Complementary Two-Way Algorithms for Negative Radix Conversions

E. V. KRISHNAMURTHY

Abstract—This paper describes two sets of algorithms in positive radix arithmetic for conversions between positive and negative integral radix representation of numbers. Each set consists of algorithms for conversions in either direction; these algorithms are mutually complementary in the sense they involve inverse operations depending upon the direction of conversion. The first set of algorithms for conversion of numbers from positive to negative radix (negative to positive radix) proceeds serially from the least significant end of the number and involves complementation and addition (subtraction) of unity on single-digit numbers. The second set of algorithms for conversion of numbers from positive to negative radix (negative to positive radix) proceeds in parallel starting from the full number (the most significant end of the number) and involves complementation and right (left) shift operations. The applications of these algorithms to integers, mixed integer–fractions, floating-point numbers, and for real-time conversions are given.

Index Terms—Algorithms, complementary two-way algorithms, complementation, complement representation, left-to-right parallel algorithms, negative radix, positive radix, pseudodivision and pseudomultiplication algorithms, radix conversion, real-time radix conversions, right-to-left serial algorithms, shift operations, two-way algorithms.

I. Introduction

RECENTLY, Zohar [1] has described algorithms for conversions between negative and positive integral radix representations of numbers. Zohar points out that there is an asymmetry in his algorithms regarding the direction of conversion, if positive radix arithmetic is used. It is the object of this paper to describe two different sets of algorithms in positive radix arithmetic for conversions between positive and negative integral radix representation of numbers. Each set consists of algorithms for conversions in either direction; these algorithms are mutually complementary in the sense that they involve inverse operations depending upon the direction of conversion. In this sense, these algorithms are not asymmetric and can be loosely termed symmetric.

The first set of algorithms proceeds serially from the least significant or right end of the number towards the most significant or left end of the number; here the conversion of numbers from positive to negative radix (negative to positive radix) involves complementation and addition (subtraction) of unity on single-digit numbers.

The second set of algorithms for converting numbers from positive to negative radix (negative to positive radix) proceeds in parallel starting from the full number (most significant end of the number) and involves complementation and right (left) shift operations. As both sets of algorithms use positive radix arithmetic, each will be useful depending upon the facilities available in the computer.

Unlike the algorithms described by Zohar [1], the simplicity and symmetry in our algorithms arise due to the assumption that negative numbers in positive integral radix are represented in true complement form. This helps in simplifying the logical design considerably inasmuch as the conversion algorithms for positive and negative numbers are alike except for a simple terminal step. It also turns out that it is unnecessary to know the sign of the number in positive radix (which is conventionally available only at the most significant end) until the terminal step is reached.

In addition, the theory presented here shows a similarity between the use of negative radix and complement notation techniques for representing the negative numbers. This suggests that there is no special advantage in choosing a negative radix representation over a true complement representation in positive radix, if the aim is to handle negative numbers conveniently.

It will be further noted by logical designers that the algorithms described here are similar to those used for conversion of negative numbers in true complement form, from one positive radix to another. As a general introduction to the subject of radix conversions, reference is made to excellent surveys available: Sikdar [2], Knuth [3], and Cadden [4]. (See also Dietmeyer [5].)

II. Notation and Definitions

We will assume that the numbers used are integers in a radix β or $-\beta$, where β is a positive integer. Let us denote a number a in radix β by $a(\beta)$. Although our initial discussion will confine to $a(\beta)$ integral, we will indicate in the last section how the algorithms described here can be extended to fractional and floating-point numbers.

Let us also assume that the negative numbers in positive radix are represented in the true complement form, with the sign available only at the most significant end of the number, as is conventional.

Thus when $a(\beta) \geq 0$,

$$a(\beta) = \sum_{i=0}^{n} a_i \beta^i \tag{1}$$

where $0 \leq a_i \leq (\beta - 1)$, and when $a(\beta) < 0$, $\bar{a}(\beta)$ is the true or radix complement form given by

$$\bar{a}(\beta) = \beta^{n+1} - \sum_{i=0}^{n} a_i \beta^i = \sum_{i=0}^{n} \bar{a}_i \beta^i \tag{2}$$

Manuscript received May 18, 1970; revised December 7, 1970.

The author is with the Department of Applied Mathematics, Indian Institute of Science, Bangalore 12, India.

