UNIT 1

Functional Blocks of a Computer: Introduction, Block diagram of digital computer, Instruction codes, Computer Registers, Common bus system, Computer instructions, Instruction cycle and Instruction set, Register Transfer Language.

Data Representation: Fixed and floating point arithmetic- Addition, Subtraction, Multiplication, Division.

Control unit Design: Hardwired control unit, Control memory, Address sequencing, Micro-programmed control unit design, Hardwired Vs Micro-programmed design.

Addition and Subtraction with Signed-Magnitude Data

Addition and Subtraction
- Sign-magnitude
- 2's complement

TABLE 10-1 Addition and Subtraction of Signed-Magnitude Numbers

Operation	Add Magnitudes	Subtract Magnitudes		
		When $A > B$	When $A < B$	When $A = B$
$(+A) + (+B)$	$+(A + B)$			
$(+A) + (-B)$		$+(A - B)$	$-(B - A)$	$+(A - B)$
$(-A) + (+B)$		$-(A - B)$	$+(B - A)$	$+(A - B)$
$(-A) + (-B)$	$-(A + B)$			
$(+A) - (+B)$		$+(A - B)$	$-(B - A)$	$+(A - B)$
$(+A) - (-B)$	$+(A + B)$			
$(-A) - (+B)$	$-(A + B)$			
$(-A) - (-B)$		$-(A - B)$	$+(B - A)$	$+(A - B)$

Hardware implementation:

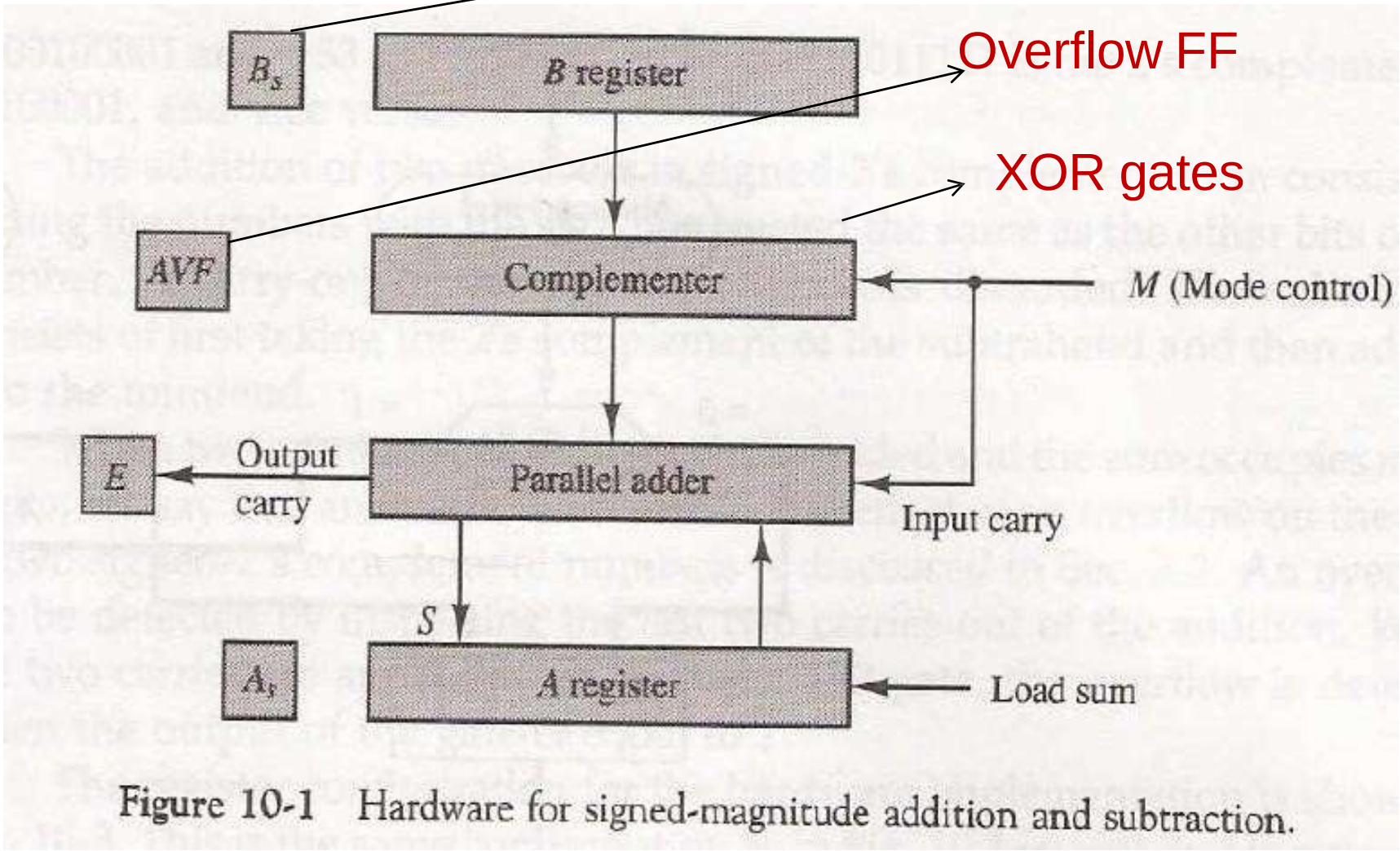


Figure 10-1 Hardware for signed-magnitude addition and subtraction.

