digit is 0, subtract the multiplicand from the partial product (add its 2's complement).

(ii) If the multiplier digit under consideration is 0 and the previous lower order digit is 1, add the multiplicand to the partial product.

(iii) If the multiplier digit under consideration is the same as the previous lower order digit, do nothing.

It should be noted that a double length product register is required to hold the answer. The arithmetic operations, however, are performed on the most significant half only. Further, after each of the three possible operations above, the entire product register is shifted one place to the right, but the sign digit is retained.

To illustrate the algorithm, consider a multiplier of +9 (01001) and a multiplicand of −11 (10101) both given in 2's complements form.

Multiplicand Multiplier	1 0 1 0 1 0 1 0 0 1	
Initial content of product register	0 0 0 0 0	0 0 0 0 0
Subtract the multiplicand	0 1 0 1 1	0 0 0 0 0
Shift one position to the right	0 0 1 0 1	1 0 0 0 0
Add the multiplicand	1 1 0 1 0	1 0 0 0 0
Shift one position to the right	1 1 1 0 1	0 1 0 0 0
Shift one position to the right	1 1 1 1 0	1 0 1 0 0
Subtract the multiplicand	0 1 0 0 1	1 0 1 0 0
Shift one position to the right	0 0 1 0 0	1 1 0 1 0
Add the multiplicand	1 1 0 0 1	1 1 0 1 0
Shift one position to the right	1 1 1 0 0	1 1 1 0 1

The product (1110011101) is equal to −99 in 2's complement form.

6.8 BINARY DIVISION CIRCUITS

6.8.1 Introduction

It was mentioned earlier that multiplication can be reduced to a series of additions. Similarly division can be reduced to a series of subtractions. In fact even the comparison process can be achieved by subtracting and noting the sign of the answer. Division can also be accomplished by multiplying the dividend by the reciprocal of the divisor. The reciprocals can be read from a reference table. Division occurs less frequently than multiplication, and therefore a slow speed can be tolerated. Roughly speaking, division takes four times as long as multiplication, depending on the technique used.

In the binary division described in Section 6.3.3, the divisor is compared with the part of the dividend (or current partial remainder) above it. If the divisor is greater, a 0 is placed in the quotient, no subtraction occurs, and the dividend is shifted one place left (or the divisor is shifted right). If the divisor is smaller, a 1 is placed in the quotient, the divisor is subtracted, and the dividend is shifted one place to the left. This can be shown by means of a flow diagram as in Fig. 6.32.

6.8.2 Restoring Division

In the algorithm of Section 6.8.1, a comparison of the divisor and dividend was assumed.

