NEGATIVE BASE NOTATION AND CALCULATION

Waldean Schulz

September 3, 1966

NSF Research Grant

University of Colorado

Boulder, Colorado

Abstract

Any negative ~~number~~ integer less than -1 may be used for the base in radix representation of the integers and the rationals. The need for a minus sign for negative numbers is not then required. This paper studies the theory of this type of numerical notation and provides algorithms for addition, subtraction, multiplication, division, extraction of square root, and checking. The ninth chapter is a specific study of base minus two — motivated by digital computer application.

TABLE OF CONTENTS

It has been very convenient to assign a unique name to each of the natural numbers (or at least a large set of them). Nevertheless, people seldom differentiate between a natural number, or integer, as a concept and the arbitrary name for it. In nearly all cultural systems of notation each natural number is a combination of a small number of symbols, or digits. The characteristics of the natural number, of course, are independent of the name given to it, but the ease of calculation is very directly related to the way in which it is represented. The Romans, for example, foozled disasterously in providing a system of notation with a simple multiplication algorithm. Also, new symbols were required as the number became larger.

The growth of the computer industry has accented the study of new, convenient notations for integers. The most common representation, linear and positional, is defined by

$$N = a_0 + a_1 b + a_2 b^2 + \ldots + a_k b^k$$

with N written as $a_k a_{k-1} \ldots a_2 a_1 a_0$

where N = any positive integer
b = some fixed integer called the base or radix
a_i, the symbol of an integer, $0 \leq a_i < b$ for $0 \leq i \leq k$ but $a_k \neq 0$.

In the ring of integers the negative of N is, of course, written as $-a_k \ldots a_1 a_0$ and the additive and multiplicative identity elements as 0 and 1. However, whenever an integer is refered to as an entity in this paper, it will sometimes be spelled out (e.g. ZERO, ONE...; NEGATIVE ONE, NEG TWO...). Even this is a (positional) notation, but it is more suggestive of the integer itself

rather than a symbol for it.

After proving the above representation is a one-to-one onto mapping, most number theory books dismiss the subject after giving a specific example:

$$N = v10^4 + w10^3 + x10^2 + y10 + z \quad \text{written} \quad vwxyz.$$

This is a somewhat circular, sloppy definition of base ten notation, because 10 is itself of this form: $10 = 1 \cdot 10 + 0$. The reader still does not know what this pair of symbols, 10, means. Similarly, in base seven the term $a_n 7^n$ is ambiguous, since only $0,1,\ldots,6$ are "legal" symbols. This might be avoided by replacing 7 with SEVEN, but the subscripts and exponents still present a similar problem. For instance, in Theorem 1.1 the symbol 2 appears, which is meaningless in base TWO or NEGATIVE TWO. Like everybody else, I will ignor this fine point and write all exponents and subscripts in base TEN.

However, there is an important, generally unrecognized point that needs to be made about radix notation: the base, b, need not be positive. The usual definition or theorem actually forces b to be positive: $0 \leq a_i < b$. It is the object of this paper to demonstrate the consequences of a negative base, b.

1. Negative base representation

All variables are integers unless otherwise noted.

1.1 Theorem Let b be any fixed integer less than -1. Then every integer n is representable in the form

$$n = a_0 + a_1 b + a_2 b^2 + \ldots + a_k b^k$$

where $0 \leq a_i < |b|$ for $0 \leq i \leq k$ for some k.

PROOF: Case A. Suppose $n > 0$. Use induction on n.

1) $n = a_0 = 1$ So 1 is representable.

2) Assume n-1 is representable. Show that n is.

$$n-1 = a_0 + a_1 b + a_2 b^2 + \ldots + a_k b^k$$

$$\therefore\ n = (a_0 + 1) + a_1 b + a_2 b^2 + \ldots + a_k b^k$$

If $(a_0 + 1) < |b|$, n is of proper form and therefore is representable in the desired way.

If not, i.e. $a_0 + 1 = |b| = -b$, (1.1.1)

$$n = -b + a_1 b + a_2 b^2 + \ldots + a_k b^k$$

$$= 0 + (a_1 - 1)b + a_2 b^2 + \ldots + a_k b^k \quad \bullet$$

If $0 \leq a_1 - 1$, then n is in desired form.

If not, i.e. $a_1 = 0$, note that $-b^i = (-b-1)b^i + b^{i+1}$.

$n = 0 - b + a_2 b^2 + \ldots + a_k b^k$ set i = 1 in the above, so

$$= 0 + (-b-1)b + b^2 + a_2 b^2 + \ldots + a_k b^k$$

$$= 0 + (-b-1)b + (a_2+1)b^2 + \ldots + a_k b^k \quad \bullet$$

If $a_2 + 1 < |b| = -b$, n is then in proper form.

If not, i.e. $a_2 + 1 = -b$, then continue as in (1.1.1) and following. Clearly this (carrying) process can take place no more than k+1 times since there are only k+1 terms. If the process continues through the k^{th} term, it stops; for

if the rightmost term has become $(a_k-1)b^k$ and $a_k-1<0$, i.e. $a_k = 0$, then the term can be replaced by

$$\ldots+(-b-1)b^k + b^{k+1}.$$

But if $a_k-1 = 0$ then no changes are needed; it is already in proper form. If, on the other hand, the rightmost term has become $(a_k+1)b^k$ and $a_k+1 = -b$, then this can be replaced by

$$\ldots+0\,b^k + (-b-1)b^{k+1} + b^{k+2}$$

But if $a_k+1 < -b$, it is already in the desired form.

Thus all $n>0$ is representable in the desired form.

Case B. Suppose $n<0$. Let $n = -m$. Therefore $m>0$. Use induction on m.

1) $m = 1$ or $n = -m = -1 = (-b-1) + 1\cdot b$

2) Assume $(n+1) = -(m-1)$ is representable. Show that $n = -m$ is.

$$(n+1) = -(m-1) = a_0 + a_1 b + \ldots + a_k b^k$$

$$-m = -1 + a_0 + a_1 b + \ldots + a_k b^k$$

$$= (a_0-1) + a_1 b + \ldots + a_k b^k$$

If $a_0 > 0$, $0 \leq a_0-1 < |b|$ and $-m = n$ is in proper form.

If $a_0 = 0$, again note that $-b^i = (-b-1)b^i + b^{i+1}$

So $-m = (-b-1) + b + a_1 b + \ldots + a_k b^k$

$$= (-b-1) + (a_1+1)b + \ldots + a_k b^k \qquad **$$

If $a_1+1 < |b| = -b$, then $-m = n$ is in the desired form.

If not, i.e. $a_1+1 = -b$, then follow the (carrying) procedure of case A.* The result is, as in A, a polynomial of the desired form. Therefore, the theorem holds for all positive m — that is, all negative n.

Case C. $n = 0$ is in above form. $k = 0$, $a_0 = 0$.

*Note the structure of carrying. There are "two ways" of carrying— depending on the conditions.

**Note the structure of borrowing. (Actually, there are "two corresponding ways" of borrowing, too.)

1.2 Definitions Given a fixed integer b, called the base or radix, and given a set of (arbitrary) symbols for ZERO, ONE,...(-b-ONE), then $n = a_0 + a_1 b + \ldots + a_k b^k$ in theorem 1.1 $(a_k \neq 0)$ is written as $a_k a_{k-1} \ldots a_1 a_0$ where a_i is one of the above symbols corresponding to the integral coefficient a_i and *juxtaposition here does not indicate multiplicatio* a_i is called a digit.

If n = ZERO then n is written as 0.

1.3 Corollary Any positive integer is representable in any integral base, B, if $|B|>1$.

PROOF: Combine thm 1.1 with the corresponding theorem for positive B. A trivial case for B = +1 or -1 may be defined if a_i is allowed to be equal to 1. For example, if B = 1, $3 = 1 + 1\cdot 1 + 1\cdot 1^2$. Also, if B = -1, $-3 = 0 + 1(-1) + 0(-1)^2 + 1(-1)^3 + 0(-1)^4 + 1(-1)^5$. However, without further restrictions these representetions are not unique.

In order to make the polynomial in theorem 1.1 unique for a given n, a_k must be limited: $a_k \neq 0$. That is, k must be the index of the largest non-zero term. Otherwise, the addition of zero terms might be construed as changing the polynomial. This condition necessitates a separate treatment of n = 0.

1.4 Theorem Let b be any integer such that $|b|>1$. If k is a positive integer and $0 \leq a_i < |b|$, $0 \leq i \leq k$, then

$$0 = a_0 + a_1 b + \ldots + a_k b^k \qquad (1.4.1)$$

implies $a_0 = a_1 = \ldots = a_k = 0$

PROOF: $a_0 = b(-a_1 - a_2 b - \ldots - a_k b^{k-1}) = bx$ for some integer x.

$\therefore a_0 \geq |b|$ unless $a_0 = 0$

since $m|n \Rightarrow n = 0$ or $|n| \geq |m|$.

But $a_i < |b|$. So $x = 0$ and $a_0 = 0$.

$\therefore$ (1.4.1) reduces to

$$0 = a_1 + a_2 b + \ldots a_k b^{k-1} .$$

Repeating the above logic k-1 times proves

$a_0 = a_1 = \ldots = a_k = 0$.

This is the same as saying that zero has only one representation up to the addition of zero terms:

$$0 = 0 + 0\cdot b + \ldots + 0\cdot b^k \quad \text{for any } k.$$

By modifying the standard proof [7], we can prove the uniqueness of a radix expression for any given integer.

1.5 Theorem Let b be a fixed integer, $|b|>1$. Then there is a unique representation for any non-zero integer in base b:

$$n = a_0 + a_1 b + \ldots + a_k b^k$$

where $0 \leq a_i < |b|$ for $0 \leq i \leq k$, but $a_k \neq 0$. (1.5.1)

Of course n must be greater than zero if b is.

PROOF: By theorem 1.1 there is at least one representation of n in base b radix notation. Assume two (or more) exist. Then

$$n = a_0 + a_1 b + \ldots + a_k b^k = c_0 + c_1 b + \ldots + c_j b^j$$

with the conditions on the coefficients given in (1.5.1).

Then upon subtracting the two different representations, we get

$n - n = 0 = d_0 + d_1 b + \ldots + d_s b^s$ so that $d_s \neq 0$.

If $s = 0$, $d_s = d_0 = 0$, a contradiction.

If $s > 0$,
$$|b^s| \leq |-d_s b^s| = |d_0 + d_1 b + \ldots + d_{s-1} b^{s-1}|$$
$$\leq |d_0| + |d_1 b| + \ldots + |d_{s-1} b^{s-1}|$$
$$\leq (|b|-1)(|1| + |b| + \ldots + |b^{s-1}|) = |b^s| - 1 .$$

Or $|b^s| \leq |b^s| - 1$, a contradiction. Conclusion: $k = j$ and $a_i = c_i$.

The next theorem is an entirely different proof of the preceding three theorems. It it interesting because it determines the largest and smallest integers representable with a given number of powers of b, b a negative base here. It also shows that

all integers between these extremes are representable with that number of terms, and the theorem shows the position of zero in the interval bounded by the extremes. First comes a necessary lemma.

1.6 Lemma Given $b<-1$.

$$(x^0+x^{b^0}+x^{2b^0}+\ldots+x^{(-b-1)b^0})(x^0+x^{b^1}+x^{2b^1}+\ldots+x^{(-b-1)b^1})\ldots(x^0+x^{b^k}+x^{2b^k}+\ldots+x^{(-b-1)b^k})$$
$$= x^N + x^{N+1} + \ldots + x^{-1} + x^0 + x^1 + \ldots + x^{M-1} + x^M$$

where if k is even, $N=\frac{-b}{b-1}(b^k-1)$ and $M=\frac{-1}{b-1}(b^{k+2}-1)$ and
if k is odd, $N=\frac{-b}{b-1}(b^{k+1}-1)$ and $M=\frac{-1}{b-1}(b^{k+1}-1)$.

PROOF: $-b>1$. We know that the polynomial $a_0+a_1(-b)^1+a_2(-b)^2+\ldots+a_k(-b)^k$ where $0\le a_i<-b$ FOR $0\le i\le k$ generates the integers 0 through $|b|^{k+1}-1$ once for each possible set of a_i's. Therefore, we see why the following identity holds.

$$x^0+x^1+x^2+\ldots+x^{|b|^{k+1}-1} = (x^{0(-b)^0}+x^{1(-b)^0}+x^{2(-b)^0}+\ldots x^{(-b-1)(-b)^0})(x^{0(-b)^1}+x^{1(-b)^1}+\ldots+x^{(-b-1)(-b)^1})\ldots$$
$$\ldots(x^{0(-b)^k}+x^{1(-b)^k}+\ldots+x^{(-b-1)(-b)^k}) \qquad \text{Assume } k \text{ even, so}$$
$$=\left(\sum_{i=0}^{-b-1} x^{i(-b)^0}\right)\left(\sum_{i=0}^{-b-1} x^{i(-b)^1}\right)\left(\sum_{i=0}^{-b-1} x^{i(-b)^2}\right)\ldots\left(\sum_{i=0}^{-b-1} x^{i(-b)^k}\right)$$
$$=(\textstyle\sum x^{ib^0})(\sum x^{i(-b)^1})(\sum x^{ib^2})(\sum x^{i(-b)^3})\ldots(\sum x^{ib^k})$$
$$=(\textstyle\sum x^{ib^0})(x^{(-b-1)(-b)}\sum x^{ib^1})(\sum x^{ib^2})(x^{(-b-1)(-b)^3}\sum x^{ib^3})\ldots(\sum x^{ib^k}) \qquad *$$
$$=(x^{(-b-1)(-b)})(x^{(-b-1)(-b)^3})\ldots(x^{(-b-1)(-b)^{k-1}})\left(\prod_{r=0}^{k}\sum x^{ib^r}\right)$$

$$= \left(x^{(-b-1)(-b-b^3-b^5\ldots-b^{k-1})}\right) \prod \sum x^{ib^r}$$

$$= \left(x^{b(b^k-1)/(b-1)}\right) \prod \sum x^{ib^r}$$

Divide both sides of the equation by the first term on the righthand side.

$$x^{-b(b^k-1)/(b-1)} + \ldots + x^{-1} + x^0 + x^1 + \ldots + x^{-(b^{k+2}-1)/(b-1)} = \prod_{r=0}^{k} \sum x^{ib^r}$$

For odd k the values of N and M are similarly found.