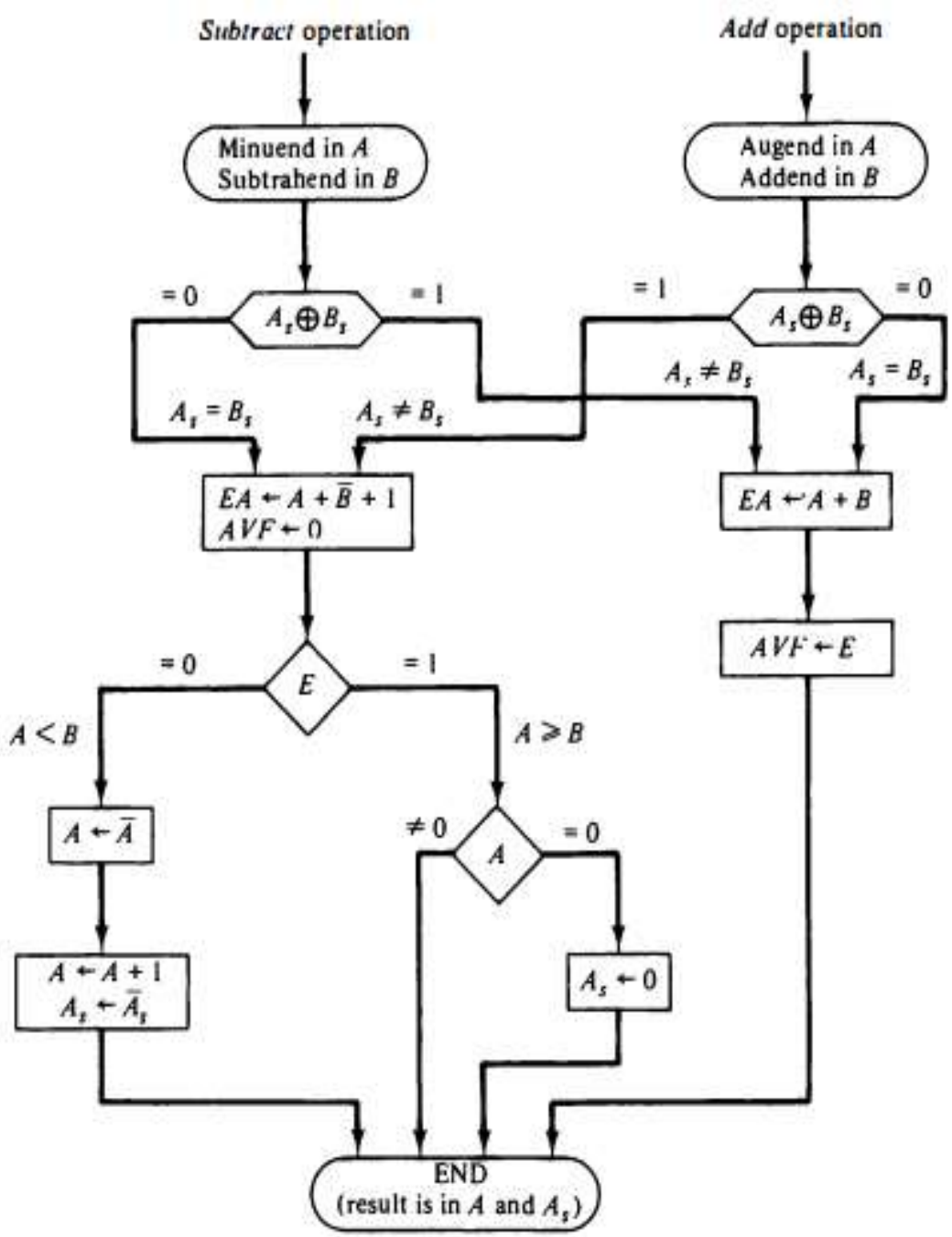
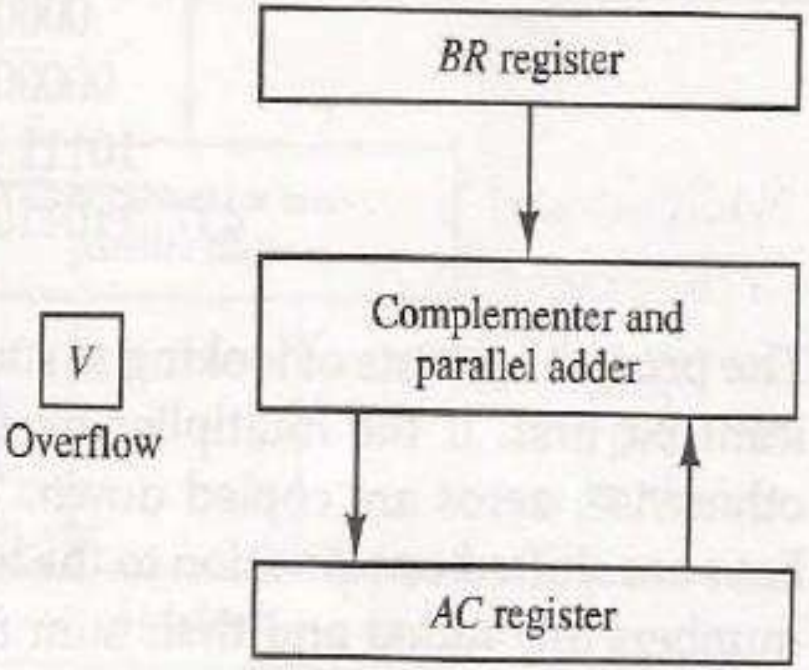