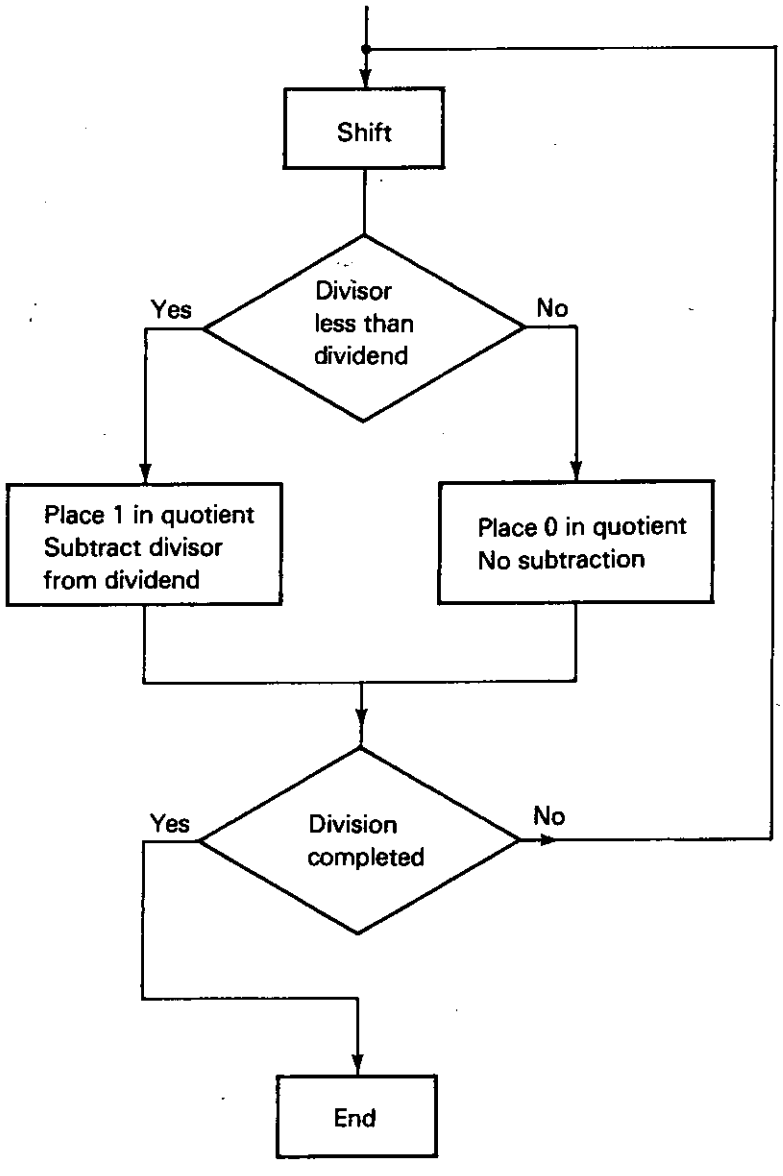


Fig. 6.32 Flow diagram for the division procedure

To avoid using expensive comparators, computers usually carry out the test by subtraction. In this case the divisor is subtracted from the part of the dividend under consideration. A positive answer indicates that the divisor is smaller and a 1 is placed in the quotient. A negative answer, however, indicates that the divisor is larger and subtraction was not necessary; then the divisor is added back to the dividend. The addition process restores the original value of the dividend, which gives rise to the name *restoring division*.

This can be represented by the flow chart of Fig. 6.33.

To implement the restoring procedure the divisor is placed in the divisor register. The dividend or partial dividend is placed in the accumulator register, as shown in Fig. 6.34. At each clock pulse subtraction is performed using the 2's complement method. The answer determines whether restoring is necessary and whether a 1 or 0 is to be placed in the quotient register.

