INFORMATION TO USERS

While the most advanced technology has been used to photograph and reproduce this manuscript, the quality of the reproduction is heavily dependent upon the quality of the material submitted. For example:

- Manuscript pages may have indistinct print. In such cases, the best available copy has been filmed.

- Manuscripts may not always be complete. In such cases, a note will indicate that it is not possible to obtain missing pages.

- Copyrighted material may have been removed from the manuscript. In such cases, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, and charts) are photographed by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps. Each oversize page is also filmed as one exposure and is available, for an additional charge, as a standard 35mm slide or as a 17”x 23” black and white photographic print.

Most photographs reproduce acceptably on positive microfilm or microfiche but lack the clarity on xerographic copies made from the microfilm. For an additional charge, 35mm slides of 6”x 9” black and white photographic prints are available for any photographs or illustrations that cannot be reproduced satisfactorily by xerography.
Chren, William Anthony, Jr.

NEW AND IMPROVED DIVISION ALGORITHMS IN RESIDUE NUMBER SYSTEMS

The Ohio State University

University Microfilms International

300 N. Zeeb Road, Ann Arbor, MI 48106
PLEASE NOTE:

In all cases this material has been filmed in the best possible way from the available copy. Problems encountered with this document have been identified here with a check mark  

1. Glossy photographs or pages  
2. Colored illustrations, paper or print  
3. Photographs with dark background  
4. Illustrations are poor copy  
5. Pages with black marks, not original copy  
6. Print shows through as there is text on both sides of page  
7. Indistinct, broken or small print on several pages  
8. Print exceeds margin requirements  
9. Tightly bound copy with print lost in spine  
10. Computer printout pages with indistinct print  
11. Page(s) lacking when material received, and not available from school or author.  
12. Page(s) seem to be missing in numbering only as text follows.  
13. Two pages numbered . Text follows.  
14. Curling and wrinkled pages  
15. Dissertation contains pages with print at a slant, filmed as received  
16. Other  

University Microfilms International
NEW AND IMPROVED DIVISION ALGORITHMS
IN RESIDUE NUMBER SYSTEMS

DISSERTATION

Presented in Partial Fulfillment of the Requirements for
the degree Doctor of Philosophy in the Graduate
School of The Ohio State University

By

William A. Chren, Jr.

The Ohio State University

1987

Dissertation Committee:
Prof. Bostwick Wyman
Prof. Hooshang Hemami
Prof. Charles Klein

Approved by:

[Signature]
Advisor
Department of Electrical Engineering
ACKNOWLEDGEMENT

The author thanks Professor Stuart A. Collins, Jr. for suggesting the topic, and Professor Bostwick Wyman for his timely help and advice.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>ACKNOWLEDGEMENTS</th>
<th>ii</th>
</tr>
</thead>
<tbody>
<tr>
<td>LIST OF TABLES</td>
<td>vii</td>
</tr>
<tr>
<td>LIST OF FIGURES</td>
<td>viii</td>
</tr>
</tbody>
</table>

## CHAPER

### I INTRODUCTION

A. Introduction to the RNS

A.1 The Residue Representation
A.2 Addition, Subtraction and Multiplication
A.3 Division

A.3.1 Integer Division
A.3.2 Non-Integer Division

The Subtractive Algorithms
The Multiplicative Algorithms

### B. Previous Work

B.1 History of the RNS and Its Applications
B.2 Previous RNS Division Algorithms
B.3 Discussion of the New Algorithms

### C. Statement of the Problem and Why It Is Important

### D. Outline of the Dissertation

### II DETAILS OF THE RNS AND THE OSRA

A. Four Basic Residue Procedures

A.1 The Scaling Procedure
A.2 The Mixed Radix Conversion Procedure
A.3 The Sign Detection Procedure
A.4 The Base Extension Procedure
A.5 Summary

iii
APPENDICES

A. Proofs of Lemmas 2 and 3 226
B. Proof of Lemma 5 230
C. Proof of Lemma 7 238
D. Simulator Program for the TSRA and OSRA 244
E. Derivation of Stored Table Size for SA 258
F. The Simulator Program for the SA and the OSRA 261
G. Derivation of Expressions for P\left(\left|\frac{X}{Y}\right|<j\right) and P\left(E_1=j\land P=n\right) 276
H. Demonstration that an Overflow Modulus is Not Required by the RA when \(m_{n-1} = m_{n-1}\) 284
I. Proofs of Lemmas 9 and 10 287
J. Derivation of the Size of Table \(T^2\), Used by the RA 292
K. The Simulator Program for the RA and the OSRA 298
## LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Test Problems and Number of Operations Required for TSRA and OSRA Simulator</td>
<td>98</td>
</tr>
<tr>
<td>2</td>
<td>Mean Running Time and Standard Deviation for the TSRA and OSRA</td>
<td>113</td>
</tr>
<tr>
<td>3</td>
<td>Test Problems and Number of Operations Required for SA and OSRA Simulator</td>
<td>121</td>
</tr>
<tr>
<td>4</td>
<td>Mean Running Time and Standard Deviation for the SA and OSRA</td>
<td>145</td>
</tr>
<tr>
<td>5</td>
<td>Theoretically Predicted Probabilities That OSRA Estimates E₁ and E₂ Are Zero, and the Theoretical Lower Bound For Mean Running Time</td>
<td>154</td>
</tr>
<tr>
<td>6</td>
<td>Amount of Storage Required by the RA and OSRA</td>
<td>188</td>
</tr>
<tr>
<td>7</td>
<td>Test Problems and Number of Operations Required for RA and OSRA Simulator</td>
<td>194</td>
</tr>
<tr>
<td>8</td>
<td>Mean Running Time and Standard Deviation for the RA and OSRA</td>
<td>218</td>
</tr>
</tbody>
</table>
# LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Flowchart of the OSRA.</td>
<td>39</td>
</tr>
<tr>
<td>2. Flowchart of the TSRA.</td>
<td>63</td>
</tr>
<tr>
<td>3. Flowchart of the SA.</td>
<td>80</td>
</tr>
<tr>
<td>4. General flowchart for TSRA and OSRA simulating program.</td>
<td>93</td>
</tr>
<tr>
<td>5. Detailed flowchart of TSRA simulator.</td>
<td>94</td>
</tr>
<tr>
<td>6. Detailed flowchart of OSRA simulator.</td>
<td>95</td>
</tr>
<tr>
<td>7. TSRA and OSRA average running time as a function of sample size.</td>
<td>101</td>
</tr>
<tr>
<td>8. TSRA and OSRA sample standard deviation as a function of sample size.</td>
<td>102</td>
</tr>
<tr>
<td>9. TSRA experimental probability density of running time.</td>
<td>103</td>
</tr>
<tr>
<td>10. OSRA experimental probability density of running time.</td>
<td>104</td>
</tr>
<tr>
<td>11. TSRA and OSRA average running time as a function of sample size.</td>
<td>105</td>
</tr>
<tr>
<td>12. TSRA and OSRA sample standard deviation as a function of sample size.</td>
<td>106</td>
</tr>
<tr>
<td>13. TSRA experimental probability density of running time.</td>
<td>107</td>
</tr>
<tr>
<td>14. OSRA experimental probability density of running time.</td>
<td>108</td>
</tr>
<tr>
<td>15. TSRA and OSRA average running time as a function of sample size.</td>
<td>109</td>
</tr>
<tr>
<td>16. TSRA and OSRA sample standard deviation as a function of sample size.</td>
<td>110</td>
</tr>
<tr>
<td>17. TSRA experimental probability density of running time.</td>
<td>111</td>
</tr>
<tr>
<td>18. OSRA experimental probability density of running time.</td>
<td>112</td>
</tr>
<tr>
<td>Figure</td>
<td>Page</td>
</tr>
<tr>
<td>--------</td>
<td>------</td>
</tr>
<tr>
<td>19. Global flowchart for SA and OSRA simulating program.</td>
<td>117</td>
</tr>
<tr>
<td>20. Detailed flowchart for SA simulator.</td>
<td>118</td>
</tr>
<tr>
<td>21. Detailed flowchart for OSRA simulator.</td>
<td>120</td>
</tr>
<tr>
<td>22. SA and OSRA average running time as a function of sample size.</td>
<td>125</td>
</tr>
<tr>
<td>23. SA and OSRA sample standard deviation as a function of sample size.</td>
<td>126</td>
</tr>
<tr>
<td>24. SA experimental probability density of running time.</td>
<td>127</td>
</tr>
<tr>
<td>25. OSRA experimental probability density of running time.</td>
<td>128</td>
</tr>
<tr>
<td>26. SA and OSRA average running time as a function of sample size.</td>
<td>129</td>
</tr>
<tr>
<td>27. SA and OSRA sample standard deviation as a function of sample size.</td>
<td>130</td>
</tr>
<tr>
<td>28. SA experimental probability density of running time.</td>
<td>131</td>
</tr>
<tr>
<td>29. OSRA experimental probability density of running time.</td>
<td>132</td>
</tr>
<tr>
<td>30. SA and OSRA average running time as a function of sample size.</td>
<td>133</td>
</tr>
<tr>
<td>31. SA and OSRA sample standard deviation as a function of sample size.</td>
<td>134</td>
</tr>
<tr>
<td>32. SA experimental probability density of running time.</td>
<td>135</td>
</tr>
<tr>
<td>33. OSRA experimental probability density of running time.</td>
<td>136</td>
</tr>
<tr>
<td>34. SA and OSRA average running time as a function of sample size.</td>
<td>137</td>
</tr>
<tr>
<td>35. SA and OSRA sample standard deviation as a function of sample size.</td>
<td>138</td>
</tr>
<tr>
<td>36. SA experimental probability density of running time.</td>
<td>139</td>
</tr>
<tr>
<td>37. OSRA experimental probability density of running time.</td>
<td>140</td>
</tr>
<tr>
<td>Figure</td>
<td>Page</td>
</tr>
<tr>
<td>--------</td>
<td>------</td>
</tr>
<tr>
<td>38. SA and OSRA average running time as a function of sample size.</td>
<td>141</td>
</tr>
<tr>
<td>39. SA and OSRA sample standard deviation as a function of sample size.</td>
<td>142</td>
</tr>
<tr>
<td>40. SA experimental probability density of running time.</td>
<td>143</td>
</tr>
<tr>
<td>41. OSRA experimental probability density of running time.</td>
<td>144</td>
</tr>
<tr>
<td>42. Experimental and theoretical OSRA running time probability density functions.</td>
<td>167</td>
</tr>
<tr>
<td>43. Experimental and theoretical OSRA running time probability density functions.</td>
<td>157</td>
</tr>
<tr>
<td>44. Experimental and theoretical OSRA running time probability density functions.</td>
<td>158</td>
</tr>
<tr>
<td>45. Distribution function for truncated quotient.</td>
<td>159</td>
</tr>
<tr>
<td>46. Approximate probability density function for OSRA initial estimate.</td>
<td>160</td>
</tr>
<tr>
<td>47. Density function for truncated quotient.</td>
<td>161</td>
</tr>
<tr>
<td>48. RA flowchart.</td>
<td>172</td>
</tr>
<tr>
<td>49. Global flowchart for RA and OSRA simulating program.</td>
<td>190</td>
</tr>
<tr>
<td>50. Detailed flowchart for RA simulator.</td>
<td>191</td>
</tr>
<tr>
<td>51. Detailed flowchart for OSRA simulator.</td>
<td>192</td>
</tr>
<tr>
<td>52. RA and OSRA average running time as a function of sample size.</td>
<td>196</td>
</tr>
<tr>
<td>53. RA and OSRA sample standard deviation as a function of sample size.</td>
<td>197</td>
</tr>
<tr>
<td>54. RA experimental probability density of running time.</td>
<td>198</td>
</tr>
<tr>
<td>55. OSRA experimental probability density of running time.</td>
<td>199</td>
</tr>
<tr>
<td>56. RA and OSRA average running time as a function of sample size.</td>
<td>200</td>
</tr>
<tr>
<td>Figure</td>
<td>Page</td>
</tr>
<tr>
<td>--------</td>
<td>------</td>
</tr>
<tr>
<td>57. RA and OSRA sample standard deviation as a function of sample size.</td>
<td>201</td>
</tr>
<tr>
<td>58. RA experimental probability density of running time.</td>
<td>202</td>
</tr>
<tr>
<td>59. OSRA experimental probability density of running time.</td>
<td>203</td>
</tr>
<tr>
<td>60. RA and OSRA average running time as a function of sample size.</td>
<td>204</td>
</tr>
<tr>
<td>61. RA and OSRA sample standard deviation as a function of sample size.</td>
<td>205</td>
</tr>
<tr>
<td>62. RA experimental probability density of running time.</td>
<td>206</td>
</tr>
<tr>
<td>63. OSRA experimental probability density of running time.</td>
<td>207</td>
</tr>
<tr>
<td>64. RA and OSRA average running time as a function of sample size.</td>
<td>208</td>
</tr>
<tr>
<td>65. RA and OSRA sample standard deviation as a function of sample size.</td>
<td>209</td>
</tr>
<tr>
<td>66. RA experimental probability density of running time.</td>
<td>210</td>
</tr>
<tr>
<td>67. OSRA experimental probability density of running time.</td>
<td>211</td>
</tr>
<tr>
<td>68. RA and OSRA average running time as a function of sample size.</td>
<td>212</td>
</tr>
<tr>
<td>69. RA and OSRA sample standard deviation as a function of sample size.</td>
<td>213</td>
</tr>
<tr>
<td>70. RA experimental probability density of running time.</td>
<td>214</td>
</tr>
<tr>
<td>71. OSRA experimental probability density of running time.</td>
<td>215</td>
</tr>
</tbody>
</table>
CHAPTER I
INTRODUCTION

This dissertation deals with division in residue number systems. A Residue Number System (henceforth abbreviated RNS) is a set of representations (known as "residue numbers" or "residue codes") of integers, together with rules for adding, subtracting and multiplying them. These operations are fast and simple because they don't involve carries, borrows or partial products. The operations are performed in each digit of the representation independently, in one step.

The division operation in an RNS is relatively slow and complicated. The fastest known division algorithm for arbitrary moduli (henceforth called the One Sided Rounding Algorithm, or OSRA) [14], has an average execution time which is approximately 50 times as long as that required for addition, subtraction or multiplication. Furthermore, it requires a large stored table.

In this dissertation, three improved division algorithms for the RNS are presented. One has a mean running time and standard deviation about the mean which are less than those of the OSRA, while using the same amount of storage. Another is a modification of this algorithm for performing signal division. The third uses as much as 70% less storage than the OSRA, and has a smaller mean running time and standard deviation about the mean.
A. Introduction to the RNS

A.1 The Residue Representation

The residue representation of an integer \( X \) is the \( n \)-tuple of positive integral remainders formed when \( X \) is divided by \( n \) specially chosen, fixed positive integers. These remainders are called the residue digits of \( X \) (or, "the residues of \( X \)", or "the residue code for \( X \)"), and are denoted by \( (x_1, x_2, \ldots, x_n) \). The specially chosen positive integers which produced them are called moduli, and are denoted by \( m_1, m_2, \ldots, m_n \). We will sometimes use the notation

\[
X \leftrightarrow (x_1, x_2, \ldots, x_n)
\]

(1.1)

to denote the phrase "\( X \) has the residue digits \( (x_1, x_2, \ldots, x_n) \)". Furthermore, we will sometimes refer to the integer remainder formed when \( X \) is divided by a positive integer \( j \) as "\( X \) modulo \( j \)" , and use subscripted parallel bars to denote the modulo operator, viz.,

\[
X \mod j = |X|_j .
\]

(1.2)

If the moduli are chosen relatively prime in pairs, then the residue representation of \( X \) is unique provided that we choose \( X \) from a set of consecutive integers containing no more than \( M \) elements, where \( M \) is the product of the moduli. This restriction on the moduli is made throughout the literature on RNS division algorithms, and will therefore be made in this dissertation. It is a consequence of the Chinese Remainder Theorem [33], which tells us that for the relatively prime (in pairs) moduli \( m_1, m_2, \ldots, m_n \), we can allow \( X \) to take a value from any set
of consecutive integers as long as the difference between the largest and smallest elements is at most M-1. In the literature, these sets of possible values are usually chosen as

\[ 0 < X < M-1 \]  

(1.3)

if only positive arithmetic is to be performed. If operands of both signs are desired (i.e., signed operands), then the interval

\[ -\frac{M-1}{2} < X < \frac{M-1}{2} \]  

(1.4)

is used if M is odd, or

\[ -\frac{M}{2} < X < \frac{M}{2} - 1 \]  

(1.5)

is used if M is even. The particular interval used is called the "interval of definition" (or "ID") for the RNS corresponding to \( m_1, m_2, \ldots, m_n \). It will sometimes be specified by enclosing, in brackets, the least and greatest integers in the ID. For example, "[0,M-1]" is the set of nonnegative integers less than or equal to M-1. Of the three new division algorithms to be presented later, one performs signed division, while two others assume positive operands only.

A.2 Addition, Subtraction and Multiplication

The residue operations of addition, subtraction and multiplication are done on individual digits, in parallel as follows. Let \( \cdot \) denote any of these operations, and let X and Y be any operands chosen from the ID, where
\[ X \leftrightarrow (x_1, x_2, \ldots, x_n) \quad (1.6) \]

and

\[ Y \leftrightarrow (y_1, y_2, \ldots, y_n) \quad (1.7) \]

Then,

\[ Z = X \circ Y \leftrightarrow (z_1, z_2, \ldots, z_n) \quad (1.8) \]

where \( z_i \) is defined as

\[ z_i = x_i \cdot y_i \mod m_i \quad (1.9) \]

As an example, consider the operands \( X = 47 \) and \( Y = 21 \) in the RNS defined by the moduli 11,13,17. For simplicity, we want to do positive arithmetic only, so we define the ID to be \([0,2430]\). The residue digits of \( X \) and \( Y \) are (3, 8, 13) and (10, 8, 4), respectively. Therefore, the residue codes for their sum and product are (2, 3, 0) and (8, 12, 1), respectively. As expected, the latter residue codes are those for the sum, 68, and product, 987.

If the ID were redefined as \([-1215,1215]\), we could perform signed operations. For example, let \( Y = -21 \), for which the residue code is (1, 5, 13). Then the sum and product of \( X \) and \( Y \) have the residue representations (4, 0, 9) and (3, 1, 16). As before, these are the correct residue codes for 26 and -987, respectively.

From the definition of addition, subtraction and multiplication, we see that the \( i \)th residue digit of the result depends solely on the \( i \)th residue digits of the operands. This means that the operations can be...
implemented in hardware using independent, small arithmetic units operating in parallel. The operations are performed in each arithmetic unit concurrently and independently, without the need for carries, borrows or partial products. Consequently, these operations can be done quickly.

A.3 Division

In contrast to the fast and simple operations of addition, subtraction and multiplication, the division operation is relatively slow. This is because the result of a division operation is not, in general, an integer, and therefore rounding of the quotient to a neighboring integer must be performed. Only if the quotient is known beforehand to be an integer, and only for certain types of divisors, can it be found as easily as the three preceding operations. The method used, called "integer division", will now be discussed.
A.3.1 Integer Division

Integer Division can be used to find the quotient if it is known beforehand that the quotient is an integer and if each residue digit of Y is relatively prime to its corresponding modulus. We will denote the numerator by X, and will assume that it and the divisor Y are contained in the ID. Thus, as before, we let

\[ X \leftrightarrow (x_1, x_2, \ldots, x_n), \]  

\[ Y \leftrightarrow (y_1, y_2, \ldots, y_n), \]  

and now using Z to denote the integral quotient, we let

\[ Z \leftrightarrow (z_1, z_2, \ldots, z_n). \]

It can be shown that if Y and each of the moduli possess no common divisors other than 1, then there exist unique integers \( y_i^{-1} \) such that

\[ 1 < y_i^{-1} < m_i - 1, \]

where \( y_i^{-1} \), called the multiplicative inverse of \( y_i \) modulo \( m_i \), is the unique integer such that

\[ y_i y_i^{-1} \text{ modulo } m_i = 1. \]

The integers \( y_i^{-1} \) can be found most easily by trial and error (for small moduli), and by the Euclidean Algorithm for large moduli. In such a case,

\[ z_i = x_i y_i^{-1} \text{ modulo } m_i. \]

Clearly, the integer division operation has the same "modular", "one-step" property possessed by addition, subtraction and multiplication,
and can therefore be performed just as quickly and simply.
Unfortunately, the conditions on its use are very restrictive.

For example, suppose we want to find $87/29 = 3$ using integer division in the modulus set $11, 13, 17$. The residue representations for $X$ and $Y$ are $(10, 9, 2)$ and $(7, 3, 12)$, respectively. The multiplicative inverse of $Y$, $Y^{-1}$, has the residue representation $(8, 9, 10)$, because $YY^{-1} = (1, 1, 1)$, which is the residue representation of $1$. Therefore,

$$87/29 = (10, 9, 2) (8, 9, 10) = (3, 3, 3),$$

which is the correct answer.

A.3.2 Non-Integer Division

If it is not known beforehand that the quotient is an integer, or if some residues of the divisor and their respective moduli have a common divisor other than one, then integer division cannot be used. In these cases, one of two different methods must be used. They are called "scaling" and "general division".

The scaling method, or "scaling procedure" as it is called, can only be used when the divisor is a product of first powers of moduli. It computes the "truncated quotient"

$$Z = \left\lfloor \frac{X}{Y} \right\rfloor,$$

i.e., the greatest integer less than or equal to the quotient $X/Y$, because in general rounding must be performed since the quotient is not an integer. The scaling procedure will be presented in detail in Chapter II.
The general division method is actually a generic term given to a class of methods, all of which find a neighboring integer to the quotient $X/Y$. Henceforth, these methods will be referred to as "RNS division algorithms" or "division algorithms". They are iterative in general, and therefore do not possess the speed and simplicity of addition, subtraction and multiplication.

They can be classified into two groups, as is the case with binary division algorithms [38]. These groups are referred to as the subtractive algorithms and the multiplicative algorithms. Members of the former employ subtraction of multiples of the denominator from the numerator, until the numerator becomes less (in absolute value) than the denominator. The multiplicative algorithms, however, compute the reciprocal of the divisor, and the truncated quotient by multiplication of this reciprocal by the numerator.

The Subtractive Algorithms

Subtractive algorithms find the truncated quotient by "long division". That is, they proceed by subtracting estimated multiples of the denominator from the numerator, and stop when the difference is less than (or close to) the denominator. At such a time, the truncated quotient is the sum of the estimates plus a small error term whose value is easy to determine.

Subtractive algorithms use the iterative procedure

$$X_{i+1} = X_i - E_{i+1}Y,$$  \hspace{1cm} (1.18)
where \( X_0 = X \), \( X_i \) is called the \( i \)th reduced numerator, and \( E_{i+1} \) is called the \( (i+1) \)st quotient estimate. The estimate is an educated guess of the value of the quotient \( X_i / Y \). The stopping condition is \( X_r < \bar{Y} \), where we have used an overbar to denote absolute value. When this condition is satisfied, the truncated quotient is

\[
\left| \frac{X}{Y} \right| = \sum_{i=0}^{r-1} E_{i+1} + E',
\]

where the error term, \( E' \), is

\[
E' = \begin{cases} 
0, & \text{if the sign of } X_r / Y \text{ is positive} \\
-1, & \text{if the sign of } X_r / Y \text{ is negative.}
\end{cases}
\]

The estimate and the rule by which it is found characterize a particular subtractive algorithm. In general, an estimate is found by residue computations in conjunction with stored tables. The values it can assume will modify the stopping condition of the algorithm, because it could be zero for certain values of \( X_i \) and \( Y \).

The Multiplicatively Algorithms

Multiplicatively algorithms are used with binary number systems. The closest one comes to a multiplicatively algorithm in the residue number system is the Vyshynskyy Algorithm [16], which computes a reciprocal of the divisor \( |S^2 / Y| \) where \( S \) is a suitably defined constant and the "ceiling" operator \( \lceil \cdot \rceil \) denotes the least integer greater than or equal to \( S^2 / Y \). However, no explanation is given in [16] as to how to use the reciprocal to find the truncated quotient. We suggest that it is
calculated by "dividing" (by $S^2$) the product of the numerator and the reciprocal, viz.,

$$\left\lfloor \frac{X}{Y} \right\rfloor = \left\lfloor \frac{X \cdot \frac{S^2}{Y}}{S^2} \right\rfloor,$$

As mentioned, no multiplicative algorithms have been published. However, this approach is used in the Reciprocal Algorithm, which is one of three new subtractive algorithms to be presented in this dissertation.

B. Previous Work

B.1. History of the RNS and Its Applications

The idea of residue number systems appears to have originated with the Chinese mathematician Chhin Chiu Shao [1]. In 1247 he stated and proved the Chinese Remainder Theorem, which is a statement of the uniqueness of representation for the residue code. (Interestingly, an example of a residue code was invented in the first century by Sun-Tsu [2], but no generalization of it has been found. As a result, Chhin is credited with the invention of the residue code).

Between 1247 and 1955 no further work on residue number systems was published. However, the elegant notation for integer congruence developed by Gauss in the nineteenth century is sometimes used today to describe residue operations.

Garner [6] (1958) introduced to this country the idea of using the RNS to implement computer arithmetic. He discussed addition, subtraction, negative number representation and residue to natural number conversion. By 1963, several publications had appeared, including those by Aiken and Semon [7], Driese et al. [8], Keir et al. [9], Tanaka et al. [10], Szabo [11], Lindamood and Shapiro [12], and Eastman [13]. The paper by Keir contains results concerning overflow detection, and Szabo presented necessary conditions for residue number sign detection. Lindamood and Shapiro considered magnitude comparison and overflow detection, and Eastman considered sign detection.


The majority of publications since 1955 have concerned the applications of RNS-based computers. In 1967, Szabo and Tanaka [21] published a comprehensive book on the RNS and its applications. Recent applications include optical computers (Collins [34,35], Habiby [36], Huang [22], and Guest [37]), digital filters, number theoretic transform and FFT implementation, as well as error detection and correction [26]. In particular, the optical processor applications include a
matrix-vector multiplier [22,36] and a decimal/residue/decimal converter [23]. Taylor [24] has published a paper which includes an extensive bibliography of recent work on RNS applications. Waser and Flynn [27] have published a textbook on digital arithmetic which contains a chapter on Residue Number Systems and their implementation in hardware.

B.2 Previous RNS Division Algorithms

There have been eight RNS division algorithms published to date. Henceforth, these will be referred to as the Keir [9], Szabo-Tanaka Modification [39], Tanaka [40], One Sided Rounding and Banerji-II [14], Kinoshita [28], Lin [30] and Vyshynskyy [16] Algorithms. The first seven of these are subtractive and will be discussed immediately. The discussion will focus primarily on the estimate used, because it is the characterizing feature of a particular subtractive algorithm. The Vyshynskyy algorithm is multiplicative, and will be described after the subtractive algorithms.

The Keir Algorithm [9] computes, in residue, the binary expansion of the truncated quotient. Its estimate $E_{i+1}$ is the largest power of 2 contained in $X_i/Y$. It is found essentially by trial and error, in two different ways. The first way is used to find the initial estimate, and uses sign detection and a binary search of a stored table containing residue encoded powers of 2. Each selected power of 2 is substituted for $E_1$ in the basic subtractive iteration

$$x_{i+1} = x_i - E_{i+1} y \quad . \quad (1.22)$$
The sign detection procedure is used on $X_1$ to determine whether the chosen power must be increased or decreased. The search continues until the largest power of 2 that does not exceed the truncated quotient $\lfloor X/Y \rfloor$ is found.

The second way is used to find subsequent estimates. It consists of dividing the initial estimate by successively higher powers of 2 and substituting each of the resulting quotients for the estimate $E_{i+1}$ in Equation (1.22). The sign detection procedure is used to determine whether the value that was substituted is too large. If it is, then it is divided by 2. Otherwise, it is used as the $(i+1)$st estimate, and the iteration proceeds. The algorithm stops when the successive divisions by 2 have reduced the initial estimate to less than 1.

The Szabo-Tanaka Modification Algorithm [39] is the same as the Keir Algorithm, except that it uses an additional stored table to facilitate the determination of the initial estimate. The extra table, indexed by mixed radix information about Y, stores residue encoded "good guesses" of the highest power of 2 contained in Y. This power of 2 is necessary to find the initial estimate. Any guess is guaranteed to differ from the desired value by at most a factor of 2, and therefore a single sign detection is used to detect a bad guess.

The preceding two algorithms use a "trial and error" approach to compute truncated quotients. As a result, their mean running time is relatively large. The One-Sided Rounding Algorithm, which will now be described, avoids this by making "educated guesses" about the truncated quotient, and as a result has a smaller mean running time.
The One Sided Rounding Algorithm (OSRA) [14] uses mixed radix information about rounded approximations of $X_i$ and $Y$ to determine the estimate. $X_i$ is rounded down and $Y$ is rounded up. Information derived from these rounded quantities is used to access two stored tables which contain residue encoded values of quotients and products of moduli from which the estimate is calculated. This algorithm stops when the estimate is equal to zero.

The OSRA is the most attractive RNS division algorithm published to date. However, it uses a large amount of storage (proportional to the square of the maximum modulus) and has a standard deviation of mean running time which can be as high as 40% for some modulus sets. It will be presented in detail at the end of Chapter II.

The Banerji-II Algorithm [14] is the same as the OSRA except that the divisor is rounded down to its most significant mixed radix term. It cannot be used for arbitrary modulus sets, because its stopping condition is unknown (assuming that it exists) for the arbitrary moduli case. For certain modulus sets, however, the authors were able empirically to find a suitable stopping condition. No information about these special modulus sets is given in [14]. This algorithm uses the same amount of storage as the OSRA.

The Kinoshita Algorithm [28] uses the Symmetric Residue Number System (SRNS), which is different from the RNS in that it allows residue digits to be of either sign. The use of signed residue digits facilitates sign detection [41].
Its estimate, like that used by the OSRA and the Banerji-II algorithm, is found in terms of rounded approximations of the mixed radix representations of $X_i$ and $Y$. $X_i$ is rounded down to its most significant nonzero mixed radix term. $Y$ is rounded to the product of a decimal "rounding parameter" and its most significant mixed radix term. The estimate is computed by using information derived from these rounded versions to access two stored tables, which contain residue encoded quotients and products of moduli. The table-derived quantities are multiplied to form the estimate. The algorithm stops when the absolute value of the $r$th reduced numerator, $\bar{X}_r$, is less than $\bar{Y}$.

The decimal-valued rounding parameter, used to find the approximation to $Y$, accounts for the effect of the next-most-significant mixed radix digit of $Y$. Its value depends on the two most significant nonzero mixed radix digits of $Y$ as well as their positions. Its calculation is not discussed in [28]. However, it most likely must involve the use of a decimal divider, because a stored table would be exceedingly large. It would be of size proportional to the sum of ordered length-2 products of moduli.

The Tanaka Algorithm [40] uses the scaling and base extension procedures to compute its estimates. The denominator $Y$ is rounded to an approximate divisor $\tilde{Y}$, which is defined to be the nearest product of first powers of moduli that equals or exceeds $Y$. The $i$th reduced numerator, $X_i$, is then scaled by $\tilde{Y}$, and the result is base extended to generate the estimate. The algorithm stops when the estimate is equal to zero.
The rounding of $Y$ to $\tilde{Y}$ is done by a stored table, which is indexed by the most significant nonzero mixed radix digit of $Y$. The table contains binary masks whose one-bits indicate the moduli contained in $\tilde{Y}$.

The Tanaka Algorithm can only be used for particular modulus sets. These sets must contain small moduli, because, for small divisors, $\tilde{Y}$ must likewise be small. Modulus sets which contain small moduli are undesirable, because the number of residue operations required for the mixed radix conversion and sign detection procedures is proportional to the number (not the size) of the moduli.

The Lin Algorithm [30] is an adaptation to residue of the well known CORDIC Algorithm. Essentially, it computes the truncated quotient by finding, in residue, the terms in its binary expansion. It does this by substituting a fixed series of successively smaller powers of 2 for the estimate $E_{i+1}$ in the basic subtractive iteration

$$X_{i+1} = X_i - E_{i+1}Y,$$  \hspace{1cm} (1.23)

and using sign detection on $X_{i+1}$. This algorithm stops when the estimate is reduced to less than 1 after some $X_{i+1}$ is negative.

The successively smaller powers used as estimates change only in sign. Their magnitudes do not change. The sign of an estimate $E_{i+1}$ is equal to the sign of $X_i$. The estimates start at a design-specified power of 2 which is "approximately" equal to the largest modulus. The sign detection procedure is an adaptation to the RNS of the "exhaustion of cases" technique developed by Kaushik and Arora [31], which detects sign using two simultaneous residue operations performed on a vast
number of operands simultaneously. This algorithm requires the use of
an overflow modulus whose size is equal to the maximum power of 2 in the
fixed series of estimates.

The Lin Algorithm uses an amount of hardware that is proportional
to $M$, the product of the moduli, because its sign detection procedure
uses simultaneous processing of a large number of operands.
Consequently, this algorithm is unacceptable for use with "practical"
modulus sets (i.e., those for which $M$ is large).

The Vyshynskyy Algorithm [16] computes, to an error of 1, a
suitably defined reciprocal of the divisor. It is not a "division
algorithm" per se. It defines the reciprocal to be $|S^2/Y|$, where $S$ is
the largest mixed radix coefficient. It assumes that the divisor $Y$ is
not less than $S$.

This algorithm finds the reciprocal by using a first order
iterative process. This process is initialized by a stored-table-
derived estimate of the reciprocal, and produces a series of integers
which converge to it. This series of reciprocal approximations is
computed using the mixed radix digits of $Y$, together with the scaling
and base extension procedures and an overflow modulus whose size is at
least twice the size of the maximum modulus. The algorithm stops after
$n$ iterations. That is, for any divisor in the specified subset of the
ID, the algorithm uses a constant number, $n$, of iterations, and this
number is equal to the number of moduli.

The Vyshynskyy Algorithm does not work for all divisors in the ID.
It can only be used for divisors which are not less than the largest
mixed radix coefficient. Furthermore, it requires the use of an extra "overflow" modulus whose size is at least twice the maximum modulus. This is undesirable, because we would like any overflow moduli to be of size approximately equal to the largest.

In summary, all of these algorithms have drawbacks. In particular, five of them have drawbacks which are severe enough to prevent their consideration for use in general purpose RNS-based computers. The Tanaka and Banerji-II algorithms can be used only for very restricted modulus sets. The Kinoshita Algorithm requires the use of a decimal divider in order to avoid a stored table whose size is proportional to the sum of ordered double products of moduli. The Lin Algorithm requires an amount of hardware which increases linearly with M, because the sign detection method it uses relies on simultaneous processing of operands. The Vyshynskyy Algorithm does not work for all divisors, and requires the inclusion of an overflow modulus whose size is too large.

The three remaining algorithms (Keir, Szabo-Tanaka Modification and One Sided Rounding) do not have the limitations mentioned above. However, of these three, the OSRA is the fastest [32]. For this reason, the OSRA will be considered the most attractive RNS division algorithm published to date, and will be the algorithm with which our new algorithms are compared. However, as mentioned above, the OSRA has two shortcomings. The first is that its storage, although reasonable, is quite large. It requires an amount proportional to the square of the maximum modulus. This large amount is needed because the majority of the quotient estimates are stored rather than computed. The second
drawback of the OSRA is that the standard deviation about the mean of its execution time is large. As mentioned previously, it can be as large as 40% for some modulus sets.

B.3 Discussion of the New Algorithms

The three new algorithms presented in this dissertation each improve on the OSRA in at least one respect. They all resemble the OSRA in that they are subtractive and use mixed radix information about $X_i$ and $Y$ to determine quotient estimates. The first improves the mean running time and standard deviation about the mean without using any more storage. It performs positive (unsigned) division, and uses "two-sided", rather than "one-sided", rounding on the most significant mixed radix digit of $Y$. The second new algorithm is a modification of the first to allow signed division, which cannot be done by the OSRA. It has a smaller standard deviation, while using the same amount of storage. The third one employs a suitably defined integer reciprocal of the divisor. It computes, rather than stores, quotient estimates. It is faster than the OSRA, and has a smaller standard deviation of running time about the mean. Most significantly, it reduces the OSRA storage by as much as 70%.

C. Statement of the Problem and Why It Is Important

The problem that we set out to solve is to invent an RNS division algorithm that possesses smaller mean and standard deviation of running time than the OSRA, while not using more storage. This problem is
important because such an algorithm would enhance the appeal of the RNS for use as a general purpose computer number system. Although the RNS has been shown to be faster than binary for applications in which addition, subtraction and multiplication predominate [25], it has not received much consideration for general purpose use.

The results of this research are three algorithms, called the Two Sided Rounding Algorithm (TSRA), the Signed Algorithm (SA), and the Reciprocal Algorithm (RA). The TSRA uses the same amount of storage as the OSRA, and possesses a smaller standard deviation about the mean running time and a slightly smaller mean running time. The SA is a modification of the TSRA which performs signed division. It has a smaller standard deviation about the mean running time, while using the same amount of storage. The RA uses as much as 70% less storage than the OSRA, and has a smaller mean running time and slightly better standard deviation about the mean.

D. Outline of the Dissertation

Chapter II contains background material. Specifically, there is a review of several commonly used residue procedures and a detailed explanation of the OSRA. The commonly used procedures include mixed radix conversion, scaling, sign detection and base extension.

Chapter III contains a discussion of the first two new algorithms, namely the Two Sided Rounding and Signed Algorithms. They are similar in nature, both using two-sided approximations to make quotient estimates, and therefore are grouped together in the same chapter. The
chapter includes examples of their use as well as their formal statements and proofs of their validity. It also contains computationally derived performance data and formulas for the storage used by each of these algorithms. The chapter is concluded with a theoretical derivation and computer verification of a lower bound for the mean running time of the OSRA.

Chapter IV contains a discussion of the Reciprocal Algorithm. It includes examples of its use as well as its formal statement and a proof that it is correct. The chapter also contains formulas for the amount of storage it uses, and computationally derived performance data.

Chapter V is a summary of the dissertation and conclusions of the work, and contains suggestions for further study.
CHAPTER II
DETAILS OF THE RNS AND THE OSRA

This chapter contains background on specific concepts which will be used later in the dissertation. The first ones deal with the arithmetic procedures called scaling, mixed radix conversion, sign detection and base extension. These are used by the algorithms to follow, and are given in Section A. The last item is a discussion of the One Sided Rounding Algorithm (OSRA). This discussion, Section B, serves as a review of the most attractive RNS division algorithm published to date, as a detailed introduction to subtractive algorithms, and as an indication of the method of presentation for the new algorithms to follow.

A. Four Basic Residue Procedures

This section contains descriptions of four basic residue procedures called scaling, mixed radix conversion, sign detection and base extension. The notation used in Section A is the same as that used in Chapter I.

A.1 The Scaling Procedure

As mentioned in Chapter I, the Scaling Procedure computes the truncated quotient $|X/Y|$ when $Y$ is a product of first powers of moduli. It can be thought of as the RNS equivalent of radix division (right shifting) used with positional number systems.
To begin the discussion, we will consider the simplest case, which is when \( Y \) is equal to a single modulus \( m_i \). In such a case, the truncated quotient is given by

\[
\left\lfloor \frac{X}{Y} \right\rfloor = \frac{X-x_i}{Y}
\]  

(2.1)

Since \( \lfloor X/Y \rfloor \) is an integer, and any residue digit of \( Y \) is relatively prime to its corresponding modulus, we can find the truncated quotient by subtracting \( x_i \) from \( X \), and using the integer division method explained in Chapter I. We have that the \( j \)th residue digit of the truncated quotient is

\[
\left\lfloor \frac{X}{Y} \right\rfloor \mod m_j = (X-x_i) m_i^{-1} \mod m_j
\]  

(2.2)

for all \( j \neq i \). The \( i \)th residue digit cannot be found because \( m_i^{-1} \mod m_i \) does not exist. It has been "erased" by the scaling procedure and must be "restored" by a method called the Base Extension Procedure, which will be explained in Section A.4.

As an example, consider the problem of finding \( \lfloor 59/13 \rfloor \) in the modulus set \( \{11, 13, 17\} \). The residue representations of \( X \) and \( Y \) are \( X \leftrightarrow (4, 7, 8) \) and \( Y \leftrightarrow (2, 0, 13) \). The multiplicative inverse of \( Y \) (in all moduli except the second) is \( (6, d, 4) \), where the symbol "d" in a digit position means that the digit is undefined. The truncated quotient is found as the result of a subtraction (of \( X \mod 13 \)) from \( X \), followed by a multiplication of this difference by the inverse. This is
As expected, (4, d, 4) is the residue code for the truncated quotient, which is 4.

We can now consider the more complicated general case when the divisor is a product of \( k \) single powers of moduli, i.e.,

\[
Y = m_1^{i_1} m_2^{i_2} \cdots m_k^{i_k}.
\] (2.3)

In this case, the truncated quotient can be found by successive applications (one for each of the \( k \) moduli) of the single modulus method explained above. This is because it can be shown that the truncated quotient is preserved throughout successive scalings by single moduli, i.e.,

\[
\begin{bmatrix}
\frac{X}{m_1^{i_1} m_2^{i_2} \cdots m_k^{i_k}} \\
\frac{X}{m_1^{i_1} m_2^{i_2} \cdots m_k^{i_k}} \\
\frac{X}{m_1^{i_1} m_2^{i_2} \cdots m_k^{i_k}} \\
\end{bmatrix}
= \left(\begin{array}{c}
\frac{X}{m_1^{i_1}} \\
\frac{X}{m_2^{i_2}} \\
\frac{X}{m_3^{i_3}} \\
\end{array}\right)
= \left(\begin{array}{c}
\frac{X}{m_1^{i_1} m_2^{i_2}} \\
\frac{X}{m_1^{i_1} m_3^{i_3}} \\
\frac{X}{m_1^{i_1} m_4^{i_4}} \\
\end{array}\right) = \cdots .
\] (2.4)

The Scaling Procedure computes the truncated quotient in the \( n - k \) moduli which do not divide \( Y \). For future reference, we note that scaling requires \( 2k \) residue operations to perform.
A.2 The Mixed Radix Conversion Procedure

The Mixed Radix Conversion Procedure is a method of converting the residue representation of an operand to its representation in a positional number system called the Mixed Radix System. Magnitude information about an operand is more easily determined from its mixed radix digits than from its residue digits, because mixed radix is a weighted number system, whereas residue is not. Consequently, the mixed radix conversion procedure is widely used by the various division algorithms.

The mixed radix representation of a positive integer $X$ is defined to be

$$X = x_n m_1 + x_{n-1} m_2 + \ldots + x_2 m_{n-1} + x_1,$$

(2.5)

where $x_i$ is the $i$th mixed radix digit of $X$, and $0 < x_i < m_i$ for all $i$. Note that this is similar to the standard "fixed radix" representation

$$X = x_n r^{n-1} + x_{n-1} r^{n-2} + \ldots + x_2 r + x_1,$$

(2.6)

in that $r^{n-1}$ has been replaced by a product of $n-1$ moduli $\prod_{i=1}^{n-1} m_i$. The digit $x_n$ is the most significant mixed radix digit, and $x_1$ is the least significant. We will represent mixed radix digits by angle brackets with the most significant digit on the left, and will sometimes denote the phrase "$X$ has the mixed radix digits $x_1, x_2, \ldots, x_n$" by

$$X \leftrightarrow \langle x_n, x_{n-1}, \ldots, x_1 \rangle.$$

(2.7)
The residue-to-mixed radix conversion procedure is straightforward, and is done as follows. We will let $X$ have the residue digits $r_1, r_2, \ldots, r_n$. For the first mixed radix digit, by taking both sides of Equation (2.5) modulo $m_1$, we have $X \mod m_1 = x_1$. But $X \mod m_1 = r_1$. Therefore, the least significant mixed radix digit of $X$ is its first residue digit. For the remaining mixed radix digits, the procedure is to subtract and divide. The second mixed radix digit of $X$ is obtained as the second residue digit of the result of scaling $X$ by $m_1$. That is,

$$
\left\lfloor \frac{X}{m_1} \right\rfloor \mod m_2 = x_2. \quad (2.8)
$$

This can be seen by scaling Equation (2.5) by $m_1$, and taking the result modulo $m_2$. Similarly, scaling Equation (2.5) by $m_1 m_2$, and taking the result modulo $m_3$, we obtain

$$
\left\lfloor \frac{X}{m_1 m_2} \right\rfloor \mod m_3 = \left\lfloor \frac{X}{m_2} \right\rfloor \mod m_3 = x_3. \quad (2.9)
$$

Therefore, the third mixed radix digit is the third residue digit of the result obtained by scaling $X$ by the product $m_1 m_2$. Proceeding similarly, we see that

$$
\left\lfloor \frac{X}{\prod_{j=1}^{i+1} m_j} \right\rfloor \mod m_{i+1} = x_{i+1}. \quad (2.10)
$$

In each case, the $(i+1)$st mixed radix digit is the $(i+1)$st residue digit of the result of scaling $X$ by $\prod_{j=1}^{i} m_j$. This procedure, called the Mixed Radix Conversion Procedure, is used by every division algorithm that
will be presented in this dissertation. Note that it requires \(2(n-1)\) residue operations to perform.

The mixed radix digits of a negative integer \(X\) in the ID are defined in terms of the moduli product \(M\) to be those of the positive quantity \(M+X\).

### A.3 The Sign Detection Procedure

The sign of an operand \(X\) is determined from its mixed radix digits. The procedure is to compare them with those of the maximum positive integer in the ID (where the ID is chosen, for example, to be one of those given by Equations (1.4) or (1.5)). If the mixed radix digits of \(X\) are those of an integer greater than the maximum, then \(X\) is negative. Otherwise, it is positive. Therefore, the Sign Detection Procedure consists of a mixed radix conversion followed by a magnitude comparison.

As an example, consider finding the sign of the integer \(X\) whose residue code is \((10,3,11)\) in the RNS defined by the modulus set \(\{11,13,17\}\), where the ID is defined to be \([-1215,1215]\). The mixed radix digits of \(X\) are found by the following application of the Mixed Radix Conversion Procedure, where, as before, we have used blanks to denote undefined residue digits:
We find that the mixed radix digits of $X$ are $<16,10,10>$. Since the maximum positive integer in the ID, 1215, has the mixed radix digits $<8,6,5>$, we conclude that $X$ is negative. Furthermore, since the mixed radix digits of $X$ are those of the positive integer 2408, we have that $X=2408-2431=-23$, where the product of the moduli, $M$, is 2431 for this modulus set.

A.4 The Base Extension Procedure

The Base Extension Procedure is a method for computing unknown residue digits of an operand. It is used to restore residue digits that were erased by the scaling method, as well as generate residue digits corresponding to extra "overflow" moduli.
The algorithms to be presented in this dissertation use a special form of the Base Extension Procedure called a "zero-operation" base extension, or "broadcast". Only this special case will now be discussed.

A broadcast is a base extension used when the operand, X, is nonnegative and has only one residue digit in its representation. In such a case, all residue digits of X are found simultaneously. This can be done by using, for example, a stored table which contains the residue representations for all nonnegative integers less than the maximum modulus. Such a table would be of size equal to the maximum modulus. Note for future reference that a broadcast requires no residue operations, and therefore is called a "zero-operation" base extension.

A.5 Summary

Section A is a discussion of four commonly used residue procedures, called the Scaling, Mixed Radix Conversion, Sign Detection and Base Extension Procedures. Each was described in sufficient detail in order to prepare the reader for the discussions of the division algorithms to be presented.

B. The One Sided Rounding Algorithm

This section contains a presentation of the One Sided Rounding Algorithm (OSRA). As mentioned previously, it is the most attractive RNS division algorithm published to date. It is subtractive and uses a "rounded up" approximation of the divisor and a stored table of quotients to determine quotient estimates.
The section begins with a discussion of notation and an introduction to the algorithm. Subsequently, the algorithm is summarized in words and flowchart form. This is followed by a discussion of the figures of merit which will be used to evaluate all algorithms in the dissertation. This will be illustrated by an example problem. The OSRA is then formally stated, and shown to be correct. The storage used by the algorithm, which is used as another figure of merit, will be given in Section C of Chapter III. Also included in Section C are computationally derived values of mean and standard deviation of running time.

