MIRROR ARITHMETIC

Jean P. Chinal

Abstract

Mirror coding for signed numbers is defined by means of a set of primitive powers of two $\{+2^n, -2^{n-1}, \ldots -2^0\}$ where signs of the usual set used in 2's complement representation are reversed. Use of the mirror representation is shown as an alternate design approach and is illustrated by a special purpose adder design in mirror code, by an alternate proof of a basic property of signed-digit arithmetic and as another interpretation of cells used in some array multipliers for signed numbers. Lastly, the concept is used to define a variable mode redundant coding, allowing simple sign-flipping without overflow.

1. Principle

Given a signed number x, its ordinary two's complement representation[1,4] is a binary word X with at least four distinct interpretations. For a (n + 1)-digit X (with sign bit) let us write:

$$X = a\, x_{n-1} \ldots x_i \ldots x_0 \quad (a, x_i \in \{0,1\}) \tag{1}$$

$$X' = 0\, x_{n-1} \ldots x_i \ldots x_0 \tag{2}$$

$$X'' = 0\, a\, x_{n-1} \ldots x_i \ldots x_0$$

Also, let $v(u)$ be the arithmetic value of Boolean u (arithmetic 1 for Boolean 1 and vice versa) and let us denote by x, x', and x'' the arithmetic values of X, X' and of X''. If $|x|_m$ is the residue of x modulo n, we have:

$$x = -v(a)2^n + x' \tag{3}$$

$$x' = |x|_{2^n} \tag{4}$$

$$x'' = +v(a)2^n + x' \tag{5}$$

$$x'' = |x|_{2^{n+1}} \tag{6}$$

$$x = v(a)(-2^n) + \sum_{i=0}^{n-1} v(x_i)2^i \tag{7}$$

Thus X may be seen as a residue modulo 2^n with appended sign a on the left, as a residue modulo 2^{n+1}, as a special signed-digit number (digit range: $\{-1, 0, +1\}$) where the -1 would be restricted to the leftmost position only or, still (3,7) of a weighted sum with coefficients in $\{0, +1\}$ but with weights taken from an inventory of powers of two of both signs, namely

$$I = \{-2^n, +2^{n-1}, \ldots +2^i, \ldots +2^0\} \tag{8}$$

We now turn our attention to this last representation, based upon (7) and (8). In the set I, only one power of two is negative, which is the least number allowed to represent signed numbers. However, representation of numbers in signed-digit[5-7] is, except for o, not unique and we may similarly try to use other powers of two to represent signed numbers. Doubling the inventory I, by adjoining to it all powers in it with signs reversed, would be the general signed-digit case. Let us constrain it for economy's sake and use only the set formed of the powers with signs reversed.

$$\hat{I} = \{+2^n, -2^{n-1}, \ldots -2^i, \ldots -2^0\} \tag{9}$$

Here, all powers except the highest, are negative. Let us use this set ("mirror set") to represent signed numbers, i.e. let us look for digits $\hat{x}_i$, such that

$$\hat{X} = \hat{a}\, \hat{x}_{n-1} \ldots \hat{x}_i \ldots \hat{x}_0 \quad (\hat{a}, \hat{x}_i \in \{0,1\}) \tag{10}$$

$$x = v(\hat{a})2^n - v(\hat{x}_{n-1})2^{n-1} \ldots - v(\hat{x}_i)2^i \ldots - v(\hat{x}_0)2^0 \tag{11}$$

where $\hat{X}$ may be called <u>mirror code</u> for x and its use, <u>mirror arithmetic</u>, owing to the name just given to I. (Fig. 2). Let us denote the (new) number $\hat{x}$ with digits $\hat{x}_i$ in ordinary binary.

2. Relation of Mirror Code and Ordinary Code

From (7) and (11) we deduce that

$$-[v(a) + v(\hat{a})]2^n + \sum_{k=0}^{i} [v(x_k) + v(\hat{x}_k)]2^k = 0$$
$$x + \hat{x} = 0 \tag{12}$$

and that digits $\hat{x}_i$ are those of the additive inverse[4] given by[2,4]

$$\begin{cases} \hat{x}_i = x_i \oplus r_i & (r_0 = 0) \\ r_{i+1} = x_i + r_i & \\ \quad = \sum_{k=0}^{i} x_k & \end{cases} \tag{13}$$

Thus going from I to $\hat{I}$ for a given number x, changes its representation to $\hat{X}$ associated with the additive inverse of x[2,4] also arithmetically obtainable by subtraction from zero or 2^{n+1} (inversion base[8]). Zero is unchanged, and non-zero numbers are transformed according to the well-known set of rules (13). These, although required for combinational inverters[4] (sign-flippers[4,9]) yet are actually rarely used because generation of the additive inverse is most frequently combined with a simultaneous addition or subtraction, which allows then to dispense with $\hat{X}$ and use only $\bar{X}$ (Boolean bit by bit complement) combined with a least significant weight "hot one"[4]. This is to say that although the Boolean relation between ordinary code and mirror code may be shown to be a known one (13), the hardware to switch has a non-negligeable cost, which, if avoided, implies to choose between the ordinary code and the mirror code. In this respect the situation is different from the distinction between positive and negative logic which amounts only to bitwise reinterpretation of digit values.

To use one system rather than the other, does not make great practical difference at the start. There is only a minute difference in the range of numbers represented:

$$x \in [-2^n, 2^n-1] = D \qquad (I)$$
$$x \quad [+2^n, -2^n+1] = \hat{D} \qquad (\hat{I}) \qquad (14)$$

$$D \cap \hat{D} = [-2^n+1, 2^n-1]$$
$$\bar{D} \cap \hat{\bar{D}} = \{+2^n\}$$
$$D \cap \bar{\hat{D}} = \{-2^n\}$$
$$D \cup \hat{D} = [-2^n, +2^n] \qquad (15)$$

For usual values of n, such differences may be neglected for a choice between the two systems, which then leads to staying with the existing one. These differences will be of interest, only if n is very small, which give them great relative value, or if for ordinary n, both systems are used in alternance within the arithmetic unit and logic of the computer.

3. One Bit Mirror Arithmetic

If n is made equal to 1, then the domaines D and $\hat{D}$ become significantly different (Fig. 3).

$$D = \{-2^1, -2^0, 0, 2^0 \quad \} \qquad (16)$$
$$\hat{D} = \{ \quad -2^0, 0, 2^0, 2^1\} \qquad (17)$$

Set $\hat{D}$, instead of D, will be useful whenever we wish to represent +2 with the same ease as -2, i.e., with two digits only. We now give two examples of application where introducing $\hat{I}$, $\hat{D}$ will provide an alternate approach to a problem.