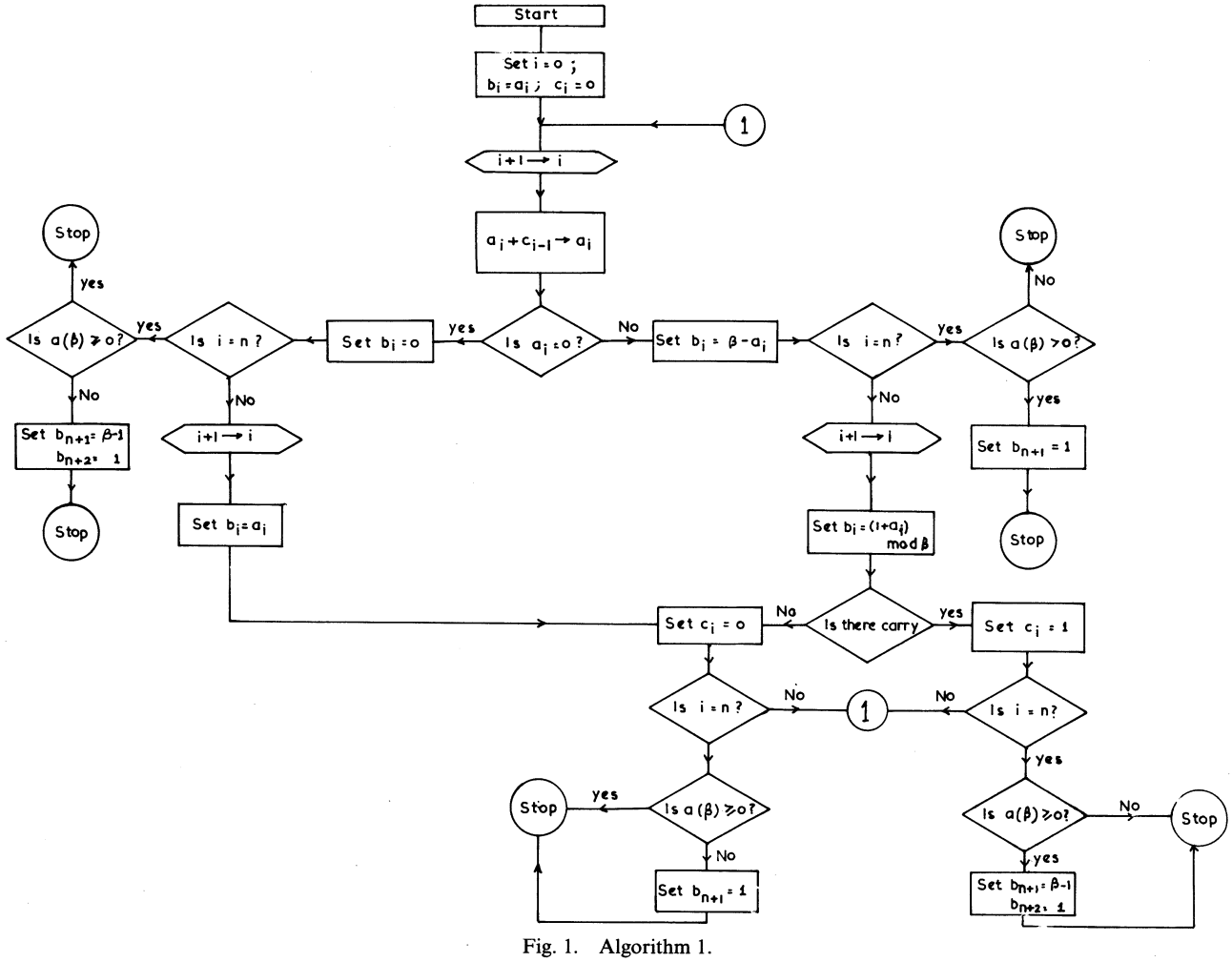

Fig. 1. Algorithm 1.

where

$$0 \leq \bar{a}_i \leq (\beta - 1).$$

We will also assume that unless otherwise specified, the numbers are normalized, i.e., if $a(\beta) > 0$, $a_n \neq 0$ and if $a(\beta) < 0$, $\bar{a}_n \neq (\beta - 1)$.

Let us denote $a(\beta)$ in radix $-\beta$ by

$$a(-\beta) = \sum_{i=0}^{m} b_i(-\beta)^i \tag{3}$$

where

$$0 \leq b_i \leq (\beta - 1)$$

and $m \geq n$ depending upon whether or not there is overflow in conversion. We also assume that unless otherwise specified $b_m \neq 0$.

III. Right-to-Left Serial Algorithms

These algorithms aim to find b_i's $(i = 0, 1, \cdots, m)$ given the a_i's $(i = 0, 1, \cdots, n)$ (or conversely) starting from $i = 0$ and proceeding to $i = n$ (or m).

A. Conversion from $a(\beta)$ to $a(-\beta)$

Algorithm 1 (see flow chart—Fig. 1): Set

$$b_0 = a_0; \qquad c_0 = 0;$$

then for each $i = (2j-1)$

$$[j = 1, 2, 3, \cdots \text{etc.},];$$

if

$$(a_{2j-1} + c_{2j-2}) = 0$$

then set

$$b_{2j-1} = 0 \quad \text{and} \quad b_{2j} = a_{2j}. \tag{4}$$

Otherwise, set

$$b_{2j-1} = \beta - a^*_{2j-1}$$

where

$$a^*_{2j-1} = a_{2j-1} + c_{2j-2} \tag{5}$$

and set

$$b_{2j} = (1 + a_{2j}) \bmod \beta \tag{6}$$

where

$$c_{2j} = \text{carry in forming } b_{2j}. \tag{7}$$

Then for the two cases n(odd) and n(even) carry out the following terminal steps.

Case 1—n(odd): After the step for $n = 2j-1$, namely setting $b_n = (\beta - a^*_n)$, do the following.

Subcase a—$a(\beta)>0$: Set

$$b_m = b_{n+1} = 1 \tag{8}$$

and terminate. This would give one extra digit to $a(-\beta)$.

Subcase b—$a(\beta)<0$: If $\bar{a}_n^* \neq 0$ terminate; otherwise set

$$b_{n+1} = (\beta - 1)$$

and

$$b_{n+2} = 1 \tag{9}$$

and terminate.

This would give two more extra digits to $a(-\beta)$; see proof below.

Case 2—n(even): After the step for $n=2j$, namely setting $b_n=a_n$ or $b_n=(1+a_n) \bmod \beta$, do the following.

Subcase a—$a(\beta)\geq 0$: If there is a carry from a_n to a_{n+1} set

$$b_{n+1} = (\beta - 1)$$

and

$$b_{n+2} = 1 \tag{10}$$

and terminate. This would give two more extra digits to $a(-\beta)$; see proof below. Otherwise (no carry from a_n), terminate.

