ON ARITHMETIC INTER-RELATIONSHIPS AND HARDWARE INTERCHANGEABILITY OF NEGABINARY AND BINARY SYSTEMS

Dharma P. Agrawal

Wayne State University
Detroit, Michigan 48202

Abstract

Recent use of the negabinary system in the application oriented digital hardware, has encouraged the search for suitable arithmetic algorithms in -2 base. These algorithms have been directly utilized in designing logic circuits and several logic implementations have been reported in the literature. The main objective of this paper is to show the close relationship between +2 base addition and -2 base negative addition. Two possible ways of utilizing binary adders for performing negabinary addition and their underlying theories are presented. Two similar techniques of using negabinary adders for binary addition are also considered in detail. An interesting aspect of this investigation about negabinary base is that negative addition (rather than just addition) seems to be the primitive operation from logic complexity and interchangeability of +2 and -2 adders point of view.

The technique of adding two numbers in one system by the adders of the other system is extended here for multiple operand addition. This requires inclusion of an additional correction factor. Further, the additive algorithms of this work lead to four simple conversion processes of number from one system to another. This paper seems to be a realistic step towards the use of similar hardware for +2 and -2 bases and hence this allows an instantaneous flexibility on the selection of number system. It is believed that this paper will attract more attention on the use of -2 base system for the design of special purpose digital machines and process controllers.

Index terms:

Addition, binary numbers, base conversion, even positioned bits, multiple addition, negabinary system, negative addition, negative radix, odd positioned bits.

I. Introduction

The negative base number system first proposed by Wadel[1] has been investigated by several authors from time to time. In particular, this idea was advanced and considered in detail by M.P. de Regt[2]. In a series of articles, he pointed out the feasibility of such a system for the design of an arithmetic unit by defining exactly the algorithms for addition, subtraction, multiplication and division. Although these algorithms were not well suited for their hardware implementation, they generated enormous interest in negative base and later results were concentrated in the area of positive-negative radix conversion. The short-comings of de Regt's algorithms further acted as stimulant to look into more cost effective and efficient ways of performing arithmetic[3],[4].

Amongst the general value of the negative radix, -2 base system has always been considered a suitable candidate for its practical application as -2 is in a loose sense, much closer to most-widely utilized and universally accepted binary or +2 base system. This commonality is attributed to -2 base system being usually designated as negabinary system. This similarity has also encouraged the design of logic circuits for -2 base[5-9] which are primarily direct applications or extensions of existing algorithms[3],[4].

The irrefutable advantage of unique representability for both positive and negative numbers (i.e. sign independent representation) offered by negabinary system, has been stressed by researchers working in this area. This property also makes multiple-operand addition process a lot easier[4] and thus shows its superiority over binary system only for special purpose applications like the design of digital filters[10],[11] and the fast-fourier transformation[12]. But it is well-established that for a general purpose application, the unit for the negabinary numbers are more complex than the binary system. A simple example is that of addition operation. The binary addition of two numbers needs only one carry while two carries (one positive and the second negative to the next significant position or both carries positive each one being sent to two successive positions) are essential in -2 base. Similar arguments can be given for fast carry-look-ahead adders and other complex functions.

Any theoretical work done on the design of arithmetic units can gain a momentum only when complex integrated circuit chips for such functions are commercially available. As binary number system will always have an adge over negabinary system, an optimistic view is to conclude that the negabinary units will never be produced in large scale. Thus, no advantage is gained if gates are employed as building blocks for negabinary arithmetic units and hence much emphasis should be given to devise appropriate ways of converting off-the-shelf binary units for negabinary operations thereby necessitating as less additional integrated

CH1412-6/78/0000-0088$00.75 (c) 1978 IEEE

circuits as possible.

Very recently, the generalized notion of all -2 base functions demanding more hardware than +2 base system, has been shown to be not exactly true. As a significant contribution to the advancement of -2 base system, it has been shown that negative addition of two negabinary numbers requires only one carry and the logic functions for such a negative adder and the corresponding carry-look-ahead terms for fast addition, are no more complex than the binary addition. The main objective of this paper is to perform a detailed investigation of this innovation and to have a closer look into relevant relationships between binary addition and base -2 negative addition. The most interesting property of hardware interchangeability consideration for such additions in these two bases gives us enough background to infer that negative addition seems to be the primitive function for -2 base system and can be equated to +2 base addition.