3.1 Design of adder

Assume we want to design an adder implementing

$$x + y - v(r_o) \qquad (18)$$

Where x,y are (n+1)-digit numbers, r_o is o or 1 and, contrary to ordinary addition, minus sign precedes $v(r_o)$. Such adders may be used for instance to implement addition modulo $2^k + 1$. Then $x+y-v(r_o)$ may be obtained easily in mirror code, and then reconverted to ordinary code. The mirror code just plays then the role of a useful intermediate representation in the solution process, just as sometimes a change of positive logic to negative logic may be used to afford a convenient reasoning. Here consider first the ranks of x_o and y_o (least significant digits) of x and y. We must compute in mirror code

$$S_o = v(x_o) + v(y_o) - v(r_o) \qquad (19)$$

$x_oy_or_o$	$v(x_o)+v(y_o)-v(r_o)$	$s_2s_1s_o$	$\hat{s}_1\hat{s}_o$
0 0 0	0	0 0 0	0 0
0 0 1	$\bar{1}$	1 1 1	0 1
0 1 0	$\bar{1}$	0 0 1	1 1
0 1 1	0	0 0 0	0 0
1 0 0	1	0 0 1	1 1
1 0 1	0	0 0 0	0 0
1 1 0	2	0 1 0	1 0
1 1 1	1	0 0 1	1 1

Table 1

(Table 1: $s_2s_1s_o$: ordinary code
$\hat{s}_1\hat{s}_o$: mirror code)

$$v(x_o)+v(y_o)-v(r_o)=2^2s_2+2^1s_1+2^os_o=2^1\hat{s}_1-2^o\hat{s}_o$$
$$(s_2, s_1, s_o, \hat{s}_1, \hat{s}_o \in \{0,1\})$$

From Table 1, we deduce

$$s_o = x_o \oplus y_o \oplus r_o \qquad (20)$$

$$r_1' = \bar{x}_o y_o r_o + x_o \bar{y}_o \bar{r}_o + x_o y_o \bar{r}_o + x_o y_o r_o$$
$$= x_o y_o + x_o \bar{r}_o + y_o \bar{r}_o \qquad (21)$$

$$r_1' = \mathrm{maj}\,(x_o, y_o, \bar{r}_o) \qquad (22)$$

where maj (u,v,w) denotes the three-variable majority function. This can be seen[4] as a borrow logic with permuted inputs and combined with a code converter according to (13) yields a possible adder design (Fig.4). A more classical approach would be to write

$$x + y - v(r_o) = -[(-x) + (-y) + v(r_o)] \qquad (23)$$

which would require three sign-flippers [4,9] and an adder. Another would be to write

$$x + y - v(r_o) = x - (-y) - v(r_o) \qquad (24)$$

which would require one subtractor and a sign-flipper on the y input (Fig. 5) a solution with complexity similar to that of Fig. 4 but different. Finally, a solution with a layout close to the first one (Fig. 4) would be to use an ordinary adder with zero carry-in form x + y and conditionally subtract 1 with a combinatorial decrementer[4].

3.2 Signed-digit binary arithmetic

Consider two numbers x,y with

$$x = \sum_{i=0}^{m-1} x_i' 2^i \qquad\qquad x_i', y_i' \in \{\bar{1}, 0, 1\}$$
$$y = \sum_{i=0}^{m-1} y_i' 2^i \qquad\qquad (\bar{1} = -1) \qquad (25)$$

Mirror representation affords a more visual proof of the property[5] according to which generalized carry propagation may be restricted to three stages (two interstage intervals) by a suitable coding of interstage carries ("transfers") and use of three layers of cells which extend the usual half-adders[5]. To do this, we start from operators with two inputs in $\bar{1}$,0,1 and proceed as for an adder made of half-adders only[10-12]. We want to determine the nature of operators at each level and show that at level 3, carries out are identically zero, which limits the growth of the network to three levels, and consequently, the propagation length. Fig. 6 and 7 show the 0-carry, 1 and $\bar{1}$ carry conditions, for the different values of the sum $\bar{s}_i'$ of x_i', y_i' and incoming transfer t_i. We illustrate now on Fig. 9, 10 the evolution of sum digits and partial transfer terms (t_i', t_i'') as we go through levels A,B,C. At each level, use is made of Fig. 6 and 7, partially reproduced in Fig.9, 10. Fig. 9 illustrates the condition[4] that $(t_i', t_i'') \notin \{(1,1),(\bar{1},\bar{1})\}$ and Fig 10 uses this to show that at level C the sum s_{i-1}'' is in $\{\bar{1},0,1\}$ which can be coded with a zero most significant digit, i.e carries out of (C) cells are zero. ((a) cells generate carries whenever possible, (b) cells only when unavoidable[5]). Thus, partial sums $x'_i + y'_i$, $s'_i + t'_i$ and $s''_i + t''_i$ are coded with two digits, in ordinary mirror code, as indicated in Fig. 10 and carries out of (c) cells may be kept equal to zero.

3.3 Adders with base 2 inputs and base (-2) output.

Adders with ordinary base (+2) inputs may also be modified to have an output in signed-digit code or base (-2) code (Fig. 11, 12, 13). Cells are defined arithmetically in the corresponding graphs. In Fig 13, (a)and (d) cells are defined by the following tables

u v	value of u + v	direct code	mirror code	variable mode code
0 0	0	0 0	0 0	0;00
0 1	1	0 1	1 1	0;01
$\bar{1}$ 0	$\bar{1}$	1 1	0 1	1;01
$\bar{1}$ 1	0	0 0	0 0	1:00
		(d^0) cell	(d^1) cell	

u v	u + v	mirror code
0 0	0	0 0
0 1	1	1 1
1 0	1	1 1
1 1	2	1 0
		(a) cell

3.3 Variable mode arithmetic

Let us explicitly encode the convention (direct, mirror) used to represent the numbers, using an additional binary digit equal to zero for the direct code (D) and 1 for the mirror code (M).

Consider now two numbers in this code and their sum

$$x = Aax_{n-1}\dots x_i \dots x_0$$
$$y = Bby_{n-1}\dots y_i \dots y_0$$
$$x + y = Sss_n s_{n-1}\dots s_i \dots s_0$$

$$A,B,S,x_i,y_i,s_i,a,b,s \in \{0,1\} \qquad (26)$$

Addition in this variable mode code, where A,B and S explicitly represent the code used, requires to determine the rules for obtaining s_i at each rank i, together with S. To do this we consider the sum of digits at rank zero, for which we do not know yet whether to assume a positive or negative carry-in and try to encode it in direct or mirror code. Then we deduce the operation to be performed at a rank i.

