDICTOR-CORRECTOR EQUATIONS BECOME

(2) W(I+1) = W(I-3) + (4H/3) (2U(I) - U(I-1) + 2U(I-2)) PREDICTOR

(3) W(I+1)' = W(I-1) + (H/3) (U(I-1) + 4U(I) + U(I+1)) CORRECTOR

ONE MAY COMBINE THE PREDICTOR-CORRECTOR EQUATIONS TO OBTAIN

(4) W(I+1) = (-H/3)W(I-3) + (8H**2/9)U(I-2)
- ((4H**2 + 3H - 9)/9)U(I-1) + (H(8H - 12)/9)U(I) .

STARTING VALUES FOR THE MILNE SIMPSON METHOD ARE PROVIDED BY THE EULER METHOD,

(5) W(I+1) = W(I) (1 + H) ,

OR BY THE RUNGE-KUTTA FOURTH-ORDER METHOD, WHICH FOR THE INITIAL VALUE PROBLEM (1) BECOMES

(6) W(I+1) = W(I) + (K1 + 2K2 + 2K3 + K4)/6
= W(I) (1 + H(1 + H/2 + H**2/6 + H**3/24)) .

FOR EACH ITERATION I, WE HAVE
(7) T(I+1) = (I+1) H .

TO SHORTEN RESIDUE COMPUTATIONS, VALUES OF COEFFICIENTS ("W_I_FACTOR") IN (4) AND THE MULTIPLIER ("START_FACTOR") IN (5) AND (6) ARE PRE-COMPUTED FROM THE CHOSEN STEP SIZE H AND READ AS INPUT DATA. NUMERATORS AND DENOMINATORS OF EACH OF THESE QUANTITIES ARE THUS PRESUMED TO FIT IN SINGLE 32-BIT INTEGER WORDS.

REQUIRED INPUT DATA, IN ORDER, UNFORMATTED:

START_TYPE -- A 5-CHARACTER VALUE INDICATING METHOD OF OBTAINING STARTING VALUES:
'EULER' - EULER METHOD
'RUNGE' - RUNGE-KUTTA FOURTH ORDER METHOD

H -- NUMERATOR OF STEP SIZE

H_DENOM -- DENOMINATOR OF STEP SIZE

START_FACTOR -- NUMERATOR OF MULTIPLIER FOR COMPUTING STARTING VALUES

START-FACTOR_DENOM -- DENOMINATOR OF MULTIPLIER FOR COMPUTING STARTING VALUES

W_I_FACTOR(0) - NUMERATOR OF COEFFICIENT OF W(I) IN EQUATION (4)

W_I_FACTOR_DENOM(0) -- DENOMINATOR OF COEFFICIENT OF W(I) IN EQUATION (4)

W_I_FACTOR(1) -- NUMERATOR OF COEFFICIENT OF W(I-1) IN EQUATION (4)

W_I_FACTOR_DENOM(1) - DENOMINATOR OF COEFFICIENT OF W(I-1) IN EQUATION (4)