This paper is divided into eight sections and starts with a brief introduction. The second section deals with the theory of binary and negabinary number representations and simple and obvious formulations given in this section form the central idea for this paper. Two ways of using binary adders for negative negabinary addition (n.n.b.a.) is discussed in section III. The next section covers two complementary ways of performing binary addition with n.n.b. adders. Two operand addition technique of negabinary numbers given in section III is extended for multiple operand addition and associated difficulties, restrictions and modifications are detailed in section V. The relations that allow hardware interchangeability between ±2 bases, are seen to provide four different algorithms for number conversion from one base to another. This is not the theme of this paper and can be said to be a side result of this work. Finally, the last section contains concluding remarks and future trend as a consequence of this work.

II. Notations and Theory

Negabinary arithmetic

Let A be a n-bit negabinary number represented as:

$$A \rightarrow a_{n-1}\, a_{n-2} \cdots a_2 a_1 a_0 \qquad (1.a)$$

Its value can be given as

$$A = \sum_{i=0}^{n-1} a_i (-2)^i \qquad (1.b)$$

where $a_i \in (0,1)$ and $0 \le i \le n-1$.

Separating the even and odd positioned bits and forming two summation terms, equation (1) can be rewritten as:

$$A = \sum_{i\ \text{even}} a_i (2)^i - \sum_{i\ \text{odd}} a_i (2)^i \qquad (2)$$

Equation (2) can be further modified to a convenient form by adding and subtracting a summation term $\sum_{i\ \text{odd}} \bar{a}_i (2)^i$ ($\bar{a}_i$ represents the complement of a_i) and simplifying as:

$$A = \sum_{i\ \text{even}} a_i (2)^i + \sum_{i\ \text{odd}} \bar{a}_i (2)^i - \sum_{i\ \text{odd}} 2^i \qquad (3)$$

Defining $a_i^* = a_i$ for i even (4.a)

$= \bar{a}_i$ for i odd (4.b)

and let $r = \left\lfloor \frac{n-1}{2} \right\rfloor$ (5)

with $\lfloor r \rfloor$ representing the smallest integer not less than r.

Substituting (4) and (5) into (3) gives:

$$A = \sum_{i=0}^{n-1} a_i^* (2)^i - \sum_{i=1}^{r} (2)^{2i-1} \qquad (6)^1$$

It may be noted that A is a negabinary number and the two summation terms on the right side of the relation (6) are essentially numbers in +2 base. A complementary relation of equal importance can be obtained by utilizing $\sum_{i\ \text{even}} a_i (2)^i$ as the summation term for addition and subtraction to relation (2) and a similar simplification leads to

$$A = -\sum_{i=0}^{n-1} a_i^{**} (2)^i + \sum_{i=0}^{s} 2^{2i} \qquad (7)$$

with

$a_i^{**} = \bar{a}_i$ for i even (8.a)

$= a_i$ for i odd (8.b)

and $s = \left\lceil \frac{n-1}{2} \right\rceil$

where $\lceil s \rceil$ means the largest integer not greater than s.

As is discussed in the later sections, relations (6) and (7) not only play a dominant role in devising appropriate algorithms for performing n.n.b. addition using binary adders, but also indicate ways for achieving multiple operand addition of negabinary numbers with only binary hardware. They are also foundations for the two bases conversion process.

Binary Arithmetic

Equations (6) and (7) also act as a motive to look about resembling relations for a binary number. This requires binary number to be shown as functionally equivalent to two negabinary summation terms. This can be done exactly the same way we reached to relation (6).

Let C be a n-bit binary number represented as:

$$C \rightarrow c_{n-1}\, c_{n-2} \cdots c_2 c_1 c_0 \qquad (10.a)$$

and its value as:

$$C = \sum_{j=0}^{n-1} a_i (2)^j \qquad (10.b)$$

1. This form has already been used in[13].

with $c_j \in (0_1 1)$ and $0 \le j \le n-1$.

Separating even and odde positioned terms gives

$$C = \sum_{j\ \text{even}} c_j(-2)^i - \sum_{j\ \text{odd}} c_j(-2)^j \qquad (11)$$

Addition and subtraction of $\sum_{j\ \text{odd}} \bar{c}_j(-2)^j$ and simple concatenation of terms leads to

$$C = \sum_{j=0}^{n-1} c_j^* (-2)^j - \sum_{j=1}^{r} (-2)^{2j-1} \qquad (12)$$

where

$$c_j^* = c_j \text{ for } j \text{ even} \qquad (13.a)$$

$$= \bar{c}_j \text{ for } j \text{ odd} \qquad (13.b)$$

and r is the same as given by (5).