At rank o we compute the sum of digits x_0, y_0, (with suitable signs due to A,B) and express it in direct or mirror code. Thus we have

$$D_0 = (-1)^{v(A)}v(x_0) + (-1)^{v(B)}v(y_0) = (-1)^{v(T)}v(r_1)+(-1)^{v(U)}v(s_0) \qquad (27)$$

and this entails a carry $(-1)^{v(T)}v(r_1)$ into rank 1, so that generally, for rank i we must execute:

$$D_i = (-1)^{v(A)}v(x_i) + (-1)^{v(B)}v(y_i) + (-1)^{v(T)}v(r_i) = (-1)^{v(T)}v(r_{i+1}) + (-1)^{v(U)}v(s_i) \qquad (28)$$

Values of A,B,S,T,U and the domains of D_0, D_i are given in Table 2. Allowed codes for 2-digit D_i are derived from Tables 3 (a-g).

Table 2. Ranges and allowed encodings

ABS	Domain of Do	TU	Domain of Di	Allowed Code
000	0,1,2	00	0,1,2,3	D
001	0,1,2	01	0,1,2,3	D
010	$\bar{1}$,0,1	10	$\bar{2}$,$\bar{1}$,0,1	D
011	$\bar{1}$,0,1	01	$\bar{1}$,0,1,2	M
100	$\bar{1}$,0,1	10	$\bar{2}$,$\bar{1}$,0,1	D
101	$\bar{1}$,0,1	01	$\bar{1}$,0,1,2	M
110	$\bar{2}$,$\bar{1}$,0	10	$\bar{3}$,$\bar{2}$,$\bar{1}$,0	M
111	$\bar{2}$,$\bar{1}$,0	11	$\bar{3}$,$\bar{2}$,$\bar{1}$,0	M

	D	M
2	10	10
1	01	11
0	00	00

[0,1,2]
(a)

	D	M
1	01	11
0	00	00
$\bar{1}$	11	01

[$\bar{1}$,0,1]
(b)

	D	M
0	00	00
$\bar{1}$	11	01
$\bar{2}$	10	10

[$\bar{2}$,$\bar{1}$,0]
(c)

	D	M
0	000	00
$\bar{1}$	111	01
$\bar{2}$	110	10
$\bar{3}$	101	11

[0,$\bar{1}$,$\bar{2}$,$\bar{3}$]
(d)

	D	M
3	11	101
2	10	110
1	01	111
0	00	000

[0,1,2,3]
(e)

	D	M
1	01	111
0	00	000
$\bar{1}$	11	001
$\bar{2}$	10	010

[$\bar{2}$,$\bar{1}$,0,1]
(f)

	D	M
2	010	10
1	001	11
0	000	00
$\bar{1}$	111	01

[$\bar{1}$,0,1,2]
(g)

Tables 3 (a-g). Direct and mirror codes in D_0 and D_i number ranges.

The output code then must be chosen so that U equals S. The second and seventh lines of Table 2 may be deleted as then lead to contradiction between assumed value of S and allowed code in the last column and we have now the reduced Table 4:

Table 4. Bitwise operations.

A B	Bit operation	Output Code
0 0	$v(x_i) + v(y_i) + v(r_i)$	D
0 1	$v(x_i) - v(y_i) - v(r_i)$	D
	$v(x_i) - v(y_i) + v(r_i)$	M
1 0	$-v(x_i) + v(y_i) - v(r_i)$	D
	$-v(x_i) + v(y_i) + v(r_i)$	M
1 1	$-v(x_i) - v(y_i) - v(r_i)$	M

For the direct code (D) we have to implement ordinary addition sum and carry logic

1-4(line 1) or subtraction logic[2,4](lines 2 and 4). For the mirror mode we need a logic which is the same as the one studied above in example 3.1 (up to relabeling of variables), in lines 3 and 5. For the last line it can be checked that this is the same as addition logic. The sum digit is the same in all cases. The carry digit then is given by Table 5.

Table 5. Carries for variable mode addition.

A B	Carry	S
0 0	maj (x_i, y_i, r_i)	0
0 1	maj $(\overline{x_i}, y_i, r_i)$	0
	maj $(x_i, \overline{y_i}, r_i)$	1
1 0	maj $(x_i, \overline{y_i}, r_i)$	0
	maj $(\overline{x_i}, y_i, r_i)$	1
1 1	maj (x_i, y_i, r_i)	1

Thus the output mode is variable if we insist on single-bit carries and there is some choice for it when A and B are different. We can use two types of carry only by a suitable choice of S. Then at most one fixed variable will have to be complemented when A,B vary. There are two such solutions (Tables 6)

Table 6. Solutions with fixed place complementation.

A B	Carry (I)	S	Carry (II)	S^1
0 0	maj (x_i, y_i, r_i)	0	maj (x_i, y_i, r_i)	0
0 1	maj $(\overline{x_i}, y_i, r_i)$	0	maj $(x_i, \overline{y_i}, r_i)$	1
1 0	maj $(\overline{x_i}, y_i, r_i)$	1	maj $(x_i, \overline{y_i}, r_i)$	0
1 1	maj (x_i, y_i, r_i)	1	maj (x_i, y_i, r_i)	1

Thus we have

$$(I)\begin{cases} s_i = x_i \oplus y_i \oplus r_i \\ r_{i+1} = \text{maj}\ (x_i \oplus A \oplus B,\ y_i, r_i) \\ S\ = A \end{cases}$$

$$(II)\begin{cases} s_i = x_i \oplus y_i \oplus r_i \\ r_{i+1} = \text{maj}\ (x_i,\ y_i \oplus A \oplus B,\ r_i) \\ S\ = B \end{cases} \qquad (29)$$

Addition may be performed with an ordinary adder logic and conditionally complementing x (bitwise) if mode of x is kept for the sum and bitwise complementing y if mode of y is used for the output. The condition is whether A and B are different. The operation on r_{i+1} if e.g. $x_i \oplus (A \oplus B)$ is performed outside the adder stage is similar to condition switching between add and subtract when we perform $x_- + (-1)^{v(C)}y$ with an adder: $y_i \oplus C$ (or $\overline{C}y_i + C\overline{y}_i$) has to be performed for all stages. Here the condition variable has to be obtained from A and B (one gate). Furthermore, substraction may now be obtained by simply changing the mode of the subtrahend y, which forms its complement on a single bit basis. If conditional add-subtract is desired of the form $x+(-1)^{v(C)}y$, then the above expressions for s_i are unchanged, and for the carries we have

$$r_{i+1} = \text{maj}(x_i \oplus \overline{A} \oplus B \oplus C, y_i, r_i);\ S=\overline{\overline{A}}=A$$
$$r_{i+1} = \text{maj}(x_i, y_i \oplus A \oplus B \oplus C, r_i);\ S=B \qquad (30)$$

which is similar to ordinary add-subtract, the only difference being that the condition variable is $A \oplus B \oplus C$ (or its complement) which may be synthesized in one or two gates. Also, we may perform without difficulty the more complex operation

$$(-1)^{v(D)}\ x + (-1)^{v(C)}\ y \qquad (31)$$

which in ordinary binary requires either to exclude the case where both D and C are one, or to use additional incrementing or complementing logic[4].