W_I_FACTOR(2) -- NUMERATOR OF COEFFICIENT OF W(I-2) IN EQUATION (4)

```
        W_I_FACTOR_DENOM(2) -  DENOMINATOR OF COEFFICIENT OF W(I-2) IN
                               EQUATION (4)
        W_I_FACTOR(3)       -- NUMERATOR OF COEFFICIENT OF W(I-3) IN
                               EQUATION (4)
        W_I_FACTOR_DENOM(3) -- DENOMINATOR OF COEFFICIENT OF W(I-3) IN
                               EQUATION (4)

PROGRAM MODIFICATIONS:
        IN ADDITION TO PROVIDING THE DATA LISTED ABOVE FOR EACH
PROGRAM EXECUTION, THE PRECISION AND NUMBER OF ITERATIONS ("ITERATION_
NO") MAY BE CHANGED BY MODIFYING THE INITIALIZATIONS IN DECLARATIONS
OF THESE VARIABLES.  A MAXIMUM PRECISION OF 101 IS ASSUMED IN INITIAL-
IZATIONS OF W_I, W_I_DENOM, W_I_FACTOR, W_I_FACTOR_DENOM, H, H_DENOM,
START_FACTOR, AND START_FACTOR_DENOM.

PROGRAM OUTPUT:
        THE VALUE OF H, THE PRECISION, AND METHOD OF OBTAINING STARTING
VALUES ARE REPORTED.  THE VALUES OF I, T(I), AND W(I) ARE REPORTED
FOR EACH ITERATION.

SUBROUTINES CALLED:
        SETUP  -- INITIALIZES MERSENNE PRIME MODULUS AND OPENS OUTPUT
                  PRINT FILE "FT06F001"
        CLSOUT -- CLOSES OUTPUT PRINT FILE FT06F001
        OPNRES -- COMPUTES THE RESIDUE EQUIVALENT OF MULTIPLE PRE-
                  CISION RATIONAL NUMBER
        RATMAP -- MAPS RESIDUE TO ORDER-N FAREY FRACTION EQUIVALENT
        OUTPUT -- CONVERTS TO EBCDIC CHARACTERS AND PRINTS A MULTIPLE
                  PRECISION RATIONAL NUMBER
        RESADD -- PERFORMS THE MULTIPLE PRECISION RESIDUE ADDITION
                  !A+B! MOD N
        RESMLT -- PERFORMS THE MULTIPLE PRECISION RESIDUE MULTIPLICA-
                  TION !A*B! MOD N
*********************************************************************/

DEFAULT RANGE(*) VALUE(FIXED(31,0) BINARY);
/*******  PROCEDURE DECLARATIONS  *******/
DCL SETUP ENTRY OPTIONS(ASSEMBLER,INTER);
DCL CLSOUT ENTRY OPTIONS(ASSEMBLER,INTER);
DCL OUTPUT ENTRY OPTIONS(ASSEMBLER,INTER);
DCL OPNRES ENTRY OPTIONS(ASSEMBLER,INTER);
DCL RATMAP ENTRY OPTIONS(ASSEMBLER,INTER);
DCL RESADD ENTRY OPTIONS(ASSEMBLER,INTER);
DCL RESMLT ENTRY OPTIONS(ASSEMBLER,INTER);

/****  DECLARATIONS OF THE PRECISION, MODULUS, AND PRINT FILE  ****/
DCL PRECISION FIXED(31,0) BIN     /* # OF FULLWORDS (K) IN        */
              INIT(69);           /* MULTIPLE PRECISION OPERANDS */
DCL MODULUS(PRECISION)            /* K-WORD MERSENNE PRIME        */
      FIXED(31,0) BIN;            /* MODULUS                      */
DCL ITERATION_NO FIXED(31,0)      /* NUMBER OF ITERATIONS OF      */
   BIN INIT(75);                  /* NEWTON-SIMPSON METHOD        */
DCL TITLE CHAR(64) ;              /* TITLE FOR CALL TO OUTPUT     */
```

```
DCL FT06F001 FILE PRINT EXT,        /* PRINT FILE ESTABLISHED IN    */
                                    /* ASSEMBLER SUPPORT ROUTINES   */

/***  ARRAY DECLARATIONS OF H, START_FACTOR, W_I, AND W_I_FACTOR  ***/
DCL H(PRECISION) FIXED(31,0)        /* STEP SIZE H = (B - A) / N    */
   BIN INIT((101)0);
DCL H_DENOM(PRECISION) FIXED(31,0)  /* DENOMINATOR OF H             */
   BIN INIT((101)0);
DCL START_FACTOR(PRECISION)         /* MULTIPLICATIVE FACTOR FOR    */
   FIXED(31,0) BIN INIT((101)0);    /* OBTAINING STARTING VALUES    */
DCL START_FACTOR_DENOM(PRECISION)   /* FOR MILNE-SIMPSON METHOD     */
   FIXED(31,0) BIN INIT((101)0);    /* FROM EULER OR RUNGE-KUTTA    */
                                    /* METHOD:                      */
                                    /* W(I) = W(I-1) * START_FACTOR */
DCL W_I(0:3,PRECISION) FIXED(31,0)  /* ARRAY CONTAINING CURRENT     */
       BIN INIT((404)0);            /* ITERATION VALUE (W_I(0)) AND */
                                    /* THREE PREVIOUS VALUES        */
                                    /* (W(I-3) IS STORED IN W_I(3). */
DCL W_I_DENOM(PRECISION)            /* DENOMINATOR OF ANY W_I       */
   FIXED(31,0) BIN INIT((101)0);    /* -- RESET AS NECESSARY        */
DCL W_I_FACTOR(0:3,PRECISION)       /* FACTORS BY WHICH TO MULTIPLY */
   FIXED(31,0) BIN INIT((404)0);    /* RESPECTIVE W_I TO GET W(I-1) */
DCL W_I_FACTOR_DENOM(PRECISION)     /* DENOM. OF ANY W_I_FACTOR     */
   FIXED(31,0) BIN INIT  ((101)0);  /* -- RESET AS NECESSARY        */

/** STORAGE FOR RESIDUE SUMS AND PRODUCTS AND RATIONAL EQUIVALENTS **/
DCL PRODUCT(PRECISION) FIXED(31,0)  /* RESIDUE PRODUCT IN MILNE-    */
   BIN;                             /* SIMPSON COMPUTATION          */
DCL PRODUCT_DENOM(PRECISION)        /* DENOMINATOR OF RATIONAL      */
   FIXED(31,0) BIN;                 /* EQUIVALENT OF W(I-1)         */
DCL SUM(PRECISION) FIXED(31,0)      /* RESIDUE SUM IN COMPUTING     */
   BIN;                             /* STARTING VALUES OR IN MILNE- */
                                    /* SIMPSON COMPUTATION          */
DCL SUM_DENOM(PRECISION)            /* DENOMINATOR OF RATIONAL      */
   FIXED(31,0) BIN;                 /* EQUIVALENT OF W(I)           */

DCL H_FLOAT FLOAT;                  /* SINGLE PRECISION FLOATING    */
                                    /* POINT REPRESENTATION OF H    */
DCL T_I FLOAT INIT(0.0);            /* PRODUCT OF ITERATION NUMBER  */
                                    /* I AND STEP SIZE H            */
DCL START_TYPE CHAR(5);             /* METHOD USED TO GET STARTING  */
                                    /* VALUES:  "EULER" OR "RUNGE"  */



/****** DETERMINE THE LARGEST MERSENNE PRIME THAT WILL FIT IN    */
/****** "PRECISION" WORDS.  THIS PRIME BECOMES THE MODULUS.      */
/****** OPEN OUTPUT PRINT FILE "FT06F001" AND READ IN CHOICE     */
/****** OF STARTING METHOD, VALUE OF H, AND THE MULTIPLICATIVE   */
/****** FACTOR "START_FACTOR" FOR USE IN THE EULER OR RUNGE-     */
/****** KUTTA METHOD.                                           */
CALL SETUP(PRECISION,MODULUS);
PUT FILE(FT06F001) EDIT ('1','A','A',  MILNE-SIMPSON SOLUTION TO THE ',
```

```
        'INITIAL VALUE PROBLEM  Y' = -Y,  Y(0) = 1.  -*-*-*-')
