1

National Library
of Canada

Bibliothèque nationale
du Canada

Canadian Theses Service

Services des thèses canadiennes

Ottawa, Canada
K1A 0N4

CANADIAN THESES

THÈSES CANADIENNES

NOTICE

The quality of this microfiche is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Previously copyrighted materials (journal articles, published tests, etc.) are not filmed.

Reproduction in full or in part of this film is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30. Please read the authorization forms which accompany this thesis.

AVIS

La qualité de cette microfiche dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

Les documents qui font déjà l'objet d'un droit d'auteur (articles de revue, examens publiés, etc.) ne sont pas microfilmés.

La reproduction, même partielle, de ce microfilm est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30. Veuillez prendre connaissance des formules d'autorisation qui accompagnent cette thèse.

THIS DISSERTATION
HAS BEEN MICROFILMED
EXACTLY AS RECEIVED

LA THÈSE A ÉTÉ
MICROFILMÉE TELLE QUE
NOUS L'AVONS REÇUE

National Library
of Canada

Bibliothèque nationale
du Canada

0-315-22162-3

Canadian Theses Division Division des thèses canadiennes

Ottawa, Canada
K1A 0N4

PERMISSION TO MICROFILM — AUTORISATION DE MICROFILMER

• Please print or type — Écrire en lettres moulées ou dactylographier

Full Name of Author — Nom complet de l'auteur

RALLIS ANTONIADIS

Date of Birth — Date de naissance

OCTOBER 10, 1957

Country of Birth — Lieu de naissance

GREECE

Permanent Address — Résidence fixe

~~1800~~ DEPARTMENT OF MATHEMATICS AND STATISTICS
CARLETON UNIVERSITY
COLONEL BY DRIVE
OTTAWA, ONTARIO K1S 5B6

Title of Thesis — Titre de la thèse

COMPUTER SYSTEMS FOR HIGH PRECISION ARITHMETIC

University — Université

CARLETON UNIVERSITY

Degree for which thesis was presented — Grade pour lequel cette thèse fut présentée

MASTER OF SCIENCE

Year this degree conferred — Année d'obtention de ce grade

1984

Name of Supervisor — Nom du directeur de thèse

MICHAEL ATKINSON

Permission is hereby granted to the NATIONAL LIBRARY OF CANADA to microfilm this thesis and to lend or sell copies of the film.

The author reserves other publication rights, and neither the thesis nor extensive extracts from it may be printed or otherwise reproduced without the author's written permission.

L'autorisation est, par la présente, accordée à la BIBLIOTHÈQUE NATIONALE DU CANADA de microfilmer cette thèse et de prêter ou de vendre des exemplaires du film.

L'auteur se réserve les autres droits de publication; ni la thèse ni de longs extraits de celle-ci ne doivent être imprimés ou autrement reproduits sans l'autorisation écrite de l'auteur.

Date

Signature

Antoniadis Rallis

COMPUTER SYSTEMS FOR HIGH PRECISION ARITHMETIC

by

RALLIS ANTONIADIS

(B.Sc. Athens)

A thesis submitted to
the Faculty of Graduate Studies and Research
in partial fulfilment of
the requirements for the degree of
Master of Science

Department of Mathematics and Statistics

Carleton University

Ottawa, Ontario

Date : May 1984.

© copyright

1984, Rallis Antoniadis

LICENCE TO CARLETON UNIVERSITY

In the interests of facilitating research by others at this institution and elsewhere, I hereby grant a licence to:

CARLETON UNIVERSITY

to make copies of my thesis:

COMPUTER SYSTEMS FOR HIGH PRECISION ARITHMETIC

or substantial parts thereof, the copyright which is invested in me, provided that the licence is subject to the following conditions:

Only singled copies shall be made or authorized to be made at any one time, and only in response to a written request from the library of any university or similar institution on its own behalf or on behalf of one of its users.

This licence shall continue for the full term of the copyright, or for so long as may be legally permitted.

The Universal Copyright Notice shall appear on the title page of all copies made under the authority of this licence.

This licence does not permit the sale of authorized copies at a profit, but does permit the collection by the institution or institutions concerned of charges covering actual costs.

All copies under the authority of this licence, shall bear a statement to the effect that the copy in question "is being made available in this form by the authority of the copyright owner solely for the purpose of private study and research and may not be copied or reproduced except as permitted by the copyright laws without written authority from the copyright owner."

The foregoing shall in no way preclude the granting by the author of a licence to the National Library of Canada to reproduce the thesis and to lend or to sell copies of the same. For this purpose it shall also be permissible for Carleton University to submit the abovementioned thesis to the National Library of Canada.

[Signature]
Signature of Witness

Date

M. Sc.
Degree

[Signature]
Signature of Student

Mathematics and Statistics
Department of Student

The undersigned hereby recommend to
The Faculty of Graduate Studies and Research
acceptance of the thesis, submitted by
RALLIS ANTONIADIS, B.Sc.
in partial fulfilment of the requirements
for the degree of Master of Science

B. 557. Puthanwalia

Chairman, Department of Mathematics and Statistics

[Signature]
Thesis Supervisor

[Signature]
External Examiner

Carleton University

Date : May 1984.

ABSTRACT

Several methods for representing rational numbers are surveyed and compared.

A unified approach to radix number systems is first presented. Next, the residue and p-adic number systems are described along with associated algorithms for arithmetic operations. Some applications to matrix computation are given. Finally, a new representation due to E.C.R. Hehner and R.N.S. Horspool which is suitable for multiprecision exact arithmetic is described, together with a computer implementation based on linked lists.

ACKNOWLEDGEMENTS

I would like to thank my supervisor Dr. M.D. Atkinson for his helpful guidance throughout the present thesis.

I would also like to thank Dr. Brian Mortimer for the occasional discussions we had.

Finally, I would like to express my gratitude to Dr. C.W.L. Garner for his sincere interest, and to the excellently organized Department of Mathematics and Statistics which made my stay in Canada one of the most pleasant periods of my life.

Table of Contents

Acceptance sheet	ii
Abstract	iii
Acknowledgements	iv
Table of contents	v
1 Introduction	1
1.1 Discussion of some properties of number systems	4
2 Radix Systems	6
2.1 Introduction	6
2.2 Definitions and basic properties	6
2.3 Arithmetic operations	10
2.3.1 Addition and Subtraction	10
2.3.2 Multiplication and Division	12
2.4 Imaginary-radix Number Systems	15
2.5 The Classical Algorithms	20
2.6 Digital methods	27
3 Modular Arithmetic	37
3.1 The modular approach	37
3.1.1 Conversion techniques	39
3.1.2 Selection of moduli	45
3.2 Applications	46
4 P-adic Systems	48
4.1 Introduction	48
4.2 Background	50
4.2.1 P-adic numbers	50
4.2.2 Farey fractions	54
4.2.3 Continued fractions	55
4.3 P-adic representation	61
4.4 Hensel Codes	64
4.4.1 Arithmetic operations using Hensel Codes	71
4.5 A New Representation	77
4.5.1 Arithmetic operations	83
4.5.2 Length of representation	90
4.5.3 Length bounds	91
4.5.4 Approximation	92
4.6 Conversion Algorithms	93
4.6.1 Conversion Algorithms of $H(p,r)$ codes	94
4.6.2 Conversion Algorithms for the new representation	97
4.7 Exact solution of Linear Equations	97
4.8 Concluding Remarks	103

- ii) Multiplication or division by a power of the base (in fixed-radix systems) can be accomplished merely by shifting the digits in a storage register; i.e. moving the radix point
- iii) Extending the range of the number system is easily realized by adding more digit positions
- iv) Overflow detection is easily mechanized.

Chapter 1

Introduction

The very nature of digital computers makes them dependent upon number systems and upon the rules which define the relationships among numbers. However, this dependence is sometimes obscured or forgotten, since only the decimal and the binary number systems are commonly used in computers, and having become so familiar, they are often taken for granted; however, numerous algorithms for arithmetic operations in these systems have been developed and implemented in computer hardware.

A unified approach to the above finite number systems as well as to other fixed-radix number systems, like the negative radix systems or even the imaginary radix number systems, is presented in chapter 2 and enables us to develop a single set of algorithms for arithmetic operations

The single-precision arithmetic (i.e. dealing with values which can be stored in a single machine word) may not yield sufficient accuracy for a given application. In such a case, we may increase the precision by suitable programming techniques which use two or more words of memory to represent each number.

The Classical Algorithms for multiple-precision arithmetic (i.e. dealing with numbers which have arbitrarily high precision) and a fast-multiplication algorithm for binary numbers due to Toom and Cook complete chapter 2.

Since an automatic digital computer is a finite machine, it is capable of representing, internally, only a finite set of numbers. Therefore, any attempt to use an automatic digital computer to do arithmetic in the field of real numbers $(R, +, \cdot)$ is doomed to failure, because R is an infinite

set and all but a finite subset of its elements cannot be represented in a finite machine. Of course, there is the alternative of approximating the non-representable numbers and, if rounding errors can be tolerated, standard digital computer arithmetic can be used to find approximate solutions to numerical problems. These methods can be applied to the so called "well-conditioned" problems, where one may obtain reliable results in spite of small round-off errors which occurred throughout the computational process.

On the other hand, there is another class of problems which are called "ill-conditioned" and, unfortunately, many practical problems fall in this category. In this class, even small errors may result in a final solution which bears no resemblance to the actual solution of the given problem. Some matrix computations suffer from problems of this kind. For example, suppose that it is desirable to obtain the exact rational solution to a nonsingular system $Ax = b$ of linear equations with integer coefficients. There are presently two approaches to find such exact solutions: direct computation using multiple-precision arithmetic or residue arithmetic.

The concept of modular arithmetic and residue techniques are presented in chapter 3. Residue arithmetic, which is the manipulation of numbers expressed in the residue number system, offers an unusual set of characteristics. For example, it provides the ability to add, subtract, or multiply in one step, regardless of the length of the numbers involved, without recourse to intermediate carry digits or internal delays. Hence, by a change in the number system, rather than in the computer hardware, it is possible in principle to make a computer operate faster.

In the residue number system, any rational $\frac{a}{b}$ can be mapped onto an integer in $I_p = \{0, 1, 2, \dots, p-1\}$ by defining $(a/b) \bmod p = (a \cdot b^{-1}) \bmod p$ (where b^{-1} is the multiplicative inverse of $b \pmod{p}$), which integer is then represented in a residue number system using one or more prime moduli. Unfortunately, since division is not an easy operation while using several prime moduli, the computational problems get involved. The advantages as well as the disadvantages of residue arithmetic and the conversion algorithms complete chapter 3.

Instead of using several primes, one can use p -adic arithmetic, which is similar to using a base p^r , where r is the number of digits in base p . This system provides more flexibility since it permits a suitable choice of p and r , and also has the advantage that there is no need to combine the results since the results modulo p^r are directly obtained; this is the subject presented in chapter 4.

After a short introduction to p -adic numbers, we first introduce the segmented p -adic arithmetic system and we discuss its properties and its advantages for exact computation. However, problems like the detection of overflow when the rationals exceed the range under consideration -inherent in all fixed-length number systems -, or detection of sign and magnitude of these codes (which we shall call $H(p,r)$ Hensel codes) directly without explicit conversion, cannot be avoided.

These difficulties motivated us to look for another representation which we present in the section following the segmented p -adic codes. This new representation of the rational numbers due to E.C.R. Hehner and R.N.S. Horspool ([9]) uses a compact variable-length encoding that may be viewed as a generalization of radix complement notation. It allows exact arithmetic, and approximate arithmetic under programmer's control. It is superior to existing coding methods, because the arithmetic operations take particularly simple, consistent forms. These attributes make this new representation attractive for use in computer hardware. The conversion algorithms, the applications of p -adic arithmetic in matrix computations, and a short comparison of the two p -adic representations complete chapter 4.

In chapter 5, we try to implement this last attractive new representation of the rational numbers by giving it an equally attractive structure which can be handled by computer hardware, namely, the linked list structure. The arithmetic operations between numbers which have been represented as linked lists as well as the process which we follow to carry them out, are stated in chapter 5.

A computer listing can be found in the Appendix.

1.1 Discussion of some properties of number systems.

Range: The range of a number system is defined as the interval over which every integer can be represented by the system, without having two numbers with the same representation. In practice, of course, truncated number systems are employed, because physical systems can accommodate only a limited number of digits. In computers, the range is generally specified by stating the word length and the type of number system used.

Uniqueness: A number system is said to be unique if each number in the system has one only representation. For example, the 2's complement representation, frequently used in computers, is unique. However, many of the commonly used number systems are not unique. For example, the decimal number system, when extended to include negative integers, usually has two representations for 0, namely, +00...000 and -00...000. Obviously, any system with sign and magnitude representation is not unique.

Redundancy: A number system is defined to be redundant if there are fewer numbers than are combinations of digits. Therefore, for some combinations of digits, there may not exist any defined number. Alternatively, different combinations may correspond to the same number.

Weighted Number Systems: A number system is said to be weighted if there exists a set of weights w_i such that for any number x in the system, x can be represented in the form

$$x = \sum_{i=1}^n a_i w_i,$$

where a_i are a set of permissible digits. If the values for w_i are successive powers of the same number, then the number system has a *fixed base* or *fixed radix* (e.g. base 10 for decimal, base 2 for the binary system, etc.). Number systems in which the weights are not powers of the same radix are called *mixed-radix systems*.

Some of the advantages of the weighted number systems are the following ([20]):

- i) Relative-magnitude comparison of two numbers can be mechanized easily.

- ii) Multiplication or division by a power of the base (in fixed-radix systems) can be accomplished merely by shifting the digits in a storage register; i.e. moving the radix point
- iii) Extending the range of the number system is easily realized by adding more digit positions.
- iv) Overflow detection is easily mechanized.

Chapter 2

Radix Systems

2.1 Introduction

The positive-radix and radix complement number systems are the most commonly used finite number representation systems

However, other fixed-radix number systems have received a great amount of attention in recent years.

In this chapter, we shall present a unified approach to these finite number systems, and we shall also show that not only the above mentioned systems are members of a broad class of finite number representation systems, but that we can extend this class to include imaginary-radix number systems, as well.

The obvious advantage of this approach is that it enables us to develop a single set of algorithms for arithmetic operations.

In the first three sections we shall follow the notation used, and give the results stated in [13].

2.2 Definitions and basic Properties

Consider a class of n -digit, fixed radix number systems in which every member is characterized by the triple $\langle n, \beta, A \rangle$ and has the standard digit set $\{0, 1, \dots, \beta - 1\}$, where β is the radix, which

is a positive integer, and A is a vector of length n , i.e. $A = (\lambda_{n-1}, \lambda_{n-2}, \dots, \lambda_0)$, where $\lambda_i \in \{-1, 1\}$

The algebraic value X of an n -tuple $(x_{n-1}, \dots, x_0)$ in the system $\langle n, \beta, A \rangle$ is defined by

$$X = \sum_{i=0}^{n-1} \lambda_i x_i \beta^i \quad (1)$$

We observe, that for a given radix β , there are 2^n distinct number systems in this class, which are also nonredundant and complete¹

Special cases of radix systems which belong in this class are the following

- i) The positive radix number system, which can be obtained for $\lambda_i = +1$ for every i
- ii) The negative radix number system, which can be obtained for $\lambda_i = (-1)^i$ for every i and
- iii) The radix complement number system, with x_{n-1} as a sign digit (i.e. x_{n-1} can either be 0 or 1) and the characterizing vector $A = (-1, 1, 1, \dots, 1)$

Suppose now that $A = \langle n, \beta, A \rangle$ is a member of the class mentioned above. Then, the largest representable integer in A is the positive number P , which can be obtained by allowing the digits corresponding to $\lambda_i = +1$ to take their maximum value, i.e. $\beta - 1$ and the ones corresponding to $\lambda_i = -1$ to take their minimum value, i.e. 0. Hence, if p_i denotes the i -th digit of P , we have

$$p_i = \begin{cases} \beta - 1 & , \text{ if } \lambda_i = +1, \\ 0 & , \text{ otherwise} \end{cases} \quad i = 0, 1, 2, \dots, n-1 \quad (2)$$

The value of the largest representable integer P is given by

$$P = \frac{1}{2} \sum_{i=0}^{n-1} (\lambda_i + 1)(\beta - 1)\beta^i \quad (2a)$$

¹ i.e. in a given number system A , we have unique representation for any number within the range of A . For more details, see [13] and references given there.

and to verify it, suppose at first that $\lambda_i = 1$, for every i . Then, (2a) gives

$$P = \frac{1}{2} \sum_{i=0}^{n-1} (\lambda_i + 1)(\beta - 1)\beta^i = \frac{1}{2} \sum_{i=0}^{n-1} 2(\beta - 1)\beta^i = \sum_{i=0}^{n-1} (\beta - 1)\beta^i$$

which obviously is the value of the largest representable integer in the positive-radix number system. Suppose now, that not all λ_i 's are equal to 1. For the terms of (1) which correspond to $\lambda_i = -1$, the greatest value they can take is 0, which is again obtained using the formula (2a). Hence,

$$P = \frac{1}{2} \sum_{i=0}^{n-1} (\lambda_i + 1)(\beta - 1)\beta^i = \frac{1}{2} \left[\sum_{i=0}^{n-1} \lambda_i (\beta - 1)\beta^i + \sum_{i=0}^{n-1} (\beta - 1)\beta^i \right] = \frac{1}{2} [R + (\beta^n - 1)] \quad (3)$$

where R is the algebraic value of the number $(\beta - 1)\beta^{n-1} + (\beta - 1)\beta^{n-2} + \dots + (\beta - 1)$ in A .

Quite similarly, the smallest integer is the negative number N whose representation is

$$n_i = \begin{cases} \beta - 1 & \text{if } \lambda_i = -1, \\ 0 & \text{otherwise} \end{cases} \quad ; i = 0, 1, \dots, n-1 \quad (4)$$

and the corresponding value is

$$N = -\frac{1}{2} \sum_{i=0}^{n-1} (\lambda_i - 1)(\beta - 1)\beta^i = -\frac{1}{2} [R + (\beta^n - 1)] \quad (4a)$$

The number of integers in the range $N \leq X \leq P$ is $P - N + 1 = \beta^n$, and the range of the system A is, in general, asymmetric about zero and a measure of the asymmetry can be the difference $P - |N| = P + N = R$.

Examples:

- i) The negative number system $\langle n, \beta, A = (-1, -1, +1, -1, +1) \rangle$ is asymmetric. For n even there are β times as many negative numbers as positive ones; the opposite holds if we take $A = (-1, +1, -1, +1, -1)$.
- ii) For the 2's complement system $\langle n, \beta = 2, A = (-1, 1, \dots, 1) \rangle$, we have $P = \beta^{n-1} - 1$, $R = -1$ and therefore it is nearly symmetric.

Case 1: $z_i \geq \lambda_i s_i$; then $b_{i+1} = 0$ and $c_{i+1} = (z_i - \lambda_i s_i)/\beta$

Case 2: $z_i < \lambda_i s_i$; then $c_{i+1} = 0$ and $b_{i+1} = (\lambda_i s_i - z_i)/\beta$

Example : Returning to our example, we can see that

$$z_0 = (-1)^0(z_0 + y_0) + c_0 - b_0 = 1 \equiv 1 \pmod{10},$$

hence $s_0 = 1$.

$$z_1 = (-1)^1(8 + 2) + 0 - 0 = -10 \equiv 0 \pmod{10},$$

hence $s_1 = 0$. Also, $z_1 < 0$ and therefore $c_2 = 0$ and $b_2 = 1$.

$$z_2 = (-1)^2(9 + 2) + 0 - 1 = 10 \equiv 0 \pmod{10},$$

hence $s_2 = 0$, and so on.

Clearly, the carry/borrow outputs of stage $n - 1$ indicate overflow/underflow, respectively.

Next, we shall specify similar rules for binary systems $\langle n, \beta = 2, A \rangle$. In these systems we may construct truth tables and derive logical expressions. First, we replace the vector A by a binary vector whose elements satisfy $\gamma_i = \frac{1}{2}(1 + \lambda_i)$; $i = 0, 1, \dots, n - 1$. We also define a new variable α , in order to distinguish addition from subtraction, which satisfies $\alpha = 0$ for addition and $\alpha = 1$ for subtraction. Now, the logical expressions to be satisfied are:³

$$s_i = z_i \oplus y_i \oplus (c_i + b_i), \quad (11)$$

$$c_{i+1} = \delta_i(u_i + y_i)c_i + \bar{\delta}_i \bar{u}_i \bar{y}_i c_i + \delta_i u_i y_i \bar{b}_i, \quad (12)$$

$$b_{i+1} = \bar{\delta}_i(u_i + y_i)b_i + \delta_i \bar{u}_i \bar{y}_i b_i + \bar{\delta}_i u_i y_i \bar{c}_i, \quad (13)$$

where $u_i = z_i \oplus \alpha$ and $\delta_i = \gamma_i \oplus \alpha$.

³ + : or, . : and, $\oplus$: eor (i.e. exclusive or)

where only one addition with carry propagation is needed.

2.3 Arithmetic operations

2.3.1 Addition and Subtraction

We present a single algorithm for addition/subtraction which can be applied to all the radix number systems of the class previously mentioned.

Let X and Y be two numbers to be added or subtracted. In a general case of a radix number system, the corresponding vector A has entries which are either $+1$ or -1 . This implies that we have a "mixture" of positive and negative digit weights in our system, and therefore we have to use *borrow*s in addition, and *carries* in subtraction. This in turn, means that there is no difference between addition and subtraction, and therefore we may apply the same rules for $S = X \pm Y$. These rules have to specify the values of the sum digit s_i , the carry c_{i+1} , and the borrow b_i . The following equation should be satisfied

$$\lambda_i(x_i \pm y_i) + c_i - b_i = \lambda_i s_i + \beta c_{i+1} - \beta b_{i+1} ; i = 0, 1, \dots, n-1 \quad (9)$$

where, of course, $c_0 = b_0 = 0$.

Example :

Suppose that A is the nega-decimal system, i.e. $\beta = 10$ and $\lambda_i = (-1)^i$ for every i , and we want to perform the addition $91980 + 22221$. At the third stage, we shall have :

$$\lambda_3(x_3 + y_3) + c_3 - b_3 = \lambda_3 s_3 + \beta c_4 - \beta b_4 ; \text{ i.e. } -1 \cdot (1 + 2) + 1 - 0 = (-1) \cdot 2 + 10 \cdot 0 - 10 \cdot 0.$$

Now let $z_i := \lambda_i(x_i \pm y_i) + c_i - b_i$; then the rule for s_i becomes

$$s_i = (\lambda_i \cdot z_i) \bmod \beta \quad (10)$$

For c_{i+1} and b_{i+1} we have two cases:

Case 1: $z_i \geq \lambda_i s_i$; then $b_{i+1} = 0$ and $c_{i+1} = (z_i - \lambda_i s_i)/\beta$

Case 2: $z_i < \lambda_i s_i$; then $c_{i+1} = 0$ and $b_{i+1} = (\lambda_i s_i - z_i)/\beta$

Example : Returning to our example, we can see that

$$z_0 = (-1)^0(z_0 + y_0) + c_0 - b_0 = 1 \equiv 1 \pmod{10},$$

hence $s_0 = 1$.

$$z_1 = (-1)^1(8 + 2) + 0 - 0 = -10 \equiv 0 \pmod{10},$$

hence $s_1 = 0$. Also, $z_1 < 0$ and therefore $c_2 = 0$ and $b_2 = 1$.

$$z_2 = (-1)^2(9 + 2) + 0 - 1 = 10 \equiv 0 \pmod{10},$$

hence $s_2 = 0$, and so on.

Clearly, the carry/borrow outputs of stage $n - 1$ indicate overflow/underflow, respectively.

Next, we shall specify similar rules for binary systems $\langle n, \beta = 2, \lambda \rangle$. In these systems we may construct truth tables and derive logical expressions. First, we replace the vector λ by a binary vector whose elements satisfy $\gamma_i = \frac{1}{2}(1 + \lambda_i)$; $i = 0, 1, \dots, n - 1$. We also define a new variable α , in order to distinguish addition from subtraction, which satisfies $\alpha = 0$ for addition and $\alpha = 1$ for subtraction. Now, the logical expressions to be satisfied are:³

$$s_i = z_i \oplus y_i \oplus (c_i + b_i), \quad (11)$$

$$c_{i+1} = \delta_i(u_i + y_i)c_i + \bar{\delta}_i \bar{u}_i \bar{y}_i c_i + \delta_i u_i y_i \bar{b}_i, \quad (12)$$

$$b_{i+1} = \bar{\delta}_i(u_i + y_i)b_i + \delta_i \bar{u}_i \bar{y}_i b_i + \bar{\delta}_i u_i y_i \bar{c}_i, \quad (13)$$

where $u_i = z_i \oplus \alpha$ and $\delta_i = \gamma_i \oplus \alpha$.

³ + : or, . : and, $\oplus$: xor (i.e. exclusive or)

These general equations can be applied to any given binary system $\langle n, \beta = 2, A \rangle$ ([13]).

Example :

We shall verify the example of the negabinary system given at the previous section using the equations (11)-(13). The results are summarized below :

i	λ_i	x_i	y_i	c_i	b_i	γ_i	δ_i	u_i	s_i
0	1	0	1	0	0	1	1	0	1
1	-1	0	0	0	0	0	0	0	0
2	1	1	1	0	0	1	1	1	0
3	-1	0	0	1	0	0	0	0	1
4	1	1	1	1	0	1	1	1	1
5	-1	1	0	1	0	0	0	1	0
6	1	0	1	0	0	1	1	0	1
7	-1	0	0	0	0	0	0	0	0
8	1	1	1	0	0	1	1	1	0
9	-1	1	0	1	0	0	0	1	0
10	1	0	1	0	0	1	1	0	1
11	-1	0	0	0	0	0	0	0	0

i.e. $001100110100 + 010101010101 = 010001011001$.

2.3.2 Multiplication and Division

In fixed-radix systems, multiplication and division are usually done as a series of additions (or subtractions) and multiplications or divisions by some integral power of β , which can be obtained by a proper shifting of the intermediate result to the left for multiplication or to the right for division. However, this is not the case (in general) in mixed-radix systems. In such a system, when a number is shifted k positions to the left, the weight of the digit shifted from position i to position

$j = i + k$, is either multiplied by β^k , if $\lambda_i = \lambda_j$, or by $-\beta^k$, if $\lambda_i \neq \lambda_j$. Therefore, in order that a complete shift left operation correspond to some *multiply* operation, all shifted digits have to be multiplied by the same factor, which is either β^k or $-\beta^k$, and k is the number of positions we shift these digits.

To summarize, a k -position shift left operation corresponds to a *multiply* operation in the following cases:

- 1) $\lambda_i = \lambda_{i+k}; \quad i = 0, 1, \dots, n-k-1$
- 2) $\lambda_i \neq \lambda_{i+k}; \quad i = 0, 1, \dots, n-k-1.$

We may observe that in the first case, the values of λ_i 's, which are k positions apart, are the same; i.e. there exists a cycle of length k (in A). For the second case, since the values of λ_i 's which are k positions apart are different, we get that $\lambda_i \neq \lambda_{i+k}$, and $\lambda_{i+k} \neq \lambda_{i+2k}$; but λ_i 's can take only two values, namely $+1$ or -1 ; therefore, $\lambda_i = \lambda_{i+2k}$, i.e. there exists a cycle of length $2k$ in A , for $i = 0, 1, \dots, n-2k-1$.