Thus with variable mode arithmetic a slightly redundant code (1 bit) is used. Addition and addition-subtraction may be kept similar in complexity to ordinary conditional add-subtract. Furthermore, the domain of numbers is now totally symmetrical $(-2^n, +2^n)$. Complementation is then a one-bit operation without overflow and the quantity $-(x + y)$ (called cosum[15] in the case of base minus 2,[12-15]) may be obtained as easily as $x + y$. The possibility of using cosums instead of sums in addition, multiplication and division, which appeared as a possible advantage[15] of base minus two versus ordinary binary, thus appears to be applicable also to variable mode arithmetic and without the range dissymetry of base (-2) representation.

3.4 Cellular multiplier

In the Pezaris[16] cellular multiplier for signed-numbers cells of several types are needed, some of which reduce to ordinary adders, others requiring new logic for the derivation of which a special formalism is introduced. For 1-bit cells the following operators are necessary[4,16]

$$\text{(a)}\ v(x_0) + v(y_0) + v(r_0) = 2^1 v(r_1) + 2^0 v(s_0)$$
$$\text{(b)} -v(x_0) + v(y_0) + v(r_0) = 2^1 v(r_1) - 2^0 v(s_0)$$
$$\text{(c)}\ v(x_0) - v(y_0) - v(r_0) = -2^1 v(r_1) + 2^0 v(s_0)$$

Cells (a) and (c) can be reduced to ordinary add and subtract stages respectively, while (b) is typically a cell as studied in 3.1 and 3.3 above, with output in mirror code.

For 2-bit cells[4,16] one is an ordinary 2-bit adder, the other a mirror code 2-bit adder and the third may be viewed as a cascade of both types for 1-bit length.

References

1. I. Flores, The Logic of Computer Arithmetic, Prentice Hall, 1963.

2. N. R. Scott, Electronic Computer Technology, McGraw Hill, New York, 1970.

3. R. K. Richards, Digital Design, Wiley, New York, 1971.

4. J. P. Chinal, Microsystèmes numériques, 270 p, ENSAe, Toulouse, 1972.

5. A. Avižienis, "Signed-digit Number Representation for Fast Parallel Arithmetic", IRE Transactions on Electronic Computers, Vol. EC-10, pp, 389-400, Sept. 1961.

6. A. Avižienis, "Digital Computer Arithmetic: a Unified Algorithmic Specification," Symposium on Computers and Automata, Pol. Inst. of Brooklyn (13-15 April 1971) pp. 509-525.

7. J. P. Chinal, Numérations redondantes pour ordinateurs, 100 p. ENSAe, Toulouse, 1972.

8. G. C. Langdon, "Subtraction by Minuend Complementation", IEEE Trans. Computers, Vol. C-18, No. 1, pp. 74-76, Jan. 1969.

9. J. P. Chinal, "Boolean Features of Sign-flipping and Signed-carry Logic in Base Minus Two", IEEE Trans. Computers, (to appear).

10. R. J. Mercer, "Microprogramming", JACM, pp. 157-171, April 1957.

11. B. V. Anisimov (Ed.) Vycislitelnaja tekhnika, Oborongiz, Moscow, 1963.

12. A. M. Oranski, A. M., Vycislitelnaja tekhnika, Izdatelstro Nauka i tekhnika, Minsk, 1964.

13. L. B. Wadel, "Negative Base Number Systems" IRE Trans. El. Computers, EC-16, p. 123, 6/57.

14. Z. Pawlak, "An Electronic Digital Computer Based on the Minus 2 System", Bulletin de l'Académie des Sciences, Série des Sciences Techniques, Vol. VII, No. 12, 1959, pp. 713-721.

15. M. P. de Regt, "Negative Radix Arithmetic", Computer Design, June 1967, pp. 53-63, July 1967, pp, 36-43, Dec. 1967, pp, 70-77. Jan. 1968, pp. 62-66.

16. S. D. Pezaris, "A 40ns 17 Bit by 17 Array Multiplier", IEEE Trans Computers, No. 24 vol. C-20, pp. 442-447, April 1971.

ACKNOWLEDGEMENTS

This work was done while the author was with UCLA Computer Science Department as a US-France exchange scientist. The support of National Science Foundation, Washington and of ENSAE, Toulouse, is gratefully acknowledged.