Subcase b—$a(\beta)<0$: If there is a carry from $\bar{a}_n$ (which can arise only when $\bar{a}_n=(\beta-1)$ and $\bar{a}_{n-1}^* \neq 0$) ignore; otherwise set

$$b_{n+1} = 1 \tag{11}$$

and terminate. This would give one more extra digit to $a(-\beta)$; see proof below.

For the sake of convenience, in Table I we summarize the rules for the terminal step. Here we assume that $a(\beta)$ and $\bar{a}(\beta)$ are normalized or $a_n \neq 0$ and $\bar{a}_n \neq (\beta-1)$. Also we denote by P the parity of n by setting odd = 1, even = 0, and the sign S of $a(\beta)$ by setting positive = 0, negative = 1, and the carry C arising from a_n (or $\bar{a}_n$) by yes = 1, no = 0.

It is easily observed that the rules are: if $C=1$, then $b_{n+1}=(\beta-1)$ and $b_{n+2}=1$; otherwise, if $P \oplus S=1$, then $b_{n+1}=1$ and $b_{n+2}=0$; otherwise ($P \oplus S=0$), $b_{n+1}=b_{n+2}=0$. Here $\oplus$ denotes EXCLUSIVE OR or sum modulo-2 operation.)

Note that the combinations $P=1$, $S=0$, $C=1$ and $P=0$, $S=1$, $C=1$, which are missing from Table I, correspond to the unnormalized cases $a_n=0$ and $\bar{a}_n=(\beta-1)$, respectively. In these cases we ignore the carry and set $b_{n+1}=b_{n+2}=0$.

Proof: Consider

$$a(\beta) = \sum_{i=0}^{n} a_i \beta^i \tag{1}$$

or

$$\bar{a}(\beta) = \sum_{i=0}^{n} \bar{a}_i \beta^i \tag{2}$$

and

$$a(-\beta) = \sum_{i=0}^{m} b_i (-\beta)^i. \tag{3}$$

TABLE I

P	S	C	b_{n+1}	b_{n+2}
1	0	0	1	0
1	1	0	0	0
1	1	1	$(\beta-1)$	1
0	0	1	$(\beta-1)$	1
0	0	0	0	0
0	1	0	1	0

Case 1—n(odd): Consider a function $f_1(\beta)$ given by

$$\begin{aligned} f_1(\beta) = a_0 &+ (\beta - a_1)(-\beta) + (a_2 + 1)(-\beta)^2 + \cdots \\ &+ (\beta - a_{2j-1})(-\beta)^{2j-1} + (a_{2j} + 1)(-\beta)^{2j} + \cdots \\ &+ (a_{n-1} + 1)(-\beta)^{n-1} + (\beta - a_n)(-\beta)^n \cdots \end{aligned} \tag{12}$$

which is derived from $a(\beta)$ (or $\bar{a}(\beta)$) by the rules of Algorithm 1. We see that

$$f_1(\beta) = a(\beta) - \beta^{n+1}. \tag{13}$$

Subcase a—$a(\beta)\geq 0$: Comparing the coefficients of $(-\beta)^i$ in (12) and (3) we obtain

$$a(-\beta) = f_1(\beta) + \beta^{n+1}. \tag{14}$$

Thus $a(-\beta)$ is obtained by setting $b_m=b_{n+1}=1$ and terminating. [Compare (8).]

Subcase b—$a(\beta)<0$: Here the number is denoted by $\bar{a}(\beta)$ and indicated by the negative sign at the most significant end. If $\bar{a}_n^* \neq 0$ no carry arises from $\bar{a}_n^*$ and we have

$$f_1(\beta) = \bar{a}(\beta) - \beta^{n+1} = -a(\beta) = a(-\beta) \tag{15}$$

the result being directly available. If $\bar{a}_n^*=0$, we would have set

$$b_n = \beta - \bar{a}_n^* = 0$$

and the overflow digit $-\beta^{n+1}$ in (12) has to be taken care of by adding digits $b_{n+1}=(\beta-1)$ and $b_{n+2}=1$ so that $(\beta-1) \cdot (-\beta)^{n+1} + 1 \cdot (-\beta)^{n+2} (n \text{ odd}) = -\beta^{n+1}$. [Compare (9).]

Case 2—n(even): Consider a function $f_2(\beta)$ given by

$$\begin{aligned} f_2(\beta) = a_0 &+ (\beta - a_1)(-\beta) + (a_2 + 1)(-\beta)^2 + \cdots \\ &+ (\beta - a_{2j-1})(-\beta)^{2j-1} + (a_{2j} + 1)(-\beta)^{2j} + \cdots \\ &+ (\beta - a_{n-1})(-\beta)^{n-1} + (a_n + 1)(-\beta)^n \end{aligned} \tag{16}$$

which is derived from $a(\beta)$ by the rules of the Algorithm 1. It is easy to see that

$$f_2(\beta) = a(\beta). \tag{17}$$

Subcase a—$a(\beta)\geq 0$: If $a_n<(\beta-1)$ carry does not arise from (a_n+1), and comparing the coefficients of $(-\beta)^i$ in (16) and (3), we obtain

$$f_2(\beta) = a(-\beta). \tag{18}$$

Also, if $a_n=(\beta-1)$ and $a_{n-1}^*=0$, carry does not arise and the result is obtained directly (18).