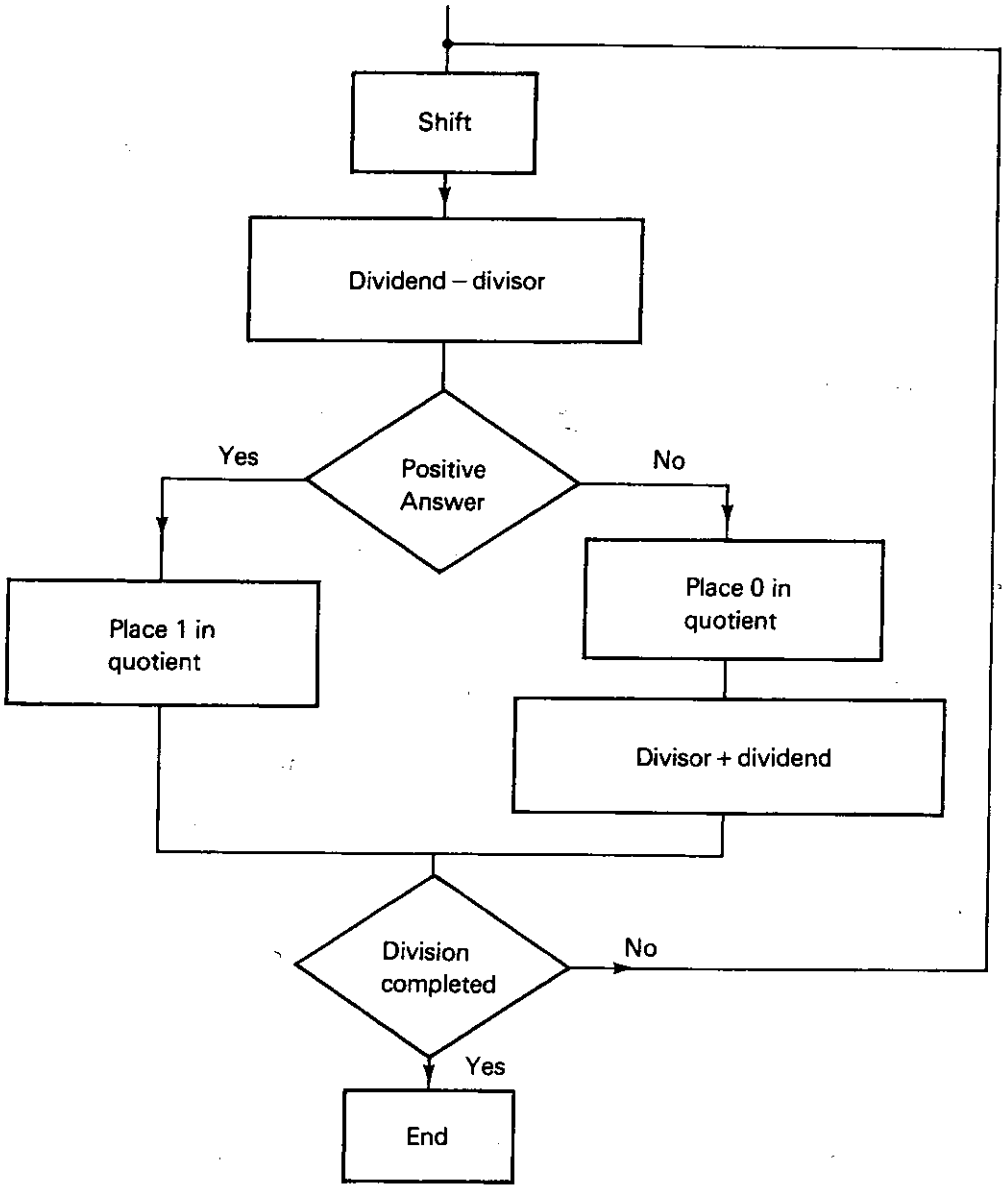


Fig. 6.33 Flow diagram representation of restoring division

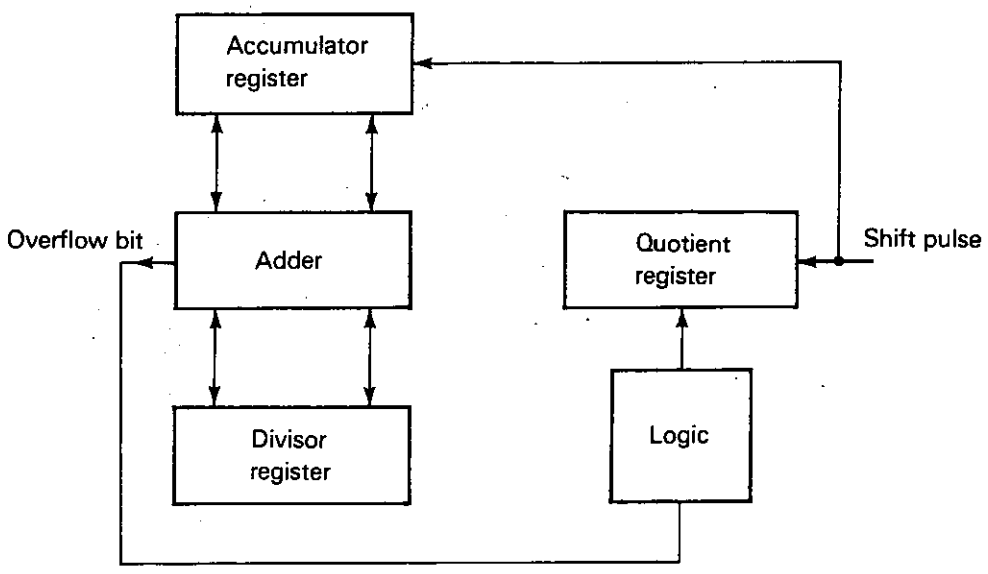


Fig. 6.34 Block diagram for the restoring division procedure

Division, like multiplication, is carried out on fractional operands. These are then scaled by the program. The decimal number 981, for example, is taken as 0.981 with a scale factor of 1000.