        (A,A),
GET LIST(START_TYPE,H(PRECISION),H_DENOM(PRECISION),START_FACTOR
         (PRECISION),START_FACTOR_DENOM(PRECISION));

/******* PRINT APPROPRIATE HEADINGS. ******/
IF START_TYPE = 'EULER'
   THEN PUT FILE(FT06F001) EDIT('EULER METHOD IS USED TO GET STARTING',
        ' VALUES -- W(I-1) = W(I) (1 - H).') (COL(9),A,A);
   ELSE PUT FILE(FT06F001) EDIT('RUNGE-KUTTA FOURTH ORDER METHOD IS ',
        'USED TO GET STARTING VALUES -- W(I-1) = W(I) (1 - H (1 - H/2',
        ' + H**2/6 - H**3/24)).') (COL(1),A,A,A);
PUT FILE(FT06F001) SKIP(3) EDIT('- - SOLUTION USING K-WORD RESIDUE ',
       'ARITHMETIC, WHERE K = ',PRECISION,', AND STEP SIZE H = ',
       H(PRECISION),'/',H_DENOM(PRECISION),' --') (A,A,F(3),A,F(9),
                                                   A,F(9),A);
PUT FILE(FT06F001) SKIP(2);

/******* COMPLETE INITIALIZATIONS OF H AND START_FACTOR (EXTEND  **/
/******* A NEGATIVE SIGN INTO LEFT WORDS), AND COMPUTE RESIDUE   **/
/******* EQUIVALENTS OF H AND START_FACTOR.                      **/
H_FLOAT = H(PRECISION)/FLOAT(H_DENOM(PRECISION));
IF H(PRECISION) < 0
   THEN DO I = 1 TO PRECISION-1;
           H(I) = -1;
        END;
IF START_FACTOR(PRECISION) < 0
   THEN DO I = 1 TO PRECISION-1;
           START_FACTOR(I) = -1;
        END;
CALL OPNRES(PRECISION,H,H_DENOM,MODULUS);
CALL OPNRES(PRECISION,START_FACTOR,START_FACTOR_DENOM,MODULUS);

/**** INITIALIZE W(I) TO INITIAL VALUE PROBLEM, Y' = -Y,Y(0) = 1. */
/**** INITIAL VALUE W(0) = 1 IS STORED IN W_I(3).                 */
W_I(3,*) = 0;
W_I(3,PRECISION) = 1;
W_I_DENOM(*) = 0;
W_I_DENOM(PRECISION) = 1;  /* INITIALIZE DENOMINATOR(S) TO 1. */
TITLE = ' - -   ITERATION I =    .  T(I) =          .  VALUE OF W(I) IS: - ',


/******** USE EULER'S METHOD OR THE RUNGE KUTTA FOURTH ORDER    */
/******** METHOD (WHICHEVER IS SPECIFIED) TO GET 3 STARTING     */
/******** VALUES FOR USE BY THE MILNE-SIMPSON METHOD.  VALUES   */
/******** FOR THE INITIAL VALUE PROBLEM Y' = -Y, Y(0) = 1       */
/******** ARE GIVEN BY                                          */
/********  W(I-1) = W(I) (1 - H)                (EULER'S METHOD) */
/********  W(I-1) = W(I) (1 - H(1 - H/2 + H**2/6 - H**3/24)) )  */
/********                                  (RUNGE-KUTTA METHOD) */
/********                                                       */
/********    FIRST  ITERATION W(1) IS STORED IN W_I(2).         */
/********    SECOND ITERATION W(2) IS STORED IN W_I(1).         */
```

```
/********     THIRD  ITERATION W(3) IS STORED IN W_I(0).          */
DO I = 2 TO 0 BY -1;
   T_I = (3-I) * H_FLOAT;
   CALL RESMLT(PRECISION,W_I(I+1,1),START_FACTOR,MODULUS,W_I(I,1));
   PUT STRING (SUBSTR(TITLE,21,2)) EDIT(3-I)(F(2));
   PUT STRING (SUBSTR(TITLE,33,8)) EDIT(T_I) (F(8,4));
   CALL RATMAP(PRECISION,W_I(I,1),MODULUS,SUM,SUM_DENOM);
   CALL OUTPUT(PRECISION,SUM,SUM_DENOM,TITLE);
   PUT FILE(FT06F001) EDIT ('COMPARISON VALUE OF E**(-T), ',
       'SINGLE PRECISION, IS ',EXP(-T_I)) (A,A,E(15,6,8));
   PUT FILE(FT06F001) SKIP;  /* FLUSH OUTPUT BUFFER. */
END;


/********  READ IN COEFFICIENTS FOR THE MILNE-SIMPSON METHOD **/
/********  AND COMPUTE RESIDUE EQUIVALENTS.                   **/
DO I = 0 TO 3;
   GET LIST(W_I_FACTOR(I,PRECISION),W_I_FACTOR_DENOM(PRECISION));
   /**** EXTEND NEGATIVE NUMERATOR SIGN INTO LEFT K-1 WORDS. */
   IF W_I_FACTOR(I,PRECISION) < 0
      THEN DO J = 1  TO PRECISION - 1;
              W_I_FACTOR(I,J) = -1;
           END;
   CALL OPNRES(PRECISION,W_I_FACTOR(I,1),W_I_FACTOR_DENOM,MODULUS);
END;
PUT FILE(FT06F001) SKIP;    /* FLUSH OUTPUT BUFFER.  */



/******** USE THE MILNE-SIMPSON METHOD TO FIND W(I) APPROXI-   */
/******** MATIONS TO THE SOLUTION OF THE INITIAL VALUE         */
/******** PROBLEM  Y' = -Y,  Y(0) = 1.  TO SIMPLIFY THE COM-   */
/******** PUTATION, THE PREDICTOR-CORRECTOR STEPS ARE COM-     */
/******** BINED, AND THE MULTIPLICATIVE FACTORS FOR THE 4      */
/******** PREVIOUS ITERATIONS ARE PRE-DETERMINED.  W(I+1) IS   */
/******** COMPUTED BY THE RELATION                             */
/*                                                            */
/*   W(I+1) =  ((8H**2 - 12H)/9)      * W(I)                  */
/*           - ((4H**2 + 3H - 9)/9) * W(I-1)                  */
/*           + ((8H**2) / 9)        * W(I-2)                  */
/*           - (H/3)                * W(I-3)                  */
/*                                                            */

DO I = 4 TO ITERATION_NO;
   /**** COMPUTE W(I+1). ****/
   T_I = I * H_FLOAT;
   CALL RESMLT(PRECISION,W_I(0,1),W_I_FACTOR(0,1),MODULUS,SUM);
   DO J = 1 TO 3;
      CALL RESMLT(PRECISION,W_I(J,1),W_I_FACTOR(J,1),MODULUS,PRODUCT);
      CALL RESADD(PRECISION,SUM,PRODUCT,MODULUS,SUM);
   END;

   /******* CONVERT NEWLY COMPUTED  W(I+1) (STORED IN SUM) TO   */
```

```
      /****** RATIONAL EQUIVALENT AND PRINT.                          */
      PUT STRING (SUBSTR(TITLE,21,2)) EDIT(I)(F(2));
      PUT STRING (SUBSTR(TITLE,33,8)) EDIT(T_I) (F(8,4));
      CALL RATMAP(PRECISION,SUM,MODULUS,PRODUCT,PRODUCT_DENOM);
      CALL OUTPUT(PRECISION,PRODUCT,PRODUCT_DENOM,TITLE);
      PUT FILE(FTO6F001)  EDIT ('COMPARISON VALUE OF E**(-T), ',
          'SINGLE PRECISION, IS  ',EXP(-T_I)) (COL(1),A,A,E(15,6,0));
      PUT FILE(FTO6F0C1) SKIP;  /* FLUSH OUTPUT BUFFER. */

      /****** MOVE CURRENT ITERATION RESULTS DOWN ONE POSITION TO SET  */
      /****** UP FOR NEXT ITERATION.  LAST FOUR VALUES OF W(I) ARE     */
      /****** SAVED FOR USE IN COMPUTING NEXT W(I).                    */

      /**** MOVE W(2) THROUGH W(0) DOWN ONE POSITION.    */
      DO J = 3 TO 1 BY -1;
         DO K = 1 TO PRECISION;
            W_I(J,K) = W_I(J-1,K);
         END;
      END;
      /**** COPY COMPUTED W(I+1) INTO W(0) POSITION. */
      DO K = 1 TO PRECISION;
         W_I(0,K) = SUM(K);
      END;
   END;   /*  TO MILNE-SIMPSON LOOP  */



CALL CLSOUT;  /* CLOSE OUTPUT PRINT FILE. */

END MLN_SMP;
```