If $a_n=(\beta-1)$ and $a_{n-1}^* \neq 0$, then carry arises from (a_n+1) which equals β^{n+1}. If it arises it is to be replaced in radix $-\beta$ representation by the two digits $b_{n+1}=(\beta-1)$ in the $(-\beta)^{n+1}$ position and $b_{n+2}=1$ in the $(-\beta)^{n+2}$ position. [Compare (10).]

Subcase b—$a(\beta)<0$: If $\bar{a}_n=(\beta-1)$ (which is a superfluous digit) and $\bar{a}^*_{n-1}\neq 0$, then a carry arises from $\bar{a}_n+1$.

Since

$$f_2(\beta) = \beta^{n+1} - a(\beta) = \bar{a}(\beta)$$
$$-a(\beta) = f_2(\beta) - \beta^{n+1} = a(-\beta) \tag{19}$$

and we get the result by ignoring the carry. Otherwise (if $\bar{a}_n=(\beta-1)$ and $\bar{a}^*_{n-1}=0$ or $\bar{a}_n<(\beta-1)$) carry does not arise from a_n+1 or $\beta^{n+1}=0$.

Thus

$$\bar{a}(\beta) - \beta^{n+1} = -a(\beta) = a(-\beta)$$
$$= f_2(\beta) + 1\cdot(-\beta)^{n+1} \tag{20}$$

and we obtain the result $a(-\beta)$ by setting $b_{n+1}=1$ in the $(-\beta)^{n+1}$ position. [Compare (11).]

Remark 1: When $a^*_{2j-1}=0$, we set

$$b_{2j-1} = 0$$
$$b_{2j} = a_{2j}$$

and thus we do not modify $a(\beta)$ at a preceding step and correct at a succeeding step. This aspect has been omitted from consideration in the proof, since it is trivial.

Remark 2: When $a_{2j}=(\beta-1)$ a carry arises from $b_{2j}=(a_{2j}+1) \bmod \beta$ and this is taken care of by setting

$$b_{2j} = 0 \tag{21}$$

and adding 1 to $a_{2(j+1)-1}$; thus at a subsequent step

$$b_{2j+1} = \beta - (a_{2j+1} + 1) \tag{22}$$

and the carry is assimilated without further propagation since $a_{2j+1}\leq\beta-1$.

Example 1—n(odd), $a(\beta)>0$, no carry from a^*_n:

$$a(10) = 9\ 1\ 9\ 5$$
$$a(-10) = 1\ 1\ 2\ 1\ 5.$$

Note one extra digit arising here.

Example 2—n(odd), $a(\beta)<0$, $\bar{a}^*_n\neq 0$, no carry from $\bar{a}^*_n$:

$$\bar{a}(10) = 8\ 1\ 9\ 5$$
$$a(-10) = 2\ 2\ 1\ 5$$
$$a(10) = -1\ 8\ 0\ 5.$$

Example 3—n(odd), $a(\beta)<0$, $\bar{a}^*_n=0$, carry from $\bar{a}^*_n$:

$$\bar{a}(10) = 0\ 8\ 9\ 9$$
$$a(-10) = 1\ 9\ 0\ 9\ 1\ 9$$
$$a(10) = -9\ 1\ 0\ 1.$$

Note the extra two digits arising here.

Example 4—n(even), $a(\beta)>0$, carry from a_n:

$$a(10) = 9\ 1\ 0\ 1\ 9\ 0\ 9\ 2\ 9\ 7\ 1$$
$$a(-10) = 1\ 9\ 0\ 9\ 1\ 8\ 0\ 9\ 0\ 7\ 0\ 3\ 1.$$

Note the extra two digits arising here.

Example 5—n(even), $a(\beta)>0$, no carry from a_n:

$$a(10) = 9\ 0\ 0\ 1\ 9\ 0\ 9\ 2\ 9\ 7\ 1$$
$$a(-10) = 9\ 0\ 1\ 8\ 0\ 9\ 0\ 7\ 0\ 3\ 1.$$

Example 6—n(even), $a(\beta)<0$, carry from $\bar{a}_n$:

$$\bar{a}(10) = 9\ 1\ 9\ 5\ 2$$
$$a(-10) = 0\ 8\ 0\ 5\ 2$$
$$a(10) = -8\ 0\ 4\ 8.$$

Example 7—n(even), $a(\beta)<0$, no carry from $\bar{a}_n$:

$$\bar{a}(10) = 9\ 0\ 8\ 5\ 2$$
$$a(-10) = 1\ 9\ 0\ 9\ 5\ 2$$
$$a(10) = -9\ 1\ 4\ 8.$$

Note one extra digit arising here.

B. Conversion from $a(-\beta)$ to $a(\beta)$

Let

$$a(-\beta) = \sum_{i=0}^{m} b_i(-\beta)^i \tag{3}$$

where

$$b_m \neq 0$$

and

$$a(\beta) = \sum_{i=0}^{n} a_i\beta^i. \tag{1}$$

The following algorithm finds a_i's ($i=0, 1, 2, \cdots, n$) given b_i's.

Algorithm 2 (see flow chart—Fig. 2): Set

$$a_0 = b_0; \qquad c_0 = 0;$$

then for each $i=(2j-1)$

$$[j = 1, 2, 3, \cdots, \text{etc.}].$$

If

$$(b_{2j-1} + c_{2j-2}) = 0 \tag{23}$$

then set

$$a_{2j-1} = 0 \tag{24}$$

and

$$a_{2j} = b_{2j}. \tag{25}$$

Otherwise, set

$$a_{2j-1} = (\beta - b^*_{2j-1}) \tag{26}$$

where

$$b^*_{2j-1} = b_{2j-1} + c_{2j-2} \tag{27}$$

and set

$$a_{2j} = (b_{2j} - 1) \bmod \beta \tag{28}$$

where

$$c_{2j} = \text{borrow in forming } a_{2j}. \tag{29}$$

Then for the two cases m(odd) and m(even) carry out the following terminal steps.