When compilers are written for fixed-point arithmetic machines, scaling is provided and the user is not aware of it.

6.8.3 Non-Restoring Division

(*a*) Divisor is Greater than Dividend

In restoring division, each time the divisor is added to restore the dividend it is subtracted during the next cycle after the shift operation.

A right-hand shift of the divisor reduces its value by half. The right-hand branch of the flow chart in Fig. 6.33 can be interpreted as follows:

Add present divisor.
Subtract half present divisor (subtract shifted divisor).

These two steps can be replaced by 'add shifted divisor'.

(*b*) Divisor is Less than Dividend

In this case the next instruction is 'subtract shifted divisor'. The two parts can be combined together in the modified flow chart of Fig. 6.35.

This method avoids the restoration of the dividend which makes it faster compared with the previous method. As usual, the penalty is more logic, and more complex control circuits are required.

It should be observed that not all computers incorporate multiply and divide logic. In some cases, the processes of add, complement and shift are used to achieve

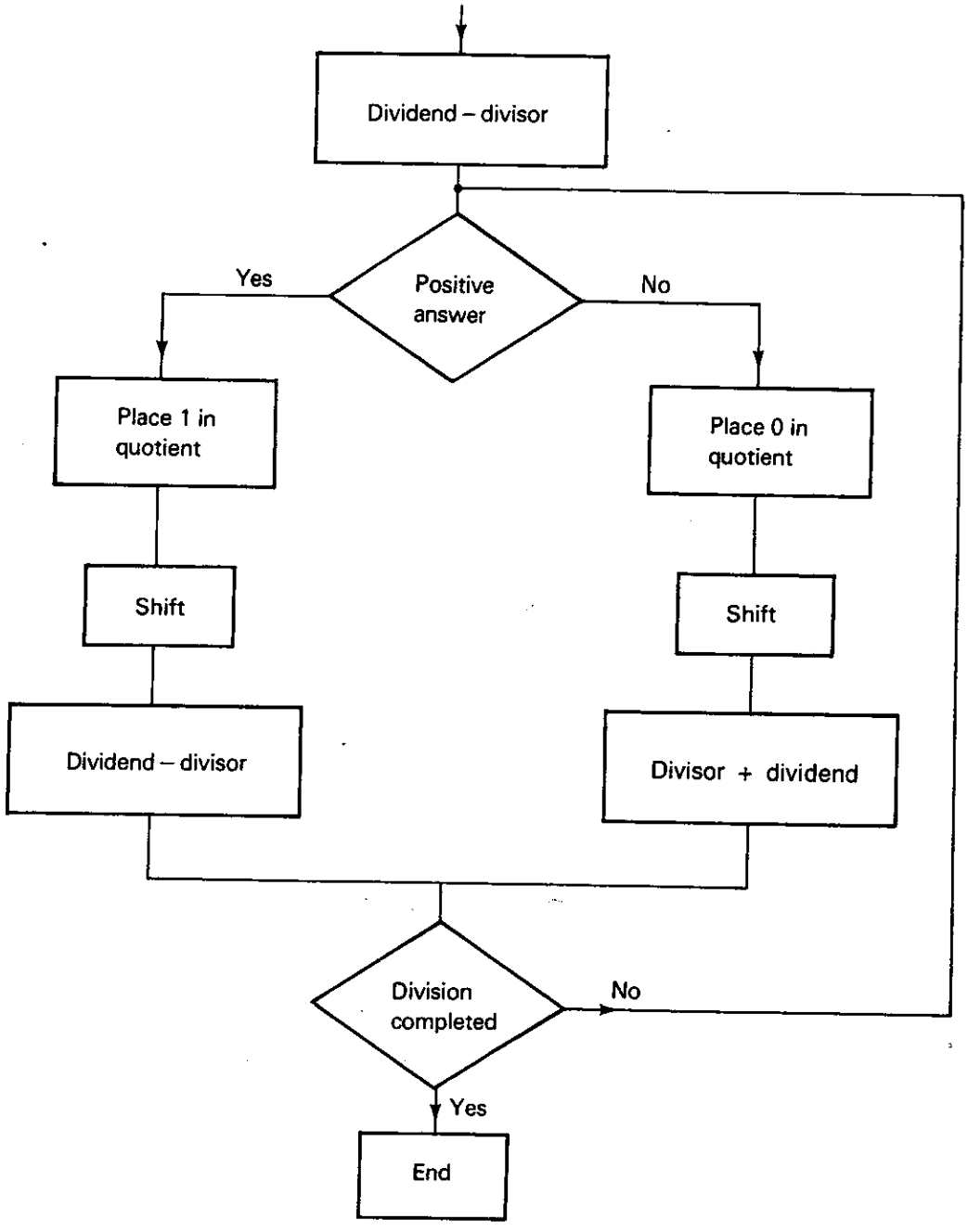
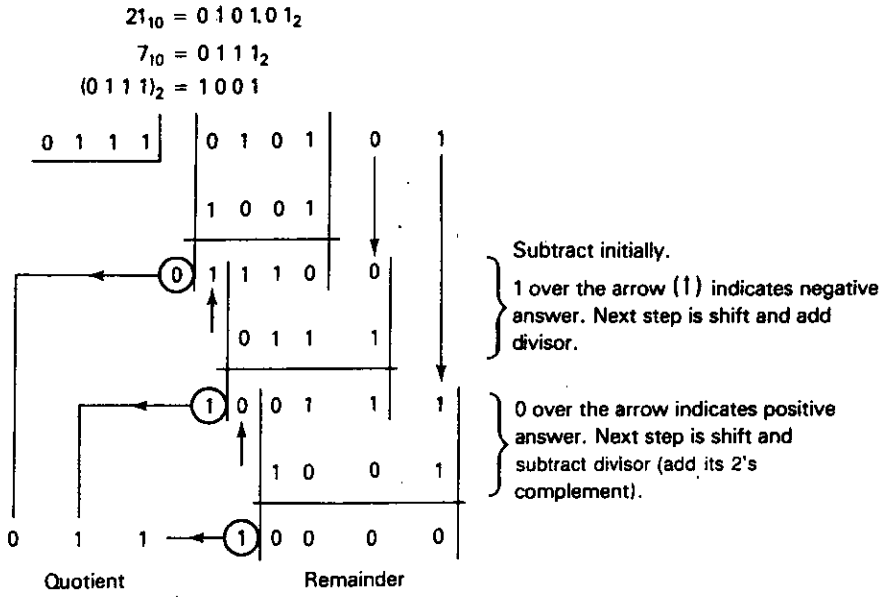