A similar observation holds for the shift right operations.

From the above discussion, we may conclude that left/right shift operations do not correspond to *multiply/divide* by β operations, unless there exists in A a cycle of length less than or equal to two.

To overcome these difficulties, we follow the modifications given in [13], which enable us to obtain the multiply/divide operation.

Suppose that $(x_{n-1}, x_{n-2}, \dots, x_1, x_0)$ is the representation of a number X in a given number system A and that $(s_{n-1}, s_{n-2}, \dots, s_1, s_0)$ is the representation in A of $S = \beta^a X$, where a equals $+1$ or -1 , depending on whether we multiply or divide by β . The digit s_i is obtained from the shifted digit x_{i-a} , and the carry and the borrow digits c_i and b_i correspondingly, which correct the result of the shift operation whenever $\lambda_i \neq \lambda_{i-a}$. To obtain the equation, which has to be

satisfied by x_{i-a} , s_i , the carry and the borrow digits, we simply replace the $\lambda_i(x_i \pm y_i)$ term in (9) with the term $\lambda_{i-a}x_{i-a}$; i.e.

$$\lambda_{i-a}x_{i-a} + c_i - b_i = \lambda_i s_i + \beta c_{i+1} - \beta b_{i+1}; \quad i = 0, 1, \dots, n-1 \quad (14)$$

where, $c_0 = b_0 = x_{-1} = x_n = 0$.

Example: Suppose that we work in the negabinary system; i.e. $\beta = 2$ and $A = \{(-1)^i = \lambda_i, i = 0, 1, \dots, n-1\}$. We want to multiply the number 1011 by β , i.e. we want to find $S = 2^1 X$; hence $a = 1$.

i	0	1	2	3	4
x_i	1	0	1	1	0
λ_i	1	-1	1	-1	1

Applying the formula (14) to our data, we get:

$$i = 0: \lambda_{-1}x_{-1} + c_0 - b_0 = \lambda_0 s_0 + 2c_1 - 2b_1$$

$$\text{i.e. } 0 + 0 - 0 = 1 \cdot s_0 + 2 \cdot c_1 - 2 \cdot b_1, \text{ and therefore, } s_0 = c_1 = b_1 = 0.$$

$$i = 1: \lambda_0 x_0 + c_1 - b_1 = \lambda_1 s_1 + 2c_2 - 2b_2; \text{ i.e. } 1 \cdot 1 + 0 - 0 = -1 \cdot s_1 + 2 \cdot c_2 - 2 \cdot b_2,$$

and therefore $s_1 = 1$, $c_2 = 1$ and $b_2 = 0$. Similarly,

$$i = 2: -1 \cdot 1 + 1 - 0 = 1 \cdot s_2 + 2 \cdot c_3 - 2 \cdot b_3; \quad \text{i.e. } s_2 = 0, c_3 = 0, b_3 = 0.$$

$$i = 3: 1 \cdot 0 + 0 - 0 = -1 \cdot s_3 + 2 \cdot c_4 - 2 \cdot b_4; \quad \text{i.e. } s_3 = 0, c_4 = 0, b_4 = 0.$$

$$i = 4: -1 \cdot 1 + 0 - 0 = 1 \cdot s_4 + 2 \cdot c_5 - 2 \cdot b_5; \quad \text{i.e. } s_4 = 1, c_5 = 0, b_5 = 1.$$

Hence, the result of the multiplication of 1011, whose algebraic value is -9, by 2 is 110010, which has algebraic value equal to -18. Note that by simply shifting 1011 one position to the left, the result will be 10110 with algebraic value +18 (i.e. multiplication by -2) instead.

Most shift and add type algorithms can be extended to a general number system, if the shift operations are replaced, when necessary, by multiplications or divisions by β . ([13])

2.4 Imaginary-Radix Number Systems

The reason for presenting an imaginary-radix number system in this section, is its possible usefulness in manipulating complex numbers on machines.

The first imaginary-radix number system was introduced by D Knuth ([12]) under the name *quarter-imaginary* number system, using the number $2i$ as its base.

The imaginary-radix system to be presented in this section, is related to the negative-radix number system and the algorithms for arithmetic operations in complex numbers are similar to the ones in the negative-radix system

Any of the previously mentioned systems can serve as a basis for an imaginary system. However, most interesting is the case of the imaginary binary system based upon the 2's complement system since the latter is still the most frequently used one, with numerous arithmetic algorithms and hardware implementations available. The usefulness of this system is its facility of multiplication, as we shall explain later in this section.

Let A be a $2n$ -digit system with an imaginary radix $i\sqrt{\beta}$ ($i = \sqrt{-1}$ and β is a positive integer) and a vector $A = (\lambda_{2n-1}, \dots, \lambda_1, \lambda_0)$. Using (1), the algebraic value of a $2n$ -tuple $(x_{2n-1}, \dots, x_1, x_0)$ is given by:

$$X = \sum_{j=0}^{2n-1} x_j \lambda_j (i\sqrt{\beta})^j \quad (15)$$

The weight w_j of the j th digit x_j is given by:

$$w_j = \lambda_j (i\sqrt{\beta})^j = \begin{cases} \lambda_j \beta^{j/2}, & \text{if } j = 0 \bmod 4; \\ i\lambda_j \sqrt{\beta} \beta^{(j-1)/2}, & \text{if } j = 1 \bmod 4; \\ -\lambda_j \beta^{j/2}, & \text{if } j = 2 \bmod 4; \\ -i\lambda_j \sqrt{\beta} \beta^{(j-1)/2}, & \text{if } j = 3 \bmod 4; \end{cases} \quad (16)$$

One may observe that the weights of the odd-positioned digits are imaginary, and the weights of the even-positioned digits are real. This fact enables us to partition the complex number X into imaginary and real parts. More precisely, X can be written in the form

$$X = \sum_{j \text{ even}} x_j \lambda_j (-1)^{j/2} \beta^{j/2} + i\sqrt{\beta} \sum_{j \text{ odd}} x_j \lambda_j (-1)^{(j-1)/2} \beta^{(j-1)/2} \quad (17)$$

If we replace j by $2k$ for the real part, and by $2k+1$ for the imaginary part, we get

$$X = \sum_{k=0}^{n-1} x_{2k} \lambda_{2k} (-1)^k \beta^k + i\sqrt{\beta} \sum_{k=0}^{n-1} x_{2k+1} \lambda_{2k+1} (-1)^k \beta^k := \text{Re}[X] + i \cdot \text{Im}[X] \quad (18)$$

i.e. the real part belongs to the system $\langle n, \beta, M_{\text{Re}} = \{\mu_k = (-1)^k \lambda_{2k}\} \rangle$, and the imaginary part -when divided by $\sqrt{\beta}$ - belongs to the system $\langle n, \beta, M_{\text{Im}} = \{\mu_k = (-1)^k \lambda_{2k+1}\} \rangle$. Different choices of Λ result in representation of the real and imaginary parts in one of the systems we considered in previous sections. In particular, when $\Lambda = \{\lambda_j = 1\}$, both parts (real and imaginary) are represented in the negative-radix number system.

The representations of the real and imaginary parts are interleaved in the $2n$ -tuple, and therefore we can handle each part separately applying all the properties and the algorithms for the arithmetic operations presented in the previous sections.

Next, we consider the *bi-imaginary* system $\langle 2n, i\sqrt{\beta}, \Lambda \rangle$ based upon the 2's complement system; this means that the real and imaginary parts are presented in the $\langle n, 2, M = \{-1, 1, 1, \dots, 1\} \rangle$ system.

The bi-imaginary number system has a few advantages. One of them is that the real and imaginary parts have a nearly symmetric range; i.e. overflow is not likely to occur when changing a number x to $-x$. Another advantage is the simplicity in detecting the signs (of the real and imaginary parts); only the most significant bits (*sign bits*) have to be checked.

To obtain the 2's complement representations for the real and imaginary parts, we select the vector A as follows: The weights w_{2n-1} and w_{2n-2} must be negative (i.e. the most significant bits or sign bits); then, according to (16) we get (for $\beta = 2$):

If $n = 2k$ (for some integer k), then $2n-1 = 4k-1 \equiv 3 \pmod{4}$; i.e. $w_{2n-1} = -i\lambda_{2n-1}\sqrt{\beta}\beta^{n-1}$ and since we want it to be negative, $\lambda_{2n-1} = 1$. Also, $2n-2 = 4k-2 \equiv 2 \pmod{4}$; hence $w_{2n-2} = -\lambda_{2n-2}\beta^{n-1}$, i.e. $\lambda_{2n-2} = 1$.

Similarly, if $n = 2k-1$, then $2n-1 = 4k-3 \equiv 1 \pmod{4}$ and therefore $\lambda_{2n-1} = -1$, and finally $2n-2 = 4k-4 \equiv 0 \pmod{4}$; hence $\lambda_{2n-2} = -1$. In summary,

$$\lambda_{2n-1} = \lambda_{2n-2} = \begin{cases} +1, & \text{if } n \text{ is even} \\ -1, & \text{if } n \text{ is odd} \end{cases} \quad (19)$$

All other weights have to be positive; hence

$$\lambda_j = \begin{cases} +1, & \text{if } j = 0, 1 \pmod{4} \\ -1, & \text{if } j = 2, 3 \pmod{4} \end{cases} \quad ; j = 0, 1, 2, \dots, 2n-3 \quad (20)$$

The resulting weights are

$$w_j = \begin{cases} \beta^{j/2}, & \text{if } j \text{ is even} \\ i\sqrt{\beta}\beta^{(j-1)/2} & \text{if } j \text{ is odd.} \end{cases} \quad ; j = 0, 1, \dots, 2n-3 \quad (21)$$

The following example illustrates the conversion of a number from 2's complement representation into the bi-imaginary number system.

Example:

Suppose that we are given the number $Z = 5 + i9\sqrt{2}$ to be represented in the system $\langle n = 10, \beta = i\sqrt{2}, A = (-1, -1, -1, -1, 1, 1, -1, -1, 1, 1) \rangle$.

According to (21) the weights are:

$j :$	0	1	2	3	4	5	6	7
$w_j :$	1	$i\sqrt{2}$	2	$i2\sqrt{2}$	4	$i4\sqrt{2}$	8	$i8\sqrt{2}$

The representation of $Z = 5 + i9\sqrt{2}$ is obtained by interleaving the 5-bit 2's complement representations of the imaginary and real parts; i.e. the 2's complement representation of 5 is 00101 and of 9 is 01001. Therefore, $Z = 5 + i9\sqrt{2}$ can be represented as 0010010011, where the underlined digits come from the imaginary part, and the non-underlined digits come from the real part.

Although attractive for implementation in digital hardware, the bi-imaginary number system still includes a few problems. Namely, the number $Im[Z]/\sqrt{\beta}$ will always be irrational; hence no finite representation can be obtained, while truncating it results in rounding errors.

To avoid the non-terminating representation of $Im[Z]/\sqrt{\beta}$ we shall represent the complex number Z by the complex number Z^* , where

$$Z^* = Re[Z^*] + iIm[Z^*] = Re[Z] + i(\sqrt{2}Im[Z]).$$

Now, to convert to the bi-imaginary representation, we find the radix -2 representations of $A^* = Re[Z]$ and $B^* = (\sqrt{2}Im[Z])/\sqrt{2} = Im[Z]$.

Every complex number Z now has a terminating (exact) bi-imaginary representation Z^* , which differs from Z by the scale factor $\sqrt{2}$ that has been applied to $Im[Z]$. The presence of this scale factor causes changes in the ranges of representable numbers and requires corresponding modifications of arithmetic algorithms.([19])

A division by $\sqrt{\beta}$ must be performed on $Im[Z^*]$ in order to get $Im[Z]$ during or after the conversion of the final results into radix β for output.⁴

Example :

Convert $Z = -5 + 8i$ to modified bi-imaginary radix representation.

$$A^* = Re[Z]_{-2} = 01111$$

$$B^* = Im[Z]_{-2} = 11000$$

Hence, $Z^* = \underline{10110101}$, where the underlined digits come from the imaginary part, and the non-underlined digits come from the real part.

When adding (subtracting) complex numbers, we add (subtract) the real and imaginary parts separately and there is no advantage in representing the numbers in an imaginary-radix number system. The usefulness of these systems is their facility of multiplication.

When multiplying two complex numbers, we can multiply $2n$ -bit complex numbers instead of performing four multiplications of n -bit real numbers and two add/subtract operations. If a shift and add type multiplication algorithm is employed, an ordinary shift can be used for even-indexed multiplier bits and a multiplication by $i\sqrt{2}$ is needed for an odd-indexed multiplier bit.

When a $2n$ -tuple is shifted to the left, the even-positioned bits (i.e. the real part) are shifted to odd positions having the same λ , while the odd-positioned bits (i.e. the imaginary part) are shifted to even positions having the opposite value of λ ; i.e. these bits are multiplied by $-i\sqrt{2}$ instead of $i\sqrt{2}$. This can be corrected by forming the 2's complement of the imaginary part before shifting.

Similarly, when dividing by $i\sqrt{2}$ the 2's complement of the real part is formed before shifting to the right.

Example :

⁴ $\beta = 2$ in our case.

Suppose that we want to multiply the complex numbers $Z_1 = 5 + i2\sqrt{2}$ and $Z_2 = 4 + i\sqrt{2}$

Then, $Re\{Z_1\} = 0101$ and $Im\{Z_1\} = 0010$

$Re\{Z_2\} = 0100$ and $Im\{Z_2\} = 0001$

Hence, $[Z_1] = 00011001$ and $[Z_2] = 00010010$

The product is .

$$\begin{array}{r}
 00010010 \\
 \times 00011001 \\
 \hline
 00010010 \\
 10111010 \\
 00010010 \\
 \hline
 000110100010
 \end{array}$$

The algebraic value of the result is $Z_1 \cdot Z_2 = 16 + i13\sqrt{2}$.

We observe that in order to form the partial result 10111010 we have to shift for three positions, and therefore a correction is required. To obtain the correction described previously, we form the 2's complement of the imaginary part of Z_2 , i.e. we form 1111 and then we *merge* it with the real part of Z_2 .

In addition, the generation of the conjugate of a given complex number is simplified since only the 2's complement of the imaginary part has to be formed.

Summarizing, the existing hardware for 2's complement arithmetic can be used for complex number manipulations. Such hardware includes high-speed adders, multioperand adders, and fast multipliers.

2.5 The Classical Algorithms

In this section we shall illustrate algorithms for the basic arithmetic operations over numbers

with arbitrarily high precision.

We assume that we are working with integers expressed in radix b notation, where b is any given integer greater than or equal to 2, and for extended-precision numbers, b can be regarded as the computer's word size, on a "sign-magnitude oriented" computer; i.e. according to the notation introduced in section 2.2, n is an arbitrarily large integer, $A = (1, 1, \dots, 1)$, and b is any integer ≥ 2 .

The operations to be described are the following :

- 1) Addition/Subtraction of two n -place integers,⁵ giving an n -place integer and a carry;
- 2) Multiplication of an n -place integer by an m -place integer, which will give an $(m + n)$ -place answer;
- 3) Division of an $(m + n)$ -place integer by an n -place integer, which will give an $(m + 1)$ -place quotient and an n -place remainder.

The primitive operations which will help us to perform 1), 2), 3), are given below .

- i) Addition/Subtraction of two one-place integers, giving an one-place integer and a carry;
- ii) Multiplication of two one-place integers, giving a two-place integer;
- iii) Division of a two-place integer by an one-place integer, provided that the quotient is an one-place integer, and giving also a one-place remainder.

For reasons of simplicity, we illustrate the algorithms for the operations 1), 2), 3) mentioned above, in terms of positive integers, since determining the sign (and the resulting modifications) can be easily performed.

We first describe the addition algorithm (of nonnegative integers). I.e. given the nonnegative integers $u_{n-1}u_{n-2}\dots u_1u_0$ and $v_{n-1}v_{n-2}\dots v_1v_0$ expressed in radix b , we form their sum $w_nw_{n-1}\dots w_1w_0$ (expressed in radix b), where w_n denotes the carry digit and as such, it will always

⁵ Under the conventional positional notation, any integer expressed in radix b , which is less than b^n , can be represented in at most n places.


```

    carry = 0
    while j ≤ n - 1 do
        begin
            wj = (uj - vj + carry) mod b           (subtract digits)
            carry := (uj - vj + carry) div b       (find new "carry")
            j := j + 1                               (increment position)
        end;
    end                                           (terminate)

```

Again, we observe that $0 - (b - 1) + (-1) \leq u_j - v_j + \text{carry} \leq (b - 1) - 0 + 0$,

i.e. $-b \leq u_j - v_j + \text{carry} < b$, and therefore *carry* can only be -1 or 0 .

At the termination stage, we always have *carry* = 0; the only case in which *carry* = -1 occurs, is when $u_{n-1} \dots u_1 u_0 < v_{n-1} \dots v_1 v_0$, which contradicts our hypothesis.

The reason we describe both Addition and Subtraction algorithms is that it is generally better to use two different routines in computer programs, so that the inner loop of computation can be performed as rapidly as possible (D.Knuth [11]).

The similarities of the addition and subtraction algorithms given above to the general ones (given in section 2.3.1), are quite obvious; the equations (9) and (10) are certainly satisfied by the algorithms we have just described. (Note that $\lambda_i = 1$, $i = 0, 1, \dots, n - 1$, while in the case of the subtraction algorithm, *carry* = b_i).

The third algorithm to be presented is the Multiplication algorithm (of nonnegative integers), which accepts as input two integers $u = (u_{n-1} u_{n-2} \dots u_1 u_0)$ and $v = (v_{m-1} v_{m-2} \dots v_1 v_0)$ expressed in radix b , and forms their product $w = (w_{m+n-1} \dots w_1 w_0)$. Working by hand, we usually form $(u_{n-1} \dots u_1 u_0) \times v_j$ for $0 \leq j \leq m - 1$, and then we add these partial products together with an appropriate scale factor. However, in computers it is best to perform the addition and the

multiplication at the same time

The algorithm to be presented, also checks for zero multipliers, since this test results in less running time, as we can avoid a few operations

Algorithm Multiplication

```

begin
  for  $l := 0$  to  $n - 1$  do  $w_l := 0$ 
   $j := 0$  (take the least significant digit of  $v$ )
  repeat
    if  $v_j = 0$  then  $w_j := 0$  (test for zero multiplier)
    else begin
       $i := 0$  (take the least significant position of  $u$ )
      carry := 0
      repeat
         $t := u_i \cdot v_j + w_{i+j} + \text{carry}$  (multiply and add)
         $w_{i+j} := t \bmod b$  (find new digits)
        carry :=  $t \div b$  (find new carry)
         $i := i + 1$  (increment position)
      until  $j = n$ 
       $w_j := \text{carry}$ 
    end;
     $j := j + 1$  (increment position)
  until  $j = m$ 
end;
```

For this algorithm the carry is always in the range $0 \leq \text{carry} < b$ and also $0 \leq t < b^2$, since

$$t = u_j \cdot v_j + w_{j+1} + \text{carry} \leq (b-1)(b-1) + (b-1) + (b-1) = b^2 - 1 < b^2$$

The last algorithm to be described in this section is the Division algorithm, which divides an $(n+m)$ -place integer by an n -place integer (both of them being nonnegative).

The whole process can be broken down into simpler steps. At each step we divide an $(n+1)$ -place number u by an n -place divisor v , where $0 \leq (u/v) < b$, and since the remainder of this particular division is less than v , it may be used in the succeeding steps.

Example :

We are asked to divide 39524 by 51.

Step 1: Divide 395 by 51; Quotient=7 and Remainder=38;

Step 2: Divide 382 by 51; Quotient=7 and Remainder=25;

Step 3: Divide 254 by 51; Quotient=4 and Remainder=50.

Hence, dividing 39524 by 51, we get Quotient=774 and Remainder= 50.

In general, the problem we face, is given the nonnegative integers $u = (u_n u_{n-1} \dots u_1 u_0)$ and $v = (v_{n-1} \dots v_1 v_0)$ (expressed in radix b), such that $u/v < b$, how to find an algorithm to determine the quotient $q = u \text{ div } v$ (i.e. $\lfloor u/v \rfloor$).

However, we may observe that $u/v < b$ is equivalent to $u/b < v$ (i.e. $u \text{ div } b < v$), which means that $u_{n-1} \dots u_0 < v_{n-1} \dots v_0$.

As when dividing by hand, unfortunately we can't avoid guessing; and this will be based on the most significant digits of u and v . We first divide the two leading digits of u by the leading digit of v , and if the quotient of this division is less than $b-1$, then this will be the leading digit for the quotient of the division of u by v ; otherwise $b-1$ will be the leading digit of the division of u by v . Then, we proceed in the same manner.

Hence, for the most significant digits u_n, u_{n-1} of u and v_{n-1} of v , we calculate

$$\hat{q} = \min((u_n b + u_{n-1}) \operatorname{div} v_{n-1}, b - 1) \quad (22)$$

and as D Knuth proves in [11], $\hat{q} \geq \frac{u}{v}$ and if in addition $v_{n-1} \geq b \operatorname{div} 2$, then $\hat{q} - 2 \leq q \leq \hat{q}$, i.e. no matter how large b is, the trial quotient $\hat{q}$ will never be more than 2 in error

One way to ensure that $v_{n-1} \geq b \operatorname{div} 2$ holds, is to multiply both numbers (u and v) by $\lceil b/(v_{n-1} + 1) \rceil$, which although does not affect the value of u/v , it has been proved ([11]) that it will always make the new value of v_{n-1} large enough

The algorithm which follows, accepts two nonnegative integers $u = (u_{m+n-1} \dots u_1 u_0)$ and $v = (v_{n-1} \dots v_1 v_0)$ (expressed in radix b), and forms the quotient $q = u/v = u \operatorname{div} v = q_m \dots q_1 q_0$ and the remainder $r = u \operatorname{mod} v = r_{n-1} r_{n-2} \dots r_1 r_0$

Algorithm Division

begin

$d := b \operatorname{div} (v_{n-1})$

Normalization

$(u_{m+n} \dots u_1 u_0) := (u_{m+n-1} \dots u_0) \cdot d$

$(v_{n-1} \dots v_1 v_0) := (v_{n-1} \dots v_0) \cdot d$

for $j = m$ **downto** 0 **do**

begin

if $u_{j+n} = v_{n-1}$ **then** $\hat{q} := b - 1$;

$\hat{q} := (u_{j+n} b + u_{j+n-1}) \operatorname{div} v_{n-1}$

calculate $\hat{q}$

while $v_{n-2} \hat{q} \leq (u_{j+n} b + u_{j+n-1} - \hat{q} v_{n-1}) \cdot b + u_{j+n-2}$

do $[\hat{q} := \hat{q} - 1]$

$(u_{j+n} \dots u_j) := (u_{j+n} \dots u_j) - \hat{q} \cdot (v_{n-1} \dots v_0)$

multiply and subtract

$q_j := \hat{q}$

```

    If  $(u_{j+n} \dots u_j) < 0$  then  $q_j := q_j - 1$ ;
         $(u_{j+n} \dots u_j) := (u_{j+n} \dots u_j) + (0v_{n-1} \dots v_0)$ 
    end;

     $q := (q_m \dots q_1 q_0)$  set quotient
     $r := ((u_{n-1} \dots u_1 u_0)/d)$  set remainder
end;

```

In line 11 of the algorithm, we try to keep the digits of the number $(u_{j+n} \dots u_j)$ positive. However, if the result of this subtraction is negative, then $(u_{j+n} \dots u_j)$ should be left as the true value plus b^{n+1} (b 's complement) and a borrow also occurs, which cancels with the carry which occurs in line 13 of the algorithm.

In order to avoid the signed-magnitude notation, we may use the b 's complement notation to represent the negative numbers. For a multiple-precision number, the sign digits of all but the most significant word must be zero; this will prevent us from possible trouble.

Example :

Using 5-bit words the 2's complement number 11001 1100 0011 can be considered as $-1 \cdot 2^{12} + (1001)_2 \cdot 2^8 + (1100)_2 \cdot 2^4 + (0011)_2$.

Since the sign appears only in the first word, addition and subtraction appear to be a little easier, while multiplication and division seem to work better using nonnegative numbers and doing suitable complementation to ensure that both operands are nonnegative ([11]).

2.6 Digital Methods

Throughout this section we shall work with integers expressed in binary notation.

Let $u = (u_{n-1} u_{n-2} \dots u_0)$ and $v = (v_{n-1} v_{n-2} \dots v_0)$ be two n -bit numbers. Without loss of generality, we may assume that n is a power of 2.

The traditional method for multiplying two n -bit numbers, requires $O(n^2)$ bit operations. Fortunately, we may reduce this number of bit operations as follows

We partition u and v into two halves as shown below

where a = the most significant half of u
 b = the least significant half of u
 c = the most significant half of v
 d = the least significant half of v

Treating each half as an $(n/2)$ -bit-number, we can express the product of u and v as follows

$$u \cdot v = (a \cdot 2^{n/2} + b) \cdot (c \cdot 2^{n/2} + d) = ac \cdot 2^n + (ad + bc) \cdot 2^{n/2} + bd \quad (23)$$

The above formula requires four multiplications of $(n/2)$ -bit numbers plus some additions and shifts (multiplications by powers of 2).

However, the same result can be obtained with fewer multiplications as shown below

Let

$$\begin{aligned} x &= (a + b) \cdot (c + d) \\ y &= a \cdot c \\ w &= b \cdot d \end{aligned} \quad (24)$$

Then,

$$u \cdot v = y \cdot 2^n + (x - y - w) \cdot 2^{n/2} + w. \quad (25)$$

The numbers $(a + b)$ and $(c + d)$ may be $n/2 + 1$ -bit numbers (due to carry) but the multiplication $(a + b) \cdot (c + d)$ may be reduced to $n/2$ -bit multiplication. We write

$$a + b = l_1 \cdot 2^{n/2} + r_1, \quad c + d = l_2 \cdot 2^{n/2} + r_2, \quad l_i = 0, 1,$$

and then the product

$$(a + b) \cdot (c + d) = (l_1 \cdot l_2) \cdot 2^n + [(l_1 \cdot r_2) + (r_1 \cdot l_2)] \cdot 2^{n/2} + (r_1 \cdot r_2)$$

can be formed using an $n/2$ -bit multiplication together with some auxiliary operations which take linear time

Hence the formula (25) allows us to compute an n -bit product using three $n/2$ -bit products and additions and shift operations. Applying the method recursively, the complexity of multiplying two n -bit numbers is given by

$$T(n) \leq \begin{cases} k, & \text{for } n = 1 \\ 3T(n/2) + kn, & \text{for } n > 1, \end{cases} \quad (26)$$

where $T(n)$ is the time taken for the multiplication of two n -bit numbers, and k is a constant for the additions and shift operations.

We can prove by induction that the solution for the recursive relation given above, is

$$T(n) = 3kn^{\log_2 3} - 2kn \leq 3kn^{\log_2 3},$$

i.e. $T(n) = O(n^{\log_2 3})$, which is significantly better than $O(n^2)$, especially when n is large enough.

The running time can be reduced even further as follows:

We examine the more general case where we can break the numbers to be multiplied into $(r + 1)$ pieces, for any fixed r (not just $r = 1$). Again, we suppose that $u = (u_{(r+1)n-1} \dots u_1 u_0)$ and $v = (v_{(r+1)n-1} \dots v_1 v_0)$ are the binary representations of u and v .