APPENDIX X

MACLAURIN SERIES PROGRAMS FOR APPROXIMATING E^X (ASSEMBLER)

```
         PRINT    ON
         PRINT    NOGEN
         TITLE    '-- -  MACLAURIN SERIES EXPANSION OF E**X  ---'
*********************************************************************
*                    ------        M A I N        ------
*
*       MACLAURIN SERIES EXPANSION OF E ** X, WHERE P(0) = 1
*
*     THIS PROGRAM USES MULTIPLE PRECISION RATIONAL ARITHMETIC TO
*  COMPUTE AN APPROXIMATION TO E**X USING TAYLOR'S THEOREM.  FOR
*  F(X) = E**X AND X(0) = 0, WE GET
*
*                E**X = PN(X) + RN(X)     ,
*
*  WHERE THE NTH-DEGREE TAYLOR POLYNOMIAL PN(X) APPROXIMATING E**X IS
*
*       PN(X) = 1 + X/(1 FACTORIAL) + X**2/(2 FACTORIAL)
*                 + X**3/(3 FACTORIAL) + ... + X**N/(N FACTORIAL)    ,
*
*  AND THE TRUNCATION ERROR RN(X) ASSOCIATED WITH PN(X) IS GIVEN BY
*
*       RN(X) = E**(ZETA(X)) * (X**(N+1)/(N+1) FACTORIAL)     ,
*
*  WITH ZETA(X) SOME POINT LYING BETWEEN 0 AND X.
*  HENCE, FOR POSITIVE X, ZETA(X) <= X, AND
*
*       RN(X) <= E**X * (X**(N+1)/(N+1) FACTORIAL)     .
*
*  FOR NEGATIVE X, ZETA(X) <= 0, AND HENCE
*
*       RN(X) <= X**(N+1)/(N+1) FACTORIAL     .
*
*     FOR A SPECIFIED N AND RATIONAL X (NUMERATOR AND DENOMINATOR
*  OF X ARE EACH ASSUMED TO FIT IN A SINGLE 32-BIT WORD), THE PROGRAM
*  COMPUTES X**N, N FACTORIAL, THE N-TH TERM X**N/N FACTORIAL, THE
*  TAYLOR POLYNOMIAL PN(X), AND BOUND ON TRUNCATION ERROR RN(X).
*  INTERMEDIATE VALUES OF THESE QUANTITIES ARE REPORTED AFTER MUL-
*  TIPLES OF A SPECIFIED NUMBER OF ITERATIONS.
*
*  SUBROUTINES CALLED:
*       RATSET -- VERIFIES THAT PRECISION AND K AGREE AND OPENS OUTPUT
*                 PRINT FILE FT06F001
*       OPNRAT -- REDUCES RATIONAL NUMBERS TO STANDARD FORM WITH
*                 NUMERATOR CONTAINING SIGN
*       RATMLT -- PERFORMS MULTIPLICATION OF TWO MULTIPLE PRECISION
*                 RATIONAL NUMBERS
*       RATDIV -- PERFORMS DIVISION OF TWO MULTIPLE PRECISION RATIONAL
```

```
*                    OPERANDS
*         RATADD --- ADDS TWO MULTIPLE PRECISION RATIONAL NUMBERS
*         OUTPUT --- CONVERTS TO EBCDIC CHARACTERS AND PRINTS MULTIPLE
*                    PRECISION, 2'S COMPLEMENT RATIONAL NUMBERS
*         CLSOUT --- CLOSES OUTPUT PRINT FILE FT06F001
*
*    REQUIRED PROGRAM INPUT:   NONE
*
*    DATA INITIALIZATIONS/ PROGRAM MODIFICATIONS:
*        THE FOLLOWING VARIABLES MAY BE MODIFIED AS NECESSARY FOR EACH
*    PROGRAM EXECUTION BY CHANGING THEIR DC STORAGE INITIALIZATIONS.
*        ITERNO   --- TOTAL NUMBER OF TERMS N TO COMPUTE (NUMBER OF
*                     ITERATIONS)
*        PRTINTVL --- NUMBER OF ITERATIONS AFTER WHICH INTERMEDIATE
*                     RESULTS ARE PRINTED (A VALUE 1 TO N)
*        PRECSION --- NUMBER OF 32-BIT WORDS TO HOLD NUMERATOR OR
*                     DENOMINATOR OF OPERANDS (INTEGER FROM 2 TO 303)
*        X        --- NUMERATOR OF X
*        XDEN     --- ASSOCIATED DENOMINATOR OF X
*        TITLEXN  --- LABEL FOR X**N REPORTING X/XDEN, SET TO AGREE
*                     WITH INITIALIZATION OF X/XDEN.
*********************************************************************
          SPACE
MAIN      MIENTRY  R10,MAINSAVE
****** OPEN OUTPUT PRINT FILE AND VERIFY THAT "PRECSION" OF MAIN
****** PROGRAM AGREES WITH "K" OF CSECT SETUP.  CONVERT PRECISION
****** TO CHARACTERS AND PRINT.
          CALL     RATSET,,(PRECSION) OPEN OUTPUT PRINT FILE AND
*                                        VERIFY THAT PRECSION = K.
          L        R12,=A(PRINTOUT)  ESTABLISH ADDRESSABILITY TO
          USING    PRINTOUT,R12        PRINTOUT DCB USING R12.
          PUTOUT   =CL63'       *-*-*  MACLAURIN SERIES APPROXIMATION TOX
                E ** X  *-*-*',63
          L        R5,PRECSION       LOAD PRECISION INTO R5
          CVD      R5,PACKK             AND CONVERT TO PACKED DECIMAL.
          MVC      OUTK(4),=X'40202120' MOVE EDIT PATTERN TO OUTK.
          ED       OUTK(4),PACKK+6   CONVERT K TO CHARACTERS
          PUTOUT   OUTKMSG,68           AND PRINT PRECISION MESSAGE.
          SPACE
****** GET STANDARD REDUCED RATIONAL FORM OF X (X/XDEN) AND
****** ESTABLISH APPROPRIATE TITLE FOR BOUND ON RN ERROR TERM.
          CALL     OPNRAT,,(PRECSION,X,XDEN)  REDUCE X/XDEN.
          TM       X,B'10000000'     TEST SIGN OF X.
          BO       NEGX1             BRANCH IF X IS NEGATIVE.
          MVC      TITLERN(64),=CL64' *   FACTOR X**(N-1)/(N-1) FACTORIAX
                L IN BOUND ON ERROR TERM    *'
          B        LDARGS1           GO LOAD ARGUMENT TABLES.
NEGX1     MVC      TITLERN(64),=CL64' *      BOUND ON ERROR TERM RN(X), X
                USING ZETA(X) = 0, IS       *'
          SPACE
****** LOAD ARGUMENT TABLES FOR CALLS WITHIN COMPUTATION LOOP.
LDARGS1   LDARGTBL FCTTABL,(PRECSION,NFACT,NFACTDEN,N1,N1DENOM,NFACT,NFAX
                CTDEN)
```

```
         LDARGTBL MLTTBL2,(PRECSION,XTON,XTONDEN,X,XDEN,XTCN,XTONDEN)
         LDARGTBL DIVTAB,(PRECSION,XTON,XTONDEN,NFACT,NFACTDEN,TERM,TERX
               NDEN)
         LDARGTBL ADDTBL,(PRECSION,PN,PNDEN,TERM,TERMDEN,PN,PNDSN)
         LDARGTBL ADDTBL2,(PRECSION,ONE,ONEDEN,N1,N1DENOM,N1,N1DENOM)
         LDARGTBL OUTP1,(PRECSION,TERM,TERMDEN,TITLETM)
         LDARGTBL OUTP2,(PRECSION,XTON,XTONDEN,TITLEXN)
         LDARGTBL OUTP3,(PRECSION,NFACT,NFACTDEN,TITLENF)
         LDARGTBL OUTP4,(PRECSION,TERM,TERMDEN,TITLETRM)
         LDARGTBL OUTP5,(PRECSION,PN,PNDEN,TITLEPN)
         SPACE
****** LOAD NUMBER OF TERMS TO BE COMPUTED IN MACLAURIN SERIES EXPAN-
****** SION AND DETERMINE ITERATION PRINTING INTERVAL FOR INTERMEDIATE
****** RESULTS.
         L        R6,ITERNO          LOAD NUMBER OF TERMS TO COMPUTE.