Case 1—m(odd): After the step for $m=2j-1$, namely setting $a_m=(\beta-b^*_m)$, terminate; the result is negative and in the complement form (note $b_m\neq 0$ by assumption).

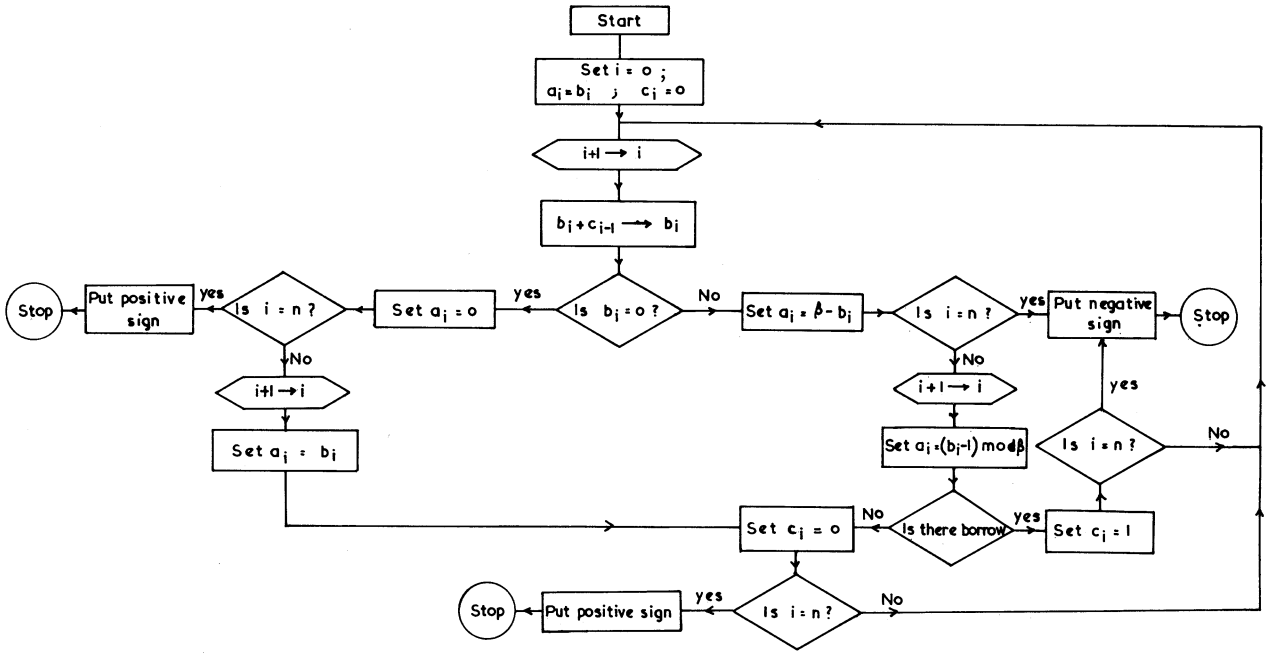

Fig. 2. Algorithm 2.

Case 2—m(even): After the step for $m=2j$, namely setting $a_m = b_m$ or $a_m = (b_m - 1) \bmod \beta$, terminate; the result is positive (note $b_m \neq 0$ by assumption).

The proof is omitted here as it is similar to Algorithm 1.

Example 1—m(odd):

$$\begin{aligned} a(-10) &= \quad 1\ 0\ 0\ 0\ 7\ 5\ 4\ 8 \\ \bar{a}(10) &= \quad 8\ 9\ 9\ 9\ 3\ 4\ 6\ 8 \\ a(10) &= -1\ 0\ 0\ 0\ 6\ 5\ 3\ 2. \end{aligned}$$

Example 2—m(even):

$$\begin{aligned} a(-10) &= 1\ 9\ 0\ 9\ 1\ 8\ 0\ 9\ 0\ 7\ 0\ 3\ 1 \\ a(10) &= 0\ 0\ 9\ 1\ 0\ 1\ 9\ 0\ 9\ 2\ 9\ 7\ 1. \end{aligned}$$

Example 3—m(odd):

$$\begin{aligned} a(-10) &= \quad 1\ 9\ 0\ 1\ 8\ 0\ 9\ 0\ 7\ 0\ 3\ 1 \\ \bar{a}(10) &= \quad 9\ 9\ 0\ 0\ 1\ 9\ 0\ 9\ 2\ 9\ 7\ 1 \\ a(10) &= -0\ 0\ 9\ 9\ 8\ 0\ 9\ 0\ 7\ 0\ 2\ 9. \end{aligned}$$

Note the redundant digits in $\bar{a}(10)$ in the two most significant positions.

IV. Left-to-Right Parallel Algorithms or Pseudo-division-Multiplication Algorithms

A. Principle of the Algorithms

It is well known [2]–[4] that while converting integers from a positive radix β to another positive radix γ, one can employ a division technique in radix β or a multiplication technique in radix γ. The division technique employs a recursion

$$X_i = X_{i+1}\gamma + r_i \tag{30}$$

for $i=0, 1, 2, \cdots, m$ with $X_0 = a(\beta)$; the procedure obtains a quotient X_{i+1} and a remainder r_i each time by dividing X_i by γ. This recursion terminates when $X_i = 0$.