We break u into the pieces $U_r, U_{r-1}, \dots, U_1, U_0$, and v into the pieces $V_r, V_{r-1}, \dots, V_1, V_0$, each of which is an n -bit number. Now, we can write u and v in the following forms

$$u = U_r 2^{rn} + \dots + U_1 2^n + U_0$$

and

$$v = V_r 2^{rn} + \dots + V_1 2^n + V_0$$

Next, we take the polynomials

$$U(x) = U_r x^r + U_{r-1} x^{r-1} + \dots + U_1 x + U_0$$

$$V(x) = V_r x^r + V_{r-1} x^{r-1} + \dots + V_1 x + V_0 \quad (27)$$

and we observe that $u = U(2^n)$ and $v = V(2^n)$

Since our objective is to multiply u and v , the above observations suggest that we can find the product of $U(x)$ and $V(x)$, say $W(x)$, and then in order to find $u \cdot v$, we just have to evaluate $W(2^n)$. More precisely, let

$$W(x) = U(x) \cdot V(x) = W_{2r} x^{2r} + \dots + W_1 x + W_0 \quad (28)$$

Then, $u \cdot v = U(2^n)V(2^n) = W(2^n)$; and this means that we have to find an efficient way to compute the coefficients of $W(x)$; we shall give a way using just $2r + 1$ multiplications of n -bit numbers, plus, of course, some other operations which require time proportional to n .

It is known that a polynomial of degree $n - 1$ can be uniquely represented by the list of its values at n distinct points, and its coefficients can be written as a linear combination of these distinct values. Since the polynomial $W(x)$ is of degree $2r$, we have to find $2r + 1$ distinct values. Hence, we take x to be $0, 1, 2, \dots, 2r$ and :

$$W(0) = U(0)V(0), W(1) = U(1)V(1), \dots, W(2r) = U(2r)V(2r).$$

i.e. it will take $2r + 1$ multiplications of n -bit numbers, plus time proportional to n , in order to form the linear combinations which will give us the coefficients of $W(x)$

Actually, the products $U(x)V(x)$ for $x = 0, 1, 2, \dots, 2r$, are not products of strictly n -bit numbers, but rather of $(n + t)$ -bit numbers, where t is a constant which depends on r , and as D.Knuth shows in [11], we can design a multiplication routine for $(n + t)$ -bit numbers, which requires only $T(n) + cn$ operations. Similar arguments to those above with $r = 1$ show that the method, when applied recursively, has execution time

$$T(n) \leq k n^{\log_{r+1}(2r+1)}, \quad (29)$$

where k is a constant depending on r alone.

The following example will illustrate the method for finding the coefficients of $W(x)$ from the $2r + 1$ distinct values of $W(x)$.

Example :

Suppose we want to find the product $u \cdot v$, where $u = 2786$ and $v = 1715$. We first convert these numbers into binary notation; hence, $u = (101011100010)_2$ and $v = (011010110011)_2$. Suppose that $r = 2$; i.e. we break the number into $2 + 1 = 3$ parts of length $n = 4$ each.

$$u = (1010 \ 1110 \ 0010)_2$$

$$v = (0110 \ 1011 \ 0011)_2$$

The corresponding polynomials $U(x)$ and $V(x)$, are

$$U(x) = 10x^2 + 14x + 2$$

$$V(x) = 6x^2 + 11x + 3 \quad (30)$$

In order to find the coefficients of $W(x) = U(x)V(x)$, we evaluate $W(x)$ in $2 \cdot 2 + 1 = 5$ distinct

points, namely 0, 1, 2, 3, and 4. I.e.

$$\begin{aligned} U(0) &= 2 & U(1) &= 26 & U(2) &= 70 & U(3) &= 134 & U(4) &= 218 \\ V(0) &= 3 & V(1) &= 20 & V(2) &= 49 & V(3) &= 90 & V(4) &= 143 \\ W(0) &= 6 & W(1) &= 520 & W(2) &= 3430 & W(3) &= 12060 & W(4) &= 31174 \end{aligned} \quad (31)$$

Next, we apply the interpolation algorithm given in [11], which computes the coefficients of a polynomial $W(x) = W_{m-1}x^{m-1} + \dots + W_1x + W_0$, when the values $W(0), W(1), \dots, W(m-1)$ are given. More precisely, let

$$W(x) = \theta_{m-1}x^{m-1} + \theta_{m-2}x^{m-2} + \dots + \theta_1x + \theta_0 \quad (32)$$

where $x^k = x(x-1)\dots(x-(k-1))$, and θ_j 's, W_j 's are known.

We may observe that

$$\begin{aligned} W(x+1) - W(x) &= [\theta_{m-1}(x+1)^{m-1} + \theta_{m-2}(x+1)^{m-2} + \dots + \theta_1(x+1) + \theta_0] \\ &\quad - [\theta_{m-1}x^{m-1} + \dots + \theta_1x + \theta_0] \\ &= [\theta_{m-1}(x+1)x(x-1)\dots(x-m+3) + \theta_{m-2}(x+1)x(x-1)\dots(x-m+4) + \dots + \theta_0] \\ &\quad - [\theta_{m-1}x(x-1)\dots(x-m+2) + \dots + \theta_0] \\ &= [(m-1)\theta_{m-1}x(x-1)\dots(x-m+3) + (m-2)\theta_{m-2}x(x-1)\dots(x-m+4) + \dots + \theta_1] \\ &= (m-1)\theta_{m-1}x^{m-2} + (m-2)\theta_{m-2}x^{m-3} + \dots + \theta_1. \end{aligned} \quad (33)$$

and by induction we find that

$$\begin{aligned} \frac{1}{k!} (W(x+k) - \binom{k}{1}W(x+k-1) + \binom{k}{2}W(x+k-2) - \dots + (-1)^k W(x)) &= \binom{m-1}{k} \theta_{m-1} x^{m-1-k} + \\ &\quad + \binom{m-2}{k} \theta_{m-2} x^{m-2-k} + \dots + \binom{k}{k} \theta_k. \end{aligned} \quad (34)$$

But it is known that we may write

$$\Delta W(x) = W(x+1) - W(x) \quad (35)$$

where x is an integer, and in general

$$\Delta^k W(x) = \Delta^{k-1} W(x+1) - \Delta^{k-1} W(x); \quad k = 2, 3, \dots \quad (36)$$

Using (35) and (36), we get the following formula

$$\Delta^k W(x) = \sum_{l=0}^k (-1)^{k-l} \binom{k}{l} W(x+l) \quad (37)$$

This fact, enables us to write the left hand side of (34) in the form $(1/k!) \Delta^k W(x)$, and to observe that

$$\frac{1}{k!} \Delta^k W(x) = \frac{1}{k} \left(\frac{1}{(k-1)!} \Delta^{k-1} W(x+1) - \frac{1}{(k-1)!} \Delta^{k-1} W(x) \right) \quad (\text{from (40)})$$

and also that $(1/k!) \Delta^k W(0) = \theta_k$, (see (33) for $k = 1$).

Therefore, the coefficients of (32), in our example can be evaluated as follows :

$W(0) = 6$				
	514			
$W(1) = 520$		$2398/2 = 1198$		
	2910		$1662/3 = 554$	
$W(2) = 3430$		$5720/2 = 2860$		$240/4 = 60$ (38)
	8630		$2382/3 = 794$	
$W(3) = 12060$		$10484/2 = 5242$		
	19114			
$W(4) = 31174$				
$k = 0$	$k = 1$	$k = 2$	$k = 3$	$k = 4$

I.e. the k -th column is obtained by computing the difference between successive values of the $k-1$ column and dividing by k . The coefficients of (32) appear at the top of the columns; i.e.

$\theta_0 = 6$, $\theta_1 = 514$, $\theta_2 = 1198$, $\theta_3 = 554$, and $\theta_4 = 60$. Therefore,

$$\begin{aligned} W(x) &= 60x^4 + 554x^3 + 1198x^2 + 514x^1 + 6 \\ &= (((60(x-3) + 554)(x-2) + 1198)(x-1) + 514)x + 6 \end{aligned} \quad (39)$$

In general,

$$W(x) = (\dots((\theta_{m-1}(x-m+2) + \theta_{m-2})(x-m+3) + \theta_{m-3})(x-m+4) + \dots + \theta_1)x + \theta_0,$$

and this formula suggests the following way to obtain $W_{m-1}, \dots, W_1, W_0$:

60	554				
	-3 · 60				
60	374	1198			
	-2 · 60	-2 · 374			
60	254	450	514		
	-1 · 60	-1 · 254	-1 · 450		
60	194	196	64	6	(40)

In this table, the numbers below the horizontal lines, show the coefficients of θ_{m-1} , $\theta_{m-1}(x-m+2) + \theta_{m-2}$, ... etc.

The coefficients of $W(x) = U(x) \cdot V(x)$ are given at the last row of the table (40). I.e. $W(x) = 60x^4 + 194x^3 + 196x^2 + 64x + 6$.

The final step, is to compute $W(2^4)$, which will give us the product of $u = 2786$ by $v = 1715$. (We compute $W(2^4)$, because each subpart of u and v -in binary notation- has length equal to 4 as we have previously seen.)

Indeed, $W(2^4) = 4777990$, which is equal to 2786×1715 . Note that because of the Stirling formula $x^n = \binom{n}{n}x^n + \dots + \binom{n}{1}x^1 + \binom{n}{0}$, the coefficients of $W(x)$ are nonnegative, so are θ_j 's, and hence all of the intermediate results above are nonnegative.

We shall now present the Toom-Cook algorithm (more precisely, the high-precision multiplication algorithm) for multiplying two n -bit integers u and v (in binary notation). The algorithm repeatedly breaks u and v into $(r + 1)$ parts, unless u and v have a number of digits less than or equal to some constant δ ,⁶ in which case it simply multiplies them using a built-in routine which multiplies two δ -bit numbers.

An asymptotically *optimum* choice of r , which gives an execution time for the Toom-Cook algorithm of order $O(n^{2^{3.5\sqrt{\log_2 n}}})$, is to take $r = 2^{\lfloor \sqrt{\log_2 n} \rfloor}$. The dedicate analysis for this proper choice of r as well as a stack-oriented implementation of the Toom-Cook algorithm can be found in [11]

Our implementation is in terms of a recursive procedure and is given below in a pseudo-code format. (Without loss of generality, we may assume that the number of digits of both u and v is equal to n).

Toom-Cook high precision multiplication algorithm of binary numbers :

Multiply(u, v, n, w)

begin

if $n > \delta$ **then**

begin

$r := 2^{\lfloor \sqrt{\log_2 n} \rfloor}$

split u and v into $r+1$ parts

for $i = 0$ **to** $2r$ **do**

begin

find $U(i)$

find $V(i)$

set $m = \text{maximum number of digits of } U(i), V(i)$

⁶ δ may vary in different computers.

multiply($U(i), V(i), m, W(i)$)

end;

calculate $W(x)$ by interpolation using $W(0), \dots, W(2r)$

Evaluate $W(2^q)$

end

else set $w = u * v$ (using a built-in routine for multiplying two δ -bit numbers)

end;

Chapter 3

Modular Arithmetic

3.1 The Modular approach.

Another interesting method for doing arithmetic on large integer numbers uses residue arithmetic. That is, instead of representing an integer by a fixed-radix notation, we represent it by its residues modulo a set of pairwise relatively prime integers.

If $m_0, m_1, m_2, \dots, m_{r-1}$ are pairwise relatively prime integers and $m = m_0 m_1 \dots m_{r-1} = \prod_{i=0}^{r-1} m_i$, then we can represent any integer u , $0 \leq u < m$, uniquely by the set of residues $u_0, u_1, \dots, u_{r-1}$, where $u_i = u \text{ modulo } m_i$, $0 \leq i \leq r-1$. Hence, for given $m_0, m_1, \dots, m_{r-1}$, we write that $u \longleftrightarrow (u_0, u_1, \dots, u_{r-1})$, and it is easy to compute $(u_0, u_1, \dots, u_{r-1})$ from an integer u by means of division, while, as we shall see later in this section, we can always recompute u from $(u_0, u_1, \dots, u_{r-1})$ (provided of course that u is not too large).

The obvious advantage of the residue arithmetic is that it is easy to do addition, subtraction and multiplication, provided that the results continue to lie between 0 and $m-1$ (or these calculations can be regarded as being done *modulo* m).

I.e. given $(u_0, u_1, \dots, u_{r-1})$ and $(v_0, v_1, \dots, v_{r-1})$ as the *modular* representations of u and v correspondingly, we add, subtract and multiply them as follows :

$$(u_0, u_1, \dots, u_{r-1}) + (v_0, v_1, \dots, v_{r-1}) = ((u_0 + v_0) \bmod m_0, (u_1 + v_1) \bmod m_1, \dots, (u_{r-1} + v_{r-1}) \bmod m_{r-1})$$

$$(u_0, u_1, \dots, u_{r-1}) - (v_0, v_1, \dots, v_{r-1}) = ((u_0 - v_0) \bmod m_0, (u_1 - v_1) \bmod m_1, \dots, (u_{r-1} - v_{r-1}) \bmod m_{r-1})$$

$$(u_0, u_1, \dots, u_{r-1}) \cdot (v_0, v_1, \dots, v_{r-1}) = ((u_0 \cdot v_0) \bmod m_0, (u_1 \cdot v_1) \bmod m_1, \dots, (u_{r-1} \cdot v_{r-1}) \bmod m_{r-1}) \quad (1)$$

since all these formulas are basic facts of number theory.

The range of the numbers that can be handled using modular arithmetic, is equal to the product of the numbers $m_0, m_1, \dots, m_{r-1}$, i.e. equal to m .

Another advantage of the modular representation is that we can implement arithmetic with less hardware than is required for conventional arithmetic, since we may perform the calculations for each modulus independently of the others, and if these calculations take place simultaneously, then we obtain savings of execution time in all three operations (addition, subtraction, multiplication), this cannot be achieved using the conventional methods because of the consideration which has to be taken for carry propagation.

On the other hand, modular arithmetic has also disadvantages, which have to be considered.

The residue number system is an integer number system, and this fact makes it inconvenient for fractional quantities. It is not clear how to do division economically using modular arithmetic. Given two integers u and v , then u/v is not necessarily an integer and even if it were, we could not necessarily find its modular representation by computing $(u_i/v_i) \text{ modulo } m_i$, for each i . For example, suppose that m_i is not a prime number, say $m_i = 8$, and take u_i and v_i to be $u_i = 4$, $v_i = 4$. Then w can be either 1 or 3 or 5 or 7, since $1 \times 4 = 3 \times 4 = 5 \times 4 = 7 \times 4 \equiv 4 \text{ modulo } 8$. i.e. in general, there are many integers w between 0 and $m_i - 1$ which could be $(u_i/v_i) \text{ modulo } m_i$, i.e. $wv_i \equiv u_i \text{ modulo } m_i$.

The residue number system is not a weighted number system, and therefore it does not have their advantages with respect to computers.

One of them is the lack of detecting overflows, i.e. a way to tell whether the result of a given operation is outside the range 0 to $m - 1$, effectively.

Another one is the difficulty of checking whether a number is positive or negative. Various methods have been proposed concerning the representation of negative numbers in the residue number system ([20]), but they are not very efficient.

A third disadvantage, is that we cannot check whether or not $(u_0, u_1, \dots, u_{r-1})$ is greater than $(v_0, v_1, \dots, v_{r-1})$.

3.1.1 Conversion techniques

In order to be able to work in the residue number system, we need algorithms to convert from radix notation to modular notation and back.

L-Conversion from radix notation to modular notation.

One method to convert a given integer u from radix notation to modular notation is to divide u by each of the moduli m_i , $0 \leq i \leq r-1$. Suppose that each m_i is an n -bit number in binary notation, then, according to [1], this technique requires $O_B(r^2 D(n))$ time, where $D(n)$ is the time for integer division (at most $O_B(n \log n \log \log n)$ [1]).

Another method for the computation of residues, as proposed in [1], is the following.

Suppose we are given the moduli $m_0, m_1, \dots, m_{r-1}$ and an integer u , $0 \leq u \leq m-1$, where $m = m_0 m_1 \dots m_{r-1}$. Without loss of generality, we may assume that r is a power of 2, say $r = 2^l$, for some positive integer l ; otherwise we add moduli equal to 1 until this is obtained.

For $0 \leq j < l$, i a multiple of 2^j , and $0 \leq i \leq r-1$, let $q_{i,j} = \prod_{s=i}^{i+2^j-1} m_s$.

Then, $q_{i,0} = m_i$, and $q_{i,j} = q_{i,j-1} \times q_{i+2^{j-1},j-1}$.

We first find the $q_{i,j}$'s then the remainder $u_{i,j}$ when u is divided by each of the $q_{i,j}$'s. The u_i 's, $0 \leq i \leq r-1$, where $u_i \equiv u \text{ modulo } m_i$, are the $u_{i,0}$'s. More precisely, instead of dividing u by each of the m_i 's separately, we form the products $m_0 m_1, m_2 m_3, \dots, m_{r-2} m_{r-1}$, then the products $m_0 m_1 m_2 m_3, m_4 m_5 m_6 m_7, \dots$ and so on. In order to find the residues, we first divide u

by $m_1 m_2 \dots m_{r/2-1}$ and $m_{r/2} m_{r/2+1} \dots m_{r-1}$, to obtain the residues u modulo $m_1 m_2 \dots m_{r/2-1}$ and u modulo $m_{r/2} m_{r/2+1} \dots m_{r-1}$; i.e. we reduce the initial problem into two problems of half size each, and we continue in this way until the required residues are found.

Algorithm I

```

begin
  for  $i = 0$  to  $r - 1$  do [ $q_{i0} := m_i$ ]
  for  $j = 1$  to  $l - 1$  do
    begin
      for  $i = 0$  (step  $2^j$ ) to  $r - 1$ 
        do [ $q_{i,j} = q_{i,j-1} \cdot q_{i+2^{j-1},j-1}$ ]
      end;
       $u_{0l} := u$ 
      for  $j = l$  downto  $1$  do
        begin
          for  $i = 0$  (step  $2^j$ ) to  $(r - 1)$  do
            begin
               $u_{i,j-1} := u_{i,j} \bmod q_{i,j-1}$ ;
               $u_{i+2^{j-1},j-1} := u_{i,j} \bmod q_{i+2^{j-1},j-1}$ 
            end
          end;
          for  $i = 0$  to  $r - 1$  do [ $u_i := u_{i0}$ ]
        end;
      end;
    end;
  end;

```

This algorithm, as shown in [1], requires at most $O_B(nr \log r \log nr \log \log nr)$ time, where n is the length of the binary representation of the m_i 's.

Next, we give an example which illustrates this algorithm.

Example :

We are given $u = 400$ and $m_0 = 3$, $m_1 = 4$, $m_2 = 5$, $m_3 = 7$, and we want to find $u_i \equiv u \text{ modulo } m_i$, $0 \leq i \leq 3$.

Obviously, $0 < u = 400 < m_0 m_1 m_2 m_3 = 420$ Line (2) of the algorithm will give us :

$$q_{00} = 3$$

$$q_{10} = 4$$

$$q_{20} = 5$$

$$q_{30} = 7$$

From lines (3)-(7) we get :

$$j = 1 \quad i = 0 \quad q_{01} = q_{00} \quad q_{10} = 3 \cdot 4 = 12$$

$$j = 1 \quad i = 2 \quad q_{21} = q_{20} \cdot q_{30} = 5 \cdot 7 = 35$$

From line (8) we get $u_{02} = 400$, while lines (9)-(16) will give us :

$$j = 2 \quad i = 0 \quad u_{01} = u_{02} \text{ mod } q_{01} = 400 \text{ mod } 12 = 4$$

$$j = 2 \quad i = 0 \quad u_{21} = u_{02} \text{ mod } q_{21} = 400 \text{ mod } 35 = 15$$

$$j = 1 \quad i = 0 \quad u_{00} = u_{01} \text{ mod } q_{00} = 4 \text{ mod } 3 = 1$$

$$j = 1 \quad i = 0 \quad u_{10} = u_{01} \text{ mod } q_{10} = 4 \text{ mod } 4 = 0$$

$$j = 1 \quad i = 2 \quad u_{20} = u_{21} \text{ mod } q_{20} = 15 \text{ mod } 5 = 0$$

$$j = 1 \quad i = 2 \quad u_{30} = u_{21} \text{ mod } q_{30} = 15 \text{ mod } 7 = 1$$

Finally, line (17) will give us :

$$u_0 = 1 \quad (400 \text{ mod } 3)$$

$$u_1 = 0 \quad (400 \text{ mod } 4)$$

$$u_2 = 0 \quad (400 \text{ mod } 5)$$

$$u_3 = 1 \quad (400 \bmod 7)$$

The number of divisions using this method is not smaller than dividing u by each of the moduli m_i , but the "length" of each particular division is small, and this is the advantage of this method

II.-Conversion from modular notation to radix notation

Suppose that we are given the relatively prime moduli $m_0, m_1, m_2, \dots, m_{r-1}$ and the residues $u_0, u_1, u_2, \dots, u_{r-1}$, where again $r = 2^l$, for some positive integer l , and we want to find an integer u , which will satisfy the conditions $u_i \equiv u \bmod m_i$; $i = 0, 1, 2, \dots, r-1$.

Theorem (Chinese Remainder Theorem). Let $m_0, m_1, \dots, m_{r-1}$ be positive integers which are pairwise prime; i.e. $\gcd(m_i, m_j) = 1$, when $i \neq j$. Let also $m = m_0 m_1 m_2 \dots m_{r-1}$, and $a, u_0, u_1, \dots, u_{r-1}$ be integers. Then, there is exactly one integer u which satisfies the conditions

$$a \leq u < a + m,$$

$$u \equiv u_j \pmod{m_j}; \quad 0 \leq j \leq r-1 \quad (2)$$

*Proof:*¹ This method uses $\binom{r}{2}$ constants c_{ij} , where

$$c_{ij} \equiv 1 \pmod{m_j} \quad 0 \leq i < j \leq r-1 \quad (3)$$

These constants can be computed using the Euclidean Algorithm; since it is known that we can find a and b such that $am_i + bm_j = \gcd(m_i, m_j) = 1$ and we can take $c_{ij} = a$. Once we have found the c_{ij} 's which satisfy (3), we set

$$v_0 := u_0 \bmod m_0$$

$$v_1 := (u_1 - v_0)c_{01} \bmod m_1$$

$$v_2 := ((u_2 - v_0)c_{02} - v_1)c_{12} \bmod m_2$$

¹ Suggested by H.L. Garner in (IRE Transactions EC-8 (1959), pp.140-147)

$$v_{r-1} := (\dots((u_{r-1} - v_0)c_{0,r-1} - v_1)c_{1,r-1} - \dots - v_{r-2})c_{r-2,r-1} \bmod m_{r-1} \quad (4)$$

Then, the number which satisfies the conditions

$$0 \leq u < m, \quad u \equiv u_j \pmod{m_j}, \quad 0 \leq j \leq r-1$$

is given by

$$u = v_{r-1}m_{r-2}m_{r-3}\dots m_0 + \dots + v_2m_1m_0 + v_1m_0 + v_0 \quad (5)$$

In order to verify (5), we observe that $u \equiv v_0 \bmod m_0 \equiv u_0 \bmod m_0$, since all its terms are divisible by m_0 except the last one. Also $u \bmod m_1 \equiv (v_1m_0 + v_0) \bmod m_1 \equiv ((u_1 - v_0)c_{0,1}m_0 \bmod m_1 + v_0) \bmod m_1 \equiv (u_1 - v_0 + v_0) \bmod m_1 \equiv u_1 \bmod m_1$; i.e. $u \equiv u_1 \bmod m_1$, and similarly for the rest of them.

The integer u obtained in (5) is unique. Indeed, suppose that w was another integer, different from u , such that $u \equiv w \pmod{m_i}$ for $0 \leq i \leq r-1$. Then $u - w$ is a multiple of m . This means that there is only one solution to satisfy (2).

Of course, (5) gives u in a mixed-radix representation which can be converted to binary or decimal notation ([11]).

The advantage of this method is that we can compute the v_j 's using arithmetic $\bmod m_j$, which is built into the modular arithmetic algorithms, although we cannot compute all v_j 's at the same time, since each one depends on the previous ones. If the moduli m_i are n -bit numbers, the method requires $O(r^2)$ multiplications of n -bit numbers which take $O(n \log n \log \log n)$ time each; i.e. a total of $O(r^2 n \log n \log \log n)$ time.²

Example :

² For faster methods, see also [1] and [11].

Suppose that we are given $m_0 = 3$, $m_1 = 4$, $m_2 = 5$, $m_3 = 7$, $m_4 = 11$, which are pairwise prime integers, and $u_0 = 1$, $u_1 = 1$, $u_2 = 4$, $u_3 = 3$, $u_4 = 7$. We want to find a number u which will satisfy the conditions

$$0 \leq u < m, \quad u \equiv u_j \pmod{m_j} \quad 0 \leq j \leq 4$$

We first compute the constants c_{ij} which satisfy the condition $c_{ij} \cdot m_i \equiv 1 \pmod{m_j}$, for $0 \leq i < j \leq 4$. I.e.

$$c_{01} \cdot 3 \equiv 1 \pmod{4} \implies c_{01} = 3$$

$$c_{02} \cdot 3 \equiv 1 \pmod{5} \implies c_{02} = 2$$

$$c_{03} \cdot 3 \equiv 1 \pmod{7} \implies c_{03} = 5$$

$$c_{04} \cdot 3 \equiv 1 \pmod{11} \implies c_{04} = 4$$

$$c_{12} \cdot 4 \equiv 1 \pmod{5} \implies c_{12} = 4$$

$$c_{13} \cdot 4 \equiv 1 \pmod{7} \implies c_{13} = 2$$

$$c_{14} \cdot 4 \equiv 1 \pmod{11} \implies c_{14} = 3$$

$$c_{23} \cdot 5 \equiv 1 \pmod{7} \implies c_{23} = 3$$

$$c_{24} \cdot 5 \equiv 1 \pmod{11} \implies c_{24} = 9$$

$$c_{34} \cdot 7 \equiv 1 \pmod{11} \implies c_{34} = 8$$

Next, we compute the v_j 's, following the formulas (4). Hence,

$$v_0 \equiv 1 \pmod{3} = 1$$

$$v_1 \equiv (1 - 1) \cdot 3 \pmod{4} \equiv 0 \pmod{4} = 0$$

$$v_2 \equiv ((4 - 1) \cdot 2 - 0) \cdot 4 \pmod{5} \equiv 24 \pmod{5} = 4$$

$$v_3 \equiv (((3 - 1) \cdot 5 - 0) \cdot 2 - 4) \cdot 3 \pmod{7} \equiv 48 \pmod{7} = 6$$

$$v_4 \equiv (((((7 - 1) \cdot 4 - 0) \cdot 3 - 4) \cdot 9 - 6) \cdot 8 \pmod{11} \equiv 4848 \pmod{11} = 8$$

Therefore, according to (5),

$$u = 8 \cdot (3 \cdot 4 \cdot 5 \cdot 7) + 6 \cdot (3 \cdot 4 \cdot 5) + 4 \cdot (3 \cdot 4) + 0 \cdot (3) + 1 \implies u = 3769.$$

Indeed, $0 < 3769 < 4620 = 3 \cdot 4 \cdot 5 \cdot 7 \cdot 11$ and also,

$$3769 \equiv 1 \pmod{3}$$

$$3769 \equiv 1 \pmod{4}$$

$$3769 \equiv 4 \pmod{5}$$

$$3769 \equiv 3 \pmod{7}$$

$$3769 \equiv 7 \pmod{11}$$

From the Chinese Remainder Theorem, we can see that we may use modular representation for numbers in any consecutive interval of $m = m_0 m_1 m_2 \dots m_{r-1}$ numbers, by adjusting a properly.