The addition and subtraction of $\sum_{j\ \text{even}} \bar{c}_j (-2)^j$ to (11) reduces it to the form

$$C = -\sum_{j=0}^{n-1} c_j^{**} (-2)^j + \sum_{j=0}^{s} (-2)^{2j} \qquad (14)$$

with

$$C_j^{**} = c_j \text{ for } j \text{ even} \qquad (15.a)$$

$$= c_j \text{ for } j \text{ odd} \qquad (15.b)$$

and s is the same as given by (9).

III. Negative Negabinary Addition Using Binary Adders

The results of preceding section essentially constitute the basis for all succeeding sections of this paper. The important relations 6, 7, 12, and 14 associating binary and negabinary terms or expressions form the heart of this paper. Different ways of their utilization or unlike interpretation of these equalities enables adders for binary system to be used for negabinary numbers or vice versa. To consider the adoption of binary adders for the negative negabinary addition (n.n.b.a.), let A and B be two n-bit negabinary numbers to be added and A be the result of their n.n.b.a. Let A be given by (1) and B be represented as

$$B \rightarrow b_{n-1}\ b_{n-2} \cdots b_2 b_1 b_0 \qquad (16.a)$$

and its value

$$B = \sum_{i=0}^{n-1} b_i(-2)^i \qquad (16.b)$$

First Method[2]

Utilizing relation (6), the n.n.b. addition can be obtained as

$$Z = -(A+B) = -\left[\sum_{i=0}^{n-1} a_i^* (2)^i + \sum_{i=0}^{n-1} b_i^* (2)^i\right] + \sum_{i=1}^{r} (2)^{2i} \qquad (17)$$

Adding one to both summation parts on the right-hand side and remembering $(2)^0=1$, we get

$$Z = -\left[\sum_{i=0}^{n-1} a_i^* (2)^i + \sum_{i=0}^{n-1} b_i^* (2)^i + 1\right] + \sum_{i=0}^{r} (2)^{2i} \qquad (18)$$

Now comparing the similarities in the formats of relations (18) and (7), it can be easily revealed that (18) is exactly the same form as (7) and the terms inside the square parenthesis of (18) can be said to be as equivalent to $\sum_{i=0}^{n-1} z_i^{**} (2)^i$ summation. This term can be directly obtained by adding $\Sigma a_i^* (2)^i$ and $\Sigma b_i^* (2)^i$ with the help of a binary adder and by selecting the carry-in at the lowest significant bit position as "1". Once $\sum_{i=0}^{n} z_i^{**}(2)^i$ is evaluated, obtaining Z is just a matter of inverting even positioned bits. This leads to a simple logic realization of n.n.b.a. using binary adder and is shown in Fig. 1.

Second Method

An alternative is to utilize relation (7) to express A and B in their second alternative form and this leads to n.n.b.a. as:

$$Z = -(A+B) = \left[\sum_{i=0}^{n-1} a_i^{**} (2)^i + \sum_{i=0}^{n-1} b_i^{**} (2)^i\right] + \sum_{i=1}^{s+1} (2)^{2i-1} \qquad (19)$$

Now, relations (19) and (6) can be compared to conclude that the term inside the square parenthesis of (19) is simply $\sum_{i=0}^{n} z_i^* (2)^i$ and can be obtained by adding $\sum_{i=0}^{n-1} a_i^{**} (2)^i$ and $\sum_{i=0}^{n-1} b_i^{**} (2)^i$ numbers by n-bit binary adders. The result of summation can be properly complemented to produce the correct sum. The logic circuit arrangement is shown in Fig. 2.

2. This has already been discussed in[13]. It is included here for the completeness of this paper.

From the logic diagrams of Figs. 1 and 2, it can be easily said that the two methods are complementary to each other and this is included in Example 1 to emphasize the practicality of the algorithms and the close relations of binary addition and n.n.b.a.

IV. Binary Addition with n.n.b. Adders

In the previous section we considered how to augment the usefulness of binary adders to simulate n.n.b.a. This has been undertaken because of the dominance of +2 base system and the nonavailability of n.n.b.a. I.Cs. There is no controversy about the fact that -2 system is inferior to +2 base system and it is quite unlikely that -2 base system will ever be given preference (except for special purpose application). But still, at least from academic inter viewpoint, it is worth considering the feasibility of binary addition to be performed by n.n.b.a. Surprisingly, such an addition is seen to be possible. A remarkable symmetry between the results of Section III and IV is worth noticing and also poses an open question about negabinary number systems.