B.1 Notation and Discussion of the OSRA

B.1.1 Notation

The notation used here will be used in similar situations throughout the dissertation. As mentioned earlier, we will assume numerator X and denominator Y are initially given in residue representation with respect to the pairwise-relatively-prime moduli \( m_1, m_2, \ldots, m_n \), where the moduli are assumed to be ordered as \( m_n > m_{n-1} > \ldots > m_1 \). We will also represent X and Y using the mixed radix representation, where we will use \( \ell \) and \( y_\ell \) to denote the position and value of the most significant nonzero mixed radix digit of the denominator, respectively, and will use \( k \) and \( x_k \) to denote the corresponding quantities for the \( i \)th reduced numerator \( X_i \). That is, the mixed radix expansions of \( X_i \) and \( Y \) are assumed to be
The One Sided Rounding Algorithm finds the truncated quotient \( \lfloor X/Y \rfloor \) for nonnegative \( X \) and \( Y \) in the ID [0,M-1]. It finds the truncated quotient in a way that is typical of all division algorithms to be presented in this dissertation, and therefore will be given after a discussion of the features that are common to all these algorithms.

B.1.2.1 Discussion of Subtractive Algorithms in General

The OSRA along with the three algorithms that will be presented is subtractive. As discussed earlier, such algorithms find the truncated quotient by the well-known grade school method of "estimating and subtracting". That is, it finds the truncated quotient by iteratively subtracting estimated multiples of the denominator from the numerator until the absolute value of the difference is less than the absolute value of the denominator. At such a time, the truncated quotient is equal to the sum of the "estimates" plus a small error term. Loosely speaking, it proceeds by making an "educated guess" (or "estimate") for the quotient. If the guess is not correct, it is used to form a new division problem whose numerator is smaller (in absolute value) than the
original. The process is then repeated, with a guess being made of the quotient of the new problem. In essence, this process is one of substitution of a series of "smaller" (and hopefully easier) division problems for the original.

B.1.2.2 Discussion of the OSRA

The OSRA uses an iterative procedure,

$$X_{i+1} = X_i - E_{i+1} Y,$$  \hspace{1cm} (2.13)

where $X_0 = X$ and $E_{i+1}$ is the estimate ("educated guess") of the quotient $X_i/Y$. The OSRA, like the algorithms to come, has a particular stopping condition. It stops in the $r$th iteration when $X_r = 0$ or when the $(r+1)$st quotient estimate $E_{r+1} = 0$. When either of these stopping conditions is met, the truncated quotient is found from the sum of the estimates plus a small error term $E'$, viz.,

$$\left\lfloor \frac{X}{Y} \right\rfloor = \sum_{i=0}^{r-1} E_{i+1} + E',$$  \hspace{1cm} (2.14)

where

$$E' = \begin{cases} 
0, & \text{if } X_r = 0 \\
0, & \text{if } 0 < X_r < Y \\
1, & \text{if } X_r > Y .
\end{cases}$$  \hspace{1cm} (2.15)

The quotient estimate used by the OSRA is
\[
E_{i+1} = \begin{cases} 
\frac{x_k}{y_{k+1}}, & \text{if } k < \varepsilon \\
0, & \text{if } k = \varepsilon \\
\frac{m_k}{y_{k+1}}, & \text{if } k = \varepsilon + 1 \\
x_k \frac{m_k}{y_{k+1}} \cdot m_{k+1} \cdots m_{k-1}, & \text{if } k > \varepsilon + 1
\end{cases}
\]

(2.16)

The quantities \( m_k \) and \( m_x \) are the moduli corresponding to the most significant nonzero mixed radix digits \( x_k \) and \( y_k \) of the \( i \)th reduced numerator \( X_i \) and denominator \( Y \), respectively. The relative sizes of \( x_k \) and \( y_k \), as well as \( k \) and \( \varepsilon \), give information on the relative sizes of the \( i \)th reduced numerator and denominator.

In practice, the evaluation of \( E_{i+1} \) in Equation (2.16) is straightforward. Its factors are obtained in residue code and multiplied. The factors come from stored tables, or are the result of short residue calculations.

The quotient estimate \( E_{i+1} \) and the formula for finding it are the "heart" of any subtractive algorithm. It is here that the running time and storage requirements are largely determined, and the fundamental differences between the subtractive algorithms arise. Accordingly, the remainder of this section deals with the OSRA estimate and an examination of its properties.

The expression for the estimate in Equation (2.16) is found by going to mixed radix approximations of the \( i \)th reduced numerator \( X_i \) and
Y. This is done because the mixed radix representation allows rounding and magnitude comparison. Later, we will go back to using the residue representation when actually evaluating it.

For the first step in deriving the estimate, the mixed radix expression for the denominator Y is rounded up to its next largest mixed radix term, giving

\[ Y = (y_{k+1} + 1)m_{k-1} \cdots m_1. \] (2.17)

The \( i \)th reduced numerator, \( X_i \), is approximated with its most significant nonzero mixed radix term as

\[ X_i = x_k m_{k-1} \cdots m_1 \] (2.18)

Combining Equations (2.17) and (2.18) and cancelling those moduli common to numerator and denominator, we get an approximation of \( X_i/Y \) that depends on the relative magnitudes of \( X_i \) and \( Y \) as expressed by the values of \( k \) and \( \ell \), viz.,

\[
\frac{X_i}{Y} = \begin{cases} 
\frac{x_k}{(y_{\ell+1} + 1)m_{k-1} \cdots m_k} & \text{if } k<\ell \\
\frac{x_k}{y_{\ell+1}} & \text{if } k=\ell \\
\frac{x_k m_\ell}{y_{\ell+1}} & \text{if } k=\ell+1 \\
\frac{x_k m_\ell}{y_{\ell+1} m_{\ell+1} \cdots m_{k-1}} & \text{if } k>\ell+1
\end{cases}
\] (2.19)
Note that the rounded versions of $X_i$ and $Y$ have been specially chosen so that the approximation never exceeds $X_i/Y$, and is therefore "one-sided". This is how the OSRA got its name. For some algorithms to be presented later, the estimate is allowed to exceed $X_i/Y$, and is then called a "two-sided" estimate.

The approximation in Equation (2.19) is unsuitable because it produces fractional values, while the RNS is an integer number system. Therefore, the values in Equation (2.19) are truncated to give

$$
\frac{X_i}{Y} = \begin{cases} 
\frac{x_k}{(y_{\ell}+1)^{m_{\ell-1}} \cdots m_k}, & \text{if } k < \ell \\
\frac{x_k}{y_{\ell}+1}, & \text{if } k = \ell \\
\frac{x_km_k}{y_{\ell}+1}, & \text{if } k = \ell + 1 \\
\frac{x_km_km_{\ell}+1 \cdots m_{k-1}}{y_{\ell}+1}, & \text{if } k > \ell + 1.
\end{cases}
$$

(2.20)

Further approximation of Equation (2.20) is needed because the truncated quotients in it are difficult to find using residue operations. The scaling operation cannot be used because in general the quantity $y_{\ell}+1$ is not a product of first powers of moduli. An effective alternative is to use a stored table. Such a table could store the residue codes for all required quotients with denominator $y_{\ell}+1$. However, such a table would be exceedingly large, because it must be

35
indexed by \( x_k, y_k, \ell \) and \( k \). To avoid this we approximate part of Equation (2.20) further, using, for \( k=\ell+1 \),
\[
\left[ \frac{x_k m_2}{y_k+1} \right] = x_k \left[ \frac{m_2}{y_\ell+1} \right]
\]
(2.21)

and, for \( k>\ell+1 \),
\[
\left[ \frac{x_k m_2 m_2+1 \cdots m_k-1}{y_\ell+1} \right] = x_k \left[ \frac{m_2}{y_\ell+1} \right] m_{\ell+1} \cdots m_{k-1} \cdot
\]
(2.22)

These approximations allow the estimate in Equation (2.20) to be calculated using two stored tables of relatively small size. One of these stores residue coded values of \( \lfloor \frac{\alpha}{y_\ell+1} \rfloor \), for all possible values of \( \alpha \) and \( y_\ell \). The other stores residue coded values of all possible products of the form \( m_{\ell+1} \cdots m_{k-1} \). Using Equations (2.21) and (2.22) in Equation (2.20), we have the final expression for the quotient estimate used by the OSRA, viz.

\[
E_{\ell+1} = \begin{cases} 
0 & \text{if } k<\ell \\
\left[ x_k \frac{m_2}{y_\ell+1} \right] & \text{if } k=\ell \\
x_k \left[ \frac{m_2}{y_\ell+1} \right] & \text{if } k=\ell+1 \\
x_k \left[ \frac{m_2}{y_\ell+1} \right] m_{\ell+1} \cdots m_{k-1} & \text{if } k>\ell+1.
\end{cases}
\]
(2.23)
At this point we have been thinking in terms of the mixed radix representation with $x_k$, $y_\ell$ and $m_{\ell+1} \ldots m_{k-1}$. It is explained below how the residue representation of $E_{i+1}$, needed to calculate $X_{i+1}$, is found.

In order to calculate the residue code for $E_{i+1}$, the residue codes for each of $x_k$, $\lfloor x_k/y_\ell+1 \rfloor$, $\lfloor m_\ell/y_\ell+1 \rfloor$ and $m_{\ell+1} \ldots m_{k-1}$ are needed. The residue code for $x_k$ is found by a zero-operation base extension (discussed earlier) of $x_k$ from the single modulus $m_k$ to all other moduli. For the remaining quantities, the desired information is found with the aid of two stored tables. One of the tables stores residue encoded values of $\lfloor \alpha/y_\ell+1 \rfloor$ for $\alpha=1,2,\ldots,m_{\ell-1}$ and $y_\ell=1,2,\ldots,m_{\ell-1}$. It is indexed by $x_k$ (or $m_\ell$) and $y_\ell$. The other table stores residue encoded products $m_{\ell+1} \ldots m_{k-1}$, and is indexed by $k$ and $\ell$ for $k>\ell+1$. Expressions for the size of these tables is derived in Section C.1 of Chapter III.

There is one exception to the OSRA estimate given in Equation (2.23), and it occurs only for the $n$ divisors which are equal to the mixed radix coefficients (i.e., when $Y=m_{\ell-1} \ldots m_1$ for $\ell=1,2,\ldots,n$). In such cases, $Y$ need not be rounded up, and the special estimate

$$E_{i+1}' = \begin{cases} 0 & \text{, if } k<\ell \\ x_k & \text{, if } k=\ell \\ x_k m_\ell & \text{, if } k=\ell+1 \\ x_k m_\ell m_{\ell+1} \ldots m_{k-1} & \text{, if } k>\ell+1 \end{cases} \quad (2.24)$$
(which is Equation (2.23) evaluated when $y_L$ is not incremented) is used. This special estimate represents an improvement of the algorithm published by Banerji [14], in that it avoids the use of the Scaling and Base Extension Procedures used for these divisors.

B.2 Summary of the OSRA

To summarize, the OSRA is shown in flowchart form in Figure 1. In the flowchart, the input is shown at the top and the output is shown at the bottom right. The main steps in the flowchart are: 1) the conversion to the mixed radix representation so that the estimate $E_{i+1}$ can be found; 2) determination of the estimate from the table; and 3) the test to see if the stopping condition is satisfied. After the stopping condition is met, the error $E'$ is determined, and the truncated quotient is the value of the variable $Q$, which is the "running quotient sum".

The steps of the OSRA are:

**Step 1)** Set $Q=0$ and the iteration counter $i=0$.

**Step 2)** If $X_i=0$, then Stop with answer $Q$.

**Step 3)** If $i=0$ (that is, if this is the first iteration), then do a mixed radix conversion on $Y$, thus determining $\ell$ and $y_L$.

**Step 4)** Do a mixed radix conversion on $X_i$, thus determining $k$ and $x_k$.

**Step 5)** If $k<\ell$, then Stop with answer $Q$.

**Step 6)** If $k>\ell$, then go to Step 7). Otherwise, $k=\ell$, so do the following:
Figure 1. Flowchart of the OSRA.
i) If $Y$ is not a mixed radix coefficient, then access the stored table of quotients $\lfloor x_k/y_k + 1 \rfloor$ with $x_k$ and $y_k$ to get the residue code for $E_{i+1}$. Go to Step 9).

ii) Otherwise, $Y$ is a mixed radix coefficient, so set $E_{i+1} = x_k$ (from Equation (2.24)) and go to Step 9).

**Step 7** If $k > k+1$, then go to Step 8). Otherwise, $k = k+1$, so do the following:

i) If $Y$ is not a mixed radix coefficient, then compute $E_{i+1} = x_k \lfloor m_k/y_k + 1 \rfloor$ where the residue code for $x_k$ is found by a zero-operation base extension and the residue code for $\lfloor m_k/y_k + 1 \rfloor$ is found from the quotient table, indexed by $m_k$ and $y_k$. Go to Step 9).

ii) Otherwise, $Y$ is a mixed radix coefficient, so compute $E_{i+1} = x_k m_k$ in residue, and go to Step 9).

**Step 8** If $Y$ is not a mixed radix coefficient, then compute $E_{i+1} = x_k \lfloor m_k/y_k + 1 \rfloor m_{k+1} \ldots m_{k-1}$ in residue, where the residue codes for $x_k$ and $\lfloor m_k/y_k + 1 \rfloor$ are found as in Step 7), and the residue code for $m_{k+1} \ldots m_{k-1}$ is found from the table of modulus products, indexed by $k$ and $k$. Otherwise, $Y$ is a mixed radix coefficient, so compute $E_{i+1} = x_k m_k m_{k+1} \ldots m_{k-1}$ in residue.
Step 9) If $E_{i+1} \neq 0$ then add $E_{i+1}$ to $O$, and compute the new numerator $X_{i+1} = X_i - E_{i+1}Y$. Add 1 to the iteration counter $i$, and go to Step 2).

Step 10) Determine $E'$ by comparing the mixed radix digits of $X_i$ and $Y$. If $X_i > Y$, then add $E' = 1$ to $O$ and Stop. Otherwise, Stop.

B.2.1 Evaluation of the OSRA

The OSRA and the other algorithms presented in this dissertation are evaluated and compared in this section using three figures of merit. These are mean running time, standard deviation about the mean, and stored table size.

Running time is measured in units of elementary residue operations required for finding the truncated quotient. That is, the speed of an algorithm in solving a particular division problem is defined to be the number of residue additions, subtractions and multiplications used. This is the method prevalent in the literature for measuring the speed of RNS algorithms (see, for example [9,14,17 and 29]). Standard deviation of the running time is largely ignored in the literature, and is included as a measure of the significance of mean running time values. Smaller values of standard deviation imply that there is a smaller chance of encountering a very long division problem. This has obvious implications for consistently fast throughput. Stored table size is almost universally used as a figure of merit for algorithm evaluation.
B.3 Example of the Use of the OSRA

In this example, we choose the modulus set \( \{m_3, m_2, m_1\} = \{17, 13, 11\} \), for which the ID is \([0, 2430]\). We want to find \( \lfloor 2425/140 \rfloor \), where the residue codes for the operands are \( X \leftrightarrow (11, 7, 5) \) and \( Y \leftrightarrow (4, 10, 8) \). As mentioned before, mixed radix digits will be enclosed in angle brackets with the least significant digit on the right. Residue digits will be enclosed in parentheses. The solution step number corresponds to the number of the same step in the summary.

Reference is made to the operation count. This is the total number of residue additions, subtractions and multiplications that have been used so far, and it is contained in the operation counter. Note that the number of operations required for a mixed radix conversion in this example is 4, because the number of moduli \( n=3 \), and \( t=2(n-1) \), as mentioned in Section A of this chapter.

Solution: 3) Do a mixed radix conversion on \( Y \) to get \( Y \leftrightarrow <0, 12, 8> \).

Therefore \( \lambda = 2 \) and \( y_2 = y_2 \). Add 4 to the operation counter, because of the mixed radix conversion.

4) Do a mixed radix conversion on \( X_0 \) to get \( X_0 \leftrightarrow <16, 12, 5> \).

Therefore \( k = 3 \) and \( x_k = x_3 = 16 \). Add 4 to the operation count because of the mixed radix conversion.

7) Since \( k = \lambda + 1 \), we compute \( E_1 = x_k \left\lfloor \frac{-m_2}{y_2} + 1 \right\rfloor \) from Equation (2.23), using \( x_3 = 16 \leftrightarrow (16, 3, 5) \) and \( \left\lfloor \frac{-m_2}{y_2} + 1 \right\rfloor = 1 \leftrightarrow (1, 1, 1) \) obtained from the quotient table. We have

42
Add 1 to the operation counter to account for the multiplication operation.

9) The new numerator $X_1$ is computed in residue using $X_1 = X_0 - E_1 Y$. We have

<table>
<thead>
<tr>
<th>Moduli: 17 13 11</th>
</tr>
</thead>
<tbody>
<tr>
<td>$E_1$: 16 3 5</td>
</tr>
<tr>
<td>$Y$: 4 10 8</td>
</tr>
<tr>
<td>$E_1 Y$: 13 4 7</td>
</tr>
<tr>
<td>$X_0$: 11 7 5</td>
</tr>
<tr>
<td>$X_1$: 15 3 9</td>
</tr>
</tbody>
</table>

Add 2 to the operation counter. Also, the estimate is added to $Q$ to get $Q \leftarrow (0,0,0) + (16,3,5) = (16,3,5)$. Add 1 to the operation counter.

4) Do a mixed radix conversion on $X_1$ to get $X_1 \leftarrow \langle1,3,9\rangle$. Therefore, $k = 3$ and $x_k = x_3 = 1$. Add 4 to the operation counter.
7) Since \( k=a+l \) we compute \( E_2 \) using \( x_3=1 \) and 
\[ |\frac{m_2}{y_2+1}|+(1,1,1) \] in Equation (2.23). We have

\[
\begin{array}{c|c|c|c}
\text{Moduli} & 17 & 13 & 11 \\
\hline
x_3: & 1 & 1 & 1 \\
\hline
|\frac{m_2}{y_2+1}|: & x & 1 & 1 \\
\hline
E_2: & 1 & 1 & 1 \\
\end{array}
\]

where the residue code for \( |\frac{m_2}{y_2+1}| \) was found from the quotient table. Add 1 to the operation counter to account for the multiplication operation.

9) The new numerator \( X_2 \) is computed as

\[
\begin{array}{c|c|c|c}
\text{Moduli} & 17 & 13 & 11 \\
\hline
E_2: & 1 & 1 & 1 \\
\hline
Y: & x & 4 & 10 & 8 \\
\hline
E_2Y: & 4 & 10 & 8 \\
\hline
X_1: & 15 & 3 & 9 \\
\hline
X_2: & 11 & 6 & 1 \text{ where } X_2=X_1-E_2Y. \\
\end{array}
\]

Add 2 to the operation counter. Also, the estimate is added to \( Q \) to get \( Q+(16,3,5)+(1,1,1)=(0,4,6) \). Add 1 to the operation counter.
4) Do a mixed radix conversion on $x_2$ to get $x_2 \leftrightarrow <0,4,1>$. Therefore, $k=2$ and $x_2=4$. Add 4 to the operation counter.

6) Since $k=2$, we get the residue code for $E_3 = \left\lfloor \frac{x_2}{y_2+1} \right\rfloor \leftrightarrow (0,0,0)$ from the quotient table.

10) Since $E_3=0$, processing will stop after $E'$ is determined. Since $x_2 \leftrightarrow <0,4,1> < y \leftrightarrow <0,12,8>$, we have that $E'=0$. So stop with answer $0 \leftrightarrow (0,4,6) \leftrightarrow 17$, as expected.

### B.4 Formal Statement of the OSRA

In each of the three algorithms presented in this dissertation the discussion contains a formal statement of the algorithm. To present the standard form, the same will be done here.

The One Sided Rounding Algorithm finds the truncated quotient $\lfloor X/Y \rfloor$ for any numerator $X$ and denominator $Y \neq 0$ in the ID $[0,M-1]$. The moduli $m_n, m_{n-1}, \ldots, m_1$ are assumed to be any positive pairwise relatively prime integers, ordered such that $m_n > m_{n-1} > \ldots > m_1$.

The method uses the iteration

$$x_{i+1} = x_i - E_{i+1}y$$  \hspace{1cm} (2.25)

with $x_0 = X$, and stopping conditions $x_r = 0$ or $E_r+1 = 0$. The estimate $E_{i+1}$ is computed as
\[ E_{i+1} = \begin{cases} 
0, & \text{if } k < \ell \\
\frac{x_k}{y_{\ell}+1}, & \text{if } k = \ell \\
x_k \left( - \frac{m_{\ell}}{y_{\ell}+1} \right), & \text{if } k = \ell + 1 \\
x_k \left( - \frac{m_{\ell}}{y_{\ell}+1} \right)^{m_{\ell}+1} \cdots m_{k-1}, & \text{if } k > \ell + 1 .
\end{cases} \]

Residue encoded values of \( \frac{x_k}{y_{\ell}+1} \) or \( \frac{m_{\ell}}{y_{\ell}+1} \), for all possible values of \( x_k \) or \( m_{\ell} \) and \( y_{\ell} \), are stored in a table indexed by \( x_k \) or \( m_{\ell} \) and \( y_{\ell} \). Residue encoded values of all possible products of the form \( m_{\ell+1} \cdots m_{k-1} \) are stored in another table indexed by \( k \) and \( \ell \). If either of the stopping conditions is reached on the \( r \)-th iteration, that is, \( X_r = 0 \) or \( E_{r+1} = 0 \), then

\[ \left\lfloor \frac{X}{Y} \right\rfloor = \sum_{i=0}^{r-1} E_{i+1} + E', \quad (2.27) \]

where

\[ E' = \begin{cases} 
0, & \text{if } X_r = 0 \\
0, & \text{if } 0 < X_r < Y \\
1, & \text{if } X_r > Y .
\end{cases} \]

46
B.5 Proof of the Validity of the OSRA

The proof of the validity of the OSRA will now be given, and it will consist of four steps. The first, Lemma 1, deals with convergence. It gives the range of values that nonzero quotient estimates must have in order that successive reduced numerators form a decreasing sequence of positive integers. The second step, Lemma 2, shows that the estimate defined in Equations (2.23) and (2.24) is either zero or lies in the range specified by Lemma 1. The third step, Lemma 3, shows that if the stopping condition \( E_{r+1} = 0 \) is satisfied, then \( X_r < 2Y \). This lemma is needed when deriving the value of \( E' \) in Theorem 1. The fourth step, Theorem 1, shows that the quotient formulas in Equations (2.27) and (2.28) are correct, and that the OSRA eventually halts. The proofs of Lemmas 2 and 3 are given in Appendix A because they are tedious.

The sequence of steps in the above proof will be used throughout this dissertation. There will be two, or possibly three, lemmas, followed by a theorem. The first lemma deals with convergence of the subtractive procedure. It states the range of estimate values for which successive reduced numerators, or their absolute values, decrease. In the second lemma, the estimate defined for the new algorithm is stated either to lie in this range or equal zero, depending on the stopping condition. For a new algorithm whose estimate could be zero, a third lemma states that such a value is produced by a final reduced numerator which is less than twice the denominator. This step is always required if an estimate could be zero, because it guarantees that the truncated quotient can be found (by magnitude comparison) in spite of the errors.
introduced by the approximations used in finding estimate. The final step is always a theorem which states that the new algorithm eventually halts and correctly computes the truncated quotient.

**Lemma 1:** For the iteration \( X_{i+1} = X_i - E_{i+1} Y \), we have \( 0 < E_{i+1} \leq Y \) if \( k < \ell \)

\[
0 < X_{i+1} - X_i = X_i - E_{i+1} Y < \frac{X_i}{Y} \cdot Y = 0.
\]

Proof: We have

\[
X_{i+1} = X_i - E_{i+1} Y > X_i - \left( \frac{X_i}{Y} \right) Y = 0.
\]

Also,

\[
X_{i+1} < X_i - (0) Y = X_i.
\]

Therefore,

\[
0 < X_{i+1} < X_i.
\]

Q.E.D

**Lemma 2:** For the estimate given in Equations (2.23) and (2.24), viz.,

\[
E_{i+1} = \begin{cases} 
0, & \text{if } k < \ell \\
\frac{x_k}{y_{\ell+1}}, & \text{if } k = \ell \\
x_k \frac{m_{\ell}}{y_{\ell+1}}, & \text{if } k = \ell + 1 \\
x_k \frac{m_{\ell}}{y_{\ell+1}} \ldots m_k, & \text{if } k > \ell + 1
\end{cases}
\]

(2.29)

when \( Y \) is not a mixed radix coefficient, or
\[ E_{i+1} = \begin{cases} 
0, & k < \ell \\
x_k, & k = \ell \\
x_k m_{\ell}, & k = \ell + 1 \\
x_k m_{\ell+1} \cdots m_{k-1}, & k > \ell + 1 
\end{cases} \quad (2.30) \]

when \( Y \) is a mixed radix coefficient, we have \( 0 < E_{i+1} < X_i/Y \).

**Proof:** See Appendix A.

**Lemma 3:** For the estimate \( E_{i+1} \) given in Equations (2.29) or (2.30), we have \( E_{i+1} = 0 \Rightarrow X_i < 2Y \) for all \( X_i, Y \).

**Proof:** See Appendix A.

**Theorem 1:** The OSRA eventually halts, at which time

\[ \left| \frac{X}{Y} \right| = \sum_{i=0}^{r-1} E_{i+1} + E', \quad (2.31) \]

where

\[ E' = \begin{cases} 
0, & \text{if } X_r = 0 \\
0, & \text{if } 0 < X_r < Y \\
1, & \text{if } X_r > Y 
\end{cases} \quad (2.32) \]

**Proof:** We will first show that the OSRA halts.

Let \( \{E_{i+1}\} \) and \( \{X_i\} \) denote the sequence of estimates and reduced numerators, respectively. By Lemma 2, we have that \( 0 < E_{i+1} < X_i/Y \) for all \( i \). If \( E_{i+1} = 0 \), then the OSRA halts. If \( E_{i+1} > 0 \), then \( X_{i+1} < X_i \) by Lemma 1. Therefore, \( \{X_i\} \) is a decreasing sequence of nonnegative integers, and, because of Lemma 2, \( \{E_{i+1}\} \) is also. Therefore, eventually for some \( r \), either \( E_{r+1} = 0 \) or \( X_r = 0 \), and the OSRA halts.
Now to derive the quotient expressions, we have

\[ X_1 = X_0 - E_1 Y, \]
\[ X_2 = X_1 - E_2 Y, \]
\[ \vdots \]
\[ X_{r-1} = X_{r-2} - E_{r-1} Y, \]
\[ X_r = X_{r-1} - E_r Y \]

and either \( X_r = 0 \) or \( E_{r+1} = 0 \). Combining the above equations, we have

\[ X_r = X_{r-1} - E_r Y, \]
\[ \quad = X_{r-2} - E_{r-1} Y - E_r Y, \]
\[ \quad = X_{r-3} - E_{r-2} Y - E_{r-1} Y - E_r Y, \]
\[ \vdots \]
\[ \vdots \]
\[ = X_0 - E_1 Y - E_2 Y - \ldots - E_{r-1} Y - E_r Y. \]

Therefore

\[ X = X_0 = \sum_{i=0}^{r-1} E_{i+1} Y + X_r, \]

and so

\[ \left| \frac{X}{Y} \right| = \sum_{i=0}^{r-1} E_{i+1} Y + \left| \frac{X_r}{Y} \right|. \]

If \( X_r = 0 \), then \( \left| \frac{X_r}{Y} \right| = 0. \)

If \( E_{r+1} = 0 \), then, by Lemma 3, \( X_r < 2Y \). Therefore, \( \left| \frac{X_r}{Y} \right| \) is equal to 0 if \( X_r < Y \), or equal to 1 if \( X_r > Y \). Denoting by \( E' \) the quantity \( \left| \frac{X_r}{Y} \right| \), we have, finally,
\[ \frac{x}{y} = \sum_{i=0}^{r-1} E_{i+1} + E', \]

where

\[ E' = \begin{cases} 
0, & \text{if } X_r = 0 \\
0, & \text{if } 0 < X_r < Y \\
1, & \text{if } X_r > Y 
\end{cases} \]

QED

B.6 Summary

In this section we have presented the most attractive RNS division algorithm so far published, called the One Sided Rounding Algorithm (OSRA). The OSRA is iterative and employs a "rounded up" approximation of the divisor in conjunction with a stored table of quotients to compute quotient estimates. The section began with a statement of notation and an introduction to the OSRA. Calculation of the estimate was discussed in detail. The algorithm was then summarized, and an example of its use was presented. The OSRA was then formally stated and shown to be correct.
CHAPTER III
TWO NEW RESIDUE NUMBER SYSTEM DIVISION ALGORITHMS

In this chapter, the first two of the three new residue number system division algorithms, respectively called the Two Sided Rounding Algorithm and the Signed Algorithm, are presented. They are described and statistics of their performance are given. Also a study is made of some of the statistics of the OSRA.

The chapter begins with an overview of the Two Sided Rounding Algorithm (TSRA) in Section A. After a statement of notation, the method is introduced. The algorithm is then summarized, and an example given. The algorithm is then formally stated, and shown to be correct. Subsequently, in Section B, the Signed Algorithm (SA) is presented, described and shown to be correct. Next, in Section C, is a comparison of the storage requirements, mean running time and standard deviation about the mean of the TSRA with those of the OSRA. In Section D, the same comparisons are made between the SA and the OSRA. The subject changes somewhat in Section E where a partial study is made of some of the statistical aspects of the OSRA. In particular, expressions for the mean running time and probability density function of the running time are derived. These expressions are compared with computer generated data.
A. The Two Sided Rounding Algorithm

A.1 Introduction

The TSRA is an iterative division algorithm that employs "two sided" rounding on the numerator and denominator of a fraction to find quotient estimates. After a statement of notation the method is motivated. Subsequently, the method is summarized and an example problem is solved. Then, the TSRA is formally stated, and is shown to be correct.

The notation used in Chapters I and II is used in Section A of this chapter. In particular, we use an overbar to denote absolute value, and the mixed radix digits of the absolute value of the numerator are denoted by

\[ \bar{x} = <x_n, x_{n-1}, \ldots, x_1> \quad (3.1) \]

Furthermore, since some of the most significant mixed radix digits can be zero, it is convenient to use \( k \) and \( \ell \) to denote the most significant nonzero mixed radix digits of the absolute value of the numerator and denominator, respectively. For such a case,

\[ \bar{x} = <0, \ldots, 0, x_k, x_{k-1}, \ldots, x_1> \quad (3.2) \]

and

\[ y = <0, \ldots, 0, y_\ell, y_{\ell-1}, \ldots, y_1> \quad (3.3) \]
Also, since the TSRA is iterative, we let $X_i$ denote the $i$th reduced numerator, $\bar{X}_i$ denote its absolute value, $E_{i+1}$ denote an estimate of the quantity $\bar{X}_i/Y$, and $\text{Sgn}_{i+1}$ denote the following function:

$$\text{Sgn}_{i+1} = \begin{cases} 
1, & \text{if } X_i > 0 \\
-1, & \text{if } X_i < 0.
\end{cases} \quad (3.4)$$

With these definitions we can now consider the algorithm itself.

**A.2 Discussion of the TSRA**

The TSRA is subtractive, and uses the iteration

$$X_{i+1} = X_i - \text{Sgn}_{i+1} E_{i+1} Y \quad (3.5)$$

with $X_0 = X$ to obtain successive reduced numerators. For purposes of explanation, Equation (3.5) will be rewritten by expressing (from Equation (3.4)) $X_i$ as $\text{Sgn}_{i+1} \bar{X}_i$, and factoring out $\text{Sgn}_{i+1}$ in Equation (3.5) to give

$$X_{i+1} = \text{Sgn}_{i+1} (\bar{X}_i - E_{i+1} Y) \quad (3.6)$$

The TSRA stops in the $r$th iteration when $\bar{X}_r < Y$. When this stopping condition is met, the truncated quotient is the sum of signed quotient estimates plus a small error term $E'$, viz.,

$$\left| \frac{X}{Y} \right| = \sum_{i=0}^{r-1} \text{Sgn}_{i+1} E_{i+1} + E' \quad (3.7)$$

54
where
\[ E' = \begin{cases} 
0, & \text{if } \text{Sgn}_{r+1} = 1 \\
-1, & \text{if } \text{Sgn}_{r+1} = -1 .
\end{cases} \] (3.8)

The TSRA finds the truncated quotient for nonnegative numerator and denominator. However, its quotient estimates are such that \( X_{i+1} \) can become negative in Equation (3.5). Consequently, the stopping condition requires the use of the absolute value, and the expression for the truncated quotient incorporates the Sgn\( _{i+1} \) function.

The \((i+1)\)st quotient estimate \( E_{i+1} \) in Equation (3.5) is an estimate of the quantity \( \bar{X}_i / Y \) (see Equation (3.6)), and is found in terms of mixed radix information about \( \bar{X}_i \) and \( Y \). Clearly, the number of residue operations required to find the truncated quotient depends on how closely the estimate approximates \( \bar{X}_i / Y \). The method of finding \( E_{i+1} \) is therefore the "heart" of the TSRA, and will now be discussed.

The estimate \( E_{i+1} \) is found using the mixed radix digits of rounded approximations of \( \bar{X}_i \) and \( Y \). It is defined by
\[
E_{i+1} = \begin{cases} 
\bar{x}_k \left\lfloor \frac{m_k}{\bar{y}_k} \right\rfloor, & \text{if } k = 2 \\
-\bar{x}_k \left\lfloor \frac{m_k}{\bar{y}_k} \right\rfloor, & \text{if } k = 2 + 1 \\
\bar{x}_k \left\lfloor \frac{m_k}{\bar{y}_k} \right\rfloor \left( m_{k+1} \ldots m_k \right), & \text{if } k > 2 + 1
\end{cases} .
\] (3.9)

This expression will now be derived.
To decrease stored table size (to be discussed later in this section), the mixed radix expression for $Y$ is rounded by discarding the less significant mixed radix digits of the denominator. The most significant nonzero digit is rounded up or down, depending on the size of the next less significant digit $y_{\ell-1}$. If $y_{\ell-1}$ is less than or equal to $\lfloor m_{\ell-1}/2 \rfloor$, then $y_{\ell}$ is rounded down (i.e., left as is). Otherwise, it is increased by 1. This produces a rounded approximation $\tilde{Y}$ of $Y$, and we have

$$Y = \tilde{Y} = \tilde{y}_{\ell} m_{\ell-1} \ldots m_1,$$

where

$$\tilde{y}_{\ell} = \begin{cases} y_{\ell} & \text{if } y_{\ell-1} < \left\lfloor \frac{m_{\ell-1}}{2} \right\rfloor \text{ or } \ell = 1 \\ y_{\ell} + 1 & \text{if } y_{\ell-1} > \left\lfloor \frac{m_{\ell-1}}{2} \right\rfloor \text{ and } \ell > 1. \end{cases}$$

This rounding operation expresses $\tilde{Y}$ as a function of only the two variables $\tilde{y}_{\ell}$ and $\ell$, and these variables are used in calculating $E_{i+1}$ in Equation (3.9).

Likewise, the TSRA rounds the absolute value of the $i$th reduced numerator in much the same way. That is, all but the most significant nonzero digit of $\tilde{X}_i$ are discarded, and the most significant is increased by 1 if $y_{\ell}$ were rounded up. Otherwise, it is left as is. However, there is one exception to this rounding scheme. If $x_{k-1} = m_n - 1$, then it is left as is. This is to prevent the storage used by the TSRA from exceeding that used by the OSRA. Formally, this rounding scheme for $\tilde{X}_i$ produces a rounded $\tilde{X}_i$ called $\tilde{X}_i$ given by
\[ \bar{x}_i = \tilde{x}_i = \tilde{x}_{k^m_{k-1\cdots m1}}, \quad (3.12) \]

where

\[ \tilde{x}_k = \begin{cases} 
  x_k, & \text{if } \tilde{y}_k = y_k \\
  x_k + 1, & \text{if } \tilde{y}_k = y_k + 1 \\
  m_{n-1}, & \text{if } x_k = m_{n-1}
\end{cases} \quad (3.13) \]

This rounding approximates \( \bar{x}_i \) by a quantity that depends only on the two variables \( k \) and \( \tilde{x}_k \), and expresses \( \tilde{x}_k \) for use in calculating \( E_{i+1} \) in Equation (3.9).

Using the rounded approximations \( \tilde{x}_i \) and \( \tilde{y} \), we approximate \( \bar{x}_i / \tilde{y} \) as

\[ \frac{\tilde{x}_i}{\tilde{y}} = \frac{\tilde{x}_{k^m_{k-1\cdots m1}}}{\tilde{y}_{k^m_{k-1\cdots m1}}}, \quad (3.14) \]

where the \( k<\ell \) case does not occur because then \( \bar{x}_i < \tilde{y} \), in which case the algorithm would have stopped with \( r=i \).

The fractions in Equation (3.14) must be truncated because the RNS is an integer number system. Accordingly, the integer parts of Equation (3.14) are used to get

57
In order to minimize the number of residue operations required to
determine the estimate, stored tables are used when computing it. When
\( k = n \) in Equation (3.9), the estimate \( E_{i+1} \) is defined to be
\[
\begin{bmatrix}
\tilde{x}_k \\
\tilde{y}_\ell \\
\end{bmatrix}
\]
and a table containing residue encoded values of \( \tilde{x}_k/\tilde{y}_\ell \) for all
possible \( \tilde{x}_k \) and \( \tilde{y}_\ell \) can be used to find it. Furthermore, in order to
minimize storage for the cases \( k = n + 1 \) and \( k > n + 1 \), the expressions in
Equation (3.15) are approximated as
\[
\begin{bmatrix}
\tilde{x}_k m_\ell \\
\tilde{y}_\ell \\
\end{bmatrix} = \tilde{x}_k \frac{m_\ell}{\tilde{y}_\ell}
\]
(3.16)
when \( k = n + 1 \), and
\[
\begin{bmatrix}
\tilde{x}_k m_\ell \cdots m_{k-1} \\
\tilde{y}_\ell \\
\end{bmatrix} = \tilde{x}_k \frac{m_\ell}{\tilde{y}_\ell} m_{k+1} \cdots m_{k-1}
\]
(3.17)
when \( k > n + 1 \). In the latter case, another table storing all possible
residue encoded products \( m_{k+1} \cdots m_{k-1} \) is used. Expressions (3.16) and
(3.17) are the estimates \( E_{i+1} \) in Equation (3.9) for the cases \( k = n + 1 \) and
\( k > n + 1 \), respectively.
When \( y_{\alpha} = m_{\alpha} \), the estimate is set to 1 without a table access. This can be done because it is known a-priori that \( \tilde{X}_i > Y \) (i.e., the stopping condition is not satisfied), and in such a case \( |\tilde{X}/Y| \) cannot exceed 1.

### A.2.1 Required TSRA Capabilities

Special capabilities must be developed for the TSRA to handle the fact that the estimate \( E_{i+1} \) can exceed \( \tilde{X}_i / Y \) (see Equation (3.6)) for some combinations of \( \tilde{X}_i \) and \( Y \). This can cause \( X_{i+1} \) to be negative. Three special capabilities are needed. The first is the capability of representing negative operands, even though the ID is defined to be positive. The second is the capability of fast sign detection. The third is the capability of generating the mixed radix digits of an absolute value such as \( \tilde{X}_i \) from those of \( X_i \). These capabilities will now be discussed.

#### The Representation Capability

The negative operand representation capability is provided by an extra modulus \( m_{n+1} \). It must be relatively prime to the first \( n \) moduli so that negative operands can have unique representations. The simplest and most efficient choice for this modulus is \( m_{n+1} = 2 \). This value will be assumed throughout the remainder of the TSRA discussion and we stipulate that moduli \( m_n, m_{n-1}, \ldots, m_1 \) must be odd.
Calculation of $X_{i+1}$ is done in all $n+1$ moduli, and thus the modulo-2 digits of $E_{i+1}$, $Y$ and $X_i$ must be found. $E_{i+1}$ modulo 2 is found from the stored tables, as previously explained. $Y$ modulo 2 is found by an XOR of the least significant binary bits of the mixed radix digits of $Y$. That is, if
\[ Y = y_n m_{n-1} \cdots m_1 + y_{n-1} m_{n-2} \cdots m_1 + \cdots + y_2 m_1 + y_1, \] (3.18)
then since all moduli $m_1, m_2, \ldots, m_n$ are odd, we have
\[ Y \text{ modulo 2} = \left( \sum_{j=1}^{n} y_j \right) \text{ modulo 2} = \sum_{j=1}^{n} b_{0,j}, \] (3.19)
where $b_{0,j}$ is the least significant bit of the binary coded mixed radix digit $y_j$. This exclusive-OR operation is assumed to be done using zero residue operations by an $n$-input XOR gate. The value $X_i$ modulo 2 is calculated "as we go", once $X_0$ modulo 2 is found initially, using an XOR operation on the mixed radix digits of $X_i$. Note that the output of the XOR gate is the quantity $(X_i \text{ modulo } M) \text{ modulo } 2$.

The Sign Detection Capability

Sign detection on an operand $X_{i+1}$ is performed by using the extra modulo 2 residue digit as follows. The value $X_{i+1}$ modulo 2 (calculated using $E_{i+1}$ modulo 2, $Y$ modulo 2 and $X_i$ modulo 2 as described above) is compared with the output $(X_{i+1} \text{ modulo } M) \text{ modulo } 2$ of the XOR gate. If these quantities are equal, then $X_{i+1}$ is positive. Otherwise, it is negative. This is because (as will be shown in Lemmas 4 and 5) $-M < X_{i+1} < M$. Therefore, if $X_{i+1}$ is positive,
\[(X_{i+1} \text{ modulo } M) \text{ modulo } 2 = X_{i+1} \text{ modulo } 2. \quad (3.20)\]

However, if \(X_{i+1}\) is negative, then it is represented in the first \(n\) moduli by the positive quantity \(M + X_{i+1}\), so that

\[(X_{i+1} \text{ modulo } M) \text{ modulo } 2 = (\text{(M + X}_{i+1}) \text{ modulo } M) \text{ modulo } 2 \quad (3.21)\]

\[= (M + X_{i+1}) \text{ modulo } 2 \neq X_{i+1} \text{ modulo } 2\]

since \(M\) is odd. In summary, then, the sign of \(X_{i+1}\) is determined by comparing the output of the XOR gate with \(X_{i+1}\) modulo 2. If these quantities are equal, then \(X_{i+1}\) is positive, Otherwise, it is negative.

**The Absolute Value of Mixed Radix Digits Capability**

The mixed radix digits of \(\tilde{X}_i\) are calculated from those of \(X_i\) in one residue subtraction, as follows. Denoting the mixed radix digits of \(X_i\) as

\[X_i \leftrightarrow <x_n, x_{n-1}, \ldots, x_1> \quad (3.22)\]

where \(x_1\) is least significant, and using \(x_a\) to denote the least significant nonzero mixed radix digit of \(X_i\), then the mixed radix digits of \(\tilde{X}_i\) are

\[\tilde{X}_i \leftrightarrow <(m_n - 1) - x_n, (m_{n-1} - 1) - x_{n-1}, \ldots, m_a - x_a, 0, \ldots, 0> \quad (3.23)\]
These digits are computed by a single residue subtraction, since there are no borrows involved. In what follows, the residue quantity $(m_{n-1}, m_{n-1} - 1, \ldots, m_{a+1} - 1, 0, \ldots)$ will be called the "conversion word" for absolute value computation.

A.3 Summary of the TSRA

The TSRA is shown in flowchart form in Figure 2, and is summarized as follows. Q is defined to be the partial sum for the quotient.

**Step 0)** Set $Q = 0$ and the iteration counter $i = 0$.

**Step 1)** Do a mixed radix conversion on $Y$. The XOR gate generates $Y$ modulo 2.

**Step 2)** Do a mixed radix conversion on $X_j$. The XOR gate gives $(X_j \mod M) \mod 2$. If $i = 0$ (that is, if this is the first iteration), then do the following:

a) Set $X_0 \mod 2 = (X_0 \mod M) \mod 2$.

b) Set $Sgn_i = 1$.

c) Go to step 4.

**Step 3)** If $(X_i \mod M) \mod 2 = X_i \mod 2$ (that is, if $X_i$ is positive), then set $Sgn_{i+1} = 1$. Also, let the mixed radix digits of $\bar{X}_i$ be those of $X_i$, and go to Step 4. Otherwise, $X_i$ is negative, so find the mixed radix digits of $\bar{X}_i$, and set $Sgn_{i+1} = -1$.

62
Figure 2. Flowchart of the TSRA.
Step 4) If $X_j > Y$ (that is, if the stopping condition is not satisfied), then do the following:

a) Compute $E_{i+1}$ in all $n+1$ moduli.

b) Calculate new numerator $X_{i+1} = X_i + Sgn_{i+1}E_{i+1}Y$ in all $n+1$ moduli.

c) Add $Sgn_{i+1}E_{i+1}$ to $Q$. That is, add the signed quotient estimate to the "running quotient sum."

d) Increment $i$, the iteration counter.

e) Go to Step 2.

Otherwise, the stopping condition is satisfied, so do the following:

a) If $Sgn_{i+1}=-1$, then subtract 1 from $Q$ and stop.

Otherwise, stop.

A.4 Example of Use of the TSRA

For this example, we choose the modulus set $\{m_3, m_2, m_1\} = \{17, 13, 11\}$, and want to find $\lfloor 2425/27 \rfloor$. As before, angle brackets \langle a_3, a_2, a_1 \rangle denote mixed radix digits with $a_1$ as the least significant digit, and parentheses $(r_3, r_2, r_1)$ denote the residue digits. The number of each solution step corresponds to the number of the same step in the summary. The number of residue operations required to find the truncated quotient is indicated by the operation counter. At each step, it is incremented by the number of residue operations required.
Solution: 1) Mixed radix conversion on $Y$ gives $Y \leftrightarrow <0,2,5>$ and $\varepsilon = 2$. XOR gate gives $Y \mod 2 = 1$. Add 4 to operation counter, because the mixed radix conversion requires $2(n-1)$ operations, and $n = 3$.

2) Mixed radix conversion on $X_0$ gives $X_0 \leftrightarrow <16,12,5>$ and $k = 3$. Add 4 to the operation counter. XOR gate gives $(X_0 \mod M) \mod 2 = 1$. Since $i = 0$ (that is, this is the first iteration), we set $X_0 \mod 2 = 1$ and $\text{Sgn}_1 = 1$ in Equation (3.5).

4) Comparing the mixed radix digits of $X_0$ and $Y$, we find that $X_0 > Y$. So we compute $E_1$. We have that $k = 3$ and $\varepsilon = 2$. Since $5 < \lfloor \frac{11}{2} \rfloor = \lfloor \frac{m_{k-1}}{2} \rfloor$ in Equation (3.11) (where $m_{k-1} = m_1 = 11$), we have $y_2 = 2$ and $x_3 = 16$. This is the case $k = \varepsilon + 1$, and so we compute $E_1 = 16 \lfloor \frac{13}{2} \rfloor = 96$. The operation counter is incremented by 1. In residue notation, we have
where \( X_1 = X_0 - E_1Y \) because \( \text{Sgn}_1 = 1 \). Add 2 to operation counter. Note that \( X_1 \text{mod} 2 = 1 \).

2) A mixed radix conversion on \( X_1 \) gives \( X_1 \leftrightarrow <15, 10, 9> \). XOR gate gives \( (X_1 \text{mod} M) \text{mod} 2 = 0 \). Add 4 to operation counter.

3) Since \( (X_1 \text{mod} M) \text{mod} 2 \neq X_1 \text{mod} 2 \), we have \( \text{Sgn}_2 = -1 \), and calculate the mixed radix digits of \( \bar{X}_1 \) by subtracting the mixed radix digits of \( X_1 \) from the conversion word, as shown for Equation (3.23), viz.,

\[
\begin{array}{c|c c c c}
\text{Moduli} & 17 & 13 & 11 & 2 \\
\hline
E_1: & 11 & 5 & 8 & 0 \\
Y: & 10 & 1 & 5 & 1 \\
\hline
E_1Y: & 8 & 5 & 7 & 0 \\
X_0: & 11 & 7 & 5 & 1 \\
\hline
X_1 = X_0 - E_1Y: & 3 & 2 & 9 & 1 \\
\end{array}
\]

\[
\begin{array}{c|c c c c}
\text{Moduli} & 17 & 13 & 11 \\
\hline
\text{Conversion word}: & 16 & 12 & 0 \\
X_1: & 15 & 10 & 9 \\
\hline
\bar{X}_1: & 1 & 2 & 2 \\
\end{array}
\]
Note that this "sign flip" is a single subtraction of the mixed radix digits of $X_j$, treated as a residue number, from the residue number ($16, 12, 0$) as defined for Equation (3.23). The rightmost digit of the latter is zero because $a=1$ and $m_1 \mod m_1 = 0$. Add 1 to the operation counter to account for the subtraction.