         LA       R6,1(R6)           ADD 1 FOR # LOOP ITERATIONS.
         L        R8,PRTINTVL        LOAD ITERATION INTERVAL FOR PRINT-
*                                       ING OF RESULTS.
         LA       R9,1               INITIALIZE CURRENT ITERATION N TO 1
         LR       R7,R9              COPY 1 INTO R7 FOR COMPARISONS.
         SPACE    2
********** COMPUTE THE NEXT TERM OF THE MACLAURIN SERIES EXPANSION
********** FOR E**X AND ACCUMULATE IT TO THE TOTAL.  PRINT INTERMEDIATE
********** RESULTS AS SPECIFIED BY ITERATION PRINT INTERVAL.
ACCUM    CALL     RATMLT,FCTTABL,    COMPUTE NEXT TERM OF N FACTORIAL.
         CALL     RATMLT,MLTTBL2,    COMPUTE NEXT VALUE OF X ** N.
         CALL     RATDIV,DIVTAB,     GET NEXT TERM OF SERIES.
         CR       R6,R7              IS THIS THE FINAL ERROR TERM?
         BE       PRNTRM             IF SO, DO NOT ADD TERM.
         CALL     RATADD,ADDTBL,     ADD NEXT TERM TO TOTAL.
         CALL     RATADD,ADDTBL2,    ADD 1 TO RUNNING COUNT OF N.
         SPACE
****** DETERMINE IF N IS A MULTIPLE OF THE ITERATION INTERVAL FOR
****** PRINTING OF RESULTS.  IF THE ITERATION INTERVAL IS 1, RESULTS
****** ARE PRINTED FOR EACH ITERATION.
         CR       R9,R7              IS THIS ITERATION #1?
         BE       TESTN              IF SO, PRINT NO INTERMEDIATE RE-
*                                       SULTS (BOUND ON R(0) ERROR TERM)
         CR       R8,R7              IS ITERATION INTERVAL =1?
         BNE      TESTN              IF NOT, TEST REMAINDER OF DIVISION
*                                       N / ITERATION INTERVAL.
         CALL     OUTPUT,OUTP1,      PRINT CURRENT VALUE OF R(N).
         BAL      R11,OUTPUT2        PRINT CURRENT PN, ETC.
         B        NEXT               CONTINUE WITH NEXT ITERATION.
         SPACE
TESTN    LR       R14,R9             COPY N INTO R14
         SRDA     R14,32               AND SHIFT RIGHT INTO R15.
         DR       R14,R8             DIVIDE N BY ITERATION INTERVAL.
         CR       R14,R7             CHECK REMAINDER.
         BH       NEXT               IF N NOT EVENLY DIVISIBLE BY THE
*                                       ITERATION INTERVAL, CONTINUE.
         BL       PRINTIT            GO PRINT CURRENT RESULTS.
         SPACE
```

```
*******  CONVERT FOR OUTPUT THE QUANTITY X**(N+1) / (N+1) FACTORIAL ---
*******  VALUE OF "TERM" ON THE N+1 ITERATION.  A BOUND ON THE RE
*******  ERROR TERM IS GIVEN BY
*******           "TERM(N+1)" IF N IS NEGATIVE, USING ZETA(X) = 0,
*******  OR BY
*******           TERM(N+1) * E**X IF X IS POSITIVE, USING ZETA(X) = X.
*******  IF N IS A MULTIPLE OF THE SPECIFIED ITERATION NUMBER, PRINT
*******  INTERMEDIATE VALUES OF X**N, N FACTORIAL, TERM = X**N /
*******  N FACTORIAL, AND APPROXIMATION PN(X) TO E ** X.
PRNTRN   CR       R9,R7              IS THIS ITERATION #1?
         BE       NEXT               IF SO, DO NOT PRINT BOUND ON RN.
         CALL     OUTPUT,OUTP1,      PRINT CURRENT VALUE OF R(N).
         B        NEXT               CONTINUE NUMERICAL APPROXIMATION.
PRINTIT  BAL      R11,OUTPUT2        GO PRINT CURRENT RESULTS.
NEXT     LA       R9,1(R9)           INCREMENT N.
         BCT      R6,ACCUM           LOOP UNTIL ALL TERMS ARE COMPUTED.
         SPACE    2
         B        GOHOME             RETURN TO CALLER
         SPACE    2
*****************************************************************
*       CONVERT RATIONAL RESULTS TO CHARACTERS AND PRINT RESULTS:
*   X ** N, N FACTORIAL, TERM = (X**N) / N FACTORIAL, PN(X), AND
*   BOUND ON RN(X).
*****************************************************************
         SPACE
**** CONVERT N TO CHARACTERS AND PRINT.
OUTPUT2  CVD      R9,PACKN           CONVERT ITERATION # TO PACKED
*                                      DECIMAL.
         MVC      OUTN(6),=X'402020202120' MOVE EDIT PATTERN TO OUTPUT
         ED       OUTN(6),PACKN+5    CONVERT N TO CHARS. FOR PRINTING
         PUTOUT   SKIP,4             LEAVE 4 BLANK LINES BEFORE OUTPUT.
         PUTOUT   ASTERISK,44        PRINT LINE OF ASTERISKS.
         PUTOUT   OUTNMSG,44         PRINT CURRENT VALUE OF N.
         PUTOUT   ASTERISK,44        PRINT LINE OF ASTERISKS.
         SPACE
**** PRINT INTERMEDIATE RESULTS X**N, N FACTORIAL, TERM, AND PN(X).
         CALL     OUTPUT,OUTP2,      PRINT CURRENT VALUE OF X**N.
         CALL     OUTPUT,OUTP3       PRINT CURRENT N FACTORIAL..
         CALL     OUTPUT,OUTP4       PRINT TERM X**N / NFACTORIAL..
         CALL     OUTPUT,OUTP5       PRINT P(N) APPROXIMATION TO E**X..
*
         BR       R11                RETURN TO CALLER.
         SPACE
*        ******     DATA STORAGE FOR OUTPUT OF RESULTS      ******
PACKN    DS       D                  PACKED DECIMAL VALUE OF PRECS. K
OUTNMSG  DC       CL59'0---  SOLUTION USING K-WORD RATIONAL ARITHMETICX
               , WHERE K ='
OUTK     DS       4C                 EBCDIC CHARS. FOR PRECISION K
         DC       C' ---'
PACKN    DS       D                  PACKED DECIMAL VALUE OF ITER. #
OUTNMSG  DC       22C' ',CL12'**** N = .  '
OUTN     DS       6C                 EBCDIC CHARS. FOR ITERATION #
         DC       CL4' ***'
```

```
ASTERISK DC       22C' ',22C'*'      BLANKS & ASTERISKS
TITLENF  DC       CL63' #                        N FACTORIAL IS:',C'*'
TITLETRM DC       CL63' #             N-TH TERM OF SERIES -- X**T / N FACTX
               ORIAL',C'*'
TITLEPN  DC       CL63' #                  PN(X), APPROXIMATING E**X, IS.'X
               ,C'*'
TITLERM  DS       CL64                STORAGE FOR ERROR BOUND TITLE
*********************************************************************
         SPACE    2