To consider the generalization of negabinary adders, let C and D be 2 n-bit binary numbers and Y be the result of their summation. Let C be given by (10) and D be represented as

$$D \rightarrow d_{n-1}\, d_{n-2} \cdots d_2 d_1 d_0 \tag{20.a}$$

and its value by

$$D \sum_{i=0}^{n-1} d_i (2)^i \tag{20.b}$$

First Method

Utilizing relation (12), the sum Y can be written as

$$Y = \left[\sum_{j=0}^{n-1} c_j^* (-2)^j + \sum_{j=0}^{n-1} d_j^* (-2)^j \right] - 2 \sum_{j=1}^{r} (-2)^{2j-1} \tag{21}$$

Remembering $(-2)^0=1$ and adding this to the two parts of the above relation gives

$$Y = -\left[-\sum_{j=0}^{n-1} c_j^* (-2)^j - \sum_{j=0}^{n-1} d_j^* (-2)^j \right] + \sum_{j=0}^{r} (-2)^{2j} \tag{22}$$

This makes the form of relation (22) exactly similar to (14) and once $\sum_{j=0}^{n} y_j^{**} (-2)^j$ is obtained then finding Y is straight-forward. This summation term can be obtained by n.n.b. addition of $\sum_{j=0}^{n-1} c_j^* (-2)^j$ and $\sum_{j=0}^{n-1} d_j^* (-2)^j$. This leads to a simple logic diagram for performing binary addition and such a simple scheme employing only n.n.b.a. is shown in Fig. 3.

Second Method

The second solution is to use relation (14) as the starting strategy and this yields to:

$$Y = \left[-\sum_{j=0}^{n-1} C_j^{**} (-2)^j - \sum_{j=0}^{n-1} d_j^{**} (-2)^j \right] - \sum_{j=1}^{s+1} (-2)^{2j-1} \tag{23}$$

The relation (23) matches the form of (12) and the first term on the right-hand side of (23) can be said as similar $\sum_{j=0}^{n} y_j^* (-2)^j$ and can be obtained by n.n.b. addition of $\sum_{j=0}^{n-1} c_j^{**}(-2)^j$ and $\sum_{j=0}^{n-1} d_j^{**}(-2)^j$. This gives an alternative scheme for adding two binary numbers while n.n.b. adders are to be used as a basic adding element. The logic diagram is shown in Fig. 4. An example for such an addition utilizing both the techniques is given in Example 2.

Another glance at Figures 1 to 4 reveals that the schemes of Fig. 1 become exactly identical to Fig. 3 if only the type of adder is changed. The same property is also true for Figs. 2 and 4. This makes us feel that +2 and -2 base systems are much closer than they ever were thought to be. Also, the hardware interchange compatibility consideration inspires us to think that negative addition is a more primitive function in the base -2 system than the simple addition of two negabinary numbers.

V. Multiple Operand Negative Addition of Negabinary Numbers

In Section III, we have considered negative addition of two negabinary numbers and two different ways of adopting n-bit binary adders for such addition have been shown in Figs. 1 and 2. The design strategy behind these schemes revolves around relations (6) and (7). The basic idea is to write two negabinary numbers as numerically equivalent to two binary numbers of the form given by (6) and to obtain negative addition in the form of (7). On the contrary, if we start with the relation (7), we reach at the result in the form of (6). This is possible only because we add two numbers that changes the second summation term from odd powers of 2 to even powers of 2 or vice-versa and the change of sign, from one form to another is automatically taken care of by negative addition. Similar but slightly modified techniques

can be utilized to add multiple negabinary numbers (more than 2). But our efforts must always be directed towards arriving at any one of the two forms amongst (6) or (7) and this puts a restriction on the number of operands to be only some power of 2. This keeps the complexity of multiple addition of negabinary numbers within reasonable limits and the overhead does not mar the effectiveness of binary adders used for this purpose.

The multiple addition has emerged as an important function because of its straight-forward application to multiplication operation. Recent increased interest in this area is concerned with devising a simple algorithm for multiple addition of 2's complement numbers [15] and this further justifies inclusion of this section. But here, there is an additional target to use binary adders for multiple negabinary addition. To start these considerations, let m negabinary numbers A, B, ..., etc. are to be added with m defined as

$$m = 2^k \tag{24}$$

An additional information crucial to m-operand negative negabinary addition is whether k is an even number or an odd integer and based on this, different ways of performing multiple negative addition are taken up next.