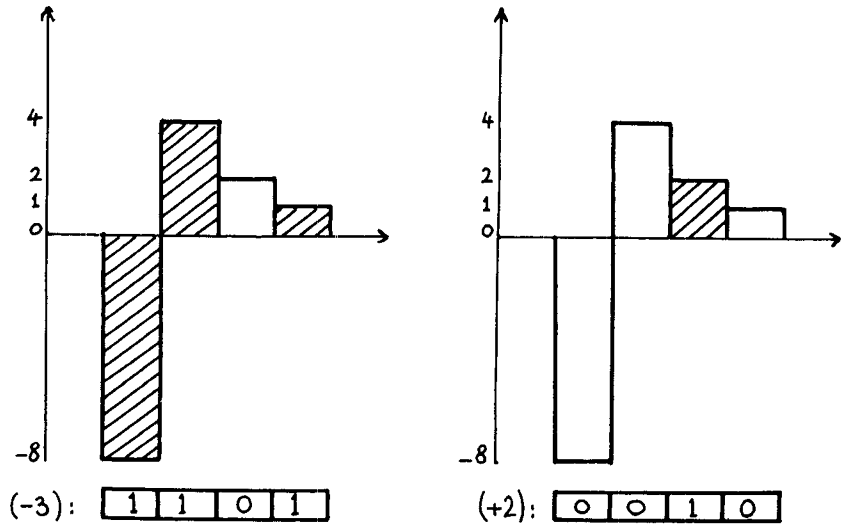

Fig 1. Coding of x and $\bar{x}$ using set $\{-2^3,2^2,2^1,2^0\}$

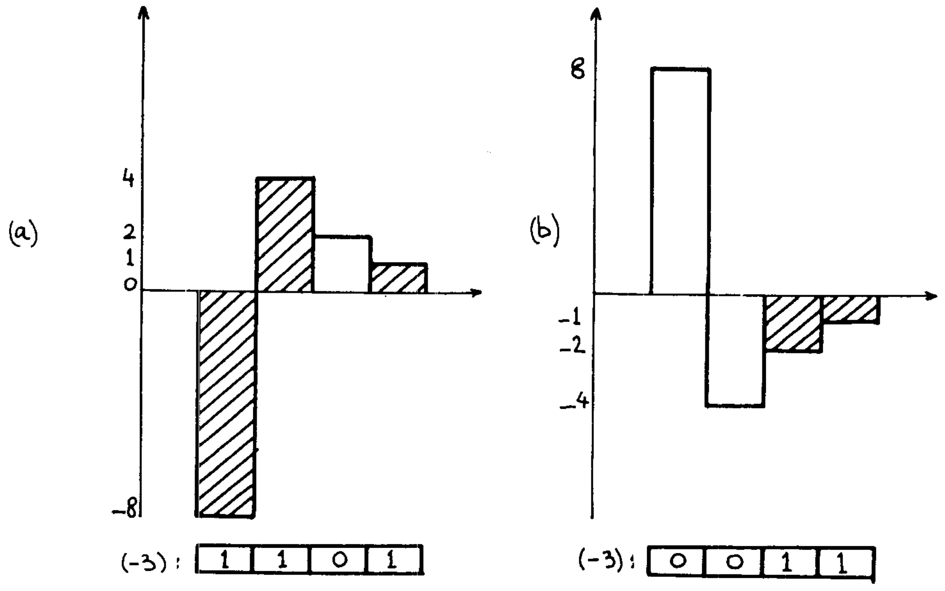

Fig. 2. Coding of x: (a) ordinary set I of primitive powers 2^i
(b) mirror image set $\hat{I}$
$I = \{-2^3,2^2,2^1,2^0\}$
$\hat{I} = \{+2^3,-2^2,-2^1,-2^0\}$

