the divisor, the digit was chosen correctly. Otherwise, another digit is chosen and the subtraction is repeated. In each step of the operation a digit of the quotient is obtained.

Binary numbers contain only 0 and 1, so binary division is restricted to these two choices. The division operation consists of a series of subtractions of the divisor from the partial remainder, which are only executed if the divisor is smaller than the partial remainder, when the digit of the quotient is 1; otherwise, the corresponding digit of the quotient is 0.

Consider the division of two numbers, 74 ($100\ 1010_2$) by 8 (1000_2).

```
 1001010 : 1000 = 1001      Quotient
-1000
   10                       Partial remainders
   101
   1010
  -1000
    10                      Remainder
```

3.3.1. Restoring Division

We assume that both the dividend and divisor are positive and hence the quotient and the remainder are positive or zero.

Figure 3.20 shows the circuit which implements the division algorithm. In each step of the algorithm, the divisor is shifted one position to the right, and the quotient is shifted one position to the left.

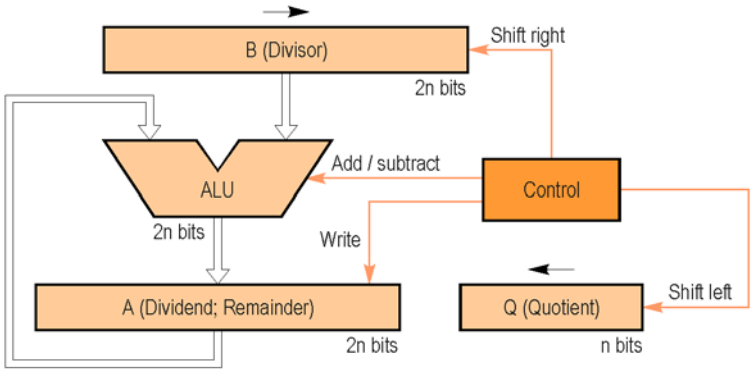

Figure 3.20. First version of the divider circuit.

Figure 3.21 shows the steps of the first version of the division algorithm.

Initially, the dividend is loaded into the right half of the $2n$-bit A register, and the divisor is loaded into the left half of the $2n$-bit B register. The n-bit quotient register (Q) is set to 0, and the counter N is set to $n+1$. In order to determine whether the divisor is smaller than the partial remainder, the divisor register (B) is subtracted from the remainder register (A). If the result is negative, the next step is to restore the pre-

vious value by adding the divisor back to the remainder, generating a 0 in the Q_0 position of the quotient register. This is the reason why this method is called *restoring division*. If the result is positive, a 1 is generated in the Q_0 position of the quotient register. In the next step, the divisor is shifted to the right, aligning the divisor with the dividend for the next iteration.

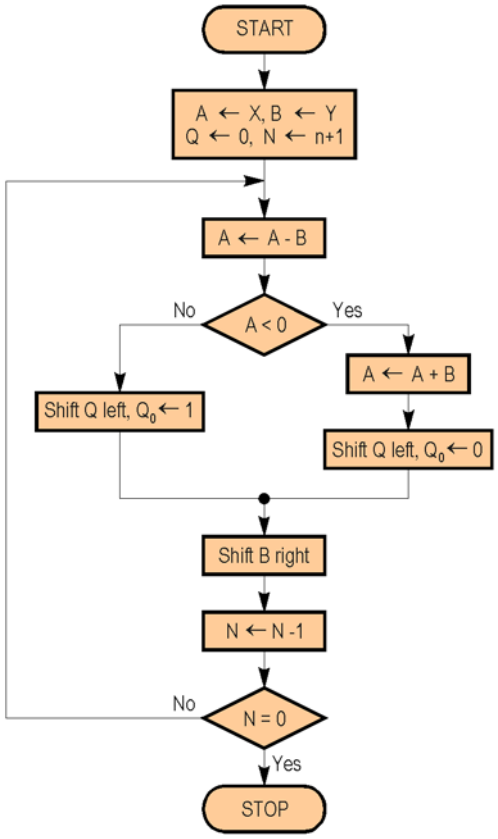

Figure 3.21. The first version of the restoring division algorithm.