*NOTE THAT $\sum x^{ib^s} = x^{0b^s} + x^{1b^s} + \ldots + x^{(-b-1)b^s} = \dfrac{x^{(-b-1)(b)^s} + x^{(-b-2)(-b)^s} + \ldots x^{0(-b)^s}}{x^{(-b-1)(-b)^s}} = \dfrac{\sum x^{i(-b)^s}}{x^{(-b-1)(-b)^s}}$

for odd s.

◀

1.7 Theorem Let b be any fixed integer less than -1. Then any integer n has a unique representation (or notation) in base b up to the addition of powers of b with zero coefficients.

PROOF: Let k be the smallest integer such that $N \leq n \leq M$ in lemma 1.6. Since -N and M increase exponentially with k, we know we can find some large k that works; by the well-ordering property for non-negative integers, we know that a smallest one exists.

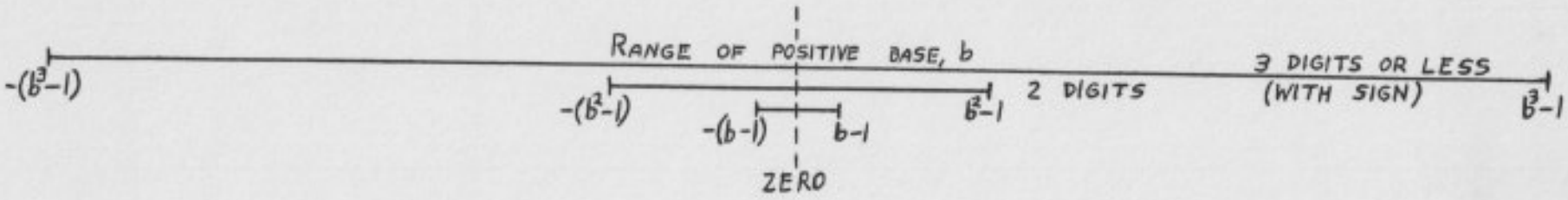
$$(x^0+x^1+\ldots+x^{(-b-1)})(x^0+x^b+\ldots+x^{(-b-1)b})\ldots(x^0+x^{b^k}+\ldots+x^{(-b-1)b^k})$$
$$= x^N + x^{N+1} + \ldots + x^{-1} + x^0 + x^1 + \ldots + x^{M-1} + x^M$$

where N and M are given in lemma 1.6.

The theorem follows from this identity, for notice that in the righthand side the coefficient of each x term is one. Each term is generated by multiplying one term from the first factor on the lefthand side times one from the second factor, etc. Only one such combination can generate a given term on the righthand side (otherwise it would have a coefficient greater than one or two terms would have the same value for exponents). Therefore, the $(-b)^{k+1}$ combinations (terms in the expansion of the L.H.S.) produce the $(-b)^{k+1}$ terms on the R.H.S. Therefore, there is a unique combination of factors (or equivalently, a unique representation) for each x^m or for each m, $N \leq m \leq M$. But -N and M can be arbitrarily large, so the theorem holds for any n.

Observe that the position of zero is not in the middle of the interval bounded by N and M (which are all the numbers that can be written as k+1 or less digits). This is diagramed in Fig. 1.

Table 1 gives specific examples of integer notation.



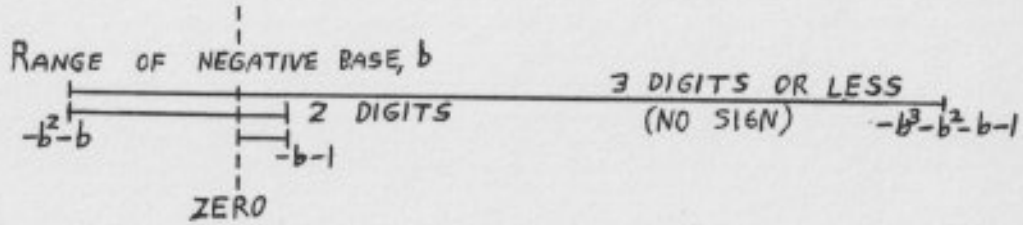
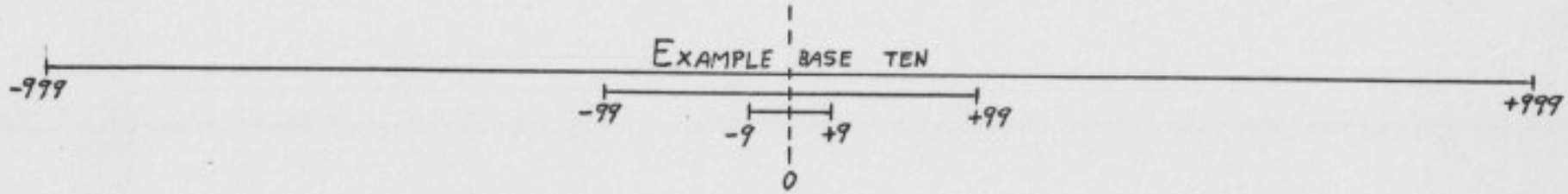
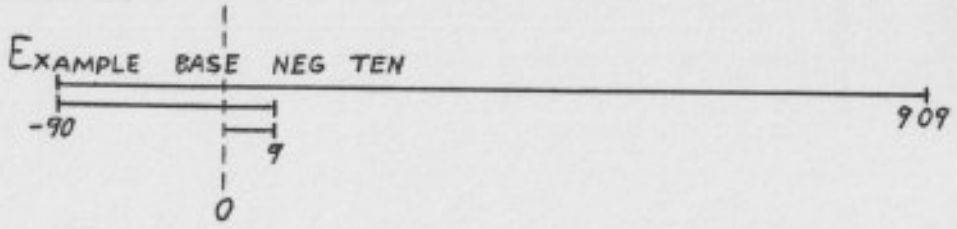


FIG. 1





NOTE: ALL NUMBERS
BASE TEN
NOTATION

Table 1

INTEGER	base TEN	base TWO	base NEG TEN	base NEG TWO	base NEG THREE	base NEG FIVE
TWENTY	20	10100	180	10100	202	110
NINETEEN	19	10011	199	10111	201	124
EIGHTEEN	18	10010	198	10110	200	123
SEVENTEEN	17	10001	197	10001	212	122
SIXTEEN	16	10000	196	10000	211	121
FIFTEEN	15	1111	195	10011	210	120
FOURTEEN	14	1110	194	10010	222	134
THIRTEEN	13	1101	193	11101	221	133
TWELVE	12	1100	192	11100	220	132
ELEVEN	11	1011	191	11111	102	131
TEN	10	1010	190	11110	101	130
NINE	9	1001	9	11001	100	144
EIGHT	8	1000	8	11000	112	143
SEVEN	7	111	7	11011	111	142
SIX	6	110	6	11010	110	141
FIVE	5	101	5	101	122	140
FOUR	4	100	4	100	121	4
THREE	3	11	3	111	120	3
TWO	2	10	2	110	2	2
ONE	1	1	1	1	1	1
ZERO	0	0	0	0	0	0
NEG. ONE	-1	-1	19	11	12	14
NEG. TWO	-2	-10	18	10	11	13
" THREE	-3	-11	17	1101	10	12
" FOUR	-4	-100	16	1100	22	11
" FIVE	-5	-101	15	1111	21	10
" SIX	-6	-110	14	1110	20	24
" SEVEN	-7	-111	13	1001	1202	23
" EIGHT	-8	-1000	12	1000	1201	22
" NINE	-9	-1001	11	1011	1200	21
" TEN	-10	-1010	10	1010	1212	20
" ELEVEN	-11	-1011	29	110101	1211	34
" TWELVE	-12	-1100	28	110100	1210	33
" THIRTEEN	-13	-1101	27	110111	1222	32
" FOURTEEN	-14	-1110	26	110110	1221	31
" FIFTEEN	-15	-1111	25	110001	1220	30
" SIXTEEN	-16	-10000	24	110000	1102	44
" SEVENTEEN	-17	-10001	23	110011	1101	43
" EIGHTEEN	-18	-10010	22	110010	1100	42
" NINETEEN	-19	-10011	21	111101	1112	41
" TWENTY	-20	-10100	20	111100	1111	40

Some more observations are obvious. First (again referring to Fig. 1), it superficially seems that twice as many integers can be represented by a given number of digits in some positive base than in the corresponding negative base. However, the minus/plus sign required by a positive base (but not by a negative base) adds the one extra bit of information responsible for the extra representations. Obviously, (especially when comparing bases +2 and -2) negative radix notation is as efficient as positive radix notation when expressing integers with a given set and number of symbols. This is essentially what the values of N and M imply.

A second observation is recorded below.

1.8 Theorem N is negative if, and only if, there are an even number of digits in the notation of N, base b ($b<-1$).

PROOF: Let $b<-1$ be fixed and let

$N = a_0 + a_1 b + \ldots + a_k b^k$, $a_k \neq 0$, written $a_k \ldots a_1 a_0$.

$N - a_k b^k \leq (-b-1)(1+b+\ldots+b^{k-1}) \leq (-b-1)(|1|+|b|+\ldots+|b^{k-1}|)$

$= (-b-1)(|b^k|-1)/(|b|-1) = |b^k|-1 < a_k|b^k|$ for odd k

$\therefore a_k|b^k| > a_0 + a_1 b + \ldots + a_{k-1} b^{k-1}$. $\therefore$ the sign of N depends on k; vice-versa.

$N<0 \Leftrightarrow a_k b^k < 0 \Leftrightarrow$ k is odd $\Leftrightarrow$ there are an even number of terms.

1.9 Corollary An integer is positive or zero if and only if it has an odd number of digits.

1.10 Theorem The number of nontrivial digits (excludes any preceding left zeros) of a positive or a negative integer increases with its magnitude.

PROOF: For negative bases this follows from application of lemma

1.6. Consider the positive and negative cases separately.

Let $A = a_n \cdots a_1 a_0$ and $C = c_m \cdots c_1 c_0$

Show 1) If $n \geq m+2$, then $|A| > |C|$
2) If $|A| > |C|$, then $n \geq m$.

} A & C both positive or both negative.

Although the structure of negative base notation could be explored further, the properties of the binary operations are more immediately interesting. However, before passage to this subject several remarks about notation and the clarification of some terms are necessary.

First, "long" notations of integers commonly have their digits grouped by threes for easier reading. It will become more apparent why grouping the digits in pairs will be more convenient. (Instead of 547,320,043 write 5 47 32 00 43). Already one can see that this more quickly shows that a number is positive or not. See theorem 1.8. Secondly, in this paper the digits of an arbitrary base b will be written 0,1,...,X,Y, and Z where

$$0 < 1 < \ldots < X < Y < Z = |b| - 1 .$$

Of course, if $\pm b = 2$, then Y is 0 and Z is 1. Occasionally a digit may appear with a bar above it; this signifies the negative of the digit's value. For example, $\overline{1} = 0-1$.

Little distinction is made between "+" and "-" as binary algebraic operations and as the first symbol of the notation of a real number in a positive base system. In this paper whenever + and - appear in a neg. base calculation, interpret them only as operations of addition and subtraction. If -A appears, it will mean 0-A.

Normally when we speak of the sign of a number, we mean that it is positive or negative—corresponding with the "sign" in front of the first digit. Although real numbers do not have such a "sign" in negative bases, it is easiest to retain this terminology in some instances. However, when comparing two numbers it might be more descriptive to say they are on "opposite sides of zero" rather than of "opposite sign."

Another word must be clarified. Although the word decimal is derived from a word meaning ten, in this paper it is most convenient to use it as an adjective describing the "decimal-type" form of positional notation (e.g. 15.342) incorporating the decimal point rather than specifying base ten in particular. This period is called a binary point in base two, but there is no name for it in an arbitrary base b. Therefore, decimal point is the adopted general name.

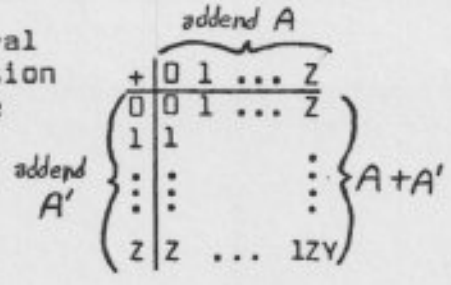
Finally, carry will be used in the normal sense except that the carry will usually be a negative number. (This results from the alternation of sign of the powers of b). Borrow will refer to the corresponding phenomenon in subtraction. The word borrow will be used exclusively in connection with subtraction; carry with addition. The borrow may be of either sign and in this paper is considered the quantity added to the next left column in a subtraction problem.

2. Addition

An algorithm is a systematic method for computation [9]. Computation is generally the manipulation of symbols of numbers in a fixed scheme of numerical notation. The algorithms we wish to explore are methods which determine digitwise the notation for the sums, differences, products, quotients, or roots of numbers. For some types of notations there may exist no algorithm—or at least no simple algorithm—for one or more of these operations. Radix notation, nevertheless, has fairly simple algorithms for such computation and they can be expressed in one of several ways called operation tables. We will see how these may be thought of as a mapping of the operand digits onto the digits of the result. These tables show the digitwise process and are applied to determining the result by an equation which shows the relationship of the digitwise calculations. Naturally, addition will be the first algorithm studied.

We have chosen |b| symbols (digits) for the |b| possible values that the coefficients of the powers of b may take in the representation-polynomial. I will write these as 0,1,...,Y,Z. We know TWO plus THREE equals FIVE, but the porblem is to find for the manipulation of the symbols an algorithm which is entirely similar to the actual structure of the addition of two integers. For one-digit integers we determine an addition table:

general addition table



base four example

+	0	1	2	3
0	0	1	2	3
1	1	2	3	130
2	2	3	130	131
3	3	130	131	132

This table then becomes a part of a larger algorithm which analyzes two multidigit addends digitwise and employs the carry to generate the multidigit sum. In theorem 1.1 the carry was demonstrated and addition was performed where one of the addends was one. A more general proof for arbitrary addends is sought. First I submit a base four example.