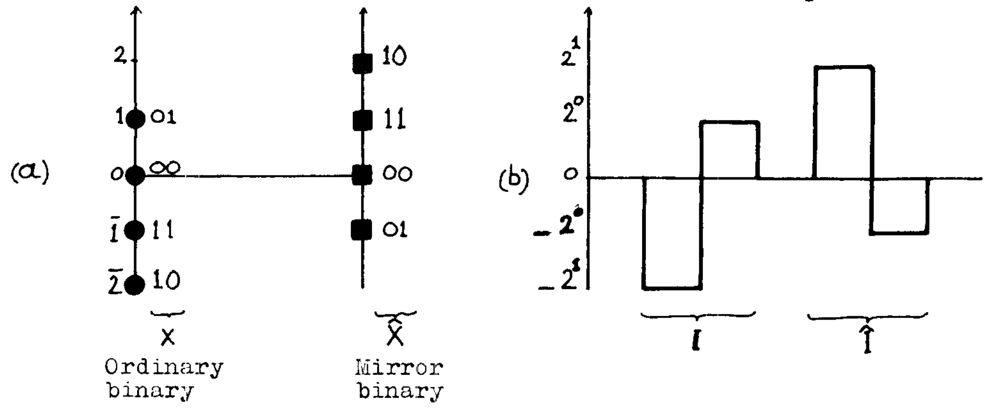

Fig. 3. One bit mirror and ordinary codes

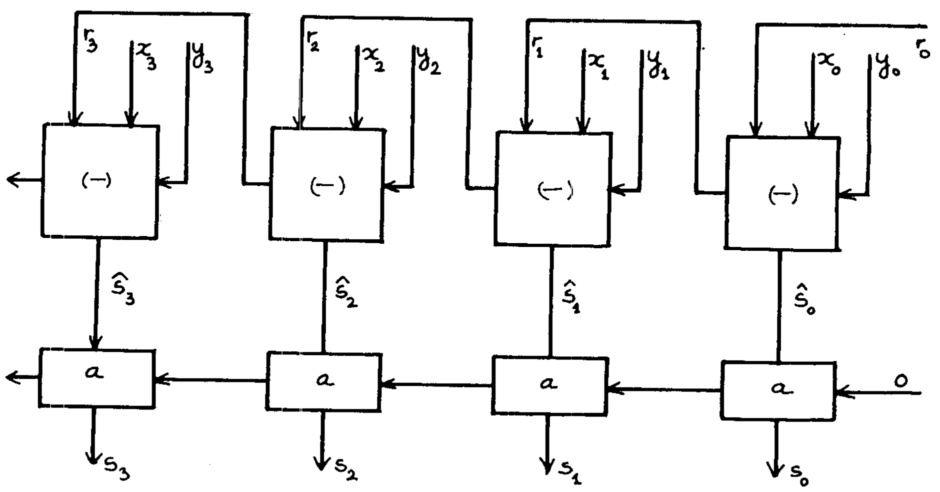

Fig. 4. Adder $x + y - v(r_0)$
$x = x_3x_2x_1y_0$
$y = y_3y_2y_1y_0$

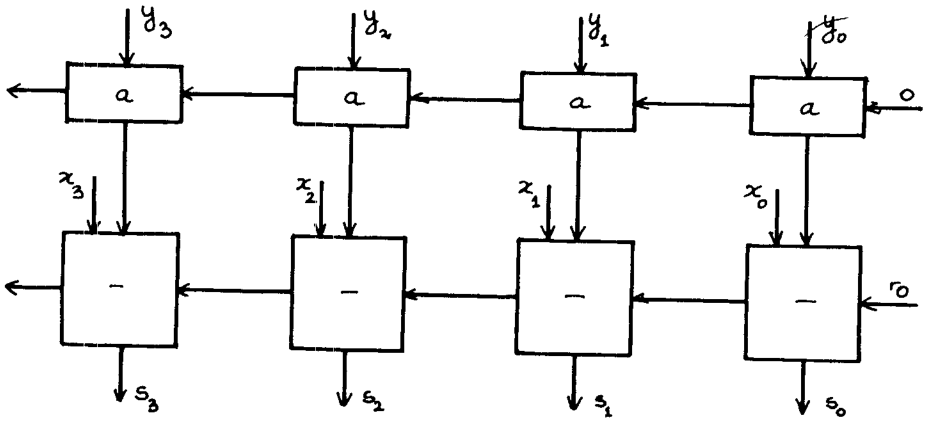

Fig. 5. Another design of adder for $x + y - v(r_0)$

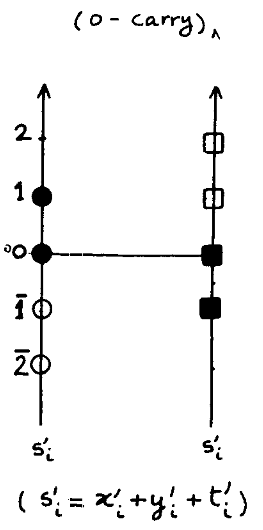

Fig. 6. No-carry condition

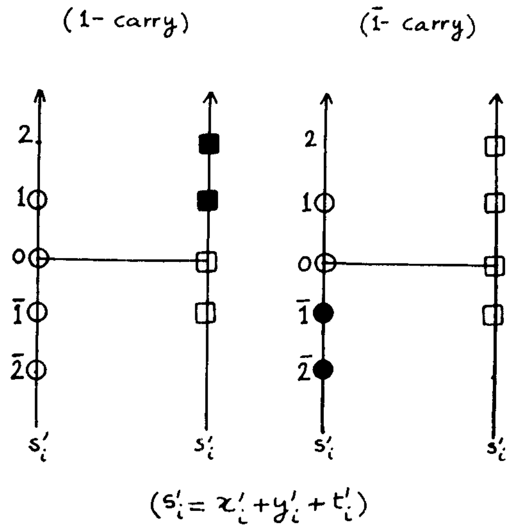

Fig. 7. 1 and $\bar{1}$ carry conditions

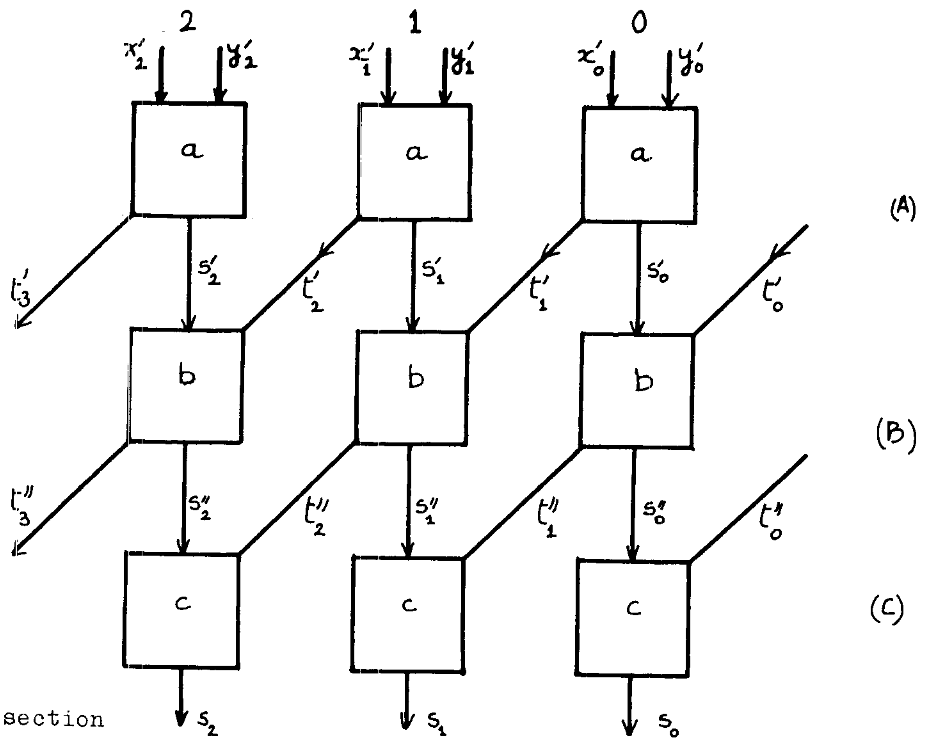

Fig. 8.

Signed digit adder section

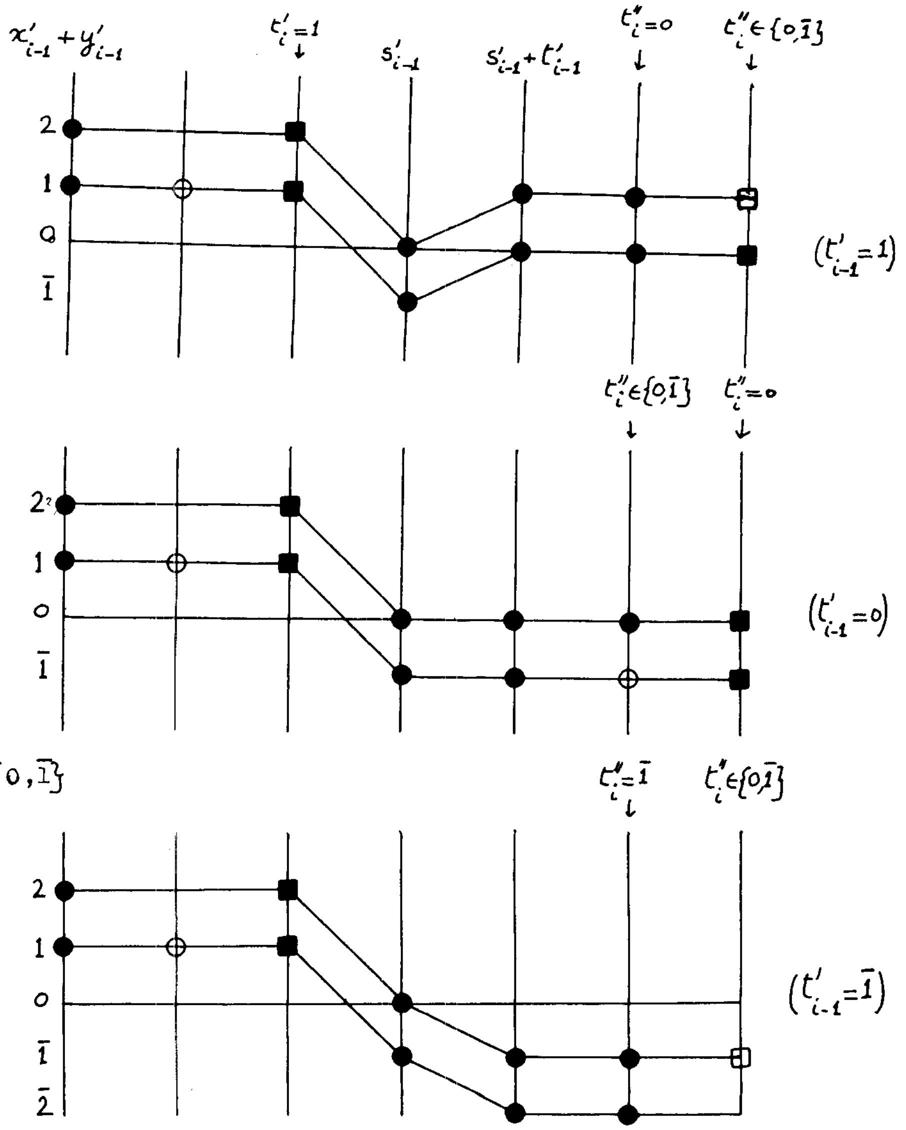

Fig. 9.