4) Since $X_1 > Y$, we compute $E_2$. We have $k=3$, $s=2$, $\bar{y}_2=2$ and $\bar{x}_3=1$. So, $E_2=1 | 13/2 | = 6$, from Equation (3.11). Calculation of this estimate uses one residue multiplication so add 1 to the operation counter. In residue, we have

<table>
<thead>
<tr>
<th>Moduli:</th>
<th>17</th>
<th>13</th>
<th>11</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>$E_2$:</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td>$Y$:</td>
<td>x</td>
<td>10</td>
<td>1</td>
<td>5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>10</th>
<th>1</th>
<th>5</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$E_2Y$:</td>
<td>9</td>
<td>6</td>
<td>8</td>
<td>0</td>
</tr>
<tr>
<td>$X_1$:</td>
<td>3</td>
<td>2</td>
<td>9</td>
<td>1</td>
</tr>
</tbody>
</table>

$X_2 = X_1 + E_2Y$: 12 8 6 1,

where $X_2 = X_1 + E_2Y$ (the plus sign arising because $Sgn_2 = 1$ in Equation (3.5)). Add 2 to operation counter.

2) A mixed radix conversion on $X_2$ gives $X_2 = <16, 12, 6>$. XOR gate gives $(X_2 \mod M) \mod 2 = 0$. Add 4 to operation counter.
3) Since \((X_2 \mod M) \mod 2 \neq X_2 \mod 2\), \(X_2\) is negative, so we have \(\text{Sgn}_3 = -1\) and calculate the mixed radix digits of \(\tilde{X}_2\) as shown in Equation (3.23), viz.,

<table>
<thead>
<tr>
<th>Moduli:</th>
<th>17</th>
<th>13</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conversion word:</td>
<td>16</td>
<td>12</td>
<td>0</td>
</tr>
<tr>
<td>(X_2):</td>
<td>-16</td>
<td>12</td>
<td>6</td>
</tr>
<tr>
<td>(\tilde{X}_2):</td>
<td>0</td>
<td>0</td>
<td>5</td>
</tr>
</tbody>
</table>

Add 1 to the operation counter because the sign flip requires the use of a single residue subtraction, as discussed for Equation (3.23).

4) Since \(\tilde{X}_2 < Y\), we know to stop. Also, since \(\text{Sgn}_3 = -1\), we have

\[
\left\lfloor \frac{X}{Y} \right\rfloor = \text{sgn}_1 E_1 + \text{sgn}_2 E_2 - 1 = 96 + (-1)6 - 1 = 89 \text{ , (3.24)}
\]

and the operation counter must be increased by 3 because 3 terms are combined in the above sum.

A.5 Formal Definition of the TSRA

The TSRA finds the truncated quotient for any numerator \(X\) and denominator \(Y \neq 0\) in the ID

\[
0 < X, Y < M-1 \text{ . (3.25)}
\]

The system moduli \(\{m_1, m_2, \ldots, m_n\}\) must be odd, pairwise relatively prime integers, and ordered such that \(m_n > m_{n-1} > \ldots > m_1\).
The method uses the iteration

\[ X_{i+1} = X_i - \text{Sgn}_{i+1} E_{i+1} Y, \tag{3.26} \]

with \( X_0 = X \), and stopping condition \( X_i < Y \). The estimates \( E_{i+1} \) are computed as

\[
E_{i+1} = \begin{cases} 
\frac{x_k}{y_\ell}, & \text{if } k = \ell \\
\frac{m_\ell}{y_\ell}, & \text{if } k = \ell + 1 \\
\frac{x_k}{y_\ell} m_{\ell+1} \cdots m_{k-1}, & \text{if } k > \ell + 1,
\end{cases}
\tag{3.27}
\]

where

\[
\hat{y}_\ell = \begin{cases} 
y_1, & \text{if } \ell = 1 \\
y_\ell, & \text{if } y_{\ell-1} < \frac{m_{\ell-1}}{2} \text{ and } \ell \neq 1 \\
y_{\ell+1}, & \text{if } y_{\ell-1} > \frac{m_{\ell-1}}{2} \end{cases}
\tag{3.28}
\]

and

\[
\hat{x}_k = \begin{cases} 
x_k, & \text{if } \hat{y}_\ell = y_\ell \\
x_{k+1}, & \text{if } \hat{y}_\ell = y_\ell + 1 \\
m_{\ell-1}, & \text{if } x_k = m_{\ell-1}
\end{cases} \text{ and } x_k < m_{\ell-1}
\tag{3.29}
\]

Residue encoded values of \( \left\lfloor \frac{x_k}{y_\ell} \right\rfloor \), for all possible values of \( \hat{x}_k \) and \( \hat{y}_\ell \), are stored in a table indexed by \( \hat{x}_k \) and \( \hat{y}_\ell \). Residue encoded values of all possible products of the form \( m_{\ell+1} \cdots m_{k-1} \) are stored in...
another table indexed by \( k \) and \( \lambda \).

If the stopping condition is reached on the \( r \)th iteration, that is, \( X_r \leq Y \), then

\[
\left| \frac{X}{Y} \right| = \sum_{i=0}^{r-1} \text{Sgn}_{i+1} E_{i+1} + E', \tag{3.30}
\]

where

\[
E' = \begin{cases} 
0, & \text{if Sgn}_{r+1} = 1 \\
-1, & \text{if Sgn}_{r+1} = -1.
\end{cases} \tag{3.31}
\]

A.6 Proof of the Validity of the TSRA

The TSRA will be shown valid in three steps. The first step, Lemma 4, gives the range of values that quotient estimates \( E_{i+1} \) must have in order that the stopping condition is reached eventually. Note that the proof involves the absolute value of \( X_i \), required because \( X_i \) can be of either sign, and therefore convergence of absolute value is necessary. The second step, Lemma 5, states that the estimates \( E_{i+1} \) defined for the TSRA lie in the range specified by Lemma 4. The third step, Theorem 2, is a statement that the TSRA computes the truncated quotient and stops. The proof of Lemma 5 is lengthy and is therefore given in Appendix B.

Lemma 4: For the iteration

\[
x_{i+1} = x_i - \text{Sgn}_{i+1} E_{i+1}Y \tag{3.32}
\]
we have $0 < E_{i+1} < \frac{2X_i}{Y} \Rightarrow X_{i+1} < X_i$.

**Proof:** We have

$$X_{i+1} = X_i - \text{Sgn}_{i+1} E_{i+1} Y > X_i - \text{Sgn}_{i+1} \left( \frac{2X_i}{Y} \right) Y = -X_i.$$  

Also,

$$X_{i+1} < X_i - \text{Sgn}_{i+1}(0) Y = X_i.$$  

Therefore,

$$-X_i < X_{i+1} < X_i \iff X_{i+1} < X_i.$$  

QED

**Lemma 5:** For the estimate used by the TSRA, viz.,

$$E_{i+1} = \begin{cases} \tilde{x}_k \frac{m_k}{\tilde{y}_k} & \text{if } k = 2 \\ \tilde{x}_k \frac{m_k}{\tilde{y}_k} & \text{if } k = 2+1 \\ \tilde{x}_k \frac{m_k}{\tilde{y}_k} m_{x+1} \cdots m_{k-1} & \text{if } k > 2+1, \end{cases}$$  

and $E_{i+1} = 1$ when $\tilde{y}_k = m_n$, we have $X_i > Y \Rightarrow 0 < E_{i+1} < \frac{2X_i}{Y}$.

**Proof:** See Appendix B.

**Theorem 2:** The TSRA eventually halts, at which time

$$\left| \frac{X}{Y} \right| = \sum_{i=0}^{r-1} \text{Sgn}_{i+1} E_{i+1} + E',$$  

(3.34)
where

\[ E' = \begin{cases} 
0, & \text{if } \text{Sgn}_{r+1} = 1 \\
-1, & \text{if } \text{Sgn}_{r+1} = -1 .
\end{cases} \]  

(3.35)

Proof: From Lemmas 4 and 5, we have

\[ X_{i} > Y \Rightarrow X_{i+1} < X_{i} , \]

so that the sequence \( \{X_{i}\} \) is a decreasing sequence of positive integers. Therefore, for some \( r \), \( X_{r} < Y \) and the algorithm will stop.

To derive the quotient expression, we have

\[ X_{1} = X_{0} - \text{Sgn}_{1} E_{1} Y , \]
\[ X_{2} = X_{1} - \text{Sgn}_{2} E_{2} Y , \]
\[ \vdots \]
\[ X_{r-1} = X_{r-2} - \text{Sgn}_{r-1} E_{r-1} Y , \]
\[ X_{r} = X_{r-1} - \text{Sgn}_{r} E_{r} Y \]

and \( X_{r} < Y \). Combining these equations, we have
\[ X_r = X_{r-1} - \text{Sgn}_r E_r Y , \]
\[ = X_{r-2} - \text{Sgn}_{r-1} E_{r-1} Y - \text{Sgn}_r E_r Y , \]
\[ = X_{r-3} - \text{Sgn}_{r-2} E_{r-2} Y - \text{Sgn}_{r-1} E_{r-1} Y - \text{Sgn}_r E_r Y , \]
\[ \vdots \]
\[ = X_0 - \text{Sgn}_1 E_1 Y - \text{Sgn}_2 E_2 Y - \ldots - \text{Sgn}_r E_r Y . \]

Therefore,
\[ X_0 = Y \sum_{i=0}^{r-1} \text{Sgn}_{i+1} E_{i+1} + X_r . \]

So
\[ \left| \frac{X_0}{Y} \right| = \sum_{i=0}^{r-1} \text{Sgn}_{i+1} E_{i+1} + \left| \frac{X_r}{Y} \right|. \]

But \( X_r < Y \). So, letting \( E' = \left| \frac{X_r}{Y} \right| \), we have
\[ \left| \frac{X}{Y} \right| = \left| \frac{X_0}{Y} \right| = \sum_{i=0}^{r-1} \text{Sgn}_{i+1} E_{i+1} + E' , \]

where
\[ E' = \begin{cases} 
0, & \text{if } \text{Sgn}_{r+1} = 1 \\
-1, & \text{if } \text{Sgn}_{r+1} = -1 .
\end{cases} \]

QED
A.7 Summary of Section A

In this section we have presented a new RNS division algorithm called the Two Sided Rounding Algorithm (TSRA). The TSRA is iterative and uses "two sided" rounding on approximations of the numerator and denominator to find the quotient estimates. The section began with a statement of notation and an introduction to the TSRA. Three capabilities needed for negative operand management were then discussed. The algorithm was then summarized, an example given, and then it was formally stated and shown to be correct.
B. The Signed RNS Division Algorithm

B.1 Introduction

This section is a presentation of the Signed Algorithm (SA), which is a modification of the TSRA for signed division. The SA employs one of the sign management techniques developed for the TSRA to find the truncated quotient for numerators and denominators of either sign. After a statement of notation, an overview of the method is presented. Next, a discussion of how the "heart" of the algorithm (the quotient estimating rules) is derived is given, followed by a summary of the algorithm and an example. The section is concluded with a formal statement and proof that the algorithm halts and computes the truncated quotient.

The notation used in Section B is the same as that used in Section A, with the following exceptions. We define \( sgn_{i+1} = Sgn(X_i)Sgn(Y) \) to include sign information about the numerator as well as the denominator. Also, we denote the mixed radix digits of the absolute value of the denominator as

\[
\bar{Y} = \langle y_n, y_{n-1}, \ldots, y_1 \rangle \quad (3.36)
\]

Furthermore, we let \( y_k \) denote the most significant nonzero mixed radix digit of \( \bar{Y} \). With these definitions we can now discuss the algorithm itself.

B.2 Discussion of the Signed Algorithm

The Signed Algorithm is subtractive, and finds the truncated quotient in the standard ID for signed arithmetic. That is, it finds
the truncated quotient \(|X/Y|\) for numerator \(X\) and denominator \(Y\) of either sign. Like the TSRA, it does this by reducing the absolute value of the numerator by successive subtractions of multiples of the denominator. Symbolically, it performs the iteration

\[ X_{i+1} = X_i - \text{sgn}_{i+1} E_{i+1} Y \]  

(3.37)

with \(X_0=X\), to obtain successive reduced numerators. For purposes of later explanation, Equation (3.37) will be rewritten as

\[ X_{i+1} = (X_i - E_{i+1} \bar{Y})\text{sgn}(X_i) \]  

(3.38)

This was done by substituting \(\text{sgn}(X_{\bar{i}})\bar{X}_{\bar{i}}\) for \(X_{\bar{i}}\), and \(\bar{Y}\) for \(\text{sgn}(Y)Y\) in Equation (3.37), and factoring out the \(\text{sgn}(X_{\bar{i}})\) term.

The SA stops in the \(r^{th}\) iteration when \(\bar{X}_r < \bar{Y}\). At such a time, the truncated quotient is the sum of signed quotient estimates \(\text{sgn}_{i+1} E_{i+1}\) plus a small error term \(E'\), viz.,

\[ |X/Y| = \sum_{i=0}^{r-1} \text{sgn}_{i+1} E_{i+1} + E' \]  

(3.39)

where

\[ E' = \begin{cases} 
0, & \text{if } X_r = 0 \\
0, & \text{if } X_r \neq 0 \text{ and } \text{sgn}_{r+1} = 1 \\
-1, & \text{if } X_r \neq 0 \text{ and } \text{sgn}_{r+1} = -1 
\end{cases} \]  

(3.40)
The \((i+1)\)st quotient estimate \(E_{i+1}\) in Equation (3.37) is an estimate of the quantity \(\tilde{x}_i / \tilde{y}\) (see Equation (3.38)), and is found in terms of mixed radix information about \(\tilde{x}_i\) and \(\tilde{y}\). Clearly, the number of residue operations required to find the truncated quotient depends on how closely the estimate approximates \(\tilde{x}_i / \tilde{y}\). The method of finding \(E_{i+1}\) is therefore the "heart" of the SA, and will now be discussed.

The estimate \(E_{i+1}\) used by the SA is almost the same as that used by the TSRA, and is defined to be

\[
E_{i+1} = \begin{cases} 
\frac{\tilde{x}_k}{\tilde{y}} & \text{, if } k = 2 \\
\frac{\tilde{x}_k}{\tilde{y}} + \frac{m_2}{\tilde{y}} & \text{, if } k = 2 + 1 \\
\frac{\tilde{x}_k}{\tilde{y}} + \frac{m_2}{\tilde{y}} + \cdots + \frac{m_{k+1}}{\tilde{y}} & \text{, if } k > 2 + 1
\end{cases}
\]

This estimate differs from the TSRA estimate in two respects. The first is that it is defined in terms of the rounded approximation \(\tilde{y}\) of \(y\) instead of \(y\). The second is that \(x_k\) is rounded up whenever \(y_k\) is.

This is different from the rounding used by the TSRA, where \(\tilde{x}_k\) was prevented from assuming a value equal to the maximum modulus. The SA estimate (viz., Equation (3.41)) is now derived.

The SA estimate, like that of the TSRA, uses rounded approximations of the mixed radix digits of \(\tilde{x}_i\) and \(\tilde{y}\). The mixed radix expression for \(\tilde{y}\) is rounded in exactly the same way as used for the TSRA. That is,

\[
\tilde{y} = \bar{y} = \tilde{y}_k \cdot m_{k-1} \cdots m_1,
\]

(3.42)
where
\[
\tilde{y}_l = \begin{cases} 
  y_1, & \text{if } \ell = 1 \\
  y_{i-1} + \lfloor \frac{m_{i-1}}{2} \rfloor, & \text{if } \ell > 1 \text{ and } y_{i-1} < \lfloor \frac{m_{i-1}}{2} \rfloor \\
  y_{i-1} + 1, & \text{if } \ell > 1 \text{ and } y_{i-1} \geq \lfloor \frac{m_{i-1}}{2} \rfloor
\end{cases}
\] (3.43)

As is the case for the TSRA, this rounding approximates \( \tilde{y} \) by a quantity that depends only on the two variables \( \tilde{y}_l \) and \( \ell \), and gives \( \tilde{y}_l \) for use in calculating \( E_{1+l} \) in Equation (3.41).

Likewise, the SA rounds the mixed radix form for \( \tilde{x}_l \) to the approximate numerator defined by
\[
\tilde{x}_l = \tilde{x}_i = \tilde{x}_k m_{k-1} \cdots m_1,
\] (3.44)

where
\[
\tilde{x}_k = \begin{cases} 
  x_k, & \text{if } \tilde{y}_l = y_l \\
  x_k' + 1, & \text{if } \tilde{y}_l = y_l + 1
\end{cases}
\] (3.45)

This rounding operation expresses an approximation of \( \tilde{x}_l \) (i.e., \( \tilde{x}_i \)) as a function of only the two variables \( \tilde{x}_k \) and \( k \), and gives \( \tilde{x}_k \) for use in calculating \( E_{1+l} \) in Equation (3.41).

Using these approximations in exactly the same way as for the TSRA, we obtain the definition of the quotient estimate given in Equation (3.41) above.

The estimate is computed by two stored tables, exactly as was done for the TSRA. One stored table contains residue encoded values of
\[ \left| \frac{x_k}{y_2} \right| \] for all possible \( x_k \) and \( y_2 \). The other contains residue encoded values of all possible products \( m_{k+1} \ldots m_{k-1} \).

The first two of the three capabilities needed by the TSRA (viz., negative operand representation and sign detection capabilities) are not needed by the SA. This is because the SA assumes an ID which includes negative numbers. Consequently, representation of negative operands is a "non-issue", and an extra modulus is not required. Furthermore, sign detection can be done by comparing the mixed radix digits of an operand with those of the maximum positive integer in the ID, as explained in Chapter II.

The third capability, that of computing absolute values, is needed by the SA, and is done in exactly the same way as used in the TSRA. As explained for that case, one residue subtraction is used to convert the mixed radix digits of a negative operand to the mixed radix digits of its absolute value.

B.3 Summary of the Signed Algorithm

The SA is shown in flowchart form in Figure 3 and is summarized as follows. \( Q \) is defined to be the partial sum for the quotient.

Step 1) Set \( Q=0 \) and iteration counter \( i=0 \).

Step 2) Do a mixed radix conversion of \( Y \). If \( Y<0 \), then find the mixed radix digits of \( Y \), and set \( \text{sgn}(Y)=-1 \). Otherwise, set \( \text{sgn}(Y)=1 \).
Figure 3. Flowchart of the SA.
Step 3) Do a mixed radix conversion of \( X_i \). If \( X_i < 0 \), find the mixed radix digits of \( X_i \), and set \( \text{sgn}_{i+1} = -\text{sgn}(Y) \). Otherwise, set \( \text{sgn}_{i+1} = \text{sgn}(Y) \).

Step 4) If \( X_i > Y \), then do the following:
   a) Compute the quotient estimate \( E_{i+1} \).
   b) Compute the new numerator \( X_{i+1} = X_i - \text{sgn}_{i+1} E_{i+1} Y \).
   c) Update \( Q \) by adding \( \text{sgn}_{i+1} E_{i+1} \) to it.
   d) Increment \( i \), the iteration counter, and go to step 3.

Otherwise, the algorithm will stop, so do the following:
 a) If \( X_i = 0 \) or \( \text{sgn}_{i+1} = 1 \), then \( E' \) is zero, so stop.
   Otherwise, \( E' \) is -1, so subtract 1 from \( Q \) and stop.

B.4 Example of Use of the Signed Algorithm

In this example, we choose the modulus set \( \{m_3, m_2, m_1\} = \{17, 13, 11\} \), for which the ID (from Eq. (1.4) in Chapter I) is \([-1215, 1215]\). We want to find \( |X/Y| = |\ -1211/49\ | \). As before, mixed radix digits will be enclosed in angle brackets with the least significant digit on the right. Residue digits will be enclosed in parentheses. The solution step number corresponds to the number of the same step in the summary. The number of residue operations used so far in the calculation is stored in the operation counter, which is updated with the number of operations required at each step.
Solution:

Note: The maximum positive integer in the ID is 1215, and its mixed radix digits are <8,6,5>. These digits are used when sign determinations are performed, as explained in Chapter II.

2) Do a mixed radix conversion on Y to get Y+=<0,4,5> and z=2. Y is positive because the maximum positive integer in the ID has mixed radix digits <8,6,5>, and Y+=<0,4,5> < <8,6,5>. So sgn(Y)=1. Add 4 to the operation counter, because the mixed radix conversion requires 2(n-1) operations, and n=3.

3) Do a mixed radix conversion on X₀ to get X₀+=<8,6,10>. Add 4 to the operation count. X₀ is negative because <8,6,10> > <8,6,5>. Therefore, sgn₁=-1. Find the mixed radix digits of X₀ by the same method as used for the TSRA. We have

<table>
<thead>
<tr>
<th>Moduli:</th>
<th>17</th>
<th>13</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conversion word:</td>
<td>16</td>
<td>12</td>
<td>0</td>
</tr>
<tr>
<td>X₀:</td>
<td>-</td>
<td>8</td>
<td>6</td>
</tr>
<tr>
<td>X₀:</td>
<td>8</td>
<td>6</td>
<td>1</td>
</tr>
</tbody>
</table>

So X₀+=<8,6,1>, and k=3.

4) Since X₀>Y, we compute E₁ using y₂=4 and x₃=8 in Equation (3.41). We have
where the residue code for \( \tilde{z}_{m_2/y_2} \) was found from the quotient table. Add 1 to the operation counter to account for the multiplication operation. \( X_1 \) is computed as

\[
\begin{align*}
\text{Moduli:} & \quad 17 \quad 13 \quad 11 \\
\tilde{x}_3: & \quad 8 \quad 8 \quad 8 \\
\tilde{z}_{m_2/y_2}: & \quad x \quad 3 \quad 3 \quad 3 \\
E_1: & \quad 7 \quad 11 \quad 2 \\
\end{align*}
\]

where \( X_1 = X_0 + E_1 Y \) because \( \text{sgn}_1 = -1 \). Add 2 to the operation counter.

3) Do a mixed radix conversion on \( X_1 \) to get \( X_1++<16,9,9> \). Add 4 to the operation counter. Since \( X_1++<16,9,9> > <8,6,5> \), \( X_1 \) is negative. So \( \text{sgn}_2 = -1 \). Find the mixed radix digits of \( \tilde{x}_1 \) as
Add 1 to the operation counter. So $\tilde{x}_1 \leftrightarrow <0,3,2>$.

4) Since $\tilde{x}_1 < Y < 0,4,5$ we compute the error term $E'$. Since $\text{sgn}_2 = -1$, we have $E' = -1$. So the quotient is $(0,0,0) - (2,11,7) - (1,1,1) = (8,1,9) \leftrightarrow -25$. Add 2 to the operation counter because there are two terms added to $(0,0,0)$ in the above sum.

B.5 Formal Definition of the Signed Algorithm

The Signed Algorithm finds the truncated quotient $|X/Y|$ for any numerator $X$ and denominator $Y \neq 0$ in the ID

$$- \left\lfloor \frac{M}{2} \right\rfloor < X, Y < \left\lceil \frac{M}{2} \right\rceil$$

(3.46)

for $M$ odd, or

$$- \frac{M}{2} < X, Y < \frac{M}{2} - 1$$

(3.47)

for $M$ even. The moduli $\{m_1, m_2, \ldots, m_n\}$ are assumed to be positive pairwise relatively prime, and ordered such that $m_n > m_{n-1} > \ldots > m_1$.

The method uses the iteration

$$x_{i+1} = x_i - \text{sgn}_{i+1} E_{i+1} Y$$

(3.48)
with $x_0 = x$, and stopping condition $\bar{x}_r < \bar{y}$. The estimates $E_{i+1}$ are computed as

$$
E_{i+1} = \begin{cases} 
\tilde{x}_k / \tilde{y}_\ell, & \text{if } k = \ell \\
\tilde{x}_k / \tilde{y}_\ell, & \text{if } k = \ell + 1 \\
\tilde{x}_k / \tilde{y}_\ell, & \text{if } k > \ell + 1,
\end{cases}
$$

(3.49)

where

$$
\tilde{y}_\ell = \begin{cases} 
y_1, & \text{if } \ell = 1 \\
y_\ell, & \text{if } \ell \neq 1 \text{ and } y_{\ell-1} < \frac{m_{\ell-1}}{2} \\
y_\ell + 1, & \text{if } \ell \neq 1 \text{ and } y_{\ell-1} > \frac{m_{\ell-1}}{2}
\end{cases}
$$

(3.50)

and

$$
\tilde{x}_k = \begin{cases} 
x_k, & \text{if } \tilde{y}_\ell = y_\ell \\
x_k + 1, & \text{if } \tilde{y}_\ell = y_\ell + 1
\end{cases}
$$

(3.51)

Residue encoded values of $|\tilde{x}_k/\tilde{y}_\ell|$, for all possible values of $\tilde{x}_k$ and $\tilde{y}_\ell$ are stored in a table indexed by $\tilde{x}_k$ and $\tilde{y}_\ell$. Residue encoded values of all possible products of the form $m_\ell + 1 \ldots m_{\ell-1}$ are stored in another table indexed by $k$ and $\ell$. If the stopping condition is reached on the $r$th iteration, that is, $\bar{x}_r < \bar{y}$, then

$$
\left| \frac{x}{y} \right| = \sum_{i=0}^{r-1} \text{sgn}_{i+1} E_{i+1} + E',
$$

(3.52)
where

\[ E' = \begin{cases} 
0, & \text{if } X_r = 0 \\
0, & \text{if } X_r \neq 0 \text{ and } \text{sgn}_{r+1} = 1 \\
-1, & \text{if } X_r \neq 0 \text{ and } \text{sgn}_{r+1} = -1.
\] (3.53)

B.6 Proof That the Signed Algorithm is Valid

A proof that the SA halts and computes the truncated quotient is now given. It proceeds in three steps. The first step, Lemma 6, states what range of values is permissible for \( E_{i+1} \) so that the stopping condition is reached eventually. The second step, Lemma 7, states that the defined \( E_{i+1} \) in Equation (3.49) satisfy Lemma 6. The third step, Theorem 3, is a statement that the Signed Algorithm computes the truncated quotient and stops. The proof of Lemma 7 is lengthy, and therefore is given in Appendix C.

**Lemma 6:** For the iteration

\[ X_{i+1} = X_i - sgn_{i+1} E_{i+1} Y, \] (3.54)

with \( sgn_{i+1} = sgn(X_i) sgn(Y) \), we have

\[ 0 < E_{i+1} < \frac{2X_i}{Y} \Rightarrow \frac{X_i}{Y} < X_{i+1} < X_i. \] (3.55)

**Proof:** We have that

\[ X_{i+1} = X_i - sgn_{i+1} E_{i+1} Y \]

\[ = (sgn(X_i)) X_{i+1} = \frac{X_i}{Y} - E_{i+1} Y, \]
as explained for Equation (3.38). Now,

\[ 0 < E_{i+1} < \frac{2x_i}{\gamma} \Rightarrow \bar{x}_i - \left(\frac{2x_i}{\gamma}\right)Y < (\text{sgn}(x_i))x_{i+1} < \bar{x}_i \]

\[ \Rightarrow -x_i < (\text{sgn}(x_i))x_{i+1} < x_i . \]

Therefore,

\[ X_i \text{ positive} \Rightarrow -x_i < x_{i+1} < x_i \]

and

\[ X_i \text{ negative} \Rightarrow x_i < -x_{i+1} < -x_i \]

\[ \Rightarrow x_i < x_{i+1} < -x_i . \]

Therefore, \( x_{i+1} < x_i . \)

QED

**Lemma 7:** For the \( E_{i+1} \) as defined in Equation (3.49), viz.,

\[
E_{i+1} = \begin{cases} 
\begin{vmatrix} \tilde{x}_k \\ \tilde{y}_2 \end{vmatrix} & , \text{if } k=\ell \\
\begin{vmatrix} m_2 \\ \tilde{y}_2 \end{vmatrix} & , \text{if } k=\ell+1 \\
\begin{vmatrix} m_2 \\ \tilde{y}_2 \end{vmatrix} m_{\ell+1} \cdots m_{k-1} & , \text{if } k>\ell+1 , 
\end{cases}
\]

(3.56)
where
\[
\tilde{y}_k = \begin{cases} 
  y_1, & \text{if } k=1 \\
  y_k, & \text{if } k\neq 1 \text{ and } y_{k-1} < \frac{m_k-1}{2} \\
  y_k+1, & \text{if } k\neq 1 \text{ and } y_{k-1} > \frac{m_k-1}{2}
\end{cases}
\quad (3.57)
\]
and
\[
\tilde{x}_k = \begin{cases} 
  x_k, & \text{if } \tilde{y}_k = y_k \\
  x_k+1, & \text{if } \tilde{y}_k = y_k+1
\end{cases}
\quad (3.58)
\]
we have \(\tilde{x}_i > \tilde{y} \Rightarrow 0 < E_{i+1} < \frac{2\tilde{x}_i}{\tilde{y}}\).

**Proof:** See Appendix C.

**Theorem 3:** The Signed Algorithm halts, and computes \(|X/Y|\) for every \(X\) and \(Y\neq 0\) in the ID.

**Proof:** By Lemmas 6 and 7, we know that the sequence \(\{\tilde{x}_i\}\) is a decreasing sequence of positive integers. Therefore, for some \(\tilde{x}_r\), we will have \(\tilde{x}_r < \tilde{y}\), and the SA stops.

Now to derive the quotient expression, we have
\[
X_1 = X_0 - \text{sgn}_1 E_1 Y, \\
X_2 = X_1 - \text{sgn}_2 E_2 Y, \\
\quad \vdots \\
X_{r-2} = X_{r-3} - \text{sgn}_{r-2} E_{r-2} Y, \\
X_{r-1} = X_{r-2} - \text{sgn}_{r-1} E_{r-1} Y, \\
X_r = X_{r-1} - \text{sgn}_r E_r Y
\]
and $X_r < \bar{Y}$. Combining these equations, we have

\[ X_r = X_{r-1} - \text{sgn}_r E \bar{Y}. \]

\[ = X_{r-2} - \text{sgn}_{r-1} E_{r-1} \bar{Y} - \text{sgn}_r E \bar{Y}, \]

\[ = X_{r-3} - \text{sgn}_{r-2} E_{r-2} \bar{Y} - \text{sgn}_{r-1} E_{r-1} \bar{Y} - \text{sgn}_r E \bar{Y}, \]

\[ \vdots \]

\[ = X_0 - \text{sgn}_1 E_1 \bar{Y} - \text{sgn}_2 E_2 \bar{Y} - \cdots - \text{sgn}_r E_r \bar{Y}. \]

Therefore,

\[ X_0 = \sum_{i=0}^{r-1} \text{sgn}_{i+1} E_{i+1} \bar{Y} + X_r \]

and so

\[ \left| \frac{X_0}{\bar{Y}} \right| = \left| \frac{X}{\bar{Y}} \right| = \sum_{i=0}^{r-1} \text{sgn}_{i+1} E_{i+1} + \left| \frac{X_r}{\bar{Y}} \right|. \]

But $X_r < \bar{Y}$, and letting $E'$ denote $\left| \frac{X_r}{\bar{Y}} \right|$, we have

\[ \left| \frac{X}{\bar{Y}} \right| = \sum_{i=0}^{r-1} \text{sgn}_{i+1} E_{i+1} + E'. \]

where

\[ E' = \begin{cases} 
0, & \text{if } X_r = 0 \\
0, & \text{if } X_r \neq 0 \text{ and } \text{sgn}_{r+1} = 1 \\
-1, & \text{if } X_r \neq 0 \text{ and } \text{sgn}_{r+1} = -1. 
\end{cases} \]

QED
B.7 Summary of Section B

In section B we have presented a modification of the TSRA called the Signed Algorithm (SA). The SA finds the truncated quotient \( \lfloor \frac{X}{Y} \rfloor \) for signed numerator \( X \) and denominator \( Y \) in the system ID, and uses essentially the same "two sided" rounding scheme on \( Y \) as used by the TSRA. The algorithm was discussed, summarized and an example of its use given. Subsequently, the SA was formally stated and shown to be correct.

C. A Comparison of the TSRA and OSRA

Section C is a comparison of the TSRA's and OSRA's storage requirements, mean running times, and standard deviations of running time about the mean. Storage requirements will be compared first. Subsequently, computationally derived values of mean running time and standard deviation about the mean will be presented. A theoretical derivation of a lower bound for the mean running time of the OSRA is given in Section E.

C.1 Storage Requirements of Both Algorithms

The size of the stored tables used by the TSRA and OSRA is the same, and is equal to

\[
\frac{(m_n-1)(m_n-2)}{2} + \frac{(n-1)(n-2)}{2}.
\]

(3.59)

This fact will now be shown.
The OSRA requires a "quotient" table which stores residue encoded values of $\lfloor \alpha/y^+ \rfloor$ and is indexed by $\alpha$ and $y^+$. The variable $\alpha$ can assume any value in $[1,m_n-1]$, and $y^+$ any value in $[1,m_n-1]$. Normally, such a table would contain $(m_n-1)^2$ values. However, the OSRA assumes that quotient values for $\alpha<y^++1$ are not stored, because they are zero when truncated. Therefore, the quotient table for the OSRA is of size $(m_n-1)(m_n-2)/2$.

The TSRA requires a quotient table which stores residue encoded values of $\lfloor \alpha/\tilde{y}_\xi \rfloor$ and is indexed by $\alpha$ and $\tilde{y}_\xi$. From Equations (3.28) and (3.29), $\alpha$ can assume any value in $[1,m_n-1]$, and $\tilde{y}_\xi$ any value in $[1,m_n]$. However, a table access is not done when $\tilde{y}_\xi=m_n$ as explained in Section A. Furthermore, the table need not store an entry for the case $\alpha<\tilde{y}_\xi$ because the algorithm stops when this occurs. Also, the TSRA does not store a quotient value for $\tilde{y}_\xi=1$, since in this case $\lfloor \alpha/\tilde{y}_\xi \rfloor=\alpha$. Therefore, the quotient table for the TSRA must store entries only for values of $\alpha$ and $\tilde{y}_\xi$ such that $\alpha<\tilde{y}_\xi$ and $\alpha$ is in $[1,m_n-1]$ and $\tilde{y}_\xi$ is in $[2,m_n-1]$. Clearly, the size of this table is $(m_n-1)(m_n-2)/2$. Therefore, the quotient table used by the TSRA is of the same size as that used by the OSRA.

Both algorithms require a table of residue encoded products $m_{\xi+1}\cdots m_{k-1}$ for $k>\xi+2$. Clearly, this table is of size $(n-2)(n-1)/2$.

Adding the storage required by both tables for each algorithm, we find that both the TSRA and the OSRA require storage equal to that given by Equation (3.59).
C.2 Computer Evaluation of the Mean and Standard Deviation of Running Time for the TSRA and OSRA

The following paragraphs are a presentation of computationally derived statistics of running time of the TSRA and OSRA. The statistics presented are the mean, standard deviation, and probability density. A simulating program is discussed first, and then the choice of modulus sets is discussed. The subsection is concluded with a discussion of the data produced by simulation.

A computer program which simulates both algorithms was written in the VAX-Fortran language for the VAX computer system and is listed in Appendix D. The program generated a sample set of division problems randomly selected from the ID \([0,M-1]\), and determined the sample mean running time and the standard deviation about the mean for each algorithm. The running time is defined to be the number of elementary residue operations (viz., additions, subtractions and multiplications) required to find the truncated quotient. Table accesses and compares were not counted. This is the method prevalent in the literature for measuring running time of residue arithmetic algorithms ([14]) (see, for example, [9,14,17 and 29].

The flowchart for the simulating program is given in Figures 4 thru 6. Figure 4 is a high level flowchart which gives a global view of information flow for both the simulating and statistical parts of the program. Note blocks 1 and 2. These are the TSRA and OSRA simulators, respectively, and are shown in Figures 5 and 6. Figure 5 gives details of the TSRA simulator, and is essentially Figure 2 with counters added. Figure 6 is essentially Figure 1 with counters added.
Figure 4. General flowchart for TSRA and OSRA simulating program.
Figure 5. Detailed flowchart of TSRA simulator.
Figure 6. Detailed flowchart of OSRA simulator.
In Figure 4, we see the input of modulus set and number of samples is followed by random choice of numerator $X$ and denominator $Y$. For each $X,Y$ pair, the TSRA is simulated first, and the truncated quotient is checked. Then, the OSRA is simulated, and its quotient checked. Running sums for the statistics, shown at the bottom, are incremented and the process is repeated. When enough samples have been simulated, the final statistical quantities are computed as shown on the right side.

In Figure 5, we see the detailed flowchart for the TSRA simulator. This figure is essentially the same as Figure 2 except that places where a residue counter is incremented are added and highlighted, because increments to the operation count are basic to the statistical evaluation of the TSRA. Also, the details of the calculation of $\tilde{x}_k, \tilde{y}_k$ and $E_{i+1}$ have been included.

In Figure 6, we see the detailed flowchart for the OSRA simulator. This figure is essentially the same as Figure 1 except that places where a residue counter is incremented are added and highlighted, for the same reason stated above. Details of estimate computation have also been included.

There are several features to notice in the flowcharts. First, in Figure 4, numerator $X$ and denominator $Y$ were chosen at random over the ID using the VAX random number generator command RAND(ISEED). This routine generates a random real number $\alpha$ such that $0<\alpha<1$ when ISEED is initialized to a large, odd integer (chosen to be 65069 for all data). The quantity $n_j$, the $j^{th}$ residue digit of either $X$ or $Y$, was chosen independently using
\[ n_j = |_{-m_j} \alpha_j | \quad j=1,2,...,n \]  

where \( m_j \) is the \( j^{th} \) modulus and \( \alpha_j \) is the \( j^{th} \) random number generated by \textsc{rand}. This method produces the residue code for an \( X \) or \( Y \) which is more nearly uniformly distributed than would be the case for letting \( X \) modulo \( m_j = |_{-a_j M_j} | \) modulo \( m_j \) and is suggested by Knuth [42]. Second, the value of truncated quotient calculated by each algorithm is checked with the correct value for each sample problem. Third, the standard deviation was calculated as the square root of the sample variance. The sample variance was calculated using the standard unbiased estimator, which is found as the quotient of the sum of squared deviations from the mean and the number of sample problems decreased by 1. The formula for standard deviation is given on Figure 4, at the right. Fourth, in Figure 5, an extra residue addition is required to round up on each of \( x_k \) and \( y_k \). However, if \( y_k \) is rounded up, the required addition is counted only once, because \( \tilde{y}_k \) remains the same for a particular division problem. Fifth, the quotient estimate \( E_{i+1} \) requires 0, 1 or 2 residue multiplications to find in the cases \( k=z \), \( k=z+1 \) and \( k>z+1 \) respectively. This is because of the TSRA stored table arrangement. For \( k=z \), the quantity \( E_{i+1} \) is found from the quotient table without using any residue operations. For \( k=z+1 \), the quantity \( |_{m_k/\tilde{y}_k} | \) is found from the quotient table, and then multiplied by \( \tilde{x}_k \). For \( k>z+1 \), the quantity \( |_{m_k/\tilde{y}_k} | \) is found from the quotient table, and the quantity \( m_{k+1}...m_{k-1} \) is found from the product table. The product of these quantities is multiplied by \( \tilde{x}_k \). Sixth, in Figure 6, the estimate \( E_{i+1} \) is found in the same number of residue operations as was needed by the
TSRA, for the same reasons stated above. The variable \( a \) is used to allow calculation (rather than storage) of the quantities that would be stored in the OSRA quotient table. The addition operation used to round it up does not count as a residue operation because the OSRA quotient table contains quotients with denominator \( y^k+1 \), by assumption. Note that \( a \) equals 1 only for the cases when \( Y \) is a mixed radix coefficient. As explained in Chapter II, these divisors are the special ones for which the quotient table is not accessed.

The simulating program was checked on the modulus set \( \{11, 13, 17\} \) by means of the example problems given in Table 1. The calculated number of residue operations required to solve each problem is also given.

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>OPERATIONS</th>
<th>X</th>
<th>Y</th>
<th>OPERATIONS</th>
</tr>
</thead>
<tbody>
<tr>
<td>2425</td>
<td>226</td>
<td>24</td>
<td>2425</td>
<td>226</td>
<td>22</td>
</tr>
<tr>
<td>2430</td>
<td>144</td>
<td>15</td>
<td>2425</td>
<td>27</td>
<td>47</td>
</tr>
<tr>
<td>2425</td>
<td>29</td>
<td>34</td>
<td>2425</td>
<td>7</td>
<td>76</td>
</tr>
<tr>
<td>2425</td>
<td>27</td>
<td>27</td>
<td>2425</td>
<td>144</td>
<td>37</td>
</tr>
<tr>
<td>2365</td>
<td>10</td>
<td>43</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

For the OSRA, the first three test problems were chosen to test each of the 3 possible estimate cases \( k=\ell, k=\ell+1 \) and \( k>\ell+1 \). The latter test problem ensures that the extra residue addition required for the case \( E'=1 \) is counted in the operation count. For the TSRA, the test problems were the case \( k=3 \) with \( Y \) chosen to exercise the following aspects of the simulator:
1) \( \varepsilon = 3 \) and \( Y \) is rounded both up and down;
2) \( \varepsilon = 2 \) and \( Y \) is rounded both up and down;
3) \( \varepsilon = 1 \) and \( Y \) is not rounded.

The next to last TSRA test problem also tests if the extra residue subtraction used for a nonzero \( E' \) is counted by the simulator.

The TSRA and OSRA were simulated on the following three modulus sets:

- \( M_1 = \{3, 5, 7, 11, 13, 17, 19, 23, 29, 31\} \) (3.61)
- \( M_2 = \{23, 29, 31, 37, 41, 43, 47, 53, 59, 61\} \) (3.62)
- \( M_3 = \{31, 37, 41, 43, 47, 53, 55, 59, 61, 63\} \) (3.63)

These sets were chosen with the following three considerations in mind. First, they are the same size because the algorithms' average running times are dependent on the number of moduli. Making the sets of equal size eliminates any effect the value of \( n \) may have on the average running time.

The second consideration is that all moduli were chosen so that their residue digits could be encoded in at most 6 bits binary. With this choice, the quotient tables required by the algorithms could be implemented using commercially available 4K PROM chips.

The third consideration is that the modulus sets \( M_1 \) and \( M_3 \) contain, respectively, the smallest and largest odd moduli, subject to the two conditions previously stated. Set \( M_2 \) is the set of largest prime moduli less than 64. The former sets were chosen to observe the effect of modulus size on the mean and standard deviation of running time.
Computer simulations provided a sample set from which the average running time and the standard deviation about that average were obtained. Probability density information about the running time was also generated. For each of the three modulus sets, 40000 sample division problems were simulated. Plots of the 1) sample average running time and 2) sample standard deviation as a function of sample size were computed for both the TSRA and OSRA. Also found were plots of the probability density functions of running time for each algorithm. The 12 plots are in Figures 7 through 18. A theoretical lower bound for the mean running time of the OSRA will be given in Section E.

Graphs of sample average running time as a function of sample size contain two plots, one for the TSRA and one for the OSRA. As seen in the graphs, the number of sample problems required for convergence is large and depends on the modulus set. For the set M1, the average running time "settles down" at a value between 5000 and 10000 samples. For M2, settling appears between 10000 and 20000 samples, and for M3 all 40000 seem to be required. The standard deviation converges at about the same number of samples as the running time. Furthermore, the average running time and standard deviation plots of the TSRA and OSRA seem to track each other. This is because of similarities in the algorithms. Note that there are fluctuations in average running time.
Figure 7. TSRA and OSRA average running time as a function of sample size.
Figure 8. TSRA and OSRA sample standard deviation as a function of sample size.
Figure 9. TSRA experimental probability density of running time.
EXPERIMENTAL PROBABILITY DENSITY
OF RUNNING TIME

MODULI (3, 5, 7, 11, 13, 17, 19, 23, 29, 31)

OSRA

WORST CASE PROBLEM AT 350

Figure 10. OSRA experimental probability density of running time.
Figure 11. TSRA and OSRA average running time as a function of sample size.
Figure 12. TSRA and OSRA sample standard deviation as a function of sample size.
Figure 13. TSRA experimental probability density of running time.
Figure 14. OSRA experimental probability density of running time.
Figure 15. TSRA and OSRA average running time as a function of sample size.
Figure 16. TSRA and OSRA sample standard deviation as a function of sample size.
Figure 17. TSRA experimental probability density of running time.
Figure 18. OSRA experimental probability density of running time.
These are due to runs of difficult problems, and arise because of peculiarities in the random number generator.

The running time probability density function values decrease quickly for large numbers of operations, as expected. Note that there are clusters of required operations. These clusters correspond to different numbers of estimates. Note that for the TSRA, roughly 50% of all problems are solved in zero estimates, as would be expected. These are the cases where the numerator is less than the denominator.

The average running times and standard deviations for each algorithm were taken from the corresponding graph. They were taken to be the values to which the corresponding graphs converge. These values were determined to the nearest tenth of an operation by visual means. That is, the value to which the plot appeared to converge was measured to the nearest tenth of an operation. These values are listed in Table 2.

**TABLE 2**

<table>
<thead>
<tr>
<th>MODULUS SETS</th>
<th>RUNNING TIME AND % DIFFERENCE</th>
<th>STANDARD DEVIATION AND % DIFFERENCE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>TSRA</td>
<td>OSRA</td>
</tr>
<tr>
<td>{ 3, 5, 7,11,13,17,19,23,29,31}</td>
<td>49.1</td>
<td>50.35</td>
</tr>
<tr>
<td>{23,29,31,37,41,43,47,53,59,61}</td>
<td>48.4</td>
<td>49.2</td>
</tr>
<tr>
<td>{31,37,41,43,47,53,55,59,61,63}</td>
<td>48.5</td>
<td>49.2</td>
</tr>
<tr>
<td>AVERAGE:</td>
<td>48.7</td>
<td>49.6</td>
</tr>
</tbody>
</table>
From Table 2, we see that the average running time and standard deviation are largely independent of the modulus set, although the average running time is slightly better for larger moduli. One can also see that the TSRA uses .9 (2%) fewer residue operations per problem than the OSRA, for the modulus sets used. Furthermore, the standard deviation of the TSRA is 3.2 operations (17%) less than that of the OSRA, for the modulus sets used. We conclude that the TSRA has slightly better mean running time and better standard deviation while, as indicated earlier, using the same amount of storage as the OSRA.

D. A Comparison of the SA and OSRA

The following paragraphs contain a comparison of the Signed and One Sided Rounding Algorithms' storage requirements and running time statistics. Storage requirements will be presented first, followed by computationally derived values for mean running times and standard deviations about the mean.

D.1 Storage Requirements of Both Algorithms

The size of the stored tables used by the Signed Algorithm is at most equal to that used by the One Sided Rounding Algorithm. As shown in Appendix E, the Signed Algorithm uses

$$\text{MAX}\left( \frac{\left| \frac{m_n}{2} \right| + 1}{2} - 1, \frac{m_{n-1}(m_{n-1}-1)}{2} \right) + \frac{(n-2)(n-1)}{2} \quad (3.64)$$

114
total storage, and for those modulus sets for which the moduli are large and approximately equal (i.e., "practical" modulus sets), the SA storage is at most equal to the amount used by the OSRA. As an example, for the modulus set \{23, 29, 31, 37, 41, 43, 47, 53, 59, 61\}, we have \(m_{10}=61\) and \(m_{g}=59\). Therefore, for this modulus set, the SA uses 1747 storage locations, while the OSRA uses 1806. This is a 3\% reduction of 59. However, for the modulus set \{37, 41, 43, 47, 53, 55, 59, 61, 63, 64\}, we have \(m_{10}=64\) and \(m_{g}=63\). Therefore, for this modulus set, the SA and OSRA use the same amount of storage.

D.2 Computer Evaluation of the Mean and Standard Deviation of Running Time for the SA and OSRA

The following paragraphs contain a discussion of computationally derived average running times and standard deviations for the Signed and One Sided Rounding Algorithms. A simulating program flowchart is presented first, and then the choice of modulus sets is discussed. Subsequently, plots of mean running time, standard deviation about the mean and computationally derived probability density functions are given. The section is concluded with a table of summarized performance data and conclusions.