Figure 10-2 Flowchart for add and subtract operations.

Addition and subtraction with signed-2's complement data

Figure 10-3 Hardware for signed-2's complement addition and subtraction.



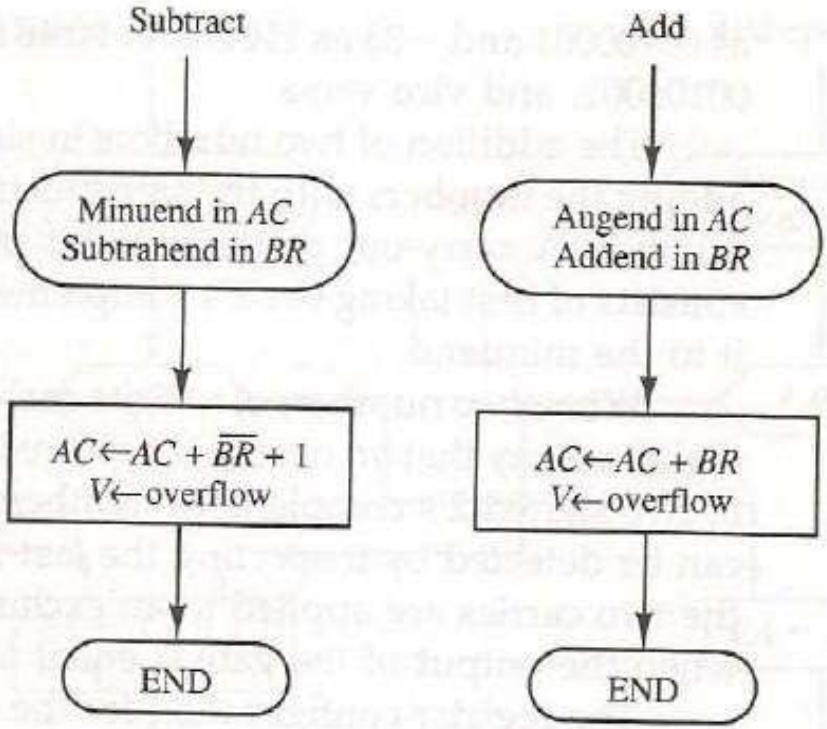
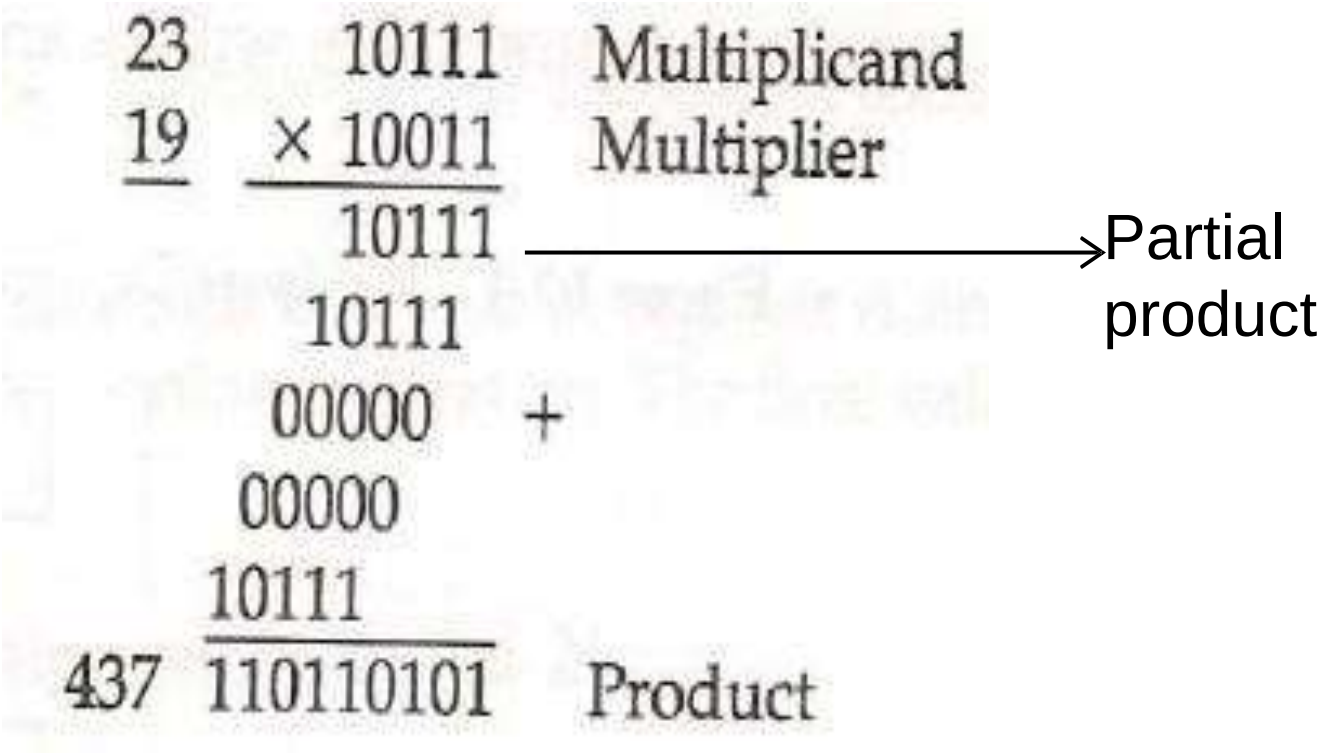


Figure 10-4 Algorithm for adding and subtracting numbers in signed-2's complement representation.