COHOME   CALL     CLSOUT              CLOSE OUTPUT PRINT FILE.
         MIRETURN
         DROP     R12
         SPACE    2
******   DATA STORAGE FOR MACLAURIN SERIES COMPUTATION OF E**X   ******
         SPACE
*     FOLLOWING DATA CAN BE MODIFIED FOR EACH PROGRAM EXECUTION     *
*                                                                   *
ITERNO   DC       F'75'               # TERMS IN SERIES TO COMPUTE
PRTINTVL DC       F'75'               ITERATION FREQUENCY FOR PRINTING
*                                        OF RESULTS
PRECSION DC       F'34'               NUMBER OF FULLWORDS IN OPERANDS
X        MULTPREC 1,F'-1',1,F'-326'          INITIALIZE X TO -326/917.
XDEN     MULTPREC 1,F'0',1,F'917'
TITLEXM  DC       CL63' #         X**M, WHERE X =      -326/917, IS.'
         DC       CL1'*'
*                                                                   *
         SPACE
X1       MULTPREC 1,F'0',1,F'1'     MULTIPLE PRECISION STORAGE FOR N,
X1DENOM  MULTPREC 1,F'0',1,F'1'        INITIALIZED TO 1/1
XTON     MULTPREC 1,F'0',1,F'1'     CURRENT VALUE OF X ** N,
XTONDEN  MULTPREC 1,F'0',1,F'1'        INITIALIZED TO 1/1.
NFACT    MULTPREC 1,F'0',1,F'1'     N FACTORIAL,
NFACTDEN MULTPREC 1,F'0',1,F'1'        INITIALIZED TO 1/1
TERM     MULTPREC 1                 STORAGE FOR CURRENT TERM X**N /
*                                      N FACTORIAL
TERMDEN  MULTPREC 1                    AND ASSOCIATED DENOMINATOR.
PN       MULTPREC 1,F'0',1,F'1'      STORAGE FOR ACCUMULATED SUM OF
PNDEN    MULTPREC 1,F'0',1,F'1'         TERMS =  APPROXIMATION TO E**X
ONE      MULTPREC 1,F'0',1,F'1'      PRECSION-WORD CONSTANT OF 1
ONEDEN   MULTPREC 1,F'0',1,F'1'
         SPACE    3



         PRINT    ON
         PRINT    NOGEN
         TITLE    '1     MACLAURIN SERIES EXPANSION OF E**X  '
*********************************************************************
*                      M A I N                                      *
*                                                                   *
*        MACLAURIN SERIES EXPANSION OF E ** X, WHERE P(0) = 1       *
*                                                                   *
```

```
C        THIS PROGRAM USES MULTIPLE PRECISION RESIDUE ARITHMETIC TO
C    COMPUTE AN APPROXIMATION TO E**X USING TAYLOR'S THEOREM.  FOR
C    F(X) = E**X AND X(0) = 0, WE GET
C
C                 E**X = PN(X) + RN(X)     ,
C
C    WHERE THE NTH-DEGREE TAYLOR POLYNOMIAL PN(X) APPROXIMATING E**X IS
C
C        PN(X) = 1 + X/(1 FACTORIAL) + X**2/(2 FACTORIAL)
C                  + X**3/(3 FACTORIAL) + ... + X**N/(N FACTORIAL)    ,
C
C    AND THE TRUNCATION ERROR RN(X) ASSOCIATED WITH PN(X) IS GIVEN BY
C
C        RN(X) = E**(ZETA(X)) * (X**(N+1)/(N+1) FACTORIAL)     ,
C
C    WITH ZETA(X) SOME POINT LYING BETWEEN 0 AND X.
C    HENCE, FOR POSITIVE X, ZETA(X) <= X, AND
C
C        RN(X) <= E**X * (X**(N+1)/(N+1) FACTORIAL)     .
C
C    FOR NEGATIVE X, ZETA(X) <= 0, AND HENCE
C
C        RN(X) <= X**(N+1)/(N+1) FACTORIAL     .
C
C        FOR A SPECIFIED N AND RATIONAL X (NUMERATOR AND DENOMINATOR
C    OF X ARE EACH ASSUMED TO FIT IN A SINGLE 32-BIT WORD), THE PROGRAM
C    COMPUTES X**N, N FACTORIAL, THE N-TH TERM X**N/N FACTORIAL, THE
C    TAYLOR POLYNOMIAL PN(X), AND BOUND ON TRUNCATION ERROR RN(X).
C    INTERMEDIATE VALUES OF THESE QUANTITIES ARE REPORTED AFTER MUL-
C    TIPLES OF A SPECIFIED NUMBER OF ITERATIONS.
C
C    SUBROUTINES CALLED:
C        SETUP  --  COMPUTES MODULUS AND ASSOCIATED ORDER-N OF FAREY
C                   FRACTIONS AND OPENS OUTPUT PRINT FILE FT06F001
C        OPNRES --  COMPUTES MULTIPLE PRECISION RESIDUE !P/Q! MOD N OF
C                   A MULTIPLE PRECISION RATIONAL NUMBER P/Q
C        RESMLT --  PERFORMS THE RESIDUE MULTIPLICATION !A*B! MOD N
C                   OF TWO MULTIPLE PRECISION INTEGERS A AND B
C        RESDIV --  PERFORMS THE RESIDUE DIVISION !A/B! MOD N OF TWO
C                   MULTIPLE PRECISION INTEGERS A AND B
C        RESADD --  PERFORMS THE RESIDUE ADDITION !A+B! MOD N
C        RATMAP --  PERFORMS THE INVERSE MAPPING FROM RESIDUE RESULT
C                   TO THE ORDER-N FAREY FRACTION EQUIVALENT
C        OUTPUT --  CONVERTS TO EBCDIC CHARACTERS AND PRINTS MULTIPLE
C                   PRECISION, 2'S COMPLEMENT RATIONAL NUMBERS
C        CLSOUT --  CLOSES OUTPUT PRINT FILE FT06F001
C
C    REQUIRED PROGRAM INPUT:  NONE
C
C    DATA INITIALIZATIONS/ PROGRAM MODIFICATIONS:
C        THE FOLLOWING VARIABLES MAY BE MODIFIED AS NECESSARY FOR EACH
C    PROGRAM EXECUTION BY CHANGING THEIR DC STORAGE INITIALIZATIONS.
C        ITERNO  --  TOTAL NUMBER OF TERMS N TO COMPUTE (NUMBER OF
```

```
*                       ITERATIONS)
*         PRTINTVL --   NUMBER OF ITERATIONS AFTER WHICH INTERMEDIATE
*                       RESULTS ARE PRINTED (A VALUE 1 TO N)
*         PRECSION --   NUMBER OF 32-BIT WORDS TO HOLD OPERANDS (INTEGER
*                       FROM 2 TO 303)
*         X        --   NUMERATOR OF X
*         XDEN     --   ASSOCIATED DENOMINATOR OF X
*         TITLEXN  --   LABEL FOR X**N REPORTING X/XDEN, SET TO AGREE
*                       WITH INITIALIZATION OF X/XDEN.
*********************************************************************
          SPACE
MAIN      MIENTRY  R10,MAINSAVE
****** COMPUTE MODULUS AND ASSOCIATED ORDER N OF FAREY FRACTIONS.
****** OPEN OUTPUT PRINT FILE AND VERIFY THAT "PRECSION" OF MAIN
****** PROGRAM AGREES WITH "N" OF CSECT SETUP.  CONVERT PRECISION
****** TO CHARACTERS AND PRINT.
          CALL     SETUP,,(PRECSION,MODULUS) INITIALIZE MODULUS,
*                                            HALFMOD, AND N.
          L        R12,=A(PRINTOUT)  LOAD ADDRESSS OF PRINT FILE DCB
          USING    PRINTOUT,R12         AND ESTABLISH ADDRESSABILITY