Verification of $t'_i = 1 \Rightarrow t''_i \in \{0,\bar{1}\}$

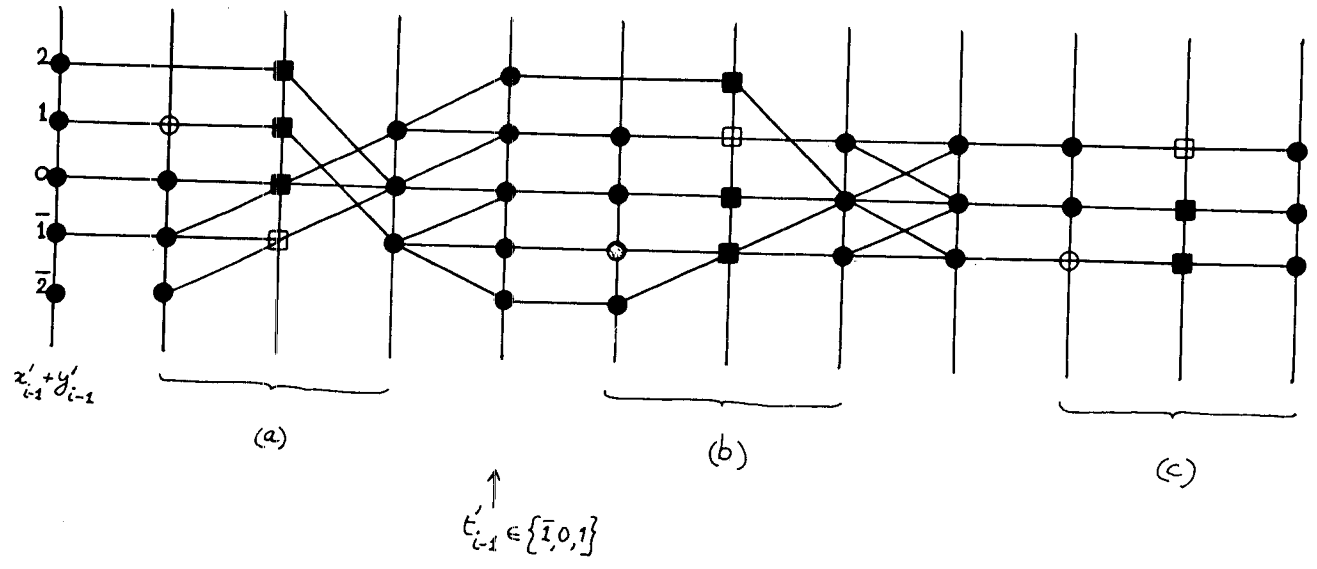

Fig. 10. Complete graph for 3-level adder

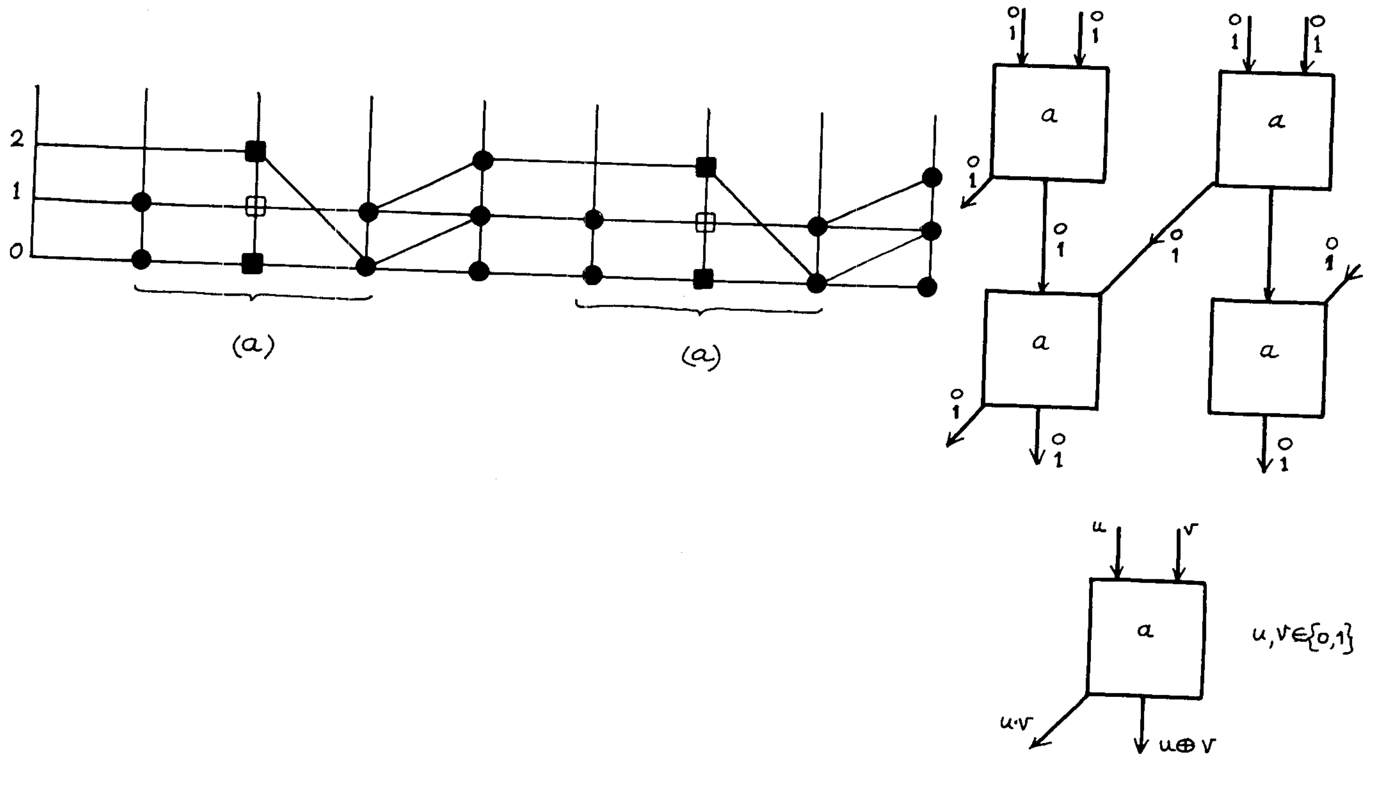

Fig. 11. Base 2 adder with half-adders (2 sections)