Case 1: k odd; 1st Algorithm:

Using the number in the form of (6), the negative addition of n-negabinary numbers, A, B, ..., etc. gives

$$\begin{aligned} Z &= -(A+B+\ldots) \\ &= -\left[\sum_{i=0}^{n-1}(a_i^{*}+b_i^{*}+\ldots)(2)^i\right] + \sum_{i=1}^{r}(2)^{k+2i-1} \\ &= -\left[\sum_{i=0}^{n-1}(a_i^{*}+b_i^{*}+\ldots)(2)^i + \sum_{i=0}^{(k-1)/2}(2)^{2i}\right] \\ &\quad + \sum_{i=0}^{(k+2r-1)/2}(2)^{2i} \end{aligned} \tag{25}$$

The relation (25) is of the same format as of relation (7) and once $\sum_{i=0}^{k+n-1} Z_i^{**}(2)^i$ has been computed, Z can be directly obtained. The value of the summation Z-terms here are slightly different than in section III and contains an addition term of $\sum_{i=0}^{(k-1)/2}(2)^{2i}$. This can be called as a correction factor [15]. The steps for 2^k operand (with k as odd integer) negative addition of negabinary numbers can be given as

(a) Complement all add positioned bits of each 2^k negabinary numbers.

(b) Obtain $\sum_{i=0}^{(k-1)/2}(2)^{2i}$ as a correction factor.

(c) Add all the resultant numbers by using binary adders and considering and treating all the numbers as if they are binary numbers.

(d) After evaluating the final binary sum, complement all even positioned bits to obtain the negative negabinary sum, Z.

Case 2: K odd; 2nd Algorithm

The application of (7) for each of m negabinary numbers A, B, ..., etc. gives

$$\begin{aligned} Z &= -(A+B\ldots) \\ &= \sum_{i=0}^{n-1}(a_i^{**}+b_i^{**}+\ldots)(2)^i - \sum_{i=0}^{s}(2)^{(k+2i)} \\ &= \left[\sum_{i=0}^{n-1}(a_i^{*}+b_i^{*}+\ldots)(2)^i + \sum_{i=1}^{(k=1)/2}(2)^{2i-1}\right] \\ &\quad - \sum_{i=1}^{(k+2s+1)/2}(2)^{2i-1} \end{aligned} \tag{26}$$

The relation (26) is exactly similar to (6) and thus Z can be easily obtained. This again needs a correction factor of $\sum_{i=1}^{(k-1)/2}(2)^{2i-1}$ to be added to the usual process. Thus, the steps for obtaining negative addition of 2^k negabinary numbers are similar to (24) and can be given as

(a) Complement all even positioned bits of each 2^k negabinary numbers.

(b) Obtain $\sum_{i=1}^{(k-1)/2}(2)^{2i-1}$ as a correction factor.

(c) Add all the resultant numbers by using binary adders.

(d) After evaluating the final binary sum, complement all odd positioned bits to obtain Z.

Examples of multiple operand addition using these two techniques, are given in Example 3.

Case 3: k even

When k is even, the type of summation terms of (6) remain unchanged. This means that odd powers of 2 continue to be augmented odd powers of 2 and as done in case 1, the inclusion of correction factor simply covers the missing odd powers of two. So an additional correction factor with all odd powers of two has to be added and this makes the process a bit complex but not impossible. The correction factor for the second algorithm can be obtained in the same way. At this point it is felt that the length of this paper precludes any further elaborations.

Multiple operand addition of binary numbers using negative negabinary adders can be achieved exactly in the same way and therefore only discus-

sion on such addition is intentionally avoided and the details can be easily worked out.

VI. Negabinary to Binary Conversion

A careful observation of relations (6), (7), (12), and (14) leads to four different algorithms for number conversion from negabinary to binary systems. Depending on interpretations, two of them need arithmetic to be performed in the binary system and the other two require negabinary arithmetic. These algorithms are considered one after another.

Method 1: Rewriting (6) once again:

$$A = \sum_{i=0}^{n-1} a_i^* (2)^i - \sum_{i\ odd} 2^i \qquad (27)$$

Here, A the negabinary number is given and it is our objective to obtain its binary equivalent. The right-hand side terms of (27) are purely binary numbers and their computation gives the desired binary number. This algorithm for conversion has been discussed[16,14] and it is included here for the completeness of the text. The steps are:

Step (i) First summation term of (27) can be obtained by complementing odd-positioned bits of A.