Example 3.4

Using the first version of the restoring division algorithm, divide the 4-bit numbers $X = 13$ and $Y = 5$ ($1101_2 \div 0101_2$).

Answer

Table 3.6 shows the contents of the registers in each step of the operation, finally obtaining a quotient of 2 and a remainder of 3.

Table 3.6. Division example using the first version of the algorithm.

Step	A	Q	B	Operation
0	0000 1101	0000	0101 0000	Initialization
1	1011 1101	0000	0101 0000	A = A - B
	0000 1101	0000	0101 0000	A = A + B, Shift left Q, Q_0 = 0
	0000 1101	0000	0010 1000	Shift right B
2	1101 1000			A = A - B
	1110 0101	0000	0010 1000	
	0000 1101	0000	0010 1000	A = A + B, Shift left Q, Q_0 = 0
	0000 1101	0000	0001 0100	Shift right B
3	1110 1100			A = A - B
	1111 1001	0000	0001 0100	
	0000 1101	0000	0001 0100	A = A + B, Shift left Q, Q_0 = 0
	0000 1101	0000	0000 1010	Shift right B
4	1111 0110			A = A - B
	0000 0011	0000	0000 1010	
	0000 0011	0001	0000 1010	Shift left Q, Q_0 = 1
	0000 0011	0001	0000 0101	Shift right B
5	1111 1011			A = A - B
	1111 1110	0001	0000 0101	
	0000 0011	0010	0000 0101	A = A + B, Shift left Q, Q_0 = 0
	0000 0011	0010	0000 0010	Shift right B

Shifting the partial remainder to the left instead of shifting the divisor to the right produces the same alignment and simplifies the hardware necessary for the ALU and the divisor register. Both the divisor register and the ALU could have the size reduced to half (n bits instead of $2n$ bits).

The second improvement comes from the fact that the first step of the algorithm cannot generate a digit of 1 in the quotient, because, in this case, the quotient would be too large for its register. By switching the order of the operations to shift and then to subtract, one iteration of the algorithm can be removed.

Another observation is that the size of the A register could be reduced to half, and the A and Q registers could be combined, shifting the bits of the quotient into the A register instead of shifting in zeros as in the preceding algorithm. The A and Q registers are shifted left together. Figure 3.22 shows the final version of the restoring division algorithm.

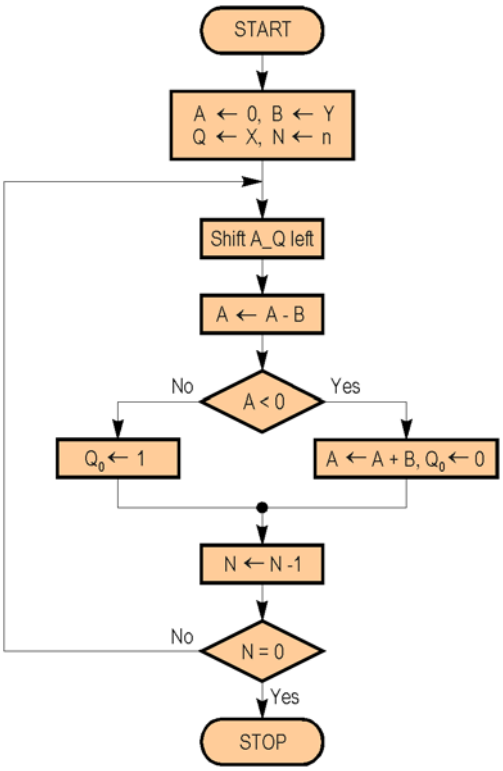

Figure 3.22. Final version of the restoring division algorithm.

The final version of the restoring divider circuit is shown in Figure 3.23.