<u>2.1 Example</u> Let $A = \sum_{i=0}^{n} a_i(-4)^i = a_n \ldots a_0$ and $A' = \sum_{i=0}^{n} a'_i(-4)^i = a'_{n'} \ldots a'_0$

where a_i, $a'_i = 0,1,2,$ or 3. Suppose $n' < n$; then consider $a'_i = 0$ for $n' < i \leq n$. Then $A + A' = S = \sum_{i=0}^{n''} s_i(-4)^i$, $s_i = 0,1,2,$ or 3.

s_i is given inductively by

$$\left.\begin{aligned} a_1 + a'_1 &= s_1 + c_1(-4) \quad \text{and} \\ a_i + a'_i + c_{i-1} &= s_i + c_i(-4) \end{aligned}\right\} \quad \text{OR} \quad \begin{cases} a_i + a'_i + c_{i-1} = s_i + c_i(-4) \\ \text{and} \quad c_{-1} = 0 \end{cases}$$

Shorter than listing all the combinations of a_i, a'_i, and c_{i-1} and the generated values of s_i and c_i is forming the following addition table matrices:

matrix I (a_i)

$^{0}+_i$	0	1	2	3
0	$^{0}0$	$^{0}1$	$^{0}2$	$^{0}3$
1	$^{0}1$	$^{0}2$	$^{0}3$	$^{0}-0$
2	$^{0}2$	$^{0}3$	$^{0}-0$	$^{0}-1$
3	$^{0}3$	$^{0}-0$	$^{0}-1$	$^{0}-2$

(a'_i ; s_i)

II

$^{-}+_i$	0	1	2	3
0	$^{+}3$	$^{0}0$	$^{0}1$	$^{0}2$
1	$^{0}0$	$^{0}1$	$^{0}2$	$^{0}3$
2	$^{0}1$	$^{0}2$	$^{0}3$	$^{0}-0$
3	$^{0}2$	$^{0}3$	$^{0}-0$	$^{0}-1$

mapping name

III

$^{+}+_i$	0	1	2	3
0	$^{0}1$	$^{0}2$	$^{0}3$	$^{0}-0$
1	$^{0}2$	$^{0}3$	$^{0}-0$	$^{0}-1$
2	$^{0}3$	$^{0}-0$	$^{0}-1$	$^{0}-2$
3	$^{0}-0$	$^{0}-1$	$^{0}-2$	$^{0}-3$

Matrix I represents $a_i + a'_i = s_i$ without a carry (i.e. the carry is zero) from the preceding column (i-1) of addition. II and III represent the cases of -1 and +1 carries. It is easily shown that in 2.1 c_{i-1} has no other values than these. The presuperscripts indicate the value of c_i.

More abstractly, these matrices represent three different mappings of the cartesian product of the set $D_i = \{0,1,2,3\}$ with itself into itself crossed with the set M of the three mappings:

$+_i : D_i \times D_i \longrightarrow D_i \times M_{i+1}$ where $+_i \in M$ and $+_0 = {}^{0}+_0$ and

$D_i = \{0,1,2,3\}$ and $M_j = \{{}^{\circ}+_j, {}^{-}+_j, {}^{*}+_j\}$.

In the addition-table matrices, the number is the element of D_i and its presuperscript indicates which mapping will be used next—when i is incremented. We can alternatively write

$$+_i:(a_i,a_i') = (s_i,+_{i+1}) \quad .$$

This problem will finish specific consideration of base neg. four:

$$\begin{array}{rcl} A & = & {}^{\circ}0\,{}^{-}3\,{}^{+}2\,{}^{-}1 \\ \underline{+A'} & = & \underline{0\,1\,2\,3} \\ S & = & 0\,3\,1\,3\,0 \end{array}$$

2.2 Addition algorithm Let $A = \sum_{i=0}^{n} a_i b^i$ and $A' = \sum_{i=0}^{n'} a_i' b^i$ where $0 \leq a_i < -b$ and $0 \leq a_i' < -b$.

Then $A + A' = S = \sum_{i=0}^{n''} s_i b^i$ where $0 \leq s_i < -b$ and

$$s_i + c_i b = a_i + a_i' + c_{i-1} \ , \quad c_{-1} = 0$$

PROOF: Assume $n \geq n'$; then let $a_i' = 0$ for $n \geq i > n'$.

So $A + A' = \sum_{0}^{n} a_i b^i + \sum_{0}^{n} a_i' b^i$

$$= \sum_{0}^{n} (a_i + a_i') b$$

$$= \sum_{0}^{n} (s_i + c_i b - c_{i-1}) b^i$$

$$= \sum_{0}^{n} s_i b^i + \sum_{0}^{n} c_i b^{i+1} - \sum_{0}^{n} c_{i-1} b^i$$

$$= \sum_{0}^{n} s_i b^i + \sum_{1}^{n+1} c_{i-1} b^i - \sum_{0}^{n} c_{i-1} b^i$$

$$= \sum_{0}^{n} s_i b^i + c_n b^{n+1} - 0$$

$$= \sum_{0}^{n''} s_i b^i \qquad (n'' > n \ \text{IF} \ c_n \neq 0)$$

More specifically, $s_i \equiv a_i + a_i' + c_{i-1}$ (modulo b). Also, $c_i = -1$, 0, or 1. The first statement is obvious; the second follows by induction:

$c_{-1} = 0$ Therefore, $c_0 = -1$ or 0 since

$0 + 0b \leq a_0 + a_0' + 0 \leq Z + Z < \bar{1}Z = Z + (\bar{1})b$

$\therefore c_1 = +1, -1, \text{or } 0$ since $Z + (1)b = -1 \leq a_1 + a_1' + c_0 \leq Z + Z = Y + (\bar{1})b$

Similarly, $c_i = +1, -1, 0$ since $Z + b \leq a_i + a_i' + c_{i-1} \leq Z + Z + 1 = Z - b$

The carry is "automatically" taken care of in the matrix/mapping approach to addition. The mappings for a general base b are shown below. They are used just as the base four mappings.

I

$+_i$	0	1	.	.	.	Y	Z
0	$^{\circ}0$	$^{\circ}1$	.	.	.	$^{\circ}Y$	$^{\circ}Z$
1	$^{\circ}1$	,				$^{\circ}Z$	$\bar{0}$
.	.				$^{\circ}Z$		.
.	.			$^{\circ}Z$			.
.	.		$^{\circ}Z$				.
Y	$^{\circ}Y$	$^{\circ}Z$					$\bar{X}$
Z	$^{\circ}Z$	$\bar{0}$	.	.	.	$\bar{X}$	$\bar{Y}$

(a_i across the top; a_i' down the left; s_i at the right)

mapping name →

II

$-_i$	0	1	.	.	.	Y	Z
0	^{+}Z	$^{\circ}0$	.	.	.	$^{\circ}X$	$^{\circ}Y$
1	$^{\circ}0$	$^{\circ}1$				$^{\circ}Y$	$^{\circ}Z$
.	.				$^{\circ}Y$		$\bar{0}$
.	.			$^{\circ}Y$			.
.	.		$^{\circ}Y$				.
Y	$^{\circ}X$	$^{\circ}Y$					.
Z	$^{\circ}Y$	$^{\circ}Z$	$\bar{0}$	.	.	.	$\bar{X}$

III

$+_i$	0	1	.	.	.	Y	Z
0	$^{\circ}1$	.	.	.	.	$^{\circ}Z$	$\bar{0}$
1	.					$\bar{0}$	$\bar{1}$
.	.				$\bar{0}$		.
.	.			$\bar{0}$			.
.	.		$\bar{0}$				.
Y	$^{\circ}Z$	$\bar{0}$					$\bar{Y}$
Z	$\bar{0}$	$\bar{1}$	.	.	.	$\bar{Y}$	$\bar{Z}$

Note: X, Y, & Z are the three largest digits (if there are that many).

It is noteworthy that positive base addition is basically concerned with positive numbers only. We normally do not directly add a positive and a negative number in calculation. This is particularly a problem with computing machines in which the negative is changed into a complementary or modulo-equivalent positive value before addition can take place. Then the sum must be converted back to standard notation. When manually adding two negatives we essentually disregard the minus signs and add the numerical portions

of the numbers. However, in negative bases we can directly add all numbers regardless of their sign.

Examples of base neg. ten addition:

```
   9 34 52          75 00 02
 + 8 00 71        + 6 70 08
 ---------        ---------
 1 97 33 23       61 71 90
```

Notice the left end of example one. The carry generates <u>two</u> extra digits—thus retaining the correct sign (odd number of digits—positive, in this case) in the sum.

3. Subtraction

The characteristics of negative bases allow two approaches to subtraction. A direct subtraction algorithm may be stated. Secondly, the negative of a number (determined by some algorithm) may be added to the minuend.

3.1 Subtraction algorithm Let A and A' be defined as above, and Then

$A - A' = D = \sum_{0}^{n^*} d_i b^i$, $0 \leq d_i < -b$, where

$d_i + c_i b = a_i - a'_i + c_{i-1}$ and $c_{-1} = 0$.

PROOF: Similar to addition algorithm.

As with addition $c_i = 1, 0,$ or -1. Subtraction too may be stated in the form of a set of mappings:

matrix I

$^{\circ}-_i$	0	1	.	.	.	Y	Z
0	$^{\circ}0$	$^{\circ}1$	.	.	.	$^{\circ}Y$	$^{\circ}Z$
1	^{+}Z	$^{\circ}0$					$^{\circ}Y$
.	^{+}Y		$^{\circ}0$				.
a'_i .	.			$^{\circ}0$			.
.	.				$^{\circ}0$		.
Y	.					$^{\circ}0$	$^{\circ}1$
Z	$^{+}1$	.	.	.	.	^{+}Z	$^{\circ}0$

II

$^{+}-_i$	0	1	.	.	.	Y	Z
0	$^{\circ}1$	$^{\circ}2$	.	.	.	$^{\circ}Z$	$^{-}0$
1	$^{\circ}0$	$^{\circ}1$					$^{\circ}Z$
.	.		$^{\circ}1$				
.	.			$^{\circ}1$			
.	.				$^{\circ}1$		.
Y	.					$^{\circ}1$	.
Z	$^{+}2$	.	.	.	.	$^{\circ}0$	$^{\circ}1$

III

$^{-}-_i$	0	1	.	.	.	Y	Z
0	^{+}Z	$^{\circ}0$	.	.	.	.	$^{\circ}Y$
1	^{+}Y	^{+}Z					.
.	.						.
.	.						.
.	.						.
Y	$^{+}1$					^{+}Z	$^{\circ}0$
Z	$^{+}0$	$^{+}1$	.	.	.	^{+}Y	^{+}Z

Cartesian mapping:

$-_i : (a_i , a'_i) = (d_i , -_{i+1})$

$-_0 = {}^{\circ}-_0$

This subtraction corresponds to positive base subtraction. However, negative bases have an algorithm which is uniquely theirs: negation of a number by direct subtraction from zero. In positive

bases this is accomplished much more simply by changing the sign. Nevertheless, such negatives may not always be as smoothly handled as in negative base calculation.

3.2 Negation algorithm The special case of the subtraction algorithm where $a_i = 0$ for all i. Or A = 0.

In order to subtract two numbers, we can apply this algorithm to the subtrahend and then add the result to the minuend. Of course, all algorithm 3.2 amounts to is subtracting the given number from zero—unheard of in positive bases where we multiply by -1 instead. A borrow of $c_i = 1$ is always generated except in two cases: 1) when a zero digit in the given number is encountered or 2) when a one or zero is encountered and a borrow has been generated by the previous column right. Then $c_i = 0$.

The mapping method of stating this algorithm shows its simplicity. It consists of the first columns of the subtraction mapping-matrices, since $a_i = 0$. However, since $\bar{-}_i$ is never generated, its matrix may be discarded.

$\bar{-}_i$ | $a_i = 0$
0 | $^{c}0$
1 | ^{+}Z
⋮ | ⋮
Y | $^{+}2$
Z | $^{+}1$

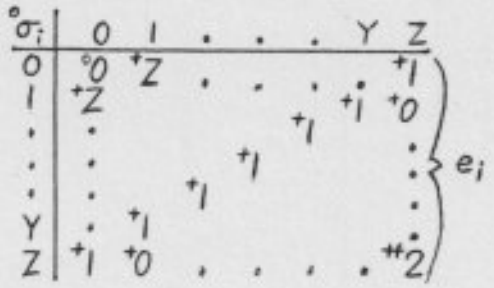
$^{+}\bar{-}_i$ | $a_i = 0$
0 | $^{c}1$
1 | $^{c}0$
⋮ | ⋮
Y | ^{+}Z
Z | $^{+}2$

$$D - A' = N \quad \text{where}$$

$$A' = \sum_{c}^{m} a_i' b^i$$

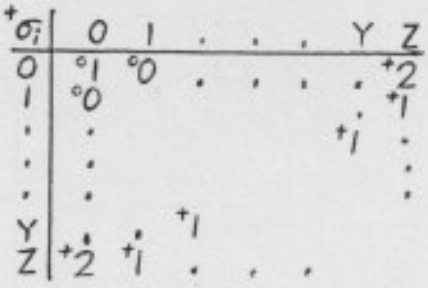
$$\text{and } N = \sum_{c}^{m+l'} n_i b^i$$

The foregoing suggests examination of the algorithm for D - A - A'. We have no name or symbol for such an operation. I will call it negative addition. The symbol for the mapping-matrices will be σ. These matrices are listed at the top of the next page.