Step (ii) From this, subtract a number 0101...10 in base +2 (by using binary subtractor) to obtain the desired number.

Method 2: Having a careful look on relation (7) reveals that right-hand side summation terms are again purely binary. Hence the conversion process directly follows:

Step (i) First summation term is obtained by complementing all even positioned bits of A.

Step (ii) Subtracting this number (in +2 base) from 0....101 results in the desired binary number.

Method 3: This method is based on relation (12) and needs negabinary subtraction. As we have to convert a negabinary number to a binary number, we can assume that C', the negabinary equivalent of C is given and can be written as

$$C' = \sum_{i=0}^{n-1} c_i^* (-2)^i - \sum_{i\ odd} (-2)^i \qquad (28)$$

Addition of the second summation term to C' (in -2 base system arithmetic to be used) gives us the first summation term of (12). This needs complementation of odd positioned bits and results into the desired binary number. So the steps are[14]:

Step (i) Add 0101...010 term to the given negabinary number, the addition being performed in -2 base.

Step (ii) Complement all odd positioned bits of the result.

Method 4: This method is based on relation (14) and the algorithmic steps can be given as:

Step (i) Subtract the given negabinary number from 1010..1 (in -2 base).

Step (ii) Complement all even positioned bits to get the binary equivalent.

VII. Binary to Negabinary Conversion

The last section gives us enough background to consider the reverse conversion process. Similar arguments can be given to come up with four algorithms. No detailed explanation is given here and only the steps are indicated.

Method 1[16,14]:

Step (i) Add $\sum_{i\ odd} 2^i$ i.e. 1010...10 to the given binary number (using binary adder)

Step (ii) Complement all odd positioned bits to obtain the equivalent negabinary number.

Method 2:

Step (i) Subtract the given binary number from $\sum_{i\ even} 2^i$ (i.e. 0101...01) using binary subtractor.

Step (ii) Complement all even positioned bits.

Method 3:

Step (i) Complement all odd positioned bits of the given binary number.

Step (ii) Subtract $\sum_{i\ odd} (-2)^i$ i.e. 1010...10 from the above result. Use negabinary arithmetic.

Method 4:

Step (i) Complement all even positioned bits of the given binary number.

Step (ii) Subtract this number from $\sum_{i\ even} (-2)^i$. Use negabinary arithmetic.

VIII. Concluding Remarks

A new concept of close relationships between binary addition and -2 base negative addition has been introduced in this paper. The hardware interchangeability, multiple additions and convenient conversion processes of the two bases substantiate this concept and makes us feel that negabinary negative addition is the primitive operation and can be said to be functionally equivalent to addition of binary numbers. It is also believed that the exposure of the results reported in this paper will have profound effect and will advance the use of negabinary system for the design of more complex digital hardware. It will also encourage the design of a special purpose hardware with a mixed use of both binary and negabinary systems so that the advantages inherent in each system can be intelligently utilized for logic implementation of different sections associated with the system.

Last, but not least, it is worth mentioning that some of the results presented in this paper, can be easily extended for any general value of

positive and negative radixes and probably it is worth investigating other possible inter-relations between positive and negative bases.

Acknowledgments

This work is partially supported by the Wayne State University Research Grant 167-1622 xxx 00210.

References

1. L.B. Wadel, "Negative base number systems," IEEE Trans. on Electronic Computers, vol. Ec-6, p. 123, June 1957.

2. M.P. de Regt, "Negative radix arithmetic," Computer Design, vol. 6, part 1 to part 8, May 1967-January 1968.

3. P.V. Sankar, S. Chakrabarti and E.V. Krishnamurthy, "Arithmetic algorithms in a negative base," IEEE Trans. on Computers, vol. C-22, pp. 120-125, February 1973.

4. D.P. Agrawal, "Arithmetic algorithms in a negative base," IEEE Trans. on Computers, vol. C-24, pp. 998-1000, October 1975.

5. D.P. Agrawal, "Negabinary carry-look-ahead adder and fast multiplier," Electronics Letters, vol. 10, pp. 312-313, July 25, 1974.