A computer program which simulates both algorithms was written in the VAX-Fortran language for the VAX computer system, and is given in Appendix F. The program generated a sample set of division problems randomly selected from the ID \([0, M-1]\) for the OSRA, or, for the SA, from the ID \([-\lfloor M/2 \rfloor, \lfloor M/2 \rfloor]\) if \(M\) is odd, or \([-M/2, M/2 -1]\) if \(M\) is even. The program determined the sample mean running time and the standard
deviation about the mean for each algorithm. The running time is
defined to be the number of elementary residue operations (viz.,
additions, subtractions and multiplications) required to find the
truncated quotient. Table accesses and compares are not counted. This
is the method prevalent in the literature for measuring the running time
of residue arithmetic algorithms (see, for example, [9, 14, 17 and 29].

The flowchart for the simulating program is given in Figures 19
thru 21. Figure 19 is a high level flowchart which gives a global view
of the information flow for both the simulating and statistical parts of
the program. Blocks 1 and 2, in the center and lower center
respectively, are the SA and OSRA simulators, respectively. The
remainder of the figure gives details of error checking, data input and
output and statistical calculations. The formulas used for calculation
of the statistics are given in the upper right of Figure 19. Figure 20
gives details of the SA simulator. This figure is essentially Figure 3
with counters added, because the number of residue operations required
for division is basic for statistical evaluation of RNS division
algorithms. Figure 21 gives details of the OSRA simulator, and is the
same as Figure 6.

Figure 19 is the same as Figure 4, except that an SA simulator is
substituted for the TSRA simulator.

In Figure 20 we see the detailed flowchart for the SA simulator.
This figure is essentially the same as Figure 3 except that places where
a residue counter is incremented are added and highlighted. Also, the
details of the calculation of $\tilde{x}_k$, $\tilde{y}_2$ and $E_{i+1}$ have been included.
Figure 19. Global flowchart for SA and OSRA simulating program.
Figure 20. Detailed flowchart for SA simulator.
Figure 20. Continued.
Figure 21. Detailed flowchart for OSRA simulator.
There are several features to notice in the flowcharts. First, in Figure 19, numerator X and denominator Y are chosen at random in each modulus, as explained in Section C.2. Second, the value of truncated quotient calculated by each algorithm is checked with the correct value for each sample problem. Third, the standard deviation is calculated as the square root of the sample variance. The sample variance from which the standard deviation is obtained is found as the quotient of the sum of squared deviations from the mean and the number of sample problems decreased by 1. The formula for standard deviation is given on Figure 19, at the right.

In Figure 20, the flowchart variable "sgnwhy" corresponds to the algorithm variable sgn(Y), as defined in Section B. An extra residue addition is required to round up on \( x_k \) and \( y_\tilde{\omega} \). However, when \( y_\tilde{\omega} \) is rounded up, the addition is counted only once, because \( y_\tilde{\omega} \) does not change for a particular division problem. The quotient estimate requires 0, 1 or 2 residue multiplications to find in the respective cases \( k=\ell \), \( k=\ell+1 \) and \( k>\ell+1 \). This is for the same reasons as discussed in Section C.2 for the TSRA.

Figure 21 is the same as Figure 6. The latter figure is discussed in Section C.2.

The simulating program was checked on the modulus set \{11,13,17\} by means of the test problems in Table 3. The test problems used to check the OSRA simulating routine are the same as those used in Section C.2. The number of residue operations required to solve each test problem is also given.
Table 3

Test Problems and Number of Operations Required for SA and OSRA Simulator

<table>
<thead>
<tr>
<th></th>
<th>OPERATIONS</th>
<th></th>
<th>OPERATIONS</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>Y</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>-1214</td>
<td>1215</td>
<td>10</td>
<td>2425</td>
</tr>
<tr>
<td>1214</td>
<td>176</td>
<td>25</td>
<td>2425</td>
</tr>
<tr>
<td>-1214</td>
<td>-176</td>
<td>25</td>
<td>2425</td>
</tr>
<tr>
<td>-1214</td>
<td>176</td>
<td>23</td>
<td>2425</td>
</tr>
<tr>
<td>1214</td>
<td>-176</td>
<td>25</td>
<td></td>
</tr>
<tr>
<td>1214</td>
<td>73</td>
<td>44</td>
<td></td>
</tr>
<tr>
<td>-1214</td>
<td>73</td>
<td>50</td>
<td></td>
</tr>
<tr>
<td>1214</td>
<td>7</td>
<td>66</td>
<td></td>
</tr>
<tr>
<td>1214</td>
<td>-7</td>
<td>68</td>
<td></td>
</tr>
</tbody>
</table>

For the OSRA, the first three test problems were chosen to test each of the three possible estimate cases \( k=2 \), \( k=2+1 \) and \( k>2+1 \). The latter test problem tests if the extra residue addition required to find \( \lfloor X/Y \rfloor \) when \( E'=1 \) is counted in the operation count. For the SA, the first test problem exercises the case \( X<Y \) when the truncated quotient is \(-1\). Subsequent problems test the cases:

1) \( k=2=3 \) and \( X \) and \( Y \) are of either sign;
2) \( k=3 \), \( \ell=2 \) and the truncated quotient is of either sign;
3) \( k=3 \), \( \ell=1 \) and the truncated quotient is of either sign.

Note the the first SA test problem checks to see if the extra residue subtraction required when \( E' \) is nonzero is included in the operation count.

The SA and OSRA were simulated using the five modulus sets.
\[ M_1 = \{3, 5, 7, 11, 13, 17, 19, 23, 29, 31\} \] 
\[ M_2 = \{23, 29, 31, 37, 41, 43, 47, 53, 59, 61\} \] 
\[ M_3 = \{31, 37, 41, 43, 47, 53, 55, 59, 61, 63\} \] 
\[ M_4 = \{37, 41, 43, 47, 53, 55, 59, 61, 63, 64\} \] 
\[ M_5 = \{2, 3, 5, 7, 11, 13, 17, 19, 23, 29\} \]

These sets were chosen with three considerations in mind. First, they are the same size because the mean running times of both algorithms are dependent on the number of moduli. Making the sets of equal size eliminates any effect the value of \( n \) may have on the mean running time.

The second consideration is that all moduli were chosen so that their residue digits could be encoded in at most 6 binary bits. With this choice, the quotient tables required by the algorithms could be implemented using commercially available 4K PROM chips.

The third consideration is that of modulus sizes. Modulus set \( M_1 \) is the set of smallest odd relatively prime moduli. Set \( M_3 \) is the set of largest odd relatively prime moduli. Set \( M_2 \) is the set of largest odd prime moduli. Set \( M_4 \) is the set of largest relatively prime moduli. Set \( M_5 \) is the set of smallest relatively prime moduli. Sets \( M_4 \) and \( M_5 \) were chosen to provide information about the effect of modulus size on the mean and standard deviation of running time. Sets \( M_1, M_2 \) and \( M_3 \) are the same as those used for simulation of the TSRA, and are used to allow comparison of the SA with the TSRA.
Computer simulations provided a sample set from which average running time and standard deviation about that average were obtained. Probability density information about running time was also generated. For each of the five modulus sets, 40000 sample division problems were simulated. Plots of the 1) sample average running time and 2) sample standard deviation as a function of sample size were computed for both the SA and OSRA. Also found were plots of the probability density functions of running time for each algorithm. The 20 plots are in Figures 22 through 41.

Graphs of sample average running time as a function of sample size contain two plots, one for the SA and one for the OSRA. As seen in the graphs, the number of sample problems required for convergence is large and depends on the modulus set. For the set M1, the average running time "settles down" at a value of approximately 10000 samples. For M2, settling appears at around 15000 samples. For M3, M4 and M5, settling occurs around 30000, 15000 and 30000 samples, respectively. The standard deviation plots settle down at the same values except for M4, where the standard deviation seems to require all 40000 samples. Furthermore, the average running time and standard deviation plots of the SA and OSRA seem to track each other. This is because of similarities in the algorithms. Note that there are fluctuations in average running time. These are due to runs of difficult problems, and arise because of peculiarities in the random number generator.
Figure 22. SA and OSRA average running time as a function of sample size.
Figure 23. SA and OSRA sample standard deviation as a function of sample size.
Figure 24. SA experimental probability density of running time.
EXPERIMENTAL PROBABILITY DENSITY OF RUNNING TIME
MODULI (3, 5, 7, 11, 13, 17, 19, 23, 29, 31)
OSRA
WORST CASE PROBLEM AT 350

Figure 25. OSRA experimental probability density of running time.
Figure 26. SA and OSRA average running time as a function of sample size.
Figure 27. SA and OSRA sample standard deviation as a function of sample size.
Figure 28. SA experimental probability density of running time.
EXPERIMENTAL PROBABILITY DENSITY OF RUNNING TIME

MODULI (23, 29, 31, 37, 41, 43, 47, 53, 59, 61)
OSRA

WORST CASE PROBLEM AT 412

Figure 29. OSRA experimental probability density of running time.
Figure 30. SA and OSRA average running time as a function of sample size.
Figure 31. SA and OSRA sample standard deviation as a function of sample size.
EXPERIMENTAL PROBABILITY DENSITY OF RUNNING TIME

MODULI (31, 37, 41, 43, 47, 53, 59, 61, 63)

SA

WORST CASE PROBLEM AT 260

Figure 32. SA experimental probability density of running time.
Figure 33. OSRA experimental probability density of running time.
AVERAGE RUNNING TIME AS A FUNCTION OF SAMPLE SIZE

MODULI {37, 41, 43, 47, 53, 59, 81, 83, 84}

Figure 34. SA and OSRA average running time as a function of sample size.
Sample standard deviation as a function of sample size

Moduli (37, 41, 43, 47, 53, 55, 59, 61, 63, 64)

Figure 35. SA and OSRA sample standard deviation as a function of sample size.
Figure 36. SA experimental probability density of running time.
Figure 37. OSRA experimental probability density of running time.
Figure 38. SA and OSRA average running time as a function of sample size.
Figure 39. SA and OSRA sample standard deviation as a function of sample size.
PROBABILITY DENSITY
OF RUNNING TIME
MODULI (2, 3, 5, 7, 11, 13, 17, 19, 23, 29)
SA
WORST CASE PROBLEM AT 415

Figure 40. SA experimental probability density of running time.
EXPERIMENTAL PROBABILITY DENSITY OF RUNNING TIME

MODULI (2, 3, 5, 7, 11, 13, 17, 19, 23, 29)

OSRA

WORST CASE PROBLEM AT 282

Figure 41. OSRA experimental probability density of running time.
The run time probability density function values decrease quickly for large numbers of operations, as expected. Note that there are clusters of required operations. These clusters correspond to different numbers of estimates. For the SA, roughly 50% of all problems are solved in zero estimates, as would be expected. These are the cases where the absolute value of the numerator is less than that of the denominator.

The average running times and standard deviations for each algorithm were taken from each graph, and are listed in Table 4. They were taken to be the values to which the corresponding graphs converge. These values were determined to the nearest tenth of an operation by visual means. That is, the value to which the plot appeared to converge was measured to the nearest tenth of an operation.

### TABLE 4

**Mean Running Time and Standard Deviation for the SA and OSRA**

<table>
<thead>
<tr>
<th>MODULUS SETS</th>
<th>EXECUTION TIME AND % DIFFERENCE</th>
<th>STANDARD DEVIATION AND % DIFFERENCE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SA</td>
<td>OSRA</td>
</tr>
<tr>
<td>{3, 5, 7, 11, 13, 17, 19, 23, 29, 31}</td>
<td>51.2</td>
<td>50.35</td>
</tr>
<tr>
<td>{37, 41, 43, 47, 53, 55, 59, 61, 63, 64}</td>
<td>50.75</td>
<td>49.4</td>
</tr>
<tr>
<td>{23, 29, 31, 37, 41, 43, 47, 53, 59, 61}</td>
<td>50.7</td>
<td>49.2</td>
</tr>
<tr>
<td>{2, 3, 5, 7, 11, 13, 17, 19, 23, 29}</td>
<td>51.4</td>
<td>50.4</td>
</tr>
<tr>
<td>{31, 37, 41, 43, 47, 53, 55, 59, 61, 63}</td>
<td>50.7</td>
<td>49.2</td>
</tr>
<tr>
<td><strong>AVERAGE</strong></td>
<td><strong>51.0</strong></td>
<td><strong>49.7</strong></td>
</tr>
</tbody>
</table>
From Table 4, we see that as with the TSRA, the average running time and standard deviation are largely independent of the modulus set, although the average running time is slightly less for larger moduli. One can also see that the SA uses 1.3 (3%) more residue operations per problem than the OSRA, for the modulus sets used. However, the standard deviation of the SA is 2.3 operations (12%) less than that of the OSRA, for the modulus sets used. We conclude that the SA has slightly greater mean running time and better standard deviation while, as indicated earlier, using at most the same amount of storage as the OSRA.

Comparing the SA statistics with the corresponding values for the TSRA given in Section C.2, we see that for the modulus sets $M_1$, $M_2$ and $M_3$ the SA uses 2.2 (5%) more residue operations per problem, on the average, than the TSRA. Furthermore, the SA has a standard deviation which is 1 residue operation (6%) greater than that of the TSRA.

The fact that the average running time of the SA is greater than that of the TSRA and OSRA is due in large part to the fact that the absolute value of the mixed radix digits of both $X$ and $Y$ must be taken if they are negative. This requires two extra residue operations that were not needed by either the TSRA or OSRA, because the latter algorithms assume positive numerator and denominator. In this sense, the SA solves a more difficult problem.

E. Theoretical Evaluation of a Lower Bound on the Average Running Time of the OSRA

To give some intuitive feeling for the statistical behavior of the OSRA, a derivation of an expression for a lower bound on the average
running time of the simplest algorithm, the OSRA, is given below. The lower bound is compared with the computationally derived average running times presented in Section C.2, and is shown to be within 3% of the computationally derived values. Additionally, expressions for estimates of the first three terms in the running time probability density function of the OSRA are also derived, and these estimates are compared with computationally derived data. The material in this section refers to the discussion of the OSRA in Chapter II.

The notation used in this section is the same as that used in Section A, with the following additions. The probability of event \( E \) is denoted by \( P(E) \). The probability of event \( E \), given that event \( F \) has occurred, i.e., the "conditional probability of \( E \) given \( F \)" is denoted by \( P(E|F) \). Finally, the event \( 'E \text{ and } F' \), called the "conjunction of \( E \) and \( F \)" is denoted by \( E \land F \). We use \( t \) to denote the number of residue operations required to perform a mixed radix conversion. As shown in Chapter II, \( t=2(n-1) \).

The running time lower bound to be derived is an approximate expected value. Its derivation is based on the number of estimates used to solve division problems. In the derivation, the set of all division problems is partitioned into classes according to how many estimates are required for solution. The number of residue operations required to solve the problems in a given class, \( T_i \), is multiplied by the probability density of that class, to give the contribution of that class to the expected value. The sum of all such products is the average value of running time.
\[ T_{OSRA} = T_1 P(E_1 = 0) + T_2 P(E_2 = 0 \land E_1 = 0) + T_3 P(E_3 = 0 \land E_2 = 0 \land E_1 = 0) + \ldots \] \hspace{1cm} (3.71)

The lower bound for average running time is found by truncating the above expression after the first two terms, and is expressed in terms of \( t \) and approximate probabilities that the OSRA stops after making one and two quotient estimates.

We begin by deriving expressions for the \( T_j \), the number of elementary residue operations the OSRA performs when making the various quotient estimates. In the case of only one estimate, the algorithm requires two mixed radix conversions (one on \( X_0 \) and the other on \( Y \)) to find. If this estimate is zero, then the OSRA stops after possibly executing one extra residue addition (for \( E' \)).

If there is more than one estimate, then the algorithm proceeds to:

1) add the estimate to the running quotient sum;
2) multiply the estimate by \( Y \) and subtract this product from \( X_0 \), to form \( X_1 \).

\( X_1 \) is then mixed radix converted, a new estimate is made and the process is repeated. Thus the algorithm uses \( t+3 \) additional operations for each subsequent estimate. Therefore, an approximate expression for the average running time of the OSRA is

\[ T_{OSRA} = 2t P(E_1 = 0) + (3t+3) P(E_2 = 0 \land E_1 = 0) \]
\[ + (4t+6) P(E_3 = 0 \land E_2 = 0 \land E_1 = 0) + \ldots \] \hspace{1cm} (3.72)

where we have ignored the one residue addition required for \( E' = 1 \).
because its contribution is small due to the small probability of occurrence of \( E^0 = 1 \).

For ease in calculation, the series is truncated by approximating the last term. The lower bound, \( \tilde{T}_{\text{OSRA}} \), on \( T_{\text{OSRA}} \) thus formed is clearly

\[
\tilde{T}_{\text{OSRA}} = 2tP(E_1 = 0) + (3t+3)P(E_2 = 0 \land E_1 \neq 0) + (4t+6)(1-P(E_1 = 0)) - P(E_2 = 0 \land E_1 \neq 0),
\]

(3.73)

where, for lower bound purposes, we have assumed that all truncated quotients are found in at most 3 estimates.

Expressions for \( P(E_1 = 0) \) and \( P(E_2 = 0 \land E_1 \neq 0) \) will now be found. The probability \( P(E_1 = 0) \) will be found in the process of finding \( P(E_2 = 0 \land E_1 \neq 0) \). The quantity \( P(E_2 = 0 \land E_1 \neq 0) \) is first approximated in terms of \( P(|X/Y| < j) \) (for \( j = 1, 2, \ldots, M-1 \)) and an approximation for the quantity \( P(E_1 = j) \), (for \( j = 1, 2, \ldots, \left\lfloor \frac{m-1}{2} \right\rfloor \)). The expressions for \( P(|X/Y| < j) \) and the approximation of \( P(E_1 = j) \) for all \( j \), including \( j = 0 \), are then stated, and Appendix G contains their derivations.

We have, using the definition of conditional probability,

\[
P(E_2 = 0 \land E_1 \neq 0) = \max \sum_{j=1} \frac{P(E_2 = 0 \land E_1 = j)}{P(E_1 = j)}
\]

\[
= \sum_{j=1} \frac{P(E_2 = 0 | E_1 = j)P(E_1 = j)}{P(E_1 = j)},
\]

(3.74)

where 'max' is the largest value that \( E_1 \) can assume, and will be given shortly.

We first consider the factor \( P(E_1 = j) \) in Equation (3.74). We have
\[ P(E_1 = j) = P(E_1 = j \land \zeta = n) + P(E_1 = j \land \zeta = n-1) + \ldots + P(E_1 = j \land \zeta = 1). \]  

(3.75)

To simplify this equation, we will approximate \( P(E_1 = j) \) by the quantity \( P(E_1 = j \land \zeta = n) \). This approximation is justified because \( Y \) is uniformly distributed over \([1, M-1]\), and consequently \( y_n > 1 \) for \((m_n-1)/m_n\) of all division problems, where, for practical moduli, \( m_n \) is large. So we have

\[ P(E_1 = j) = P(E_1 = j \land \zeta = n), \]  

(3.76)

and substituting Equation (3.76) in Equation (3.74), we have

\[ P(E_2 = 0|E_1 = j) = \max_{j=1} P(E_2 = 0|E_1 = j)P(E_1 = j \land \zeta = n). \]  

(3.77)

We now consider the factor \( P(E_2 = 0|E_1 = j) \) from Equation (3.74). We have

\[ P(E_2 = 0|E_1 = j) = \frac{P(E_2 = 0 \land E_1 = j)}{P(E_1 = j)} \]

\[ = \frac{P(E_2 = 0 \land E_1 = j \land \zeta = n) + P(E_2 = 0 \land E_1 = j \land \zeta = n-1) + \ldots + P(E_2 = 0 \land E_1 = j \land \zeta = 1)}{P(E_1 = j \land \zeta = n) + P(E_1 = j \land \zeta = n-1) + \ldots + P(E_1 = j \land \zeta = 1)} \]

\[ = \frac{P(E_2 = 0|E_1 = j \land \zeta = n)}{P(E_1 = j \land \zeta = n)} = P(E_2 = 0|E_1 = j \land \zeta = n), \]  

(3.78)

and we will therefore use the approximation

\[ P(E_2 = 0|E_1 = j) = P(E_2 = 0|E_1 = j \land \zeta = n) \]  

(3.79)

for the same reason used in dealing with \( P(E_1 = j) \).

This expression will be further simplified. We have, by the OSRA iteration and the definition of the OSRA estimate,
\[ P(E_2=0|E_1=j) = P(X-jY < (y_n+1)m_{n-1} \ldots m_1) \] (3.80)

But
\[ (y_n+1)m_{n-1} \ldots m_1 = Y + R_Y, \] (3.81)

where \( R_Y \) is the error produced by rounding up on \( y_n \). Clearly, for this rounding, we have
\[ 0 < R_Y < m_{n-1} \ldots m_1, \] (3.82)

so that
\[ P(E_2=0|E_1=j) = P(X-jY < Y+R_Y) = P(X<(j+1)Y+R_Y) \]
\[ = P(X/Y < j+1 + R_Y/Y). \] (3.83)

To simplify the situation even further, we observe that for most \( Y \) for which \( y_n>1 \), \( R_Y/Y \ll 1 \). This is because, for the one-sided rounding scheme,
\[ Y = y_n m_{n-1} \ldots m_1 + y_{n-1} m_{n-2} \ldots m_1 + \ldots + y_1 \]
rounded to \( (y_n+1)m_{n-1} \ldots m_1 \)
\[ (3.84) \]

introduces a small error for most such \( Y \). This error is negligible because small \( y_n \) occur only a small percentage of the time, assuming randomly distributed \( Y \). Therefore, we can neglect \( R_Y/Y \) in Equation (3.83). This gives
\[ P(E_2=0|E_1=j) = P(X/Y < j+1) = P(\lfloor X/Y \rfloor < j). \] (3.85)

This is the desired approximating expression for the conditional probability \( P(E_2=0|E_1=j) \).
Substituting the approximation for $P(E_2=0|E_1=j)$ in Equation (3.77), we have

$$P(E_2=0|E_1\neq 0) = \max_{j=1} P(|X/Y|<j) P(E_1=j\& \& n) .$$

(3.86)

The quantity $\max$ is the largest value that $E_1$ can have. It is

$$\max = \left\lfloor \frac{m_n-1}{2} \right\rfloor .$$

(3.87)

This maximum estimate occurs when $y_n=1$ and $x_n=m_n-1$.

We now need formulas for $P(E_1=j\& \& n)$ ($j=0,1,2,\ldots \max$) and $P(|X/Y|<j)$ ($j=1,2,\ldots \max$) in Equation (3.86). Both formulas are derived in Appendix G. For $P(|X/Y|<j)$, $0<j<M-1$, we have

$$P\left(\left|\frac{X}{Y}\right|<j\right) = \frac{1}{2} + \frac{j}{\frac{M(M-1)}{i+1} S \left\lfloor \frac{M}{i+1} \right\rfloor - \frac{i}{\frac{M(M-1)}{i+1} S \left\lfloor \frac{M}{i+1} \right\rfloor }}.$$

(3.88)
where we have used \( S_\alpha = \frac{\alpha(\alpha+1)}{2} \) to denote the sum of the first \( \alpha \) natural numbers, and as before, \( M \) is the product of the system moduli. Note that Equation (3.88) is the probability distribution function for the truncated quotient \(|X/Y|\), assuming uniformly distributed \( X \) and \( Y \). A discussion and plot of \( P(|X/Y|=j) \) (\( j=0,1,2,\ldots, \text{max} \)), which is the probability density function of the truncated quotient, will be given later.

A formula for \( P(E_1=j|\Lambda\&=n) \) (\( j=0,1,2,\ldots, \text{max} \)), is derived in Appendix G also, and is

\[
P(E_1=j|\Lambda\&=n) = \begin{cases} 
\frac{1}{m_n} \left( \sum_{j=0}^{m_n} \frac{m_n}{m_n-j} \cdot \frac{m_n}{m_n-j+1} \right) + \frac{j+1}{m_n^2} \sum_{j=0}^{m_n} \frac{m_n}{m_n-j} \cdot \frac{j}{m_n^2} \cdot \frac{m_n}{m_n-j} - \frac{1}{m_n^2}, \\
0 < j < \left\lfloor \frac{m_n-1}{2} \right\rfloor \\
\frac{m_n+1}{2m_n} - \frac{1}{m_n^2}, \quad j=0
\end{cases}
\]

(3.89)

Using Equation (3.76), this gives an approximation to \( P(E_1=j) \). A plot and discussion of this expression will be given later.

Making the approximation \( P(E_1=0) \approx P(E_1=\&\&=n) \), and substituting for \( \text{max} \) in Equation (3.73), we have
\[ \hat{T}_{\text{OSRA}} = 2t \left( \frac{m_n+1}{2m_n} - \frac{1}{m_n^2} \right) + (3t+3) \left( \sum_{j=1}^{m_n-1} P \left( \left| \frac{X}{Y} \right| < j \right) P(E_1=j) P(E_2=0) \right) \]

+ (4t+6) \left( 1 - \left( \frac{m_n+1}{2m_n} - \frac{1}{m_n^2} \right) - \sum_{j=1}^{m_n-1} \left( \frac{m_n-1}{2m_n} - \frac{1}{m_n^2} \right) P \left( \left| \frac{X}{Y} \right| < j \right) P(E_1=j) P(E_2=0) \right), \tag{3.90} \]

where \( P \left( \left| \frac{X}{Y} \right| < j \right) \) and \( P(E_1=j) P(E_2=0) \) are given in Equations (3.88) and (3.89), respectively. This is the desired expression for the lower bound on the average running time of the OSRA.

Table 5 lists results of this section. The right hand column contains values of \( \hat{T}_{\text{OSRA}} \) for the modulus sets M1, M2 and M3 from Section C. Note that these values were computed using Equation (3.90) with \( t=2(n-1)=18 \). Also included in the table are computationally derived values for \( P(E_1=0) \), obtained from

<table>
<thead>
<tr>
<th>MODULUS SET</th>
<th>( P(E_1=0) )</th>
<th>( P(E_2=0, E_1=0) )</th>
<th>( \hat{T}_{\text{OSRA}} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>{23,29,31,37,41,43,47,53,59,61}</td>
<td>.508</td>
<td>.387</td>
<td>48.5</td>
</tr>
<tr>
<td>{31,37,41,43,47,53,55,59,61,63}</td>
<td>.508</td>
<td>.388</td>
<td>48.5</td>
</tr>
<tr>
<td>{3, 5, 7,11,13,17,19,23,29,31}</td>
<td>.515</td>
<td>.366</td>
<td>48.7</td>
</tr>
</tbody>
</table>

**Average:** 48.6
\[ P(E_1 = 0) = P(E_1 = 0 \wedge \lambda = n) \]  \hspace{1cm} (3.91)

and Equation (3.89), and values of \( P(E_2 = 0 \wedge E_1 \neq 0) \) obtained from Equation (3.86).

The average lower bound (48.6) is within 3\% of the average OSRA running time (49.6), which was found by the simulator.

The probabilities \( P(E_1 = 0) \) and \( P(E_2 = 0 \wedge E_1 \neq 0) \) in the table are approximations for the probabilities that the OSRA stops after making one and two estimates, respectively. As such, they are estimates of the first two terms of the probability density function of running time for the OSRA. These are plotted in Figures 42 through 44, which are plots for the same modulus sets of the computationally derived OSRA probability density functions, modified by "clustering" small values of probability into these first two terms. The theoretically predicted values of the terms are indicated on the plots, and it can be seen that there is very good agreement between the simulated and theoretical values. The values agree to within 1\% for the first term, and to within 5\% for the second.

A third term \( P(E_3 = 0 \wedge E_2 = 0 \wedge E_1 \neq 0) \) was not included in Table 5 because the approximations valid for \( P(E_1 = 0) \) and \( P(E_2 = 0 \wedge E_1 \neq 0) \) are not valid for this term, as shown in Appendix G.

In the preceding derivation expressions for \( P(j_X/Y_j < j) \) and \( P(E_1 = j \wedge \lambda = n) \) were derived (see Equations (3.88) and (3.89)). These functions are plotted in Figures 45 and 46. Also plotted, in Figure 47, is the probability density function of truncated quotient. Note that Figures 46 and 47 are monotonically decreasing, as we would expect.
CLUSTERED PROBABILITY DENSITY
OF RUNNING TIME
MODULI \{23, 29, 31, 37, 41, 43, 47, 53, 59, 61\}

OSRA
X: THEORY PREDICTION

Figure 42. Experimental and theoretical OSRA running time probability density functions.
Figure 43. Experimental and theoretical OSRA running time probability density functions.
Figure 44. Experimental and theoretical OSRA running time probability density functions.
Figure 45. Distribution function for truncated quotient.
Figure 46. Approximate probability density function for OSRA initial estimate.
Figure 47. Density function for truncated quotient.
Furthermore, note that initial OSRA estimates greater than 1 rarely occur. Also, truncated quotient values greater than 1 rarely occur. This fact is not in the literature, and is significant because it means that any RNS division algorithm that quickly solves problems with small quotients will be faster.

F. Summary of Chapter III

Chapter III is a presentation of two new RNS division algorithms and a comparison of their stored table sizes and statistics of performance with those of the OSRA. It also contains the derivation of a formula for a lower bound on the average running time of the OSRA.

Section A contains a detailed presentation of the first new algorithm, called the Two Sided Rounding Algorithm (TSRA). Section C contains a comparison of the stored table sizes, mean running times and standard deviations about the mean of the TSRA and OSRA, and contains the conclusion that the TSRA uses the same amount of storage, has 2% better mean running time, and has 17% better standard deviation about the mean. Section B contains a detailed presentation of the second new algorithm, called the Signed Algorithm (SA). Section D contains a comparison of the stored table sizes, mean running times and standard deviations about the mean of the SA and OSRA, and contains the conclusion that the SA uses at most the same amount of storage, has 3% greater average running time, and exhibits 12% smaller standard deviation about the mean. Section E contains a derivation of a lower bound on the mean running time of the OSRA, and the conclusion that this lower bound is within 3% of the computationally derived mean running time, for the modulus sets used.
CHAPTER IV
THE RECIPROCAL ALGORITHM

This chapter contains a presentation of another new residue number system division algorithm called the Reciprocal Algorithm (RA). It is subtractive and employs a suitably defined "reciprocal" of the divisor to compute quotient estimates. It uses an amount of storage proportional to the sum of the moduli (as compared with the square of the maximum modulus which is required by the OSRA), is slightly faster and has better standard deviation than the OSRA. Furthermore, it uses an extra overflow modulus of size one less than the maximum modulus if such a modulus is not included in the original set.

The chapter begins with a description of the RA in Section A. After a statement of notation, the method is introduced and given in detail. The algorithm is then summarized, and two examples are presented. It is then formally stated and shown to be correct. Subsequently, in Section B, the storage requirements, mean running times and standard deviations of running time about the mean for particular modulus sets are given and compared with those of the OSRA.
A. The Reciprocal Algorithm

A.1. Introduction

In the following paragraphs we explain the Reciprocal Algorithm. After a statement of notation the method is motivated and discussed. The RA is then summarized and two example problems are solved. Then, the RA is formally stated and shown to be correct.

The notation used in Chapters II and III is used in this chapter. Furthermore, for convenience we define the modulus product

\[ P_a = \prod_{j=1}^{a} m_j \]  

(4.1)

and \( P_0 = 1 \).

A.2. Discussion of the RA

The Reciprocal Algorithm finds the truncated quotient \( \lfloor X/Y \rfloor \) for positive \( X \) and \( Y \), and uses the standard ID \([0, M-1]\). The RA is subtractive, and uses the same iterative procedure as the OSRA, viz.,

\[ X_{i+1} = X_i - E_{i+1} Y \]  

(4.2)

with \( X_0 = X \), to obtain successive reduced numerators. The RA stops in the \( r \)th iteration when either \( X_r < Y \) or \( E_{r+1} = 0 \). When either of these stopping conditions is met, the truncated quotient is the sum of the quotient estimates plus a small error term \( E' \), viz.,
where

\[ E' = \begin{cases} 
0, & \text{if } X_Y < Y \\
1, & \text{if } E_{i+1} = 0 
\end{cases} \quad (4.4) \]

As explained in Chapter II, the estimate \( E_{i+1} \) in Equation (4.2) is an estimate of the quantity \( X_i / Y \) and is the heart of any subtractive algorithm. Accordingly, its derivation and calculation will now be discussed.

The estimate used by the RA is defined by

\[ E_{i+1} = \begin{cases} 
x_k \left\lfloor \frac{p_{i+1}}{Y} \right\rfloor_m, & k = i \\
x_k \left\lfloor \frac{p_{i+1}}{Y} \right\rfloor_{m_{i+1} \cdots m_{k-1}}, & k > i+1 \end{cases} \quad (4.5) \]

It was developed to eliminate two drawbacks of the TSRA, namely the large amount of storage required and the need for sign detection.

The estimate defined in Equation (4.5) is an approximation of a "nice" estimate whose computation involves an "integer reciprocal" of the divisor. This estimate is

\[ E_{i+1} = \left\lfloor \frac{X_i}{Y} \right\rfloor \quad (4.6) \]
and has not been used in the literature, although an algorithm \cite{16} for
for the residue computation of a "squared-range" inverse $|S^2/Y|$ has
been published in the Soviet Union. It has several attractive
properties. The first is that it differs from $|X_i/Y|$ by at most one,
and therefore offers the possibility of extremely fast division.
Furthermore, it is one-sided (that is, never exceeds $X_i/Y$), and
therefore eliminates the need for sign detection. Further, it offers
the possibility of small storage because it can be found by the scaling
operation, as opposed to the table of stored quotients required by the
TSRA. That is, the quotient estimate in Equation (4.6) can be
calculated by scaling (by M) the product of $X_i$ and $|M/Y|$, assuming
$|M/Y|$ could be found by a small stored table.

The estimate in Equation (4.6) must be modified for reasons which
will be explained shortly. The first modification is an approximation of
$X_i$ by its most significant nonzero mixed radix term, viz.:

$$X_i = x_k p_k^{-1}.$$  \hspace{1cm} (4.7)

This allows cancellation of modulus product $P_{k-1}$ in Equation (4.6).
The second is an approximation of $|M/Y|$ by the largest multiple of
$m_n..m_{k+1}$ that it exceeds or equals, viz.:

$$\left\lfloor \frac{M}{Y} \right\rfloor = m_n..m_{k+1} \left\lfloor \frac{P_k}{Y} \right\rfloor.$$  \hspace{1cm} (4.8)

It allows cancellation of moduli $m_n..m_{k+1}$ in Equation (4.6). These
approximations were made to overcome two drawbacks in Equation (4.6).
The first drawback is that the product $X_i|M/Y|$ can overflow. Several
overflow moduli would have to be included because this product could be as large as $M(M-1)$. Ideally, we would like to include only a single overflow modulus, the size of which is approximately equal to the maximum modulus.

The second drawback of the estimate in Equation (4.6) is that the quantity $|M/Y|$ is costly to find. When $Y$ is small, it requires a large amount of storage and/or a large number of residue operations to find. The second approximation replaces the costly calculation of $|M/Y|$ with the calculation of the relatively cheap quantity $|P_k/Y|$. The latter quantity, henceforth called the "divisor reciprocal", is less costly to find than $|M/Y|$ because $Y$ is never "too small" with respect to $P_k$. Consequently, the number of possible values of $|P_k/Y|$ is small (in fact, at most $m_k$), and therefore it can be found by a search of a small table, as will be explained in Section A.2.2.

Substituting Equations (4.7) and (4.8) in Equation (4.6), we obtain Equation (4.5), the quotient estimate used by the RA.

Equation (4.5) reduces or eliminates the overflow drawback of Equation (4.6), i.e., Equation (4.5) does not require overflow moduli if the original moduli are chosen so that the two largest differ by one (i.e., $m_{n-1} = m_n-1$). Furthermore, for all other modulus sets, only a single overflow modulus is required, and it is used only when $k = e = n$. This is an attractive property of the RA. The proof of these statements, however, is rather detailed. Therefore, it is given in Appendix H.
A.2.1 Computation of Estimate $E_{i+1}$

Computation of $E_{i+1}$ in Equation (4.5) requires the residue codes for the three quantities $x_k$, $|P_k/Y|$ and $m_{k+1}...m_{k-1}$. The residue code for $m_{k+1}...m_{k-1}$ is found by a stored table, indexed by $k$ and $l$, which stores the residue codes for all such products, as was done for the OSRA, TSRA and SA. The residue code for $|P_k/Y|$ is found by a search of a stored table, as will be discussed shortly. The residue code for $x_k$ is found, as for the OSRA, TSRA and SA, by a zero-operation base extension of $x_k$ from $m_k$ to all other moduli.

Now consider the computation of $E_{i+1}$. If $k>l+1$, the residue code for $E_{i+1}$ is the product of the residue codes for $x_k$, $|P_k/Y|$ and, if $k>l+1$, $m_{k+1}...m_{k-1}$. If $k=l$, the residue code for $E_{i+1}$ is found by scaling (by $m_k$) the product $x_k|P_k/Y|$, and then base extending to restore the erased $k$th residue digit. For $k=l<n$, this base extension is done from the $n$th digit, because it is shown in Appendix H that $E_{i+1}<m_n$. For $k=l=n$, it is shown in the same appendix that an overflow modulus $m_{n+1}=m_{n-1}$ must be used for this base extension if the maximum moduli are not consecutive. However, if they are, then $m_{n-1}$ is used for the base extension.

There are two special cases when the computation of $E_{i+1}$ is done differently. If $|P_k/Y|=1$, then the estimate is computed using

$$E_{i+1} = \begin{cases} 
0 & \text{, if } k=l \\
x_k & \text{, if } k=l+1 \\
x_km_{l+1}...m_{k-1} & \text{, if } k>l+1
\end{cases} \quad (4.9)$$
which is Equation (4.5) evaluated for \(|_P^k/Y_|=1\). This is done to
increase efficiency, since \(|_P^k/Y_|=1\) for approximately half of all
division problems. The second special case occurs when \(k=x\). A test for
the condition \(|_P^k/Y_|=m_k\) is made for the same reason. In such a case,
\(E_{i+1}=x_k\) and no residue operations are needed.

A.2.2 Determination of \(|_P^k/Y_|\) Using the Divisor Reciprocal Table

We now discuss how the divisor reciprocal \(|_P^k/Y_|\) is found. For \(Y\)
in the range

\[P_{k-1} < Y < P_k-1\]  \hspace{1cm} (4.10)

(that is, for a given \(k\)), it is found by a stored table which stores the
largest value of \(Y\) for each unique value of \(|_P^k/Y_|\). As discussed
previously, the table has small size for any \(k\) because \(Y\) is never "too
small" compared with \(P_k\). Consequently, many \(Y\) share the same value of
\(|_P^k/Y_|\). In general, the size of the table is at most \(m_k^e\). We will now
discuss the table contents, and subsequently explain how \(|_P^k/Y_|\) is
found.

The stored table used to find the divisor reciprocal \(|_P^k/Y_|\) for a
given \(k\) will be denoted by \(T_k\). The size of \(T_k\) is small. This is
because the mapping \(|_P^k/Y_|\) over \(Y\) is many-to-one. That is, many
values of \(Y\) are mapped to one value of \(|_P^k/Y_|\).

In order to discuss the entries in \(T_k\), we need the following
notation. We denote by \(j_1<j_2<...<j_s\) the unique values that \(|_P^k/Y_|\) can
assume as \(Y\) ranges throughout \([P_{k-1},P_k-1]\). Furthermore, we let \(Y_{j_i}\)

169
denote the largest \( Y \) in \([P_{\alpha-1}, P_{\alpha-1}]\) such that \(|\_\_ P_\alpha / Y_| = j_1 \). The values \( Y_{j_1} \) can be thought of as the values of \( Y \) for which \(|\_\_ P_\alpha / Y_| \) "changes" as \( Y \) assumes values starting at \( P_{\alpha-1} \) and decreases to \( P_{\alpha-1} \). The values \( j_1 \) are then equal to \(|\_\_ P_\alpha / Y_{j_1} \_| \).

Divisor reciprocal table \( T_\alpha \) stores the ordered pairs of integers \((Y_{j_1}, j_1)\) for \( i=1, 2, \ldots, s \) in increasing order on \( j_1 \). The integers \( Y_{j_1} \) are mixed radix encoded, while the integers \( j_1 \) are residue coded. The size of \( T_\alpha \), at most, is \( m_\alpha \), because \( 1 < |\_\_ P_\alpha / Y_| < m_\alpha \). This makes the storage used by the RA proportional to the sum of the moduli, which is an improvement of the OSRA and TSRA storages.

The residue coded value of \(|\_\_ P_\alpha / Y_| \) is found by using the mixed radix digits of \( Y \) to search the entries \( Y_{j_1} \) in \( T_\alpha \). The search continues until the pair \((Y_{j_a}, j_a)\) such that

\[
Y_{j_a+1} \leq Y < Y_{j_a}
\]

is found. At such a time, the residue encoded value of \(|\_\_ P_\alpha / Y_| = j_a \) is read out. This process needs to be done only once, at the beginning of a division problem.
A.3 Summary of the RA

The RA is shown in flowchart form in Figure 48, and is summarized as follows. $Q$ is defined to be the partial sum for the quotient.

**Step 1)** Set $Q = 0$ and the iteration counter $i = 0$.

**Step 2)** Do a mixed radix conversion on $Y$.

**Step 3)** Do a mixed radix conversion on $X_i$.

**Step 4)** If $X_i < Y$, (that is, if the stopping condition is satisfied), then stop with $|X/Y| = Q$.

**Step 5)** If $i = 0$, find $|P/Y|$ in residue by searching table $T_2$ with the mixed radix digits of $Y$.

**Step 6)** If $k > 2$, go to Step 8). Otherwise do the following:
   a) If $|P/Y| = 1$, then $E_{i+1} = 0$, so add $E' = 1$ to $Q$ and stop.
   b) If $|P/Y| = m_k$, then go to Step 10) with $E_{i+1} = x_k$.
   c) Compute $|X| = |P/Y|/m^k = E_{i+1}$ using the scaling operation. Restore the $k$th residue digit using the $n$th if $k < n$, or using the $(n+1)st$ if $k = n$.

**Step 7)** If $E_{i+1} = 0$ (that is, if the stopping condition is satisfied), then add $E' = 1$ to $Q$ and stop. Else go to Step 10).

**Step 8)** If $k > 2 + 1$, go to Step 9). Otherwise, do the following:
   a) If $|P/Y| = 1$, then set $E_{i+1} = x_k$ and go to Step 10).
   b) Compute $E_{i+1} = x_k|P/Y|$, and go to Step 10).

**Step 9)** If $|P/Y| = 1$, compute $E_{i+1} = x_k m_{k+1} \cdots m_{k-1}$. Otherwise, compute $E_{i+1} = x_k|P/Y| m_{k+1} \cdots m_{k-1}$.

**Step 10)** Add $E_{i+1}$ to the running quotient sum $Q$, and compute the new numerator $X_{i+1} = X_i - E_{i+1} Y$. Add 1 to the iteration counter $i$, and go to Step 3).
Figure 48. RA flowchart.
A.4 Examples of Use of the RA

The following two examples illustrate the use of the RA using the modulus set \( \{m_3, m_2, m_1\} = \{17, 13, 11\} \). The first shows details of calculations when the overflow modulus is not used, while the second shows the details when the overflow modulus \( m_4 = 16 \) is used. As before, angle brackets "\( <a_3, a_2, a_1> \)" denote mixed radix digits, with \( a_1 \) the least significant digit, and parentheses "\( (r_3, r_2, r_1) \)" denote residue digits. The overflow residue digit \( r_4 \) in the second example is listed on the right as \( (r_3, r_2, r_1, r_4) \). The number of each solution step corresponds to the number of the same step in the Summary. The number of residue operations required to find the truncated quotient is indicated by the operation counter. At each step, it is incremented by the number of residue operations required. In the second example, "d" denotes a "don't care" residue digit.

Example 1: Compute \( \frac{2200}{20} \).

Solution: 2) Mixed radix conversion of \( Y \) gives \( Y\rightarrow<0, 1, 9> \) and \( k=2 \). Add 4 to the operation counter, because the mixed radix conversion requires \( 2(n-1) \) operations, and \( n=3 \).

3) Mixed radix conversion on \( X_0 \) gives \( X_0\rightarrow<15, 5, 0> \) and \( k=3 \). Add 4 to the operation counter.

4) Since \( X_0 > Y \), we proceed to make a quotient estimate.

5) Since \( i=0 \), we search the table \( T_2 \) with \( Y\rightarrow<0, 1, 9> \). \( T_2 \) contains the entries...
We find \(<0,1,6> < Y \leftrightarrow <0,1,9> < <0,1,9>, so that \(|P_2/Y|\leftrightarrow (7,7,7)\) from the table.

8) Since \(k=\lambda+1\), and \(|P_2/Y|\#1, we compute the residue digits of \(E_{j+1}\) as

<table>
<thead>
<tr>
<th>Moduli:</th>
<th>17</th>
<th>13</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(x_3:)</td>
<td>15</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>(</td>
<td>P_2/Y</td>
<td>:)</td>
<td>x</td>
</tr>
<tr>
<td>(E_1:)</td>
<td>3</td>
<td>1</td>
<td>6</td>
</tr>
</tbody>
</table>
10) The new numerator $X_1$ is calculated as

<table>
<thead>
<tr>
<th>Moduli:</th>
<th>17</th>
<th>13</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>$E_1$:</td>
<td>3</td>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>$Y$:</td>
<td>x</td>
<td>3</td>
<td>7</td>
</tr>
<tr>
<td>$E_1Y$:</td>
<td>9</td>
<td>7</td>
<td>10</td>
</tr>
<tr>
<td>$X_0$:</td>
<td>7</td>
<td>3</td>
<td>0</td>
</tr>
</tbody>
</table>

$X_1$: 15 9 1

Add 2 to operation counter. Also, the quotient sum is updated as $Q\leftarrow(0,0,0)+(3,1,6)=(3,1,6)$. Add 1 to the operation counter.

3) Mixed radix conversion on $X_1$ gives $X_1\leftarrow<0,9,1>$ and $k=2$.

Add 4 to the operation counter.

4) Since $X_1>Y$, we proceed to compute another estimate.

6) Since $|P_2/Y|\neq1$ or $m_2$, we compute
<table>
<thead>
<tr>
<th>Moduli:</th>
<th>17</th>
<th>13</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_2$:</td>
<td>9</td>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td>$\lceil P_2/Y\rceil$:</td>
<td>x</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td>12</td>
<td>11</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>-11</td>
<td>11</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>x</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>$E_2$:</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>restore using $m_3$:</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
</tbody>
</table>

Add 3 to the operation counter.

7) Since $E_2 \neq 0$, we proceed to Step 10).

10) The new numerator $X_2$ is calculated as

<table>
<thead>
<tr>
<th>Moduli:</th>
<th>17</th>
<th>13</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>$E_2$:</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>$Y$: x</td>
<td>3</td>
<td>7</td>
<td>9</td>
</tr>
<tr>
<td>$E_2Y$:</td>
<td>12</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>$X_1$:</td>
<td>15</td>
<td>9</td>
<td>1</td>
</tr>
<tr>
<td>$X_2$:</td>
<td>3</td>
<td>7</td>
<td>9</td>
</tr>
</tbody>
</table>

Add 2 to the operation counter. Also, the quotient sum $Q$ is updated as $Q\leftarrow(3,1,6)+(4,4,4)=(7,5,10)$. Add 1 to the operation counter.
3) Mixed radix conversion on \( X_2 \) gives \( X_2 = (0, 1, 9) \) and \( k = 2 \).

Add 4 to the operation counter.

4) Since \( X_2 > Y \), we proceed to make a quotient estimate.

6) Since \( |P_2/Y| = 1, m_2 \), we compute

<table>
<thead>
<tr>
<th>Moduli:</th>
<th>17</th>
<th>13</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x_2 ):</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>(</td>
<td>P_2/Y</td>
<td>): x</td>
<td>7</td>
</tr>
<tr>
<td>( _ )</td>
<td>7</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>( _ )</td>
<td>7</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>( x )</td>
<td>4</td>
<td>6</td>
<td></td>
</tr>
</tbody>
</table>

\( E_3: 0 \)

restore using \( m_3: 0 0 0 \).

Add 3 to the operation counter.

7) Since \( E_3 = 0 \), we must add \( E' = 1 \) to \( Q \). We have

\( Q = (7, 5, 10) + (1, 1, 1) = (8, 6, 0) \) and \( 110 \). Add 1 to the operation counter because of this additional operation.
Example 2: For the same modulus set, compute \[ \frac{2043}{171} \].