On binary computers, it is better to choose $m_j = 2^{e_j} - 1$, i.e. one less than a power of 2, because this simplifies the arithmetic operations since it is easier to work *modulo* $2^{e_j} - 1$, as in one's complement notation (Knuth [11]).

3.1.2 Selection of Moduli

The selection of moduli which are used to represent a residue number, affect the speed and the hardware complexity in the design of a residue system, and because of conflicting conditions, this is not an easy task at all. However, moduli of the form 2^n and $2^n - 1$ have some remarkable advantages. To mention a few, the multiplicative inverses are generally simple to determine, the storage efficiency (which is maximized by nonredundant coding of the residues)³ is very high and, also, there is an easy determination of additive inverses ([20]).

Since the amount of hardware required in a feedback type of adder increases as the logarithm of the modulus ([20]), one could use larger modulus and design a computer system using four moduli: 2^n , $2^n - 1$, $2^{n-1} - 1$, $2^{n-2} - 1$, which are relatively prime, if and only if n is odd.

Another choice of moduli is to select them as large prime numbers. This is a quite good choice

³ I.e. a minimum number of digits is used to represent each modulus

when we want to solve a system of linear equations of the form

$$A \cdot \mathbf{x} = \mathbf{b} \quad (6)$$

If D denotes the determinant of A and m_i the i -th modulo, then the choice of these moduli to be large primes, increases the probability that

$$\gcd(D, m_i) = 1$$

and

$$D \pmod{m_i} \neq 0 \quad ; i = 0, 1, \dots, n-1 \quad (7)$$

But if these two last conditions are satisfied, then A is a nonsingular matrix (*modulo* m_i) and therefore we can solve the system $A \cdot \mathbf{x} = \mathbf{b} \pmod{m_i}$ for $D \pmod{m_i}$ and $\mathbf{y} = \text{Adj}(A) \pmod{m_i}$; otherwise we select another prime for a modulus.

It is also obvious that by selecting the moduli as prime numbers, the condition

$$\gcd(m_i, m_j) = 1 \quad (i \neq j) \quad (8)$$

is always satisfied, and this fact enables us to apply the Chinese Remainder Theorem.

For a computer system which is capable of multiplying or adding any pair of integers x, y such that $|x| < C$ and $|y| < C$, C being a given constant for the particular computer, Newman (in *Solving equations exactly*, J.Res.Nat.Bur.Stds.17B) suggests choosing as moduli the n largest primes less than C . Then, the sum of any two numbers (or their product), when reduced *modulo* m , will be less than C .

This technique prevents large numbers from accumulating during computations and therefore it prevents the possibility of overflow.

3.2 Applications

Some important applications of residue arithmetic are those involved with matrix computa-

tions, and in particular solving a system of linear equations, finding the inverse or the generalized inverse of a matrix, or finding the determinant of it, etc.

An efficient way to obtain the exact rational solution to a nonsingular system $A \cdot x = b$ of linear equations with integer coefficients, is the following ([4]):

- i) Computation of the $\det(A)$ and $\text{adj}(A)$ modulo m_i , for a number of different prime moduli m_i ;
- ii) use of the Chinese Remainder Theorem to combine the results for a modulus $m = m_1 m_2 \dots m_n$ (for an appropriate integer n), and
- iii) application of the Hadamard inequality $|\det(A)|^2 \leq \sum_{j=1}^n |a_{1j}|^2 \sum_{j=1}^n |a_{2j}|^2 \dots \sum_{j=1}^n |a_{nj}|^2$ to $\det(A)$ and the entries of $(\text{adj}(A)) \cdot b$ to deduce the required rational solution from the congruential values.

The advantage of residue arithmetic is that, for suitable choice of the moduli, most calculations can be carried out in single precision (although it involves redundancy).

Chapter 4

P-adic systems

4.1 Introduction

Usually, digital computers work with numbers of finite length in a p -ary scale, where p is a positive integer called *radix*. In such a system, many of the rational numbers do not have exact representations, because either they may not terminate within the length of the number of digits available for calculations, or they may be periodic. Therefore, any attempt to use a computer to do arithmetic in the field of real numbers fails.

Of course, there is always the possibility of approximating these "non representable" rationals. But in this case, we face the difficulty of accumulated round-off and truncation errors. Unfortunately, we cannot avoid such errors because even with simple operations, the number of digits of the results may increase so that they cannot be held in the registers available in the computer; this will force us to discard a number of digits of our number. These errors originate new errors, and so on.

Numerical problems can be classified as *well-conditioned* and *ill-conditioned*. In the first case, although small round-off or truncation errors occurred throughout the computations, we can still obtain reliable results.

In the other case, even small errors in computation may get amplified and propagate in such a way that the final result lies far away from the actual solution, and unfortunately, many "real-world" problems lie in this category.

Such ill-conditioned problems frequently occur in matrix computations (when the determinant is small), such as solving a system of linear equations, finding the inverse of a matrix, reduction of a matrix to a specified canonical form, the determination of the characteristic equation, etc

Residue arithmetic can prevent the problem of round-off errors for an exact solution. However, because of the disadvantages which we mentioned in the previous chapter, (i.e. the difficulties involved in division and the computational problems involved if we choose several prime moduli in the case where the numbers lie in a wide range), this method is not so attractive.

Hence, there is a need of an alternate number system having the advantages of both the p -ary and residue number systems. Such a number system, which is similar to the residue number system (*modulo* p^r , where p is prime and r the number of digits) and has all these attractive features, is the p -adic arithmetic system (and more precisely, the segmented p -adic system) which was first introduced by Hensel in 1900.

Some of its attractive features are the following ([14]):

- 1) The conversion of integers and other rationals into 2-adic representation is simple and realizable by existing hardware.
- 2) The evaluation of multiplicative inverses is of many orders less in complexity ([14]).
- 3) The fast high-precision multiplication due to Toom-Cook, which we mentioned in the previous chapter and other high speed switching schemes can be used to speed up the 2-adic arithmetic.
- 4) The conversion from *modulo* 2^r to the positional representation is easy.
- 5) The choice of r can be made in regular integral steps to meet the requirements of a specific matrix problem. The hardware design also becomes simple.

Every rational number other than integers or radix fractions has an infinite recurring p -adic expansion and, unlike the p -ary representations, any truncation of this expansion would yield a disastrous error; this is one reason that the p -adic number systems have partly been despised.

In the following sections of this chapter we shall show how one can handle the p -adic expansion of rationals for exact computation.

We first describe the *finite-segment* p -adic representation of numbers, but in later sections we shall give a new representation of p -adic numbers removing at the same time the restriction of fixed finite length.

4.2 Background

In this section we shall give a few definitions, properties and examples which will help us in the study of the following sections.

4.2.1 P -adic numbers.

First we define the p -adic numbers and state some of their properties.

Consider the congruence

$$x^2 \equiv 3 \pmod{11^n} \quad (1)$$

For $n = 1$, we get $x^2 \equiv 3 \pmod{11}$, which has two solutions

$$x_0 \equiv \pm 6 \pmod{11} \quad (2)$$

For $n = 2$, we get

$$x^2 \equiv 3 \pmod{11^2} \quad (3)$$

which means that 11^2 divides $x^2 - 3$; therefore 11 divides $x^2 - 3$, which implies that $x^2 \equiv 3 \pmod{11}$, and hence any solution of (3) must be of the form $x_0 + 11t_1$, where x_0 is as in (2).

Without loss of generality, we choose $x_0 \equiv 6 \pmod{11}$. In order for $x_1 = 6 + 11t$ to be a solution for (3), it must satisfy the following :

$$(6 + 11t)^2 \equiv 3 \pmod{11^2},$$

$$\text{or } 36 + 2 \cdot 6 \cdot 11t_1 + 11^2 t_1^2 \equiv 3 \pmod{11^2}$$

$$\text{or } 33 + 2 \cdot 6 \cdot 11t_1 + 11^2 t_1^2 \equiv 0 \pmod{11^2}$$

$$\text{or } 3 + 2 \cdot 6t_1 \equiv 0 \pmod{11}$$

$$\text{i.e. } t_1 \equiv 8 \pmod{11}.$$

Hence, $x_1 \equiv 6 + 11 \cdot 8 \pmod{11^2} \equiv 94 \pmod{11^2}$. Similarly, for $n = 3$ we get $x^2 \equiv 3 \pmod{11^3}$, which will have a solution of the form $x_2 = x_1 + 11^2 t_2$; i.e.

$$(x_1 + 11^2 t_2)^2 \equiv 3 \pmod{11^3}$$

$$\text{or } 94^2 + 2 \cdot 94 \cdot 11^2 t_2 + 11^4 t_2^2 \equiv 3 \pmod{11^3}$$

$$\text{or } 8833 + 2 \cdot 11^2 \cdot 94 t_2 + 11^4 t_2^2 \equiv 0 \pmod{11^3}$$

$$\text{or } 73 + 2 \cdot 94 t_2 \equiv 0 \pmod{11};$$

$$\text{i.e. } t_2 \equiv 4 \pmod{11}$$

This means that $x_2 \equiv 94 + 11^2 \cdot 4 \pmod{11^3}$, i.e. $x_2 \equiv 578 \pmod{11^3}$. If we continue this way, we obtain a sequence

$$x_0, x_1, x_2, \dots, x_n, \dots \quad (4)$$

which satisfies the conditions :

$$x_0 \equiv 6 \pmod{11} \quad (a)$$

$$x_n \equiv x_{n-1} \pmod{11^n} \quad (b)$$

$$x_n^2 \equiv 3 \pmod{11^{n+1}} \quad (c) \quad (5)$$

Indeed : $94 \equiv 6 \pmod{11}$, $578 \equiv 94 \pmod{11^2}$, and so on. Also, from (5c) we have that 11^{n+1} divides $x_n^2 - 3$.

Definition : We say that two integers are *p-close*, for some prime *p*, when their difference is divisible by a sufficiently large power of *p*.

We may observe that following the process given above, the squares of the numbers of the sequence (4) may become arbitrarily 11-close to 3 as *n* increases.

Definition 1([3]) Let p be some prime number. A sequence of integers

$$\{x_n\} = \{x_0, x_1, x_2, \dots, x_n, \dots\}$$

which satisfies

$$x_n \equiv x_{n-1} \pmod{p^n} \quad ; \quad n = 1, 2, \dots \quad (6)$$

determine an object α , called a *p-adic integer*; we write : $\{x_n\} \rightarrow \alpha$.

An ordinary integer is determined by the sequence $\{x, x, \dots, x, \dots\}$. Note that two distinct ordinary integers x and y , correspond to distinct *p-adic integers* $\{x\}$ and $\{y\}$; since if $\{x\} = \{y\}$, then $x \equiv y \pmod{p^n}$ for all n , which implies that $x = y$.

Let x and y be two *p-adic integers* determined by the sequences $\{x_n\}$ and $\{y_n\}$ respectively. Then,

$$x + y = \{x_n\} + \{y_n\} = \{x_n + y_n\},$$

$$x \cdot y = \{x_n\} \cdot \{y_n\} = \{x_n \cdot y_n\}.$$

Under these operations, the set of *p-adic integers* becomes a commutative ring.

Divisibility of *p-adic integers* is defined as in any commutative ring; x divides y if there is a *p-adic integer* z , such that $y = x \cdot z$. In order to investigate divisibility, we must know which *p-adic integers* have a multiplicative inverse. We shall call these *p-adic integers* *p-adic units*.

Now, we shall introduce the *p-adic metric*. (see also [2])

It has been proved in [3] that a *p-adic integer* x , which is determined by the sequence $\{x_0, x_1, \dots, x_n, \dots\}$, is a unit if and only if $x_0 \not\equiv 0 \pmod{p}$. If x is a nonzero *p-adic integer*, then it has a unique representation of the form

$$x = p^m \cdot \epsilon \quad (7),$$

where ϵ is a unit of the ring of *p-adic integers* (proved in [3]).

Definition : The number m in (7) is called the *p-adic value* or *p-value* of x , and will be denoted by $v_p(x)$.

Definition : A fraction of the form x/p^k , where x is a p -adic integer, and $k \geq 0$, determines a *fractional p-adic number*. Two fractions x/p^k and y/p^m , determine the same p -adic number if and only if $xp^m = yp^k$ in the ring of p -adic integers.

Addition and multiplication between two p -adic numbers, is defined as follows :

$$\frac{x}{p^k} + \frac{y}{p^m} = \frac{x p^m + y p^k}{p^{m+k}}$$

and

$$\frac{x}{p^k} \cdot \frac{y}{p^m} = \frac{x \cdot y}{p^{k+m}}$$

Under these operations, the set of p -adic numbers is turned into a field. Suppose now that $x = p^m \epsilon$ and $y = p^k \eta$, where ϵ and η are units in the ring of p -adic integers, and x, y nonzero. Then, $x \cdot y = p^{m+k} \cdot (\epsilon \cdot \eta)$, i.e. $v_p(x \cdot y) = m + k = v_p(x) + v_p(y)$.

Definition : The sequence $\{x_n\} = \{x_0, x_1, \dots, x_n, \dots\}$ of p -adic numbers *converges* to the p -adic number x (i.e. $\lim_{n \rightarrow \infty} x_n = x$) if $\lim_{n \rightarrow \infty} v_p(x_n - x) = \infty$ (i.e. arbitrarily p -close)

Now let c be a real number, where $0 < c < 1$. We set

$$|x|_p = \begin{cases} c^{v_p(x)} & \text{for } x \neq 0 \\ 0 & \text{for } x = 0 \end{cases} \quad (8)$$

Definition : The function $|x|_p$, x p -adic number, defined as in (8), is called *p-adic metric*.¹ The number $|x|_p$ is called *p-adic size* of x .

This p -adic metric has the following properties : (proved in [2])

$$|x \cdot y|_p = c^{m+k} = |x|_p \cdot |y|_p$$

¹ Note that in terms of the p -adic metric, the definition of convergence in the field of p -adic numbers takes the following form : The sequence $\{x_n\}$, (x_n p -adic number) converges to the p -adic number x if $\lim_{n \rightarrow \infty} |x_n - x|_p = 0$.

$$|x + y|_p \leq \max(|x|_p, |y|_p) \quad (9)$$

It has also been proved (in [2]) the following

Theorem : Any p -adic number x can be written in the form

$$x = \sum_{i=-n}^{\infty} x_i p^i, \quad (10)$$

where each $x_i \in Z$, and n is such that $|x|_p = |p|_p^n$.

The integer coefficients x_i of (10) are unique *modulo* p . If in addition the x_i 's are such that $0 \leq x_i \leq p-1$, then we shall call (10) the *canonical representation or expansion* of x .

4.2.2 Farey fractions.

Definition : A *Farey sequence* of order n is the ascending sequence of reduced fractions between 0 and 1, whose denominators do not exceed n .

Therefore, a fraction h/k belongs to the Farey sequence F_n if $0 \leq h \leq k \leq n$, and $\gcd(h, k) = 1$.

Example : For $n = 7$, the Farey sequence F_7 is .

$$\begin{array}{cccccccccccccccccccc} 0 & 1 & 1 & 1 & 1 & 2 & 1 & 2 & 3 & 1 & 4 & 3 & 2 & 5 & 3 & 4 & 5 & 6 & 1 \\ \hline 1 & 7 & 6 & 5 & 4 & 7 & 3 & 5 & 7 & 2 & 7 & 5 & 3 & 7 & 4 & 5 & 6 & 7 & 1 \end{array}$$

Some of the properties of Farey sequences are the following : ([8])

Theorem 1: If h/k and h'/k' are two successive terms of F_n , then $kh' - hk' = 1$.

Theorem 2: If h/k , h''/k'' and h'/k' are three successive terms of F_n , then $\frac{h''}{k''} = \frac{h + h'}{k + k'}$.

Theorem 3: If h/k and h'/k' are two successive terms of F_n , then $k + k' > n$. The *mediant* $\frac{h + h'}{k + k'}$ of h/k and h'/k' falls in the interval $(\frac{h}{k}, \frac{h'}{k'})$.

Theorem 4: If $n > 1$, then no two successive terms of F_n have the same denominator.

Definition : An *order- n Farey fraction* is defined as a fraction $\pm h/k$, for which neither $h \geq 0$ nor $k > 0$ is greater than n .

According to the definition of order n Farey fractions, we may observe that the set of order n Farey fractions contains the Farey sequence of order n as a proper subset. For the number of terms in the Farey sequence of order n , the following theorem holds ([8]).

Theorem 5: The number of terms in the Farey sequence of order n is approximately $3n^2/\pi^2$

4.2.3 Continued fractions.

In this section we shall give definitions and state some basic results of continued fractions. A detailed introduction can be found in [10] or [8].

Definition : An expression of the form

$$a_0 + \frac{1}{a_1 + \frac{1}{a_2 + \frac{1}{\ddots}}} \quad (11)$$

is called *regular* or *simple continued fraction*. For our purposes, we assume $a_1, a_2, \dots, a_n, \dots$ to be positive integers, while a_0 may be an arbitrary real number. These numbers, are called *the elements* or *the partial quotients* of the continued fraction (11).

If the number of elements of the continued fraction is finite, then we can write the continued fraction in the form

$$a_0 + \frac{1}{a_1 + \frac{1}{a_2 + \frac{1}{\ddots a_{n-1} + \frac{1}{a_n}}}} \quad (12)$$

and we shall call it a *finite continued fraction*.

A simpler way to denote continued fractions is the following :

- a) infinite continued fraction : $[a_0; a_1, a_2, \dots]$
- b) finite continued fraction : $[a_0; a_1, a_2, \dots, a_n]$ (13)

Definition : The continued fraction $s_k = [a_0, a_1, a_2, \dots, a_k]$, where $0 \leq k \leq n$, is called a *segment* of the continued fraction (13b). For $k \geq 0$, we shall call s_k a segment of (13a). We shall also call the continued fraction $r_k = [a_k; a_{k+1}, \dots, a_n]$ a *remainder* of the continued fraction (13b), and the continued fraction $r_k = [a_k; a_{k+1}, \dots]$ a remainder of (13a).

Since any continued fraction $[a_0; a_1, \dots, a_n]$ is the result of a finite number of rational operations on its elements, it is a rational function of these elements; hence, it can be represented as the ratio of two polynomials in $a_0, a_1, \dots, a_n$ with integral coefficients. For specific values of these elements, the given continued fraction is represented as an ordinary fraction (p/q).

Definition : The *canonical representation* of a finite continued fraction, is a definite representation of it which we construct as follows :

For a 0-th order fraction, set $[a_0] = a_0$.

If we set $[a_0; a_1, \dots, a_n] = p/q$ and $r_1 = [a_1; a_2, \dots, a_n] = p'/q'$, then the numerators and denominators of their canonical representations must satisfy the following expressions :

$$p = a_0 p' + q' \quad \text{and} \quad q = p' \quad (14)$$

Definition : We shall call *k-th order convergent (or approximation)* of the continued fraction $[a_0; a_1, a_2, \dots]$ the canonical representation of the segment $s_k = [a_0; a_1, \dots, a_k]$, and we shall denote it by p_k/q_k .

The rule for finding the convergents, is given by the following

Theorem 6: For arbitrary $k \geq 2$,

$$\begin{aligned} p_k &= a_k p_{k-1} + p_{k-2} & (a) \\ q_k &= a_k q_{k-1} + q_{k-2} & (b) \end{aligned} \quad (15)$$

which can be easily proved by induction, using also the formulas (14).

Theorem 7: For all $k \geq 0$,

$$q_k p_{k-1} - p_k q_{k-1} = (-1)^k \quad (16)$$

Proof. From (15a), we get

$$p_k q_{k-1} = a_k p_{k-1} q_{k-1} + p_{k-2} q_{k-1} \quad (17)$$

and from (15b):

$$q_k p_{k-1} = a_k q_{k-1} p_{k-1} + q_{k-2} p_{k-1} \quad (18)$$

$$(18) - (17) \quad q_k p_{k-1} - p_k q_{k-1} = -(q_{k-1} p_{k-2} - p_{k-1} q_{k-2}) \quad (19)$$

But $q_0 p_{-1} - p_0 q_{-1} = 1$,² and using also (19), we conclude the validity of (16)

By dividing (16) by $q_k q_{k-1}$ and assuming that $k \geq 1$, in order $q_k q_{k-1}$ to be nonzero, we have the following

Corollary 1: For $k \geq 1$,

$$\frac{p_{k-1}}{q_{k-1}} - \frac{p_k}{q_k} = \frac{(-1)^k}{q_k q_{k-1}} \quad (20)$$

Now, we multiply (15a) by q_{k-2} and (15b) by p_{k-2} . If we subtract (15a) from (15b) and make use of Theorem 7, we obtain the following

Theorem 8: For all $k \geq 1$,

$$q_k p_{k-2} - p_k q_{k-2} = (-1)^{k-1} a_k \quad (21)$$

And from this, we get the following

Corollary 2: For all $k \geq 2$,

$$\frac{p_{k-2}}{q_{k-2}} - \frac{p_k}{q_k} = \frac{(-1)^{k-1} a_k}{q_k q_{k-2}} \quad (22)$$

Using the results given above, we may conclude the following important

Theorem 9: The even-order convergents of a continued fraction form an increasing sequence, while the odd-order convergents form a decreasing sequence. Also, every odd-order convergent is greater than any even-order convergent.

² When we need a convergent of order -1 , we set $p_{-1} = 1$ and $q_{-1} = 0$. Then, (15) still holds for $k = 1$. We also know that $q_0 = 1$ since the 0-th order convergent is $a_0/1$.

Using the definition of the remainder r_k of a continued fraction $[a_0, a_1, \dots, a_n]$, we may write $[a_0, a_1, a_2, \dots, a_n] = [a_0, a_1, \dots, a_{k-1}, r_k]$. But the continued fraction $[a_0, a_1, \dots, a_{k-1}, r_k]$ is equal to its k -th order convergent p_k/q_k ; and since it has a $(k-1)$ -st order convergent p_{k-1}/q_{k-1} , using Theorem 8, we conclude that :

Theorem 10: For arbitrary k , $1 \leq k \leq n$,

$$[a_0, a_1, \dots, a_n] = \frac{p_{k-1}r_k + p_{k-2}}{q_{k-1}r_k + q_{k-2}} \quad (23)$$

It has also been proved (in [10]) that the following theorem holds :

Theorem 11: For arbitrary $k \geq 1$,

$$\frac{q_k}{q_{k-1}} = [a_k, a_{k-1}, \dots, a_1] \quad (24)$$

Next, we state some results for infinite continued fractions.³

Suppose that we have the infinite continued fraction

$$[a_0; a_1, a_2, \dots] \quad (25)$$

which has an infinite sequence of convergents, say

$$\frac{p_0}{q_0}, \frac{p_1}{q_1}, \dots, \frac{p_k}{q_k}, \dots \quad (26)$$

consisting of real numbers.

Definition : If the sequence of the convergents (26) of the continued fraction (25) converges to a real number a , we say that the *infinite fraction itself converges*; the real number a is called *the value* of (25), and we write $a = [a_0; a_1, \dots]$.

Theorem 12: If the infinite continued fraction $[a_0; a_1, \dots]$ converges, then so do all its remainders; conversely, if at least one of the remainders of this continued fraction converges, then the continued fraction itself converges.

³ For constructive proofs refer to [10].

Some of the consequences of Theorem 9, are the following properties for convergent infinite continued fractions ([10]):

Theorem 13: The value of a convergent infinite continued fraction is greater than any of its even-order convergents, and less than any of its odd-order convergents.

Theorem 14: The value of the convergent continued fraction satisfies the inequality

$$\left| a - \frac{p_k}{q_k} \right| < \frac{1}{q_k q_{k+1}} \quad (27)$$

Theorem 15: For the continued fraction $[a_0; a_1, a_2, \dots]$ to converge, it is necessary and sufficient that the series

$$\sum_{n=1}^{\infty} a_n \quad (28)$$

diverges. And finally,

Theorem 16: For all $k \geq 0$,

$$\left| a - \frac{p_k}{q_k} \right| > \frac{1}{q_k(q_{k+1} + q_k)} \quad (29)$$

Our next step is to approximate an irrational number a by a rational fraction, which is one of the convergents of the infinite continued fraction representing a . Using the results of the Theorems 14 and 16, we get that

$$\frac{1}{q_n(q_n + q_{n+1})} < \left| a - \frac{p_n}{q_n} \right| \leq \frac{1}{q_n q_{n+1}} \quad (30)$$

Definition : We shall call a rational fraction (a/b) ($b > 0$) a *best approximation* of a real number α , if every other rational fraction which has the same or smaller denominator, differs from α by a greater amount; i.e. the inequalities $0 < d \leq b$ and $a/b \neq c/d$ imply that

$$\left| \alpha - \frac{c}{d} \right| > \left| \alpha - \frac{a}{b} \right| \quad (31)$$

Definition : The fractions

$$\frac{p_{k-2}}{q_{k-2}}, \frac{p_{k-2} + p_{k-1}}{q_{k-2} + q_{k-1}}, \dots, \frac{p_{k-2} + a_k p_{k-1}}{q_{k-2} + a_k q_{k-1}}, \frac{p_k}{q_k}$$

form, for even k , an increasing and, for odd k , a decreasing sequence. The first and the last terms are either both even or both odd order convergents. The intervening terms (if any), we shall call *intermediate fractions*.

Theorem 17: Every best approximation of a number α is a convergent or an intermediate fraction of the continued fraction representing that number. We shall call these approximations, *the best approximations of the first kind*.

Definition : We shall call the rational fraction a/b , ($b > 0$) a *best approximation of the second kind* of a number α , if the inequalities $c/d \neq a/b$ and $0 < d \leq b$ imply

$$|d\alpha - c| > |b\alpha - a| \quad (32)$$

Clearly, every best approximation of the second kind is a best approximation of the first kind (since by multiplying (31) by $d \leq b$, we obtain (32)); while the converse is not true. Hence, we may conclude the following

Theorem 18: Every best approximation of the second kind is a convergent.

In order to prove this theorem, we use the remarks previously mentioned, and Theorem 17. We shall conclude this section with the following important

Theorem 19: Every irreducible rational fraction a/b that satisfies the inequality

$$|\alpha - \frac{a}{b}| < \frac{1}{2b^2}$$

is a convergent of the number α .

Proof : According to theorem 18, every best approximation of the second kind is convergent; hence, it is sufficient to show that a/b is a best approximation of the second kind of the number α .

Suppose that $|d\alpha - c| \leq |b\alpha - a| < \frac{1}{2b}$, where $d > 0$ and $\frac{c}{d} \neq \frac{a}{b}$.