The remainders r_i, each being less than γ, form the digits $b_i (i=0, 1, \cdots, m)$ of the converted number $a(\gamma)$ given by

$$a(\gamma) = \sum_{i=0}^{m} b_i \gamma^i. \tag{31}$$

Note that if $\beta > \gamma$ then $m > n$ and the valid digits of γ are subsets of those of β, and hence no facility for translating the individual digits r_i are needed.

If $\beta < \gamma$ then $m < n$, and in order to carry out the arithmetic in radix β, γ has to be expressed in radix β; then during division each one of the remainders $r_i (\leq \gamma - 1)$ would only be obtained in the β coded form. Therefore the r_i's will have to be finally translated into radix γ. The multiplication technique for converting from radix β to radix γ employs arithmetic in radix γ. For $\beta > \gamma$, β is expressed in radix γ, say β_γ, and each one of the coefficients a_i in $a(\beta)$ is expressed as $a_{i\gamma}$, and each one of the coefficients a_i in $a(\beta)$ is expressed as $a_{i\gamma}$ in radix γ, and the polynomial $a(\beta)$ is evaluated as a nested multiplication by the recursion

$$X_i = X_{i-1}\beta_\gamma + a_{(n-i)\gamma} \tag{32}$$

for $i=1, 2, \cdots, n$ with

$$X_0 = a_{n\gamma}. \tag{33}$$

This recursion terminates at the nth step yielding

$$X_n = a(\gamma). \tag{34}$$

Naturally the evaluation of (32) demands facilities for expressing a_i as $a_{i\gamma}$ and facilities for arithmetic in radix γ. For $\beta < \gamma$, the a_i's are valid digits in γ; so the facility for translating a_i to $a_{i\gamma}$ is not needed.

Now let us consider the application of these schemes to conversion between numbers expressed in positive and negative radices of equal magnitude. First, let us consider the multiplication scheme (note asymmetry arises here due

to the fact that while conversion from $a(-\beta)$ to $a(\beta)$ is possible using radix β arithmetic, conversion from $a(\beta)$ to $a(-\beta)$ is possible only by using radix $-\beta$ arithmetic and not in radix β arithmetic). Since the valid set of digits for β and $-\beta$ remain identical, conversion from $a(-\beta)$ to $a(\beta)$ does not involve translation of the digits b_i of $a(-\beta)$ in radix β; also, since the multiplication by $-\beta$ can be replaced by complementation and a left shift by one digit in radix β, this technique is very convenient.

As mentioned, conversion from $a(\beta)$ to $a(-\beta)$ in radix β arithmetic is possible only by employing division technique. In this case we have to divide $a(\beta)$ by $-\beta$ and obtain the remainders r_i. Since division by $-\beta$ involves only right shift and complementation, this technique is very convenient. Also, since the valid set of digits for β and $-\beta$ remain the same, facility for translating the digits $b_i=r_i$ (which are actually in $(-\beta)$ coded form) in radix β are not needed.

Thus the scheme for conversion from $a(\beta)$ to $a(-\beta)$ and the scheme for conversion from $a(-\beta)$ to $a(\beta)$ are inverses of each other inasmuch as the former proceeds with a sequence of right shifts and complementation and the latter proceeds with a sequence of complementation and left shifts. From the nature of the operations involved, one can call these schemes pseudodivision and pseudomultiplication algorithms, respectively.

It is in fact possible to use circular shift registers and implement the algorithms so that $a(\beta)$ [or $a(-\beta)$] is replaced by $a(-\beta)$ [or $a(\beta)$] at the termination of the operations. (See Examples.)

B. Conversion from $a(\beta)$ to $a(-\beta)$

Algorithm 3 (see flow chart—Fig. 3): Set

$$X_0 = a(\beta) \tag{35}$$

and use the recursion (30) setting $\gamma=-\beta$; thus

$$X_i = X_{i+1}(-\beta) + r_i \tag{36}$$

for $i=0, 1, 2, \cdots, m$. Here we restrict that $r_i \geq 0$, and the corresponding quotient X_{i+1} is chosen. The recursion terminates when $X_i=0$.

Since division by $(-\beta)$ is equivalent to a right shift of X_i through a single digit and then complementing the right shifted X_i, we can rewrite the above recursion (36) thus:

$$X_{i+1} = \overline{R(X_i)} \tag{37}$$

and

$$b_i = \text{shifted out digit of } X_i = X_{i0} \text{ (say)} \tag{38}$$

with

$$b_0 = a_0$$

where

$$R(X_i) = X_i \text{ shifted right through one digit}$$

$$\overline{X_i} = \text{true complement of } X_i.$$

Since the conversion of $a(\beta)$ to $a(-\beta)$ may result in an overflow of two digits (see Section III-A), it is necessary to start with two guarding zero digits (or $(\beta-1)$ digits) for $a(\beta)>0$ (for $a(\beta)<0$ expressed in true complement form).

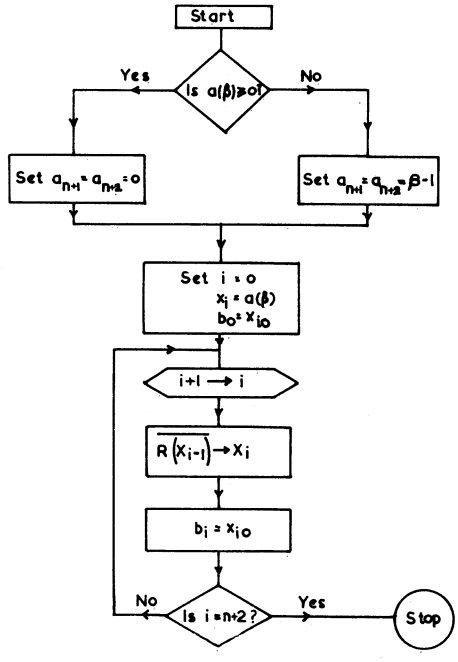

Fig. 3. Algorithm 3.