Solution: Note that the overflow modulus \( m_4 = 16 \) is needed for this example.

2) Mixed radix conversion on \( Y \) gives \( Y \leftrightarrow 1, 2, 6 \) and \( k = 3 \).
   Add 4 to the operation counter.

3) Mixed radix conversion on \( X_0 \) gives \( X_0 \leftrightarrow 14, 3, 8 \) and \( k = 3 \).
   Add 4 to the operation counter.

4) Since \( X_0 > Y \), we proceed to make a quotient estimate.

5) Since \( i = 0 \), \( \lfloor P_3/Y \rfloor \) is found by a search of the reciprocal table \( T_3 \) with \( Y \leftrightarrow 1, 2, 6 \). \( T_3 \) contains the entries

\[
\begin{align*}
&(<16,12,10>, (1,1,1,1)) \\
&(<8,6,5>, (2,2,2,2)) \\
&(<5,8,7>, (3,3,3,3)) \\
&(<4,3,2>, (4,4,4,4)) \\
&(<3,5,2>, (5,5,5,5)) \\
&(<2,10,9>, (6,6,6,6)) \\
&(<2,5,6>, (7,7,7,7)) \\
&(<2,1,6>, (8,8,8,8)) \\
&(<1,11,6>, (9,9,9,9)) \\
&(<1,9,1>, (10,10,10,10)) \\
&(<1,7,1>, (11,11,0,11)) \\
&(<1,5,4>, (12,12,1,12)) \\
&(<1,4,0>, (13,0,2,13)) \\
&(<1,2,8>, (14,1,3,14)) \\
&(<1,1,8>, (15,2,4,15)) \\
&(<1,0,8>, (16,3,5,0)) \\
&(<1,0,0>, (0,4,6,1)).
\end{align*}
\]
We find \(<1, 1, 8> < Y + + <1, 2, 6> < <1, 2, 8>\), so that \\(_{P3/Y_1}^{++}(14, 1, 3, 14)\) from the table. Note that modulus 16 has been added and that in general, \(T_n\) will be the only table which will store information for the \((n+1)\)st residue digit.

6) Since \(_{P3/Y_1}^{++1}, m_3\), we compute

<table>
<thead>
<tr>
<th>Moduli:</th>
<th>17</th>
<th>13</th>
<th>11</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>(x_3:)</td>
<td>14</td>
<td>1</td>
<td>3</td>
<td>14</td>
</tr>
<tr>
<td>(_{P3/Y_1}: x)</td>
<td>14</td>
<td>1</td>
<td>3</td>
<td>14</td>
</tr>
<tr>
<td></td>
<td>9</td>
<td>1</td>
<td>9</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>9</td>
<td>9</td>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>5</td>
<td>0</td>
<td>11</td>
</tr>
<tr>
<td>(x)</td>
<td>10</td>
<td>2</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>(E_1: )</td>
<td>11</td>
<td>0</td>
<td>11</td>
<td></td>
</tr>
</tbody>
</table>

restore using \(m_4:\) 11 11 0 11.

Add 3 to operation counter.

7) Since \(E_1 \neq 0\), we proceed to Step 10).

10) The new numerator \(X_1\) is calculated as
Moduli: 17 13 11 16

| E1:  | 11 11 0 11 |
| Y: x | 1 2 6 d   |

| E1Y:  | 11 9 0 d  |
| X1:  | 9 6 8 d   |

Add 2 to the operation counter. Also, the quotient sum Q is updated as Q += (0,0,0) + (11,11,0) = (11,11,0). Add 1 to the operation counter.

3) Mixed radix conversion on X1 gives X1 «< 1, 1, 8>, and k = 3. Add 4 to operation counter.

4) Since X1 < Y, we stop with Q += (11,11,0) += 11.

A.5 Formal Statement of the RA

The RA finds the truncated quotient \( \lfloor X/Y \rfloor \) for any numerator X and denominator Y≠0 in the ID \([0,M-1]\). The system moduli \( m_n, m_{n-1}, \ldots, m_1 \) are assumed to be positive pairwise relatively prime integers, ordered as \( m_n > m_{n-1} > \ldots > m_1 \). The RA requires the use of an overflow modulus \( m_{n+1} = m_{n-1} \) if \( m_{n-1} = m_{n-1} \).

The RA uses the iteration

\[
X_{i+1} = X_i - E_{i+1}Y
\]

with \( X_0 = X \) and stopping conditions \( X_r < Y \) or \( E_{r+1} = 0 \). The estimates \( E_{i+1} \) are given by
The residue encoded value of \( E_{i+1} \) is found by means of stored table \( T_{\lambda} \) as indicated in the algorithm discussion. Residue encoded products \( m_{\lambda+1} \ldots m_{k-1} \) are found by a stored table, indexed by \( k \) and \( \lambda \), as is done for the TSRA, OSRA and SA.

If either stopping condition is satisfied on the \( r \)th iteration, then

\[
X_r = \sum_{i=0}^{r-1} E_{i+1} + E',
\]

(4.14)

where

\[
E' = \begin{cases} 
0, & \text{if } X_r < Y \\
1, & \text{if } E_{r+1} = 0.
\end{cases}
\]

(4.15)

\section*{A.6 Proof of the Validity of the RA}

The proof of the validity of the RA will now be given, and it will consist of 4 parts. The first part, Lemma 8, gives the range of values that nonzero quotient estimates must have if the \( X_r < Y \) stopping condition is to be satisfied eventually. The second part, Lemma 9, states that if the stopping condition \( E_{r+1} = 0 \) is encountered, then \( X_r < 2Y \). This lemma is needed when showing that \( E' = 1 \) in Theorem 4. Lemma 10 states that the
estimate defined in Equation (4.13) either lies in the range specified by Lemma 8, or else equals zero. Theorem 4 states that the quotient is as given in Equation (4.14), and that the RA eventually halts.

**Lemma 8:** For the iteration \( X_{i+1} = X_i - \frac{E_{i+1}Y}{X_i} \), we have

\[ 0 < E_{i+1} < \frac{X_i}{Y} \Rightarrow 0 < X_{i+1} < X_i. \]

**Proof:** We have

\[ 0 < E_{i+1} < \frac{X_i}{Y} \Rightarrow X_i = \frac{X_i}{Y} Y < X_{i+1} < X_i - (0)Y \]

\[ \iff 0 < X_{i+1} < X_i \]

QEd

**Lemma 9:** For the estimate \( E_{i+1} \) given in Equation (4.13), viz.,

\[
E_{i+1} = \begin{cases} 
\frac{x_k}{m_k} \frac{p_{\xi}}{Y} & \text{if } k=\xi \\
\frac{x_k}{m_k} \frac{p_{\xi+1}}{Y} & \text{if } k=\xi+1 \\
\frac{x_k}{m_k} \frac{p_{\xi}}{Y} \frac{m_{\xi+1} \cdots m_{k-1}}{X_i} & \text{if } k>\xi+1
\end{cases}
\]  

(4.16)

we have \( E_{i+1} = 0 \Rightarrow X_i < 2Y. \)

**Proof:** See Appendix I.

**Lemma 10:** For the estimate \( E_{i+1} \) given in Equation (4.13), we have

\[ 0 < E_{i+1} < X_i Y, \text{ for all } X_i, Y \neq 0. \]

**Proof:** See Appendix I.
Theorem 4: The RA eventually halts, at which time
\[ \left| \frac{X}{Y} \right| = \sum_{i=0}^{r-1} E_{i+1} + E', \]  
(4.17)

where
\[ E' = \begin{cases} 0, & \text{if } X_r < Y \\ 1, & \text{if } E_{r+1} = 0 \end{cases} \]  
(4.18)

Proof: Halting will be proven first.

Let \( \{X_i\} \) and \( \{E_{i+1}\} \), for \( i=0,1,... \), denote the sequence of numerators and estimates, respectively. By Lemma 10, we have \( 0 < E_{i+1} < Y \) for all \( i \). If \( E_{i+1} = 0 \), the RA halts. If \( E_{i+1} > 0 \), then \( X_{i+1} < X_i \) by Lemma 8. Therefore, \( \{X_i\} \) is a decreasing sequence of positive integers, and so \( \{E_{i+1}\} \) is also. Therefore, eventually for some \( r \), \( X_r < Y \) or \( E_{r+1} = 0 \), and the RA halts.

Now, to show that the RA computes \( \left| \frac{X}{Y} \right| \), we have
\[ X_1 = X - E_1 Y, \]
\[ X_2 = X_1 - E_2 Y, \]
\[ \vdots \]
\[ X_r = X_{r-1} - E_r Y, \]

and either \( X_r < Y \) or \( E_{r+1} = 0 \).

We have
\[ X_r = X_{r-1} - E_r Y \]
\[ = X_{r-2} - E_{r-1} Y - E_r Y \]
\[ = X_{r-3} - E_{r-2} Y - E_{r-1} Y - E_r Y \]
\[ = \ldots = X - E_1 Y - E_2 Y - \ldots - E_{r-1} Y - E_r Y \]

Therefore,
\[ X = (E_1 + E_2 + \ldots + E_r) Y + X_r \]

and so
\[
\left| \frac{X}{Y} \right| = \sum_{i=0}^{r-1} E_{i+1} + \left| \frac{X_r}{Y} \right|
\]

Let \( E' = \left| \frac{X_r}{Y} \right| \). If \( X_r < Y \), then \( E' = 0 \). If \( E_{r+1} = 0 \), then by Lemma 9, \( X_r < 2Y \). But \( X_r > Y \) because otherwise the RA would have stopped before calculating \( E_{r+1} \). Therefore, \( Y < X_r < 2Y \), and so \( E' = 1 \). Therefore,

\[
E' = \begin{cases} 
0 & \text{if } X_r < Y, \\
1 & \text{if } E_{r+1} = 0.
\end{cases}
\]

QED

A.7 Summary of Section A

In this section we have presented a new RNS division algorithm called the Reciprocal Algorithm (RA). The RA is iterative and uses a specially defined reciprocal of the divisor in conjunction with the scaling operation to compute quotient estimates. It requires an extra overflow modulus of size one less than the maximum modulus if such a
modulus is not already included in the original set. The section begins with a statement of notation and an introduction to the RA. Estimate calculation is then discussed in detail. The algorithm is then summarized, examples of its use given, and then it is formally stated and shown to be correct.

Section B: A Comparison of the RA and OSRA

The following paragraphs contain a comparison of the Reciprocal and One-Sided Rounding Algorithms' storage requirements and running time statistics. Storage requirements will be given first, followed by computationally derived values of mean running times and standard deviations about the mean.
B.1 Storage Requirements of Both Algorithms

The RA uses at most
\[ \sum_{i=1}^{n} (m_i - 1) + \frac{(n-2)(n-1)}{2} \] (4.19)
storage.

The RA uses n+1 stored tables. Tables \( T_\ell \), for \( \ell = 1, 2, \ldots, n \), are used to find the divisor reciprocal \( \lfloor \frac{\ell}{Y_{j_\ell}} \rfloor \) as discussed in Section A. The other table is used to find the residue code for products \( m_{\ell+1} \cdots m_{k-1} \).

The latter table is identical to that used by the TSRA, SA and OSRA. It is of size \( (n-2)(n-1)/2 \), as was given in Chapter III.

The first summation in Equation (4.19) comes from an upper bound placed on the size \( s \) of Table \( T_\ell \), namely \( s \leq m_\ell - 1 \). An exact expression for \( s \) is given in Appendix J, but is too complicated for easy interpretation. The upper bound is derived as follows. Recall from Section A that Table \( T_\ell \) stores ordered pairs \( (Y_{j_\ell}, j_\ell) \), where \( j_\ell \) for \( i = 1, 2, \ldots, s \) are the values that the reciprocal \( \lfloor \frac{\ell}{Y_{j_\ell}} \rfloor \) can have, and \( Y_{j_\ell} \) is the largest value of \( Y \) such that \( \lfloor \frac{\ell}{Y_{j_\ell}} \rfloor = j_\ell \). To get the expression for \( s \), recall from Equation (4.10) of Section A that
\[ P_{\ell-1} < Y < P_{\ell-1}. \] (4.20)

From these inequalities, we conclude that
\[ 1 < \lfloor \frac{P_\ell}{Y} \rfloor < m_\ell. \] (4.21)
Since $|P_j/Y|$ is monotonic over $Y$, there are at most $m_k$ values $Y_j$ with values $|P_j/Y|$ in that range. However, since we only need to store the values of $Y$ for which the reciprocal changes, there are at most $m_k - 1$ values in Table $T_k$.

Adding the storage required by the product table to the upper bounds for the storage required by the $n$ Tables $T_k$, we find that the RA uses, at most, the amount of storage given in Equation (4.19).

The RA uses significantly less storage than the OSRA. This is because the RA storage is proportional to the sum of the moduli (from Equation (4.19)), while the OSRA storage is proportional to the square of the maximum modulus (see Equation (3.59) of Chapter III).

The numbers of storage locations required by both algorithms for the five modulus sets used to simulate the algorithms are listed in Table 6, from which we conclude that the RA uses 69% less storage, on the average, than the OSRA.
TABLE 6

Amount of Storage Required by the RA and OSRA

<table>
<thead>
<tr>
<th>Modulus Set</th>
<th>Storage Required</th>
<th>% Difference</th>
</tr>
</thead>
<tbody>
<tr>
<td>{ 3, 5, 7,11,13,17,19,23,29,31}</td>
<td>184</td>
<td>471</td>
</tr>
<tr>
<td>{31,37,41,43,47,53,55,59,61,63}</td>
<td>516</td>
<td>1927</td>
</tr>
<tr>
<td>{23,29,31,37,41,43,47,53,59,61}</td>
<td>450</td>
<td>1806</td>
</tr>
<tr>
<td>{37,41,43,47,53,55,59,61,63,64}</td>
<td>549</td>
<td>1989</td>
</tr>
<tr>
<td>{ 2, 3, 5, 7,11,13,17,19,23,29}</td>
<td>155</td>
<td>414</td>
</tr>
</tbody>
</table>

Average: 69%

B.2 Computer Evaluation of the Mean and Standard Deviation of Running Time for the RA and OSRA

The following paragraphs contain a discussion of computationally derived mean running times and standard deviations for the Reciprocal and One Sided Rounding Algorithms. A simulating program flowchart is presented first, and then the choice of modulus sets is discussed. Subsequently, plots of mean running time, standard deviation about the mean and simulator-derived probability density functions are given. The section is concluded with a table of summarized performance data and conclusions.

A computer program which simulates both algorithms was written in the VAX-Fortran language for the VAX computer system, and is given in Appendix K. The program generated a sample set of division problems...
randomly selected from the IN [0,M-1], and determined the sample mean running time and the standard deviation about the mean for each algorithm. The running time is defined, as before, to be the number of elementary residue operations (viz., additions, subtractions and multiplications) required to find the truncated quotient. Table accesses and compares are not counted. This is the method prevalent in the literature for measuring the running time of RNS algorithms (see, for example [9,14,17 and 29]).

The flowchart for the simulating program is given in Figures 49 thru 51. Figure 49 is a high level flowchart which gives a global view of the information flow for both the simulating and statistical parts of the program. It is essentially the same as Figure 4 of Chapter III, except that an RA simulator is substituted for the TSRA simulator. Blocks 1 and 2, in the center and lower center respectively, are the RA and OSRA simulators, respectively. The remainder of the figure gives details of error checking, data input and output and statistical calculations. The formulas used for calculation of the statistics are given in the upper right of Figure 49. Figure 50 gives details of the RA simulator. This figure is essentially Figure 48 with counters added, because the number of residue operations required for division is basic for statistical evaluation of RNS division algorithms. Figure 51 gives details of the OSRA simulator, and is the same as Figure 21 of Chapter III.

There are several features to notice in the flowcharts. As with previous algorithms, in Figure 49, numerator X and denominator Y are
Figure 49. Global flowchart for RA and OSRA simulating program.
Figure 50. Detailed flowchart for RA simulator.
Figure 51. Detailed flowchart for OSRA simulator.
chosen at random in each modulus, as explained in Section C.2 of Chapter III. Second, the value of truncated quotient calculated by each algorithm is checked with the correct value for each sample problem. Third, the standard deviation was calculated using the formula given on Figure 49, at the right.

In Figure 50, three operations are needed to find the quotient estimate when $k = 2$ and scaling is employed. The base extension used in this case to restore the $k$th residue digit of $E_{i+1}$ requires no residue operations, because it is a "zero-operation" base extension, as discussed in Chapter II.

Figure 51 is the flowchart of the OSRA simulating program, and is identical to Figure 21 of Chapter III. It is included here for completeness. It is discussed in Section C.2 of Chapter III, where a discussion of the TSRA is given.

The simulating program was checked using the modulus set $\{11, 13, 17\}$ by means of the test problems in Table 7. The test problems used to check the OSRA simulating routine are the same as those used in Section C.2 of Chapter III, and are repeated here for convenience. The number of residue operations required to solve each test problem is also given.

For the OSRA, as explained in Chapter III, the first three test problems were chosen to test each of the 3 possible estimate cases $k = 2$, $k = 2 + 1$ and $k > 2 + 1$. The latter test problem tests if the extra residue addition required to find $\lfloor X/Y \rfloor$ when $E' = 1$ is counted in the operation count. For the RA test problems we use the case $k = 3$ with $Y$ chosen so that:

193
Table 7
Test Problems and Number of Operations Required
for RA and OSRA Simulator

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>RA OPERATIONS</th>
<th>X</th>
<th>Y</th>
<th>OSRA OPERATIONS</th>
</tr>
</thead>
<tbody>
<tr>
<td>2430</td>
<td>1216</td>
<td>9</td>
<td>2425</td>
<td>226</td>
<td>22</td>
</tr>
<tr>
<td>2430</td>
<td>211</td>
<td>28</td>
<td>2425</td>
<td>27</td>
<td>47</td>
</tr>
<tr>
<td>2430</td>
<td>143</td>
<td>15</td>
<td>2425</td>
<td>7</td>
<td>76</td>
</tr>
<tr>
<td>2430</td>
<td>79</td>
<td>37</td>
<td>2425</td>
<td>144</td>
<td>37</td>
</tr>
<tr>
<td>2430</td>
<td>71</td>
<td>24</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2430</td>
<td>11</td>
<td>23</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2430</td>
<td>10</td>
<td>32</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2430</td>
<td>5</td>
<td>48</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2430</td>
<td>1</td>
<td>32</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

1) $\xi=3$ and $|P_3/Y|=1$, $m_3$ and in between;
2) $\xi=2$ and $|P_2/Y|=1$, $m_2$ and in between;
3) $\xi=1$ and $|P_1/Y|=1$, $m_1$ and in between.

Note that the first RA test problem checks if the extra residue addition required when the divisor reciprocal equals one is counted in the operation count. The fourth RA test problem tests if the extra addition is counted when an estimate equals zero.

To determine their statistical properties, the RA and OSRA were simulated using the same five modulus sets used when simulating the SA and OSRA. They are
M1 = { 3, 5, 7, 11, 13, 17, 19, 23, 29, 31} \quad (4.22)
M2 = {23, 29, 31, 37, 41, 43, 47, 53, 59, 61} \quad (4.23)
M3 = {31, 37, 41, 43, 47, 53, 55, 59, 61, 63} \quad (4.24)
M4 = {37, 41, 43, 47, 53, 55, 59, 61, 63, 64} \quad (4.25)
M5 = { 2, 3, 5, 7, 11, 13, 17, 19, 23, 29} \quad (4.26)

These sets were chosen for the reasons discussed in the description of the SA in Section D.2 of Chapter III.

Computer simulations provided a sample set from which average running time and standard deviation about that average were obtained. Probability density information about running time was also generated. For each of the five modulus sets, 40000 sample division problems were simulated, and the following four plots were found:

1) sample average running time as a function of sample size for both algorithms;

2) standard deviation about the mean as a function of sample size for both algorithms;

3) the probability density function of running time for the RA;

4) the probability density function of running time for the OSRA.

These 20 plots are in Figures 52 through 71.

Graphs of sample average running time as a function of sample size contain two plots, one for the RA and one for the OSRA. As seen in the graphs, the number of sample problems required for convergence is large...
Figure 52. RA and OSRA average running time as a function of sample size.
Figure 53. RA and OSRA sample standard deviation as a function of sample size.
Figure 54. RA experimental probability density of running time.
Figure 55. OSRA experimental probability density of running time.
Figure 56. RA and OSRA average running time as a function of sample size.
SAMPLE STANDARD DEVIATION AS A FUNCTION OF SAMPLE SIZE

MODULI \{31, 37, 41, 43, 47, 53, 55, 59, 61, 63\}

\( \times \) RA

\( \triangle \) OSRA

Figure 57. RA and OSRA sample standard deviation as a function of sample size.
Figure 58. RA experimental probability density of running time.
PROBABILITY DENSITY OF RUNNING TIME

MODULI (31, 37, 41, 43, 47, 53, 59, 61, 63)

OSRA

WORST CASE PROBLEM AT 389

Figure 59. OSRA experimental probability density of running time.
Figure 60. RA and OSRA average running time as a function of sample size.
SAMPLE STANDARD DEVIATION AS A FUNCTION OF SAMPLE SIZE

MODULI (37, 41, 43, 47, 53, 55, 59, 61, 63, 64)

X RA
△ OSRA

Figure 61. RA and OSRA sample standard deviation as a function of sample size.
Figure 62. RA experimental probability density of running time.
Figure 63. OSRA experimental probability density of running time.
Figure 64. RA and OSRA average running time as a function of sample size.
Figure 65. RA and OSRA sample standard deviation as a function of sample size.
Figure 66. RA experimental probability density of running time.
Figure 67. OSRA experimental probability density of running time.
Figure 68. RA and OSRA average running time as a function of sample size.
SAMPLE STANDARD DEVIATION AS A FUNCTION OF SAMPLE SIZE

MODULI (3, 5, 7, 11, 13, 17, 19, 23, 29, 31)

× RA
▲ OSRA

Figure 69. RA and OSRA sample standard deviation as a function of sample size.
Figure 70. RA experimental probability density of running time.
Figure 71. OSRA experimental probability density of running time.
and depends on the modulus set. For the set M1, the average running time "settles down" at a value somewhere between 5000 and 10000 samples. For M2, M3 and M4, settling appears at around 10000 samples. For M5, settling occurs around 25000 samples. The standard deviation plots settle down at the values 20000, 20000, 30000 and 15000 for M1, M2, M3 and M5 respectively. The standard deviation for M4 appears to require all 40000 samples for convergence. Furthermore, the average running time and standard deviation plots of the RA and OSRA seem to track each other. This is because of similarities in the algorithms. Note that there are fluctuations in average running time. In particular, notice the pronounced "hump" between 20000 and 30000 samples on the average running time plot for M3. These fluctuations are due to runs of difficult problems, and arise because of peculiarities in the random number generator.

The run time probability density function values decrease quickly for large numbers of operations, as expected. Note that there are clusters of required operations. These clusters correspond to different numbers of estimates. For the RA, at least 50% of all problems are solved in two mixed radix conversions, as would be expected. These are the cases where the numerator is less than the denominator, or when \( k = \alpha \) and \( |P_\alpha/Y| = 1 \).

The average running times and standard deviations for each algorithm were taken from each graph, and are listed in Table 8. They were taken to be the values to which the corresponding graphs converge. These values were determined to the nearest tenth of an operation by
visual means. That is, the value to which the plot appeared to converge was measured to the nearest tenth of an operation.

From Table 8, we see that as with the TSRA and SA, the average running time and standard deviation are largely independent of the modulus set, although the average running time is slightly less for larger moduli. One can also see that the RA uses 2.3 (5%) fewer residue operations per problem than the OSRA, for the modulus sets used. Also, the standard deviation of the RA is 1.5 operations (8%) less than that of the OSRA, for the modulus sets used. We conclude that the RA has a smaller mean running time and standard deviation while, as indicated earlier, using as much as 70% less storage than the OSRA.

Comparing the RA statistics with the corresponding values for the TSRA given in Section C.2 of Chapter III, we see that for the modulus sets M1, M2 and M3 the RA uses 1.4 (3%) fewer residue operations per problem, on the average, than the TSRA. However, the RA has a standard deviation which is 1.8 residue operations (12%) greater than that of the TSRA.

Comparing the RA statistics with the corresponding values for the SA given in Section D.2 of Chapter III, we see that for the modulus sets M1 through M5, the RA uses 3.6 (7%) fewer residue operations per problem, on the average, than the SA. However, the RA has a standard deviation which is .8 residue operations (5%) greater than that of the SA.
Table 8
Mean Running Time and Standard Deviation for the RA and OSRA

<table>
<thead>
<tr>
<th>MODULUS SETS</th>
<th>RA (%)</th>
<th>OSRA (%)</th>
<th>% DIFFERENCE</th>
<th>RA (%)</th>
<th>OSRA (%)</th>
<th>% DIFFERENCE</th>
</tr>
</thead>
<tbody>
<tr>
<td>{3, 5, 7, 11, 13, 17, 19, 23, 29, 31}</td>
<td>47.6</td>
<td>50.35</td>
<td>5 18.1 %</td>
<td>20.2</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td>{37, 41, 43, 47, 53, 59, 61, 63, 64}</td>
<td>47.3</td>
<td>49.2</td>
<td>4 17.1 %</td>
<td>18.2</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>{23, 29, 31, 37, 41, 43, 47, 53, 59, 61}</td>
<td>47.0</td>
<td>49.2</td>
<td>4 16.8 %</td>
<td>17.9</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>{2, 3, 5, 7, 11, 13, 17, 19, 23, 29}</td>
<td>47.8</td>
<td>50.2</td>
<td>5 18.3 %</td>
<td>20.7</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>{31, 37, 41, 43, 47, 53, 59, 61, 63}</td>
<td>47.3</td>
<td>49.2</td>
<td>4 17.1 %</td>
<td>17.9</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>AVERAGE:</td>
<td>47.4</td>
<td>49.7</td>
<td>5 17.5 %</td>
<td>19.0</td>
<td>8</td>
<td></td>
</tr>
</tbody>
</table>

C. Summary of Chapter IV

Chapter IV contains a presentation of a third new RNS division algorithm, the Reciprocal Algorithm (RA), and a comparison of its stored table size and statistics of performance with those of the OSRA.

In Section A there is a detailed presentation of the algorithm. Section B contains a comparison of the stored table sizes, mean running times and standard deviations about the mean of the RA and OSRA, and contains the conclusion that the RA uses an amount of storage proportional to the sum of the moduli (as compared with an amount proportional to the square of the maximum modulus as is required by the OSRA). Furthermore, the RA has 5% better mean running time, and has 8% better standard deviation about the mean than the OSRA.
CHAPTER V
SUMMARY AND CONCLUSIONS

The problem considered in this dissertation is to devise better division algorithms for residue number systems.

Chapter I contains an introduction to residue number systems and a presentation of previous division algorithms. It also contains a statement of the problem and an outline of the dissertation.

Chapter II contains a discussion of the Scaling, Mixed Radix Conversion, Sign Detection and Base Extension Procedures, which are commonly used operations in residue division algorithms. It also contains a detailed presentation of the most attractive residue division algorithm so far published, called the One Sided Rounding Algorithm (OSRA). The OSRA is discussed, summarized and an example of its use is given. The example is followed by a formal statement of the OSRA and a proof that it is correct.

Chapter III contains a presentation of two of the three new residue division algorithms that are presented in this dissertation. The first of these, the Two Sided Rounding Algorithm (TSRA), employs "two-sided" rounding on numerator and denominator to determine quotient estimates. The second, called the Signed Algorithm (SA), is a modification of the TSRA for performing signed division. Both of these algorithms are presented in the same way. First, the algorithm is motivated by a discussion. Then, it is summarized and an example of its use is given.
Subsequently, the algorithm is formally stated and shown to be correct. Finally, the algorithm is compared with the OSRA in three respects. These are stored table size, mean running time and standard deviation of running time about the mean. A formula for stored table size is derived, and computationally derived plots of mean running time and standard deviation about the mean are given. The chapter concludes with a theoretical derivation of a lower bound for the mean running time of the OSRA. Computationally derived results comparing this bound with the actual (also computationally derived) mean running time are also given.

In Chapter IV we find a presentation of the third new algorithm, the Reciprocal Algorithm (RA). This algorithm computes estimates in a brand new way, which uses a suitably defined reciprocal of the divisor. It is presented in exactly the same way as the TSRA and SA.

We draw four conclusions from this study. They are as follows:

1. The TSRA has significantly (17%) smaller standard deviation about the mean running time than the OSRA. Furthermore, it has a (2%) smaller mean running time, while using the same amount of storage.

2. The SA uses at most the same amount of storage as the OSRA. Furthermore, to the extent that they can be compared, the SA has significantly (12%) smaller standard deviation about the mean running time than the OSRA. However, its mean running time is 3% larger than that of the OSRA.
3. The theoretical lower bound for the mean running time of the OSRA, derived in Section E of Chapter III, agrees with the computationally derived values to within 3%, for the modulus sets used in this study.

4. The RA has a 5% smaller mean running time and an 8% smaller standard deviation about the mean than the OSRA. Most importantly, it uses up to 70% less storage than the OSRA.

We have two suggestions for further study. The first is an investigation of multiplicative approaches to residue division. As explained previously, algorithms of this type calculate the truncated quotient by multiplication of the numerator by a suitably defined reciprocal of the divisor. Such an investigation would indicate whether the calculation of the reciprocal (which is essentially a division problem with a fixed numerator) is "easier" than the calculation of the original division problem (in which the numerator is not fixed). One possibility for reciprocal calculation is to use Newton's Method, modified for integer operands.

The second suggestion is to investigate the benefits that parallel processing of operands would provide for residue division. An algorithm which employs simultaneous magnitude comparisons of the numerator with several different multiples of the denominator can be made to have a mean running time which is less than that of the OSRA. However, its standard deviation of running time about the mean must be determined, and the cost of the extra hardware it requires must be weighed against its performance advantages.
Bibliography


[32] [14], p. 162.


[38] [27], p. 172.
[39] [21], pp. 88-89.
[40] [21], pp. 91-94.
[41] [28], pp. 134.
[42] [1], pp. 114-115.
APPENDIX A
Proofs of Lemmas 2 and 3

This appendix contains the proofs of Lemmas 2 and 3.

Lemma 2: For the estimate given in Equations (2.23) and (2.24), viz.,

\[ E_{i+1} = \begin{cases} 
0 & \text{if } k < \ell \\
\frac{x_k}{y_{\ell+1}} & \text{if } k = \ell \\
x_k \frac{m_\ell}{y_{\ell+1}} & \text{if } k = \ell + 1 \\
x_k \frac{m_\ell}{y_{\ell+1}} m_{\ell+1} \cdots m_{k-1} & \text{if } k > \ell + 1
\end{cases} \]

when \( Y \) is not a mixed radix coefficient, or

\[ E_{i+1} = \begin{cases} 
0 & , \, k < \ell \\
x_k & , \, k = \ell \\
x_k m_\ell & , \, k = \ell + 1 \\
x_k m_\ell m_{\ell+1} \cdots m_{k-1} & , \, k > \ell + 1
\end{cases} \]

when \( Y \) is a mixed radix coefficient, we have \( 0 < E_{i+1} < X_i / Y \).

Proof: Clearly, \( E_{i+1} > 0 \) for either type of \( Y \). Now,

Case 1: \( Y \) is not a mixed radix coefficient.
We have,

\[ X_i = x_{k} m_{k-1} \cdots m_1 + x_{k-1} m_{k-2} \cdots m_1 + \cdots + x_1 \]

and

\[ Y = y_{k} m_{k-1} \cdots m_1 + y_{k-1} m_{k-2} \cdots m_1 + \cdots + y_1. \]

Therefore, \( X_i > x_{k} m_{k-1} \cdots m_1, Y < (y_{k+1}) m_{k-1} \cdots m_1 \) and we have

\[
\frac{x_{k} m_{k-1} \cdots m_1}{(y_{k+1}) m_{k-1} \cdots m_1} < \frac{X_i}{Y}.
\]

Now if \( k < \xi \), then

\[ E_{i+1} = 0 < \frac{x_{k} m_{k-1} \cdots m_1}{(y_{\xi+1}) m_{\xi-1} \cdots m_1}. \]

If \( k = \xi \),

\[ E_{i+1} = \frac{x_{\xi}}{y_{\xi+1}} < \frac{x_{\xi} m_{\xi}}{y_{\xi+1}} = \frac{x_{k} m_{k-1} \cdots m_1}{(y_{\xi+1}) m_{\xi-1} \cdots m_1}. \]

If \( k = \xi + 1 \),

\[ E_{i+1} = x_{k} \frac{m_{\xi}}{y_{\xi+1}} < \frac{x_{k} m_{\xi}}{y_{\xi+1}} = \frac{x_{k} m_{k-1} \cdots m_1}{(y_{\xi+1}) m_{\xi-1} \cdots m_1}. \]

If \( k > \xi + 1 \),

\[ E_{i+1} = x_{k} \frac{m_{\xi}}{y_{\xi+1}} m_{\xi+1} \cdots m_{k-1} < \frac{x_{k} m_{\xi} m_{\xi+1} \cdots m_{k-1}}{y_{\xi+1}} = \frac{x_{k} m_{k-1} \cdots m_1}{(y_{\xi+1}) m_{\xi-1} \cdots m_1}. \]

Therefore in each case when \( Y \) is not a mixed radix coefficient,

\[ E_{i+1} < \frac{x_{k} m_{k-1} \cdots m_1}{(y_{\xi+1}) m_{\xi-1} \cdots m_1} < \frac{X_i}{Y}. \]
Case 2: \( Y \) is a mixed radix coefficient. In this case, \( Y = m_{k-1} \ldots m_1 \),
and
\[
\frac{x_k m_{k-1} \ldots m_1}{m_{k-1} \ldots m_1} < \frac{x_i}{Y}.
\]

For \( k < i \), we have
\[
E_{i+1} = 0 < \frac{x_k m_{k-1} \ldots m_1}{m_{k-1} \ldots m_1}.
\]

For \( k = i \),
\[
E_{i+1} = x_k \frac{x_k m_{k-1} \ldots m_1}{m_{k-1} \ldots m_1}.
\]

For \( k = i + 1 \), we have
\[
E_{i+1} = x_k m_k \frac{x_k m_{k-1} \ldots m_1}{m_{k-1} \ldots m_1}.
\]

For \( k > i + 1 \), we have
\[
E_{i+1} = x_k m_k m_{k+1} \ldots m_{i-1} < \frac{x_k m_{k-1} \ldots m_1}{m_{k-1} \ldots m_1}.
\]

Therefore in each case when \( Y \) is a mixed radix coefficient, we have
\[
E_{i+1} < \frac{x_k m_{k-1} \ldots m_1}{m_{k-1} \ldots m_1} < \frac{x_i}{Y}.
\]

Therefore, for both cases 1 and 2, we have
\[
0 < E_{i+1} < \frac{x_i}{Y}.
\]

QED
Lemma 3: For the estimate $E_{i+1}$ given in Equations (2.29) or (2.30), we have $E_{i+1}=0 \implies X_i<2Y$ for all $X_i, Y$.

Proof: We have

$$X_i = x_k m_{k-1} \ldots m_1 + x_{k-1} m_{k-2} \ldots m_1 + \ldots + x_1 < (x_k+1) m_{k-1} \ldots m_1.$$ 

Furthermore,

$$Y = y_k m_{k-1} \ldots m_1 + y_{k-1} m_{k-2} \ldots m_1 + \ldots + y_1,$$

so that $2Y > 2y_k m_{k-1} \ldots m_1$.

Case 1: For $Y$ not a mixed radix coefficient, we have

$$E_{i+1} = 0 \implies k<\ell, \text{ or } k=\ell \text{ and } x_k < y_k.$$ 

If $k<\ell$, then $X_i < 2Y$ clearly.

If $k=\ell$ and $x_k < y_k$, then $x_{k+1} < y_{k+1} < 2y_k$ because $y_k > 1$. So,

$$X_i < (x_k+1) m_{k-1} \ldots m_1 < 2y_k m_{k-1} \ldots m_1 = 2y_k m_{k-1} \ldots m_1 < 2Y.$$ 

Case 2: For $Y$ a mixed radix coefficient, we have

$$E_{i+1} = 0 \implies k<\ell \implies X_i < 2Y,$$ 

clearly.

So, in either case, $X_i < 2Y$.

\text{QED}
APPENDIX B

Proof of Lemma 5

This appendix contains the proof of Lemma 5 from Section A of Chapter III.

Lemma 5: For the estimate used by the TSRA, viz.,

\[ \frac{\hat{x}_k}{\hat{y}_k}, \text{ if } k = \ell \]

\[ \frac{m_\ell}{\hat{y}_k}, \text{ if } k = \ell + 1 \]

\[ \frac{m_\ell}{\hat{y}_k} \cdot m_{\ell+1} \cdot \ldots \cdot m_{k-1}, \text{ if } k > \ell + 1 \]

\[ \frac{2X_1}{Y}. \]

We have \( \hat{x}_1 > Y \Rightarrow 0 < E_{i+1} < \frac{2X_1}{Y}. \)

Proof: By assumption

\[ \hat{x}_1 = \sum_{j=1}^{k-1} m_j + x_{k-1} m_j + \ldots + x_1 \]

and

\[ \hat{y}_1 = \sum_{j=1}^{\ell-1} m_j + y_{\ell-1} m_j + \ldots + y_1. \]

We will first show that \( E_{i+1} > 0. \) Clearly, \( E_{i+1} > 0 \) in all cases, and \( E_{i+1} > 0 \) when \( \hat{y}_1 = m_1. \)
When \( \tilde{y}_k < m_n^{-1} \), we have the two cases \( k=\ell \) and \( k>\ell+1 \) (the case \( k<\ell \) cannot occur because \( \bar{x}_i > Y \) by assumption).

If \( k=\ell \), then

\[
E_{i+1} = \left[ \begin{array}{c} \frac{x_k}{\tilde{y}_k} \end{array} \right] \quad \text{or} \quad \left[ \begin{array}{c} \frac{x_{k+1}}{\tilde{y}_{k+1}} \end{array} \right], \quad \text{if } x_k < m_n^{-1}: \quad \text{Case 1}
\]

\[
E_{i+1} = \left[ \begin{array}{c} \frac{m_n-1}{\tilde{y}_k} \end{array} \right] \quad \text{or} \quad \left[ \begin{array}{c} \frac{m_n-1}{\tilde{y}_{k+1}} \end{array} \right], \quad \text{if } x_k = m_n^{-1}: \quad \text{Case 2}
\]

We have \( E_{i+1} > 0 \) in Case 1 because \( x_k > \tilde{y}_k \) (since \( \bar{x}_i > Y \) and \( k=\ell \)).

We have \( E_{i+1} > 0 \) in Case 2 because \( \tilde{y}_k < m_n^{-1} \). Therefore, \( E_{i+1} > 0 \) when \( k=\ell \).

If \( k>\ell+1 \), \( E_{i+1} \) is the product of terms \( \frac{m_n}{\tilde{y}_k} \), \( \tilde{x}_k \), and (for \( k>\ell+1 \)) \( \frac{m_n^{-1}}{\tilde{y}_{k+1}} \). Each of these terms is nonzero (the first because \( \tilde{y}_k < m_n^{-1} \)) and therefore \( E_{i+1} > 0 \) when \( k>\ell+1 \).

Therefore, in all cases, \( E_{i+1} > 0 \).

We will now show that \( E_{i+1} < 2\tilde{x}_i/Y \), first in the case \( \tilde{y}_k = m_n \), and then in the case \( \tilde{y}_k < m_n^{-1} \).

For \( \tilde{y}_k = m_n \), we want to show \( E_{i+1} = 1 < 2\tilde{x}_i/Y \). We have \( \tilde{y}_k = m_n \Rightarrow \ell=n \) and \( y_k = m_n^{-1} \). But \( \bar{x}_i > Y \). Therefore, \( k=n \) and \( x_k = m_n^{-1} \), and so \( \bar{x}_i > (m_n^{-1})m_n^{-1} \cdots m_1 \). Furthermore, \( Y < M \). Therefore,

\[
\frac{(m_n-1)m_{n-1} \cdots m_1}{M} < \frac{\bar{x}_i}{Y}, \quad \text{and so} \quad \frac{2(m_n-1)m_{n-1} \cdots m_1}{M} < \frac{2\bar{x}_i}{Y}.
\]

But

231
\[ \frac{2(m_n-1)m_{n-1} \cdots m_1}{m} = \frac{2(m_n-1)}{m_n} = 2 - \frac{2}{m_n} > 1 \text{ (because } m_n > 2). \]

Therefore, \(1 < \frac{2x_i}{y}\), and so \(E_{i+1} < \frac{2x_i}{y}\) in case \(y \geq m_n\).

When \(y < m_n - 1\), we have \(\bar{x}_i > x_k m_{k-1} \cdots m_1\), so that \(2\bar{x}_i > 2x_k m_{k-1} \cdots m_1\).

Also, \(Y < (y+1)m_{k-1} \cdots m_1\), so that

\[ \frac{2x_k m_{k-1} \cdots m_1}{(y+1)m_{k-1} \cdots m_1} < \frac{2x_i}{y}. \]

We will show \(E_{i+1} < \frac{2x_k}{(y+1)}\) in each of the cases \(k=2\), \(k=2+1\), and \(k>2+1\).

For \(k=2\): we must show that \(E_{i+1} < \frac{2x_k}{(y+1)}\). We have

\[ E_{i+1} = \begin{vmatrix} \frac{x_k}{y} \\ \frac{x_k+1}{y+1} \end{vmatrix} = \begin{cases} \frac{x_k}{y}, & \text{or } \frac{x_k+1}{y+1}, & \text{if } x_k < m_n - 1: \text{ Case 1} \\ m_n - 1, & \text{or } \frac{m_n - 1}{y+1}, & \text{if } x_k = m_n - 1: \text{ Case 2} \end{cases} \]

For Case 1:

\[ E_{i+1} = \begin{vmatrix} \frac{x_k}{y} \\ \frac{x_k+1}{y+1} \end{vmatrix} < \frac{x_k}{y} = \frac{2x_k}{2y} < \frac{2x_k}{y+1}, \]

the last inequality being true because \(y > 1\), and so \(y+1 < 2y\). Also,

\[ E_{i+1} = \begin{vmatrix} \frac{x_k+1}{y} \\ \frac{x_k+1}{y+1} \end{vmatrix} < \frac{x_k+1}{y+1} < \frac{2x_k}{y+1}, \]

the last inequality being true because \(x_k > 1\), and so \(2x_k > x_k + 1\).

Therefore, for Case 1,
For Case 2: In this case, \( x_k = m_n - 1 \), so we must show

\[
E_{i+1} \leq \frac{2(m_n - 1)}{y^* + 1}.
\]

We have

\[
E_{i+1} = \left| \frac{m_n - 1}{y^*} \right| \leq \frac{m_n - 1}{y^*} = \frac{2(m_n - 1)}{2y^*} < \frac{2(m_n - 1)}{y^* + 1}.
\]

Also,

\[
E_{i+1} = \left| \frac{m_n - 1}{y^* + 1} \right| < \frac{m_n - 1}{y^* + 1} < \frac{2(m_n - 1)}{y^* + 1}.
\]

Therefore, for Case 2, we have that

\[
E_{i+1} < \frac{2(m_n - 1)}{y^* + 1}.
\]

Therefore, for \( k = x \),

\[
E_{i+1} < \frac{2x_k}{(y^* + 1)}.
\]

For \( k = x + 1 \): We must show that

\[
E_{i+1} < \frac{2x_k m_k}{y^* + 1}.
\]

We have
For Case 1:

\[ E_{i+1} = x_k \left| \frac{m_2}{y_k} \right| \leq \frac{x_k m_2}{y_k} = \frac{2x_k m_2}{y_k} < \frac{2x_k m_2}{y_k + 1} . \]

Also,

\[ E_{i+1} = (x_k + 1) \left| \frac{m_2}{y_k + 1} \right| < \frac{(x_k + 1)m_2}{y_k + 1} < \frac{2x_k m_2}{y_k + 1} . \]

Therefore, for Case 1,

\[ E_{i+1} < \frac{2x_k m_2}{y_k + 1} . \]

For Case 2: In this case, \( x_k = m_n - 1 \), so we must show

\[ E_{i+1} < \frac{2(m_n - 1)m_2}{y_k + 1} . \]

We have

\[ E_{i+1} = (m_n - 1) \left| \frac{m_2}{y_k} \right| < \frac{(m_n - 1)m_2}{y_k} = \frac{2(m_n - 1)m_2}{2y_k} < \frac{2(m_n - 1)m_2}{y_k + 1} . \]

Also,

\[ E_{i+1} = (m_n - 1) \left| \frac{m_2}{y_k + 1} \right| < \frac{(m_n - 1)m_2}{y_k + 1} < \frac{2(m_n - 1)m_2}{y_k + 1} . \]
Therefore, for Case 2,
\[ E_{i+1} < \frac{2(m_n-1)m_\varphi}{y_\varphi + 1} \cdot \]

Therefore, when \( k=\varepsilon+1 \),
\[ E_{i+1} < \frac{2x_km_\varphi}{y_\varphi + 1} \cdot \]

For \( k>\varepsilon+1 \): We must show that
\[ E_{i+1} < \frac{2x_km_{k-1} \ldots m_\varphi}{y_\varphi + 1} \cdot \]

We have
\[ E_{i+1} = x_k \left| \frac{m_\varphi}{y_\varphi} \right| \frac{m_\varphi+1 \ldots m_{k-1}}{m_\varphi+1} \]

or
\[ (x_{k+1}) \left| \frac{m_\varphi}{y_\varphi+1} \right| \frac{m_\varphi+1 \ldots m_{k-1}}{m_\varphi+1}, \text{ if } x_k < m-1: \]

\[ \text{Case 1} \]

\[ (m_{n-1}) \left| \frac{m_\varphi}{y_\varphi} \right| \frac{m_\varphi+1 \ldots m_{k-1}}{m_\varphi+1} \]

or
\[ (m_{n-1}) \left| \frac{m_\varphi}{y_\varphi+1} \right| \frac{m_\varphi+1 \ldots m_{k-1}}{m_\varphi+1} \text{, if } x_k = m-1: \]

\[ \text{Case 2} \]
For Case 1:

\[
E_{i+1} = x_k \left\lfloor \frac{m_2}{y_2} \right\rfloor m_{k+1} \cdots m_{k-1} < \frac{x_k m_2 m_{k+1} \cdots m_{k-1}}{y_2}
\]

\[
= \frac{2x_k m_2 m_{k+1} \cdots m_{k-1}}{2y_k} < \frac{2x_k m_{k-1} \cdots m_2}{y_k + 1}.
\]

Also,

\[
E_{i+1} = (x_{k+1}) \left\lfloor \frac{m_2}{y_2+1} \right\rfloor m_{k+1} \cdots m_{k-1} < \frac{(x_{k+1}) m_2 m_{k+1} \cdots m_{k-1}}{y_2 + 1}
\]

\[
= \frac{2x_{k+1} m_{k-1} \cdots m_2}{y_k + 1}.
\]

Therefore, for Case 1,

\[
E_{i+1} < \frac{2x_k m_{k-1} \cdots m_2}{y_k + 1}.
\]

For Case 2: In this case \( x_k = m_n - 1 \), so we must show

\[
E_{i+1} < \frac{2(m_n - 1) m_{k-1} \cdots m_2}{y_k + 1}.
\]

We have

\[
E_{i+1} = (m_n - 1) \left\lfloor \frac{m_2}{y_2} \right\rfloor m_{k+1} \cdots m_{k-1} < \frac{(m_n - 1) m_2 m_{k+1} \cdots m_{k-1}}{y_2}
\]

\[
= \frac{2(m_n - 1) m_{k+1} \cdots m_{k-1}}{2y_k} < \frac{2(m_n - 1) m_{k-1} \cdots m_2}{y_k + 1}.
\]

236
Also,

\[ E_{i+1} = \frac{(m_n - 1) m_{2} \cdots m_{k-1}}{y_{2} + 1} \cdot \frac{m_{2} + \cdots + m_{k-1} - 1}{y_{k} + 1} < \frac{(m_n - 1) m_{2} \cdots m_{k-1}}{y_{2} + 1} \cdot \frac{2(m_n - 1) m_{k-1} \cdots m_{2}}{y_{k} + 1} \]

Therefore, for Case 2,

\[ E_{i+1} < \frac{2(m_n - 1) m_{k-1} \cdots m_{2}}{y_{2} + 1} \]

Therefore, for \( k > i + 1 \),

\[ E_{i+1} < \frac{2x_{k} m_{k-1} \cdots m_{2}}{y_{k} + 1} \]

Therefore, for \( \tilde{y}_{k} < m_n - 1 \), we have

\[ E_{i+1} < \frac{2x_{i}}{\tilde{y}_{i}} \]

QED
APPENDIX C

Proof of Lemma 7

This appendix contains the proof of Lemma 7, contained in Section B of Chapter III.