Dividing $|d\alpha - c| < \frac{1}{2b}$ by d , we get $|\alpha - \frac{c}{d}| < \frac{1}{2bd}$. Therefore,

$$|\frac{c}{d} - \frac{a}{b}| = |(\alpha - \frac{c}{d}) - (\alpha - \frac{a}{b})| \leq |\alpha - \frac{c}{d}| + |\alpha - \frac{a}{b}| < \frac{1}{2bd} + \frac{1}{2b^2} = \frac{b+d}{2b^2d} \quad (33)$$

But since $c/d \neq a/b$, one of them will lie between the other one and the next order convergent. Therefore,

$$\left| \frac{c}{d} - \frac{a}{b} \right| \geq \frac{1}{bd} \quad (34)$$

From (33),(34) we get that $\frac{1}{bd} < \frac{b+d}{2b^2d} \implies 2b < b+d \implies b < d$, and therefore the fraction a/b is a best approximation of the second kind of the number α , and according to the Theorem 18, it is also a convergent of α .

4.3 P-adic Representation.

It is true that we can write any rational number x in the form $x = \frac{a}{b} \cdot p^n$, where n, a, b , are positive or negative integers ($b \neq 0$) and $\gcd(a, b) = 1$. We can also write x in the form

$$x = \sum_{j=-n}^{\infty} z_j p^j \quad (35)$$

where the coefficients z_j are integers satisfying $0 \leq z_i \leq (p-1)$.

We may now observe that the sequence $p, p^2, p^3, \dots, p^n, \dots$ converges to zero with respect to the p -adic metric. More specifically, $|p^n|_p = c^n$ (by the definition of the p -adic metric given in section 4.2.1), for some c such that $0 < c < 1$; hence, $c^n \rightarrow 0$ as $n \rightarrow \infty$.

But this fact implies that $1, p, p^2, \dots, p^n, \dots$ is a decreasing geometric sequence, and therefore the infinite series $1 + p + p^2 + \dots + p^n + \dots$ converges to $1/(1-p)$ with respect to the p -adic metric, and the infinite series (35) to x (with respect to the p -adic metric always).

For convenience, we shall denote the expansion (35) by the sequence

$$\begin{aligned} a_n a_{n+1} \dots a_{-1} \dots a_0 a_1 \dots & \text{ for } n \text{ negative} \\ \bullet a_0 a_1 \dots & \text{ for } n = 0 \\ \bullet 00 \dots a_n a_{n+1} \dots & \text{ for } n \text{ positive} \end{aligned} \quad (36)$$

Example : Let $p = 7$ and $x = 2/5$. Then, $x = (2/5) \cdot 7^0$.

The p -adic expansion of $2/5$ is : $2/5 = \bullet 6214521452145\dots = 6 + 2p(1 + p^4 + p^8 + \dots) + p^2(1 + p^4 + p^8 + \dots) + 4p^3(1 + p^4 + p^8 + \dots) + 5p^4(1 + p^4 + p^8 + \dots)$.

The infinite series $1 + p^4 + p^8 + \dots$, according to the previous discussion, converges to $\frac{1}{1 - p^4}$ with respect to the p -adic metric.

Therefore, the series $6 + 2p(1 + p^4 + p^8 + \dots) + p^2(1 + p^4 + \dots) + 4p^3(1 + p^4 + \dots) + 5p^4(1 + p^4 + \dots)$ converges to

$$6 + \frac{2 \cdot 7}{1 - 7^4} + \frac{7^2}{1 - 7^4} + \frac{4 \cdot 7^3}{1 - 7^4} + \frac{5 \cdot 7^4}{1 - 7^4} = \frac{2}{5} \quad \text{for } p = 7$$

We may observe that the representation of integers and radix fractions are identical to the p -ary ones, except that they are written in the ascending powers of p from left to right (instead from right to left). However, as we shall see in later sections, this does not hold for all rationals.

Another interesting fact is that the negative rationals occur as the true complement (again, from left to right) of the corresponding positive number. For example,

$$-\frac{2}{5} = \bullet 145214521452\dots$$

Next, we shall give an algorithm for the conversion of a rational number to the p -adic form.

Conversion algorithm :

Input : A rational number $x = a/b$, where $\gcd(a, b) = 1$, $b \neq 0$, (p prime)

Output : The p -adic expansion of x .

begin

$\beta := x$ Find n such that $x = \frac{c}{d}p^n$, with $\gcd(d, p) = 1$

If $n > 0$ then set $x_0, x_1, \dots, x_{n-1} := 0$

repeat

Solve for a_n the congruence $d \cdot a_n \equiv 1 \pmod{p}$


```

    set  $x_n = c \cdot a_n \pmod{p}$ 
     $\gamma := \beta - x_n p^n$ 
    if  $\gamma \neq 0$  then [ $\beta := \gamma$  and find  $n$  such that  $\beta = \frac{c}{d} p^n$ ]
until  $\gamma = 0$ 
for  $i > n$  set  $x_i := 0$ 
end;

```

As we would expect, this algorithm is nonterminating since it is known that the convergence to a rational which is neither an integer nor a radix fraction, is guaranteed only for an infinite number of digits.

The following example will illustrate the Conversion Algorithm.

Example :

Suppose that $x = 2/5$ and $p = 7$. Following the algorithm, we get :

$$\begin{aligned}
 \beta &:= \frac{2}{5} \\
 x &= \frac{2}{5} 7^0 \text{ and therefore } n = 0 \\
 5a_0 &\equiv 1 \pmod{7} \implies a_0 \equiv 3 \pmod{7} \\
 x_0 &:= 2 \cdot 3 \pmod{7} \implies x_0 \equiv 6 \pmod{7} \\
 \gamma &:= \frac{2}{5} - 6 \cdot 7^0 = -\frac{28}{5} \\
 \gamma &\neq 0; \text{ Hence, } \beta := -\frac{28}{5} = -\frac{4}{5} \cdot 7^1 \text{ i.e. } n = 1 \\
 5a_1 &\equiv 1 \pmod{7} \implies a_1 \equiv 3 \pmod{7} \\
 x_1 &:= -4 \cdot 3 \pmod{7} \equiv -12 \pmod{7} \implies x_1 \equiv 2 \pmod{7} \\
 \gamma &:= -\frac{28}{5} - 2 \cdot 7 = -\frac{98}{5} = -\frac{2}{5} \cdot 7^2 \neq 0 \\
 \beta &:= -\frac{2}{5} 7^2 \text{ i.e. } n = 2 \\
 5a_2 &\equiv 1 \pmod{7} \implies a_2 \equiv 3 \pmod{7} \\
 x_2 &:= -2 \cdot 3 \equiv -6 \pmod{7} \implies x_2 \equiv 1 \pmod{7}
 \end{aligned}$$

$$\gamma := -\frac{98}{5} - 7^2 = -\frac{343}{5}$$

$$\gamma \neq 0; \text{ Hence, } \beta := -\frac{343}{5} = -\frac{1}{5}7^3; \text{ i.e. } n = 3$$

$$5a_3 \equiv 1 \pmod{7} \implies a_3 \equiv 3 \pmod{7}$$

$$x_3 \equiv -1 \cdot 3 \pmod{7} \implies x_3 \equiv 4 \pmod{7}$$

$$\gamma := -\frac{343}{5} - 4 \cdot 7^3 = -\frac{7203}{5}$$

$$\gamma \neq 0; \beta := -\frac{3}{5}7^4; \text{ i.e. } n = 4$$

$$5a_4 \equiv 1 \pmod{7} \implies a_4 \equiv 3 \pmod{7}$$

$$x_4 \equiv -3 \cdot 3 \pmod{7} \implies x_4 \equiv 5 \pmod{7}$$

$$\gamma := -\frac{28}{5}7^4$$

and so on. At this stage, we may observe that at the next step, $\beta = -\frac{4}{5}7^5$ i.e. c and d are now repeated. Hence, a repeating part has been recognized.

Therefore, the p -adic expansion (for $p = 7$) of $2/5$ is :

$$\frac{2}{5} = .621452145...$$

4.4 Hensel Codes.

Having introduced the basic properties of p -adic numbers (in section 4.2.1), we shall now make use of the p -adic expansion of a rational number to construct a code for the representation of the rationals, which can be used in a very efficient p -adic arithmetic system.⁴

The p -adic representation as introduced in the previous section, is obviously of no practical use since the convergence to the actual value is guaranteed only for an infinite number of terms and any truncation of it would result in a large number which has no resemblance to the actual value.

Therefore, if a p -adic system is to be used, one needs a finite expansion to work with; in

⁴ following the notation given in [16].

particular, a finite-length code derived from the p -adic expansion of the given rational

As mentioned in [16], the following three conditions have to be satisfied for the code to be unique and weighted :

- i) For a given N , any N -order Farey fraction has a unique code.
- ii) All the rationals contained in the given range can be represented using r digits in a base p .
- iii) The digit positions in the code have some definite weights which enable us to convert from the code to the rational representation.

The conditions for the construction of the so called *Hensel code* (in honour of K.W.S.Hensel), are the following :

- i) The rational numbers to be represented must have their numerator and denominator bounded by a prescribed integer N .
- ii) The p -adic expansions must terminate at the right such that the number of digits, say r , is even.

The resulting code will be denoted by $H(p, r)$, where p is the prime base, and r is the number of digits used (as in (ii) above).

The Hensel codes have the following properties : ([16])

- i) For a and b integers and $0 \leq a \leq N, 0 < b \leq N$ such that $\gcd(a, b) = 1$ and N is a positive integer such that $N \leq p^{r/2}/\sqrt{2}$, the rational number a/b can be uniquely represented as an r -digit ordered sequence, where each digit ranges from 0 to $p - 1$, where p is the prime base.⁵
- ii) The $H(p, r)$ code satisfies certain weight properties; in particular, as we shall see later, the digit positions have the following fractional weights :

$$p^0, p^1, p^2, \dots, p^{r/2-1}, \frac{-p^{r/2}}{p^{r/2}-1}, \frac{-p^{r/2+1}}{p^{r/2}-1}, \dots, \frac{-p^{r-1}}{p^{r/2}-1}.$$

⁵ theorem in [7].

Hence, we can easily convert them into rationals, and we can also observe that due to the fact that the weights are positive as well as negative, the positive and negative rational numbers can be represented without explicit sign.

Both positive and negative integers occupy the positions $0, 1, \dots, r/2 - 1$; however, in the first case these digits will be followed by a sequence of 0's, while in the second case by a sequence of digits equal to $p - 1$.

Therefore, the positive integers will be of the form

$$\bullet z_0 z_1 z_2 \dots z_{r/2-1} 00 \dots 0,$$

where the digits corresponding to positions from $r/2$ to $r - 1$ are all equal to zero, while the negative integers will have the form

$$\bullet z_0 z_1 z_2 \dots z_{r/2-1} (p - 1)(p - 1) \dots (p - 1),$$

i.e. they are in the complement form, and the digits occupying the positions from $r/2$ to $r - 1$ are all equal to $p - 1$.

Examples :

For $p = 7$ and $r = 8$,

$\bullet 43250000$ is the positive integer 1838

$\bullet 00360000$ is the positive integer 2205

$\bullet 00406666$ is the negative integer -2205

$\bullet 33416666$ is the negative integer -1838

By allowing the p -adic point to move to the right up to the $(r/2 - 1)$ -th position, we may also represent all radix fractions whose denominators do not exceed N . For example, (for $p = 7$ and $r = 8$)

$\curvearrowright 4 \bullet 3250000$ is the radix fraction 1838/7

$432 \bullet 50000$ is the radix fraction $-1838/343$

$33 \bullet 416666$ is the radix fraction $-1838/49$

The other rational numbers, which may occupy all the r digits, are classified into two types

a) those which have denominators which divide $p^{r/2} - 1$ and whose weighted sum gives rise to their actual value, we shall call them *soft fractions*, and

b) the ones which have a unique weighted sum having no resemblance to their actual value (in fact, the weighted sum and the actual value satisfy a congruence relation as we shall see in the next paragraphs), we shall call these fractions *hard fractions*.

Both soft and hard fractions will be of the form

$$x_{-n}x_{-n+1} \dots x_{-1} \bullet x_0x_1 \dots x_m,$$

where $n + m + 1 = r$ and $n \leq r/2 - 1$, while the digits may take any value from 0 to $(p - 1)$

Examples :

Let $p = 7$ and $r = 4$. As we shall see in later sections, the $H(7, 4)$ expansion of $5/6$ is $\bullet 2111$, which gives a weight equal to $40/48 = 5/6$, and therefore the fraction $5/6$ is a soft fraction. Note that 6 divides $7^2 - 1$. The fraction $3/5$ has an $H(7, 4)$ expansion equal to $\bullet 2452$, which gives a *pseudo-weight* equal to $509/48 \neq 3/5$; hence $3/5$ is a hard fraction. Note that 5 does not divide $7^2 - 1$.

Returning to the properties of the Hensel codes, we have that :

- iii) The negative rational numbers have a valid radix complement representation and multiplication as well as division can be carried out directly without corrections that are required in positive p -ary complement number representations.⁶
- iv) All basic arithmetic operations in $H(p, r)$ codes proceed from left to right, and the resulting length remains constant as long as the resulting rational number is as in (i).

⁶ see [14] and the references stated there.

- v) It is known that addition (or subtraction) in rational arithmetic requires three multiplications, one addition, and a reduction of the fraction to the minimal form by dividing both the numerator and the denominator by their greatest common divisor, while multiplication and division require two multiplications and, of course, the reduction of the resulting fraction to the minimal form.

The arithmetic using $H(p, r)$ codes is much simpler than rational arithmetic, and due to the fact that it uses a finite representation for all rationals it can be performed with a constant length word similar to the residue arithmetic; however, the division is now an exact operation as in rational arithmetic, and is similar to multiplication.

- vi) Unlike the p -ary number system (where truncation of the p -ary code does not affect the actual value of the number if and only if the number is an integer or a radix fraction), in the $H(p, r)$ code every order N Farey fraction is exactly representable.
- vii) The conversion from a rational to its $H(p, r)$ code is straightforward, as we shall see in section 4.6.1, for a proper choice of p and r .

The conversion algorithm from $H(p, r)$ codes to rationals for integers, radix and soft fractions (regardless of sign) can be found by computing their weighted sum. For the case of hard fractions, the conversion can be done through a solution of a congruence or a diophantine equation or by direct read out ([16]).

- viii) As a consequence of (v), the Hensel codes can be used for exact numerical computations.
- ix) As pointed out in [16], an $H(p, r)$ code with r being odd does not possess unique weights for digit positions, and therefore it cannot be easily converted to rationals by numerical computations.

We now return to examine whether the choice of $N \leq p^{r/2}/\sqrt{2}$ permits us to represent all the N -order Farey fractions.

According to Theorem 5 in section 4.2.2, the number of order N Farey fractions in the interval $[0, 1]$ is approximately equal to $3N^2/\pi^2 \approx N^2/3$, and if we take in account their reciprocals as well, we have a total of $2N^2/3$ positive rationals in this range, hence approximately $4N^2/3$ rationals to be represented if we also include their negatives.

But if we construct the $H(p, r)$ code according to (i)-(ii) mentioned previously in the present section, (including the p -adic point), we may represent more than p^r different sequences. Hence, for $N \leq p^{r/2}/\sqrt{2}$, the inequality $4N^2/3 \leq p^r$ will always be satisfied.

The way we constructed the code was that the positive integers would occupy the first $r/2$ positions followed by $r/2$ digits all equal to zero, and the negative integers would also occupy the first $r/2$ digits, but now followed by $r/2$ digits all equal to $(p-1)$, while all the other negative rationals occur in complement form, and should have a weight opposite in sign to that of the corresponding positive rationals.

For these reasons, the weights which correspond to the digit positions from 0 to $r-1$, are the following : ([16])

$$p^0, p^1, p^2, \dots, p^{r/2-1}, \frac{-p^{r/2}}{p^{r/2}-1}, \frac{-p^{r/2+1}}{p^{r/2}-1}, \dots, \frac{-p^{r-1}}{p^{r/2}-1} \quad (37)$$

as we have already mentioned earlier in this section.

Since any N -order Farey fraction has a unique $H(p, r)$ code, it will also have a unique weighted sum.⁷

One may observe that because of the weight assignment (37), every weighted sum will have the denominator $p^{r/2}-1$, while for integers, the numerator will be an integral multiple of $p^{r/2}-1$.

For the soft fractions (i.e. for the rationals a/b , where b divides $p^{r/2}-1$) the weighted sum will always be equal to their actual value ([16]); i.e. we shall have the equation

$$\frac{a}{b} = \frac{W}{p^{r/2}-1} \quad \text{or} \quad a \cdot (p^{r/2}-1) - b \cdot W = 0 \quad (38)$$

⁷ see also [16].

The hard fractions will satisfy (38) only (modulo p^r); i.e.

$$a(p^{r/2} - 1) - b \cdot W \equiv 0 \pmod{p^r} \quad (39)$$

and we shall call their weights, which are not equal to their actual value, *pseudo-weights*.

The following examples will illustrate the above discussion.

Examples :

Let $p = 7$ and $r = 4$.

- i) The expansion of $5/6$ is $\bullet 2111111\dots$. Hence, the $H(7, 4)$ code of $5/6$ is $\bullet 2111$. According to (37), the weight of $5/6$ is

$$2 + 7 - \frac{7^2}{7^2 - 1} - \frac{7^3}{7^2 - 1} = \frac{40}{48},$$

which is equal to the actual value of $5/6$. Note that 6 divides $7^2 - 1 = 48$; i.e. $5/6$ is a soft fraction.

- ii) The expansion of $3/5$ is $\bullet 245214521\dots$. The $H(7, 4)$ code of $3/5$ is $\bullet 2452$. According to (37), the weight of $3/5$ is

$$2 + 4 \cdot 7 - \frac{5 \cdot 7^2}{7^2 - 1} - \frac{2 \cdot 7^3}{7^2 - 1} = \frac{509}{48},$$

which, of course, is not equal to the actual value of $3/5$.

In order to find the actual value of the rational which corresponds to the pseudo-weight $509/48$, we have to solve for

$$48a - 509b \equiv 0 \pmod{7^4},$$

which has a solution for the right hand side equal to -2401 , i.e. -7^4 , for $a = 3$ and $b = 5$; this means that $3/5$ is the rational we are looking for. Note that 5 does not divide $7^2 - 1$; i.e. $3/5$ is a hard fraction.

- iii) The expansion of $-3/5$ is $\bullet 521452145\dots$. The $H(7, 4)$ code of $-3/5$ is $\bullet 5214$. According to

(37), the weight of $-3/5$ is

$$5 + 2 \cdot 7 - \frac{1 \cdot 7^2}{7^2 - 1} - \frac{4 \cdot 7^3}{7^2 - 1} = -\frac{509}{48},$$

which is not equal to the actual value of $-3/5$. Therefore, we have to solve for

$$48a + 509b \equiv 0 \pmod{7^4},$$

which has a solution for the right hand side equal to 2401, i.e. 7^4 , for $a = -3$ and $b = 5$;

hence, the rational is $-3/5$.

From the above examples and the preceding discussion, we can see that the $H(p, r)$ codes have features which are common to both the p -ary as well as the residue number systems. More precisely, operations with integers and soft fractions are like in the p -ary system, while operations with hard fractions are similar to the residue system. We may determine the sign and the actual value of the hard fractions by solving (39).

4.4.1 Arithmetic operations using Hensel codes.

In this section we shall state the basic algorithms for manipulating the Hensel codes. We shall see that unlike any weighted system, all operations proceed from left to right, including the division operation. The latter, is deterministic for all possible divisors, and this fact will enable us to carry it out using a fast algorithm unlike in the p -ary system.

I. Addition.

Suppose we are given the numbers x and y , where

$$x = x_{-m}x_{-m+1} \dots x_{-1} \cdot x_0x_1 \dots x_n$$

and

$$y = y_{-m}y_{-m+1} \dots y_{-1} \cdot y_0y_1 \dots y_n,$$

where $m + n + 1 = r$. Without loss of generality, we assume that both numbers have the same number of digits

Quite similarly to the p -ary number system, at each step we add the digits x_i and y_i and a carry c_i , and we generate a sum digit s_i and a new carry c_{i+1} . We assume that $c_{-m} = 0$, and we ignore the carry c_{n+1} . Of course, all the operations are carried out *modulo* p , where p is the prime base. I.e

- Set $c_{-m} = 0$

- For $i = -m, -m+1, \dots, n$

$$\text{set } s_i = (x_i + y_i + c_i) \bmod p$$

$$\text{and } c_{i+1} = (x_i + y_i + c_i) \div p$$

- Ignore c_{n+1}

Example : Let $p = 7$ and $r = 4$

i) $x = 3/5, y = 5/6$.

Then, $x = 3/5 = \bullet 2452$ and $y = 5/6 = \bullet 2111$. Hence,

$$\begin{array}{r} \bullet 2452 \\ \bullet 2111 \quad + \\ \hline \bullet 4563 \end{array}$$

i.e. $x + y = 43/30$, which can be obtained from $\bullet 4563$ by solving (39).

ii) $x = 3/5$ and $y = 2/5$.

Then, $x = 3/5 = \bullet 2452$ and $y = 2/5 = \bullet 6214$

i	0	1	2	3
x_i	2	4	5	2
y_i	6	2	1	4
c_i	0	1	1	1
s_i	1	0	0	0

i.e. $x + y = 3/5 + 2/5 = \bullet 1000 = 1$.

II. Subtraction.

Although subtraction can be done quite similarly to that in p -ary system by *borrowing* from the higher digit position, we shall handle it as complemented addition. I.e. given two numbers x and y , we shall form $z = x - y$ by taking the true complement of y , say $\bar{y}$ (starting from the left), and then adding it to x

The complementation algorithm consists of the following rules : ([16])

Rule 1 :

If $y_i \neq 0$ for $i = -m, -m+1, \dots, n$

then $\bar{y}_i := p - y_i$ for $i = -m$

and $\bar{y}_i := (p-1) - y_i$ for $-m+1 \leq i \leq n$

Rule 2 :

If $y_i = 0$ for $-m \leq i \leq j$,

then $\bar{y}_i := 0$ for $-m \leq i \leq j$

and $\bar{y}_{j+1} := p - y_{j+1}$

$\bar{y}_i := (p-1) - y_i$ for $i = j+2 \leq i \leq n$

Example : Let $p = 7$ and $r = 4$.

If $x = 3/5 = \bullet 2452$ then $\bar{x} = -3/5 = \bullet 5214$

If $x = 14 = \bullet 0200$ then $\bar{x} = -14 = \bullet 0586$.

III. Multiplication.

Let x be the multiplicand and y the multiplier. I.e.

$$x = x_{-m}x_{-m+1}\dots x_{-1} \bullet x_0x_1\dots x_n$$

$$y = y_{-m}y_{-m+1}\dots y_{-1} \bullet y_0y_1\dots y_n$$

The multiplication process is similar to the one in p -ary system except that the product will have the same length as the multiplier and the multiplicand.

We first form $P_{ij} = y_i x_j$ for $i = -m, \dots, n$ and $j = -m, \dots, n-i$; then we form the partial products P_i , and then the final product of x and y , say P , as follows : ([16])

for $i = -m$ to n

do[for $j = -m$ to $n-i$

do[$P_{ij} := y_i x_j$]]

$$P_i := \sum_{j=-m}^n P_{ij} \Delta(m+j)$$

$$x \cdot y = P = \sum_{i=-m}^n P_i \Delta(m+i),$$

where $\Delta(a)$ denotes right shift by a digits.

We place the p -adic point according to the following rule : • Replace the index k ($k = -m, -m+1, \dots, n$) of each digit of P by $(k-m)$, where m is the number of digits in the multiplier and multiplicand to the left of the p -adic point. As usual, the p -adic point lies between the indices -1 and 0 . The following example will illustrate the whole process.

Example : Let $p = 7$, $r = 4$, and $s \leq 3/4 = \bullet 6151$, $y = 2/5 = \bullet 6214$.

Table 1

i, j	0	1	2	3
x_j	6	1	5	1
y_i	6	2	1	4
P_{00}	1	5		
P_{01}		6	0	
P_{02}			2	4
P_{03}				6
P_0	1	4	3	3
P_{10}		5	1	
P_{11}			2	0
P_{12}				3
P_1		5	3	3
P_{20}			6	0
P_{21}				1
P_2			6	1
P_{30}				3
P_3				3
P	1	2	6	4

The final product is obtained by adding $P_0, P_1, P_2, P_3 \pmod{7}$.

Hence, we found that $P = x \cdot y = 3/4 \cdot 2/5 = \bullet 1264$. In order to find the rational which corresponds to $\bullet 1264$, we solve equation (39). The weight of $\bullet 1264$ is $-\frac{946}{48}$. Hence, $48a + 964b \equiv 0 \pmod{7^4}$, which has a solution for the right hand side equal to 9604 for $a = 3$ and $b = 10$.

Therefore, $P = x \cdot y = 3/4 \cdot 2/5 = \bullet 1264 = 3/10$.

IV. Division.

The division operation is similar to multiplication in p -adic arithmetic, except that we now choose the multiplicative inverse of the leading digit of the divisor.

We have already mentioned in previous sections that unlike the p -ary system, the division process does not require the trial and error process, it is deterministic, and proceeds from the left to the right.

Let $x = \bullet x_0 x_1 \dots x_n$ be the dividend, $y = \bullet y_0 y_1 \dots y_n$ be the divisor, and $q = \bullet q_0 q_1 \dots q_n$ be the quotient. We assume that y_0 is nonzero, otherwise we shift the divisor to the left keeping count until the first digit is nonzero, and we suitably adjust the quotient. Of course, we have to ensure that by shifting the divisor, it does not go outside the range of the order N Farey fractions for the given $H(p, r)$, because if this is the case, then the division operation is not valid.

The division process is as follows : ([16])

If R_i denotes the i -th partial remainder, $\bar{y}$ the complement of y , and $\Delta(i)$ the right shift by i digits, then :

- Set $R_0 := x$
- $q_i := R_i y_0^{-1} \pmod{p}$ for $i = 0, 1, \dots, n$ (R_i : the i -th digit of R_i)
- $R_{i+1} := R_i + q_i \bar{y} \Delta(i)$

R_{i+1} is the next partial remainder (after R_i).

The algorithm terminates when q_n is obtained.

The following example will illustrate the division process.

Example : Let $p = 7$ and $r = 4$.

Take $x = 3/4$ which has an $H(7, 4)$ code equal to $\bullet 6151$ and $y = 3/5$ which has an $H(7, 4)$ code equal to $\bullet 2452$.

Then, $\bar{y} = \bullet 5214$ and $y_0^{-1} \equiv 4 \pmod{7}$.

Hence, set $R_0 = \bullet 8151$.

Also, $q_0 = R_{00}y_0^{-1} = 8 \cdot 4 \equiv 3 \pmod{7}$, $q_0 \cdot \bar{y} = \bullet 1145$

$q_0 \bar{y} \Delta(0) = \bullet 1145$ and therefore $R_1 = R_0 + q_0 \bar{y} \Delta(0) = \bullet 0320$, i.e. $R_1 = \bullet 0320$.

The whole process of the division of $3/4$ by $3/5$ is shown in the table below :

Table 2

i	0	1	2	3
R_0	8	1	5	1
$q_0 \bar{y} \Delta(0)$	1	1	4	5
R_1	0	3	2	0
$q_1 \bar{y} \Delta(1)$	0	4	6	6
R_2	0	0	2	0
$q_2 \bar{y} \Delta(2)$	0	0	5	2
R_3	0	0	0	3
$q_3 \bar{y} \Delta(3)$	0	0	0	4
R_4	0	0	0	0
q	3	5	1	5

I.e. $q = \bullet 3515$. The weight of q is $60/48$ and by solving for (39), we find that $a = 5$ and $b = 4$;

i.e. q is equal to its actual value.