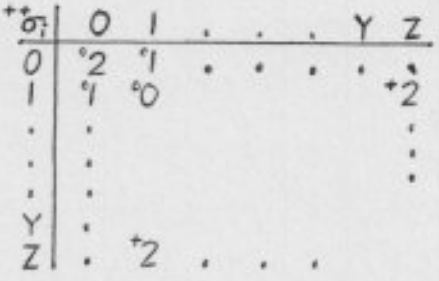


$^{\circ}\sigma_i$	0	1	.	.	.	Y	Z
0	$^{\circ}0$	^{+}Z	.	.	.	.	$^{+}1$
1	^{+}Z	.	.	.	.	$^{+}1$	$^{+}0$
.	.	.	.	$^{+}1$	.	.	.
.	.	.	$^{+}1$	.	.	.	.
.	.	$^{+}1$	.	.	.	.	.
Y	.	$^{+}1$	.	.	.	.	.
Z	$^{+}1$	$^{+}0$	.	.	.	.	$^{+}2$

e_i



$^{+}\sigma_i$	0	1	.	.	.	Y	Z
0	$^{\circ}1$	$^{\circ}0$	.	.	.	.	$^{+}2$
1	$^{\circ}0$	.	.	.	.	$^{+}1$	.
.	.	.	.	.	$^{+}1$	.	.
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.
Y	.	.	$^{+}1$	.	.	.	.
Z	$^{+}2$	$^{+}1$	.	.	.	.	.



$^{++}\sigma_i$	0	1	.	.	.	Y	Z
0	$^{\circ}2$	$^{\circ}1$	.	.	.	.	.
1	$^{\circ}1$	$^{\circ}0$	.	.	.	.	$^{+}2$
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.
Y	.	.	.	.	.	.	.
Z	.	$^{+}2$	.	.	.	.	.

$\sigma_i : (a_i, a_i') = (e_i, \sigma_{i+1})$

$\sigma_0 = {}^{\circ}\sigma_0$

3.3 Negative addition algorithm Let A and A' be defined as before. Then $0 - A - A' = E = \sum_{0}^{n''} e_i b^i$, $0 \leq e_i < -b$, where

$e_i + c_i b = -a_i - a_i' + c_{i-1}$ and $c_{-1} = 0$.

PROOF: Like the similar theorems before.

Some observations need to be recorded. First, $c_i = 0, 1,$ or 2 for the general base $b < -2$. This could be demonstrated easily by induction. However, if $Z = 1$ (i.e. b = neg. two), $c_i = 0$ or 1 only. This can easily be seen in the matrices for the mapping interpretation for this algorithm. (Of course, $Y = 0$, $Z = 1$, and $b = -2$). In other words, the algorithm for $-A - A'$ in base neg. two has a simpler borrow, since $c_i = 0$ or 1 rather than $c_i = -1, 0,$ or 1. This has already been noted by [2].

Also, this algorithm could easily be used as the fundamental operation in a computing machine (base neg. two would be especially nice). A special case of this algorithm is negation: $-0 - A'$.

Therefore, by applying the algorithm once and its special case once (not necessarily in that order), we obtain a substitute for the other algorithms:

$$A + A' = -(-A-A')$$

$$A - A' = -(-A) - A'$$

$$A' - A = -A - (-A')$$

4. Multiplication

4.1 Multiplication algorithm I Let $M = \sum_{i=u}^{v} m_i b^i$, and $N = n_0$.

Then $MN = P = \sum_{i=u}^{v''} p_i b^i$, $0 \leq p_i < b$, where p_i by induction is defined to be

$$\left.\begin{aligned} &1)\ p_u \equiv m_u n_0 \pmod b \\ &\quad c_u = (m_u n_0 - p_0)/b \\ &2)\ p_i \equiv m_i n_0 + c_{i-1} \pmod b \\ &\quad c_i = (m_i n_0 + c_{i-1} - p_i)/b \end{aligned}\right\} \quad \text{OR} \quad \begin{aligned} &m_i n_0 + c_{i-1} = p_i + c_i b \\ &c_{u-1} = 0 \end{aligned}$$

PROOF: Show that this definition of p_i produces the correct product $\sum p_i b^i = MN$. Note that setting $c_{u-1} = 0$ allows us to substitute for all four equations above the equivalent equation

$$m_i n_0 + c_{i-1} = p_i + c_i b \ .$$

$$\begin{aligned} MN = n_0 \sum_u^v m_i b^i &= \sum_u^v m_i n_0 b^i \\ &= \sum_u^v (p_i + b c_i - c_{i-1}) b^i \\ &= \sum_u^v p_i b^i + \sum_u^v c_i b^{i+1} - \sum_u^v c_{i-1} b^i \\ &= \sum_u^v p_i b^i + \sum_{u+1}^{v+1} c_{i-1} b^i - \sum_u^v c_{i-1} b^i \\ &= \sum_u^v p_i b^i + c_v b^{v+1} \end{aligned}$$

If $c_v = 0$, $MN = \sum_u^v p_i b^i$

If $b < c_v < 0$, $MN = \sum_u^v p_i b^i + p_{v+1} b^{v+1} + c_{v+1} b^{v+2}$, $c_{v+1} = 1$

$$= \sum_u^{v+2} p_i b^i \ , \quad p_{v+2} = c_{v+1} = 1$$

By induction show that the other cases for c_v do not occur. First, note that $0 \leq m_i n_0 < b^2$

I) $c_{u-1} = 0$. Therefore from the algorithm, c_u has bounds:

$$0 = (0 - 0)/b \leq c_u < (b^2 - 0)/b = b$$

II) Assume $0 = c_{i-1} < b$; then c_i similarly is

$0 = (0-0)/b \leq c_i < (b^2 + b - 1)/b = b$ by the algorithm.

4.2 Multiplication algorithm II Let $M = \sum_{i=u}^{v} m_i b^i$ and $N = \sum_{j=u'}^{v'} n_j b^j$.

Then $MN = P = \sum_{u+u'}^{v''} p_k b^k$, $0 = \begin{pmatrix} m_i \\ n_j \\ p_k \end{pmatrix} < b$, where inductively $\begin{pmatrix} u \leq v \\ u' \leq v' \end{pmatrix}$

$$\left.\begin{aligned} &1)\quad p_{u+u'} = m_u n_{u'} \pmod b \\ &\quad\ c_{u+u'} = (m_u n_{u'} - p_{u+u'})/b \\ &2)\quad p_k = \sum_{h=u}^{v} m_h n_{k-h} + c_{k-1} \pmod b \\ &\quad\ c_k = \sum_{h=u}^{v} m_h n_{k-h} + c_{k-1} - p_k\,/b \end{aligned}\right\} \quad \text{OR} \quad \begin{aligned} &\sum_{h=u}^{v} m_h n_{k-h} + c_{k-1} = p_k + c_k b \\ &c_{u+u'-1} = 0 \end{aligned}$$

PROOF: $MN = \left(\sum_{i=u}^{v} m_i b^i\right)\left(\sum_{j=u'}^{v'} n_j b^j\right) = \sum_{u+u'}^{v+v'} \left(b^k \sum_{h=u}^{k} m_h n_{k-h}\right)$

$$= \sum_{u+u'}^{v+v'} b^k (p_k + c_k b - c_{k-1})$$

$$= \sum_{u+u'}^{v+v'} b^k p_k + \sum_{u+u'}^{v+v'} c_k b^{k+1} - \sum_{u+u'}^{v+v'} c_{k-1} b^k$$

$$= \sum_{u+u'}^{v+v'} b^k p_k + c_{v+v'} b^{v+v'+1} - 0$$

$$= \sum_{u+u'}^{v+v'} p_k b^k \quad \text{IF } c_{v+v'} = 0 \qquad (p_k \text{ may be zero})$$

$$= \sum_{u+u'}^{v''} p_k b^k \quad \text{IF } c_{v+v'} \neq 0 \qquad v'' \geq v+v'+2$$

Actually, the product MN is calculable directly from the expression $MN = (\sum m_i b^i)(\sum n_j b^j)$. Each n_j in the second summation is multiplied with the first summation and with b^j. (Algorithm 4.1) All of the sub-products are then added to get the product. We will note this mechanically just as in a positive base:

$$\begin{array}{rl} & m_v \ldots m_{c\bullet} m_{\bullet} \ldots m_u \\ \times & \qquad n_{v'} \ldots n_{0\bullet} \ldots n_{u'} \\ \hline & s_{v+v'} \ldots s_{0\bullet} \quad \ldots s_{u+u'} \\ + & \quad \ddots \quad \ddots \quad \ddots \\ & s'_{v+v'} \ldots s'_{0\bullet} \quad \ldots s'_{u+u'} \\ \hline & p_{v''} \ldots \qquad p_{0\bullet} \qquad \ldots p_{u+u'} \end{array}$$

4.3 Lemma $Z^2 = (-b-1)^2 = 121$ in all negative bases less than neg. two, and $Z^2 = 1^2 = 1$ in base neg. two.

PROOF: $Z = -b-1$ so $Z^2 = (-b-1)^2 = b^2 + 2b + 1 = 121 = \overline{Y}1$

For bases other than neg. two. If $Z = 1$, then $Z^2 = 1$.

This and the next two lemmas are steps to the theorem which sets the number of digits in the product of two integers.

4.4 Lemma The k digit integer with the smallest magnitude (1ZOZO...ZO if positive; 1ZOZOZ...OZ if negative) times Z always gives a k digit product. In fact, for odd k it is 122...2210 if b -2 or 11010...10 if b = -2. If k is even the products are 122...221 or 110101...01 .

PROOF: Trivial with b = -2; Z = 1.

General case:

$$\begin{array}{r} 1ZOZO...ZO \\ \times\ \ Z \\ \hline Z1....1010 \\ +\ \ 2....202\ \} \\ +\ \ 1....101\ \} \text{ carries (see lemma 4.3)} \\ \hline 12...22210 \end{array}$$

This is the positive case; the negative case is similar.

4.5 Lemma The k digit integer with the largest magnitude (ZOZOZ...OZ if positive; ZOZO...ZO if negative) times Z always gives a k + 2 digit product: 1222...21 if b -2 and the integer is positive
122...210 if b -2 and the integer is negative

except if b = -2; then the product has k digits and equals

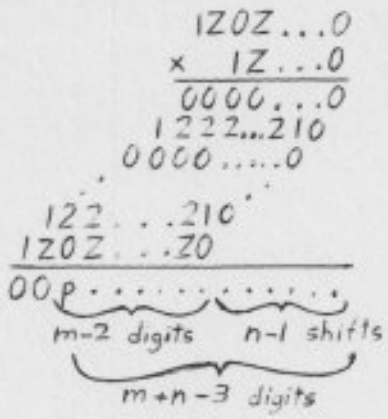
10101O1...01 if the integer is positive
101010....10 if the integer is negative.

PROOF: Much like the above lemma.

4.6 Theorem The product of an m digit integer and an n digit integer is written with at most m+n+1 digits and at least m+n-3 digits.

PROOF: The number of digits in an integer increases monotonically with its magnitude. Focus on the positive cases:

1) Smallest m digit integer times smallest n digit yields smallest product (a lower bound) of m times n digit numbers:

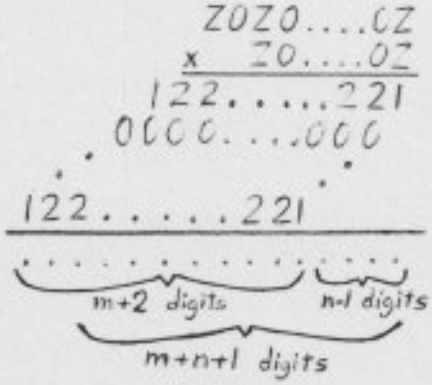


Corresponding calculations for each case with a negative integer involved can be derived from a positive case by division or multiplication by b (i.e. by 10).

1a) If the multiplier is negative, it will terminate with a Z rather than a 0. This case is the same as the one shown except that the terminal zero would be removed from the multiplier (eliminating the first zero partial product) leaving n-1 digits. The product would also have one less digit, so the theorem holds.

1b) If the multiplicand is negative, simply remove the last zero from each partial product and you would have the calculation demonstrated for this case. As in 2), the theorem holds.

2) Below is shown the calculation for the largest m digit times the largest n digit integers which give the larges product of such integers.



* Where b = -2 the partial products have two less digits; but so does the product. Therefore, the theorem holds also for b = -2.

2a) If the multiplier is negative the case is like the above calculation except that a terminal 0 should be added to the multiplier. The result is simply the addition of a 0 to the above product, also.

2b) If the multiplicand is negative, a 0 ends the integer. So change the above calculation by adding a zero to the multiplicand (and therefore each partial product) and the product is the same except for an additional 0.

4.7 Definition The order of magnitude of a real number $A = \sum_{n}^{-\infty} a_i b^i$, $a_n \neq 0$, is b^n. Note: later it will be possible to show that $\frac{1}{1-b}(-b)^n \leq A(-1)^h \leq \frac{-b}{1-b}(-b)^{n+1}$. Hereafter in this paper the order of magnitude of a number will simply be referred to as its order.

4.8 Restatement of 4.6 The product of integers N and M, which respectively have orders of b^n and b^m, is of order b^{m+n+2}, b^{m+n}, or b^{m+n-2}.

PROOF: N and M are n+1 and m+1 digit integers respectively. Apply the preceding theorem. NM has between m+n+3 and m+n-1 digits (inclusive); NM has order between b^{m+n+2} and b^{n+m-2} inclusive. Combine theorems 1.8 and 1.9 with the algebraic theorem that two positive (negative) integers have a positive product, etc. Therefore, two of the values in the above range imply a contradiction—the product would have the wrong sign.

Note: This proof does not exclude rational numbers, which will be studied in chapter 6.

5. Congruences

Before passing on to the related subjects of division and the notation of rational numbers, I will consider some of the residue or congruence properties of the foregoing integer notation. An interesting algorithm for checking calculation in a positive base B is based on congruences modulo B-1. It is commonly known as "casting out nines" in base ten.

If $e \equiv f \pmod{d}$ then $P(e) \equiv P(f) \pmod{d}$ where P is a polynomial with integer coefficients. But P(b) is the representation of some integer N in base b if the coefficients are non-negative and less than $|b|$. Also, P(1) is the sum of the digits of N. By the above $P(1) \equiv P(b) \pmod{m}$ if $b \equiv 1 \pmod{m}$. Although m could be any divisor of b-1, b-1 itself is the best choice for m when the above congruence is used for checking purposes.