Lemma 7: For the $E_{i+1}$ as defined in Equation (3.49), viz.,

$$E_{i+1} = \left\{ \begin{array}{ll}
\frac{x_k}{\tilde{y}_2}, & \text{if } k = 2 \\
\frac{m_k}{\tilde{y}_2}, & \text{if } k = \ell + 1 \\
\tilde{x}_k \frac{m_k}{\tilde{y}_2} m_{k+1} \cdots m_{k-1}, & \text{if } k > \ell + 1
\end{array} \right. \quad (3.56)$$

where

$$\tilde{y}_2 = \left\{ \begin{array}{ll}
y_1, & \text{if } \ell = 1 \\
y_\ell, & \text{if } \ell \neq 1 \text{ and } y_{\ell-1} < \frac{m_{\ell-1}}{2} \\
y_{\ell+1}, & \text{if } \ell \neq 1 \text{ and } y_{\ell-1} > \frac{m_{\ell-1}}{2}
\end{array} \right.$$

and

$$\tilde{x}_k = \left\{ \begin{array}{ll}
x_k, & \text{if } \tilde{y}_2 = y_\ell \\
x_{k+1}, & \text{if } \tilde{y}_2 = y_{\ell+1}
\end{array} \right.$$

we have $\tilde{x}_1 > \tilde{y} \Rightarrow 0 < E_{i+1} < \frac{2x_1}{\tilde{y}}$. 

238
Proof: For convenience, we repeat the mixed radix expressions for $\bar{x}_i$ and $\bar{y}$. They are

$$\bar{x}_i = x_k m_k - 1 \cdots m_1 + x_{k-1} m_{k-2} \cdots m_1 + \cdots + x_1$$

and

$$\bar{y} = y_k m_k - 1 \cdots m_1 + y_{k-1} m_{k-2} \cdots m_1 + \cdots + y_1.$$

We will first show that $E_{i+1} > 0$, and then show that $E_{i+1} < 2\bar{x}_i / \bar{y}$.

Clearly, $E_{i+1} > 0$. Furthermore, $k > \lambda$ because $\bar{x}_i > \bar{y}$. If $k = \lambda$, then from the definition of the estimate we have

$$E_{i+1} = \begin{cases} \frac{x_k}{y_\lambda}, & \text{if } \bar{y}_\lambda = y_\lambda \\ \frac{x_{k+1}}{y_\lambda + 1}, & \text{if } \bar{y}_\lambda < y_\lambda + 1. \end{cases}$$

But $x_k > y_\lambda$ because $\bar{x}_i > \bar{y}$ and $k = \lambda$. Therefore, $E_{i+1} > 0$ when $k = \lambda$.

If $k > \lambda + 1$, then $E_{i+1} > 0$, because $x_k > 1$ (by definition), and $|m_\lambda / \bar{y}_\lambda| > 1$ (because $\bar{y}_\lambda < m_\lambda$). Therefore, $E_{i+1} > 0$ when $\bar{x}_i > \bar{y}$.

Now we will show that $\bar{x}_i > \bar{y} \Rightarrow E_{i+1} < 2\bar{x}_i / \bar{y}$. We have

$$\bar{x}_i > x_k m_{k-1} \cdots m_1,$$

so that

$$2\bar{x}_i > 2x_k m_{k-1} \cdots m_1.$$

Also,
\(\bar{y} < (y_{\bar{z}} + 1)m_{\bar{z}-1} \cdots m_1\).

Therefore,
\[
\frac{2x_k m_{k-1} \cdots m_1}{(y_{\bar{z}}+1)m_{\bar{z}-1} \cdots m_1} < \frac{2\bar{x}_i}{\bar{y}}.
\]

For \(k=\bar{z}\): We have, by the definition of \(E_{i+1}\),
\[
E_{i+1} = \begin{cases} 
\left| \frac{x_k}{y_{\bar{z}}} \right|, & \text{if } \bar{y}_{\bar{z}} = y_{\bar{z}} \\
\left| \frac{x_k + 1}{y_{\bar{z}} + 1} \right|, & \text{if } \bar{y}_{\bar{z}} = y_{\bar{z}} + 1.
\end{cases}
\]

But \(\bar{x}_i > Y\), so therefore \(x_k > y_{\bar{z}}\), and so
\[
\frac{x_k}{y_{\bar{z}}} > \frac{x_k + 1}{y_{\bar{z}} + 1}.
\]

Therefore,
\[
\left| \frac{x_k}{y_{\bar{z}}} \right| > \left| \frac{x_k + 1}{y_{\bar{z}} + 1} \right|.
\]

But
\[
\frac{x_k}{y_{\bar{z}}} > \left| \frac{x_k}{y_{\bar{z}}} \right|.
\]

Therefore,
\[
\frac{x_k}{y_{\bar{z}}} > E_{i+1}.
\]

But

240
because \( y_k > 1 \). Therefore,

\[
E_{i+1} < \frac{x_k}{y_k} < \frac{2x_k}{y_k + 1} < \frac{2x_i}{y} \quad \text{for } k = \ell.
\]

For \( k = \ell + 1 \): Here we must show

\[
E_{i+1} < \frac{2x_k m_\ell}{y_k^2 + 1}. \quad \text{We have}
\]

\[
E_{i+1} = \left\{ \begin{array}{ll}
\frac{x_k m_\ell}{y_k}, & \text{if } \tilde{y}_k = y_k \\
\frac{(x_k + 1) m_\ell}{y_k + 1}, & \text{if } \tilde{y}_k = y_k + 1.
\end{array} \right.
\]

We have

\[
E_{i+1} = \frac{x_k m_\ell}{y_k} < \frac{2x_k m_\ell}{2y_k} < \frac{2x_k m_\ell}{y_k + 1},
\]

the last inequality being true because \( y_k > 1 \). Also,

\[
E_{i+1} = \frac{(x_k + 1) m_\ell}{y_k + 1} < \frac{2x_k m_\ell}{y_k + 1},
\]

the last inequality being true because \( x_k > 1 \). Therefore, when \( k = \ell + 1 \),

\[
E_{i+1} < \frac{2x_k m_\ell}{y_k^2 + 1}.
\]

For \( k > \ell + 1 \): Here we must show
We have

\[ E_{i+1} < \frac{2x_k m_{k-1} \cdots m_{k-1}}{y_{\tilde{x}} + 1} \cdot \]

We have

\[ E_{i+1} = \begin{cases} 
  x_k \left| \frac{m_{\tilde{x}}}{y_{\tilde{x}}} \right| m_{\tilde{x}+1} \cdots m_{k-1}, & \text{if } \tilde{y}_{\tilde{x}} = y_{\tilde{x}} \\
  (x_k + 1) \left| \frac{m_{\tilde{x}}}{y_{\tilde{x}} + 1} \right| m_{\tilde{x}+1} \cdots m_{k-1}, & \text{if } \tilde{y}_{\tilde{x}} = y_{\tilde{x}} + 1.
\end{cases} \]

We have

\[ E_{i+1} = x_k \left| \frac{m_{\tilde{x}}}{y_{\tilde{x}}} \right| m_{\tilde{x}+1} \cdots m_{k-1} < \frac{x_k m_{\tilde{x}+1} \cdots m_{k-1}}{y_{\tilde{x}}} \]

\[ = \frac{2x_k m_{\tilde{x}+1} \cdots m_{k-1}}{2y_{\tilde{x}}} < \frac{2x_k m_{\tilde{x}} \cdots m_{k-1}}{y_{\tilde{x}} + 1}, \]

the last inequality being true because \( y_{\tilde{x}} > 1 \). Also,

\[ E_{i+1} = (x_k + 1) \left| \frac{m_{\tilde{x}}}{y_{\tilde{x}} + 1} \right| m_{\tilde{x}+1} \cdots m_{k-1} < \frac{(x_k + 1) m_{\tilde{x}} m_{\tilde{x}+1} \cdots m_{k-1}}{y_{\tilde{x}} + 1} \]

\[ < \frac{2x_k m_{\tilde{x}} \cdots m_{k-1}}{y_{\tilde{x}} + 1}, \]

the last inequality being true because \( x_k > 1 \). Therefore, when \( k > \tilde{x} + 1 \),

\[ E_{i+1} < \frac{2x_k m_{\tilde{x}} \cdots m_{k-1}}{y_{\tilde{x}} + 1}. \]

Therefore, in all cases,

\[ E_{i+1} < \frac{2x_k m_{k-1} \cdots m_1}{(y_{\tilde{x}} + 1)m_{\tilde{x}-1} \cdots m_1} < \frac{2x_1}{\tilde{y}}. \]
Therefore, finally, we have

\[ \bar{X}_i > \bar{Y} \Rightarrow 0 < E_{i+1} < \frac{2X_i}{Y} \]

QED
APPENDIX D
Simulator Program for the TSRA and OSRA

APPENDIX D: SIMULATOR PROGRAM FOR THE TSRA AND OSRA

Note: In this program, the OSRA is sometimes referred to as the Baner-I Algorithm.

This program simulates the TSRA and OSRA NNS division algorithms.

Input for the program is file FOR002.DAT. Output is file FOR004.DAT.

Variable names are as follows:

NUM: Numerator in decimal
DEN: Denominator in decimal
Q: Quotient in decimal
MUCODE(I): The mixed radix coefficients in decimal, in increasing order
SYSMOD(I): The system modulus, in increasing order
NUMMOD: The number of moduli
SYSINV(I,J): The multiplicative inverse of I, modulo SYSMOD(J)
NUMRES(I): The Ith residue digit of the numerator
DENRES(I): The Ith residue digit of the denominator
QUOT(I): The Ith residue digit of the quotient
MUCODE(I): The mixed radix digits of the denominator, in increasing order
MCNUMRES(I): The mixed radix digits of the numerator, in increasing order
ESTRES(I): The Ith residue digit of the quotient estimate in some iteration
ESTCAL1: Flag which exceeds zero only when at least one quotient estimate
has been made
SCATER(I): The number of division problems which required exactly I residue
operations to solve using the TSRA
SCBAN1(I): Same as SCATER(I), but for the OSRA
NUMBAN(I): The numerator residues for the OSRA
DENBAN(I): The denominator residues for the OSRA
BANQ1: The quotient, in decimal, in the OSRA
NUMSET1: The number of modulus sets to be simulated
NUMBAN1: The number of division problems to be simulated within a modulus set
SAVER1: The average running time for the TSRA, found from a
previous run of the simulator. This number is used only for
variance calculations
SAVER2: The average running time for the OSRA, found from a
previous run of the simulator. This number is used only for
variance calculations
SAHDIF: The sample average difference in running times, found as OSRA
time minus TSRA time, from a previous run of the simulator.
NEVERY: This number is used for variance calculations only

Data input to the simulator must be as follows:

Line 1: The number of modulus sets to be simulated
Line 2: The number of division problems to be simulated for the first
modulus set
Line 3: The number of moduli in the set
Line 4: The moduli in the set, ordered least first, to greatest last
Line 5: All moduli must be odd
Line 6: The following three real numbers, in this order and separated by
commas:
   Average running time for the TSRA. This is needed
   only for the calculation of variance
   Average running time for the OSRA, needed only
   for the calculation of variance

244
Average difference of running times, DSRA minus TSRA time
This is needed only for calculation of variance

Lines 2 thru 6 must be repeated, in order, for each modulus set to be
simulated

SUBROUTINES:
CONINT, QUOEST, CONPINT, BANNER

REAL*8 N,NUM,NUMD,NUMCOE(20)
COMMON SYM0C0(20),NUMMOD,SPM0H(0:20),NUMCOE
REAL NUMREC(20),DENREC(20),QUTC0(20),MACGEN20),MACNUM(20),ESTREC(20)
INTEGER ESTCAL
REAL SCATER(100),NUMRAN(20),DENRAN(20),SCABAR(500)
REAL*8 BANQ
OPEN (UNIT=2,FILE="STUDENT_DISK:EG.CHREN3FOR02.DAY",STATUS="OLD")
READ(2,6) NUMSET

C Initialize the modulus set counter and input data
C
I=1
DO WHILE(J.LE.NUMSET)
READ(2,6) NUMSAN
READ(2,6) (SYM0C0(I),I=1,NUMMOD)
READ(2,6) BANR,SCABAR,SCABAR
READ(2,6) NEVERY
C
C Output number of data points and size of account arrays
C
NSIZE=500
NUNPTS=NUMSAN/NEVERY
WRITE(*,4)NUNPTS,NSIZE,NEVERY
C
C Initialize account arrays
C
DO 10 I=1,500
SCATER(I)=0.
SCABAR(I)=0.
10 CONTINUE
C
C Compute dynamic range for this particular modulus set
C
M=1000
DO 10 I=1,NUMMOD
PRINT*(15,5)SYM0C0(I)
10 CONTINUE
C
C Load table of inverses modulo the system modulus. This code will compute
C a value of zero for the inverse of integers that are not relatively prime

245
C to the modulus
C
DO 20 I=1,NUNMOD
  K=SYSDOD(I)-1.
DO 30 J=1,K
DO 40 L=1,K
  DO WHILE(J≦I,JINT(SYSDOD(I))≦EQ.1)
    SYSDVD(J,I)=L
    GO TO 10
END DO
40 CONTINUE
30 CONTINUE
20 CONTINUE
C Load the array of mixed radix coefficients
C
NRCCDE(1)=1.000
J=2
DO WHILE(J.LE.NUNMOD)
  NRCCDE(J)=NRCCDE(J-1)*1000(SYSDOD(J-1))
  J=J+1
END DO
C Set the random number generator seed
C
ISEED=45069
C Initialize I, which is the sample counter, and also initialize other
C simulator variables. In particular, OPCNT which is the aggregate
C operation count, SMSOT which is the aggregate count for the BIA,
C VARMNT which is the aggregate square deviation from the mean for the
C TSA, and SMSVAR which is the same thing for the BIA.
C
I=1
OPCNT=0.
VARMNT=0.
SMSOT=0.
SMSVAR=0.
DIFVAR=0.
DIFVAR=0.
C Done enough samples?
C
DO WHILE(I.LE.NUNSAM)
C Initialize operation counter for this particular division problem
C
CASCNT=0.
C Randomly select a numerator and denominator in residue. Copy these
C for later use by OSM
C
65 J=1
DO WHILE(J.LE.NUNMOD)
  NUMRES(J)=INT(SYSDOD(J)*RANISEED))
  DENRES(J)=INT(SYSDOD(J)*RANISEED))
  J=J+1
END DO
J=1
DO WHILE(J.LE.NUNMOD)
NUMBAN(J)=NUMRES(J)
DENBAN(J)=DENRES(J)
J=J+1
END DO
C Convert to integer form, for various purposes which are more easily done
C in decimals. Test to see if the denominator happens to be zero. If so,
C go get another denominator
CALL CONINT(NUMRES,M,NUN)
CALL CONINT(DENRES,M,DEN)
IF(DEN.LT.0.000000003M) THEN
  WRITE(*,*,'(A,0)''ZERO HIT''
  GO TO 45
ELSE
  END IF
C Initialize the quotient
J=1
DO WHILE(J.LE.NUMMOD)
  QUOT(J)=0.
  J=J+1
END DO
C Initialize the estimate counter
ESTCAL=0
C Convert the denominator to mixed radix form, and bump aggregate and problem
C counts
CALL CONMIX(DENRES,M,COUNT,TWODEN)
OPCONT=OPCONT+COUNT
CASCNT=CASCNT+COUNT
C Convert the numerator to mixed radix form, and bump counts
60 CALL CONMIX(NUMRES,M,COUNT,TWOTST)
OPCONT=OPCONT+COUNT
CASCNT=CASCNT+COUNT
C If this is iteration number zero, then XOR gate output is correct.
C Otherwise, check if XOR gate output equals the computed modulo-two value of
C the new partial numerator. If it doesn't, then the sign of the numerator is
C negative, and its mixed radix digits must be flipped. The flip requires one
C residue operation
IF(ESTCAL.EQ.0.) THEN
  TWONUM=TWOTST
  SIGN=1.
ELSE
  IF(TWOTST.EQ.TWONUM) THEN
    SIGN=1.
  ELSE
    SIGN=-1.
  END IF
  CALL FLIP(NUMRES)
  OPCONT=OPCONT+1.
  CASCNT=CASCNT+1.
END IF
END IF

If numerator is less than denominator, check that the computed quotient is
correct. Otherwise, continue

J=NUMMOD
DO WHILE(J.GE.1)
IF(NUM(J).LT.DEN(J))THEN
    GO TO 70
ELSE
IF(NUM(J).GT.DEN(J))THEN
    GO TO 90
ELSE
    END IF
END IF

Go get a quotient estimate, and bump the counts.

CALL QUOTESTNUM,NUMDEN,COUNT,ESTTWQ,ESTCAL
OPCONT=OPCONT+COUNT
CASCNT=CASCNT+COUNT

No longer first call

ESTCAL=ESTCAL+1

If the sign is positive, add the estimate to the running sum. Else, subtract
it. Numerator is treated the opposite way. Negative one is (SYRMOD(J)-1.)
in module SYRMOD(J). Bump the counts by 2 operations needed for the new
nominator, and 1 operation for the new running quotient

IF(SIGN.EQ.1.)THEN
    J=1
    DO WHILE(J.LE.NUNMOD)
        QUOT(J)=ANDD(QUOT(J)+ESTRES(J),SYRMOD(J))
        NUMRES(J)=AMOD(NUMRES(J)+(SYRMOD(J)-1.)*ESTRES(J)+DENRES(J),SYRMOD(J))
        J=J+1
    END DO
ELSE
    J=1
    DO WHILE(J.LE.NUNMOD)
        QUOT(J)=ANDD(QUOT(J)+(SYRMOD(J)-1.)*ESTRES(J),SYRMOD(J))
        NUMRES(J)=AMOD(NUMRES(J)+ESTRES(J),SYRMOD(J))
        J=J+1
    END DO
END IF

OPCONT=OPCONT+3
CASCNT=CASCNT+3

Compute the modulo-2 value of the new numerator, and branch to mixed radix
convert it. This requires no extra residue operations because it is done
in parallel with the other moduli

TNUM=AMOD(TNUM+ESTTWQ,TWODEN2)
GO TO 90

This is the beginning of the quotient adjustment code, executed when the
absolute value of the i-th partial numerator is less than the denominator.
If the sign is negative, the quotient sum is decremented by one, and count.

**70** IF (SIGN(N) .LT. 0) THEN

\[ Q(j) = AMOD(QUOT(j) + (SYSMDD(j) - 1), SYSMDD(j)) \]

ELSE END IF

ELSE END IF

Check the quotient. If it is correct, bump the operation count matrix, and
simulate DSRA on the same operands. Otherwise, print error message and
stop.

**72** CALL CONEXT(QUOT, M, Q)

IF (MOD(M, DEN) .NE. 0) THEN

WRITE(F, 2) "ERROR Q", "Q" = Q, "M" = M

ELSE END IF

SCATTER(CASCNT) = SCATTER(CASCNT) + 1

CALL BANERJ(NUMBAN, DENBAN, BANQ, BANCNT)

Check the DSRA quotient

IF (BANQ .EQ. 0) THEN

WRITE(F, 2) "ERROR BANERJ", "BANQ" = BANQ, "I" = I

ELSE END IF

Bump the operation count matrix for the Banerji algorithm, and calculate
various statistics for both algorithms.

SCABAN(BANCNT) = SCABAN(BANCNT) + 1

BANNT = BANNT + BANCNT

BANVAR = BANVAR + (BANCNT - SAMBN) * B

DIFF = BANCNT - CASCNT

DIFFM = DIFFM + I

DIFFVAR = DIFFVAR + (DIFF - SAMDIFF) * I

VARTOT = VARTOT + (CASCNT - SAMPL) * I

IF (MOD(J, NEVREV) .EQ. 0) THEN

OUTONE = OUTONE + 1

OUTRE = OPCODE + I

OUTFOR = BANTOT + I

Need an unbiased estimate of the variance, so use 1 - 1 as denominator

OUTIV = VARTOT / (I - 1)

OUTSIX = BANVAR / (I - 1)

OUTI = DIFFVAR / (I - 1)

WRITE(F, 2) OUTRE, OUTFOR, OUTONE, OUTIF1, OUTIX

ELSE END IF
This subroutine converts the mixed radix digits of a negative operand, stored in RESIDU, to the mixed radix digits of the absolute value of the operand, returned in RESIDU.

```
SUBROUTINE RLIRCRESIDU
COMMON SYSMOD(20),NUMMOD,SYSINV(96,20),ARCCOE
REAL*8 ARCCOE(20)
REAL RESIDUC(20)

Find the least significant mixed radix digit

LEESIG=0
J=1
DO WHILE(J.LE.NUMMOD)
  IF(RESIDUC(J).NE.0.0) THEN
    LEESIG=J
    GO TO 10
  ELSE
  END IF
  J=J+1
END DO

Compute the absolute value

IFC(LEESIG.NE.0.0) THEN
J=1
DO WHILE(J.LE.NUMMOD)
  RESIDUC(J)=ARCCO(T(SYSMOD(J)-1.0+SYSMOD(J)-1.0)*RESIDUC(J),SYSMOD(J))
  J=J+1
END DO
J=1
DO WHILE(J.LT.LEESIG)
  RESIDUC(J)=0.
  J=J+1
END DO
ELSE
  RESIDUC(LEESIG)=ARCCO(T(RESIDUC(LEESIG)+1.0),SYSMOD(LEESIG))
END IF
RETURN
END
```

This subroutine converts the residue operand RESIDU to its decimal equivalent, returned in I.
SUBROUTINE CONINT(RESIDU,N,X)
COMMON SYSMOD(20),NUMMOD,SYSTIN(96,20),MCCOE
REAL*8 MCCOE(20),N,M
DIMENSION RESIDU(20),SCMOD(20),CPYRES(20)

MAKE a copy of the residue input, used to restore the input when returning

DO 5 I=1,NUMMOD
CPYRES(I)=RESIDU(I)
CONTINUE
5

Do a mixed radix conversion, and sum the mixed radix terms

CALL CONXIX(RESIDU,SCMOD,COUNT,NUMMOD)
DO 10 I=1,NUMMOD
=K+SCMOD(I)+MCCOE(I)
10 CONTINUE

Restore the input and return

DO 15 I=1,NUMMOD
RESIDU(I)=CPYRES(I)
15 CONTINUE
RETURN
END

This subroutine computes a quotient estimate for the mixed radix digits
of numerator and denominator, passed in MRCNUM and MRCDEN, respectively.
The quotient estimate residues are returned in QUORES, and the modulo-2
value of the quotient estimate is returned in QOUTWO. The number of
operations required is returned in COUNT.

SUBROUTINE QOUTEST(MRCNUM,MRCDEN,QUORES,COUNT,QOUTWO,ESTCAL)
COMMON SYSMOD(20),NUMMOD,SYSTIN(96,20),MCCOE
REAL*8 MCCOE(20)
DIMENSION QUORES(20)
REAL MRCNUM(20),MRCDEN(20),MNDCO
REALQ QUOT,SUB
INTEGER ESTCAL
COUNT=0.

Find the most significant nonzero digit of the denominator

I=NUMMOD
DO WHILE(I.GE.1)
IF(MRCDEN(I).NE.0 THEN
MDCDEN=I
GO TO 10
ELSE
END IF
END DO

Find the most significant nonzero digit of the numerator

10
I=NUMMOD
DO WHILE(I.GE.1)
IF(MRCNUM(I).NE.0 THEN

251
C Perform the unsigned algorithm rounding. Find the values x-tilda and y-tilda.
C
20 IF(MSDDEN.GT.1.1) THEN
   IF(MSDDEN.GT.1.) LE. AMT(SYSTMOD(MSDDEN-1)/2.) THEN
      DRNDCO=MSD DEN(MSDDEN)
      RND = ROUND(MSDNUM)
   ELSE
      DRNDCO=MSD DEN(MSDDEN)+1.
   END IF
C Rounding up on y-tilda only counts as an operation for the first estimate
C
   IF(ESTCAL.EQ.0) THEN
      COUNT=COUNT+1.
   ELSE
   END IF
C Find x-tilda
C
   IF(MSDNUM(MSDNUM).LT.(SYSTMOD(MSDDEN)-1.)) THEN
      RND=ROUND(MSDNUM)
      COUNT=COUNT+1.
   ELSE
      RND=MSDNUM+1.
   END IF
   END IF
C In this case, no rounding was done because l=1
C
   DRNDCO=MSD DEN(1)
   RND = ROUND(MSDNUM).
   END IF
C If y-tilda is maximum, then estimate is 1 with no further calculation
C
IF(DR NDCO.EQ.SYSTMOD(MSDDEN)) THEN
   QUOT=1.000
   GO TO 30
ELSE
   END IF
C Find the quotient estimate in decimal according to the estimating rules.
C No operations needed if k=1. If k=1,k, one operation is needed. Otherwise,
C two operations are needed
C
   IF(MSDNUM.EQ.MSDDEN) THEN
      QUOT=DFLOT(JIFIK+RND/DRNDCO)
   ELSE
      IF(MSDNUM.EQ.MSDDEN+1) THEN
         QUOT=RND/DRNDCO
      ELSE
         IF(MSDNUM.EQ.MSDDEN/DRNDCO) THEN
            COUNT=COUNT+1.
         ELSE
         END IF
   END IF
   END IF

252
SUBROUTINE RACHDDO(MODNUM)(SYMOD(MODDEN)(MODDDEN))
QUOT=MNCHDDO(MODNUM)(SYMOD(MODDEN)(MODDDEN))
END IF
END IF

Convert the estimate to residue. When the algorithm is implemented, this will not be necessary because all calculations will be done in residue

DO I=1
   DO WHILE(E1.LE.NUMMOD)
      QUOTMOD(I)=M0D(SQI/QUOT),SYMOD(I))
   END DO
   I=I+1
   END DO

Find the modulo-2 value for the estimate. This will be done in parallel with the other modulus and therefore requires no extra residue operations

QUOTMOD=ANOD(SQI/QUOT),2.)
RETURN

This subroutine finds the mixed radix digits of the residue operand stored in RESIDU. The mixed radix digits are returned in MRCOUT, the number of operations required is returned in COUNT, and the XOR value of the operand is returned in TWOMOD

SUBROUTINE CONVIXCRESI0U,MRC0UT,COUNT,TU0N00)
COMMON SYSM0DC20),NUMN3D,SYSINVC96,20),HRCCOS
REALMS MRCC0EC20)
DIMENSION RESXDUC2Q)«CFYRESC20)»RESTENC20)
RE XL NRC0UTC20)

Copy the operand, and initialize the mixed radix coefficient register

I=1
   DO WHILE(I.LE.NUMMOD)
      COPYRES(I)=RESIDU(I)
      MRCOUT(I)=0.
      I=I+1
   END DO
   I=1
   DO WHILE(I.LE.NUMMOD)
      MRCOUT(I)=RESIDU(I)
      I=I+1
   END DO
   I=1
   DO WHILE(I.LE.NUMMOD)
      MRCOUT(I)=RESIDU(I)
      J=1
      DO WHILE(J.LE.NUMMOD)
         RESIDU(J)=ANOD(MRCOUT(I)),SYMOD(J))
         J=J+1
      END DO
      J=1
      DO WHILE(J.LE.NUMMOD)
         RESIDU(J)=ANOD(SYMOD(I)),SYMOD(J))
      J=J+1
   END DO

Subtract the next most significant residue digit from the residue digits

J=1
   DO WHILE(J.LE.NUMMOD)
      RESIDU(J)=ANOD(RESIDU(J)+RESTENC(J),(SYMOD(J)-1),SYMOD(J))
      J=J+1
   END DO
   J=1
   DO WHILE(J.LE.NUMMOD)
      RESTENC(J)=ANOD(SYMOD(I)),SYMOD(J))

253
Multiply by the inverse modulus in each position in which the inverse exists

```
J=A1
END DO
```

C Restore the residues, set the count and find the XOR value

```
J=1
DO WHILE(J.LE.NUMMOD)
  RESIDU(J)=CPYRES(J)
  J=J+1
END DO
```

C Set the operation count. Module 2 digit is found with no extra operations in parallel with the other moduli

```
COUNT=2*(NUMMOD-1)
TOMOD=0.
J=1
DO WHILE(J.LE.NUMMOD)
  TOMOD=AMOD(MCOUT(J)+MOMOD*Z)
  J=J+1
END DO
RETURN
END
```

C This subroutine simulates the Banerji-I algorithm. The numerator and denominator residues are passed in NUMRES and DENRES respectively. The decimal value of the truncated quotient is returned in BANQ, and the number of residue operations required to find the truncated quotient is returned in BANCNT

```
SUBROUTINE BANERJ(NUMRES,DENRES,BANQ,BANCNT)
COMMON SYSDID(20),NUMMOD,SYSINV(96,20),MACCOB
REAL MACC0E(20),BANG,IEYE,SUB
REAL NUMRES(20),DENRES(20),SCRMD(20),MACNUM(20)
REAL MACDEN(20)
BANG=0.000
BANCNT=0.
NUMRES=1.
```

C IS THE NUMERATOR ZERO? IF YES, THEN RETURN. ELSE, MRC THE DENOMINATOR IF THIS IS THE FIRST PASS, OR MRC THE NUMERATOR

```
35 TEST=.00.
  I=1
  DO WHILE(I.LE.NUMMOD)
    TEST=TEST+NUMRES(I)
    I=I+1
  END DO
```
END DO
IPCTests.T.EQ.0.3THE\nRETURN
ELSE
END IF
IPCCFEPAS.NE.1.3THE\nGO TO 30
ELSE

C Convert the denominator to mixed radix form, and bump the operation count

CALL CONXIXDENRES,ARCDEN,COU\nT,MODDEN)
BANCNT=BANCNT+COU\nNT
C Find the most significant nonzero mixed radix digit

I=NUMMOD
DO WHILE(I.GE.1)
   IF(ARCDEN(I).NE.0.3THEN
      MSDEN=I
      GO TO 29
   ELSE
      END IF
      I=I-1
END DO
C Do the denominator rounding. If digits of lesser significance are all zero, and the mos\nC t significant digit is 1, then no rounding is done

25 TEST=0.
   I=1
   DO WHILE(I.LT.MSDEN)
      TEST=TEST+ARCDEN(I)
      I=I+1
   END DO
IF TEST.EQ.0.3AND.(ARCDEN(MSDEN).EQ.1.3)THEN
   DRNDCD=ARCDEN(MSDEN)
ELSE
   DRNDCD=ARCDEN(MSDEN)*1.
   END IF
   ONEPAS=0.
   END IF
C Convert numerator to mixed radix form, and bump the count

30 CALL CONXIXNUMRES,ARCHUR,COU\nNT,MODNUM)
BANCNT=BANCNT+COU\nNT
C Find the most significant nonzero digit of the numerator

I=NUMMOD
DO WHILE(I.GE.1)
   IF(ARCHUR(I).NE.0.3THEN
      MSDNUM=I
      GO TO 40
   ELSE
      END IF
      I=I-1
END DO
C

255
IF (MSDNUM < MSDDEN) THEN
   RETURN
ELSE IF (MSDNUM = MSDDEN) THEN
   ZEYE = DBLE (INT (MSDNUM / ORNDCO))
   GO TO 50
ELSE IF (MSDNUM = 0) THEN
   ZEYE = DBLE (MSDDEN) / INT (MSDDEN / ORNDCO)
   BANCNT = BANCNT + 1.
   GO TO 50
ELSE
   SUB = ACCDEE (MSDNUM / (SYSDODC * MSDDEN))
   ZEYE = DBLE (MSDNUM) / INT (MSDDEN / ORNDCO) * SUB
   BANCNT = BANCNT + 2.
ENDIF
ENDIF

C Update the running quotient sum, stored as BANQ. If the quotient estimate
C is nonzero, then add it to the running sum; compute a new numerator; and
C increment count. If it is zero, then find the quotient correction in
C preparation for return

IF (ZEYE .NE. 0.0) THEN
   BANQ = BANQ + ZEYE
   BANCNT = BANCNT + 1.
ENDIF

C Compute the next partial numerator stored in NUMRES, and bump the count
C by two operations

I = 1
DO WHILE (I .LE. NUMMOD)
   SCRMD2 (I) = AMOD (ZEYE, DBLE (SYSMOD (I)))
   I = I + 1
END DO
I = 1
DO WHILE (I .LE. NUMMOD)
   SCRMD2 (I) = AMOD (SCRMD2 (I) * DEMRES (I), SYSMOD (I))
   I = I + 1
END DO
I = 1
DO WHILE (I .LE. NUMMOD)
   NUMRES (I) = AMOD (NUMRES (I) * (SYSDODC I - 1.) * SCRMD2 (I), SYSMOD (I))
   I = I + 1
END DO
BANCNT = BANCNT + 2.
GO TO 35
ELSE

C The quotient estimate was zero, so we must find the correction to the sum.

I = NUMMOD
DO WHILE (I .GE. 1)
   IF (MCNUM (I) .LT. MCDEN (I)) THEN
      BANQ = BANQ + 0.0
      BANCNT = BANCNT + 1.
   END IF
   I = I - 1
END DO
RETURN
ELSE
IF (NACNUM(I) >.LT. NACDEN(I)) THEN
RETURN
ELSE
END IF
END IF
I = I + 1
END DO
BAND = BAND + 1.000
BANCNT = BANCNT + 1.
RETURN
APPENDIX E

Derivation of Stored Table Size for SA

The following derivation is that of the expression for the stored table size required by the SA, as given in Section D.1.

The SA uses

$$\max \left( \frac{m_{n-2}}{2} \left( \frac{m_{n-2}}{2} + 1 \right) \right) - 1, \frac{m_{n-1}(m_{n-1}-1)}{2} + \frac{(n-2)(n-1)}{2}$$

total storage, and for practical modulus sets (viz., those for which the moduli are approximately the same size) it uses at most the same amount of storage as the OSRA. This expression is derived as follows.

From Equation (3.41), the SA estimate is

$$E_{i+1} = \begin{cases} \begin{array}{c} \frac{\tilde{x}_k}{\tilde{y}_2} \\ \frac{m_2}{\tilde{y}_2} \end{array} & , \text{if } k=\ell \\ \frac{\tilde{x}_k}{\tilde{y}_2} \end{cases} \begin{array}{c} m_{k+1} \cdots m_{k-1} \end{array} , \text{if } k=\ell+1 \tag{3.41}$$

Two stored tables are used to compute it. The first stores residue encoded quantities $\frac{\alpha/\tilde{y}_2}{\tilde{y}_2}$ for all possible values of $\alpha$ (i.e., $\tilde{x}_k$ or $m_2$) and $\tilde{y}_2$. The other table stores residue encoded products $m_{k+1} \cdots m_{k-1}$ for all possible values of $k$ and $\ell$ such that $k>\ell+1$, as indicated by the estimate definition above.
The size of the first table is found by counting the number of possible values that $\alpha$ and $\hat{y}_k$ can have, for all values of $k$ and $\lambda$, remembering that $m_n > m_{n-1} > \ldots > m_1$.

The quantity $\alpha$ can assume any value in $[1, \lfloor \frac{m_n}{2} \rfloor + 1]$ if $\lfloor \frac{m_n}{2} \rfloor > m_{n-1}$, or any value in $[1, m_{n-1}]$ if $\lfloor \frac{m_n}{2} \rfloor < m_{n-1}$. This is true because (if $\alpha = x_k$) $x_k$ can assume any value in $[1, \lfloor \frac{m_n}{2} \rfloor + 1]$ if $k=n$, or any value in $[1, m_{n-1}]$ if $k<n$, and if $\alpha = m_k$, then $\alpha$ assumes values of moduli in $[1, m_{n-1}]$.

The quantity $\hat{y}_k$ can assume any value in $[1, \lfloor \frac{m_n}{2} \rfloor]$ if $\lfloor \frac{m_n}{2} \rfloor > m_{n-1}$, or any value in $[1, m_{n-1}]$ if $\lfloor \frac{m_n}{2} \rfloor < m_{n-1}$. This is because, for $k=n$, $y_n < \lfloor \frac{m_n}{2} \rfloor$ and if $y_n = \lfloor \frac{m_n}{2} \rfloor$, then $y_{n-1} < \lfloor \frac{m_{n-1}}{2} \rfloor$ so that $y_n$ is not rounded up when $y_n = \lfloor \frac{m_n}{2} \rfloor$. Otherwise, if $k<n$, $y_k < m_{n-1}$, so that $\hat{y}_k < m_k$.

Combining the possible values for $\alpha$ and $\hat{y}_k$ in all cases, we have that:

1) If $\lfloor \frac{m_n}{2} \rfloor > m_{n-1}$, then $\alpha \in [1, \lfloor \frac{m_n}{2} \rfloor + 1]$, and $\hat{y}_k \in [1, \lfloor \frac{m_n}{2} \rfloor]$.

2) If $\lfloor \frac{m_n}{2} \rfloor < m_{n-1}$, then $\alpha \in [1, m_{n-1}]$ and $\hat{y}_k \in [1, m_{n-1}]$.

For case 1), the stored table contains residue encoded values of $\lfloor \alpha/\hat{y}_k \rfloor$ only, because $\alpha < \hat{y}_k$ does not occur (because of the stopping condition for the SA). Furthermore, the SA does not perform a table access when $\hat{y}_k = 1$, because in this case $\lfloor \alpha/\hat{y}_k \rfloor = \alpha$. Therefore, for case 1), the size of the required table is

$$\lfloor \frac{m_n}{2} \rfloor + 1 - 1.$$
For case 2), the size of the table is found in the same way. That is, 
α<\bar{y}_k does not occur, and the table is not accessed when \bar{y}_k=1.
Therefore, in case 2), the table size is

\[ \frac{m_n-1(m_n-1-1)}{2} \]

Therefore, the size of the first table is

\[ \max \left( \left\lfloor \frac{m_n}{2} \right\rfloor \left( \left\lfloor \frac{m_n}{2} \right\rfloor +1 \right) -1, \frac{m_n-1(m_n-1)}{2} \right) \]

The size of the second table (viz., the table of residue encoded produces m_{k+1}\cdots m_{k-1} for k\geq k+2) is \( \frac{(n-2)(n-1)}{2} \). This is found by summing the number of unique moduli products for all k<n-2 and k<n.

Adding the storage required by both tables, we have that the SA uses a total storage of

\[ \max \left( \left\lfloor \frac{m_n}{2} \right\rfloor \left( \left\lfloor \frac{m_n}{2} \right\rfloor +1 \right) -1, \frac{m_n-1(m_n-1)}{2} \right) + \frac{(n-2)(n-1)}{2} \]

For practical modulus sets (i.e., those for which the moduli are about the same size), the SA uses at most the same amount of storage as the OSRA. This is because, for these sets, \( \left\lfloor \frac{m_n}{2} \right\rfloor < m_{n-1} \) so that the SA uses

\[ \frac{m_n-1(m_n-1-1)}{2} + \frac{(n-2)(n-1)}{2} \]
total storage. From Section C.1 of Chapter III, we have that the OSRA uses

\[ \frac{(m_n-1)(m_n-2)}{2} + \frac{(n-2)(n-1)}{2} \]
total storage. Therefore we conclude that the SA uses storage at most equal to that of the OSRA, because m_{n-1}<m_n-1.
APPENDIX F

The Simulator Program for the SA and the OSRA

APPENDIX F: THE SIMULATOR PROGRAM FOR THE SA AND THE OSRA

Note: In this program, the OSRA is referred to as the Banerji-I Algorithm.

This program simulates the Signed and Banerji-I algorithms. Input file is
assumed to be FORMAT, and arranged as follows:

LINE 1: NUMBER OF MODULUS SETS TO BE SIMULATED, INTEGER FORMAT
LINE 2: NUMBER OF DIVISION PROBLEMS TO BE SIMULATED FOR
MODULUS SET 1, INTEGER FORMAT
LINE 3: NUMBER OF MODULI IN MODULUS SET 1, FOLLOWED BY A
comma, followed by a zero if there is an even modulus
in MODULUS SET 1, or a 1 if there is no even modulus
in MODULUS SET 1. Both these parameters must be input
in INTEGER FORMAT
LINE 4: THE MODULI IN SET 1, ORDERED SMALLEST TO LARGEST, IN
REAL FORMAT
LINE 5: THESE THREE PARAMETERS FOR MODULUS SET 1, IN REAL
FORMAT AND SEPARATED BY COMMAS: 1. AVERAGE RUNNING
TIME OF THE SIGNED ALGORITHM, FOUND BY A PREVIOUS
RUN OF THIS PROGRAM (THIS QUANTITY IS NEEDED FOR
VARIANCE CALCULATIONS ONLY) 2. AVERAGE RUNNING TIME
OF THE BANERJI-I ALGORITHM (LIKEWISE NEEDED FOR
VARIANCE CALCULATIONS ONLY) 3. AVERAGE DIFFERENCE OF
RUNNING TIMES, NEEDED ONLY FOR CALCULATION OF THE
VARIANCE OF THE RUNNING TIME DIFFERENCE
LINE 6: VALUES OF AVERAGE RUNNING TIME WILL BE COMPUTED OVER
SAMPLES OF SIZES EQUAL TO MULTIPLES OF THIS PARAMETER
IT MUST BE INPUT IN INTEGER FORMAT

LINES 2 THRU 6 MUST BE GIVEN FOR EACH MODULUS SET TO BE SIMULATED.

MAJOR VARIABLE NAMES:

H: PRODUCT OF SYSTEM MODULI
NUM: NUMERATOR IN DECIMAL
den: DENOMINATOR IN DECIMAL
DJ: QUOTIENT IN DECIMAL
MRCODE(I): THE MIXED RADIX COEFFICIENTS IN DECIMAL, IN INCREASING ORDER
SYSMOD(I): THE SYSTEM MODULI, IN INCREASING ORDER
NUMMOD: NUMBER OF MODULI
SYSINV(I,J): THE MULTIPlicative INVERSE OF I, MODULO SYSMOD(J)
NUMRES(I): THE ITH RESIDUE DIGIT OF THE NUMERATOR
DENRES(I): THE ITH RESIDUE DIGIT OF THE DENOMINATOR
QUOT(I,J): THE ITH RESIDUE DIGIT OF THE QUOTIENT
MRCDEN(I,J): THE MIXED RADIX DIGITS OF THE DENOMINATOR, INCREASING ORDER
MRCNUM(I,J): SAME AS ABOVE FOR THE NUMERATOR
ESTRES(I,J): THE ITH RESIDUE DIGIT OF THE QUOTIENT ESTIMATE IN SOME ITERATION
ESTCAL: FLAG WHICH EXCEEDS ZERO ONLY WHEN AT LEAST ONE QUOTIENT ESTIMATE
HAS BEEN MADE
SCATEN(I): THE NUMBER OF DIVISION PROBLEMS WHICH REQUIRED EXACTLY I
RESIDUE OPERATIONS TO SOLVE USING THIS ALGORITHM
SCBAN(I): SAME AS ABOVE, EXCEPT FOR THE BANERJI-I ALGORITHM
NUMRES(I): THE NUMERATOR RESIDUES FOR THE BANERJI-I ALGORITHM
DENRES(I): THE DENOMINATOR RESIDUES FOR THE BANERJI-I ALGORITHM
QUOT(I,J): THE QUOTIENT, IN DECIMAL, IN THE BANERJI-I ALGORITHM
NUMSET: THE NUMBER OF MODULUS SETS TO BE SIMULATED
SAME: THE AVERAGE RUNNING TIME FOR THIS ALGORITHM, FOUND FROM A PREVIOUS

261
RUN OF THE SIMULATOR. THIS NUMBER IS USED ONLY FOR VARIANCE CALCULATIONS.

SAME AS SAMMEL, ONLY FOR THE BANERJI-I ALGORITHM.

SAMDIIF: THE SAMPLE AVERAGE DIFFERENCE IN RUNNING TIMES, FOUND AS BANERJI TIME MINUS SIMON ALGORITHM TIME. THIS DIFFERENCE IS FOUND FROM PREVIOUS RUNS OF THE SIMULATOR, AND IS USED ONLY FOR CALCULATION OF VARIANCE.

NEVERV: THE SAMPLE AVERAGES ARE COMPUTED FOR SAMPLE SIZES WHICH ARE MULTIPLES OF THIS NUMBER.