Fig. 6.35 Flow diagram representation of non-restoring division

multiplication and division using software techniques. With the advent of LSI, however, more computers are incorporating specialized multiplication and division circuits, especially when speed is important.

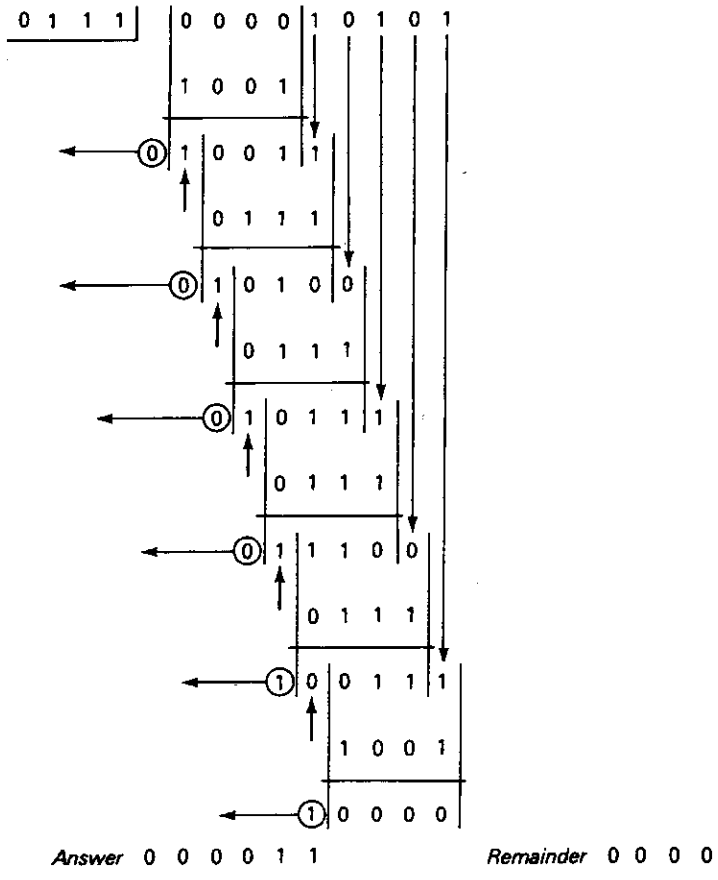
To illustrate the non-restoring-division procedure we shall assume a six-bit dividend and a four-bit divisor given in 2's complement representation. For simplicity, the two operands are assumed positive (msd = 0). Subtraction is achieved by adding the 2's complements. Let us first start with a numerical example and divide 21 by 7, using the procedure outlined in Fig. 6.35:



The quotient is obtained from the encircled overflow digits giving a quotient of 011 and a remainder of 0000. In the decimal system the answer is 3 without remainder.

The answer in this case is obtained in three levels of addition/subtraction. If however, we divide 21 by 1, six levels of addition/subtraction are required to produce a six-digit quotient. Generally speaking, the number of levels is equal to the number of digits in the dividend.

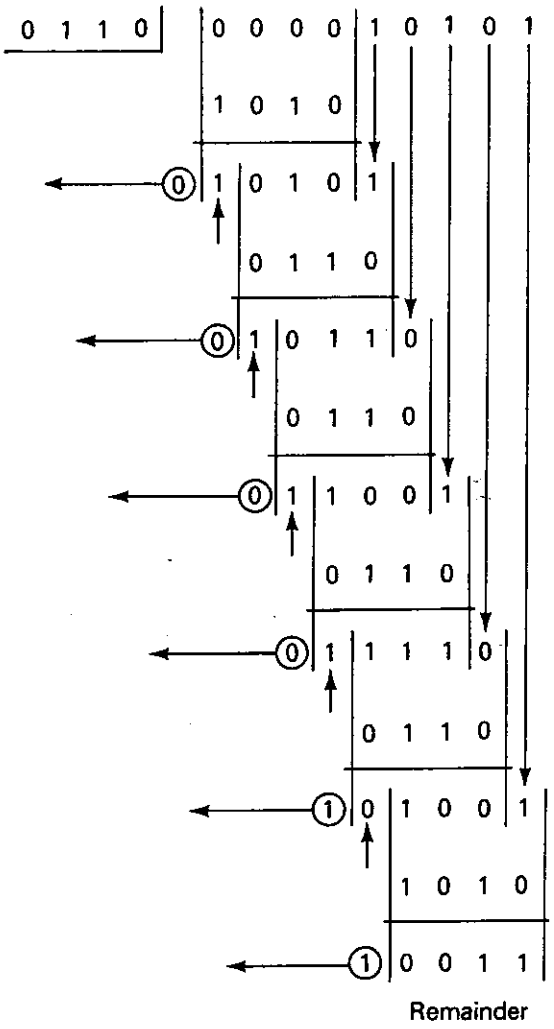
Let us repeat the same problem using six levels of addition:



The circuit to achieve this is given in Fig. 6.36. In the first level at the top, the control line feeding the EX-OR gates is 1. This inverts the divisor digits and adds 1 to the least significant digit (connected to the carry-in C_0, of the four-bit adder). The effect is a 2's complement subtraction.