Hence, $q = 3/4 \div 3/5 = \bullet 3515 = 5/4$.

4.5 A new representation.

We shall now follow the approach of Hensel's p -adic arithmetic to construct a finite representation for rational numbers, which is implementable in computer hardware and like any good

scheme for representing numbers, will have the following properties :

- i) All rationals have a finite representation, but unlike the Hensel codes, the length of the new representation is variable.
- ii) The representation is compact, and as pointed out in [9] (where this representation was first proposed), it requires less space (on average) than the other fixed-length schemes which are used in computers. The compactness can be achieved by giving to the frequently occurring numbers short encodings while giving longer encodings to the less frequent numbers.
- iii) The addition, subtraction and multiplication algorithms are similar to the ones in p -ary systems, while division is as easy as multiplication and proceeds in the same direction as the other operations

Although our representation is of variable length, this does not mean that we have to perform operations serially by digit. Instead, we could retrieve d digits at a time, and if d is large compared to the average length of the operands, this kind of retrieval results in the speed of a fixed-length design and the ability to handle operands that are not representable in a fixed-length design.

We have already examined how one can obtain the p -adic expansion of any rational number. For example, we saw that the p -adic expansion of $3/4$ (for $p = 7$) is given by

$$\frac{3}{4} = \bullet 6151515\dots$$

or that the expansion of $-2/5$ is given by

$$-\frac{2}{5} = \bullet 6214521452145\dots$$

We shall now prove ([2]) the following important

Theorem : A p -adic number x is rational if and only if its canonical expansion

$$\sum_{j=-n}^{\infty} x_j p^j, \quad 0 \leq x_j \leq p-1,$$

where n is such that $|x|_p = |p|_p^n$, is periodic

Proof: Suppose that the canonical expansion of x is periodic, i.e.

$$\begin{aligned} x = x_n p^n + x_{n+1} p^{n+1} + \dots + x_{n+k} p^{n+k} + y_1 p^{n+k+1} + \dots + y_j p^{n+k+j} + \\ + y_1 p^{n+k+j+1} + \dots + y_j p^{n+k+2j} + \\ + \dots \end{aligned}$$

in which case,

$$\begin{aligned} x &= p^n (x_n + x_{n+1} p + \dots + x_{n+k} p^k) + p^{n+k+1} (y_1 + y_2 p + \dots + y_j p^{j-1}) + \\ &\quad + p^{n+k+j+1} (y_1 + y_2 p + \dots + y_j p^{j-1}) + \\ &\quad + \dots \\ &= p^n X + p^{n+k+1} Y (1 + p^j + p^{2j} + \dots), \end{aligned}$$

where $X = x_n + x_{n+1} p + \dots + x_{n+k} p^k$, and $Y = y_1 + y_2 p + \dots + y_j p^{j-1}$

But as we have already mentioned in section 4.3, the infinite series $1 + p^j + p^{2j} + \dots$ converges to $1/(1 - p^j)$ with respect to the p -adic metric.

Hence, $x = p^n X + p^{n+k+1} Y \frac{1}{1 - p^j}$, which is obviously a rational number.

Conversely, we assume that x is a rational number.

Let $|x|_p = |p|_p^n$; so $x/p^n = \beta$, where $|\beta|_p = 1$. Therefore, we may write $\beta = c/d$, where $c, d \in \mathbb{Z}$ and p does not divide d and c .

Since p is prime and does not divide d , we conclude that $\gcd(p, d) = 1$ and by Euler's theorem

$$p^{\phi(d)} \equiv 1 \pmod{d},$$

where ϕ is the Euler's function. ($\phi(n)$ denotes the number of positive integers less than or equal to n and relatively prime to n).

Hence, there exists j such that $p^j \equiv 1 \pmod{d}$; i.e. d divides $p^j - 1$.

We may write β in the following form:

$$\beta = \frac{c(p^j - 1)}{d(p^j - 1)}$$

and since d divides $p^j - 1$, we have that

$$z = p^n \beta = p^n \frac{e}{p^j - 1},$$

where e is an integer. Next, we choose an integer k , such that

$$0 \leq e < p^{k+1} \quad \text{if } z \geq 0; \quad -p^{k+1} \leq e < 0 \quad \text{if } z < 0.$$

Since $\gcd(p^{k+1}, p^j - 1) = 1$, we can solve uniquely (mod $p^j - 1$) the congruence

$$p^{k+1} \alpha \equiv -e \pmod{p^j - 1}$$

Let Y be a solution, where $0 \leq Y \leq p^j - 2$ if $z \geq 0$, and $1 \leq Y \leq p^j - 1$ if $z < 0$. In this case,

$$p^{k+1} Y \equiv -e \pmod{p^j - 1}$$

and therefore

$$e = X(p^j - 1) - Yp^{k+1},$$

where X is an integer, and also $0 \leq X < p^{k+1}$ (it can be checked easily).

Finally,

$$z = p^n \beta = p^n \frac{e}{p^j - 1} = p^n \frac{[X(p^j - 1) - Yp^{k+1}]}{p^j - 1},$$

$$0 \leq X < p^{k+1}, \quad 0 \leq Y < p^j.$$

But we may also write X and Y in the following forms :

$$X = x_n + x_{n+1}p + \dots + x_{n+k}p^k, \quad 0 \leq x_i < p$$

$$Y = y_1 + y_2p + \dots + y_jp^{j-1}, \quad 0 \leq y_i < p$$

and therefore z can be written in the form

$$\begin{aligned} z &= p^n X + p^{n+k+1} Y \frac{1}{1 - p^j} \\ &= p^n (x_n + \dots + x_{n+k}p^k) + p^{n+k+1} (y_1 + y_2p + \dots + y_jp^{j-1}) (1 + p^j + p^{2j} + \dots) \end{aligned}$$

which implies that the canonical expansion of x is periodic (according to the first part of the proof)

This completes the proof of the theorem

Since we restrict ourselves to rationals, it is logical to construct a representation which will enable us to recognize the repeated part of the expansion of a given rational

For convenience reasons, we shall place the digits of the p -adic expansion as follows

$$x_{n+m}x_{n+m-1} \dots x_{n+1} \cdot x_n x_{n-1} \dots x_2 x_1 x_0 \quad (40)$$

I.e. we shall now represent $3/4$ as $51'6$, and $-2/5$ will now be represented as $5412'6$, where 51 is the *repeating part* of the p -adic expansion of $3/4$, and similarly 5412 is the repeating part of the p -adic expansion of $-2/5$

Definition : We shall call the digits to the left of the quote *the negative part*, and the digits to the right of the quote, *the positive part* of the expansion

The number represented by (40), is equal to

$$\sum_{i=0}^n x_i p^i - \sum_{i=n+1}^{n+m} \frac{x_i p^i}{(p^m - 1)} \quad , \quad p : \text{prime base} \quad (41)$$

Note the similarity of (41) and the weight assignment in Hensel codes. The only difference is that m is not necessarily $r/2$ (r is the number of digits used), but it may vary as well as n

To justify the formula (41), we proceed as follows:

The positive part is obviously equal to $\sum_{i=0}^n x_i p^i$, i.e.

$$x_n x_{n-1} \dots x_1 x_0 = \sum_{i=0}^n x_i p^i \quad (42)$$

as in an ordinary p -ary system.

The negative part can be obtained as follows :

$$\begin{aligned} x_{n+m} x_{n+m-1} \dots x_{n+1} &= x_{n+m} x_{n+m-1} \dots x_{n+1} \cdot x_{n+m} \dots x_{n+1} \\ &= x_{n+m} x_{n+m-1} \dots x_{n+1} \cdot 00 \dots 0 + x_{n+m} x_{n+m-1} \dots x_{n+1} \\ &= x_{n+m} x_{n+m-1} \dots x_{n+1} \cdot p^m + \sum_{i=n+1}^{n+m} x_i p^{i-n-1} \end{aligned}$$

Therefore, $(p^m - 1) \cdot x_{n+m} x_{n+m-1} \dots x_{n+1} = - \sum_{i=n+1}^{n+m} x_i p^{i-n-1}$

$$x_{n+m} x_{n+m-1} \dots x_{n+1} = - \sum_{i=n+1}^{n+m} \frac{x_i p^{i-n-1}}{(p^m - 1)} \quad (43)$$

But

$$x_{n+m} x_{n+m-1} \dots x_{n+1} \cdot x_n x_{n-1} \dots x_0 = x_{n+m} x_{n+m-1} \dots x_{n+1} \cdot p^{n+1} + x_n x_{n-1} \dots x_0 \quad (44)$$

From (42), (43) and (44), we can easily obtain (41)

After verifying formula (41), we observe that using it, the sign determination becomes trivial, by just comparing the leading digits of the positive and negative parts (if both are present). We assume that the representation is normalized, i.e. as short as possible, in which case $x_n \neq x_{n+m}$.

Hence, if $x_{n+m} < x_n$, the number is positive,

if $x_{n+m} > x_n$, the number is negative,

if the number has only one part, the sign is given by the present part;

if both parts are absent, the number is zero.

Another property of formula (41), is that we may use it to convert from our representation back to rationals. For example, $51'6 = 6 - (7 + 5 \cdot 7^2)/(7^2 - 1) = 3/4$.

We can also use formula (41) to derive an easy method for converting to a right-repeating expansion, by simply subtracting the negative part from the positive with their leading digits aligned, and the negative part repeated indefinitely to the right. ([9])

For example, in decimal,⁸ $45'482 = 482 - 454.\overline{54} = 27.\overline{45}$

or $454'82 = 82 - 45.\overline{445} = 36.\overline{554}$.

Conversely, $36 - 554' = 454'82$.

⁸ p does not have to be prime in this case.

To perform the operations suggested above, there exist two comparison algorithms. The first is to subtract the comparands and then determine the sign of the result, while the second is to convert the comparands to right repeating form and then perform the usual digit by digit comparison. As pointed out in [9], the first has the advantage that it is a right-to-left algorithm, but the second is more efficient.

4.5.1 Arithmetic operations

In this section we shall examine how one can manipulate numbers, using the new representation, and we shall point out the similarities (and differences) to the ordinary algorithms.

I. Addition, Subtraction and Multiplication.

Addition and subtraction processes are similar to the ordinary p -ary systems, except that instead of ~~borrowing~~ one from a minuend digit some distance away (in the case of subtraction operation), we simply carry one to the immediately neighboring digit of the subtrahend. For example, if the base $p = 7$, we have that

$$\begin{array}{r} 3 \quad 2 \quad 0 \quad 0 \quad 1 \\ - \quad 3_1 \quad 4_1 \quad 1_1 \quad 5 \\ \hline 2 \quad 5 \quad 2 \quad 5 \quad 3 \end{array}$$

The obvious advantage of this technique, is that we affect only the immediately neighboring digit, and not a digit at some distance away.

Let us now examine the form of the negative numbers using the new representation.

Suppose that we want to construct the representation of -2 . An obvious way, is to subtract 2 from 0. I.e. (for $p = 7$)

$$\begin{array}{r} \dots \quad 0 \quad 0 \quad 0 \quad 0 \\ - \dots \quad 0_1 \quad 0_1 \quad 0_1 \quad 2 \\ \hline \dots \quad 6 \quad 6 \quad 6 \quad 5 \end{array}$$

Beginning at the right, the subtraction process subtracts 2 from 7 and carries one to the next digit

position, then, subtracts 1 from 7 and carries one to the next digit position (to the left), and so on, generating this way a sequence of 6s in front of 5

Hence, the representation of -2 (in base 7) is $6'5$, while the representation of 2 is $0'2$. Similarly, the representation of -11 in decimal,⁹ is $9'89$, while the representation of 11 is $0'11$

Therefore, we may observe that the rule for negation is the radix complement rule *complement each digit and then add 1*

However, in binary, the negation is even simpler *starting at the right, leave trailing 0s and rightmost 1 alone and flip the rest*

The definition of radix complement is made in terms of the computer word, for example, if e bits are available for encoding integers, then two's complement is defined by performing arithmetic modulo 2^e . Radix complement is preferred in computers because of its properties, and using our representation we may also take advantage of it without wasting space or having overflow problems as in fixed-length representation. ([9])

The multiplication process is the usual one; i.e., in decimal, we shall have that

$$\begin{array}{r} 9'89 \\ \times 5 \\ \hline 9'45 \end{array}$$

⁹ The previously mentioned operations can be carried out in any base p , not necessarily prime

or

$$\begin{array}{r}
 5 \\
 \times 9'89 \\
 \hline
 45 \\
 40 \\
 45 \\
 45 \\
 45 \\
 \hline
 99945
 \end{array}$$

I.e. in either case, we obtain the same result $-11 \times 5 = 9'89 \times 5 = 9'45 = -55$

II. Division.

Let us first test the consistency of our representation by means of an example

Suppose that $p = 7$. If we subtract 1 from 0, we get $0' - 1 = 6'$, i.e. the expansion of -1 . Then, we divide -1 by 6 to take $6' \div 6 = 1'$, the representation of $-1/6$, and now complement $1'$ to take $5'6$; i.e. the representation of $1/6$. Finally, we multiply $1/6$ by 6 to take $5'6 \times 6 = 0'1$ as we expected. Note that we may also obtain the same expansion for $1/6$ by subtracting $-1/6$ from 0; i.e. $0' - 1' = 5'6$.

We shall now present the division algorithm restricting ourselves (for the moment) to integer dividends and divisors, and further to divisors whose rightmost digit is nonzero and relatively prime to the base, although we shall remove these restrictions in the next paragraphs. For example, one way to ensure that the rightmost digit of the divisor is relatively prime to the base, is to choose the base to be a prime integer.

In order to illustrate the division process, we shall divide 23 by 15 when $p = 7$. At the first step we set 23 as the zeroth partial remainder, and we try to find a digit which when multiplied by the rightmost digit of the divisor gives a result whose rightmost digit is equal to the rightmost

digit of the dividend. This digit that we look for, will be the rightmost digit of the quotient.

In our case, we can see that 2 when multiplied by 5 gives 13, whose rightmost digit is 3. Therefore, $q_0 = 2$. Our restrictions ensure that there exists exactly one digit having this property.

Next, we multiply $15 \times 2 = 33$ and we subtract 33 from 23 to get 6'0. We discard the rightmost 0, set 6' as the first partial remainder, and repeat the same process to find q_1 and so on, until we find two partial remainders which are identical, in which case the repeating part of the quotient has been detected. I.e.

$$\begin{array}{r}
 0'23 \div 15 = 042 \\
 \underline{- 33} \\
 6'0 \\
 \underline{- 66} \\
 6'00 \\
 \underline{- 0} \\
 6'0
 \end{array}$$

We can see that the first and the third partial remainders are the same, and therefore a cycle has been detected. Placing the quote at the proper position, we get that $0'23 \div 15 = 04'2$. Indeed,

$$\begin{array}{r}
 04'2 \\
 \times 15 \\
 \hline
 \dots 262626303 \\
 \dots 40404042 + \\
 \hline
 \dots 000000023
 \end{array}$$

Like multiplication, the division algorithm produces each digit by table look-up.

We may also observe that there is no guess throughout the division process. However, if we try to divide 1 by 5 in decimal, we can see that there is no multiple of 5 which has its rightmost digit equal to 1. This is because 5 is not relatively prime to the base 10.

Hence, we need a way to represent those rationals whose denominator has factors which are factors of the base. This can be obtained by allowing a radix point, whose position is independent

of the quote (or an exponent) For example in decimal,

$$253'78 \div 10 = 253'7.8$$

$$253'7.8 \div 10 = 253'.78$$

$$253'.78 \div 10 = 25.3'78$$

$$25.3'78 \div 10 = 2.53'78$$

and so on.

To divide by integers in decimal, we have to *cast out* all 2 and 5 factors from the divisor ([9]). We can determine these factors, one at a time, by inspecting the divisor's final digit. We may cast out a 2 factor by multiplying both divisor and dividend by 5, while we cast out a 5 factor by multiplying both by 2.

In a similar way, we may generalize this method of casting out factors for a general nonprime base.

However, one will never face this kind of problem working in the binary system, where multiplication and division tables are trivial, as well. For division, at each step the rightmost bit of the

dividend remaining, is the next bit of the result and also specifies whether or not to subtract the divisor. For example,

$$\begin{array}{r}
 10'11 \div 11 = 0111001 \\
 \underline{- 11} \\
 1010'00 \\
 \underline{1010'0} \\
 1010' \\
 \underline{01'} \\
 - 11 \\
 \underline{10'010} \\
 - 11 \\
 \underline{10'0110} \\
 - 11 \\
 \underline{10'000} \\
 10'00
 \end{array}$$

i.e. $10'11 \div 11 = 011100'1$.

III. Radix Conversion Algorithm.

We already know that in order to convert a positive integer expressed in base b into another base p , we repeatedly divide x by p , until a quotient equal to zero is found. The sequence of remainders from the right to the left, gives the digits of the new representation. For example, suppose that we want to convert 23 to binary.

$$23 = 2 \cdot 11 + 1$$

$$11 = 2 \cdot 5 + 1$$

$$5 = 2 \cdot 2 + 1$$

$$2 = 2 \cdot 1 + 0$$

$$1 = 2 \cdot 0 + 1$$

Therefore, 23 in binary is 10111. Let us now follow the same process for the conversion of -23 in

OF / DE

binary.

$$-23 = 2 \cdot (-12) + 1$$

$$-12 = 2 \cdot (-6) + 0$$

$$-6 = 2 \cdot (-3) + 0$$

$$-3 = 2 \cdot (-2) + 1$$

$$-2 = 2 \cdot (-1) + 0$$

$$-1 = 2 \cdot (-1) + 1$$

$$-1 = 2 \cdot (-1) + 1$$

i.e. -23 in binary is $\dots 11101001$, and placing the quote accordingly, we take our finite representation $1'01001$ which is -as we expected- the true complement of $0'10111$, the representation of 23 which we obtained above.

Let us now try to convert a fraction, say $2/7$, to binary. Then,

$$2/7 = 2 \cdot 1/7 + 0$$

$$1/7 = 2 \cdot (-3/7) + 1$$

$$-3/7 = 2 \cdot (-5/7) + 1$$

$$-5/7 = 2 \cdot (-6/7) + 1$$

$$-6/7 = 2 \cdot (-3/7) + 0$$

$$-3/7 = 2 \cdot (-5/7) + 1$$

We can see that $-3/7$ has appeared before, and therefore a cycle has been detected. Hence, the representation of $2/7$ in binary is $011'10$.

However, another choice at step one, is to take $2/7 = 2 \cdot (-5/14) + 1$, i.e. the algorithm is not deterministic. In order to avoid this situation, and in order to produce a representation without radix point, the original fraction and each quotient must have denominators relatively prime to the base (i.e. the denominator must remain unchanged ([9])).

Let us apply now the radix conversion algorithm to a number in our representation; i.e. to convert $2/7 = 571428'6$ from base 10 to binary.

The algorithm proceeds in the same way, except that we now inspect the rightmost digit of each quotient and if it is odd we subtract 1 and then divide by 2 to find the next quotient, while

if it is even we subtract 0 and divide again by 2. For example,

$$\begin{aligned}(571428'6 - 0) \div 2 &= 285714'3 \\ (285714'3 - 1) \div 2 &= 428571' \\ (428571' - 1) \div 2 &= 714285' \\ (714285' - 1) \div 2 &= 857142' \\ (857142' - 0) \div 2 &= 428571'\end{aligned}$$

At this step, we find a quotient which has appeared before and we terminate the process. Again, $2/7 = 571428'6$ in binary is $011'10$.

We may observe that division by 2 is the same as multiplication by 5 (in decimal) and discarding the rightmost digit which is zero. In general, (as pointed out in [9]) when converting from base p to base b , a finite number of rightmost digits of each quotient is sufficient to determine the next digit of the result if all factors of b are factors of p .

4.5.3 Length of representation.

Since we are restricted to rationals, we know that the result of addition, subtraction, multiplication or division of two rational numbers is rational, and therefore it will have as representation an infinite but fortunately repeating sequence of digits.

However, as we have already mentioned, in order for the arithmetic algorithms to terminate, one must be able to detect when a cycle occurs.

One way to recognize a state which has occurred before, is to store the states and compare the new one with all previously stored states until a match is found; otherwise, we store and repeat. At the end we have to normalize the result. The disadvantage of this method is that it requires a lot of space.

Another method, which requires storing only one state, is the following: Suppose that we have the states $S_1, S_2, S_3, \dots$; then make the following comparisons until a match is found :

Of course, again, we have to normalize the result, because since we always compare the state S_i with the states $S_{i+1}, S_{i+2}, \dots, S_{2i}$, in most cases a repeating part of length say k , will not be detected before the storing of the S_k state, as we shall see in the next chapter.

4.5.3 Length bounds.

We now give an alternative method of implementing the arithmetic algorithms and discuss bounds on the length of the results.

First, the method :

- i) Calculate a bound for the number of digits of the result;
- ii) calculate a correct, though not necessarily minimal, number of digits of the repeating part;
- iii) produce a sufficient number of digits to ensure that a repetition of the length (number of

digits) calculated in (ii) has occurred;

iv) normalise the result.

Although this method obviates the need to detect a repeated state, the complexity of the formulae which give the length bounds makes it unattractive.

If b is the base of the representation,

p_i is the length of the positive part in the base b representation of x_i ,

n_i is the length of the negative part in the base b representation of x_i ,

ϕ is the Euler's function,¹⁰

Then the bounds are given in the following table : ([9])

Table 3

$x_3 = x_1 + x_2$	$p_3 \leq \text{Max}(p_1, p_2) + n_3 + 2,$
$x_3 = x_1 - x_2$	$n_3 \text{ is a divisor of } LCM(n_1, n_2)$
$x_3 = x_1 \cdot x_2$	$p_3 \leq p_1 + p_2 + n_3 + 1,$
	$n_3 \text{ is a divisor of } LCM(n_1, n_2)$
$x_3 = x_1 \div x_2$	$p_3 \leq \begin{cases} p_1 + p_2 + n_3 + 1, & \text{if } p_2 \leq n_2 \\ p_1 - p_2 + n_3 + 1, & \text{if } n_2 < p_2 \leq p_1 \\ n + 1, & \text{if } p_1, n_2 < p_2 \end{cases}$ $n_3 \text{ is a divisor of } LCM(n_1, \phi(x_2(b^{n_2} - 1)))$

4.5.4 Approximation.

The representation proposed in section 4.5 has the advantage that it provides exact results.

¹⁰ $\phi(m)$ is defined for positive integers m as the number of positive integers not exceeding m that are relatively prime to m .

However, repeatedly applying the arithmetic operators may result in extra cost due to the fact that the length of the representation of the results may tend to grow, which implies extra storage cost; and if they become greater than the arithmetic unit can accomodate at a time, then processing time increases.

Using the proposed representation, the accuracy may vary from none to complete depending on the user's needs, while in the case of fixed-length floating point representation the accuracy is fixed and predetermined (single or double).

The desired approximation can be denoted by the following two forms of the approximation operator proposed in [9] :

- i) Given a number x and a number of digits d , $x \approx_d$ denotes the number which is closest to x and whose representation has at most d digits.
- ii) Given a number x and a tolerance t , $x \approx_t$ denotes a number in the range $x \pm xt$ whose representation is shortest.

In either case finding a *closest* or *shortest* result is hard; however, in practice, we may obtain a reasonably close or short and acceptable result. One way (proposed in [9]) is to convert the number into right repeating form using the method we have already mentioned, and then truncate the result. I.e. we may convert $454'82$ to $36.\overline{554}$, and then approximate this with 36.55 or 36.6 etc.

We may observe that if the truncated result has n digits to the right of the radix point, then the truncation error is less than p^{-n} , where p is the base.

4.6 Conversion Algorithms.

We shall now discuss in more detail the procedures for conversion of p -adic representations (segmented or not) of rational numbers back to their original rational form, some of which we have already mentioned in previous sections.

i.e. find an $n \times n$ matrix A^{-1} , whose integer entries lie in the range $[0, p-1]$, and such that $AA^{-1} \equiv I \pmod{p}$

This step requires $O(n^3)$ arithmetic operations.

Second step • Compute the p -adic approximation $\tilde{z}$ to z ; this can be done as follows.

Consider the two sequences of integer column vectors $\{z_i\}$, $\{b_i\}$, where $z_i \in [0, p-1]$, defined by¹³

$$b_0 = b \quad (a)$$

$$x_i \equiv A^{-1}b_i \pmod{p} \quad (b) \quad (52)$$

$$b_{i+1} = p^{-1}(b_i - Ax_i), \quad i \geq 0 \quad (c)$$

One may now observe that $b_i - Ax_i \equiv pb_{i+1} \equiv A \cdot A^{-1}b_i - Ax_i \equiv A(A^{-1}b_i - x_i) \equiv 0 \pmod{p}$, because of (52b). Therefore, the entries of b_{i+1} are integers, and since all the entries of x_i lie in $[0, p-1]$, we may conclude that all the entries of b_i lie in $[-n\beta, n\beta]$ ¹⁴. Therefore, the entries of b_i are integers of length $O(\log n\beta) = O(\log n + \log \beta) = O(\log n)$. Hence, x_i and b_{i+1} can be computed from b_i in $O(n^2 \log n)$ arithmetic operations.

The construction of the above sequences terminates after computing x_{m-1}, b_m , where m is defined in the next paragraphs, and requires $O(mn^2 \log n)$ arithmetic operations (since, as we saw above, each step requires $O(n^2 \log n)$ arithmetic operations).

Now let $\tilde{z} = \sum_{i=0}^{m-1} p^i z_i$. Then,

$$\begin{aligned} A\tilde{z} &= \sum_{i=0}^{m-1} p^i Ax_i = \sum_{i=0}^{m-1} p^i (b_i - pb_{i+1}) \\ &= \sum_{i=0}^{m-1} (p^i b_i - p^{i+1} b_{i+1}) = b_0 - p^m b_m \\ &= b - p^m b_m, \end{aligned}$$

¹³ The Hensel lemma in matrix form

¹⁴ the proof is a simple induction

which is equivalent to the diophantine equation

$$a(p^{r/2} - 1) - bW = kp^r, \quad k \in \pm 1, \pm 2, \quad (45)$$

However, using Theorem 7 (section 4.2.3), we may observe that we may obtain a solution $a = a_0$, $b = b_0$ for the diophantine equation

$$ma - nb = 1 \quad (46)$$

if we simply expand the fraction m/n as a continued fraction and take the last but one convergent, which will be b_0/a_0 .

Note that if a_0, b_0 is a solution for (46), the same is true for

$$a_0 \pm sn, \quad b_0 \pm sm, \quad s = 0, 1, 2,$$

and therefore we have to solve it for other solutions and keep those which satisfy a given property. In our case, we have to solve (45) for different values of k and choose the solution which gives an order N Farey fraction.