6. D.P. Agrawal, "Negabinary complex-number multiplier," Electronics Letters, vol. 10, pp. 502-503, November 14, 1974.

7. G.S. Rao, M.N. Rao and E.V. Krishnamurthy, "Logic design of a negative binary adder-subtractor," International Journal of Electronics, vol. 36, no. 4, pp. 537-542, 1974.

8. D.P. Agrawal, "Negabinary parallel counters," Digital Processes, vol. 1, pp. 75-85, Spring 1975.

9. D.P. Agrawal, "On negabinary division and square-rooting," Digital Processes, vol. 1, pp. 267-274, Autumn 1975.

10. S. Zohar, "New hardware realizations of non-recursive digital filters," IEEE Trans. on Computers, vol. C-22, pp. 328-338, April 1973.

11. S. Zohar, "The counting recursive digital filters," IEEE Trans. on Computers, vol. C-22, pp. 338-327, April 1974.

12. S. Zohar, "Fast hardware fourier transformation through counting," IEEE Trans. on Computers, vol. C-22, pp. 433-441, May 1973.

13. S. Murugesan, "Negabinary arithmetic circuits using binary arithmetic," IEEE Journal on Electronic Circuits and Systems, vol. 1, pp. 77-78, January 1977.

14. D.P. Agrawal and C.K. Yuen, "Comments on "A note on base-2 arithmetic logic" and author's reply," IEEE Trans. on Computers, vol. C-26, pp. 511-512, May 1977.

15. D.P. Agrawal and T.R.N. Rao, "On multiple operand addition of signed binary numbers," IEEE Trans. on Computers, vol. C-27, to appear November 1978.

16. C.K. Yuen, "A note on base -2 arithmetic logic," IEEE Trans. on Computers, vol. C-24, pp. 325-329, March 1975.

Example 1. It is desired to add $(7)_{-2}$ and $(-3)_{-2}$.

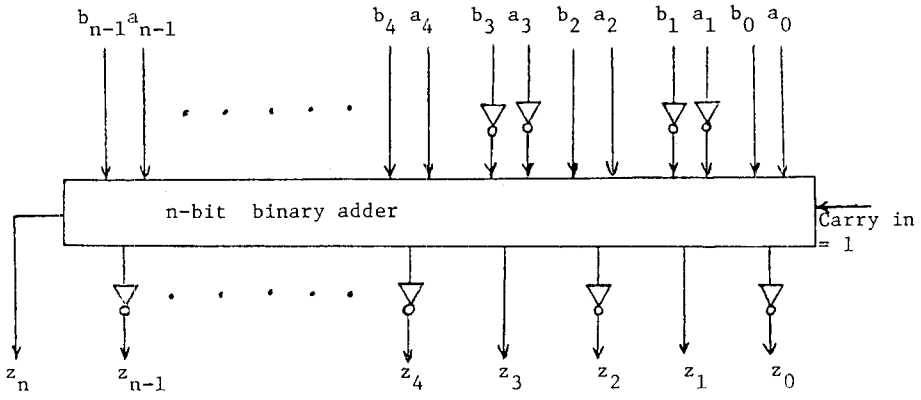
Number	First Method Complementing all odd positioned bits	Second Method Complementing all even positioned bits
$A = (7)_{-2}$ 110011	A* = 10001	A** = 01110
$B = (3)_{-2}$ 01101	B* = 00111	B** = 11000
Carry-in	1	0
Performing binary addition	Z** = 011001	Z* = 100110
Actual result by complementing even/odd bits	Z = 001100	Z = 001100

Example 2. Add $(5)_2$ and $(9)_2$.

Number	First Method Complementing all odd positioned bits	Second Method Complementing all even positioned bits
$C = (5)_2$ 0101	C* = 1111	C** = 0000
$D = (9)_2$ 1001	D* = 0011	D** = 1100
Carry in	1	0
Performing n.n.b.a	Y** = 11011	Y* = 0100
Complementing even/odd bits	Y = 01110	Y = 1110

Example 3. Add 5,-7,3,-2,1,-5,-8,-4 in negabinary system to obtain sum Z = 19.