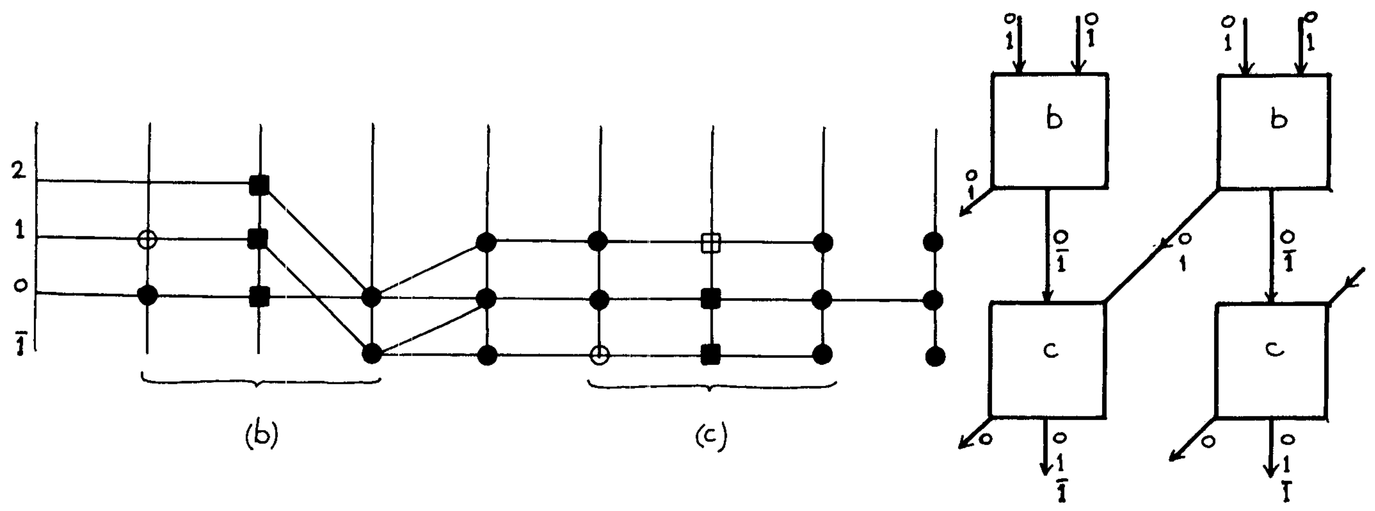

Fig. 12. Base 2 adder with signed-digit output

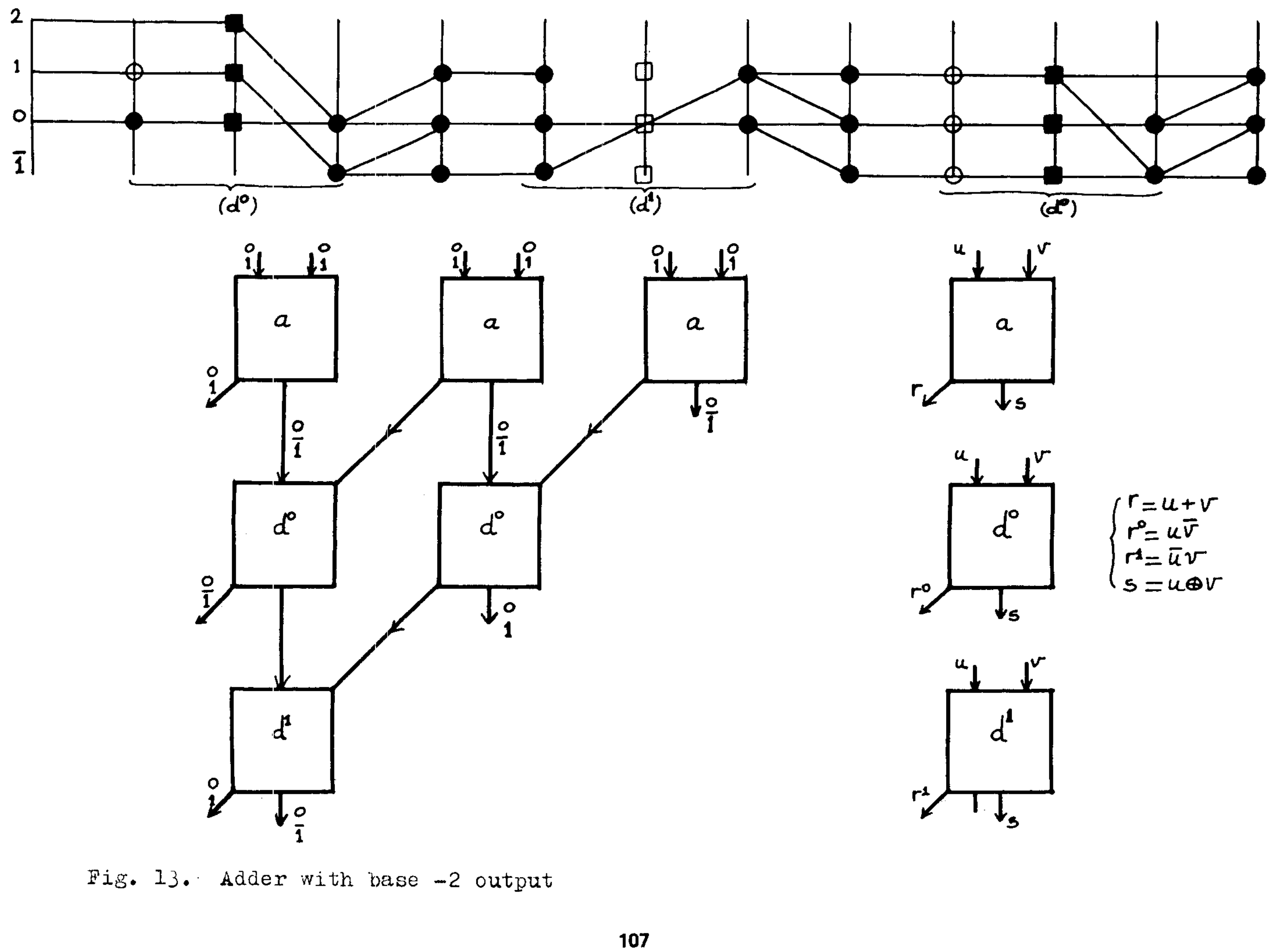

Fig. 13. Adder with base -2 output