Example 1: The computations for converting from $a(10)=0\,0\,9\,9\,0\,0\,1\,9\,0$ to $a(-10)=1\,1\,0\,0\,2\,1\,0$ are shown in Table II.

Example 2: The computations for converting from $\bar{a}(10)=0\,8\,9\,9$ to $a(-10)=1\,9\,0\,9\,1\,9$ are shown in Table III.

Remark 3: Note that a left circular shift register can be used conveniently in the above examples such that b_i enters each time the vacant leftmost position of the X_i register so that at the last step $a(10)$ is replaced by $a(-10)$.

C. Conversion from $a(-\beta)$ to $a(\beta)$

Algorithm 4 (see flow chart—Fig. 4): Let

$$a(-\beta) = \sum_{i=0}^{m} b_i(-\beta)^i \text{ with } b_m \neq 0. \tag{39}$$

Set

$$X_0 = b_m \tag{40}$$

and use the equivalent recursion (41) derived from (32):

$$X_i = L(\overline{X}_{i-1}) + b_{m-i}, \qquad i = 1, 2, \cdots, m \tag{41}$$

where

$$L(\overline{X}_{i-1}) = \text{shift } \overline{X}_{i-1} \text{ left by one digit}$$

$$\overline{X}_{i-1} = \text{true complement of } X_{i-1}.$$

The recursion terminates when $i=m$.

Case 1—m(odd): The result is negative and in true complement form.

Case 2—m(even): The result is positive and in the desired form.

Proof: The proof follows from the explanation offered earlier in this section and the recursion (32) and the equiva-

TABLE II

i	X_i								b_i
0	0	0	9	0	0	1	9	0	0
1		9	9	0	9	9	8		1
2			0	0	9	0	0		2
3				9	9	1	0		0
4					0	0	9		0
5						9	9		1
6							0		1
7									0

TABLE III

i	X_i					b_i
0	9	9	0	8	9	9
1		0	0	9	1	1
2			9	9	0	9
3				0	1	0
4					9	9
5						1

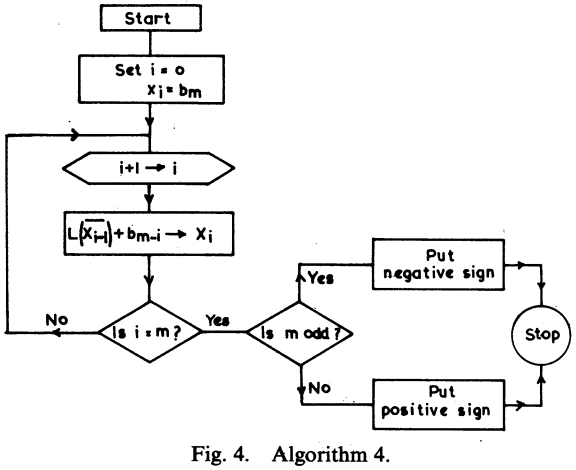

Fig. 4. Algorithm 4.

lent recursion (41). However, since complement notations are involved, we will add a short proof for the sake of clarity.

Since we set

$$X_0 = b_m \tag{40}$$

$$X_1 = L(\bar{X}_0) + b_{m-1}$$

$$= \beta^2 - \beta b_m + b_{m-1} \tag{42}$$

$$X_2 = \beta^2 b_m - \beta b_{m-1} + b_{m-2}, \tag{43}$$

and in general,

$$X_{i(\text{odd})} = (-1)^{i+1}\beta^{i+1} + (-1)^i \beta^i b_m + \cdots + b_{m-i} \tag{44}$$

$$X_{i(\text{even})} = (-1)^i \beta^i b_m + (-1)^{i-1}\beta^{i-1} b_{m-1} + \cdots + b_{m-i}; \tag{45}$$

thus

$$X_{m(\text{odd})} = \beta^{m+1} + a(-\beta) = \bar{a}(\beta) \tag{46}$$

and

$$X_{m(\text{even})} = \bar{a}(-\beta) = a(\beta) \tag{47}$$

as was to be proved.

TABLE IV

i	X_i				
0					9
1				1	0
2			9	0	1
3		0	9	9	8

TABLE V

i	X_i				
0					9
1				1	1
2			8	9	1
3		1	0	9	8
4	8	9	0	2	8

Example 1: The computations for converting from $a(-10)=9\ 0\ 1\ 8$ to $\bar{a}(10)=0\ 9\ 9\ 8$ are shown in Table IV.

Example 2: The computations for converting from $a(-10)=9\ 1\ 1\ 8\ 8$ to $a(10)=8\ 9\ 0\ 2\ 8$ are shown in Table V.

Remark 4: Note that a right circular shift register can be used conveniently in the above examples, so that the left-most digit of $a(-10)$ enters the vacant position at the right-most position of X_i register so that at the last step $a(-10)$ is replaced by $a(10)$.

V. Fractions and Floating-Point Numbers

All the above algorithms can be used with practically little or no modifications to convert fractions, mixed integer–fractions, and floating-point numbers.