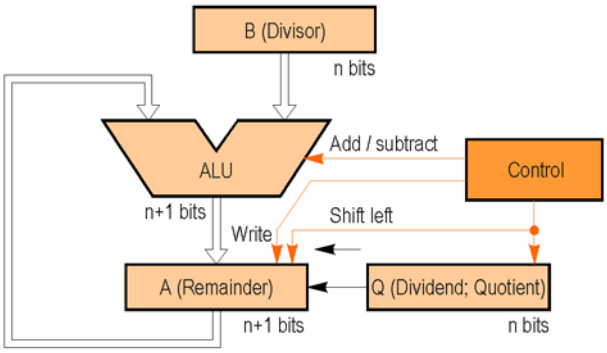

Figure 3.23. Final version of the restoring divider circuit.

Example 3.5

Using the final version of the algorithm, divide $X = 13$ by $Y = 5$.

Answer

Table 3.7 shows the contents of the registers in each step of the operation. The remainder is formed in the *A* register, and the quotient in the *Q* register.

Table 3.7. Division example using the final version of the restoring division algorithm.

Step	A	Q	B	Operation
0	0 0000	1101	0101	Initialization
1	0 0001 1 1011 1 1100 0 0101 0 0001	1010 1010	 0101	Shift left A_Q A = A - B A = A + B, $Q_0 = 0$
2	0 0011 1 1011 1 1110 0 0101 0 0011	0100 0100	 0101	Shift left A_Q A = A - B A = A + B, $Q_0 = 0$
3	0 0110 1 1011 0 0001	1000 1001	 0101	Shift left A_Q A = A - B $Q_0 = 1$
4	0 0011 1 1011 1 1110 0 0101 0 0011	0010 0010	 0101	Shift left A_Q A = A - B A = A + B, $Q_0 = 0$

3.3.2. Nonrestoring Division

Restoring the partial remainder increases the execution time of the division operation, since on average the restoring is executed in 50% of the cases. Each addition of the divisor to the partial remainder is followed by a subtraction of the divisor in the next step, after the partial remainder is shifted one position to the left. Shifting to the left is equivalent to multiplying by 2.

The sequence of operations performed is as follows:

- The divisor (*Y*) is subtracted from the partial remainder (*R*):

$$R \leftarrow R - Y$$

- If the result is negative, the partial remainder is restored in the same step:

$$R \leftarrow R - Y + Y$$

- In the next step, the partial remainder is shifted one position left:

$$R \leftarrow 2 \times R$$

- The divisor is subtracted from the partial remainder:

$$R \leftarrow 2 \times R - Y$$

The same result can be obtained by another sequence of operations:

- The divisor is subtracted from the partial remainder:

$$R \leftarrow R - Y$$

- If the result is negative, the partial remainder is not restored, but it is shifted one position left in the next step:

$$R \leftarrow 2 \times R - 2 \times Y$$

- The divisor is added to the partial remainder:

$$R \leftarrow 2 \times R - 2 \times Y + Y$$

In conclusion, in each step of the nonrestoring division, after shifting the registers *A_Q* one position left, the divisor is subtracted from or added to the partial remainder, depending on the sign of the partial remainder in the preceding step. If the partial remainder is positive, in the next step a subtraction is performed, otherwise the operation performed is an addition. After the last step, if the partial remainder is positive, the remainder obtained is correct; otherwise, the remainder must be corrected by restoring it (adding the divisor to the remainder).

3.3.3. SRT Division

The name of the SRT division stands for Dura W. Sweeney, James E. Robertson, and Keith D. Tocher, who independently proposed a fast algorithm for 2's complement numbers that use the technique of shifting over zeros for division.

To divide two *n*-bit numbers *X* and *Y*, the operands are loaded into the *Q* and *B* registers, respectively, and the *A* register is set to 0. The SRT division algorithm is as follows:

1. If register *B* has *k* leading zeros when expressed using *n* bits, shift all the registers *k* positions left.
2. The following steps are repeated *n* times:

   If the top three bits of the *A* register are equal, shift the *A_Q* registers one position left, and set $Q_0 = 0$.

   If the top three bits of the *A* register are not equal and *A* is negative, shift the *A_Q* registers one position left, set $Q_0 = -1$ ($\bar{1}$), and add *B* to *A*.