Suppose $N(b) \times M(b) = P(b)$, where these polynomials are representations of integers. If P is correct, the summation of the digits in N times the summation of the digits in M is equivalent modulo b-1 to the summation of the digits of P. That is,

$N(b) \equiv N(1)$; $M(b) \equiv M(1)$; $P(b) \equiv P(1)$. Therefore,

$N(1)M(1) \equiv N(b)M(b) \equiv P(b) \equiv P(1)$, all mod b-1.

Similarly addition, subtraction, division, and raising integers to some power may be checked.

The probability for finding an error in a such a problem is b/(b-1) —assuming this check algorithm itself is computed correctly! This type of checking gives better results in a negative base than in a positive base. For example, in base ten this checking procedure operates mod 9, but in base neg. ten it operates mod ±11 and would

be called "casting out elevens." Actually there is a modulo 11 check in base ten: $10 \equiv -1 \pmod{11}$ so $P(-1) \equiv P(10) \pmod{11}$. $P(-1)$, in words, means adding every other digit and subtracting the ones in between (this "automatically" occurs in negative bases because of the alternation of sign between columns). In base ±10 casting out 11's has an advantage over casting out nines, because 10/11 and 8/9 of the mistakes are respectively found. $10/11 > 8/9$. Since casting out ones is meaningless, computers operating with base two cannot use such a check. However, casting out threes might be an advantageous practical check on calculation and neg. two rather than two has a simpler casting-out-threes algorithm.

As a conclusion the above is formally stated:

5.1 Checking algorithm Let $P(x)$ and $Q(x)$ be polynomials with non-negative integer coefficients less than $|b|$. Then for $m|b-1$,

$$P(b) + Q(b) \equiv P(1) + Q(1) \pmod{m}$$

$$P(b) - Q(b) \equiv P(1) - Q(1)$$

$$P(b)Q(b) \equiv P(1)Q(1)$$

$$P^2(b) \equiv P^2(1) \qquad \text{etc.}$$

In a positive base (e.g. ten) the notation is symmetric about zero except for the sign. See also figure 1.

100 ... -10 ... -3 -2 -1 | 0 | 1 2 3 ... 10 ... 100

line of symmetry or reflection

As $N = -\sum a_i 8^i$ 0 increases, a_i "counts backward" mod 8; but as N increases through zero, the direction of the counting reverses. See table 1. This reversal does not occur in negative bases. a_i always increases (mod -b) as N increases if i is even; it decreases for odd i.

This property makes congruences between positive and negative integers more obvious—especially modulo b. For example, $29 \equiv 19 \equiv 9 \equiv 199 \equiv 189$ (mod 10 or 190) in base neg. ten, but $-11 \equiv -1 \equiv 9 \equiv 19$ (mod 10) in base ten. For a modulus $\neq b$, in base ten $-21 \equiv -16 \equiv -11 \equiv -6 \equiv -1 \equiv 4 \equiv 9 \equiv 14 \equiv 19 \equiv 24 \equiv 29$ (mod 5), but in base neg. $39 \equiv 24 \equiv 29 \equiv 14 \equiv 19 \equiv 4 \equiv 9 \equiv 194 \equiv 199 \equiv 184 \equiv 189$ (mod 5). In the second set of congruences the succession (4,9) is clearly repeated in the last digit for both positive and negative integers. For the positive base the last digits of the positive equivalents are different from the negative ones. Also, in base three ... $-202 \equiv -111 \equiv -20 \equiv 1 \equiv 22 \equiv 120 \equiv 211 \equiv 1002 \equiv 1100$ ... (mod seven), but ... $1000 \equiv 1111 \equiv 1222 \equiv 20 \equiv 1 \equiv 112 \equiv 210 \equiv 12021$ ... (mod seven) in base neg. three. Here the principle is a little less obvious but still holds. Notice that for base three the succession of last digits for the positives (1,2,0) is reversed for the negatives. However, for the base neg. three congruence the 1,2,0 succession is the same for both positive and negative intergers.

No doubt, the other digit positions could be similarly studied.

6. Reals and rationals

So far little has been mentioned about real and rational numbers. It would be good now to study the properties of negative-base decimals.

6.1 Theorem Every real number, α, is representable as

$$\alpha = A + \sum_{-7}^{-\infty} c_i b^i$$ where α is a real number, A is an integer (base b), b is a fixed integer < -1, c_i is an integer $0 \leq c_i < -b$

and, conversely, each such representation is a real number.

PROOF: Given a real number, α, we can construct a sequence of expansions in b,

$e_i = A + c_{-1}b^{-1} + c_{-2}b^{-2} + \ldots + c_i b^i$, $i = -1,-2,-3,\ldots$, such that $\lim_{i \to -\infty} |e_i - \alpha| = 0$. This continues in the same way as when showing the infinite decimal representations of real numbers in a positive base. Simply show that $|e_i - \alpha| \leq (-b)^i$ in a standard " -type" proof.

CONVERSE: Let the above expression be given. Consider the $i^{\underline{th}}$ interval with center $A + c_{-1}b^{-1} + \ldots + c_i b^i$ and with radius $(-b)^i$. Interval i+1 is contained in interval i, since

$(A+\ldots+c_i b^i) - b \leq A+\ldots+c_i b^i + c_{i+1}b^{i+1}\ b^{i+1} \leq (A+\ldots+c_i b^i) + b$.

The length of the i^{th} interval tends to zero as i increases. By the nested interval theorem the limit of these intervals is a single real number.

No more theorems concerning real numbers with infinite aperiodic representations will be treated in any detail since such representations can never actually be written and therefore fall outside the

subject of radix-representation number systems. So this paper is naturally limited to the subject of rational numbers expressed as finite expansions of b and to their quotients (including repeating infinite expansions).

6.2 Theorem [2] If N is representable as a finite number of positive and negative powers of b, it has a unique representation.

PROOF: Suppose $N = \sum_{m}^{n} a_i b^i$ and suppose $N = \sum_{r}^{s} a_i' b^i$ also. Of course, $0 \leq a_i < -b$ and $0 \leq a_i' < -b$. Let $p = \min(n,s)$.

$b^p N$ is an integer

$$b^p N = a_m b^{m+p} + \ldots + a_n b^{p+n}, \quad p+n \geq 0$$
$$= a_r' b^{r+p} + \ldots + a_s' b^{p+s}, \quad p+s \geq 0$$

But $b^p N$ has a unique representation by an earlier theorem, so $m = r$, $n = s$, and $a_i = a_i'$.

6.3 Definition As with integers in an earlier section, these expressions for rational numbers will be written in the convetional "shorthand" of decimal notation:

If and only if $A = \sum_{m}^{n} a_i b^i$, then A is written as $a_m a_{m-1} \ldots a_0 . a_{-1} \ldots a_n$ where $n \leq 0$ and where juztaposition does not indicate multiplication. a_i is the symbol for one of the integers $0, 1, \ldots, Z$.

6.4 Theorem The addition, subtraction, and multiplication algorithms hold for rationals. (Passage from integers to rationals just as in positive base systems).

6.5 Theorem If and only if a number in base b notation is periodic, it is a rational number.

PROOF: Assume A, a real number, repeats.

$A = D + .a_1 a_2 \ldots a_n c_1 \ldots c_m \ldots$ where the period is m and D is an integer (base b). Multiply A by b^m and subtract it

from A:

$$\begin{array}{rcll} A & = & D + & .a_1a_2\ldots a_n c_1\ldots c_m c_1\ldots c_m\ldots \\ Ab^m & = & Db^m + & a_1a_2\ldots .\ldots\ldots c_m c_1\ldots c_m c_1\ldots c_m\ldots \\ \hline A(1-b^m) & = & D(1-b^m)+ & d_1\ldots d_s .\ldots\ldots d_{m+n} \end{array}$$

$$\therefore A = D + \frac{d_1\ldots d_s\ldots d_{m+n}}{(1-b^m)b^{m+n-s}}$$

$=$ a rational number.

CONVERSE: Assume A is a rational number. Show that it repeats.

Let $A = A_0 + \frac{r}{s}$ where A_0 is an integer, $\frac{r}{s}<1$, and r and s are relatively prime. Since r and s are relatively prime, by Euler's theorem

$b^{\phi(s)} \equiv 1 \pmod{s}$ where $\phi(s)$ is the Euler phi function.

$rb^{\phi(s)} \equiv r$

$rb^{\phi(s)} - r = ks$ for some integer k

$\frac{r}{s}b^{\phi(s)} - \frac{r}{s} = k$, an integer

$\frac{r}{s}$ with the decimal point shifted $\phi(s)$ places to the right

$$\frac{r}{s} = .a_1a_2\cdots a_{\phi(s)}a_{\phi(s)+1}\cdots a_{2\phi(s)}\cdots$$

Therefore, $a_1 = a_{\phi(s)+1}$

$a_2 = a_{\phi(s)+2}$

$\vdots$

$a_i = a_{\phi(s)+i} = a_{n\phi(s)+i}$

We see a period of repetition, $\phi(s)$, which may not necessarily be the smallest period.

It is interesting to note that the largest and smallest numbers expressible to the right of the decimal point (with negative powers of b) are rational. This is part of the significance of the following theorem.

6.6 Theorem $\frac{b}{1-b} \leq .a_1a_2a_3a_4\ldots \leq \frac{1}{1-b}$

PROOF: $A = \sum_{-1}^{-\infty} a_i b^i$. Now $\sum_{-1}^{-\infty} a_i b^i$ converges absolutely; therefore, we can rearrange the terms of this series. So let

$A_N = a_{-1}b^{-1} + a_{-3}b^{-3} + a_{-5}b^{-5} + \ldots \leq 0$ and

$A_P = a_{-2}b^{-2} + a_{-4}b^{-4} + a_{-6}b^{-6} + \ldots \geq 0$.

$A = A_N + A_P$. Therefore, A is greatest when $A_N = 0$ and A_P takes on its greatest value; A is least when $A_P = 0$ and A_N takes on its least value. A_P and A_N equal zero when $a_i = 0$ (for all even and odd i, respectively). A_P is greatest when $a_i = Z$; A_N is least when $a_i = Z$. Therefore, $.0Z0Z0Z\ldots \geq A \geq .Z0Z0Z0\ldots$

By subtraction as in theorem 6.5 these two bounds may be evaluated. $.0Z0Z0Z\ldots = 1/(1-b)$ and $.Z0Z0Z0\ldots = b/(1-b)$.

6.7 Theorem All rational numbers, α, of the form

$\alpha = \pm b^k(C + \frac{b}{1-b}) = \pm b^k(C-1 + \frac{1}{1-b})$ * where C, k are integers have two (repeating decimal) representations in base b, $b \leq -1$.

Note: This is analogous to $N + 1.000\ldots = N + .ZZZ\ldots$ in a positive base, where N is any integer.

PROOF: Since multiplication by b^k only accomplishes a shift of the decimal point, consider only the expression in the parentheses.

* I believe that this is the correct formula and that the formula listed in [2] is a misprint.

Let $C = c_m c_{m-1} \cdots c_1 c_0$ and $C - 1 = c'_m c'_{m-1} \cdots c'_1 c'_0$; $c_0 \neq c'_0$.

Therefore, $C + \frac{b}{1-b} = c_m \cdots c_1 c_0 . ZOZOZO \ldots$

and $C - 1 + \frac{1}{1-b} = c'_m \cdots c'_1 c'_0 . OZOZOZ \ldots$

Algebraically the left sides of the above two equalities are equal. Therefore, the <u>two different representations</u> are equal mathematically, but not symbolically.

<u>6.8 Theorem</u> All rational numbers not of the above form have a unique notation in base b.

PROOF: All finite representations (notations) are unique by a previous theorem. Consider any infinite (periodic) decimal; assume it has two representations. We can write these representations as

$$\alpha = C + .a_1 a_2 a_3 \cdots$$
$$= C' + .a'_1 a'_2 a'_3 \cdots$$

where C and C' are integers.

$$\alpha - \alpha = C - C' + 0$$
$$\text{or } C - C' \pm 1$$

i.e. $.a_1 a_2 a_3 \cdots - .a'_1 a'_2 a'_3 \cdots = 0 \text{ or } \pm 1$

This is because the difference between two integers must be an integer. The difference cannot have a magnitude greater than one, since the decimal (fractional) parts cannot add up to a integer value with an absolute value greater than one. However, only numbers of the form mentioned in 6.7 can yield a difference of 1, $(\frac{1}{1-b} - \frac{b}{1-b})$, because if $\frac{b}{1-b} < .a_1 a_2 a_3 \cdots < \frac{1}{1-b}$ and similarly for $.a'_1 a'_2 a'_3 \cdots$, then obviously their difference must be less than one and greater than negative one. If their difference is zero, if they are equal, then their representations must be alike digit for digit, i.e. $a_i = a'_i$, because of the following argument.

Let s be any subscript where $a_s \neq a'_s$. Now then by multiplying by a power of b (shifting the decimal point right),

$a_1 \cdots a_s \cdot a_{s+1} \cdots = a'_1 \cdots a'_s \cdot a'_{s+1}$

By the preceding argument $a_s - a'_s = \pm 1$, since $a_s \neq a'_s$. Also, one of $\cdot a_{s+1} \cdots$ and $\cdot a'_{s+1} \cdots$ must be $\cdot$ZOZOZO... and the other must be $\cdot$OZOZOZ... But then these numbers are of the form stated in 6.7.

By contradiction it has been demonstrated that any numbers with two representations must be of the form in 6.7.

Note: This converse to 6.7 also shows that all irrational numbers have a unique representation.

An earlier theorem demonstrated that if a rational (or real) number, α, is of the form $\cdot a_1 a_2 a_3 \cdots$, then $\frac{b}{1-b} \leq \alpha \leq \frac{1}{1-b}$. Now the converse is stated and proved.

6.9 Theorem If a rational (or real) number is between $\frac{b}{1-b}$ and $\frac{1}{1-b}$, then all the digits appear to the right of the decimal point (α is completely representable with negative powers of b).