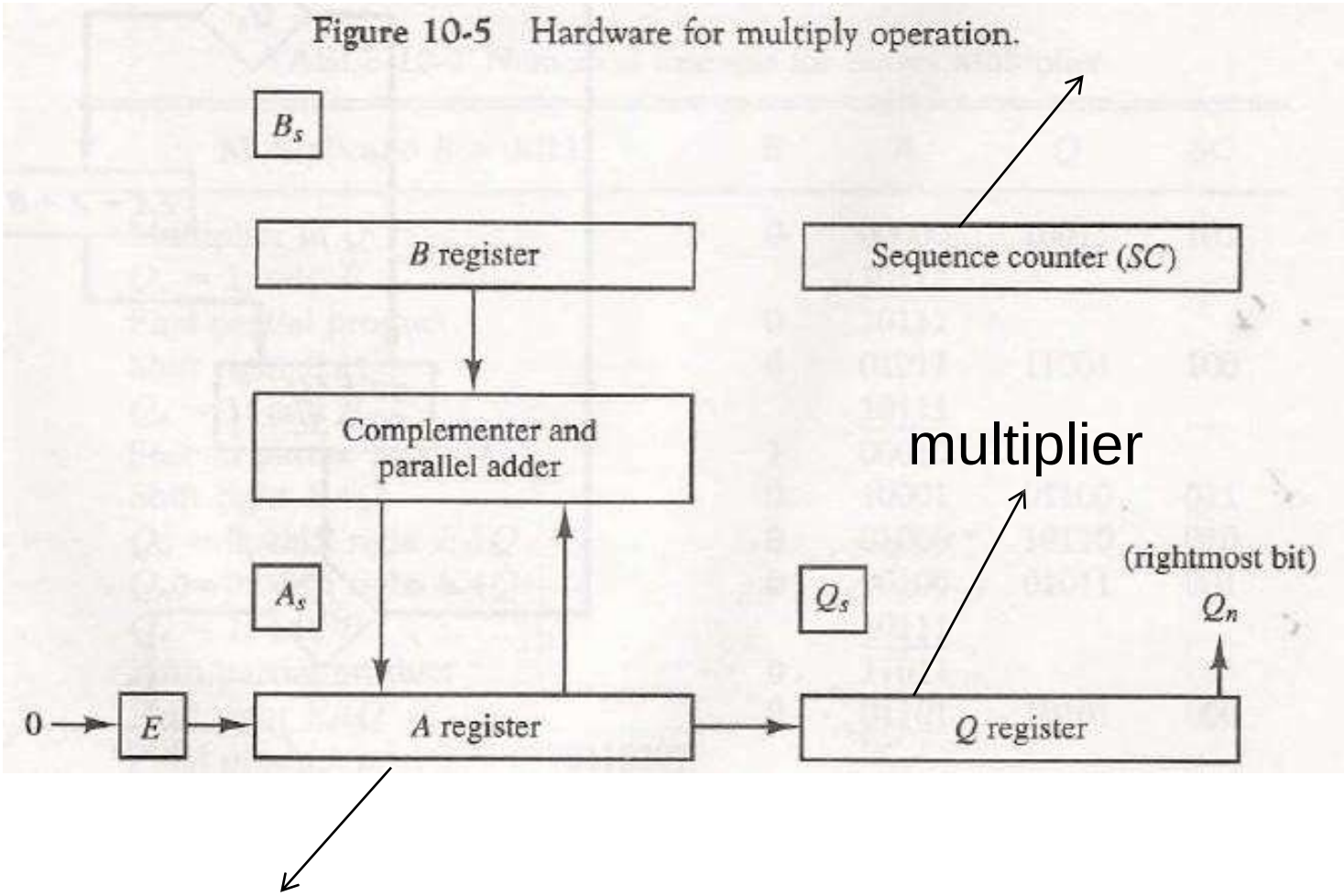
Multiplication algorithms:

A binary example:



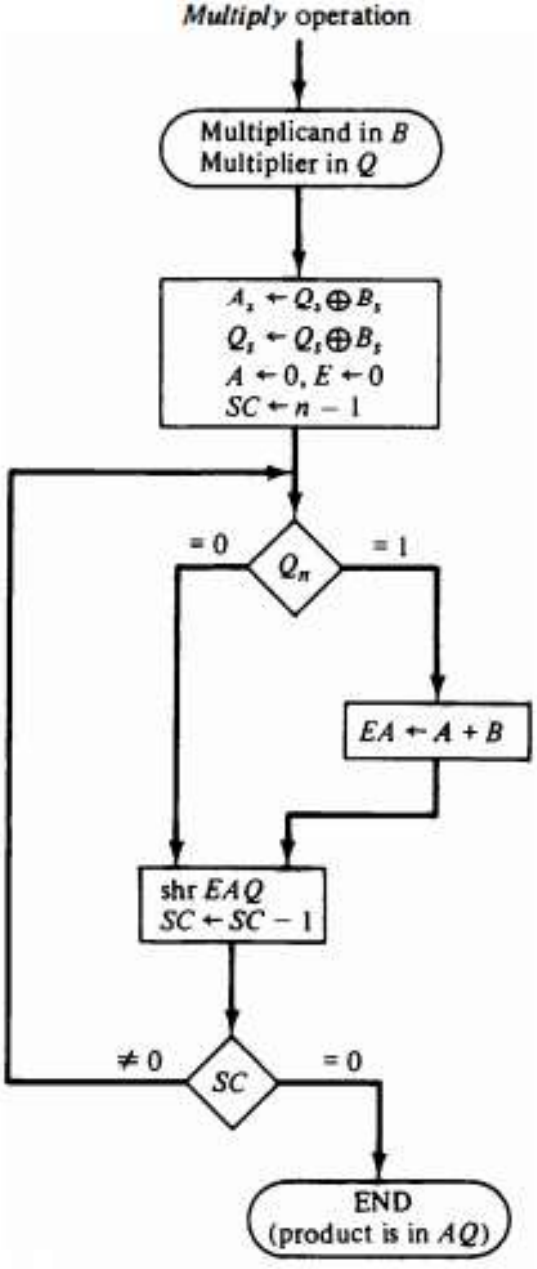
Hardware implementation for Signed-Magnitude Data