*                                       USING R12.
          PUTOUT   =CL63'        *.*.*  MACLAUREN SERIES APPROXIMATION TO X
                E ** X  *.*.*',63
          L        R5,PRECSION       LOAD PRECISION INTO R5
          CVD      R5,PACKN             AND CONVERT TO PACKED DECIMAL.
          MVC      OUTK(4),=X'40202120' MOVE EDIT PATTERN TO OUTK.
          ED       OUTK(4),PACKN+6   CONVERT K TO CHARACTERS
          PUTOUT   OUTMSG,68            AND PRINT PRECISION MESSAGE.
          SPACE
****** GET RESIDUE EQUIVALENT OF RATIONAL NUMBER (X/XDEN) AND
****** ESTABLISH APPROPRIATE TITLE FOR BOUND ON RN ERROR TERM.
          CALL     OPNRES,,(PRECSION,X,XDEN,MODULUS,X)
          TM       X,B'10000000'     TEST SIGN OF X.
          BO       NEGX1             BRANCH IF X IS NEGATIVE.
          MVC      TITLERN(64),=CL64' #   FACTOR X**(N+1)/(N+1) FACTORIAX
                L IN BOUND ON ERROR TERM    #'
          B        LDARGS1           GO LOAD ARGUMENT TABLES.
NEGX1     MVC      TITLERN(64),=CL64' #      BOUND ON ERROR TERM RN(X), X
                USING ZETA(X) = 0, IS       #'
          SPACE
****** LOAD ARGUMENT TABLES FOR CALLS WITHIN COMPUTATION LOOP.
LDARGS1   LDARGTBL FCTTABL,(PRECSION,NFACT,N1,MODULUS,NFACT)
          LDARGTBL ILTTBL2,(PRECSION,XTON,X,MODULUS,XTON)
          LDARGTBL DIVTAB,(PRECSION,XTON,NFACT,MODULUS,TERM)
          LDARGTBL ADDTBL,(PRECSION,PN,TERM,MODULUS,PN)
          LDARGTBL ADDTBL2,(PRECSION,ONE,N1,MODULUS,N1)
          LDARGTBL MAP1,(PRECSION,TERM,MODULUS,WORK,WORKDEN)
          LDARGTBL MAP2,(PRECSION,XTON,MODULUS,WORK,WORKDEN)
          LDARGTBL MAP3,(PRECSION,NFACT,MODULUS,WORK,WORKDEN)
          LDARGTBL MAP4,(PRECSION,PN,MODULUS,WORK,WORKDEN)
          LDARGTBL OUTP1,(PRECSION,WORK,WORKDEN,TITLERN)
          LDARGTBL OUTP2,(PRECSION,WORK,WORKDEN,TITLEXN)
          LDARGTBL OUTP3,(PRECSION,WORK,WORKDEN,TITLENF)
```

```
         LDARGTBL OUTP4,(PRECSION,WORK,WORKDEN,TITLETRM)
         LDARGTBL OUTP5,(PRECSION,WORK,WORKDEN,TITLEPN)
         SPACE
****** LOAD NUMBER OF TERMS TO BE COMPUTED IN MACLAURIN SERIES EXPAN-
****** SION AND DETERMINE ITERATION PRINTING INTERVAL FOR INTERMEDIATE
****** RESULTS.
         L        R6,ITERNO          LOAD NUMBER OF TERMS TO COMPUTE.
         LA       R6,1(R6)           ADD 1 FOR # LOOP ITERATIONS.
         L        R8,PRTINTVL        LOAD ITERATION INTERVAL FOR PRINT-
*                                       ING OF RESULTS.
         LA       R9,1               INITIALIZE CURRENT ITERATION N TO 1
         LR       R7,R9              COPY 1 INTO R7 FOR COMPARISONS.
         SPACE    2
********** COMPUTE THE NEXT TERM OF THE MACLAURIN SERIES EXPANSION
********** FOR E**X AND ACCUMULATE IT TO THE TOTAL.  PRINT INTERMEDIATE
********** RESULTS AS SPECIFIED BY ITERATION PRINT INTERVAL.
ACCUM    CALL     RESMLT,FCTTABL,    COMPUTE NEXT TERM OF N FACTORIAL.
         CALL     RESMLT,MLTTBL2,    COMPUTE NEXT VALUE OF X ** N.
         CALL     RESDIV,DIVTAB,     GET NEXT TERM OF SERIES.
         CR       R6,R7              IS THIS THE FINAL ERROR TERM?
         BE       PRNTRM             IF SO, DO NOT ADD TERM.
         CALL     RESADD,ADDTBL,     ADD NEXT TERM TO TOTAL.
         CALL     RESADD,ADDTBL2,    ADD 1 TO RUNNING COUNT OF N.
         SPACE
****** DETERMINE IF N IS A MULTIPLE OF THE ITERATION INTERVAL FOR
****** PRINTING OF RESULTS.  IF THE ITERATION INTERVAL IS 1, RESULTS
****** ARE PRINTED FOR EACH ITERATION.
         CR       R9,R7              IS THIS ITERATION #1?
         BE       TESTN              IF SO, PRINT NO INTERMEDIATE RE-
*                                       SULTS (BOUND ON R(0) ERROR TERM)
         CR       R8,R7              IS ITERATION INTERVAL =1?
         BNE      TESTN              IF NOT, TEST REMAINDER OF DIVISION
*                                       N / ITERATION INTERVAL.
         CALL     RATMAP,MAP1,       MAP TERM TO RATIONAL EQUIVALENT.
         CALL     OUTPUT,OUTP1       PRINT RATIONAL FORM OF TERM.
         DAL      R11,OUTPUT2        PRINT CURRENT PN, ETC.
         B        NEXT               CONTINUE WITH NEXT ITERATION.
         SPACE
TESTN    LR       R14,R9             COPY N INTO R14
         SRDA     R14,32                AND SHIFT RIGHT INTO R15.
         DR       R14,R8             DIVIDE N BY 5.
         CR       R14,R7             CHECK REMAINDER.
         BNE      NEXT               IF N NOT EVENLY DIVISIBLE BY THE
*                                       ITERATION INTERVAL, CONTINUE.
         BL       PRINTIT            GO PRINT CURRENT RESULTS.
         SPACE
******   CONVERT FOR OUTPUT THE QUANTITY X**(N-1) / (N-1) FACTORIAL --
******   VALUE OF "TERM" ON THE N-1 ITERATION.  A BOUND ON THE NU
******   ERROR TERM IS GIVEN BY
******            "TERM(N-1)" IF N IS NEGATIVE, USING ZETA(X) = 0,
******   OR BY
******            TERM(N-1) * E**X IF X IS POSITIVE, USING ZETA(X) = X.
******   IF N IS A MULTIPLE OF THE SPECIFIED ITERATION NUMBER, PRINT
```

```
******  INTERMEDIATE VALUES OF X**N, N FACTORIAL, TERM = X**N /
******  N FACTORIAL, AND APPROXIMATION PN(X) TO E ** X.
PRNTRN   CR       R9,R7              IS THIS ITERATION #1?
         BE       NEXT               IF SO, DO NOT PRINT BOUND ON RN.
         CALL     RATMAP,MAP1,       MAP TERM TO RATIONAL EQUIVALENT.
         CALL     OUTPUT,OUTP1       PRINT RATIONAL FORM OF TERM.
         B        NEXT               CONTINUE NUMERICAL APPROXIMATION.
PRNTIT   BAL      R11,OUTPUT2        GO PRINT CURRENT RESULTS.
NEXT     LA       R9,1(R9)           INCREMENT N.
         BCT      R6,ACCUM           LOOP UNTIL ALL TERMS ARE COMPUTED.
         SPACE    2