One way for determining k , is the following:

Suppose that the p -adic number has a weight equal to $W/(p^{r/2} - 1)$ and its reciprocal a weight equal to $W/(p^{r/2} - 1)$. Then, $WW^r/(p^{r/2} - 1)^2$ will be the (pseudo-)weight of the product of the p -adic number times its reciprocal, i.e. of 1 (mod p), and since even in that case (45) has to be satisfied, we shall have that

$$1 \cdot (p^{r/2} - 1) - WW^r = kp^r,$$

therefore, we take

$$k = \frac{(p^{r/2} - 1)^2 - WW^r}{p^r} \quad (47)$$

Using the value of k obtained by (47), we now look for an appropriate value of s which yields the desired order N Farey fraction¹²

¹² A direct method to solve (45) has been proposed in [5], while other direct conversion techniques can be found in [17].

Example :

Suppose that we want to find the rational form of •2452 in $H(7, 4)$.

Suppose that a/b is the desired rational form of the given number. The pseudo-weight of •2452 is $509/48$, while the pseudo-weight of its reciprocal (•4222) is $80/48$

Using (47), we get that

$$k = \frac{48^2 - 509 \cdot 80}{2401} \Rightarrow k = -16$$

In order to solve the diophantine equation $509b' - 48a' = 1$, we expand $509/48$ as a continued fraction to obtain that

$$\frac{509}{48} = 10 + \cfrac{1}{1 + \cfrac{1}{1 + \cfrac{1}{1 + \cfrac{1}{1 + \cfrac{1}{9}}}}}$$

which has the following convergents :

$$10, 11, \frac{21}{2}, \frac{32}{3}, \frac{53}{5}, \frac{509}{48}$$

Therefore, $\frac{a'}{b'} = \frac{53}{5}$

In order to find a and b , we have to solve the diophantine equation (45), i.e.

$$48a - 509b = k \cdot 2401$$

or $509b - 48a = -k \cdot 2401$; i.e. $509b - 48a = 16 \cdot 2401$, which can be written in the form

$$509b - 48a = 16 \cdot 2400 + 16 \text{ or}$$

$$509b - 48(a - 16 \cdot 50) = 16 \quad (48)$$

But due to the fact that the equation $509b' - 48a' = 1$ has a solution $b' = 5$ and $a' = 53$, we conclude that (48) will have the following solution .

$$b = 16 \cdot b' \text{ and } a - 16 \cdot 50 = 16 \cdot 53.$$

From the hypothesis, we have that $u_i = v_i s - w_i h$ and so : $u_j = v_j s - w_j h$. But since w_j and v_j are relatively prime, we have that

$$|u_j| = |v_j s - w_j h| \leq |g s - t h| = |f| \leq \lambda h^{1/2}.$$

Hence, $|u_j| \leq |f| \leq \lambda h^{1/2}$ and since k is the least integer with this property, we conclude that $j \geq k$.

However, $u_j = v_j s - w_j h$ and $u_k = v_k s - w_k h$; hence, $u_j v_k - u_k v_j \equiv 0 \pmod{h}$.¹⁶

Since $j \geq k$, we get that

$$\begin{aligned} |u_j v_k - u_k v_j| &\leq |u_j v_j + u_k v_j| \leq (|u_j| + |u_k|) |v_j| \\ &< (\lambda h^{1/2} + h^{1/2}) \lambda = (\lambda^2 + \lambda) h \\ &= (\lambda^2 + \lambda - 1 + 1) h = h \end{aligned}$$

since by the hypothesis, $\lambda^2 + \lambda - 1 = 0$. Hence, $|u_j v_k - u_k v_j| < h$, and together with $u_j v_k - u_k v_j \equiv 0 \pmod{h}$, this implies that $u_j v_k - u_k v_j = 0$, i.e. $u_j/v_j = u_k/v_k$ and therefore $j = k$, which means that $f/g = u_k/v_k$, and this completes the proof of the theorem.

As far as the choice of m is concerned, (which, as we mentioned in the second step of the algorithm above, determines the last terms of the sequences $\{x_i\}$ and $\{b_i\}$), we have that $h = p^m$ and we can ensure that the hypotheses of the above theorem hold for the components of x provided that $\delta \leq \lambda p^{m/2}$.

Hence, we take $m = 2 \lceil \log(\delta \lambda^{-1}) / \log p \rceil$. Since $\delta \leq \beta^n n^{n/2}$, assuming that p has been fixed, we have that $m = O(\log(\beta^n n^{n/2} \lambda)) = O(n \log n)$.

The way for calculating the fraction f/g , which (as mentioned in [4]) has been proved efficient in practice, is by using a modification of the Euclidean algorithm. We assume that $0 \leq s \leq h$, and replace u_i by $|u_i|$ in order to ensure that the integer sequences $\{u_i\}$, $\{v_i\}$ and $\{q_i\}$ defined below, consist of positive integers which results in a simplification of the necessary multiprecision arithmetic.

¹⁶ because $u - j v_k - u_k v_j = v_j v_k s - w_j v_k h - v_k v_j s + w_k v_j h = w_k (v_j - v_k) h \equiv 0 \pmod{h}$.

From Hadamard's theorem ([11]), we have that

$$|\det A| \leq \prod_{i=1}^n \left(\sum_{j=1}^n a_{ij}^2 \right)^{1/2}, \quad \text{where } A = (a_{ij})_{1 \leq i, j \leq n} \quad (50)$$

However, since $\det A = \det A^T$, (50) can be written in the following form

$$|\det A| \leq \prod_{i=1}^n \left(\sum_{j=1}^n a_{ji}^2 \right)^{1/2}, \quad (50a)$$

and following the notation given in [4], if we let $\lambda_i = \left(\sum_{j=1}^n a_{ji}^2 \right)^{1/2}$, $i = 1, 2, \dots, n$, then (50a) becomes.

$$|\det A| \leq \prod_{i=1}^n \lambda_i, \quad (51)$$

Assuming that $\lambda_0 = \left(\sum_{i=1}^n b_i^2 \right)^{1/2}$, we define δ to be the product over the n largest of $\lambda_0, \lambda_1, \dots, \lambda_n$. Using Cramer's rule, we may observe that the rational entries in the solution x to $A \cdot x = b$ have their numerators and denominators bounded in absolute value by δ ; because the numerator of these entries, say of x_i , will be the $\det A_i$, where A_i is the matrix obtained from A by replacing the i -th column with b , and since δ is the product over the n largest of $\lambda_0, \lambda_1, \dots, \lambda_n$, the result follows from (51).

If, in addition, β is a bound on the absolute values of the entries of A and b , then

$$\lambda_i = \left(\sum_{j=1}^n a_{ji}^2 \right)^{1/2} \leq (n\beta^2)^{1/2} = n^{1/2}\beta,$$

(of course the same holds for λ_0 as well), and therefore $\delta \leq (n^{1/2}\beta)^n = n^{n/2}\beta^n$ (because of the definition of δ).

Now, let p be a prime such that $p \leq \beta$ and p does not divide $\det A$ (the latter, in order to ensure that $\det A \not\equiv 0 \pmod{p}$). We shall now give the algorithm and calculate the time taken, in terms of the number of arithmetic operations between integers of size up to β .

First step : • Find the inverse $(\text{mod } p)$ to A ;

i.e. find an $n \times n$ matrix A^{-1} , whose integer entries lie in the range $[0, p-1]$, and such that $A \cdot A^{-1} \equiv I \pmod{p}$.

This step requires $O(n^3)$ arithmetic operations.

Second step : • Compute the p -adic approximation $\tilde{x}$ to x ; this can be done as follows:

Consider the two sequences of integer column vectors $\{x_i\}$, $\{b_i\}$, where $x_i \in [0, p-1]$, defined by:¹³

$$b_0 = b \quad (a)$$

$$x_i \equiv A^{-1} b_i \pmod{p} \quad (b) \quad (52)$$

$$b_{i+1} = p^{-1}(b_i - Ax_i); \quad i \geq 0 \quad (c)$$

One may now observe that $b_i - Ax_i \equiv pb_{i+1} \equiv A \cdot A^{-1} b_i - Ax_i \equiv A(A^{-1} b_i - x_i) \equiv 0 \pmod{p}$, because of (52b). Therefore, the entries of b_{i+1} are integers, and since all the entries of x_i lie in $[0, p-1]$, we may conclude that all the entries of b_i lie in $[-n\beta, n\beta]$ ¹⁴. Therefore, the entries of b_i are integers of length $O(\log n\beta) = O(\log n + \log \beta) = O(\log n)$. Hence, x_i and b_{i+1} can be computed from b_i in $O(n^2 \log n)$ arithmetic operations.

The construction of the above sequences terminates after computing x_{m-1}, b_m , where m is defined in the next paragraphs, and requires $O(mn^2 \log n)$ arithmetic operations (since, as we saw above, each step requires $O(n^2 \log n)$ arithmetic operations).

Now let $\tilde{x} = \sum_{i=0}^{m-1} x_i p^i$. Then,

$$\begin{aligned} A\tilde{x} &= \sum_{i=0}^{m-1} p^i Ax_i = \sum_{i=0}^{m-1} p^i (b_i - pb_{i+1}) \\ &= \sum_{i=0}^{m-1} (p^i b_i - p^{i+1} b_{i+1}) = b_0 - p^m b_m \\ &= b - p^m b_m, \end{aligned}$$

¹³ The Hensel lemma in matrix form

¹⁴ the proof is a simple induction.

because of (52a) and (52c). Hence, we get that

$$Ax \equiv b \pmod{p^m}.$$

Final step : • Recover the rational solution x from the p -adic approximation $\mathbf{z}$; in other words : Give an efficient algorithm to compute the fraction f/g , with $|f|, |g| \leq \delta$, and such that $g \cdot s \equiv f \pmod{h}$, for given integers s and h . (In our case, $s = \mathbf{z}_i$ ¹⁵ and $h = p^m$). Note that since we work (modulo p), only a sufficiently large h will ensure a unique solution.

We shall obtain the desired result by means of continued fractions, (the theory of which we have mentioned in the *background* section of the present chapter), as given by the following theorem (stated in [4]):

Theorem : Let $s, h > 1$ be integers, and suppose that there exist integers f and g such that

$$gs \equiv f \pmod{h} \quad \text{and} \quad |f|, |g| \leq \lambda h^{1/2}, \quad (53)$$

where $\lambda = 0.618\dots$ is a root of $\lambda^2 + \lambda - 1 = 0$.

Let $w_i/v_i, i = 0, 1, 2, \dots$ be the convergents to the continued fraction for s/h , and put $u_i = v_i s - w_i h$. If k is the least integer such that $|u_k| < h^{1/2}$, then $f/g = u_k/v_k$.

Proof : As we have already mentioned in section 4.2.3 (theorem 6, theorem 7 and corollary 1) the sequences $\{w_i\}, \{v_i\}$ are increasing, while $\{u_i\}$ is alternating in sign and decreasing in absolute value. Now set $f = gs - th$. Then,

$$\frac{gs - th}{gh} = \frac{f}{gh} = \frac{fg}{g^2 h}, \quad \text{i.e.} \quad \left| \frac{s}{h} - \frac{t}{g} \right| = \left| \frac{fg}{g^2 h} \right| < \frac{1}{2g^2}$$

and according to theorem 19 (section 4.2.3), this implies that t/g is one convergent, say w_j/v_j , of s/h ; i.e. $t/g = w_j/v_j$.

¹⁵ $\mathbf{z}_i$: the i -th component of $\mathbf{z}$, for some i .

From the hypothesis, we have that $u_i = v_i s - w_i h$ and so : $u_j = v_j s - w_j h$. But since w_j and v_j are relatively prime, we have that

$$|u_j| = |v_j s - w_j h| \leq |g s - t h| = |f| \leq \lambda h^{1/2}.$$

Hence, $|u_j| \leq |f| \leq \lambda h^{1/2}$ and since k is the least integer with this property, we conclude that $j \geq k$.

However, $u_j = v_j s - w_j h$ and $u_k = v_k s - w_k h$; hence, $u_j v_k - u_k v_j \equiv 0 \pmod{h}$.¹⁶

Since $j \geq k$, we get that

$$\begin{aligned} |u_j v_k - u_k v_j| &\leq |u_j v_j + u_k v_j| \leq (|u_j| + |u_k|)|v_j| \\ &< (\lambda h^{1/2} + h^{1/2})\lambda = (\lambda^2 + \lambda)h \\ &= (\lambda^2 + \lambda - 1 + 1)h = h \end{aligned}$$

since by the hypothesis, $\lambda^2 + \lambda - 1 = 0$. Hence, $|u_j v_k - u_k v_j| < h$, and together with $u_j v_k - u_k v_j \equiv 0 \pmod{h}$, this implies that $u_j v_k - u_k v_j = 0$, i.e. $u_j/v_j = u_k/v_k$ and therefore $j = k$, which means that $f/g = u_k/v_k$, and this completes the proof of the theorem.

As far as the choice of m is concerned, (which, as we mentioned in the second step of the algorithm above, determines the last terms of the sequences $\{x_i\}$ and $\{b_i\}$), we have that $h = p^m$ and we can ensure that the hypotheses of the above theorem hold for the components of x provided that $\delta \leq \lambda p^{m/2}$.

Hence, we take $m = 2\lceil \log(\delta \lambda^{-1}) / \log p \rceil$. Since $\delta \leq \beta^n n^{n/2}$, assuming that p has been fixed, we have that $m = O(\log(\beta^n n^{n/2} \lambda)) = O(n \log n)$.

The way for calculating the fraction f/g , which (as mentioned in [4]) has been proved efficient in practice, is by using a modification of the Euclidean algorithm. We assume that $0 \leq s \leq h$, and replace u_i by $|u_i|$ in order to ensure that the integer sequences $\{u_i\}, \{v_i\}$ and $\{q_i\}$ defined below, consist of positive integers which results in a simplification of the necessary multiprecision arithmetic.

¹⁶ because $u - j v_k - u_k v_j = v_j v_k s - w_j v_k h - u_k v_j s + w_k v_j h = w_k(v_j - v_k)h \equiv 0 \pmod{h}$.

The sequences $\{u_i\}, \{v_i\}, \{q_i\}$, are generated in the following way .

$$u_{-1} = h, \quad u_0 = s, \quad v_{-1} = 0, \quad v_0 = 1$$

$$\text{and } q_i = [u_{i-1}/u_i], \quad u_{i+1} = u_{i-1} - q_i u_i, \quad v_{i+1} = v_{i-1} + q_i v_i; \quad i = 0, 1, \dots$$

until the first index k , where $u_k < h^{1/2}$, in which case we have that $f/g = (-1)^k u_k/v_k$ (because of the theorem above).

Next, we shall estimate the number of arithmetic operations between integers of size up to β required to compute the fraction f/g . ([4])

We may prove by induction, that $v_i \geq \prod_{j=0}^{i-1} q_j$ and $v_i \geq 2^{i/2}$ for $i \geq 0$. Since $(-1)^k u_k/v_k$ is equal to f/g in its lowest terms, we have that $|v_k| \leq |g| < h^{1/2}$.

Therefore, $\prod_{j=0}^{k-1} q_j \leq |v_k| < h^{1/2}$ implies that $\log(\prod_{j=0}^{k-1} q_j) \leq \log|v_k| < 1/2 \log h$ or the equivalent $\sum_{j=0}^{k-1} \log q_j = O(\log h)$; and since $2^{k/2} \leq v_k$, we get that $k/2 \leq \log v_k$, i.e. $k = O(\log h)$. Hence both $\sum_{j=0}^{k-1} \log q_j$ and k are $O(\log h)$.

The integers u_i, v_i are all bounded by h , and therefore the multiprecision computation of f/g will take at most $\sum O(\log q_j + 1) O(\log h) = O(\log h)^2$ arithmetic operations.

To summarize, $m = O(n \log n)$, each component of x can be recovered from the corresponding component of $\tilde{x}$ in $O(m)$ operations and therefore x can be recovered from $\tilde{x}$ in $O(nm^2) = O(n^3 \log^2 n)$ operations; i.e. the total computation of the exact solution to $A \cdot x = b$ takes $O(n^3 \log^2 n)$ operations.

However, as pointed out in [4], the $\log n$ factors above, all enter in the form $(\log n + \log b)$ and if $\log n$ is smaller than $\log b$ (as frequently happens in practice), we may assume this factor as a constant, and therefore the expected time taken will be (in most practical cases) proportional to n^3 , which is comparable to the time taken by a single precision floating-point solution using direct methods (it actually takes 10-20 times longer according to [4]).

There is a remote possibility that the $\det A$ is nonzero but divisible by p . We can recognize this fact in the computation of A^{-1} and a new prime p has to be selected.

Some further methods to speed up the calculations involved, are also discussed in [4].

However, the method proposed in this section, although superior in finding the exact solution of a nonsingular system $A \cdot x = b$ ($A: n \times n$ integer matrix) for large values of n , lacks two features of the congruence methods. Congruence methods permit

- i) simple computation of $\det A$, and
- ii) computation of A^{-1} with just a small amount of additional work. Neither of these are readily available with the above method.

4.8 Concluding Remarks.

It should be an omission for us to proceed to the next chapter, which concerns with the implementation of the new representation (which was proposed in section 4.5), without stating the reasons which make it superior compared to the segmented p -adic codes (Hensel codes).

Despite the many advantages described in section 4.4, there are certain difficulties associated with finite-segment p -adic arithmetic which have not been solved yet. To name a few,

- i) Simplification of conversion of hard fractions to rationals.
- ii) Detection of sign and magnitude of $H(p, r)$ codes directly without explicit conversion.
- iii) Detection of overflow when the rationals exceed the range under consideration.

These problems present no difficulty to our new representation. Furthermore, with the implementation given in the following chapter, it becomes even more attractive, especially in problems in which the accuracy may vary.

The new representation has been tested on integer data ([9]) using as sample a large, well known compiler (XCOM). According to this test, even when the overhead costs associated with the use of variable-length data items and the memory compactions necessary with data items dynamically changing in size, are taken into account, there is a 31% saving compared to a 32-bit

fixed-length encoding. Note that in this test the cost was measured as a space-time product, since space is being traded for time when using a variable-length encoding.

Hence, this method is significantly more economical than a well known standard method.

Chapter 5

Implementation

We shall now describe an implementation of the new representation (as introduced in section 4.5), which features most of the advantages of p -adic expansions, is of finite length, and lacks the difficulties inherent to segmented p -adic codes (i.e. overflow detection, hard fractions, etc.).

We shall first give a special structure to this representation and later we shall explain how the basic operations can be performed under this structure.

A computer listing can be found in the Appendix.

5.1 Using linked-list structure.

An attractive way for representing the p -adic expansion of a given rational number in a computer, is in terms of a linked list.

Suppose that we are given the following p -adic expansion

$$x_{n+m}x_{n+m-1}\dots x_{n+1} \overline{x_nx_{n-1}\dots x_1x_0} \quad (1)$$

In all arithmetic operations (including division), the digits of the operands are traversed from right to left. This suggests representing (1) by storing its digits from right to left in a linked list, so that the first node on the list contains the least significant (rightmost) digit.

However, as we have already mentioned, the digits to the left of the quote consist of the repeating part of (1). This can be denoted, in terms of linked lists, as a circular linked list with the successor of x_{n+m} being the digit x_{n+1} , where m is the number of digits of the repeating part.

Hence, the representation of (1), as a linked list, will be

where r points to the head of the list.

Note the presence of a node whose data is a quotation mark. Actually, this is a *dummy* node whose only use is to denote the beginning of the repeating part, and to help us to keep track of the number of times we have traversed the full repeating part. Its data will never be processed, and therefore they can be anything but a digit in our representation. For example, since all z_i 's lie in the range $[0, p-1]$, we may assume that the dummy node contains as data the number $-m-1$, where m is the length of the repeating part, or any other negative number.

In order to take full advantage of the linked-list structure, increasing at the same time the efficiency of our representation, the z_i 's do not have to be small integers; i.e. we choose p to be a large prime integer such that $p \leq \text{maxint}$, where maxint is the largest integer that a node can accommodate.

This way, our structure becomes more economical in terms of memory space.

Note that the degree of accuracy may vary, and therefore it can be user-defined; i.e. we may choose approximations that make a computation as cheap as possible, while achieving the desired accuracy (which can be from *none* to *complete*).

Examples :

The number $54/32$ will be:

The number $8/41$ will be:

The number 235' will be

5.2 Arithmetic operations.

Let us now examine how one may apply the structure (2) to perform the basic arithmetic operations (discussed in section 4.5.1) between the numbers :

$$x = x_{n+m} x_{n+m-1} \dots x_{n+1} x_n x_{n-1} \dots x_1 x_0, \quad (3)$$

$$y = y_{s+t} y_{s+t-1} \dots y_{s+1} y_s y_{s-1} \dots y_1 y_0 \quad (4)$$

where $n+1$ and m are the lengths of the nonrepeating and of the repeating part of x correspondingly, and $s+1$ and t are the lengths of the nonrepeating and of the repeating part of y correspondingly.

5.2.1 Addition and Subtraction.

Both addition and subtraction proceed the same way, because instead of subtracting y from x , we may add $\bar{y}$ to x instead, where $\bar{y}$ is the true complement of y as defined in section 4.5.1

The function $Addition(r_1, r_2)$ will accept two pointers corresponding to the two linked lists to be added, and creates a new linked list which will represent the sum of the p -adic numbers, returning the pointer to the sum list.

At each step i , we add $x_i + y_i + c_i$ and we produce the sum digit s_i and a carry c_{i+1} , so that

$$\begin{aligned} s_i &= (x_i + y_i + c_i) \bmod p \\ c_{i+1} &= (x_i + y_i + c_i) \text{ div } p, \quad c_0 = 0 \end{aligned} \quad (5)$$

The s_i 's will be the entries of the sum list.

For the algorithm to terminate, a cycle must be recognized or a user-defined termination condition has to be met.

We may detect a cycle by applying the method proposed in section 4.5.2. I.e., store the state S_i and compare it with the states $S_{i+1}, S_{i+2}, \dots, S_{2i}$; repeat this procedure for $i = 1, 2, \dots$ until a match is found, in which case we place the quote at its proper position and terminate the algorithm.

This method suggests a proper definition of the state, which ensures that a cycle has occurred.

Note that if $S_i = (x_{i-1}, y_{i-1}, c_{i-1})$, although S_i may be equal to S_j , we cannot be sure whether a cycle has been detected, unless the digits of x and y comprising S_i and S_j are in the repeating part and at a distance of k digit positions apart, where $k = LCM(m, t)$ (m and t are as in (3) and (4)).

Note also, that if both S_i and S_j occur in the repeating parts of x and y and are at a distance of k digit positions apart, this ensures that $x_i = x_j$ and $y_i = y_j$ (or, in other words, that $x_i = x_{i+k}$ and $y_i = y_{i+k}$) and therefore the carry c itself suffices for defining the state S .

Hence, if $l = \max(n, s)$ (where n and s are as in (3) and (4)), we set $S_1 = c_{l+1}$, $S_2 = c_{l+k+1}$, $S_3 = c_{l+2k+1}, \dots$ and so on.

Example :

Suppose that $p = 7$, $x = 62'2422$ and $y = 634'22$. The linked-list representation of x and y will be:

The following table will illustrate the addition process and the cycle detection.

Table 4

i	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
x_i	...	2	6	2	6	2	6	2	6	2	6	2	6	2	2	4	2
y_i	...	6	3	4	6	3	4	6	3	4	6	3	4	6	3	4	2
c_{i+1}	...	1	①	1	1	0	1	1	①	1	1	0	1	1	①	0	1
s_i	...	2	3	0	5	6	4	2	3	0	5	6	4	1	6	1	4

In this example, $n = 3$ and $s = 1$; therefore $l = \max(3, 1) = 3$

Also, $m = 2$ and $t = 3$, hence $k = LCM(2, 3) = 6$. Therefore,

$$S_1 = c_4 = 0$$

$$S_2 = c_{10} = 1$$

$$S_3 = c_{16} = 1$$

and so on. The states have been circled in the above table.

According to the method for detecting cycles, the following steps will take place:

Store S_1 ; compare S_1 vs. S_2 ($0 \neq 1$).

Store S_2 ; compare S_2 vs. S_3 ($1 = 1$). A match is found; hence a cycle has been detected.

Therefore, place the quote between the digits s_9 and s_{10} and terminate the algorithm. The result will be the following number:

$$s = 305642'3056416144$$

Note that this result is subject to normalization. The final normalized result will be:

$$s = 230564'16144$$

The subtraction operation proceeds in a quite similar way, except that now we shall have borrows instead of carries. Of course, it may also be performed by adding to x the complement of y , as we mentioned in the beginning of the present section.

5.2.2 Division and Multiplication.

The product of a p -adic number as in (3) by an integer d , such that $0 < d \leq p-1$ can be computed in a straightforward manner as follows:

At each step i , we form the digit m_i of the product and a carry c_{i+1} , so that

$$\begin{aligned} m_i &= (x_i \cdot d + c_i) \bmod p \\ i &= 0, 1, 2, \dots \quad (6) \\ c_{i+1} &= (x_i \cdot d + c_i) \operatorname{div} p, \end{aligned}$$

where $c_0 = 0$. Now, we set $S_1 = c_n$, $S_2 = c_{n+m}$, $S_3 = c_{n+2m}$, . . . and so on, and we follow the method given in section 4.5.2 for cycle detection

Example :

Let $p = 7$, $x = 32'451$ and $d = 5$. The following table gives the product $m = x \cdot d$.

<u>Table 5</u>	
i	... 8 7 6 5 4 3 2 1 0
x_i	... 3 2 3 2 3 2 4 5 1
d	5
c_{i+1}	2 1 2 1 2 1 3 3 0
m_i	... 5 4 5 4 2 6 2 4 5

In this example we have the following states: $S_1 = 3$, $S_2 = 2$, $S_3 = 2$, $S_4 = 2, \dots$ and since $S_2 = S_3$, a cycle has been detected; the result will be:

$$m = x \cdot d = 54'26245.$$

The product of x by an integer $d = d_s d_{s-1} \dots d_1 d_0$ can now be defined by means of multiplication by digit and addition operation as follows:

Set $P_0 = x \cdot d_0$, and for $j = 1$ to s compute

$$P_j = P_{j-1} + x \cdot d_j A(j), \quad (7)$$

where $x \cdot d_j A(j)$ means that the product $x \cdot d_j$ has been shifted to the left by j digit positions

Then, $P_n = x \cdot d$, and this result is subject to normalization.

However, because of the difficulties involved in the definition of the state and therefore of the cycle detection when the multiplicand and the multiplier are in the form of (3) and (4) correspondingly, it is much easier to define the division operation when x is divided by y , in which case the state definition is quite obvious, and then define the product of x by y to be equal to the quotient when x is divided by y^{-1} .

In order to divide x by y (where x and y are as in (3) and (4)), we proceed as follows:

At the first step we set x as the zeroth partial remainder R_0 and we try to find a digit which when multiplied by the rightmost digit y_0 of the divisor y , gives a result whose rightmost digit equals to the rightmost digit x_0 of the dividend x . Call the digit which has these properties q_0 , this will be the rightmost digit of the quotient q .

Now set $R_1 = R_0 - y \cdot q_0$, discard its rightmost digit, which will be equal to zero, and repeat the same process (for R_1 and y) in order to find q_1 and so on, until two identical partial remainders have been found, in which case a cycle has been detected, and the algorithm terminates.

Hence, the states will now be equal to the partial remainders which we find at each step of the division process, with S_1 being equal to x , $S_2 = R_1$, $S_3 = R_2$, and so on.¹

¹ For the implementation of the division process by means of the linked-list structure, a procedure to compare linked lists is required at this stage.