Decimal number	Negabinary representation	First Method	Second Method
5	0101	1111	0000
-7	1001	0011	1100
3	0111	1101	0010
-2	0010	1000	0111
-1	0011	1001	0110
-5	1111	0101	1010
-8	1000	0010	1101
-4	1100	0110	1001
Correction factor		101	10
Binary Sum		1000010	0111101
Corrected Sum		0010111	0010111



Z=-(A+B)

Fig. 1. Logic diagram for negative negabinary addition (n.n.b.a.) using binary adder (for odd value of n)

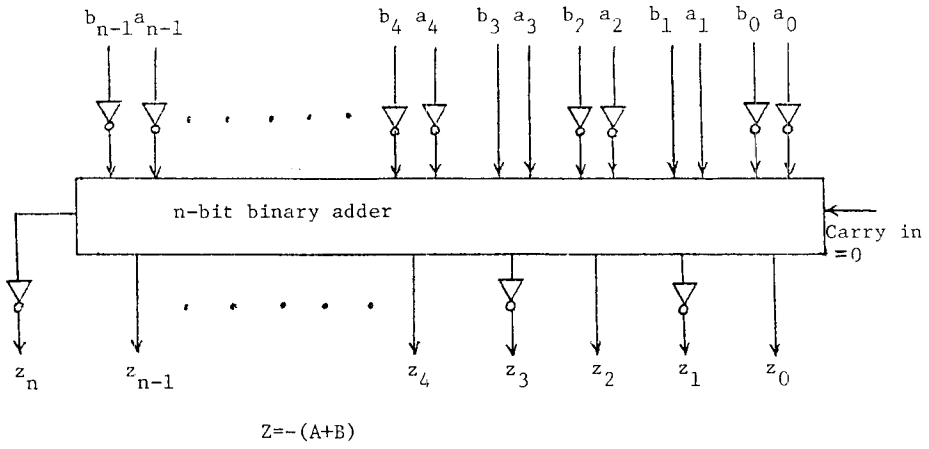


Fig. 2. Logic diagram for n.n.b.a. using binary adder - second method (for odd n)

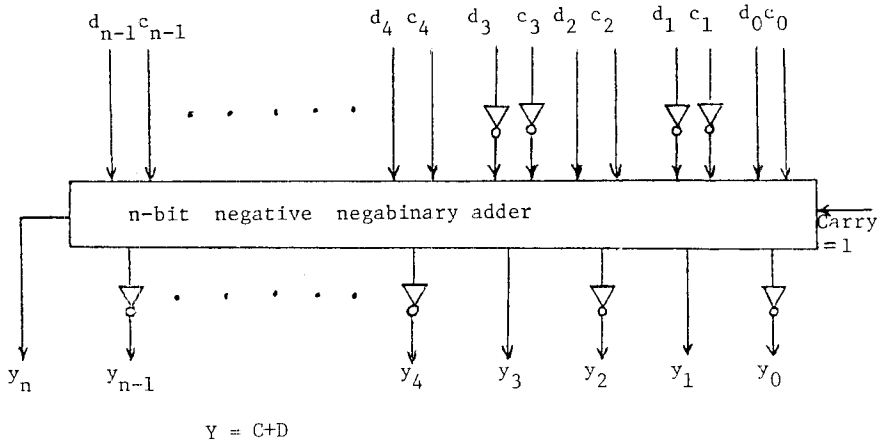


Fig. 3. Binary addition using n.n.b.a. - first method and odd value of n.

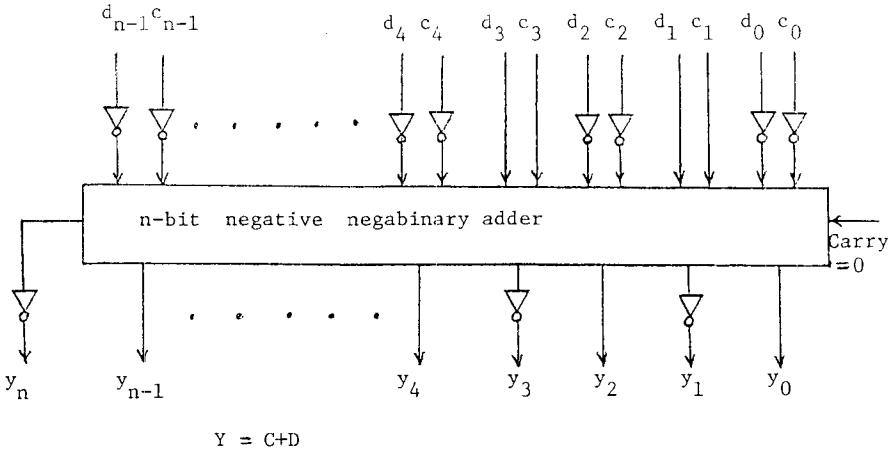


Fig. 4. Binary addition using n.n.b.a. - second method and odd n.