         B        GOHOME             RETURN TO CALLER
         SPACE    2
*****************************************************************
*       MAP BACK TO RATIONAL EQUIVALENTS, CONVERT TO CHARACTERS,
*   AND PRINT RESULTS:  X ** N, N FACTORIAL, TERM = (X**N) /
*   N FACTORIAL, PN(X), AND BOUND ON RN(X).
*****************************************************************
         SPACE
**** CONVERT N TO CHARACTERS AND PRINT.
OUTPUT2  CVD      R9,PACKN           CONVERT ITERATION # TO PACKED
*                                      DECIMAL.
         MVC      OUTN(6),=X'402020202120' MOVE EDIT PATTERN TO OUTPUT
         ED       OUTN(6),PACKN+5    CONVERT N TO CHARS. FOR PRINTING.
         PUTOUT   SKIP,4             LEAVE 4 BLANK LINES BEFORE OUTPUT.
         PUTOUT   ASTERISK,44        PRINT LINE OF ASTERISKS.
         PUTOUT   OUTNMSG,44         PRINT CURRENT VALUE OF N.
         PUTOUT   ASTERISK,44        PRINT LINE OF ASTERISKS.
         SPACE
**** PRINT INTERMEDIATE RESULTS X**N, N FACTORIAL, TERM, AND PN(X).
         CALL     RATMAP,MAP2,       MAP X**N TO RATIONAL EQUIVALENT
         CALL     OUTPUT,OUTP2,         AND PRINT.
         CALL     RATMAP,MAP3,       MAP N FACTORIAL TO RATIONAL EQUIVA-
         CALL     OUTPUT,OUTP3          LENT AND PRINT.
         CALL     RATMAP,MAP1        MAP TERM TO RATIONAL EQUIVALENT
         CALL     OUTPUT,OUTP4          AND PRINT.
         CALL     RATMAP,MAP4        MAP P(N) TO RATIONAL EQUIVALENT
         CALL     OUTPUT,OUTP5          AND PRINT.
*
         BR       R11                RETURN TO CALLER.
         SPACE
*       ******      DATA STORAGE FOR OUTPUT OF RESULTS       ******
PACKK    DS       D                  PACKED DECIMAL VALUE OF PRECS. K
OUTKMSG  DC       CL59'0---  SOLUTION USING K-WORD RESIDUE ARITHMETIC,X
                  WHERE K ='
OUTK     DS       4C                 EBCDIC CHARS. FOR PRECISION K
         DC       C'  ---- '
PACKN    DS       D                  PACKED DECIMAL VALUE OF ITER. #
OUTNMSG  DC       22C' ',CL12'**** N = '
OUTN     DS       6C                 EBCDIC CHARS. FOR ITERATION #
         DC       CL4' ***'
ASTERISK DC       22C' ',22C'*'      BLANKS + ASTERISKS
TITLENF  DC       CL63' *                          N FACTORIAL IS:',C'*'
```

```
TITLETRM DC       CL63' #          N-TH TERM OF SERIES -- X**N / N FACTX
               ORIAL',C'#'
TITLEPN  DC       CL63' #               PN(X), APPROXIMATING E**X, IS:'X
               ,C'#'
TITLERM  DS       CL64                  STORAGE FOR ERROR BOUND TITLE
*********************************************************************
         SPACE    2
GOHOME   CALL     CLSOUT                CLOSE OUTPUT PRINT FILE.
         HIRETURN
         DROP     R12
         SPACE    2
******   DATA STORAGE FOR MACLAURIN SERIES COMPUTATION OF E**X  ******
         SPACE
#------ FOLLOWING DATA CAN BE MODIFIED FOR EACH PROGRAM EXECUTION ------#
#-------------------------------------------------------------------------#
ITERNO   DC       F'25'                 # ITERATIONS TO COMPUTE
PRTINTVL DC       F'5'                  ITERATION FREQUENCY FOR PRINTING
#                                          OF RESULTS
PRECSION DC       F'41'                 NUMBER OF FULLWORDS IN OPERANDS
X        MULTPREC 1,F'-1',1,F'-326'     INITIALIZE X TO -326/917.
XDEN     MULTPREC 1,F'0',1,F'917'
TITLEXN  DC       CL63' #        X**N, WHERE X =     -326/917, IS:'
         DC       CL1'#'
#-------------------------------------------------------------------------#
         SPACE
N1       MULTPREC 1,F'0',1,F'1'         MULTIPLE PRECISION STORAGE FOR
#                                          N, INITIALIZED TO 1
XTON     MULTPREC 1,F'0',1,F'1'        CURRENT VALUE OF X ** N --
#                                          INITIALIZED TO 1
NFACT    MULTPREC 1,F'0',1,F'1'        N FACTORIAL, INITIALIZED TO 1.
TERM     MULTPREC 1                     STORAGE FOR CURRENT TERM X**N /
#                                          N FACTORIAL
PN       MULTPREC 1,F'0',1,F'1'        STORAGE FOR ACCUMULATED APPROXIMA-
#                                          TION SUM PN(X) TO E**X
WORK     MULTPREC 1                     STORAGE FOR NUMERATOR AND DENOMI-
WORKDEN  MULTPREC 1                        NATOR OR RATIONAL EQUIVALENTS--
#                                          USED IN PRINTING INTERMEDIATE
#                                          RESULTS
ONE      MULTPREC 1,F'0',1,F'1'        PRECSION-WORD CONSTANT OF 1
         SPACE
MODULUS  MULTPREC 1                     STORAGE FOR MERSENNE PRIME MODULUS
         SPACE    3
```

VITA

John L. Smyre was born in Lincolnton, N. C. on September 7, 1945. He attended elementary school in York County, S. C. and was graduated from Clover High School, Clover, S. C. in June, 1963. He attended Vanderbilt University, Nashville, Tennessee, from September, 1963 to January, 1967. In August, 1966 he was awarded the Bachelor of Arts degree in Physics-Astronomy, with a minor in Mathematics, and was elected to Phi Beta Kappa in April, 1967.

In September, 1967 the author entered the United States Air Force for a two year tour of duty and served as a security policeman at Kunsan Air Base, South Korea. Beginning in August, 1969 he taught physics, mathematics, and biology at Central High School in Manchester, Tennessee and started several years of speleological research.

In June, 1979 he entered the Graduate School of the University of Tennessee, Knoxville. In September, 1980 he received a teaching assistantship in the Computer Science Department, and he continued as a teaching assistant until December, 1982. He obtained the Master of Science degree with a major in Computer Science in March, 1983.