PROOF: By way of contradiction suppose $\alpha = N \cdot a_1 a_2 a_3 \cdots$ where N is any non-zero integer in base b notation preceding the decimal. Show $\alpha \leq \frac{b}{1-b}$ or $\alpha \geq \frac{1}{1-b}$.

$\frac{b}{1-b} \leq \cdot a_1 a_2 a_3 \cdots \leq \frac{1}{1-b}$

$N \neq 0$, so consider $|N| = 1$: $\quad 1 + \frac{b}{1-b} = \frac{1}{1-b} \leq \alpha \leq 1 + \frac{1}{1-b}$ if N=1.

Similarly, if $N = 1Z$, $\alpha = b/(1-b)$.

If $|N| > 1$, then obviously $|\alpha| > 1$

Therefore, N must equal zero, so all α between (but excluding) the stated limits have zeros to the left of the decimal point.

7. Division

Standard division algorithm Let n and d be integers. Then there exist unique integers q and r such that $n = dq + r$, where $0 \leq r < d$.

7.1 Modified division algorithm Let n and d be integers. Then there exist unique integers such that $n = dq + r$, or equivalently

$$\frac{n}{d} = q + \frac{r}{d}, \quad \text{where } \frac{b}{1-b} \leq \frac{r}{d} < \frac{1}{1-b}.$$

Note; For the purposes of this paper, the latter statement is of more value since we are interested in division in the rational number field. The former form is required in (integer) number theory.

PROOF: More generally we could show that unique q and r exist as long as the range of r is restricted to d consecutive integers: $0 + c < r \leq d + c$.

This is derived from the normal form of the division algorithm. One might well ask why the limits on r were modified from the usual. Given n/d, we would like to express this rational number in base b decimal-point notation. q essentially represents all digits to the left of the decimal point; r/d represents all digits to the right of the decimal point. Therefore, r/d in neg base notation is limited to the values stated in theorem 6.6.

7.2 Corollary: n and d are given. There exist unique integers q and r such that $\frac{n}{d} = q + \frac{r}{d}$ where $\frac{b}{1-b} \leq \frac{r}{d} \leq \frac{1}{1-b}$ except where $\frac{r}{d} = \frac{1}{1-b}$ or $\frac{b}{1-b}$. Then there are two values for q and r:

$$\frac{n}{d} = q + \frac{1}{1-b} = (q+1) + \frac{b}{1-b} \quad \text{or similarly} \quad \frac{n}{d} = q' + \frac{b}{1-b} = (q'-1) + \frac{1}{1-b}.$$

PROOF: With the preceding division algorithm is combined the obser-

vation that $\frac{1}{1-b} = 1 + \frac{b}{1-b}$.

Note: $\frac{b}{1-b} = \frac{10}{1Z1}$; $\frac{1}{1-b} = \frac{1}{1Z1}$. Therefore,

$1Z.0Z0Z\ldots = .Z0Z0\ldots = 10/1Z1 \leq r/d \leq 1/1Z1 = .0Z0Z\ldots = 1.Z0Z0\ldots$

Some of the implications of this are not immediately obvious. The modified division algorithm changes the mechanical form of long division to some extent. (It makes long division seemingly more complicated than in a positive base, but some of the seeming complexity may be due only to the strangeness of negative base notation—in particular the multiplication tables.)

In positive-base long division the largest possible multiple of the divisor times a power of ten is subtracted from the dividend to leave a positive remainder less than the divisor times that power of ten. This then is repeated again and again with the remainders instead of the original dividend until a subsequent remainder is zero, until the decimal point is reached, or until an arbitrary place is reached. In negative-base long division the intent is not necessarily to produce positive remainders, for they must fall in the range determined by algorithm 7.2. In fact, the progressive remainders will usually be negative when the divisor is positive.

<u>7.3 Long-division algorithm</u> Let the divisor (or denominator) D and the dividend (or numerator) N (both integers) be given. Then the quotient $Q = N/D = \sum_{i=p}^{-\infty} q_i b^i$ is obtained by induction:

1) q_p, such that $r_{p-1} = N - Dq_p b^p$

2) q_{p-i}, such that $r_{p-i-1} = r_{p-i} - Dq_{p-i} b^{p-i}$

where integer $i = 0,1,2,\ldots$ and $\frac{b}{1-b} \leq r_i/Db^i \leq \frac{1}{1-b}$.

Note: $N = r_p$

PROOF: Show that $q_p q_{p-1}\cdots = Q = N/D$

By 1), $N = Dq_p b^p + r_{p-1}$ where $\frac{b}{1-b} = r_i / Db^i = \frac{1}{1-b}$

$\frac{N}{D} = q_p b^p + r_{p-1}/D$

By 2) $\frac{r_{p-1}}{D} = \frac{r_{p-2}}{D} + q_{p-1} b^{p-1}$

$\therefore \frac{N}{D} = q_p b^p + q_{p-1} b^{p-1} + \frac{r_{p-2}}{D}$

$= q_p b^p + q_{p-1} b^{p-1} + q_{p-2} b^{p-2} + \frac{r_{p-3}}{D}$

$\vdots$

$= q_p b^p + \ldots + q_0 b^0 + \frac{r_{-1}}{D}$

$\vdots$

$= q_p b^p + \ldots + q_{p-i} b^{p-i} + \frac{r_{p-i-1}}{D}$

$\vdots$

etc.

Until we have $N/D = q_p q_{p-1} \cdots q_0 \cdot q_{-1} \cdots$ for as many digits as we wish to check.

Suppose, however, that D and N are rationals and we wish to write their quotient. Let $D = r/s$ and $N = t/u$, r,s,t,u are integers. Therefore, $Q = N/D = \frac{t/u}{r/s} = \frac{st}{ru}$. We can then apply the above long division algorithm to obtain the representation of Q in base b. If s is a power of b (which is possible if D is a finite decimal), then $Q = N/D = Ns/Ds = Ns/r$. The divisor is now an integer; this amounts to shifting the decimal point equal number of places in both the denominator-divisor and the numerator-dividend to make the divisor (and dividend, if desired) an integer. Reversing this logic would extend the denominator and numerator in the long-division algorithm to rationals (preferably not periodic).

Of course, for practical long-division this presents a bit

more complicated calculation than in a positive base, for the bounds on the "partial remainder," r_{p-i}, are not quite as obvious—
—at least for us who have little feeling for negative bases. r_i/Db^i must be between $\frac{b}{1-b}$ and $\frac{1}{1-b}$ inclusive or, equivalently, each r_i/b^i must be between $\frac{Db}{1-b}$ and $\frac{D}{1-b}$. $\frac{Db}{1-b}$ is just $\frac{D}{1-b}$ with the decimal point shifted one place to the right. 1-b is written 1Z1 or $\bar{1}1$ in all negative bases. Therefore, before the actual general long-division problem can be worked, one must calculate $D/\bar{1}1$, another problem. However, the bounds for each partial remainder in this latter problem are more obvious: 1 and 10 times the appropriate multiple of b. This seems to indicate that the negative

Example (see also chapter 9):

```
      2 58.85
16 ) 14 31.00
     12 00.
     ------
      2 31.
      2 00.
      -----
        31.
        48.
        ---
         3.00
         4.80
         ----
          .20
          .20
          ---
```

This is the base neg. ten equivalent of

$$-4\overline{)-629}$$

in base ten.

base division algorithm is a bit impractical, although theoretically interesting.

Computer-type non-restoring division is more efficient and particularly suited to negative base notation. In non-restoring, positive-base long division, the divisor is multiplied by the largest possible power of the base such that the divisor is still less than the dividend. This product is then repeatedly subtracted from the dividend until the remaining difference changes sign. A count is

kept of the number of subtractions. This number is the first digit of the quotient. This is similar to normal long division except that "one too many" subtractions occurs (thus changing the sign of the partial remainder). The above minuend is then shifted one digit right and repeatedly added to the partial remainder until another sign change occurs. The count determines the next digit of the quotient which is a negative digit. The above minuend is again shifted right and now subtracted repeatedly from the last partial remainder. The count of the subtractions (the third digit of the quotient) has a positive weight this time. The process is continued until a zero remainder is encountered or a specified position arbitrarily terminates the process. The digits (coefficients of the powers of the base) alternate in sign and must be converted to the standard positive-base representation. It is of interest to note that this alternating sign form is roughly a negative-base notation. For example, $2\ \bar{8}4\ \bar{3}1_{10}$ is converted to $1\ 23\ 71_{10}$ but is equal to $2\ 84\ 31_{-10}$. We see that, except in the case of a zero remainder, one too many subtractions is made, causing the non-standard form of the quotient.

However, in negative-base long division the sign change is exactly what we want most of the time, since $\frac{b}{1-b} \leq \frac{r}{d} \leq \frac{1}{1-b}$ is an interval mostly on the negative side of zero. In other words, $\frac{-b}{1-b}$ of the time we find the first sign change occuring when the partial remainder is in the correct interval. The "one too many" subtractions will occur $\frac{1}{1-b}$ of the time on the average. This eventually results in an "illegally large" digit in the quotient, which will have to be converted to standard form, where each digit, or coefficient, is

less than -b). For example, in base neg. ten.

```
          4T   =  30
      ------
1 83 ) 13 10
     - 18 3      Four
       16 80     subtractions
     - 18 3      until sign
       18 50     change of
     - 18 3      partial
          20     remainder
     - 18 3
sign ->  3 90    ten subtractions
change - 1 83    until sign change
           .     or zero remainder
           .     (in this case—
           .     which terminates
     - 1 83      problem).
         0
```

"T" stands for the digit ten, not defined for standard notation.

It is easily seen that this is the same process as the first approach to long division except that different limits are placed on the partial remainder and the quotient usually must be adjusted to fit standard notation. This eliminates the necessity of calculating the interval for the partial remainder, but necessitates the simpler process of adjusting the quotient.

7.4 Theorem Let N be of order b^n and let D be of order b^d. Then $Q = N/D$ is of order b^{n-d-2}, b^{n-d}, or b^{n-d+2}.

PROOF: Suppose N/D does not have one of these three orders.

$QD = N$. Apply theorem 4.8 to $QD = N$. N could not have order b^n in any circumstances.

Yet, examples may be given which generate all three possibilities listed in the theorem: 121/9 = 9, 1/1 = 1, and 9/17 = 17 where b = neg. ten.

Of special interest are the decimal expansions of $\frac{1}{1-b}$, $\frac{b}{1-b}$, (already shown) and other multiples of $\frac{1}{1-b}$:

$$\frac{-1}{1-b} = \frac{1}{b-1} = b^{-1} + b^{-2} + b^{-3} + \ldots = .11111\ldots$$

Therefore, $-\frac{a}{b-1}$ = .aaaaa... where $0 = a = Z$.

Also, $\frac{b}{1-b}$ = .ZZZZZ... + .11111...

Manually add this. Of course, since one cannot start at the far right, $-\infty$, one must start somewhere right of the decimal. Depending on whether an even or odd number of digits is involved, one gets

.ZZ ZZ Z...	OR	.ZZ ZZ ZZ ...	OR	(-Z)/(1-b)
+ .11 11 1...		+ .11 11 11 ...		+ (-1)/(1-b)
1Z.0Z 0Z 0...	=	.Z0 Z0 Z0 ...	=	b/(1-b)

8. Base conversion

The common algorithms for changing the notation of a positive based number into its notation on another base generalize to include negative bases. This calculation normally takes place in the original base system. Also, the symbols used for the digits of the base with the smaller absolute value usually form a subset of the set of symbols for the digits of the other base.

One problem, however, occurs in generalizing. In some situations (e.g. addition) positive-based negative numbers must be handled according to different rules than the positives. This problem is founded on the viewpoint that special representations compatible with the positive integers are not provided in positive bases. That is, -14 is really shorthand for 0 - (14). However, even in another sense where the "-" may be considered a multiplier "digit", e.g. (-1)14, this digit is handled in an entirely different manner from the other digits and therefore ruins the simplicity of some of the algorithms. For base conversion between positive bases the magnitude and sign may be handles separately, However, with negative bases the problem must be overcome by noting an exception.

8.1 A base conversion algorithm for integers Let N be any positive or negative integer. Let B be any base and b be any suitable base in which the representation of N is known. Define n'_i inductively as

$$M_j = M_{j+1}B + n'_j \qquad j = 0,1,2,\ldots,m'$$

$$\text{and } N = M_0 \qquad \text{where } 0 \leq n'_j < |B|$$

Then $N = \sum_{j=0}^{m'} n'_j B^j$. However, if $N<0$ and $B>1$, $M_j = M_{j+1} - n'_j$ and then $N = -\sum_{0}^{m'} n'_j B^j$.

Notes; 1) M_j, M_{j+1}, and B are normally written in base b for calculation.

2) n'_j is represented by the appropriate base B digit-symbol.

3) $n'_j = 0$ for $j > m'$.

PROOF: The proof for the exception is similar to this proof for the rest of the theorem:

$$N = M_0 = BM_1 + n_0$$
$$= B(BM_2 + n_1) + n_0$$
$$= B^2M_2 + n_1B + n_0$$
$$\text{etc.}$$
$$= B^{m'}M_{m'} + n_{m'-1}B^{m'-1} + \dots + n_1B + n_0$$
$$= B^{m'+1}0 + n_{m'}B^{m'} + \dots + n_0$$

$M_{m'+1} = 0$ for some m' because $|M_{j+1}| < |M_j|$ unless $M_j = 0$.

Actually the reader can see that there is no necessity of considering N as an integer in base b notation. This algorithm may be considered the algorithm for determining the base b representation of the abstract integer N. However, something more adequate than this is needed—something for rational (or real) numbers.

<u>8.2 A base conversion for real numbers</u>. Let A be any real number. Let b be any base, $b \leq -1$, in which the representation of A is to be found. Then $A = \sum_{i=m}^{-\infty} a_i b^i$ where m and a_i are determined by

m such that $(-b)^{m+2}/\overline{1}1 > (-1)^m A \geq (-b)^m/\overline{1}1$ and

a_m such that $(-b)^m/\overline{1}1 > (-1)^m(A - a_m b^m) \geq (-1)^m b^{m+1}/\overline{1}1$ and

a_j such that $(-b)^j/\overline{1}1 > (-1)^j(R_{j+1} - a_j b^j) \geq (-1)^j b^{j+1}/\overline{1}1$,

where $A - a_m b^m = R_m$ and $R_{j+1} - a_j b^j = R_j$.