REAL*8 M,NUM, DEN, Q, MRCCOE(20), POSMD, NUMY, QXAB, STUPID, DUMB
COMMON SYMDIC(20), NUMMOD, SYSMIC(20), NUMC20
REAL NUMRES(20), DNMRES(20), QUOTE(20), MRCCFEM(20), RESFQ(20)
INTEGER ESTCAL
REAL SCATVR(500), NUMSAN(20), DEBAN(20), SCABAN(500), SCAOIF(500)
REAL*0 BAND
OPEN (UNIT=2, FILE="STUDENT_DISKRICE99990000.DAT", STATUS="OLD")
READ(2*) NUMSET
C
C INITIALIZE THE MODULUS SET COUNTER AND INPUT DATA
C
IJ=1
DO WHILE (IJ.LE.NUMSET)
READ(2*) NUMSAN
READ(2*) NUMMOD, PARIT
READ (2*) (SYMSIC(I), I=1,NUMMOD)
READ (2*) SAMM, SAMD, SAMDI
READ (2*) NEVER
C
C OUTPUT NUMBER OF DATA POINTS AND SIZE OF OPCOUNT ARRAYS
C
N=SIZE=500
NUMPTS=NUMSAN/NEVER
WRITE(4*) NUMPTS, N, SIZE, NEVER
C
C INITIALIZE OPCOUNT ARRAYS
C
DO 13 I=1,N
SCATVR(I)=0.
SCABAN(I)=0.
SCAOIF(I)=0.
13 CONTINUE
C
C COMPUTE THE DYNAMIC RANGE FOR THIS MODULUS SET
C
M=1.0D0
DO 10 I=1, NUMMOD
M=M+EXP(2*SYMLOGCID(I))
10 CONTINUE
C
C LOAD TABLE OF INVERSES MODULO THE SYSTEM MODULI. THIS CODE COMPUTES A
C VALUE OF ZERO FOR THE INVERSE OF INTEGERS THAT ARE NOT RELATIVELY PRIME TO
C THE MODULUS.
DO 20 J=1,NUMMOD
   K=SYSMOD(J)-1
DO 30 J=1,K
DO WHILE(J MOD(J+1)+1)/INT(SYSMOD(J))/EQ.1)
   SYSINV(J,1)=L
GO TO 30
END DO
40 CONTINUE
30 CONTINUE
20 CONTINUE
C
C LOAD ARRAY OF MIXED RATIO COEFFICIENTS
C
MACCOE(1)=1.000
DO WHILE(J.LE.NUMMOD)
   MACCOE(J)=MACCOE(J-1)*INT(SYSMOD(J-1))
   J=J+1
END DO
C
C SET RANDOM NUMBER GENERATOR SEED, INITIALIZE I (WHICH IS THE SAMPLE COUNTER) AND OTHER VARIABLES
C
ISEED=55069
I=1
OPCONT=0.
VARCNT=0.
PREVOP=0.
PREVAR=0.
SANTO=0.
SAMVAR=0.
DIFTEAR=0.
DIFVAR=0.
PREDIF=0.
VARPRE=0.
C
C DONE ENOUGH SAMPLES?
C
DO WHILE(I.LE.NUMSAM)
C
C INITIALIZE OPERATION COUNTER
C
CASCNT=0.
45 J=1
C
C RANDOMLY SELECT A NUMERATOR AND DENOMINATOR, IN RESIDUE. COPY FOR USE BY THE BANERJI-I ALGORITHM
C
DO WHILE(J.LE.NUMMOD)
   NUMRES(J)=INT(SYSMOD(J)*RANISEED))
   DENRES(J)=INT(SYSMOD(J)*RANISEED))
   J=J+1
END DO
J=1
DO WHILE(J.LE.NUMMOD)
   NUMBAN(J)=NUMRES(J)
   DENBAN(J)=DENRES(J)
   J=J+1
END DO
C CONVERT TO "NATURAL" INTEGER FORM FOR VARIOUS PURPOSES WHICH ARE MORE
C EASILY DONE IN DECIMAL. TEST TO SEE IF DENOMINATOR HAPPENS TO BE ZERO.
C IF SO, GO GET ANOTHER DENOMINATOR
C
CALL CONINT(NUMRES,N,NUM)
CALL CONINT(DENRES,N,DEM)
IF(DEN==0.000) THEN
    WRITE(4) "ZERO HIT"
    GO TO 45
ELSE
    END IF
C
C COMPUTE CORRECT QUOTIENT FOR THE BAHERJI ROUTINE
C
DEN=INT(NUM/DEN)
C
C FIND THE MAXIMUM POSITIVE INTEGER REPRESENTABLE BY THE MODULUS SET
C
IF(MPARIT.EQ.1) THEN
    POSBND=INT((M/2.000)-1.000)
ELSE
    POSBND=INT(M/2.000)-1.000
END IF
C
C FIND "SIGNED DENOMINATOR" AND ITS SIGN
C
IF(DEN>POSBND) THEN
    SGNWH=1.
    DEN=POSBND
ELSE
    SGNWH=-1.
END IF
C
C FIND THE "SIGNED NUMERATOR"
C
IF(NUM>POSBND) THEN
    NUM=NUM-POSBND
ELSE
    END IF
J=1
C
C INITIALIZE THE QUOTIENT
C
DO WHILE(J.LE.NUHMOD)
    QUOT(J)=0.
    J=J+1
END DO
C
C INITIALIZE THE ESTIMATE COUNTER
C
ESTCAL=0
C
C CONVERT DENOMINATOR TO MIXED RADIX FORM
C
CALL CONMIX(DENRES,NRCEN,COUNT,TWODEN)
OPCDNT=OPCDNT+COUNT
CASCLNT=CASCNT+COUNT
C
C IF DENOMINATOR IS NEGATIVE * FLIP ITS MIXED RADIX COEFFICIENTS

264
IF(SGNHY.EQ.-1.) THEN
   CALL FLIP(NRCODEN)
   OPCODE=OPCODE+1.
   CASCNT=CASCNT+1.
ELSE
   END IF

CONVERT NUMERATOR I TO MIXED RADIX FORM
CALL CONMIX(NUMRES,NRCNUM,COUNT,TWONUM)
OPCODE=OPCODE+COUNT
CASCNT=CASCNT+COUNT

CONVERT TO "NATURAL" INTEGER FORM SO THAT THE SIGN CAN BE DETERMINED
MORE EASILY. IF THE ITH NUMERATOR IS NEGATIVE, FLIP ITS SIGN AND ITS
MIXED RADIX DIGITS. OTHERWISE, KEEP SIGN THE SAME AS THAT OF Y
CALL CONINT(NUMRES,M,NUMEYE)
IF(NUMTE.GT.POSNHY) THEN
   SIGN=-SGNHY
   CALL FLIP(NRCNUM)
   OPCODE=OPCODE+1.
   CASCNT=CASCNT+1.
ELSE
   SIGN=SGNHY
ENDIF
J=NUMMOD

IS ABSOLUTE VALUE OF ITH NUMERATOR LESS THAN Y? IF SO, GO UPDATE THE
QUOTIENT AND SEE IF IT IS CORRECT. OTHERWISE, PROCEED
DO WHILE(J.GE.1.)
   IF(NRCNUM(JJ).LT.NRCODEN(JJ)) THEN
      GO TO 70
   ELSE
      IF(NRCNUM(JJ).GT.NRCODEN(JJ)) THEN
         GO TO 50
      END IF
   END IF
   J=J-1
ENDDO

GO GET A QUOTIENT ESTIMATE
CALL QOUEST(NRCNUM,NRCE4,ESTRES,COUNT,ESTTWO,UPFLAG)
OPCODE=OPCODE+COUNT
CASCNT=CASCNT+COUNT

SET FIRST CALL FLAG
ESTCAL=ESTCAL+1

IF Y WAS ROUNDED UP, AND IT IS FIRST CALL, THEN ADJUST OPERATION COUNT
IF(UPFLAG.EQ.1.) THEN
   IF(ESTCAL.EQ.1.) THEN
      OPCODE=OPCODE+2.
      CASCNT=CASCNT+2.
   END IF
ENDIF
ELSE
  DPCNT=DPCNT+1.
  CASCNT=CASCNT+1.
END IF
ELSE
END IF

C FIND NEW QUOTIENT SUM AND NUMERATOR, DEPENDING ON SIGN. THEN, REPEAT THE
C CYCLE
C
IF(SIGN.EQ.1..THEN
  J=1
  DO WHILE(J.LE.NUMMOD)
    QUOT(J)=MOD(QUOT(J)+ESTRES(J),SYSMOD(J))
    NUMRES(J)=MOD(NUMRES(J)+SYSMOD(J)-1.,ESTRES(J)+DESTRS(J),SYSMOD(J))
  J=J+1
  END DO
ELSE
  J=1
  DO WHILE(J.LE.NUMMOD)
    QUOT(J)=MOD(QUOT(J)+SYSMOD(J)-1.,ESTRES(J),SYSMOD(J))
    NUMRES(J)=MOD(NUMRES(J)+ESTRES(J)+DESTRS(J),SYSMOD(J))
  J=J+1
  END DO
END IF
DPCNT=DPCNT+1.
CASCNT=CASCNT+1.
GO TO 79
TO
J=1

C PREPARE TO ADJUST THE QUOTIENT DEPENDING ON WHETHER THE NUMERATOR
C IS ZERO AND THE SIGN IS POSITIVE
C
DO WHILE(J.LE.NUMMOD)
  IF(NUMNUM(J).NE.0.)THEN
    GO TO 82
  ELSE
    J=J+1
  END IF
END DO
GO TO 84

C IF NUMERATOR IS NOT ZERO AND SIGN IS POSITIVE, THEN DECREMENT QUOTIENT
C
82 IF(SIGN.NE.1..THEN
  J=1
  DO WHILE(J.LE.NUMMOD)
    QUOT(J)=MOD(QUOT(J)+SYSMOD(J)-1.,SYSMOD(J))
  J=J+1
  END DO
  DPCNT=DPCNT+1.
  CASCNT=CASCNT+1.
ELSE
END IF

C BEGIN TO CHECK THE QUOTIENT. IF Q IS NEGATIVE, FIND THE RESIDUE DIGITS
C OF ITS ABSOLUTE VALUE, AND CONVERT IT TO INTEGER FORM. THEN TACK
C ON THE SIGN
C
84 CALL CONINT(QUOT,M,Q)
IF (Q.GT.POSBND) THEN
  J=J+1
  DO WHILE (J.LE.NUMMOD)
    QUOT(J)=MOD(SYSMOD(J)+(SYSMOD(J)-1).*QUOT(J),SYSMOD(J))
    J=J+1
  END DO
  CALL CONMWNT(QUOT,M,Q)
  Q=1.DO30Q
ELSE
  END IF
END IF

C THIS MACHINE DOESN'T GIVE TRUE "GREATEST INTEGER" WHEN THE OPEAND C IS NEGATIVE. YOU MUST DO IT THE HARD WAY
C
STUPID=NUM/DEN
DUMB=INT(STUPID)
IF (STUPID.NE.DUMB) THEN
  IF (STUPID.LT.0.D0) THEN
    DUMB=NUM-1.D000
  ELSE
   END IF
  END IF

C IS THE QUOTIENT CORRECT?
C
IF (DUMB.NE.DUMB) THEN
  WRITE(*,*) "ERROR Q", Q,"I","I
  STOP
ELSE
  END IF

C IF SO, INCREMENT THE SCATTER ARRAY POSITION AND SIMULATE BANERJI
C
SCATER(CASCHNT)=SCATER(CASCHNT)+1,
  CALL BANERJI(DUMB,NUM,0,BB,NB,BANCT)
C
C IS THE BANERJI QUOTIENT CORRECT?
C
IF (BB.NE.BB) THEN
  WRITE(*,*) "ERROR BANERJI", BB,"I","I
  WRITE(*,*) "NUM", NUM,"D", DEN
  STOP
ELSE
  END IF

C CALCULATE VARIOUS STATISTICS FOR BOTH ALGORITHMS
C
SCEBAN(BANCNT)=SCEBAN(BANCNT)+1,
  BANTOT+BANTOT>BANCT
  BANVAR+BANVAR>BANCT
  BANVAR+BBNVAR=(BANCNT-SAMNEN)*2
  DIF=BANCNT-CASCNT
  SCDFP(BSF=BSCF)=SCDFP(BSF=1)
  DIF=DIPTIM+DIF
  DIF=DIPTIM+DIF
  VANCNT+VANCNT=(CASCHNT-SAMNEN)*2
  IF (Q.EQ.0) THEN
C
C THE FOLLOWING 5 VARIABLES ARE IN ORDER: SAMPLE AVERAGE NUMBER
C
267
C OF OPERATIONS FOR MY ALGORITHM, SAME FOR BANERJI-I ALGORITHM, SAMPLE
C AVERAGE DIFFERENCE OF OPERATIONS, SAMPLE VARIANCE FOR MY ALGORITHM,
C SAME THING FOR BANERJI-I
C
OUTRE=OPC0N/T
OUTPOR=BANTOT/I
COMMON=DIPTOT/I
OUTSEY=VARTOT/I-1
OUTATE=BANVAR/I-1
OUTIN=DIPTOT/I-1
WRITE4,0)OUTRE,OUTPOR,OUTSEY,OUTATE
PREDIP=DIPTIM
VARPRE=DIPTIM
ELSE
END IF
I=I+1
END DO
C
C NO MORE SAMPLES. PRINT OUT THE SCATTER MATRICES.
C
WRITE4,3)SCATTER
WRITE4,3)SCABAN
J=J+1
C
C NO MORE MODULUS SETS. SO STOP
C
END DO
STOP
END
C
THIS SUBROUTINE CONVERTS THE MIXED RADIX DIGITS OF A NEGATIVE OPERAND.
C STORED IN RESIDU, TO THE MIXED RADIX DIGITS OF THE ABSOLUTE VALUE OF THE
C OPERAND, RETURNED IN RESIDU
C
SUBROUTINE FLIPRESDU
COMMON STSN00(20),NUM00,STSV00(20),HTVE
REAL# RESI00(20),STSN00(20)
C
C FIND THE LEAST SIGNIFICANT MIXED RADIX DIGIT
C
LEESIG=0
J=1
DO WHILE(J.LE.NUM00)
IF(RES100(J).NE.0.)THEN
LEESIG=J
DO TO 10
ELSE
END IF
J=J+1
END DO
C
C COMPUTE THE ABSOLUTE VALUE
C
10 IF(LEESIG.NE.0.)THEN
J=1
DO WHILE(J.LE.NUM00)
RESIDU(J)=MOD(SYSTMOD(J)-1.,SYSTMOD(J)-1.)*RESIDU(J),SYSTMOD(J))
J=J+1
END DO
DO WHILE(J.LT.LEESIG)
    RESIDU(J)=0.
    J=J+1
END DO
ELSE
    END IF
    RETURN
END

C THIS SUBROUTINE CONVERTS THE RESIDUE OPERAND RESIDU TO ITS DECIMAL
C EQUIVALENT, RETURNED IN X
C
SUBROUTINE CONINTCRESIOU,N,X)
    CONHON STSN0DC20),NU"HO0,STSINVt»6.20).HRCCOE
    REALAB HRCC0EC20>.X.N
    OIHENSION RESIQU(20),SCRNOD(ZO),CPVRES<20>
C
    C MAKE A COPY OF THE RESIDU OPERAND, USED TO RESTORE THE INPUT WHEN RETURNING
C
    DO 5 I=1,NUMMOD
        CRYPTO(I)=RESIDU(I)
    CONTINUE
C
    C DO A MIXED RADIX CONVERSION, AND SUM THE MIXED RADIX TERMS
C
    CALL CONNIX(RESIDU,SCRNOD,COUNT,NUMMOD)
    X=0.000
    DO 10 I=1,NUMMOD
        X=X+SCRNOD(I)*HRCC0EC(I)
    CONTINUE
C
    C RESTORE THE INPUT AND RETURN
C
    DO 15 I=1,NUMMOD
        RESIDU(I)=CRYPTO(I)
    CONTINUE
    RETURN
END

C THIS SUBROUTINE COMPUTES A QUOTIENT ESTIMATE FOR THE MIXED RADIX DIGITS
C OF NUMERATOR AND DENOMINATOR, PASSED IN NUMCNU AND ADCDEN, RESPECTIVELY.
C THE QUOTIENT ESTIMATE RESIDUES ARE RETURNED IN QUORES, AND THE MODULO-2
C-value of the quotient estimate is returned in QUOTNO. THE NUMBER OF OPER-
CATIONS REQUIRED IS RETURNED IN COUNT, AND UPFLAG IS 1 IF THE DENOMINAT-
C WAS ROUNDED UP. ELSE, UPFLAG IS ZERO
C
SUBROUTINE QUOESTCNRCNUN,NRCOEN,QUORES,COUNT,QUOTNO,UPFLAG)
    CONHON STSNOO(ZO),NUNN00,STSINVC96,20),NRCC0E
    REAL AB NRCCOE20) NRNOCO
    REAL AB QUORES(20) SUB COUNT.0. UPFLAG.0.
C
    C FIND THE MOST SIGNIFICANT NONZERO DIGIT OF THE DENOMINATOR
C
    I=NUMMOD
DO WHILE(i.ge.1)
   IF(NORTHDEN(i).NE.0) THEN
      MSDDEN=I
      GO TO 10
   ELSE
      END IF
   I=I-1
END DO
C
C FIND THE MOST SIGNIFICANT NONZERO DIGIT OF THE NUMERATOR
C
10  I=NUMNOO
    DO WHILE(i.ge.1)
       IF(NUMNUM(i).NE.0) THEN
          MSDNUM=I
          GO TO 20
       ELSE
          END IF
       I=I-1
    END DO
C
C PERFORM THE SIGNED ALGORITHM ROUNGING. FIND THE VALUES OF X-TILDA
C AND Y-TILDA, AND SET UPFLAG
C
20  IF(MSDDEN.GT.1) THEN
       IF(MSDDEN(MSDDEN-1).LE.AINT(SYSMDEN(MSDDEN-1)/2)) THEN
          DRNDC0=ARCNUM(MSDNUM)
       ELSE
          DRNDC0=ARCDEN(MSDDEN-1)
       END IF
       UPFLAG=1
    ELSE
       END IF
       DRNDC0=ARCNUM(MSDNUM)
    END IF
C
C FIND THE QUOTIENT ESTIMATE IN DECIMAL ACCORDING TO THE ESTIMATING RULES
C
   IF(MSDNUM.EQ.MSDDEN) THEN
      QUOT=DPIIT(JIPIX(NRNCDC/DRNCDC))
   ELSE IF(MSDNUM.EQ.MSDDEN+1) THEN
      QUOT=NRNCDC0*PIIT(JIPIX(SYSMDEN(MSDDEN)/DRNCDC))
      COUNT=COUNT+1
   ELSE
      SUB=INT(SYSMDEN(MSDNUM)/SYSMDEN(MSDDEN)+NRNCDC0SYSMDEN)/DRNCDC)
      QUOT=NRNCDC0*PIIT(JIPIX(SYSMDEN(MSDDEN)/DRNCDC))SUB
      COUNT=COUNT+2
   END IF
C
C CONVERT THE ESTIMATE TO RESIDUE
C
   I=1
   DO WHILE(i.le.NUMMOD)
      QUOTES(I)=NUM(SINGE(QUOT),SYSMD(I))
      I=I+1
   END DO
C FIND THE MODULO-2 VALUE FOR THE ESTIMATE
C
QUOTD=MOD(SQRTDQUOTD),2)
RETURN
END
C
C THIS SUBROUTINE FINDS THE MIXED RADIX DIGITS OF THE RESIDUE OPERAND
C STORED IN RESIDU. THE MIXED RADIX DIGITS ARE RETURNED IN MACOUT, THE
C NUMBER OF OPERATIONS REQUIRED IS RETURNED IN COUNT, AND THE XOR VALUE
C OF THE OPERAND IS RETURNED IN TWOHOD
C
SUBROUTINE CONMIX RESIDU MACOUT COUNT TWOHOD
COMMON SYSMOD(20),NUMMOD,SYNSIN(96,20),MCODE
REAL MACODE(20)
DIMENSION RESIDU(20),CPRES(20),RESTE(20)
REAL MACOUT(20)
C
COPY THE OPERAND AND INITIALIZE THE MIXED RADIX COEFFICIENT REGISTER
C
I=1
DO WHILE(I.LE.NUMMOD)
CPRES(I)=RESIDU(I)
MACOUT(I)=0
I=I+1
END DO
J=1
DO WHILE(J.LE.NUMMOD)
MACOUT(J)=RESIDU(J)
J=J+1
END DO
DO WHILE(J.LE.NUMMOD)
RESTE(J)=MOD(CPRES(J),SYSMOD(J))
J=J+1
END DO
C
SUBTRACT THE NEXT MOST SIGNIFICANT RESIDUE DIGIT FROM THE RESIDUE DIGITS
C
J=1
DO WHILE(J.LE.NUMMOD)
RESIDU(J)=MOD(RESIDU(J)+RESTE(J)MOD(SYSMOD(J)-1),SYSMOD(J))
J=J+1
END DO
J=1
DO WHILE(J.LE.NUMMOD)
RESTE(J)=MOD(RESIDU(J),SYSMOD(J))
J=J+1
END DO
C
MULTIPLY BY THE INVERSE MODULUS IN EACH POSITION IN WHICH THE INVERSE EXISTS
C
J=1
DO WHILE(J.LE.NUMMOD)
IF(J.NE.1)
RESIDU(J)=MOD(RESIDU(J)+SYNSIN(RESTE(J),J)*SYSMOD(J))
ELSE
END IF
J=J+1
END DO
I=I+1
END DO
C RESTORE THE RESIDUES, SET THE COUNT AND FIND THE XOR VALUE

I=I+1
DO WHILE(I.LE.NUMMOD)
   RESIDUE(I)=RESIDUE(I)*CPRES(I)
   I=I+1
END DO
COUNT=COUNT+NUMMOD-1
THMOD=0.
I=1
DO WHILE(I.LE.NUMMOD)
   THMOD=THMOD+MCPOUT(I)*THMOD,2.
   I=I+1
END DO
RETURN
END

C THIS SUBROUTINE IS NOT USED BY THE SIGNED ALGORITHM, BUT REMAINS FOR FUTURE REFERENCE

SUBROUTINE BANTWOD(NUMOD,MACDEN,QUOTES,COUNT,QUOTWO,UPFLAG)
COMMON SYMNO(20),NUMOD,SYMUK(96,20),MACDEN
REAL# MACDEN(20)
DIMENSION QUOTES(20)
REAL MACDEN(20),MACDEN(20),NUMOD
REAL# QUOT COUNT=0.
UPFLAG=0.
I=NUMMOD
DO WHILE(I.GE.1)
   IF(MACDEN(I).NE.0.)THEN
      MSDEN=I
      GO TO 10
   ELSE
      END IF
      I=I-1
   END DO
10 MSDEN=I
DO WHILE(I.GE.1)
   IF(MACDEN(I).NE.0.)THEN
      MSDUNUM=I
      GO TO 20
   ELSE
      END IF
      I=I-1
   END DO
20 NUMOD=MACDEN(MSDEN)
   NUMOD=MACDEN(MSDUNUM)
   IF(MSDUNUM.EQ.MSDDEN)THEN
      QUOT=FLOAT(JCOMP(21))
   ELSE
      IF(MSDUNUM.EQ.MSDDEN)THEN
         QUOT=1.0DCMP(21)
      END IF
      COUNT=COUNT+1.
   ELSE
      SUB=MACDEN(MSDUNUM)/(SYMUK(MSDDEN)=MACDEN(MSDDEN))
      QUOT=1.0DCOMP(JCOMP(21))
      COUNT=COUNT+2.
   END IF
RETURN
This subroutine simulates the Banerji-I algorithm. The numerator and
denominator residues are passed in NUMRES and DENRES, respectively. The
decimal value of the truncated quotient is returned in BANQ, and the number
of residue operations required to find the truncated quotient is returned
in BANCNT.

SUBROUTINE BANERJ(NUMRES, DENRES, BANQ, BANCNT)
COMMON SYMSOO(20), NUMMOD, SYNSINV(96, 20), HRCCOE
REAL# HRCCOE(20), BANO, ZETE, SUB
REAL NUMRES(20), DENRES(20), SERMOD(20), ARCHNUM(20)
REAL MREDEN(20)
BANQ=0.000
BANCNT=0.
ONEPAS=1.
C
C IS THE NUMERATOR ZERO? IF YES, THEN RETURN. ELSE, MRC THE DENOMINATOR IF
C THIS IS THE FIRST PASS, OR MRC THE NUMERATOR .
C
IS
TEST=0.
I=I
DO WHILE(I.LE.NUMMOD)
   TEST=TEST+NUMRES(I)
   I=I+1
END DO
IF (TEST.EQ.0.) THEN
   RETURN
ELSE
   END IF
C
C Convert the denominator to mixed radix form, and bump the operation count
C
CALL CONMMIXED(DENRES, MREDEN, COUNT, TWODEM)
BANCNT=BANCNT+COUNT
C
C Find the most significant nonzero mixed radix digit
C
I=NUMMOD
DO WHILE(I.LE.1)
   IF(MREDEN(I).NE.0.) THEN
      MIDDEN=I
      GO TO 25
   ELSE
      END IF
      I=I+1
   END DO
C
C Do the denominator rounding. If digits of lesser significance are all zero,
C and the most significant digit is 1, then no rounding is done

25 TEST=0.
I=1
DO WHILE(I.LT.MSDDEN)
    TEST=TEST+RCDEN(I)
    I=I+1
END DO
IF(TEST.EQ.0.)AND.(RCDEN(MSDDEN).EQ.1.)THEN
    DRNDCD=RCDEN(MSDDEN)
ELSE
    DRNDCD=RCDEN(MSDDEN)+1.
END IF
DRPSA=0.
END IF

C Convert numerator to fixed radix form, and bump the count

30 CALL CONV(XNUM,RACNUM,COUNT,NUM)
BANCNT=BANCNT+CUNT

C Find the most significant nonzero digit of the numerator

C
I=NUMMOD
DO WHILE(.GE.1)
    IF(XNUM.GE.1) THEN
        MSDNUM=1
        GO TO 50
    ELSE
        END IF
    END DO

C Find the quotient estimate, stored in ZEYE. Count not increased if k=1.
C If k=1, count is increased by 1. Otherwise, count is increased by two

40 IF(MSDNUM.LT.MSDDEN)THEN
    RETURN
ELSE
    IF(MSDNUM.EQ.MSDDEN)THEN
        ZEYE=DOUBLE(INT(XNUM/MSDNUM)/DRNDCD))
        GO TO 50
    ELSE
        IF(MSDNUM.EQ.11)THEN
            ZEYE=DOUBLE(INT(XNUM/MSDNUM)*INT(MSDDEN/DRNDCD))
            BANCNT=BANCNT+1.
            GO TO 50
        ELSE
            SUB=RCDEN(MSDNUM)/(SYMDEN(MSDDEN)*RCDEN(MSDDEN))
            ZEYE=DOUBLE(INT(XNUM/MSDNUM)*INT(MSDDEN/DRNDCD)*SUB)
            BANCNT=BANCNT+2.
            END IF
        END IF
    END IF
END IF

C Update the running quotient sum, stored as BAND. If the quotient estimate
C is nonzero, then add it to the running sum, compute a new numerator, and
C increment count. If it is zero, then find the quotient correction in
C preparation for return
IF(IEYE .NE. 0.0D0) THEN
  BANCH = BANCH + 1.
ELSE
  THE QUOTIENT ESTIMATE WAS ZERO, SO WE MUST FIND THE CORRECTION TO THE SUM.
  I = NNUMMOD
  DO WHILE (I .GE. 1)
    IF (ARCHNUM(I) .GT. MQCDEN(I)) THEN
      SANS = SANS + I.D0
      BANCH = BANCH + 1.
      RETURN
    ELSE
      SANS = SANS + 0.
      BANCH = BANCH + 1.
      RETURN
    END IF
  END DO
  BANCH = BANCH + 1.
END IF
END

END
APPENDIX G

Derivation of Expressions for $P\left(\left|\frac{X}{Y}\right|<j\right)$ and $P(E_1=j \land \xi=n)$

This appendix contains two derivations and justification for the omission of the third term ($P(E_3=0 \land E_2=0 \land E_1=0)$ in Table 5. The first derivation is that of the probability distribution function of the truncated quotient, $P(\left|\frac{X}{Y}\right|<j)$, as given in Section E of Chapter III. The second derivation is that of the conjunctive probability density function $P(E_1=j \land \xi=n)$ for the OSRA initial estimate $E_1$, as given in Section E of Chapter III. In both derivations, we will use $S_\alpha$ to denote the sum of the first $\alpha$ natural numbers. That is,

$$S_\alpha = \frac{\alpha(\alpha+1)}{2}.$$

**Derivation 1:** We want to show that for $X$ and $Y \neq 0$ uniformly distributed in the ID[0,M-1], the probability distribution function of the truncated quotient $\left|\frac{X}{Y}\right|$ is

$$P(\left|\frac{X}{Y}\right|<j) = \frac{1}{2} + \sum_{i=1}^{j-1} \left( \frac{1}{M} \left( \left|\frac{M}{i}\right| - \left|\frac{M}{i+1}\right| \right) + \frac{i+1}{M(M-1)} S_{\left|\frac{M}{i+1}\right|} - \frac{i}{M(M-1)} S_{\left|\frac{M}{i}\right|} \right),$$

for $0<j<M-1$. The derivation will proceed by showing that the density function for the truncated quotient is
The derivation proceeds as follows:

The probability density functions for $X$ and $Y$ are

$$f_X(x) = \frac{1}{M} R_{M-1}(x)$$

and

$$f_Y(y) = \frac{1}{M-1} R_{M-2}(y-1),$$

where $R_\alpha(k)$ is defined as

$$R_\alpha(k) = \begin{cases} 
1, & \text{if } 0 < k < \alpha \\
0, & \text{otherwise}
\end{cases}$$

Since $X$ and $Y$ are independent, we have that the joint probability density function (henceforth abbreviated p.d.f.) of $X$ and $Y$ is

$$f_{X,Y}(x,y) = f_X(x)f_Y(y) = \frac{1}{M(M-1)} R_{M-1}(x)R_{M-2}(y-1).$$

Therefore, by the definition of the joint p.d.f., we have

$$P(|X/Y| < i) = \sum_{y=1}^{M-1} \sum_{x=0}^{(i+1)y-1} \left[ \frac{1}{M(M-1)} R_{M-1}(x)R_{M-2}(y-1) \right],$$

$$= \frac{1}{M(M-1)} \sum_{y=1}^{M-1} R_{M-2}(y-1) \sum_{x=0}^{(i+1)y-1} R_{M-1}(x).$$

277
where integer $i \in [0, M-1]$ and the upper limit on the $x$-summation comes from

$$\left\lfloor \frac{x}{y} \right\rfloor < i \Rightarrow x \cdot \text{modulo } y < y_i \Rightarrow x < y_i + x \cdot \text{modulo } y$$

$$< y_i + y - 1 = (i+1)y - 1.$$

We have

$$\sum_{x=0}^{(i+1)y-1} \mathcal{R}_{M-1}(x) = \begin{cases} (i+1)y & \text{if } (i+1)y - 1 < M-1 \\ M & \text{if } (i+1)y - 1 > M-1 \end{cases}$$

$$= y(i+1) - T((i+1)y - M),$$

where $T(\alpha)$ is the "discrete ramp"

$$T(\alpha) = \begin{cases} \alpha & \text{if } \alpha > 0 \\ 0 & \text{if } \alpha < 0 \end{cases}.$$

Therefore

$$P(\left\lfloor \frac{x}{y} \right\rfloor < i) = \frac{1}{M(M-1)} \sum_{y=1}^{M-1} (\mathcal{R}_{M-2}(y-1)) (y(i+1) - T((i+1)y - M))$$

$$= \frac{1}{M(M-1)} \sum_{y=1}^{M-1} y(i+1) - \frac{1}{M(M-1)} \sum_{y=1}^{M-1} T((i+1)y - M).$$

But

$$P(\left\lfloor \frac{x}{y} \right\rfloor = i) = P(\left\lfloor \frac{x}{y} \right\rfloor < i) - P(\left\lfloor \frac{x}{y} \right\rfloor < i - 1),$$

so

278
\[
P(\mid X/Y \mid = 1) = \frac{1}{M(M-1)} \left( \sum_{y=1}^{M-1} y + \sum_{y=1}^{M-1} (T(iy-M) - T((i+1)y-M)) \right)
\]
\[
= \frac{1}{M(M-1)} S_{M-1} + \frac{1}{M(M-1)} \sum_{y=1}^{M-1} (T(iy-M) - T((i+1)y-M))
\]

But, for \( i \neq 0 \) we have
\[
\sum_{y=1}^{M-1} (T(iy-M) - T((i+1)y-M)) = \sum_{y=1}^{M-1} (T(iy-M) - T((i+1)y-M))
\]
\[
= \frac{M}{T} \sum_{y=1}^{M-1} (T(iy-M) - T((i+1)y-M) + \sum_{y=1}^{M-1} (T(iy-M) - T((i+1)y-M))
\]
\[
= \frac{M}{T} \sum_{y=1}^{M-1} ((i+1)y-M) + \sum_{y=1}^{M-1} (iy-M-(i+1)y+M)
\]
\[
= M \left( \frac{M}{T} - \frac{M}{T+1} \right) - (i+1)(S_{M-1} - S_{M-1}) - S_{M-1} + S_{M-1}
\]

So, therefore
\[
P(\mid X/Y \mid = i) = \frac{1}{M(M-1)} S_{M-1} + \frac{1}{M-1} \left( \frac{M}{T} - \frac{M}{T+1} \right)
\]
\[
- \frac{i+1}{M(M-1)} \left( S_{M} - S_{M+1} \right) + \frac{M}{M(M-1) \left( M(M-1) \right)} + \frac{1}{M(M-1)} S_{M-1}
\]

279
Also, for $i=0$, we have

$$P(|X/Y| = 0) = \frac{1}{M(M-1)} S_{M-1} = \frac{1}{2}. $$

Clearly then, using this expression for $P(|X/Y| = i)$, we have

$$P(|X/Y| < j) = \frac{1}{2} + \sum_{i=1}^{j} (\frac{1}{M(M-1)} - \frac{i+1}{M(M-1)} S_{i+1} - \frac{i}{M(M-1)} S_{M})$$

which is the expression appearing in Section E of Chapter III.

**Derivation 2:** We want to show that the OSRA initial estimate probability density function $P(E_1 = j \land \lambda = n)$ is

$$P(E_1 = j \land \lambda = n) = \begin{cases} \frac{1}{m_n} (\frac{m_n}{i} - \frac{m_n}{i+1}) + \frac{1}{m_n^2} S_{i+1} \frac{m_n}{i} \\
- \frac{1}{m_n^2} S_{i} \frac{m_n}{i} - \frac{1}{m_n^2}, & 0 < j < \frac{m_n - 1}{2} \\
\frac{m_n+1}{2m_n} - \frac{1}{m_n^2}, & j=0 \end{cases}$$
The derivation proceeds as follows:

By the definition of conditional probability we have

$$P(E_1 = j \land z = n) = P(E_1 = j \land z = n)P(z = n) = P(E_1 = j \land z = n) \frac{m_n - 1}{m_n} ,$$

where we have used $P(z = n) = \frac{m_n - 1}{m_n}$ because $Y$ is uniformly distributed over $[1,M-1]$.

But, by the definition of the OSRA estimate,

$$P(E_1 = j \land z = n) = P\left(\left| \frac{x_n}{y_n + 1} \right| = j \right) ,$$

where, by assumption, $y_n$ is uniform on $[1,m_n - 1]$ and $x_n$ is uniform on $[0,m_n - 1]$. Letting $\beta = y_n + 1$, we have that $\beta$ is uniform on $[2,m_n]$, and we want the probability density function of $\left| \frac{x_n}{\beta} \right|$. This function is found using the same methods as those used in Derivation 1 of this appendix. The result is:

$$P\left(\left| \frac{x_n}{y_n + 1} \right| = j \right) = \begin{cases} \frac{1}{m_n - 1} \left( \left| \frac{m_n}{j} \right| - \left| \frac{m_n}{j+1} \right| \right) + \frac{j+1}{m_n(m_n-1)} S \left| \frac{m_n}{j+1} \right| \\ - \frac{j}{m_n(m_n-1)} S \left| \frac{m_n}{j} \right| - \frac{1}{m_n(m_n-1)} \\ \frac{m_n+2}{2m_n} \right) , \quad 0 < j < \left| \frac{m_n - 1}{2} \right| \\
\frac{m_n+2}{2m_n} , \quad j = 0 . \end{cases}$$

Therefore, we have the expression that appears in Section E of Chapter III.
The justification for not presenting a third term $P(E_3 = 0 \land E_2 \neq 0 \land E_1 \neq 0)$ in Table 5 is now considered. It is shown to be very inaccurate. To illustrate this, a derivation similar to that in Section E was used to find an approximate expression for $P(E_3 = 0 \land E_2 \neq 0 \land E_1 \neq 0)$, which is

$$P(E_3 = 0 \land E_2 \neq 0 \land E_1 \neq 0) = \frac{m_n - 1}{2m_n^{2}} \sum_{j=1}^{i} \sum_{j=1}^{i} P(|X_i - \kappa_i| = j + 1)P(|X_i - \kappa_i| < j + 1)P(E_1 = i \land \xi = n).$$

A computer program was written to evaluate this expression for the same modulus sets given in Table 5. The results, which can be considered to be approximations of the probability that the OSRA halts after making three estimates, are plotted in Figures 42 through 44. These values differ from the computationally derived values by as much as (or greater than) 50%, as can be seen in the figures. Therefore, we must conclude that the assumptions used in deriving the above approximation are not valid for numbers of estimates exceeding two.

Note that the above expression cannot be used to enhance the lower bound $\tilde{T}_{OSRA}$ because of its inaccuracy. Such an enhancement would be the expression
\[ \tilde{T}_{\text{OSRA}} = 2tP(E_1=0) + (3t+3)P(E_2=0 \land E_1\neq 0) \]
\[ + (4t+6)P(E_3=0 \land E_2\neq 0 \land E_1\neq 0) \]
\[ + (5t+9)(1-P(E_1=0) - P(E_2=0 \land E_1\neq 0) - P(E_3=0 \land E_2\neq 0 \land E_1\neq 0)), \]

which was found in the same way as used to find \( \tilde{T}_{\text{OSRA}} \). Because the term \( P(E_3=0 \land E_2\neq 0 \land E_1\neq 0) \) is low by such a large amount, the subsequent term \( (1-P(E_1=0) - P(E_2=0 \land E_1\neq 0) - P(E_3=0 \land E_2\neq 0 \land E_1\neq 0)) \) is too high. As a result, \( \tilde{T}_{\text{OSRA}} \) is not a lower bound.
APPENDIX H

Demonstration that an Overflow Modulus is Not Required by the RA when $m_{n-1} = m_{n-1}$

This appendix shows that an overflow modulus is not required for the RA if the two largest moduli $m_n$ and $m_{n-1}$ differ by one. Also, it shows that for all other modulus sets, only a single overflow modulus $m_{n+1} = m_{n-1}$ is needed, and only when $k=n$. These facts will be shown as follows. First, the condition $k=n$ is shown to be the only one under which overflow can occur. Then, it is shown that the overflow can be prevented by the use of the single overflow modulus $m_{n+1} = m_{n-1}$, which is used only for modulus sets which don't satisfy $m_{n-1} = m_{n-1}$.

To begin the discussion, for $k+1$, we have that the RA estimate

$$E_{i+1} = x_k \left| \frac{p_k}{m_k} \right| m_{k+1} \cdots m_{k-1} m_{n-1} < m_{k} m_{n-1} \cdots m_{n-1},$$

and obviously overflow cannot occur. Overflow cannot occur for $k=n+1$ either, because in this case we have

$$E_{i+1} = x_k \left| \frac{p_k}{m_k} \right| < m_k m_n.$$

However, for $k=n$, the RA computes

$$E_{i+1} = x_k \left| \frac{p_k}{m_k} \right|,$$

and it is possible that the interim product $x_k \left| \frac{p_k}{m_k} \right|$ will overflow. For $k=n$, we have
\[ x_k \left| \frac{p_x}{T} \right| < (m_k - 1)^2 , \]

since the case \( \left| \frac{p_x}{T} \right| = m_k \) is treated differently (see Section A.2.1 of Chapter IV). Clearly in this case there is no overflow. But if \( k = \ell = n \), we have

\[ x_k \left| \frac{p_x}{T} \right| < (m_n - 1)^2 , \]

and this quantity can overflow for modulus sets for which \( p_{n-1} < m_{n-2} \). Consequently, at least one overflow modulus must be employed only for these modulus sets, and only for the case \( k = \ell = n \).

The overflow problem can be solved by the use of a single overflow modulus \( m_{n+1} = m_n - 1 \), which need be used only for modulus sets such that \( m_{n-1} \neq m_n - 1 \). This is because \( m_n \) and \( m_{n+1} \) are relatively prime and

\[ x_k \left| \frac{p_x}{T} \right| < (m_n - 1)^2 < m_n m_{n+1} . \]

The overflow modulus need be used only when \( k = \ell = n \), and is used to calculate \( E_{i+1} \) as follows. The product \( x_k \left| \frac{p_x}{T} \right| \) is computed in all moduli including \( m_{n+1} \). The result of scaling this product by \( m_n \) will be contained solely in \( m_{n+1} \), since

\[ E_{i+1} = \left| x_k \left| \frac{p_x}{T} \right| \right| < \left| \frac{(m_n - 1)(m_n - 1)}{m_n} \right| < m_n - 2 < m_{n+1} . \]

The \( n^{th} \) residue digit of \( E_{i+1} \) is restored from the overflow digit (i.e., the \((n+1)^{st}\)) using a zero-operation base extension. If \( k = \ell < n \), then \( m_{n+1} \) is not used because, as shown above, overflow cannot occur. For such
cases, the restoration of the $k^{th}$ residue digit is done using the $n^{th}$ digit, since for $k \leq n$,

$$E_{i+1} = \left\lfloor \frac{x_k}{m_k} \right\rfloor \leq \frac{(m_k-1)(m_k-1)}{m_k} < m_k - 2 < n .$$

Note that for modulus sets which do not require the overflow modulus, restoration of the $n^{th}$ digit of $E_{i+1}$ when $k = n$ is done using the $(n-1)^{st}$ digit.
APPENDIX I
Proofs of Lemmas 9 and 10

This appendix contains the proofs of Lemmas 9 and 10 from Chapter IV.

Lemma 9: For the RA estimate, viz.,

\[ E_{i+1} = \left\{ \begin{array}{ll}
\left( x_k \frac{p_k}{m_k} \right) & \text{if } k = 2 \\
\left( x_k \frac{p_k}{Y} \right) & \text{if } k = 2 + 1 \\
\left( x_k \frac{p_k}{Y} \right) m_{k+1} \cdots m_{k-1} & \text{if } k > 2 + 1
\end{array} \right. \]

we have

\[ E_{i+1} = 0 \Rightarrow x_1 < 2Y. \]

Proof: We have \( E_{i+1} = 0 \Rightarrow k = 2 \), obviously. Therefore, we must show that

\[ \left( x_k \frac{p_k}{m_k} \right) = 0 \Rightarrow x_1 < 2Y. \]

By assumption, we have

\[ x_i = x_k p_{k-1} + x_{k-1} p_{k-2} + \cdots + x_1 \]

and

237
\[ Y = y_1^p + y_2^p + \cdots + y_1 \]

since \( k = 2 \). Furthermore, since \( p_{k-1} < Y < p_k-1 \), we have

\[ 1 < \left\lfloor \frac{p_k}{Y} \right\rfloor < m_k. \]

We will first show that \( \left\lfloor \frac{p_k}{Y} \right\rfloor = j \Rightarrow \left\lfloor \frac{p_k}{Y} \right\rfloor + 1 < Y \), for \( j \in \{1, m_k\} \). This is because

\[ \left\lfloor \frac{p_k}{Y} \right\rfloor = j \Rightarrow jY < p_k < (j+1)Y-1 \Rightarrow \frac{p_k}{j+1} < Y - \frac{1}{j+1} \]

\[ \Rightarrow \frac{p_k}{j+1} + \frac{1}{j+1} < Y \Rightarrow \left\lfloor \frac{p_k}{j+1} \right\rfloor + 1 < Y, \]

since \( Y \) is an integer. Now let \( \left\lfloor \frac{p_k}{Y} \right\rfloor = j \), for \( j \in \{1, m_k\} \). We have

\[ E_{i+1} = 0 \Rightarrow \left\lfloor \frac{x_j^k}{m_{k-1}} \right\rfloor = 0 \Rightarrow x_j^k < m_{k-1} \Rightarrow j \not\equiv m_k. \]

Also, for \( j \in \{1, m_k-1\} \),

\[ x_j^k < \begin{cases} \left\lfloor \frac{m_k}{j} \right\rfloor, & \text{if } \frac{m_k}{j} \text{ is not an integer} \\ \left\lfloor \frac{m_k}{j} \right\rfloor - 1, & \text{if } \frac{m_k}{j} \text{ is an integer,} \end{cases} \]

\[ \Rightarrow \left\lfloor \frac{m_k}{j} \right\rfloor = \left\lfloor \frac{m_k}{j} \right\rfloor - 1 \]

\[ \Rightarrow x_j^k < \left( \left\lfloor \frac{m_k}{j} \right\rfloor - 1 \right)p_k-1 + p_k-1, \text{ if } \frac{m_k}{j} \text{ is not an integer,} \]

\[ \Rightarrow x_j^k < \left( \left\lfloor \frac{m_k}{j} \right\rfloor - 1 \right)p_k-1 + p_k-1 - 1, \text{ if } \frac{m_k}{j} \text{ is an integer,} \]

\[ \Rightarrow x_j^k < \left( \left\lfloor \frac{m_k}{j} \right\rfloor - 1 \right)p_k-1 - 1, \text{ if } \frac{m_k}{j} \text{ is not an integer} \]

\[ \Rightarrow x_j^k < \left( \left\lfloor \frac{m_k}{j} \right\rfloor - 1 \right)p_k-1, \text{ if } \frac{m_k}{j} \text{ is an integer,} \]

\[ \Rightarrow x_j^k < \left\lfloor \frac{m_k}{j} \right\rfloor p_k-1 - 1, \text{ if } \frac{m_k}{j} \text{ is an integer}, \]

\[ \Rightarrow x_j^k < \left\lfloor \frac{m_k}{j} \right\rfloor p_k-1, \text{ if } \frac{m_k}{j} \text{ is an integer}, \]
where \(|-\alpha|\) denotes the least integer greater than or equal to \(\alpha\).

We will show that \(X_j < 2Y\) by showing that \(jX_j < 2jY\) for \(j \in [1, m_k-1]\).

We want to determine if

\[
jX_j < j \left\lfloor \frac{m_k}{j} \right\rfloor p_{k-1} - j < 2j \left\lfloor \frac{p_k}{j+1} \right\rfloor + 2j < 2jY.
\]

But \(j \left\lfloor \frac{m_k}{j} \right\rfloor p_{k-1} - j = (m_k + |m_k| j)p_{k-1} - j = p_k + p_{k-1} - m_k j - j\), where \(|\alpha|_j\) denotes the quantity \(\alpha\) modulo \(j\). Furthermore,

\[
2j \left\lfloor \frac{p_k}{j+1} \right\rfloor + 2j = (j+1) \left\lfloor \frac{p_k}{j+1} \right\rfloor + (j-1) \left\lfloor \frac{p_k}{j+1} \right\rfloor + 2j
\]

\[
= p_k - |p_k|_{j+1} + (j-1) \left\lfloor \frac{p_k}{j+1} \right\rfloor + 2j.
\]

So we have

\[
p_k + p_{k-1} - m_k j - j < p_k - |p_k|_{j+1} + (j-1) \left\lfloor \frac{p_k}{j+1} \right\rfloor + 2j
\]

\[
\iff p_{k-1} - m_k j < (j-1) \left\lfloor \frac{p_k}{j+1} \right\rfloor + 3j - |p_k|_{j+1}.
\]

But \(3j - |p_k|_{j+1} > 0\) for \(j \in [1, m_k-1]\), and \(|-m_k| j < j-1\) for \(j \in [1, m_k-1]\). So we must show \(p_{k-1} < \left\lfloor \frac{p_k}{j+1} \right\rfloor\), for \(j \in [1, m_k-1]\). But \(\left\lfloor \frac{p_k}{j+1} \right\rfloor = \left\lfloor \frac{m_k}{j+1} \right\rfloor p_{k-1} > p_{k-1}\), for \(j \in [1, m_k-1]\). Therefore, \(X_j < 2Y\).

QED
Lemma 10: For the RA estimate

\[
E_{i+1} = \begin{cases} 
\frac{x_k}{m_k} - \frac{p_k}{m_k}, & \text{if } k = 2 \\
\frac{x_k}{m_k} - \frac{p_k}{m_k}, & \text{if } k = 2+1 \\
\frac{x_k}{m_k} - \frac{p_k}{m_k} - m_{k+1} \ldots m_{k-1}, & \text{if } k > 2+1, 
\end{cases}
\]

we have \(0 < E_{i+1} < X_i/Y\), for all \(X_i, Y \neq 0\).

Proof: Clearly, \(E_{i+1} > 0\) in each case \(k=2\), \(k=2+1\) and \(k>2+1\). Now by assumption,

\[
X_i = x_k P_{k-1} + x_{k-1} P_{k-2} + \ldots + x_1
\]

and

\[
Y = y_{k-1} P_{k-2} + y_{k-2} P_{k-3} + \ldots + y_1.
\]

We will show that \(E_{i+1} < X_i/Y\) for each case \(k=2\), \(k=2+1\) and \(k>2+1\).

For \(k=2\):

\[
E_{i+1} = \frac{x_k}{m_k} - \frac{p_k}{m_k} = \frac{x_k P_{k-1} - p_k}{m_k}.
\]

Let \(x_k P_{k-1} = X_i - \varepsilon\), for \(0 < \varepsilon < P_{k-1}\). Also, let \(|P_k| Y\) denote \(P_k \mod Y\). We have

\[
E_{i+1} = \frac{(X_i-\varepsilon) - P_k}{P_k} = \frac{(X_i-\varepsilon) P_k}{P_k Y} = \frac{(X_i-\varepsilon)(P_k - |P_k| Y)}{P_k Y}.
\]

\[
= \frac{X_i}{Y} - \frac{\varepsilon}{P_k Y} - \frac{(X_i-\varepsilon)|P_k| Y}{P_k Y}.
\]

490
Now $Y > P_{k-1}$, so $0 < \frac{e}{Y} < 1$. Also, $0 < \frac{|P_k|Y}{y} < 1$, and $\frac{x_{i-e}}{p_k} = \frac{x_k}{m_k} < 1$.

Therefore,

$$0 < \frac{(X_i - e)|P_k|Y}{p_k} < 1.$$ 

Therefore,

$$E_{i+1} < \left| \frac{x_i}{Y} \right| < \frac{x_i}{Y}.$$ 

For $k=2+1$: $E_{i+1} = x_k \left| \frac{P_l}{Y} \right| = x_k \left| \frac{P_{k-1}}{Y} \right| = \frac{x_k P_{k-1}}{Y} < \frac{x_k}{Y} < \frac{x_i}{Y}.$

For $k>2+1$: $E_{i+1} = x_k \left| \frac{P_l}{Y} \right| m_{k+1} \cdots m_{k-1} \leq x_k \frac{P_l}{Y} \frac{m_{k+1} \cdots m_{k-1}}{m_k} = \frac{x_k P_{k-1}}{Y} < \frac{x_i}{Y}.$

Therefore, in each case $k=2$, $k=2+1$, and $k>2+1$, we have

$$0 < E_{i+1} < \frac{x_i}{Y}.$$ 

QED
APPENDIX J

Derivation of the Size of Table $T_k$, Used by the RA

This appendix contains a derivation of an expression for $s$, the size of Table $T_k$, used by the RA.