Example : Let $p = 7$, $x = 23'5$ and $y = 5'1$. The division of x by y is given below:

$$23'5 \div 5'1 = 34361501620665'$$

$$\begin{array}{r}
 -1'045 \\
 \hline
 21'60 \\
 \\
 -6'26 \\
 \hline
 12'60 \\
 \\
 -6'26 \\
 \hline
 12'200 \\
 \\
 -4'32 \\
 \hline
 43'60 \\
 \\
 -6'26 \\
 \hline
 43'510 \\
 \\
 -5'1 \\
 \hline
 45'500 \\
 \\
 -1'045 \\
 \hline
 34'410 \\
 \\
 -5'1 \\
 \hline
 45'60 \\
 \\
 -6'26 \\
 \hline
 45'530 \\
 \\
 -3'13 \\
 \hline
 12'40 \\
 \\
 -2'164 \\
 \hline
 56'63 \\
 \\
 -3'13 \\
 \hline
 23'50
 \end{array}$$

(8)

However, note that the result given in (8) is normalized. In this particular division, the following comparisons will be performed before the detection of the cycle:

Table 6

<i>store</i> $S_1 = 23'5$	<i>store</i> $S_{16} = 21'6$
$S_1 \neq 21'6 = S_2$	$S_{16} \neq 12'6 = S_{17}$
<i>store</i> $S_2 = 21'6$	$S_{16} \neq 12'20 = S_{18}$
$S_2 \neq 12'6 = S_3$	$S_{16} \neq 12'2 = S_{19}$
$S_2 \neq 12'20 = S_4$	$S_{16} \neq 43'6 = S_{20}$
<i>store</i> $S_4 = 12'20$	$S_{16} \neq 43'51 = S_{21}$
$S_4 \neq 12'2 = S_5$	$S_{16} \neq 45'50 = S_{22}$
$S_4 \neq 43'6 = S_6$	$S_{16} \neq 45'5 = S_{23}$
$S_4 \neq 43'51 = S_7$	$S_{16} \neq 34'41 = S_{24}$
$S_4 \neq 45'50 = S_8$	$S_{16} \neq 45'6 = S_{25}$
<i>store</i> $S_8 = 45'50$	$S_{16} \neq 45'53 = S_{26}$
$S_8 \neq 45'5 = S_9$	$S_{16} \neq 12'4 = S_{27}$
$S_8 \neq 34'41 = S_{10}$	$S_{16} \neq 56'63 = S_{28}$
$S_8 \neq 45'6 = S_{11}$	$S_{16} \neq 23'5 = S_{29}$
$S_8 \neq 45'53 = S_{12}$	$S_{16} = 21'6 = S_{30}$
$S_8 \neq 12'4 = S_{13}$	<i>Stop.</i>
$S_8 \neq 56'63 = S_{14}$	
$S_8 \neq 23'5 = S_{15}$	
$S_8 \neq 21'6 = S_{16}$	

Therefore, the result of the division process using the method described in section 4.5.2 for cycle detection, would produce the following unnormalized result:

$$q = z \div y = 53436150162066'534361501620665.$$

Note that we may remove the restriction that the rightmost digit of the divisor must be nonzero, by allowing a p -adic point in the representation. In order to eliminate the l rightmost digits of the divisor, we divide both the dividend and the divisor by p^l ($l \geq 1$).

The same rule applies also when multiplying two numbers; the rightmost digit of the multiplier must be again nonzero. The reason is that we have defined the multiplication operation in terms of division; i.e. $x \cdot y = x \div y$, which requires the multiplicative inverse of y (i.e. $y^{-1} = 1 \div y$) and therefore, the rightmost digit of y must be nonzero.

Example : Let $p = 7$.

Instead of $3245'4300 \div 3'20$, perform $3245'430 \div 3'2$.

Instead of $25'323 \div 6'300$, perform $25'3.23 \div 6'3$, and so on.

Chapter 6

Appendix

Computer Listing

Note : For reasons of legibility, the comments' symbols have been omitted .

PROGRAM MultiplePrecisionArithmetic(input,output);

```
const maxstack=80;
      strsize=1000;
      maxcols=80;
```

```
type Pointer="Node;
      Node=record
          next:Pointer;
          info:integer
        end;
      stack=record
          item:array[1..maxstack] of Pointer;
          top:0..maxstack
        end;
      List=packed array[1..strsize] of char;
      exprtype=array[1..maxcols] of char;
```

```
var length1,length2,tdigits,finalposition,
    B,newdigits,termination,lengthR:integer;
    NReplength,Replength,lengthS:integer;
    opnd1,opnd2,value,Result,f:Pointer;
    st,opndstk:stack;
    Y:char;
    string>List;
    D,C,exit23:boolean;
    A:integer;
    expr:exprtype;
    position:1..maxcols;
```

```

end
else k = k+1;
exit4 k=1;
f = f next
until exit4,
for j =1 downto 1 do write(string[j]),
writeln
end,
*****

```

The following function adds two p-adic numbers which are represented as two linked lists pointed to by 'r' and 's', and returns a pointer to a new list (the sum list) which contains the sum of the given p-adic numbers. We assume that the first of the operands has the greater non-repeating length, otherwise we flip them around. If 'r' points to the first operand and 's' points to the second one, then

length1 length of the repeating part of 'r'
length2 length of the repeating part of 's'
tdigits counts the digits of the sum-list
lengthR length of the nonrepeating part of 'r'
lengthS length of the nonrepeating part of 's'
newdigits length of the nonrepeating part of the sum-list

The pointer 'f' points to the head of the sum-list, while the pointer 't' constructs the new list and 'u' connects its 'cells'

```

-----
FUNCTION Addition(r,s Pointer) Pointer,
var u,t,p1,p2,p3,temp Pointer,
    match,condition,exit1,exit2 boolean,
    count,i,j,sum,carry,
    length3,x,y integer,
    stage,newstage,position,templength1,templength2 integer,
begin
    FindLengths(r),
    lengthR = NRReplength,
    length1 = Replength,
    FindLengths(s),
    lengthS = NRReplength,
    length2 = Replength,
    if lengthR < lengthS then begin
        temp = r,
        r = s,
        s = temp,
        temp.length1 = lengthR,
        lengthR = lengthS,
        lengthS = templength1,
        templength2 = length1,
        length1 = length2,

```

```

PROCEDURE push(u Pointer).
This procedure places an element at the top of the stack
begin
  if opndstk top=maxstack
  then writeln('Error!!The stack is full')
  else begin
    opndstk.top =opndstk.top+1,
    opndstk.item[opndstk top] =u
  end
end,
*****

```

```

FUNCTION EQUALITY(r,s Pointer) Boolean,
A function which checks whether two p-adic numbers are equal
var k integer,
    exit20,condition10,condition20,exit50,match boolean,
begin
  k =0,
  match =true,
  repeat
    if r.info<>s.info then match =false,
    condition10:=r.info<0,
    condition20:=s.info<0;
    if condition10 or condition20 then k =k+1,
    r:=r.next,
    s:=s.next;
    exit50:=match=false,
    exit20 =k=2
  until exit20 or exit50,
  EQUALITY =match
end,
*****

```

```

FUNCTION gcd(length10,length20:integer):integer;
begin
  if length20=0 then gcd:=length10
  else gcd:=gcd(length20,length10 mod length20)
end;
FUNCTION LCM(length10,length20:integer):integer;
begin
  LCM:=length10*length20 div gcd(length10,length20)
end,
*****

```


The following function gives to a p-adic number the linked list structure and returns a pointer pointing to that list

The pointer 'w' constructs the list, while two other pointers 'q' and 's' are used to connect the 'cells' of the list.

Other variables used

digits length of nonrepeating part of the
 input number,
length length of repeating part of the
 input number.

```

FUNCTION Get.Pointer,
var  w,q,s Pointer,
     i,j,digits,length integer,
begin
  for i =1 to strsize do string[i] := '0',
  i:=0,
  repeat
    i:=i+1,
    read(Y),
    string[i] := Y,
    write(string[i])
  until eoln,
  length:=0,
  writeln,
  digits:=0,
  j:=1,
  readln,
  new(w),
  Get:=w,
  repeat
    if j<>0 then C:=string[j]='',
    if not C then begin
      w.info:=ord(string[j])-ord('0'),
      q:=w,
      digits:=digits+1,
      new(w),
      q.next:=w,
      j:=j-1
    end,
    D:=j=0
  until C or D,
  if D then begin
    w.info:=-digits-1,
    q:=w,
    new(w),
    q.next:=w,
    w.info:=0,
    length:=1,
    w.next:=q
  end
  else begin

```

```

w^.info := -digits - 1;
q := w, leave a pointer before the quote
s := q,
j := j - 1;
D := j = 0,
while not D do
begin
  new(w);
  w^.next := w,
  s := w,
  w^.info := ord(string[j]) - ord('0'),
  length := length + 1;
  j := j - 1;
  D := j = 0
end;

w^.next := q
end,
writeln
end,
*****

```

The following procedure prints the contents of a linked list pointed to by 'f' as a p-adic number.

```

PROCEDURE Print(f: Pointer);
var k, i, j, Y: integer;
    exit4: boolean;
begin
  for i := 1 to strsize do string[i] := '0',
  i := 0;
  k := 0;
  repeat
    if f^.info >= 0 then begin
      i := i + 1;
      Y := f^.info;
      A := Y + ord('0');
      string[i] := chr(A)
    end
    else k := k + 1;
    exit4 := k = 1;
    f := f^.next
  until exit4,
  i := i + 1;
  string[i] := '';
  k := 0;
  repeat
    if f^.info >= 0 then begin
      i := i + 1;
      Y := f^.info;
      A := Y + ord('0');
      string[i] := chr(A)
    end
  until exit4;

```

```

end
else k = k+1.
exit4 = k+1.
f = f next
until exit4.
for j =1 downto 1 do write(string[j]).
writeln
end.
*****

```

The following function adds two p-adic numbers which are represented as two linked lists pointed to by 'r' and 's' and returns a pointer to a new list (the sum list) which contains the sum of the given p-adic numbers. We assume that the first of the operands has the greater non-repeating length, otherwise we flip them around. If 'r' points to the first operand and 's' points to the second one, then

length1 length of the repeating part of 'r'
length2 length of the repeating part of 's'
tdigits counts the digits of the sum-list
lengthR length of the nonrepeating part of 'r'
lengthS length of the nonrepeating part of 's'
newdigits length of the nonrepeating part of the sum-list

The pointer 'f' points to the head of the sum-list, while the pointer 't' constructs the new list and 'u' connects its 'cells'

```

-----
FUNCTION Addition(r,s Pointer) Pointer,
var u,t,p1,p2,p3,temp Pointer,
    match,condition,exit1,exit2 boolean,
    count,i,j,sum,carry,
    length3,x,y integer,
    stage,newstage,position,templength1,templength2 integer.
begin
    FindLengths(r),
    lengthR =NRlength,
    length1 =Replength,
    FindLengths(s),
    lengthS =NRlength,
    length2 =Replength,
    if lengthR<lengthS then begin
        temp =r,
        r = s,
        s = temp,
        temp,length1 =lengthR,
        lengthR =lengthS,
        lengthS =templength1,
        templength2 =length1,
        length1 =length2,

```

```

var mdigits, carry, k, newstage, stage, i, proddigits, j integer,
    condition, match, exit1, exit2 boolean,
    t, u, f, p1, p3 Pointer,
begin
    match = false,
    mdigits = 0,
    carry = 0,
    new(t),
    k = 0,
    u = t, to connect the list
    f = t, to refer
    repeat
        if r~ info >= 0 then begin
            new(t),
            u~ next = t,
            t~ info = (r~ info * constant + carry) mod B,
            carry = (r~ info * constant + carry) div B,
            mdigits = mdigits + 1
        end
        else k = k + 1,
            r = r~ next,
            u = t,
        condition = k <> 0
    until condition,
    newstage = carry,
    i := 1, counts the stages that we store
    repeat
        p1 = u, leave a pointer at the stage you keep
        proddigits = mdigits,
        stage = newstage, the stages we store
        j = 1,
        repeat
            p3 = t, a pointer which follows t
            if r~ info >= 0 then begin
                new(t),
                u~ next = t,
                p3 = t,
                t~ info = (r~ info * constant +
                    carry) mod B,
                carry = (r~ info * constant +
                    carry) div B,
                mdigits = mdigits + 1,
                u = t
            end
            else begin
                newstage = carry,
                if newstage = stage
                    then begin
                        new(t),
                        t~ info = -proddigits - 1,
                        t~ next = p1~ next,
                        p1~ next = t,
                        p3~ next = p1~ next,
                        match = true
                    end
            end
        until condition
    until condition
end

```

```

        if s.info < 0 then s := s.next,
        position := position + 1
    end,
    r := r.next,
    u := t,
    p3 := t,
    if position = length3 div length1
    then begin
        newstage := carry,
        if newstage = stage
        then begin
            new(t),
            t.info := -newdigits - 1,
            t.next := p1.next,
            p1.next := t,
            p3.next := p1.next,
            match := true
        end
        else j := j + 1
    end,
    exit1 := j = 2 * i
    until match or exit1,
    exit2 := i = termination,
    i := 2 * i
    until match or exit2,
    f := f.next,
    Addition := f
end;
*****

```

The following function subtracts two p-adic numbers represented as two linked-lists pointed to by 'r' and 's' and returns a pointer to a new list which contains the difference of the given p-adic numbers. If 'r' points to the subtrahend and 's' points to the minuend, then :

The pointer 'p5' points to the head of the list, while 't' constructs the list and 'u' connects the 'cells' of the list.

```

FUNCTION Subtraction(r,s:Pointer):Pointer;
var k:integer;
    p5,m,n,l,p7,p8,p18,t,u:Pointer,
    escape:boolean,
begin
    k:=0;
    new(t);
    p5:=t;
    u:=t;
    repeat

```

```

        if s.info < 0 then begin

```

```

        if k=0 then begin
            new(t),
            u^.next:=t,
            t^.info:=s^.info,
            s:=s^.next,
            k:=k+1,
            p18:=t,
            u:=t
        end
        else k2=k+1
    end
    else begin
        new(t);
        u^.next:=t,
        t^.info:=B-1-s^.info,
        s:=s^.next;
        u:=t
    end,
    escape:=k=2
until escape,
p5:=p5^.next,
t^.next:=p18,
new(l),
l^.info:=1,
m:=1,
n:=1,
new(l);
m^.next:=1;
l^.info:=-2,
m:=m^.next;
new(l);
m^.next:=1;
l^.info:=0;
l^.next:=m;
p7:=Addition(p5,n),
p8:=Addition(r,p7),
Subtraction:=p8
end;
*****

```

The following function multiplies a p-adic number (which is represented as a linked list) by a positive integer and returns a pointer to the 'product list'.
 ndigits : counts the elements of the new list,
 proddigits : length of the nonrepeating part of the product list.
 The pointer 'f' points to the head of the product list, while 't' constructs the list and 'u' connects its elements

```

FUNCTION LongMultiplication(r:Pointer;constant:integer):Pointer;

```

```

var mdigits, carry, k, newstage, stage, i, proddigits, j integer,
    condition, match, exit1, exit2 boolean,
    t, u, f, p1, p3 Pointer,
begin
    match:=false;
    mdigits:=0,
    carry:=0,
    new(t);
    k:=0;
    u:=t, to connect the list
    f:=t; to refer
    repeat
        if r^.info>=0 then begin
            new(t);
            u^.next:=t;
            t^.info:=(r^.info*constant+carry) mod B,
            carry:=(r^.info*constant+carry) div B,
            mdigits:=mdigits+1
        end
        else k:=k+1,
            r:=r^.next,
            u:=t,
        condition:=k<>0
    until condition,
    newstage:=carry;
    i:=1; counts the stages that we store
    repeat
        p1:=u; leave a pointer at the stage you keep
        proddigits:=mdigits;
        stage:=newstage; the stages we store
        j:=1;
        repeat
            p3:=t; a pointer which follows t
            if r^.info>=0 then begin
                new(t),
                u^.next:=t;
                p3:=t;
                t^.info:=(r^.info*constant+
                    carry) mod B;
                carry:=(r^.info*constant+
                    carry) div B;
                mdigits:=mdigits+1;
                u:=t
            end
            else begin
                newstage:=carry;
                if newstage=stage
                then begin
                    new(t);
                    t^.info:=-proddigits-1,
                    t^.next:=p1^.next;
                    p1^.next:=t;
                    p3^.next:=p1^.next;
                    match:=true
                end
            end
        until condition
    until condition
end

```

```

                                else j:=j+1
                                end;
                                r:=r`next;
                                exit1:=j=2*i
                                until match or exit1;
                                exit2:=i=termination;
                                i:=2*i
                                until match or exit2;
                                f:=f`next;
                                LongMultiplication:=f
                                end;
                                *****

```

```

FUNCTION MODUL(a:integer):integer;
Finds the multiplicative inverse of a (modB)
var x,y,d,q,r:integer;
    conditionA,conditionB,conditionC:boolean;
begin
    x:=0;
    y:=1;
    d:=B;
    conditionA:=a mod d <>0;
    conditionB:=B mod d <>0;
    repeat
        conditionA:=a mod d <>0;
        if conditionA then begin
            r:=a mod d;
            q:=a div d;
            x:=1-x*q;
            y:=-q*y;
            d:=r
        end;
        conditionB:=B mod d <>0;
        if conditionB then begin
            r:=B mod d;
            q:=B div d;
            x:=-x*q;
            y:=1-q*y;
            d:=r
        end
    until not conditionA and not conditionB;
    MODUL:=x
end;
*****

```

The following function divides two p-adic numbers (which are represented as linked lists) and returns a pointer to a new list which contains their quotient. The pointer 'f' points to the head of the list (i.e. the least significant digit), while 'q' constructs the list, and 'u'

connects its elements.

We also use the following variables .

e : the rightmost digit of the dividend
 a : the rightmost digit of the divisor
 c : the multiplicative inverse of a (mod B)
 stage: the partial remainders that we store
 newstage : the intermediate stages

```

-----
FUNCTION Division(r,s:Pointer):Pointer;
var i,j,a,e,c,divdigits,newdigits:integer;
    q,l,stage,newstage,m,g,t,u,f,p10,p30:Pointer;
    exit25,exit30,match:boolean;
begin
  divdigits:=0;
  i:=1;
  new(q);
  u:=q;
  f:=q;
  newstage:=r;
  repeat
    stage:=newstage;
    p10:=u;  leave a pointer at the stage you keep
    j:=1;
    newdigits:=divdigits;
    repeat
      new(q);
      u^.next:=q;
      p30:=q;
      if s^.info<0 then begin
        g:=s^.next;
        a:=g^.info
      end
      else a:=s^.info;
      if r^.info<0 then begin
        l:=r^.next;
        e:=l^.info
      end
      else e:=r^.info;
      c:=MODUL(a);
      if c<0 then c:=c mod B;
      q^.info:=(c*e) mod B;
      divdigits:=divdigits+1;
      m:=LongMultiplication(s,q^.info);
      t:=Subtraction(r,m);
      t:=t^.next;
      newstage:=t;
      match:=EQUALITY(stage,newstage);
      j:=j+1;
      exit25:=j=2*1;
      r:=t;
      u:=q
    until match or exit25;
    if match then begin
      new(q);
      q^.info:=-newdigits-1;

```

```

        q^.next:=p10^.next;
        p10^.next:=q;
        p30^.next:=p10^.next
    end
    else begin
        exit30:=1=termination;
        i:=2*i
    end
until match or exit30;
f:=f^.next;
Division:=f
end;

```

The following function multiplies two p-adic numbers which have the linked list structure and returns a pointer to a new list which contains the product of the given p-adic numbers.

```

FUNCTION Multiplication(r,s:Pointer):Pointer;
var v,p50,p60,m,n:Pointer;

```

```

begin

```

```

    new(v);
    v^.info:=1;
    m:=v;
    n:=v;
    new(v);
    m^.next:=v;
    v^.info:=-2;
    m:=m^.next;
    new(v);
    m^.next:=v;
    v^.info:=0;
    v^.next:=m;
    p50:=Division(n,s);
    writeln;
    p60:=Division(r,p50);
    Multiplication:=p60

```

```

end;

```

```

FUNCTION oper(symb:char;op1,op2:Pointer):Pointer;

```

```

begin

```

```

    if symb in ['+', '*', '-', '/']
    then case symb of
        '+':oper:=Addition(op1,op2);
        '*':oper:=Multiplication(op1,op2);
        '-':oper:=Subtraction(op1,op2);
        '/':oper:=Division(op1,op2)
    end

```

```

    end

```

```

    else writeln('Error!! illegal operator')

```

```

end;

```

```

*****
FUNCTION opnd(symb:char) boolean,
var operator boolean,
begin
    operator:=symb in ['+', '*', '-', '/'];
    if not operator an operand is found
    then opnd:=true
    else opnd:=false
end;
*****

FUNCTION eval(expr:exptype):Pointer,
    EVAL accepts a postfix expression as input
    and returns a pointer to the value of the
    expression
const maxstack:=maxcols;
var symb:char,
    position 1 maxcols,
begin
    opndstk.top:=0,
    position:=1,
    symb:=expr[position],
    position indicates the current position in the expression
    and symb is the current symbol.Scan until ' ' is found
    while symb<>' ' do
        begin
            if opnd(symb) then begin
                writeln('The p-adic number ',symb,' is: '),
                value:=Get;
                push(value);
            end
            else begin
                opnd2:=pop;
                opnd1:=pop;
                value:=oper(symb,opnd1,opnd2),
                push(value)
            end;
            if position<finalposition then begin
                position:=position+1,
                symb:=expr[position]
            end
            else symb:=' '
        end;
    eval:=pop
end;
*****

```

The main program follows :

```

begin
  writeln('Give the base B');
  readln(B);
  writeln('Give the termination number');
  readln(termination);
  writeln('Give the expression in postfix notation');
  position:=1;
  while not eoln
    do begin
      finalposition:=position;
      read(expr[position]);
      position:=position+1
    end;
  readln;
  Result:=eval(expr);
  position:=1;
  repeat
    write(expr[position]);
    exit23:=position=finalposition;
    if not exit23 then position:=position+1
  until exit23;
  write(' = ');
  Print(Result);
  writeln('For the operands given above. ');
  writeln('*****')
end.
*****

```

Results

Give the base B

*7

Give the termination number

*100

Give the expression in postfix notation

ABC/

The p-adic number A is:

*123*54

123*54

The p-adic number B is:

*32*6

32*6

The p-adic number C is:

*32*6

32*6

ABC/* = 312*354

For the operands given above.

!*L

Give the base B

*7

Give the termination number

*1000

Give the expression in postfix notation

*ABC/*D+E-F/

The p-adic number A is:

*32*45

32*45

The p-adic number B is:

*2*5

2*5

The p-adic number C is:

*2*5

2*5

The p-adic number D is:

35

35*

The p-adic number E is:

*5

5

The p-adic number F is:

*6

6

ABC/*D+E-F/ = 011223344560*011223344560012
 For the operands given above.

!*L
 Give the base B
 *7
 Give the termination number
 *100
 Give the expression in postfix notation
 KF+E+D-C/B
 The p-adic number K is:
 *011223344560*011223344560012
 011223344560*011223344560012

The p-adic number F is:
 *6
 6

The p-adic number E is:
 *5
 5

The p-adic number D is:
 35
 35*

The p-adic number C is:
 *2*5
 2*5

The p-adic number B is:
 *2*5
 2*5

KF+E+D-C/B* = 23*2323232323245
 For the operands given above.

!*L
 Give the base B
 *7
 Give the termination number
 *100
 Give the expression in postfix notation
 *AB/
 The p-adic number A is:
 *23*5
 23*5

The p-adic number B is:
 *5*1
 5*1

AB/ = 53436150162066*53436150162066

For the operands given above.

!*L

Give the base B

*7

Give the termination number

*100

Give the expression in postfix notation

*AB/C*D+E-

The p-adic number A is

*2*3

2*3

The p-adic number B is:

*5

5

The p-adic number C is

*5

5

The p-adic number D is:

*56*321

56*321

The p-adic number E is:

*23*4

23*4

AB/C*D+E- = 55*55310

For the operands given above.

References

- [1] Aho, Hopcroft, Ullmann :*Design and Analysis of Computer Algorithms*. Reading, MA: Addison Wesley, 1974.
- [2] Bachman G. :*Introduction to P-adic numbers and valuation theory*. New York, Academic Press, 1964.
- [3] Borevich Z.I. and I.R. Shafarevich :*Number Theory*. New York, Academic Press, 1966.
- [4] Dixon J. :*Exact solution of Linear Equations using P-adic expansions*. Numer. Math. 40, pp.137-141 (1982).
- [5] Froment A. :*Error free computation : A direct method to convert finite-segment P-adic numbers into Rational numbers* IEEE trans. on computers, vol. C-32, No 4. April 1983, pp.337-343.
- [6] Gregory R.T. :*Error-free computation with rational numbers*. Bit 21, (1981), pp.194-202
- [7] Gregory R.T. :*The use of finite-segment P-adic arithmetic for exact computation*. Bit 18, (1978) pp.282-300.
- [8] G.H. Hardy, E.M. Wright :*An Introduction to the theory of Numbers*. Oxford, at the Clarendon Press (1960).
- [9] E.C.R. Hehner and R.N.S. Horspool :*A new representation of the rational numbers for exact easy arithmetic*. SIAM J. comput. vol.8, No 2, May 1979, pp.124-134.
- [10] A.V. Khinchin :*Continued fractions*. 3rd ed. Chicago: Univ. Chicago Press, (1961).
- [11] D.E. Knuth :*The Art of Computer Programming*. vol.2: *Seminumerical Algorithms*. Reading, MA: Addison Wesley, (1969).
- [12] D.E. Knuth :*An Imaginary Number System*. Comm. ACM, vol.3, April 1960, pp.245-247.

- [13] I Koren, Y Maliniak : *On classes of Positive, Negative and Imaginary Radix Number Systems* IEEE Trans. on computers, vol C-30, No 5, May 1981, pp 312-317
- [14] E V Krishnamurthy : *Matrix processors using P-adic Arithmetic for exact linear computations* IEEE Trans. on computers, vol C-26, No 7, July 1977, pp 633-63
- [15] E.V Krishnamurthy, M.Rao, Subramanian : *P-adic Arithmetic procedures for exact matrix computations*. Proc. Indian Acad. Sci., vol.82A, Nov 1975, pp 165-175
- [16] E V Krishnamurthy, T.M.Rao, K.Subramanian : *Finite-segment P-adic Number Systems with applications to exact computation* Proc. Indian Acad. Sci., vol 81A, Feb 1975, pp 58-79
- [17] E.V.Krishnamurthy : *On the conversion of Hensel codes to Farey Rationals* IEEE Trans. on computers, vol C-32, No.4, April 1983, pp.331-336.
- [18] E.V.Krishnamurthy, V.K.Murthy : *Fast iterative Division of P-adic Numbers*. IEEE Trans. on computers, vol.C-32, No.4, April 1983, pp.396-398.
- [19] A.G.Slekys, A.Avizienis : *A modified bi-imaginary number system*. Proc. 4th Symp. Comput Arithmetic, Oct. 1978, pp.48-55.
- [20] N.S.Szabo, R.I.Tanaka : *Residue arithmetic and its applications to computer technology* McGraw Hill Book Company, New York (1967).

END

10-06-186

FIN