A. Conversion of Fractions and Mixed Integer–Fractions from Radix β to Radix $-\beta$

Let

$$a(\beta) = \sum_{i=-k}^{n} a_i \beta^i \tag{48}$$

with k fractional digits and $(n+1)$ integral digits. Then Algorithm 1 can be used as such except for the initial starting step. At the initial step instead of setting $b_0 = a_0$ as in Algorithm 1, we make the following choices.

Case 1—k(odd): If

$$a_{-k} \neq 0$$

set

$$b_{-k} = (\beta - a_{-k}). \tag{49}$$

Otherwise set

$$b_{-k} = 0 \tag{50}$$

and proceed with recursions (4) and (5) or (6) and (7) as the case may be.

Case 2—k(even): Set

$$b_{-k} = a_{-k} \tag{51}$$

and proceed with recursions (4) and (5) or (6) and (7) as the case may be.

The proof is omitted here.

Remark 5: A fraction in β representation can assume a mixed integer–fraction form in $-\beta$ representation.

Example 1:

$$a(10) = \quad 2\ 9 \cdot 7\ 8\ 4\ 6$$
$$a(-10) = 1\ 7\ 0 \cdot 3\ 9\ 6\ 6.$$

Example 2:

$$\bar{a}(10) = 9\ 1 \cdot 9\ 5\ 2$$
$$a(-10) = 1\ 2 \cdot 1\ 6\ 8$$
$$a(10) = -\ 8 \cdot 0\ 4\ 8.$$

B. Conversion of Fractions and Mixed Integer–Fractions from Radix $-\beta$ to Radix β

Let

$$a(-\beta) = \sum_{i=-k}^{m} b_i(-\beta)^i \tag{52}$$

with k fractional digits and $(n+1)$ integer digits. Then Algorithm 2 can be used as such except for the initial starting step. At the initial step instead of setting $a_0 = b_0$ as in Algorithm 2 we make the following choices.

Case 1—k(odd): If

$$b_{-k} \neq 0$$

set

$$a_{-k} = (\beta - b_{-k}); \tag{53}$$

otherwise set

$$a_{-k} = 0 \tag{54}$$

and proceed with recursions (23) to (29) as the case may be.

Case 2—k(even): Set

$$b_{-k} = a_{-k} \tag{55}$$

and proceed with the recursions (23) to (29) as the case may be.

The proof is omitted here.

Remark 6: A mixed integer–fraction number in $-\beta$ representation can assume a fractional form in $+\beta$ form.

Example 1:

$$a(-10) = \quad 1 \cdot 9\ 2\ 4$$
$$a(10) = \quad 0 \cdot 1\ 1\ 6.$$

Example 2:

$$a(-10) = 1\ 9 \cdot 9\ 2\ 4$$
$$\bar{a}(10) = 9\ 8 \cdot 1\ 1\ 6$$
$$a(10) = -\ 1 \cdot 8\ 8\ 4.$$

Remark 7: Note that Algorithms 3 and 4 can work with similar minor modifications for fractional and mixed numbers.

C. Floating-Point Numbers

It is clear that in the floating-point representation, if the exponent is even, it is sufficient to convert only the mantissa and retain the same exponent.

If the exponent is odd, the mantissa is shifted left by one digit and the exponent is reduced by one so as to make it even, and the conversion of the mantissa is carried out by one of the above algorithms.

D. Real-Time Conversions

Algorithms 1 and 2 suggest that conversion of a number from radix β to radix $-\beta$ as well as from radix $-\beta$ to radix β is possible in real time, in the sense that for every digit of $a(\beta)$ spelled out, beginning from the least significant end, the corresponding digit of $a(-\beta)$ can be spelled out, and conversely. It is interesting to note that such a real-time conversion is impossible between any two general radices. In our case it is possible because of the fact that the allowed set of digits in radix β and radix $-\beta$ remain identical. Note, however, Algorithms 3 and 4 do not permit a real-time conversion due to the fact division or nested multiplications and additions are involved.

It is interesting to note from Table I that essentially the negative radix representation would need two bits of information to denote $P \oplus S$ and C, instead of single sign bit S used in the positive radix representation. It is also clearly seen from Algorithm 2 that the parity of number of digits in the negative radix representation plays the same role as the sign bit in the positive radix representation. In addition, note that Algorithms 1 and 2 essentially constitute a one-to-one mapping of individual digits of radix β to those of $-\beta$, and conversely. In this way the conversion is more a code translation in one-to-one correspondence. This, as pointed out earlier, is due to the fact that the number of information symbols used in both representations are equal and minimal or nonredundant.

Acknowledgment

The author wishes to thank the Department of Computer Science, Technion, Haifa, Israel, for their hospitality during the course of this work. Thanks are also due to the referees for their remarks in improving the presentation of this paper.

References

[1] S. Zohar, "Negative radix conversion," *IEEE Trans. Comput.*, vol. C-19, Mar. 1970, pp. 222–226.

[2] K. Sikdar, "A comparative study of algorithms for multiple-precision radix-conversions," *Sankhya: Ind. J. Stat.*, vol. 30 B, pts. 3 and 4, 1968, pp. 315–334.

[3] D. E. Knuth, *The Art of Computer Programming*, vol. 2. Reading, Mass.: Addison-Wesley, 1969.

[4] W. J. Cadden, "Binary numbers, codes and translators," in *A Survey of Switching Theory*, E. J. McCluskey and T. C. Bartee, Eds. New York: McGraw-Hill, pp. 15–30, 1962.

[5] D. L. Dietmeyer, "Conversion from positive to negative and imaginary radix," *IEEE Trans. Electron. Comput.* (Corresp.), vol. EC-12, Feb. 1963, pp. 20–22.