After the first level, the fourth digit of the adder (S_4) decides whether the next level will add the shifted divisor ($S_4 = 1$) or subtract it ($S_4 = 0$). The carry-out (C_4) digits from the six levels are taken as the answer (quotient). To understand the function of the last four-bit adder, we consider the following two examples:

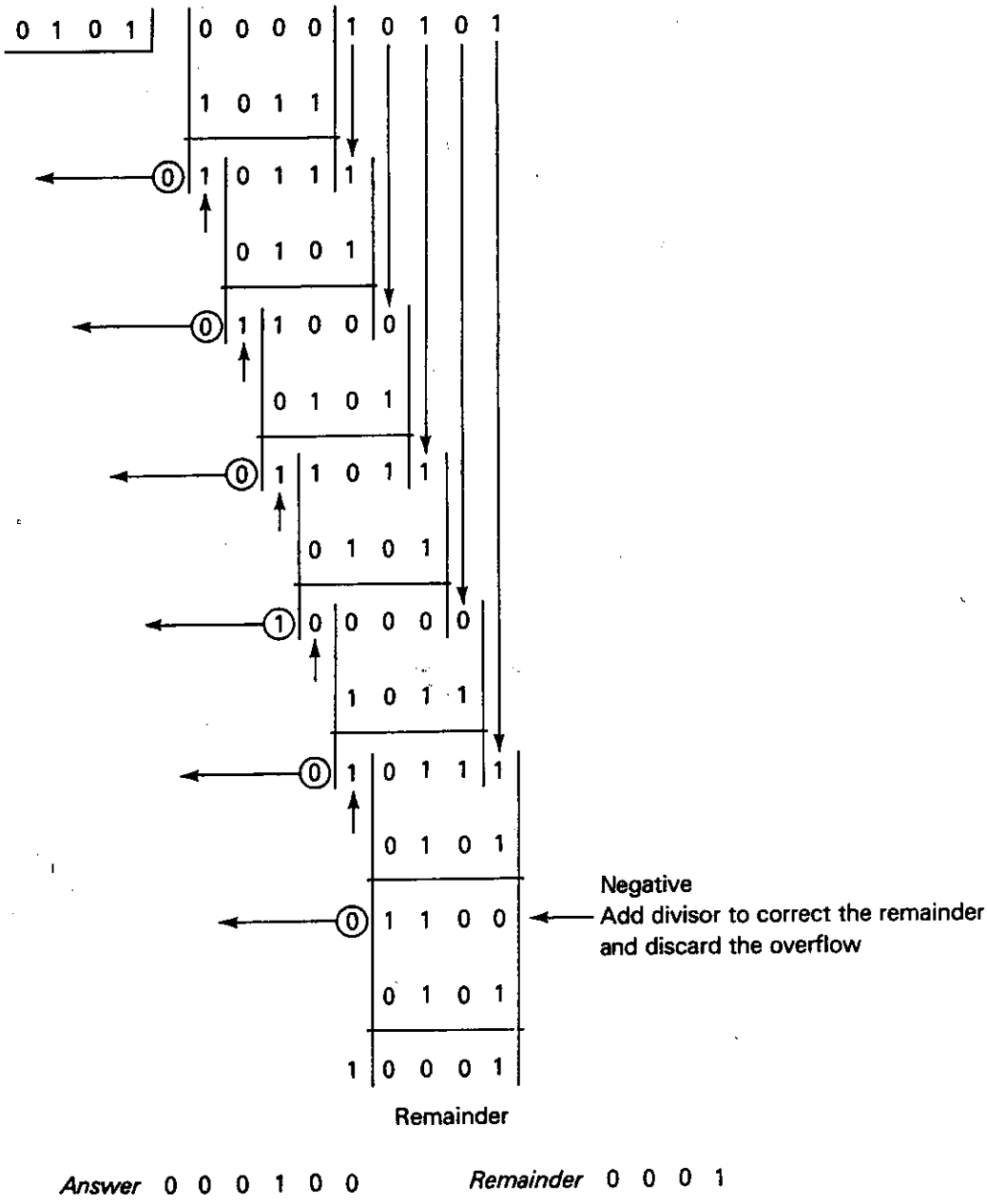
(i) 21 ÷ 6 = 3, Remainder 3



Answer 0 0 0 0 1 1 *Remainder* 0 0 1 1

The msd of the remainder (sign digits) is zero. This disables the four AND gates. The last four-bit adder will have no effect in this case.

(ii) 21 ÷ 5 = 4, Remainder 1



Note that S_4 of the last stage, which is the sign digit of the remainder, is 1. This indicates a negative remainder, which must be corrected. The logic 1 is used to enable the four AND gates and allows the divisor to be added to the remainder and produce the correct answer.