#of bit in multiplier



Partial product

Figure 10-6 Flowchart for multiply operation.



Multiplicand B = 10111(23) and Multiplier Q = 10011(19)

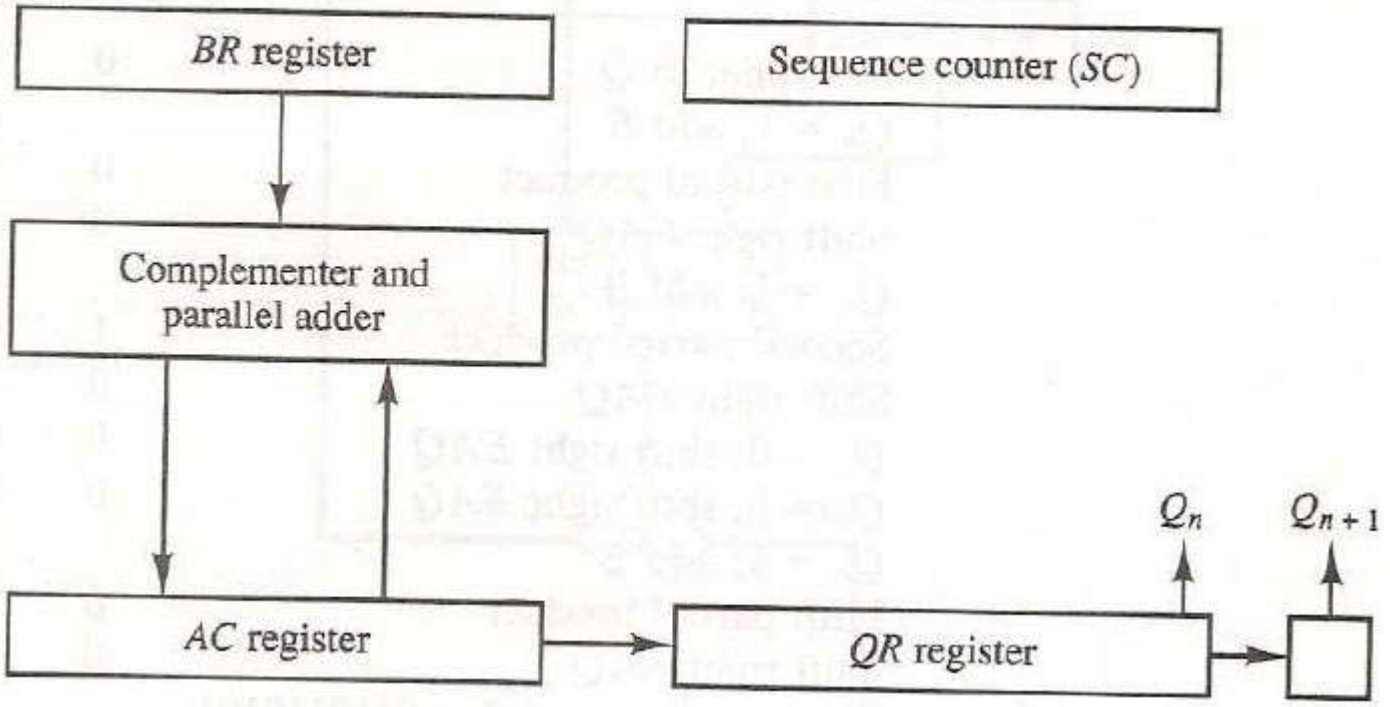
TABLE 10-2 Numerical Example for Binary Multiplier