PROOF: The structure of this is very simple, but the statement of it is unwieldly and complex, and the proof would be worse.

Note: No mention is made of an original base. It is assumed A is written in some base in order to perform the calculation implied by the theorem. As before this could also be used to find the base b notation of any abstract rational (or real) number if carried on indefinitely.

The statement of the algorithm seems more complicated than an example to which it is applied. Express -49 (base ten) in base neg. three. First, consider the powers of the new base: 1, -3, 9, -27, 81,... (written in base ten, in which all the calculation is done). $-49 - 2(-3)^3 = 5$

or $-49 = 2(-3)^3 + 5$

Now $5 = 9 - 6 + 2$

so $-49 = 2(-3)^3 + 1(-3)^2 + 2(-3)^1 + 2(-3)^0$

Therefore, $-49_{10} = 2122_{-3}$

In simple words, the multiples of appropriate powers of b which compose A are determined. A corresponding algorithm could be stated for conversion into any positive base, but this is not new except that the calculation takes place in the old base, which may be negative. The simplest way, however, to convert from a negative base is to perform the calculation (in the <u>new</u> base, b') suggested by the definition: $N = \sum_{r}^{s} n_i b^i$, $b<-1$, and $0 \leq n_i < -b$. The conversion of b and the digits, n_i, into the new base system is not difficult in most applied situations (particularly if -b is small and the absolute value of the new base is larger than that of b.) Then the multiplication and summation are performed in the new base system to arrive at the correct, new notation for N. This "algorithm" need

not be stated since it is nothing more than $N = \sum n_i b^i$ calculated in a base system other than b.

Since the coefficients of base neg. two, 1 and 0, simply indicate the presence or absence of a power of b in the composition of a real number, these algorithms take on simplified form as well as simplified calculation in base neg. two. For example, when converting from base neg. two no multiplication need be done in the above summation of $n_i b^i$, since $n_i = 1$ or 0.

There is a special case in base conversion; "translation" of a number expressed in base b to its form in base -b and vice versa. If we can find a simple algorithm for this, it could be useful in all conversions. For example, in converting from base ten to base neg. two, the number could be first conventionally converted to regular binary and then converted to "negative binary." Such a simple algorithm does, in fact, exist. Here again the incompatible representation of negatives in a positive base (specifically, the "-" sign) detract from the simple statement of the algorithm. The attempt to take the "-" sign into account yeilds the following statement of the algorithm.

<u>8.3 Base negation algorithm</u> Let $N = (-1)^{\pm} \sum_{s}^{-\infty} n_j b^j$ be a real number, base b, $|b| > 1$. Of course, $0 \leq n_j < |b|$. Then its representation in base -b is found by

$$N = (-1)^{\pm} \left(\sum_{s, \, j \text{ even}}^{-\infty} n_j b^j \; —_{-b} \sum_{s, \, j \text{ odd}}^{-\infty} n_j b^j \right)$$

where $—_{-b}$ means subtraction in base -b and

$(-1)^{\pm} = 1$ except when $b>0$ and $N<0$; then $(-1)^{\pm} = -1$.

PROOF: Suppose $b < -1$, then $-b = |b|$ and

$$N = \sum_{s}^{-\infty} n_j b^j = \sum_{j\ even}^{-\infty} n_j b^j - \sum_{j\ odd}^{-\infty} n_j b^j$$

This expression is a difference of two base -b numbers. If the subtraction is performed with base -b calculation, the result is N in base -b form.

Suppose $b > 1$. If $N > 0$,

$$N = \sum_{s}^{-\infty} n_j b^j = \sum_{j\ even}^{-\infty} n_j (-b)^j - \sum_{j\ odd}^{-\infty} n_j (-b)^j$$

This difference of two base -b numbers will yield a base -b number upon subtraction in the base -b system.

Now if $N < 0$,

$$N = -\sum_{s}^{-\infty} n_j b^j = \sum_{j\ odd}^{-\infty} n_j (-b)^j - \sum_{j\ even}^{-\infty} n_j (-b)^j$$

Hence, the bothersome "-" sign has necessitated the $(-1)^{\pm}$ function which equals -1 in this latter instance and reversed the subtraction of the two Σ's.

Example:

$$+\ 34\ 98\ 39\ 74_{10} = \left\{ \begin{array}{r} 4\ 08\ 09\ 04_{-10} \\ \underline{-_{-10}\ 30\ 90\ 30\ 70_{-10}} \\ 1\ 75\ 19\ 60\ 34 \end{array} \right.$$

$$-\ 34\ 98\ 39\ 74_{10} = \left\{ \begin{array}{r} 3\ 9\ 3\ 7_{-10} \\ \underline{-_{-10}\ 4\ 8\ 9\ 4_{-10}} \\ 45\ 02\ 41\ 86 \end{array} \right.$$

Check:

$$\begin{array}{rcl} +\ 34\ 98\ 39\ 74_{10} & = & 1\ 75\ 19\ 60\ 34_{-10} \\ \text{add}\quad \underline{-\ 34\ 98\ 39\ 74_{10}} & = & \underline{\quad 45\ 02\ 41\ 86_{-10}} \\ 0 & = & 0 \end{array}$$

Example: $34\ 09\ 63.74_{-10} = \begin{cases} 4\ 9\ 3.4_{10} \\ -_{10}\ 3\ 0\ 6.7_{10} \end{cases}$

$-25\ 91\ 57.66$

$1\ 86\ 12\ 58.46_{-10} = \begin{cases} 1\ 6\ 2\ 8.6_{10} \\ -_{10}\ 8\ 1\ 5.4_{10} \end{cases}$

$25\ 91\ 57.66$

Check:	$34\ 09\ 63.74_{-10} =$	$-25\ 91\ 57.66_{10}$
add	$1\ 86\ 12\ 58.46_{-10} =$	$25\ 91\ 57.66_{10}$
	$0 =$	0

The reader should make two observations.. First, "–" is used with two different meanings, as the negative sign for base ten and as the symbol for the subtraction operation (with a subscript to indicate which base). Second, the algorithm for directly subtracting the larger positive number from the smaller in base ten is never used; we subtract the smaller from the larger and change sign.

9. Base neg. two and machine application

The foregoing general theory may be quite directly applied to the case b = neg. two. In fact, some comments were already made concerning this case. For example, we observed that -A-A' has a simpler carry rule and that casting-out-threes does not exist in binary but may be useful in negative binary. The objective of this special consideration is the application of negative base notation to computer design. The wide use of the binary code (or such similar codes as excess-3 or base four, eight, or sixteen). suggests this study of base neg. two. However, the foregoing comments on the singular simplicity of base neg. two seem to make it a natural candidate. The only studies of negative bases that I have found [1,2] almost exclusively concentrate on this particular base.

Therefore, let b = -2 wherever possible in the forgoing. The digit/coefficients, a_i, equal 0 or 1 = -b-1 = Z. Where "negative digits" were (temporarily) needed, $\bar{a} = \bar{1} = 0\text{-}1$ if a = 1 and $\bar{a} = 0$ if a = 0.

ADDITION AND SUBTRACTION

The base neg. two digitwise addition table may be written in either a long form [1] or the abbriviated matrix form mentioned before. Let $A = \sum_{v}^{v} a_i(-2)^i = a_v \ldots a_u$ and $A' = \sum_{v'}^{v'} a'_i(-2)^i = a'_{v'} \ldots a'_u$ where $u \leq v$, $u \leq v'$, and $u \leq 0$. Then $A + A' = S = \sum_{v}^{v''} s_i(-2)^i = s_{v''} \ldots s_u$ where $u \leq v''$. Note: the lower index, u, is the same so the orders of all least significant digits are the same. If needed zero terms may be added to one of A or A' to produce lower subscripts equal to u.

$a_i + a'_i$:

a_i	0	1	0	1	0	1	0	1	0	1	0	1
a'_i	0	0	1	1	0	0	1	1	0	0	1	1
c_{i-1}	0	0	0	0	-1	-1	-1	-1	1	1	1	1
s_i	0	1	1	0	1	0	0	1	1	0	0	1
c_i	0	0	0	-1	1	0	0	0	0	-1	-1	-1

$a_i + a'_i$:

$\overset{0}{+}_i$	0	1
0	^{0}0	^{0}1
1	^{0}1	$^-$0

$\overset{-}{+}_i$	0	1
0	$^+$1	^{0}0
1	^{0}0	^{0}1

$\overset{+}{+}_i$	0	1
0	^{0}1	$^-$0
1	$^-$0	$^-$1

In this short form are defined the three addition mappings of the cartesian product $\{0,1\} \times \{0,1\}$ into $\{0,1\} \times \{\overset{0}{+}_i, \overset{-}{+}_i, \overset{+}{+}_i\}$ where $\overset{}{+}_{i+1}$ is the mapping to be used to obtain s_{i+1}. $a_u \overset{0}{+}_u a'_u$ begins the algorithm. Since this is a mapping, it might be better form to write it as $\overset{0}{+}_u : (a_u, a'_u)$.

Example:

```
   10 01 01          11 11 11
+  10 10 01       +        1
-------------     ------------
11 00 00 10          11 11 00
```

Similarly, we could list the digitwise subtraction tables for $a_i - a'_i$ or $a'_i - a_i$ for finding A-A' or A'-A respectively. However, for $-a_i - a'_i$ (for calculating 0-A-A') the table is simpler as noted earlier:

$-a_i - a'_i$:

a_i	0	1	0	1	0	1	0	1
a'_i	0	0	1	1	0	0	1	1
c_{i-1}	0	0	0	0	1	1	1	1
$\bar{s}_i$	0	1	1	0	1	0	0	1
c_i	0	1	1	1	0	0	0	0

OR

$\bar{+}$	0	1
0	0	$\bar{1}$
1	$\bar{1}$	$\bar{0}$

$\dot{\bar{+}}$	0	1
0	1	0
1	0	$\bar{1}$

Addition and subtraction would be a two-step process if the arithmetic of a computer utilized this easier, circuit-simplifying algorithm:

$$A + A' = -(-A-A') - 0$$

$$A - A' = -(-A-0) - A'$$

This would double the computing time; circuit complexity would be reduced, however, and an easy method of determining the negative of a number would be provided. An arithmetic unit employing the -A-A' algorithm might be somewhat more inefficient as the accumulator in multiplication. Therefore, if gains in simplified circuitry are made, losses in time and a longer program result.

A simple subtraction algorithm, unique for base neg. two, pops out of $-A' = A' + A'(-2)$. $A - A' = A + A' + A'(-2)$, but $A'(-2)$ is simply A' shifted one digit left. Subtraction may therefore be replaced by a three-term addition and negation by a two-term addition[1].

MULTIPLICATION & DIVISION

Multiplication is as straightforward as normal binary multiplication. This is because the digitwise multiplication tables (or matrices) are alike: $0 \times 0 = 0$, $1 \times 0 = 0$, $1 \times 1 = 1$, and

x	0	1
0	0	0
1	0	1

. However, since the character of division is different in negative bases, notes on base neg. two division are in order.

The following example will illustrate the mechanics of

(non-restoring)long division:

$1\ 01\overline{)\ 11\ 01\ 11\ 00.01}$ which in base ten is $5\overline{)\ -51.75}$.

101 is multiplied (left-shifted) by b^5 so that the order of this product is equal to the order of the dividend, i.e. b^7. As $101\,b^5$ = 10 10 00 00 is subtracted from the dividend, b^5 is added to the quotient (initially set at zero).

```
          10 00 00.00      first partial quotient
1 01 ) 11 01 11 00.01
     -10 1
     ---------
   1 10 11 11 00.01      first remainder
```

The remainder and dividend have opposite sign, so $101\,b^4$ is now subtracted from the remainder while b^4 is added to the partial quotient. (Had the remainder been of order less than b^6, say b^n, then we could take a shortcut and subtract $101\,b^{n-2}$ from the remainder while adding b^{n-2} to the quotient accumulator). However, this

```
          11 00 00.00
1 01 ) 11 01 11 00.01
     -10 1
     ---------
   1 10 11 11 00.01
   -  1 01
     ---------
     1 10 11 00.01
```

remainder is still of the same sign (positive—odd digits) as the foregoing remainder, so $101\,b^4$ is again subtracted. Simultaneously, b^4 is again added to the partial quotient. This time a sign change occurs in the remainder. (No more than two subtractions of the same number will have to be made to produce the sign change.)

```
          00 00 00.00
1 01 ) 11 01 11 00.01
     -10 1
     ---------
   1 10 11 11 00.01
   -  1 01
     ---------
     1 10 11 00.01
   -  1 01
     ---------
    11 01 11 00.01
```

Although the remainder is not restored, this division is, in some

ways, no more like base-two non-restoring long division than it is like restoring-type base-two division.

Continuation of this process leads to

```
                10 10.11 11 10 ...
          ____________________________
1 01 ) 11 01 11 00.01 00 00
     - 10 1
     ---------
     1 10 11 11 ...
      -1 01
      ---------
        1 10 ...
      - 1 01
      ---------
       11 01 1...
        - 10 1
        ---------
          11 0
        - 10 1
        ---------
        1 10 11
         - 1 01
         ---------
           1 10
         - 1 01
         ---------
          11 01 0
           - 10 1
           ---------
              10.0
            - 10.1
            ---------
               1.11
             - 1.01
             ---------
               .10 0
             - .10 1
             ---------
               .01 10
             - .01 01
             ---------
               .11 01 0
             - .00 10 1
             ---------
               .00 00 1
             - .00 10 1
             ---------
               .01 10 00
             - .00 01 01
             ---------
               .00 01 11
             - .00 01 01
             ---------
               .00 00 10
```

16th remainder

```
   00 00 00   zero th p.q.
 + 10 00 00
 ----------
   10 00 00   first p.q.
 + 1 00 00
 ----------
   11 00 00   second p.q.
 + 1 00 00
 ----------
         0    third p.q.
     + 10 00
     -------
       10 00
     + 10 00
     -------
   11 00 00
      + 1 00
      ------
   11 01 00
     + 1 00
     ------
     10 00
        10
     -------
     10 10
        + .1
     -------
     10 10.1
        + .01
     -------
     10 10.11
        + .00 1
     -------
     10 10.11 1
        + .00 01
     -------
     10 10.11 11
        + .00 00 1
     -------
     10 10.11 11 1
        + .00 00 1
     -------
     10 10.11 10 0
        + .00 00 01
     -------
     10 10.11 10 01
        + .00 00 01
     -------
     10 10.11 11 10   16th p.q.
```

The quotient is a periodic decimal with a repetitive unit of ..1110...