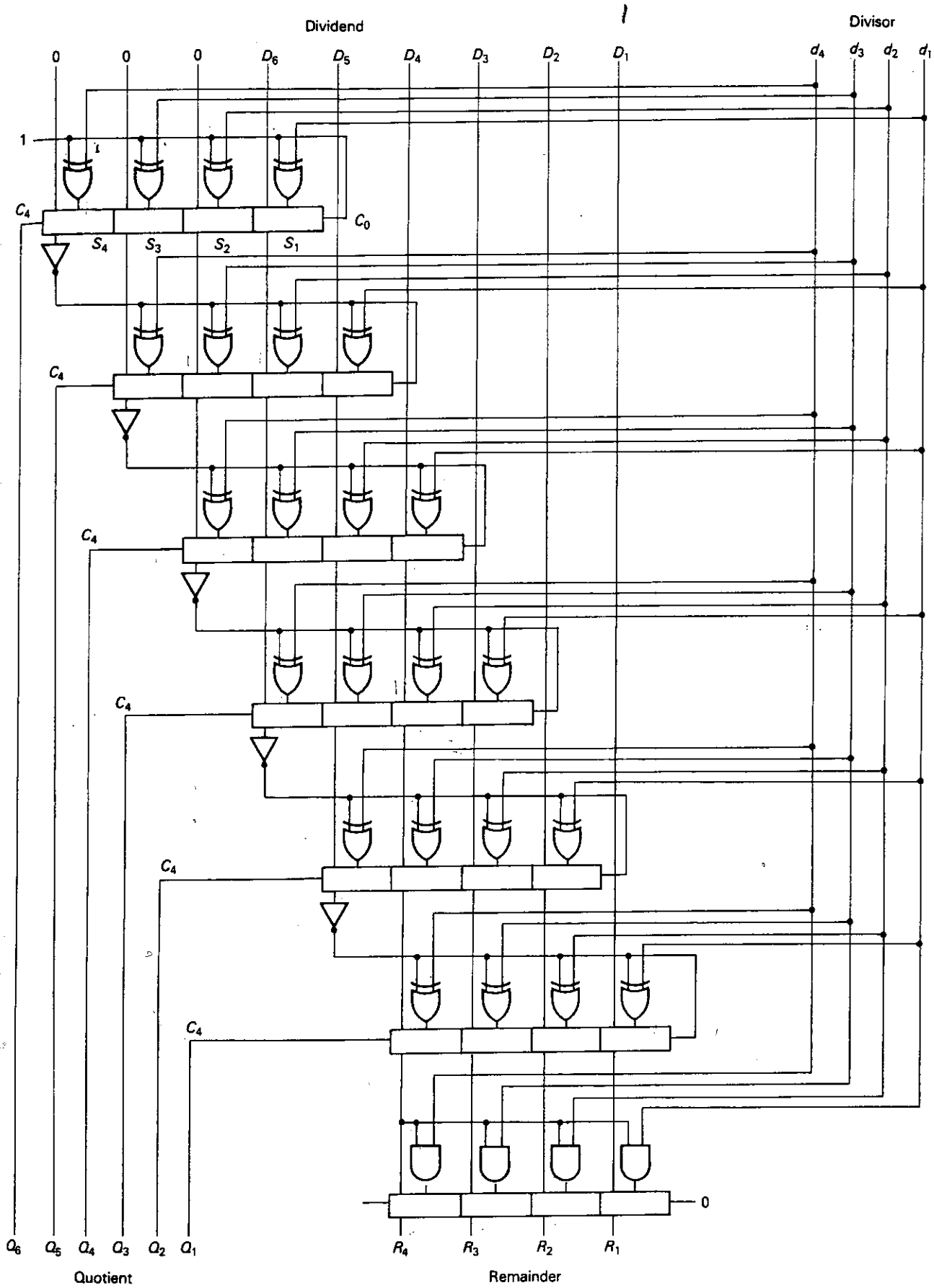


Fig. 6.36 Non-restoring division