Table $T_k$, used in the RA has size

$$ s = \begin{cases} m_k - 1, & \text{if } m_k < P_{k-1} \\ 2 \left\lfloor \frac{P_k}{2} \right\rfloor - P_{k-1}, & \text{if } m_k > P_{k-1} \text{ and } \left\lfloor \frac{P_k}{2} \right\rfloor + \left\lfloor \frac{P_k}{2} \right\rfloor = \left\lfloor \frac{P_k}{2} \right\rfloor \\ 2 \left\lfloor \frac{P_k}{2} \right\rfloor - P_{k-1} - 1, & \text{if } m_k > P_{k-1} \text{ and } \left\lfloor \frac{P_k}{2} \right\rfloor = \left\lfloor \frac{P_k}{2} \right\rfloor \\ \end{cases} $$

Proof: We will first find $s$ for $T_1$, and then show how this result can be generalized for $k > 1$.

For $k=1$, we have $1 \leq j \leq m_1 - 1$. Consider the sequence

$$ \left\lfloor \frac{m_1}{j} \right\rfloor, \left\lfloor \frac{m_1}{j+1} \right\rfloor, \ldots, \left\lfloor \frac{m_1}{\sqrt{m_1}} \right\rfloor, \left\lfloor \sqrt{m_1} \right\rfloor, \left\lfloor \sqrt{m_1} \right\rfloor - 1, \ldots, 2, 1. $$

We can deduce several of its properties:

1) $\left\lfloor \frac{m_1}{j} \right\rfloor > \left\lfloor \frac{m_1}{j+1} \right\rfloor$ for $j \in [1, \sqrt{m_1}-1].$

This is because $\left\lfloor \frac{m_1}{j} \right\rfloor > \left\lfloor \frac{m_1}{j+1} \right\rfloor$

$$ \Leftrightarrow \left\lfloor \frac{m_1(j+1)}{j(j+1)} \right\rfloor > \left\lfloor \frac{m_1j}{j(j+1)} \right\rfloor $$

$$ \Leftrightarrow \left\lfloor \frac{m_1j}{j(j+1)} + \frac{m_1}{j(j+1)} \right\rfloor > \left\lfloor \frac{m_1j}{j(j+1)} \right\rfloor.$$

and $j < \sqrt{m_1} - 1 \Rightarrow j < m_1 - 1 \Rightarrow j(j+1) < (\sqrt{m_1} - 1)\sqrt{m_1} < m_1$. 

292
2) \[ \left| \frac{m_1}{\sqrt{m_1}} \right| > \left| \sqrt{m_1} \right|, \]

because \[ \left| \frac{m_1}{\sqrt{m_1}} \right| > \left| \frac{m_1}{\sqrt{m_1}} \right| = \left| \sqrt{m_1} \right|. \]

3) For \( j \in [1, \left| \sqrt{m_1} \right| - 1] \), we have

\[ \left| \frac{m_1}{j} \right| > Y > \left| \frac{m_1}{j+1} \right| + 1 \Rightarrow \left| \frac{m_1}{Y} \right| = j. \]

This is because \[ \left| \frac{m_1}{j+1} \right| < \left| \frac{m_1}{Y} \right| < \left| \frac{m_1}{j+1} \right| + 1 \].

Using \( |a|_\beta \) to denote the quantity \( a \) modulo \( \beta \), we have

\[ \left| \frac{m_1}{j} \right| = \left| \frac{j m_1}{m_1 - \lfloor m_1 \rfloor j} \right| > j, \text{ and } \left| \frac{m_1}{j+1} \right| = \left| \frac{m_1 (j+1)}{m_1 + (j+1) - \lfloor m_1 \rfloor j+1} \right| < j. \]

Therefore, \( j < \left| \frac{m_1}{Y} \right| < j. \)

4) \[ \left| \frac{m_1}{\sqrt{m_1}} \right| > \left| \sqrt{m_1} \right| \text{ and } \left| \frac{m_1}{\sqrt{m_1}} \right| > Y > \left| \sqrt{m_1} \right| + 1 \]

\[ \Rightarrow \left| \frac{m_1}{Y} \right| = \left| \sqrt{m_1} \right|. \text{ This is because } \left| \frac{m_1}{\sqrt{m_1}} \right| < \left| \frac{m_1}{\sqrt{m_1}} \right| + 1, \text{ and } \left| \frac{m_1}{\sqrt{m_1}} \right| > \left| \frac{m_1}{\sqrt{m_1}} \right| \]

\[ \left| \frac{m_1}{\sqrt{m_1}} \right| = \left| \sqrt{m_1} \right|. \]

Furthermore, \( \sqrt{m_1} \) cannot be an integer because, if
it were, $\left\lfloor \frac{m_1}{\sqrt{m_1}} \right\rfloor = \left\lfloor \frac{m_1}{\sqrt{m_1}} \right\rfloor = \left\lfloor \frac{m_1}{\sqrt{m_1}} \right\rfloor$ which cannot happen by assumption. Therefore,

$$\left\lfloor \frac{m_1}{\sqrt{m_1}} + 1 \right\rfloor = \left\lfloor \frac{m_1}{\sqrt{m_1}} \right\rfloor < \left\lfloor \frac{m_1}{\sqrt{m_1}} \right\rfloor = \left\lfloor \frac{m_1}{\sqrt{m_1}} \right\rfloor.$$

Therefore, $\left\lfloor \frac{m_1}{\sqrt{m_1}} \right\rfloor < \left\lfloor \frac{m_1}{\sqrt{m_1}} \right\rfloor < \left\lfloor \frac{m_1}{\sqrt{m_1}} \right\rfloor$.

5) For $1 < Y_1, Y_2 < \left\lfloor \frac{m_1}{\sqrt{m_1}} \right\rfloor$ we have

$$\left\lfloor \frac{m_1}{Y_1} \right\rfloor = \left\lfloor \frac{m_1}{Y_2} \right\rfloor \Rightarrow Y_1 = Y_2.$$

This is because, by 1), integers $\left\lfloor \frac{m_1}{Y} \right\rfloor$ for $j \in \mathbb{Z}, \left\lfloor \frac{m_1}{\sqrt{m_1}} \right\rfloor$ are unique.

Based on these properties 1) thru 5) we can conclude that the ordered pairs $(Y_j, j)$ stored in $T_1$, must be either

$$(\left\lfloor \frac{m_1}{1} \right\rfloor, 1), (\left\lfloor \frac{m_1}{2} \right\rfloor, 2), \ldots, (\left\lfloor \frac{m_1}{\sqrt{m_1}} \right\rfloor, \left\lfloor \frac{m_1}{\sqrt{m_1}} \right\rfloor),$$

$$(\left\lfloor \frac{m_1}{\sqrt{m_1}} \right\rfloor, \left\lfloor \frac{m_1}{\sqrt{m_1}} \right\rfloor), (\left\lfloor \frac{m_1}{\sqrt{m_1}} - 1 \right\rfloor, \left\lfloor \frac{m_1}{\sqrt{m_1}} - 1 \right\rfloor)$$

... (1, 1)

if $\left\lfloor \frac{m_1}{\sqrt{m_1}} \right\rfloor \neq \left\lfloor \frac{m_1}{\sqrt{m_1}} \right\rfloor$,

or

$$(\left\lfloor \frac{m_1}{1} \right\rfloor, 1), (\left\lfloor \frac{m_1}{2} \right\rfloor, 2), \ldots, (\left\lfloor \frac{m_1}{\sqrt{m_1}} - 1 \right\rfloor, \left\lfloor \frac{m_1}{\sqrt{m_1}} - 1 \right\rfloor),$$

$$(\left\lfloor \frac{m_1}{\sqrt{m_1}} \right\rfloor, \left\lfloor \frac{m_1}{\sqrt{m_1}} \right\rfloor), (\left\lfloor \frac{m_1}{\sqrt{m_1}} - 1 \right\rfloor, \left\lfloor \frac{m_1}{\sqrt{m_1}} - 1 \right\rfloor), \ldots, (1, 1)$$

if $\left\lfloor \frac{m_1}{\sqrt{m_1}} \right\rfloor = \left\lfloor \frac{m_1}{\sqrt{m_1}} \right\rfloor.$
This is because of 2 facts: a) By 1) and 2), both sequences are decreasing.

b) By 3), 4) and 5), each $Y_j$ is the largest $Y$ such that $\frac{m_1}{Y} = j_i$.

Since the first ordered pair need not be stored, we have

$$s = \begin{cases} 
2\sqrt{m_1} - 1, & \text{if } \frac{m_1}{\sqrt{m_1}} \neq \sqrt{m_1} \\
2\sqrt{m_1} - 2, & \text{if } \frac{m_1}{\sqrt{m_1}} = \sqrt{m_1} 
\end{cases}$$

for the size of $T_1$.

For the general table $T_k$, $k \neq 1$, we replace $m_1$ with $P_k$, and since $Y > P_{k-1}$ we must store an "initial segment" of the ordered pairs

$$(\left\lfloor \frac{P_k}{1} \right\rfloor,1), (\left\lfloor \frac{P_k}{2} \right\rfloor,2), \ldots, (\left\lfloor \frac{P_k}{\sqrt{P_k} - 1} \right\rfloor, \sqrt{P_k} - 1), \ldots,$$

$$(1,\left\lfloor \frac{P_k}{1} \right\rfloor)$$

if $\frac{P_k}{\sqrt{P_k} - 1} \neq \sqrt{P_k}$,

or

$$(\left\lfloor \frac{P_k}{1} \right\rfloor,1), (\left\lfloor \frac{P_k}{2} \right\rfloor,2), \ldots, (\left\lfloor \frac{P_k}{\sqrt{P_k} - 1} \right\rfloor, \sqrt{P_k} - 1), \ldots,$$

$$(1,\left\lfloor \frac{P_k}{1} \right\rfloor)$$

if $\frac{P_k}{\sqrt{P_k} - 1} = \sqrt{P_k}$.
If \( m_2 < \sqrt{|P_2|} \), then clearly our initial segment is

\[
\left( \left\lfloor \frac{P_2}{1} \right\rfloor, 1 \right), \left( \left\lfloor \frac{P_2}{2} \right\rfloor, 2 \right), \ldots, \left( \left\lfloor \frac{P_2}{m_2} \right\rfloor, m_2 \right),
\]

for which \( s = m_2 - 1 \). Otherwise, if \( m_2 > \sqrt{|P_2|} \), the initial segment is

\[
\left( \left\lfloor \frac{P_2}{1} \right\rfloor, 1 \right), \left( \left\lfloor \frac{P_2}{2} \right\rfloor, 2 \right), \ldots, \left( \left\lfloor \frac{P_2}{\sqrt{|P_2|}} \right\rfloor, \sqrt{|P_2|} \right),
\]

\[
\left( \left\lfloor \sqrt{|P_2|} \right\rfloor, \left\lfloor \sqrt{|P_2|} \right\rfloor \right), \left( \left\lfloor \sqrt{|P_2|} \right\rfloor - 1, \left\lfloor \sqrt{|P_2|} \right\rfloor - 1 \right), \ldots,
\]

\((P_{2-1, m_2})\),

\[
\text{if } \left\lfloor \frac{P_2}{\sqrt{|P_2|}} \right\rfloor \neq \sqrt{|P_2|},
\]

or

\[
\left( \left\lfloor \frac{P_2}{1} \right\rfloor, 1 \right), \left( \left\lfloor \frac{P_2}{2} \right\rfloor, 2 \right), \ldots, \left( \left\lfloor \frac{P_2}{\sqrt{|P_2|}} \right\rfloor - 1, \left\lfloor \sqrt{|P_2|} \right\rfloor - 1 \right),
\]

\[
\left( \left\lfloor \sqrt{|P_2|} \right\rfloor, \left\lfloor \sqrt{|P_2|} \right\rfloor \right), \left( \left\lfloor \sqrt{|P_2|} \right\rfloor - 1, \left\lfloor \sqrt{|P_2|} \right\rfloor - 1 \right), \ldots,
\]

\((P_{2-1, m_2})\),

\[
\text{if } \left\lfloor \frac{P_2}{\sqrt{|P_2|}} \right\rfloor = \sqrt{|P_2|}.
\]

In this case, \( s = \sqrt{|P_2|} - 1 + \sqrt{|P_2|} - (P_{2-1} - 1) = 2\sqrt{|P_2|} - P_{2-1} \),

\[
\text{if } \left\lfloor \frac{P_2}{\sqrt{|P_2|}} \right\rfloor \neq \sqrt{|P_2|},
\]

or

296
So,

\[ s = |\sqrt{P_k}| - 1 + |\sqrt{P_k}| - 1 - (P_k - 1) = 2|\sqrt{P_k}| - P_k - 1, \]

if \[ \left| \frac{P_k}{\sqrt{P_k}} \right| = \left| \sqrt{P_k} \right| \]

So,

\[ s = \begin{cases} 
2|\sqrt{P_k}| - P_k - 1, & \text{if } \left| \frac{P_k}{\sqrt{P_k}} \right| \neq \left| \sqrt{P_k} \right| \\
2|\sqrt{P_k}| - P_k - 1 - 1, & \text{if } \left| \frac{P_k}{\sqrt{P_k}} \right| = \left| \sqrt{P_k} \right| 
\end{cases} \]

if \( m_k > |\sqrt{P_k}| \).

We can replace the condition \( m_k < |\sqrt{P_k}| \) with \( m_k < P_k - 1 \) because \( m_k < |\sqrt{P_k}| \Rightarrow m_k < \sqrt{P_k} \Rightarrow m_k^2 < P_k \Rightarrow m_k < P_k - 1 \), and \( m_k < P_k - 1 \Rightarrow m_k^2 < P_k \Rightarrow m_k < |\sqrt{P_k}| \). Therefore, finally, we obtain that the size of \( T_k \) is

\[ s = \begin{cases} 
m_k - 1, & \text{if } m_k < P_k - 1 \\
2|\sqrt{P_k}| - P_k - 1, & \text{if } \left| \frac{P_k}{\sqrt{P_k}} \right| \neq \left| \sqrt{P_k} \right| \text{ and } m_k > P_k \\
2|\sqrt{P_k}| - P_k - 1 - 1, & \text{if } \left| \frac{P_k}{\sqrt{P_k}} \right| = \left| \sqrt{P_k} \right| \end{cases} \]

QED
APPENDIX K

The Simulator Program for the RA and the OSRA

APPENDIX K: THE SIMULATOR PROGRAM FOR THE RA AND THE OSRA

Note: In this program, the OSRA is referred to as the Banerji-I Algorithm.

C THIS PROGRAM SIMULATES THE RECIPROCAL ALGORITHM (RA) AND THE SANERJI-I
C ALGORITHM (BIA). INPUT FOR THE PROGRAM IS FILE FOR002.DAT. OUTPUT IS
C FILE FOR003.DAT
C
C VARIABLE NAMES ARE AS FOLLOWS:
C M: PRODUCT OF SYSTEM MODULI
C NUM: NUMERATOR IN DECIMAL
C DEN: DENOMINATOR IN DECIMAL
C Q: QUOTIENT IN DECIMAL
C MRCDEN(I): THE MIXED RADIX COEFFICIENTS IN DECIMAL, IN INCREASING ORDER
C SYSMOD(I): THE SYSTEM MODULI, IN INCREASING ORDER
C NUMMOD: THE NUMBER OF MODULI
C DENRES(I): THE ITH RESIDUE DIGIT OF THE QUOTIENT
C QNRES(I): THE ITH RESIDUE DIGIT OF THE DENOMINATOR
C MRCDEN(I): THE MIXED RADIX DIGITS OF THE DENOMINATOR, IN INCREASING ORDER
C MRCNUM(I): THE MIXED RADIX DIGITS OF THE NUMERATOR, IN INCREASING ORDER
C ESTQ(I): THE ITH RESIDUE DIGIT OF THE QUOTIENT ESTIMATE IN SOME ITERATION
C ESTCAL: FLAG WHICH EXCEEDS ZERO ONLY WHEN AT LEAST ONE QUOTIENT ESTIMATE
C HAS BEEN MADE
C SCATER(I): THE NUMBER OF DIVISION PROBLEMS WHICH REQUIRED EXACTLY 1 RESIDUE
C OPERATIONS TO SOLVE USING THE RECIPROCAL ALGORITHM
C SCABAN(I): SAME AS ABOVE, EXCEPT FOR THE BIA
C DENBAN(I): THE DENOMINATOR RESIDUES FOR THE BIA
C BAND: THE QUOTIENT, IN DECIMAL, IN THE BIA
C NUMSET: THE NUMBER OF MODULUS SETS TO BE SIMULATED
C NUMSAH: THE NUMBER OF DIVISION PROBLEMS TO BE SIMULATED WITHIN A MODULUS SET
C SAMSN: THE AVERAGE RUNNING TIME FOR THE RA, FOUND FROM A PREVIOUS RUN
C OF THE SIMULATOR. THIS NUMBER IS USED ONLY FOR VARIANCE CALCULATIONS
C SAMBA: SAME AS ABOVE, EXCEPT FOR THE BIA.
C SAMDI: THE SAMPLE AVERAGE DIFFERENCE IN RUNNING TIMES, FOUND AS BIA TIME
C MINUS RA TIME, FOUND FROM A PREVIOUS RUN OF THE SIMULATOR. USED
C FOR VARIANCE CALCULATIONS ONLY
C NEVERY: THE SAMPLE AVERAGES ARE COMPUTED FOR SAMPLE SIZES WHICH ARE MULTIPLES
C OF THIS NUMBER
C
C DATA INPUT TO THE SIMULATOR MUST BE AS FOLLOWS:
C
C LINE 1: THE NUMBER OF MODULUS SETS TO BE SIMULATED
C LINE 2: THE NUMBER OF DIVISION PROBLEMS TO BE SIMULATED FOR THE FIRST
C MODULUS SET
C LINE 3: THE NUMBER OF MODULI IN THE SET
C LINE 4: THE MODULI IN THE SET, ORDERED LEAST FIRST TO GREATEST LAST.
C LINE 5: THE FOLLOWING THREE REAL NUMBERS, IN THIS ORDER AND SEPARATED
C BY COMMAS: SAMSN, SAMBA, SAMDI. THESE ARE DEFINED ABOVE
C LINE 6: AVERAGE RUNNING TIMES AND VARIANCES ARE COMPUTED FOR SAMPLE
C SIZES WHICH ARE MULTIPLES OF THIS NUMBER.
C
C LINES 2 THRU 6 MUST BE REPEATED, IN ORDER, FOR EACH MODULUS SET TO BE
SIMULATED

REAL* NUM, DEN, Q, NRCCOE(20)
COMMON SYMNUM, SYMINV(96, 20), NRCCOE
REAL NUMNSET, DENNSET(20), QUOT(20), NRCCOE(20), NRCHNUM(20), ESTRES(20)
INTEGER ESTCAL
REAL SCATTER(500), NUMBER(20), SCABAN(500)
READ* REAL* ESTMA
OPEN (UNIT=2, FILE="STUDENT_DISK48EGCHRENJFOR002.DAT" , STATUS="OLD")

INPUT DATA AND INITIALIZE SET COUNTER

READ(2,*) NUMSET
ID=1
DO WHILE(JJ.LE.NUMSET)
READ(2,*) NUMSAM
READ(2,*) NUMMOD
READ(2,*) SYMMOD(I)+1, NUMMOD
READ(2,*) SCATR(I), SCABAN(I), SCABAN(I)
READ(2,*) NUMBER(I), NEVERT

OUTPUT NUMBER OF DATA POINTS AND SIZE OF OPCOUNT ARRAYS

NSIZE=500
NUMPTS=NUMSAM/NEVERT
WRITE(*,*) NUMPTS, NSIZE, NEVERT

INITIALIZE OPCOUNT ARRAYS

DO 13 I=1,500
SCATR(I)=0.
SCABAN(I)=0.
13 CONTINUE

COMPUTE DYNAMIC RANGE FOR THIS NUMULUS SET

=1,000
DO 10 I=1,NUMMOD
=MOD(*,(SYMMOD(I)))
10 CONTINUE

LOAD TABLE OF INVERSES MODULO THE SYSTEM MODULI. THIS CODE WILL COMPUTE
A VALUE OF ZERO FOR THE INVERSE OF INTEGERS THAT ARE NOT RELATIVELY PRIME
TO THE MODULUS

DO 20 I=1,NUMMOD
=SYMMOD(I)-1.
20 CONTINUE
DO 30 J=1,4
DO 40 K=1,2
DO WHILE(JMODO(J).LT.INT(SYMMOD(I))).EQ.1
SYMINV(J,I)=I
GO TO 30
40 CONTINUE
30 CONTINUE
CONTINUE
C LOAD THE ARRAY OF MIXED RADIUS COEFFICIENTS
C
C =RCCOECE(1) = 0.000
J = 1
DO WHILE(J .LE. NUMMOD)
   RCCOECE(J) = RCCOECE(J-1) * MODLE(SYSMOD(J-1))
   J = J + 1
END DO
C SET THE RANDOM NUMBER GENERATOR SEED
C ISEED = 6069
C
C INITIALIZE J, WHICH IS THE SAMPLE COUNTER, AND ALSO INITIALIZE OTHER
C SIMULATOR VARIABLES
C
J = 1
P0 = 0.
V0 = 0.
R0 = 0.
B0 = 0.
D0 = 0.
DIF0 = 0.
VARP0 = 0.

C DONE ENOUGH SAMPLES?
C DO WHILE(J .LE. NUMSAM)
C INITIALIZE OPERATION COUNTER
C
C = 0
C RANDOMLY SELECT A NUMERATOR AND DENOMINATOR IN RESIDUE. COPY THEM FOR
C LATER USE BY THE BIA SIMULATOR
C
J = 1
DO WHILE(J .LE. NUMMOD)
   NUMRES(J) = INT(SYSMOD(J) * RAN(ISEED))
   DENRES(J) = INT(SYSMOD(J) * RAN(ISEED))
   J = J + 1
END DO
J = 1
DO WHILE(J .LE. NUMMOD)
   NUMBAN(J) = NUMRES(J)
   DENBAN(J) = DENRES(J)
   J = J + 1
END DO
C CONVERT TO INTEGER FORM FOR VARIOUS PURPOSES WHICH ARE MORE EASILY DONE
C IN DECIMAL. TEST TO SEE IF THE DENOMINATOR HAPPENS TO BE ZERO. IF SO,
C GO GET ANOTHER DENOMINATOR
C
CALL CONINT(NUMRES, N, NUM)
CALL CONINT(DENRES, N, DEN)
IF(DEN.EQ.0.0D0)THEN
   WRITE(*,*)"ZERO DEN!"
   GO TO 45
ELSE
   END IF
C
C INITIALIZE THE QUOTIENT
C
J=1
DO WHILE(J.LE.NUMMOD)
   Q(J)=0.
   J=J+1
END DO
C
C INITIALIZE THE ESTIMATE COUNTER
C
ESTCAL=0
C
C CONVERT THE DENOMINATOR TO MIXED RADIX FORM, AND BUMP COUNT
C
CALL CONNIXDENRES,MRCDEN,COUNT,TDODEN)
OPCONS=OPCONS+COUNT
CASCNT=CASCNT+COUNT
C
C CONVERT THE NUMERATOR TO MIXED RADIX FORM, AND BUMP COUNT
C
60 CALL CONNIXNUMRES,MRCHUM,COUNT,TDNUM)
OPCONS=OPCONS+COUNT
CASCNT=CASCNT+COUNT
C
C TEST IF NUMERATOR IS LESS THAN DENOMINATOR, IF IT IS, GO CHECK THE QUOTIENT
C OTHERWISE, PROCEED (STATEMENT 50)
C
J=NUMMOD
DO WHILE(J.GE.1)
   IF(MRCNUM(J).LT.MRCDEN(J))THEN
      GO TO 72
   ELSE
      IF(MRCNUM(J).GT.MRCDEN(J))THEN
         GO TO 50
      ELSE
         END IF
      END IF
   END IF
   J=J-1
END DO
C
C IF THIS IS NOT THE FIRST ITERATION, THEN CONTINUE, OTHERWISE, FIND THE
C MOST SIGNIFICANT NONZERO MIXED RADIX DIGIT POSITION (L) OF THE DENOMINATOR
C
50 IF(ESTCAL.GE.0.0D0)THEN
   J=NUMMOD
   DO WHILE(J.GE.1)
      IF(MRCDEN(J).NE.0.)THEN
         L=L-1
         GO TO 43
      ELSE
         END IF
   END IF
   J=J-1
END DO
C
C
C FOR A GIVEN 1, FIND THE DIVISOR RECIPROCAL AND RESET THE FIRST ITERATION FLAG

IF(L.EQ.NUMMOD)THEN
    RECIPA=DINT(1/DEN)
ELSE
    RECIPA=DINT(1/NMOD1/1+DEN)
END IF

ELSE
END IF

C FIND THE MOST SIGNIFICANT NONZERO MIXED RADIX DIGIT POSITION (3) OF THE
C NUMERATOR

J=NUMMOD
DO WHILE(J.GE.1)
    IF(NUMJ(J).NE.0)THEN
        K=J
    ELSE
        GO TO 44
    END IF
END DD

C BEGIN TO FIND THE ESTIMATE. IF THE RECIPROCAL IS 1, THEN CHEAPER ESTIMATES
C ARE FOUND. IN THIS CASE, THE ESTIMATE IS THE MOST SIGNIFICANT NONZERO
C MIXED RADIX DIGIT OF THE NUMERATOR TIMES A PRODUCT OF MODULI

IF(M.GT.(L-1))THEN
    IF(RECIPA.GT.1.0)THEN
        JL=1
        DO WHILE(J.LE.(K-1))
            JL=1
            END DO
        ELSE
            OPCONT=OPCONT+1.0
            CASCNT=CASCNT+1.0
        END IF
    ELSE
        C OTHERWISE, SINCE THE RECIPROCAL IS NOT EQUAL TO 1, WE MUST MULTIPLY THE
        C MOST SIGNIFICANT NONZERO MIXED RADIX DIGIT OF THE NUMERATOR BY THE
        C PRODUCT OF THE RECIPROCAL AND THE SAME MODULI AS ABOVE
        JL=1
        DO WHILE(J.LE.(K-1))
            JL=1
            END DO
        OPCONT=OPCONT+2.0
        CASCNT=CASCNT+2.0
    END IF
C GO FIND THE NEW NUMERATOR AND UPDATE THE QUOTIENT WITH THE ESTIMATE
C
GO TO 1A

ELSE
END IF
C FIND THE ESTIMATE IF k=1. IF THE RECIPROCAL IS 1, THEN CHEAPER ESTIMATE
C FORM IS USED. IT IS THE MOST SIGNIFICANT NONZERO MIXED RADIX DIGIT OF THE
C NUMERATOR
C IF (RECIP.EQ.1.1) THEN
    IF (RECIP.EQ.0.0) THEN
        ESTIMA=DOUBLE(MARCHNUM(K))
    ELSE
        C OTHERWISE, USE THE MORE EXPENSIVE ESTIMATE FOR THIS CASE, WHICH IS
        C THE PRODUCT OF THE RECIPROCAL AND THE MS NONZERO RAD OF THE DENOMINATOR
        C
        ESTIMA=DOUBLE(MARCHNUM(K))*DOUBLE(RECIP)
        OPCONT=OPCONT+1.0
        CASCNT=CASCNT+1.0
        END IF
    END IF
C GO ITERATE THE NUMERATOR, AND ADD THE ESTIMATE TO THE QUOTIENT SUM
C ELSE
    GO TO 14
C END IF
C THIS IS THE CASE k=1. IF THE RECIPROCAL IS 1, THEN THE QUOTIENT IS INCREDENTED
C AND THE COUNT IS INCREDENTED. THEN THE QUOTIENT IS CHECKED (STATEMENT 72)
C IF (RECIP.EQ.1.0) THEN
    J=1
    DO WHILE (J.LE.NUMMOD)
        QUTCO(J)=MOD((QUTCO(J)+1.0),SYNUMMOD)
        J=J+1
    END DO
    OPCONT=OPCONT+1.0
    CASCNT=CASCNT+1.0
    GO TO 72
C OTHERWISE, IF THE RECIPROCAL IS EQUAL TO THE SCALING MODULUS, THEN THE
C ESTIMATE IS THE MS NONZERO RAD OF THE NUMERATOR. GO ITERATE WITH IT
C ELSE
    IF (RECIP.EQ.SYNUMMOD) THEN
        ESTIMA=DOUBLE(MARCHNUM(K))
        GO TO 14
    ELSE
        C OTHERWISE, THE ESTIMATE IS THE PRODUCT OF THE MS NONZERO RAD OF THE
        C NUMERATOR AND THE RECIPROCAL, SCALED BY THE kTH MODULUS. BUMP COUNT
        C
        ESTIMA=DOUBLE(MARCHNUM(K))*DOUBLE(RECIP)/DOUBLE(SYNUMMOD)
        OPCONT=OPCONT+3.0
        CASCNT=CASCNT+3.0
    C IF THIS ESTIMATE IS ZERO, THEN WE CAN GO CHECK THE QUOTIENT AS SOON AS
    C WE INCREMENT IT
    C IF (ESTIMA.EQ.0.00000000000000000) THEN
        J=1
        DO WHILE (J.LE.NUMMOD)
    C INCREMENT THE QUOTIENT, BUMP THE COUNT, AND GO CHECK THE QUOTIENT
    C

BEGIN THE ITERATION PHASE OF THE ALGORITHM. CONVERT THE ESTIMATE TO RESIDUE FORM.

DO WHILE(J.LE.NUMMOD)
   ESTRES(J)=MOD(ESTIMA, DBLE(SYSMODC(J)))
END DO

ADD THE ESTIMATE TO THE QUOTIENT SUM, AND SUM THE COUNT.

DO WHILE(J.LE.NUMMOD)
   QUOT(J)=AMOD(QUOT(J)+ESTRES(J), SYSMODC(J))
END DO

COMPUTE THE NEW NUMERATOR, AND ITERATE.

DO WHILE(J.LE.NUMMOD)
   NUMRES(J)=AMOD(NUMRES(J)+<SYSMODC(J)-1.0>*ESTRES(J)*DENRES(J), SYSMODC(J))
END DO

START TO CHECK THE QUOTIENT. CONVERT IT TO NORMAL POSITIONAL NOTATION.

IF THE QUOTIENT IS NOT CORRECT, WRITE ERROR AND STOP.

CALL CONINT(CQUOT, N, Q)
   IF(Q.EQ.DINTNUM/DEN) THEN
      WRITE(C, 4) "ERROR Q", "Q", "I" "I"
      WRITE(C, 4) "N", "NUM", "D", "DEN"
      STOP
   ELSE
      END IF

TALLY THE NUMBER OF OPERATIONS, SIMULATE THE BIA ON THE SAME PROBLEM, AND CHECK THE BANERJI QUOTIENT. THEN COMPUTE VARIOUS STATISTICS, AND OUTPUT THEM.

SCATER(CASCNT)=SCATER(CASCNT)+1
CALL BANERJ NUMBAN, DENBAN, BANG, BANCNT
   IF(BANG.NE.0) THEN

304
WRITE(*,0)"ERROR BA2ERJ","BAN=",nBANG"."I=",I
WRITE(*,0)"n=",nHUM,"D=",D
STOP
ELSE END IF
SCABAN(BANCNT)=SCABAN(BANCNT)+1.
BANTOT+BANTOT+BANCNT
BANVAR+BANVAR+(BANCNT-SRMSN)+2
DIF=BANCNT-CASCNT
DIFTIM=DIFTIM+DIF
DIFVAR=DIFVAR+(DIF-SANDIF)+2
VARTUR=(VARTUR+(CASCNT-SRMSN)+2
IFCMODI.HERER)+.EQ..DOTHEN
OUTONE=DIFTIM/I
OUTTRE=OPCOUT/T
OUTFOR=BANTOT/I
OUTPI=VARTUR/(I-1)
OUTSI=BANVAR/(I-1)
WRITE(*,0)OUTTRE,OUTFOR,OUTONE,OUTPI,OUTSI
ELSE END IF

C INCREMENT THE SAMPLE COUNTER
C I=I+1
END DO
WRITE(*,0)SCATER
WRITE(*,0)SCABAN

C INCREMENT THE MODULUS SET COUNTER
C JJ=J+1
END DO
STOP
END C

C THIS SUBROUTINE CONVERTS THE MIXED RADIX DIGITS OF A NEGATIVE OPERAND, STORED
C IN RESIDU, TO THE MIXED RADIX DIGITS OF THE ABSOLUTE VALUE OF THE OPERAND
C RETURNED IN RESIDU
C SUBROUTINE FLIP(RESIDU)
COMMON SYMS0D(20),NUMMOD,SYSNNY(96,20),NROOE
REAL9 NROOE(10)
REAL RESIDU(20)
C FIND THE LEAST SIGNIFICANT MIXED RADIX DIGIT
C LEESIG=0
DO WHILE(J.LE.NUMMOD)
IF (RESIDU(J).NE.0) THEN
LEESIG=J
GO TO 10
ELSE END IF
J=J+1
END DO

C COMPUTE THE ABSOLUTE VALUE
C
10 IF(LEESIG.NE.0) THEN
   J=1
   DO WHILE(J.LT.NUMMOD)
      RESIDUC(J)=MOD(RESIDU(J)-1.,SYSMOD(J))
      J=J+1
   END DO
   DO WHILE(J.LT.LEESIG)
      RESIDUC(J)=0.
      J=J+1
   END DO
   RESIDU(LEESIG)=MOD(RESIDU(LEESIG)-1.,SYSMOD(LEESIG))
   ELSE
   END IF
END IF
RETURN
C C THIS SUBROUTINE CONVERTS THE RESIDUE OPERAND RESIDU TO ITS DECIMAL EQUIVA
C LENT, RETURNED IN X
C SUBROUTINE CONINTCRESIOU.N.X
C THIS SUBROUTINE CONVERTS THE RESIDUE OPERAND RESIDU TO ITS DECIMAL EQUIVA
C LENT, RETURNED IN X
C SUBROUTINE CONINTCRESIOU.N.X
C MAKE A COPY OF THE RESIDU INPUT, USED TO RESTORE THE INPUT WHEN RETURNING
C DO 5 I=1,NUMMOD
   CPVFRES(1)=RESIDU(1)
   CONTINUE
   CONTINUE
   DO A MIXED RADIX CONVERSION, AND SUM THE MIXED RADIX TERMS
   CALL CONMIXCRESIDU,SCRMOD,COUNT,TWOMOD)
   X=0.0
   DO 10 I=1,NUMMOD
      X=S+SCRMOD(I)*SCVFRES(I)
   10 CONTINUE
C C RESTORE THE INPUT AND RETURN
C   DO 15 I=1,NUMMOD
   RESIDU(I)=CPVFRES(I)
   CONTINUE
   RETURN
END
C C THIS ROUTINE IS NOT CALLED BY THE RA SIMULATOR. IT IS INCLUDED HERE
C FOR LATER USE IF NECESSARY
C SUBROUTINE QUOESTCMRCNUH.MRCDEN.OUORES.COUNT.QUOTIIO.UPFLRGl
C COMMON SYSMOD20),NUMMOD,SYSINV20),SCVFRES
C REAL# MRCEOEC20),A,M
C DIMENSION RESIDU20),SCRMOD20),CPVFRES20)
C DO 5 I=1,NUMMOD
   CPVFRES(1)=RESIDU(1)
   CONTINUE
   CONTINUE
   DO A MIXED RADIX CONVERSION, AND SUM THE MIXED RADIX TERMS
   CALL CONMIXCRESIDU,SCRMOD,COUNT,TWOMOD)
   X=0.0
   DO 10 I=1,NUMMOD
      X=S+SCRMOD(I)*SCVFRES(I)
   10 CONTINUE
C C RESTORE THE INPUT AND RETURN
C   DO 15 I=1,NUMMOD
   RESIDU(I)=CPVFRES(I)
   CONTINUE
   RETURN
END
C C THIS ROUTINE IS NOT CALLED BY THE RA SIMULATOR. IT IS INCLUDED HERE
C FOR LATER USE IF NECESSARY

396
IF(RECODE(I).NE.0) THEN
  MSDEN=I
  GO TO 10
ELSE
  END IF
  I=I-1
END DO
10 I=NUMMOD
DO WHILE(1.LE.I)
  IF(REMCHUN(I).NE.0) THEN
    MSDNUM=I
    GO TO 20
  ELSE
    END IF
    I=I-1
  END DO
20 IF(MSDDEN.GT.1) THEN
  IF(RECODE(MSDDEN-1).LE.AINT(SYSMOD(MSDDEN-1)/2)) THEN
    NAMDQ=RECODE(MSDDEN)
    NAMDC=RECMUM(MSDNUM)
    ELSE
      NAMDQ=RECODE(MSDDEN)/2.
      NAMDC=RECMUM(MSDNUM)/2.
      UPFLAG=1.
    END IF
  ELSE
    NAMDQ=RECODE(I)
    NAMDC=RECMUM(MSDNUM)
  END IF
  IF(MSDNUM.EQ.MSDDEN) THEN
    QUOT=DFLOTJ(JIFIX(NAMDQ/DRNDC))
    ELSE
      IF(MSDNUM.EQ.MSDDEN+1) THEN
        QUOT=NAMDQ*DFLOTJ(JIFIX(SYSMOD(MSDDEN)/DRNDC))
        COUNT=COUNT+1.
        ELSE
          SUB=INT(RECCMQE(MSDNUM)/(SYSMOD(MSDDEN)+RECCMQE(MSDDEN)))
          QUOT=NAMDQ*DFLOTJ(JIFIX(SYSMOD(MSDDEN)/DRNDC))/SUB
          COUNT=COUNT+2.
        END IF
      END IF
  END IF
  I=1
  DO WHILE(I.LE.NUMMOD)
    QUOT=QFLOTJ(JIFIX(MSDDEN(I)/2))
    I=I+1
    END DO
  QUOT=QFLOTJ(JIFIX(MSDDEN))/2.
  RETURN
END

C THIS SUBROUTINE FINDS THE MIXED RADIX DIGITS OF THE RESIDUE OPERAND STORED
C IN RESIOU. THE MIXED RADIX DIGITS ARE RETURNED IN MRCOUT, THE NUMBER OF
C OPERATIONS REQUIRED IS RETURNED IN COUNT, AND THE XOR VALUE OF THE OPERAND
C IS #RETURNED IN TWMOD

C SUBROUTINE CONMIX (RESIDU, MRCOUT, COUNT, TWMOD)
C COMMON SYSMOD(20). NUMMOD. SYSH(96.20). NAMCODE
C REALS NAMDCQ(20)
C DIMENSION RESIDU(20), KPPRES(20), XSTRM(20)
REAL MRCOUT(20)
COPY THE OPERAND, AND INITIALIZE THE MIXED RADIX COEFFICIENT REGISTER

\[ \text{C} \]
\[ \text{I}=1 \]
\[ \text{DO WHILE}(\text{I} \leq \text{NUMMOD}) \]
\[ \text{CPYRES(I)}=\text{RESDU(I)} \]
\[ \text{NRCOUT(I)}=0 \]
\[ \text{I}+1 \]
\[ \text{END DO} \]
\[ \text{I}=1 \]
\[ \text{DO WHILE}(\text{I} \leq \text{NUMMOD}) \]
\[ \text{NRCOUT(I)}=\text{RESDU(I)} \]
\[ \text{J}=1 \]
\[ \text{DO WHILE}(\text{J} \leq \text{NUMMOD}) \]
\[ \text{RESTER(J)}=\text{MOD}(\text{NRCOUT(I)} \times \text{SYSMOD(J)}) \]
\[ \text{J}+1 \]
\[ \text{END DO} \]
\[ \text{J}=1 \]
\[ \text{DO WHILE}(\text{J} \leq \text{NUMMOD}) \]
\[ \text{RESTER(J)}=\text{MOD}(\text{SYSMOD(I)} \times \text{SYSMOD(J)}) \]
\[ \text{J}+1 \]
\[ \text{END DO} \]

SUBTRACT THE NEXT MOST SIGNIFICANT RESIDUE DIGIT FROM THE RESIDUE DIGITS

\[ \text{C} \]
\[ \text{J}=1 \]
\[ \text{DO WHILE}(\text{J} \leq \text{NUMMOD}) \]
\[ \text{RESDU(J)}=\text{MOD}(\text{RESDU(J)}-\text{RESTER(J)}(\text{SYSMOD(J)}-1) \times \text{SYSMOD(J)}) \]
\[ \text{J}+1 \]
\[ \text{END DO} \]
\[ \text{J}=1 \]
\[ \text{DO WHILE}(\text{J} \leq \text{NUMMOD}) \]
\[ \text{RESTER(J)}=\text{MOD}(\text{SYSMOD(I)} \times \text{SYSMOD(J)}) \]
\[ \text{J}+1 \]
\[ \text{END DO} \]

MULTIPLY BY THE INVERSE MODULUS IN EACH POSITION IN WHICH THE INVERSE EXISTS

\[ \text{C} \]
\[ \text{J}=1 \]
\[ \text{DO WHILE}(\text{J} \leq \text{NUMMOD}) \]
\[ \text{IF}(\text{J} \neq 1) \text{THEN} \]
\[ \text{RESDU(J)}=\text{MOD}(\text{RESDU(J)} \times \text{SYNIV}(\text{RESTER(J)} \times \text{J} \times \text{SYSMOD(J)}) \text{ELSE} \]
\[ \text{END IF} \]
\[ \text{J}+1 \]
\[ \text{END DO} \]
\[ \text{J}=1 \]
\[ \text{END DO} \]

RESTORE THE RESIDUES, SET THE COUNT AND FIND THE XOR VALUE

\[ \text{C} \]
\[ \text{I}=1 \]
\[ \text{DO WHILE}(\text{I} \leq \text{NUMMOD}) \]
\[ \text{RESDU(I)}=\text{CPYRES(I)} \]
\[ \text{I}+1 \]
\[ \text{END DO} \]
\[ \text{COUNT}=2 \times \text{NUMMOD}+1 \]
\[ \text{TWOMOD}=0 \]
\[ \text{I}=1 \]
\[ \text{DO WHILE}(\text{I} \leq \text{NUMMOD}) \]
\[ \text{TWOMOD}=\text{MOD}(\text{NRCOUT(I)} \times \text{TWOMOD}+2) \]
\[ \text{I}+1 \]
\[ \text{END DO} \]

RETURN

END
C THIS SUBROUTINE IS NOT USED BY THE RA SIMULATOR, BUT IS INCLUDED FOR FUTURE REFERENCE
C
C SUBROUTINE BANERJ(NMRES,DMRES,QUORES,COUNT,QUOT,UPFLAG)
COMMON SYSMOD(20),NUMMOD,SYSTNV(96,20),NRCCOE
REALNRCCOE(20)
DIMENSION QUORES(20)
REALMACHUM(20),NRCCOE(20),NRNDCO
REAL NRCCOE
COUNT=0.
UPFLAG=0.
I=NUMMOD
DO WHILE(C.GE.1)
   IF(NRCCOE(I).NE.0.)THEN
      MDDEN=I
      GO TO 10
   ELSE
      END IF
      I=I+1
   END DO
   I=NUMMOD
   DO WHILE(C.GE.1)
      IF(NRCCOE(I).NE.0.)THEN
         MSNUM=I
         GO TO 20
      ELSE
         END IF
         I=I+1
      END DO
      DRNDCO=NRCCOE(MSDDEN)
      NRNDCO=NRCCOE(MSNUM)
      IF(MSNUM.EQ.MSDDEN)THEN
         QUOT=SYSTNV(JPXX)(MSDDEN)/DRNDCO
         COUNT=COUNT+1.
      ELSE
         IF(MSDDEN.EQ.MSDDEN)THEN
            QUOT=SYSTNV(JPXX)(SYSMOD(MSDDEN))/DRNDCO
            COUNT=COUNT+1.
         ELSE
            SUB=NRCCOE(MSDDEN)/SYSTNV(JPXX)(SYSMOD(MSDDEN))
            QUOT=SYSTNV(JPXX)(SYSMOD(MSDDEN))/DRNDCO
            COUNT=COUNT+2.
            END IF
      END IF
      I=1
      DO WHILE(C.LE.NUMMOD)
         QUORES(I)=ANDO(SNGL(QUOT),SYSTNV(I))
         I=I+1
      END DO
      QUOT=(ANDO(SNGL(QUOT),2.))
      RETURN
   END

C This subroutine simulates the Banerji-I algorithm. The numerator
C and denominator residues are passed in NUMRES and DENRES, respectively. The
C decimal value of the truncated quotient is returned in BANQ, and the number
C of residue operations required to find the truncated quotient is returned
C in BANCNT
C
C SUBROUTINE BANERJ(NUMRES, DENRES, BANQ, BANCNT)
COMMON SYSMOD(20),NUMMOD,SYSTIV(96,20),MRCOE
REAL=0. MRCOE(20),BANG,32,SUB
NCAL NUMRES20,DENRES(20),SCANDT(20),MRCNUM(20)
REAL MCOA(20)
BANG=0.D00
BANCNT=0.
ONEPAS=1.

IS THE NUMERATOR ZERO? IF YES, THEN RETURN. ELSE, ARC THE DENOMINATOR IF
THIS IS THE FIRST PASS, OR ARC THE NUMERATOR.

TEST=0.
I=1
DO WHILE(1.IE.NUMMOD) TEST=TEST*NUMRES(I)
I=I+1
END DO
IF (TEST.EQ.0.) THEN RETURN
ELSE IF (ONEPAS.EQ.1.) THEN
GO TO 30
ELSE

CALL CONXVEC(DENRES,MRCDEN,COUNT,TWODEN)
BANCNT=BANCNT+COUNT

Find the most significant nonzero mixed radix digit
I=NUNMOD
DO WHILE(1.IE.13)
IF (MCOA(I).NE.0.) THEN
NSO DEN=I
GO TO 25
ELSE
END IF
I=I+1
END DO

Do the denominator rounding. If digits of lesser significance are all zero, and the
most significant digit is 1, then no rounding is done

TEST=0.
I=1
DO WHILE(1.LT.NSODEN) TEST=TEST*MRCDEN(I)
I=I+1
END DO
IF ((TEST.EQ.0.).AND.(MRCDEN(NSODEN).EQ.1.) THEN
DNMOD=MRCDEN(NSODEN)
ELSE
DNMOD=MRCDEN(NSODEN)+1.
END IF
ONEPAS=0.
END IF

Convert numerator to mixed radix form, and bump the count
CALL CONNX NUMRES, NUMRC, COUNT, UNUM

BANCHT = BANCHT + COUNT

Find the most significant nonzero digit of the numerator

I = NUMMOD
DO WHILE (I .GE. 1)
  IF (NUMRC(I) .NE. 0) THEN
    MSDNUM = I
    GO TO 40
  ELSE
    END IF
    I = I - 1
  END DO

Find the quotient estimate stored in ZETE. Count not increased if k = 1.
If k = 1, count is increased by 1. Otherwise, count is increased by two

IF (MSDNUM .LT. MSDDEN) THEN
  RETURN
ELSE
  IF (MSDNUM .EQ. MSDDEN) THEN
    ZETE = DBLE (INT (NUMRC * (MSDNUM / MSDDEN)))
    GO TO 50
  ELSE
    ZETE = DBLE (INT (NUMRC / MSDDEN))
    BANCHT = BANCHT + 1
    GO TO 50
  END IF
END IF

Update the running quotient sum stored as BANG. If the quotient estimate is nonzero, then add it to the running sum, compute a new numerator, and increment count. If it is zero, then find the quotient correction in preparation for return.

IF (ZETE .NE. 0.0) THEN
  BANG = BANG + ZETE
  BANCHT = BANCHT + 1
ENDIF

Compute the next partial numerator stored in NUMRES, and bump the count by two operations

I = 1
DO WHILE (I .LE. NUMMOD)
  SCRNOD(I) = NUMRES(I) .MOD ZETE
END DO
I = I + 1
DO WHILE (I .LE. NUMMOD)
  SCRNOD(I) = (SCRNOD(I) .MOD ZETE) .MOD ZETE
END DO
I = I + 1

311
I=1
DO WHILE(I.LE.NUMMOD)
  NUMRES(I)=MOD(NUMRES(I)+(SYSMOD(I)-1).*SQRMOD(I),SYSMOD(I))
  I=I+1
END DO
BANCHT=BANCHT+2.
GOTO 35
ELSE
C The quotient estimate was zero, so we must find the correction to the sum.
C I=NUMMOD
  DO WHILE(I.GE.1)
    IF(CHRNUM(I).GT.NACDEN(I)) THEN
      BAND=BAND+1.0
      BANCHT=BANCHT+1.
      RETURN
    ELSE
      IF(CHRNUM(I).LT.NACDEN(I)) THEN
        RETURN
      ELSE
        END IF
      END IF
      I=I-1
    END DO
    BAND=BAND+1.0
    BANCHT=BANCHT+1.
  RETURN
END IF
END