Now the same problem will be worked using the long division method based on algorithm 7.4. First, $\frac{1}{1-b}101$ and $\frac{b}{1-b}101$ must be found:

```
      110.11...
   ____________
111 ) 101
    -11100
    ------
      1001
     -111
     ----
       11.0
      -11.1
      -----
        1.10
         :
         :
```

OR

```
   .010101...
     x 101
   ----------
   .010101...
  +1.010101...
  ------------
  110.111111...
```

Note: $\frac{1}{1-5} = 1/\bar{1}1 = 1/111 = .010101...$

$$\frac{b}{1-b}\,101 = 10\,\frac{1}{1-b}\,101 = 1101.11\ldots$$

The next problem with this type of division is to determine how many places (to the left in this example) the divisor is to be shifted—by what power of b it is to be multiplied—for the first subtraction from the dividend. This is where, in practical problems, familiarity with the base is handy. The divisor, 101, is a large three-digit integer (in fact the largest) and the dividend, 11 01 11 00.01, is fairly small in magnitude for a number of order b^7. Therefore, we guess that the quotient is probably a smaller-order number with a relatively large magnitude. We subtract $101\,b^3$ from the dividend and observe that the first remainder is within the limits calculated above:

$$\begin{array}{r} 1 \\ 1\ 01\ \overline{)\ 11\ 01\ 11\ 00.01} \\ -\ 10\ 1 \\ \hline 11\ 01\ 00.01 \end{array}$$

$$\frac{b}{1-b}(101\,b^3) < 11\ 01\ 00.01 < \frac{1}{1-b}(101\,b^3)$$

Repeating the process with the first remainder, we (of course being very familiar with base neg. two calculation tables!) immediately observe that the next digit in the quotient must be a zero. In this type of division, any time the remainder is in the proper range but is on the same side of zero as the divisor, i.e. $0 \leq r_i < \frac{1}{1-b}\,D$, the next digit in the quotient is a zero. Therefore, $101\,b$ is the next product to be subtracted. The remainder is in the desired range, so 1 is the third digit and is in the b^1 digit position. Continuation yields, as before,

```
          10 10.11 ...
1 01 ) 11 01 11 00.01
     -10 1
      11 01 00.01
        -10 1
           10.01
          -10.1
            1.11
           -1.01
             .10
              :
              :
```

by a faster but more complicated process.

CONVERSION INTO BASE NEG. TWO

For integer two algorithms exist for simple conversion from decimal to negative binary: recursive division by -2 (algorithm 8.1) or normal conversion to binary followed by conversion to negative binary (algorithm 8.3). Some examples may be more enlightening than wordy explanations.

$312_{10} = ??_{-2}$ (I)

```
          0   r. 1
-2 )      1   r. 0
-2 )     -2   r. 1
-2 )      5   r. 0
-2 )    -10   r. 0
-2 )     20   r. 1        = 101001000 -2
-2 )    -39   r. 0
-2 )     78   r. 0
-2 )   -156   remainder 0
-2 )  312
```

(II) $312_{10} = 1\ 00\ 11\ 10\ 00_2$ (found by normal means)

$$\begin{array}{rl} 1\ 0\ 1\ 0\ 0_{-2} = & 1\ 0\ \overline{1}\ \overline{0}\ 0_{-2} \\ -_{-2}\ 0\ 1\ 1\ 0_{-2} = & +\ 0\ \overline{1}\ \overline{1}\ 0_{-2} \\ \hline 1\ 01\ 00\ 10\ 00_{-2} = & 1\ 01\ 00\ 10\ 00_{-2} \end{array}$$

$3.21875_{10} = 11.00111_2$

$$\begin{array}{r} 1.\ 0\ 1\ _2 \\ -1\ .0\ \ 1\ \ 1_2 \\ \hline 1\ 11.01\ 00\ 1_{-2} \end{array}$$

The first algorithm is for integers; the second for finite decimals. How are the other reals handled?

If a binary decimal is infinite (perhaps periodic also), its neg. two equivalent decimal is also infinite (periodic). Since we cannot write out an irrational anyway, we find a base two approximation of it. Then we perform the second algorithm to find the base neg. two equivalent of this approximation. For repeating binary decimals the period of repetion of the neg. two equivalent is at most twice the length of the original binary number. Therefore, to find the unit of repetition the algorithm must be applied to at least two periods of the binary rational. But if the algorithm is applied to only two, the rightmost digit(s) may be erroneous since a borrow from the next right period could have been ignored in the conversion. By applying the algorithm to three periods and then dropping the last period, one should find the unit of repetition of the base neg. two equivalent. For example, let 100 be the repetative unit of a base two rational such as .100100100100...; find the equivalent base neg. two unit of repetition.

$$\begin{array}{r} .1\ 0\ 0\ 1\ 0\ 0\ 1\ 0\ 0\ 1\ 0 \\ -.\ 0\ 1\ 0\ 0\ 1\ 0\ 0\ 1\ 0\ 0 \\ \hline .10110010110010110100100 \end{array}$$

The last digit is erroneous, but the period is obviously 101100 .

Two statements above should be proved.

9.1 Theorem Let $N = \sum_{r}^{s} n_i b^i$ and $N = \sum_{r'}^{s'} n'_i (-b)^i$. $s \geq r$ and $s' \geq r'$ n_r and $n'_{r'} \neq 0$. Then $r = r'$.

Note: This is a stronger case than above where $b = 2$ and we were concerned that r is finite if and only if r' is finite.

PROOF: Assume $r \neq r'$ and show contradiction. Let $r > r'$.

$$\left|\left[n_s b^s + \ldots n_r b^r\right] - \left[n'_{s'}(-b)^{s'} + \ldots + n'_{r'+1}(-b)^{r'+1}\right]\right| \geq |b|^{r'+1} ,$$

since the two bracketed expressions are not equal. If they were $n'_{r'} = 0$. But

$\left[n_s b^s + \ldots + n_r b^r\right] - \left[n_{s'}(-b)^{s'} + \ldots + n'_{r'+1}(-b)^{r'+1}\right] = n'_{r'}(-b)^{r'}$ and

$|n'_{r'}(-b)^{r'}| < |b^{r'+1}|$. Contradiction!

Similarly, $r' > r$ provides a contradiction.

9.2 Theorem If the period of a base b repeating decimal is longer than the period of its equivalent in base -b, then it is twice as long.

PROOF: Let $a_1a_2a_3\ldots a_n$ represent the repetitive unit of a periodic decimal, base -b. Then by algorithm 8.3, base negation, $c_1c_2\ldots c_n$ is the equivalent in base b (ignoring a possible borrow from the next column left of c_1). If no borrow is generated in the left-most column during the conversion, then the period of the base b equivalent is no longer than the period of the original base -b expression, and $c_1c_2\ldots c_n$ is one or more repetitive units, base b. If, however, a borrow is generated, then it must be applied to the next unit left. So two units (base -b) convert to $c'_1c'_2\ldots c'_nc_1c_2\ldots c_n$ with $c'_n = c_n$. Either no borrow is generated in forming the $c'_1c'_2\ldots c'_n$ unit(s) or the borrow is again generated. If the borrow is always generated then obviously $c'_1c'_2\ldots c'_n$ is the $\overset{\text{base b}}{\wedge}$equivalent to $a_1a_2\ldots a_n$ and $c_1c_2\ldots c_n$ is the erroneous period mentioned before. In the former case, a borrow is generated every other time, so the repetitive unit is $c'_1c'_2\ldots c'_nc_1c_2\ldots c_n$. The length of this period is equal to twice the length of the base -b period.

The most obvious conversion method is not really an algorithm but the calculation of $\sum a_i b^i$ in the base neg. two system (b is the original base). This might be the most effective method for computers to change base ten numbers into base neg. two numbers. A source of the neg. binary notations for the various powers of ten and the base neg. two equivalents of the digits 0,1,...,9 would be needed. These would suffice to calculate $\sum a_i 10^i$ in the arithmetic unit of the base neg. two computing machine.

BASE CONVERSION FROM BASE NEG. TWO

The coefficients in the polynomial representing a number in base neg. two are either one or zero. Therefore, the easiest conversion from negative binary is adding the powers of neg. two which have non-zero (i.e. one) coefficients. The addition takes place in the new base system. Example:

$11\ 01\ 00\ 10_{-2}$

$(-2)^1 = -2$

$(-2)^4 = 16$

$(-2)^6 = 64$

$(-2)7 = \underline{-128}$

-50 (base ten)

Of course, the algorithms discussed before would also provide conversion. The division-type algorithm (8.1) could be used for converting integers. The division would take place in base neg. two, the divisor would be the new base, and the remainders would be translated into the digits for that base. Alternatively, the base neg. two number could be translated into normal binary and then converted into the new base (positive) system by established means.

NOTE ON COMPUTERS

One important characteristic of computers is the finiteness of their number systems [10]. Computers actually represent all numbers in a certain finite interval as n-tuples. Zeros are added to numbers requiring less than n symbols so that they may be represented as n-tuples. Zeros are added to the right or left depending on whether the machine represents numbers as integers, pure fractions, or both (floating point). Therefore, all notations of numbers in this chapter could be made to be the same length and definition 1.2 could be modified to provide notations with a fixed number of digits. However, since this paper was not particularly orientated to practical design of digital computers, this was not done.

10. Further investigation

Another interesting theorem for investigation and proof is a square root algorithm. In positive bases we normally calculate the positive root; the negative root is the same except for sign. However, since the negative of a number in a negative base system is not quite so obvious, it would be convenient to state an algorithm which could produce either root. Nevertheless, a less complicated algorithm which seems (I have not proved it) to produce a root is the following.

10.1 A square root algorithm Let $N = \sum_{i=2m}^{-\infty} n_i b^i$ be a positive real number. Then $\sqrt{N} = \sum_{i=m}^{-\infty} r_i b^i$ where r_i is the least integer such that the following holds

$$N_0 = 0$$

$$N_{j+1} = N_j + n_{2(m-j)+1} b^{2(m-j)+1} + n_{2(m-j)} b^{2(m-j)} - r_{m-j} b^{m-j} \left(r_{m-j} b^{m-j} + \sum_{m+1}^{m-j+1} r_i b^i \right).$$

I found the statement of this process difficult to express mathematically; this certainly is not the clearest and most efficient statement. With some simple but hard to state modifications this algorithm can be made to yeild either the positive or negative root. The principle change allows one to start the algorithm in either an odd or even digit position. Two examples are given to demonstrate negative base root extraction, which is in need of proof and more investigation. Some aspects of the algorithm are similar to the one for positive bases.

Base neg. ten:

```
          1  6  4  8
       √1 99 47 84
          1
      26) 99
            96
        12) 3 47
              4 76
    12 88) 71 84
           71 84
```

```
             5  7  2
          √1 99 47 84
             1 85
   19 07) 14 47
          14 69
 9 42) 1 98 84
       1 98 84
```

Further investigation along different lines (application) could solidify this theory into practica computer design. For instance, a detailded study could produce actual block diagrams of the electronics of a machine employing negative base calculation. This, however, was not the object of this research project. An alternative for such extended study is the determination of the required modifications for converting a comercially available binary machine into a negative-binary computer. The study could determine if the required changes in the input/output and in the arithmetic unit would be far too great for such modification.

My biggest problem was not with the concepts themselves, but in trying to express them in symbols and then in working with these symbols in proving theorems. Added study could easily simplify and clarify my statements of the theory.

In summary I say that even if it were not practical to utilize negative bases (particularly base neg. two) as machine codes, the theoretical study of these modified positional number representations provided an interesting and new subject for mathematical investigation. Nevertheless, although I do not recommend a crash program to change general usage from base ten to some negative base, negative bases should not be overlooked as digital computer codes.

REFERENCES

1. Pawlak, Z., and Wakulicz, A., Use of expansions with a negative basis in the arithmetic element of a digital computer. Bull. Acad. Polon. Sci. Cl. III, vol. V (1957), pp. 233-236.

2. Balasiński, W., and Mrówka, S., On algorithms of arithmetical operations. Bull. Acad. Polon. Sci. Cl. III, vol. V (1957), pp. 803-804.

3. LeVeque, William J., Elementary Theory of Numbers, Addison-Wesley Pub. Co., Reading, Mass. (1962), pp. 17-20.

4. Landau, Edmund, Elementary Number Theory, Chelsea Pub. Co., New York (1958), pp. 13-14.

5. Long, Calvin, Elementary Introduction to Number Theory, D.C. Heath & Co., Boston (1965), pp. 15-20.

6. Jones, Burton W., The Theory of Numbers, Holt, Rinehart & Winston, New York (1955), pp. 23-26.

7. Hunter, J., Number Theory, Interscience Pub., New York (1964), pp. 21-23.

8. Sierpiński, Wacław, Elementary Theory of Numbers, Państwowe wydawnictwo naukowe, Warsaw (1964), pp. 264-266.

9. Courant, R., and Robbins, H., What is Mathematics?, Oxford Univ. Press, New York (1961), pp. 4-9, 61-72.

10. Garner, H. L., Number systems and arithmetic. Advan. Computers 6 (1965), pp. 131-194.

11. Lazarkiew, A., and Balasinski, W., A simple experimental computer with negative basis. Math. Comp. 15 (1961), pp. 275-285.

12. Reitwiesner, Geo. W., Binary arithmetic. Advan. Computers 1 (1960), pp. 231-308.