Multiplicand $B = 10111$	E	A	Q	SC
Multiplier in Q	0	00000	10011	101
$Q_n = 1$; add B		10111		
First partial product	0	10111		
Shift right EAQ	0	01011	11001	100
$Q_n = 1$; add B		10111		
Second partial product	1	00010		
Shift right EAQ	0	10001	01100	011
$Q_n = 0$; shift right EAQ	0	01000	10110	010
$Q_n = 0$; shift right EAQ	0	00100	01011	001
$Q_n = 1$; add B		10111		
Fifth partial product	0	11011		
Shift right EAQ	0	01101	10101	000
Final product in $AQ = 0110110101$				

Booth multiplication algorithm

Booth algorithm gives a procedure for multiplying binary integers in signed-2's complement representation.

Figure 10-7 Hardware for Booth algorithm.



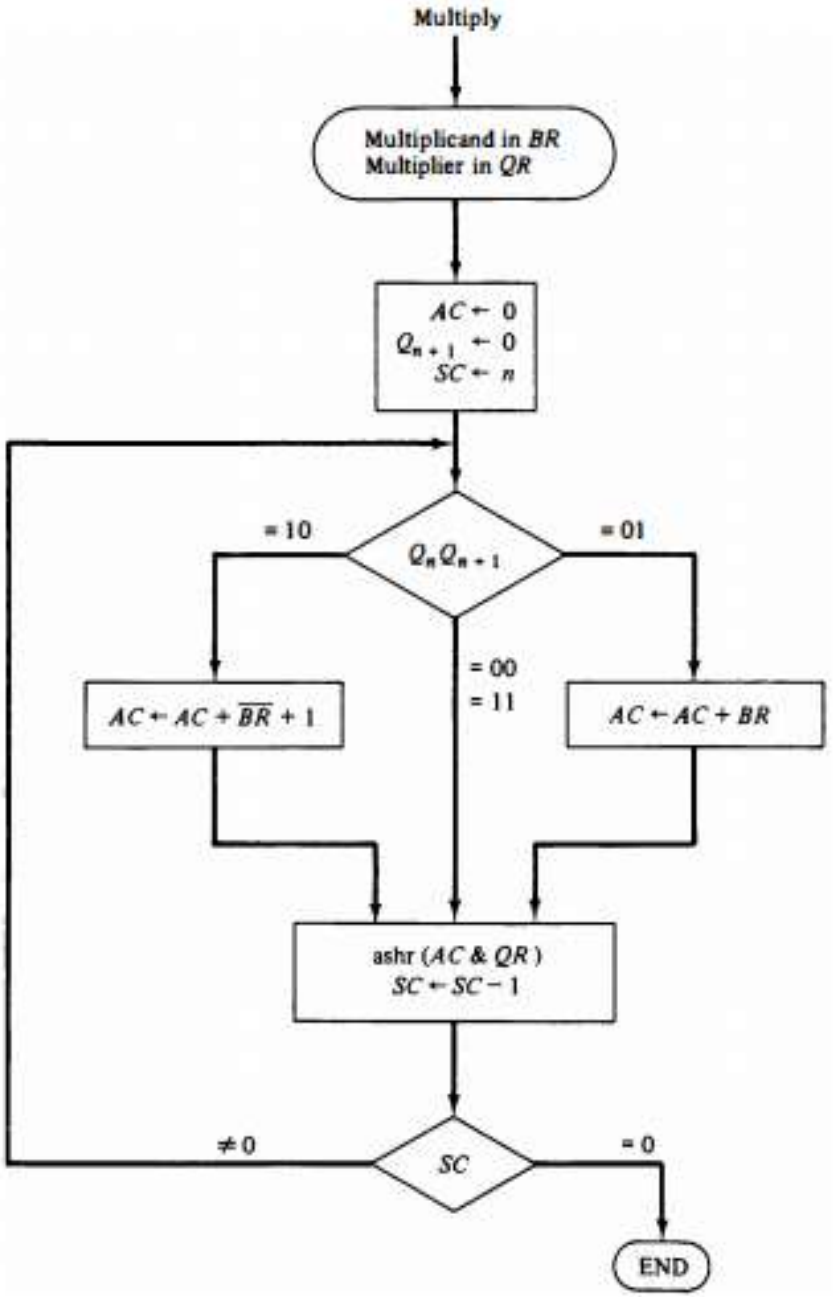


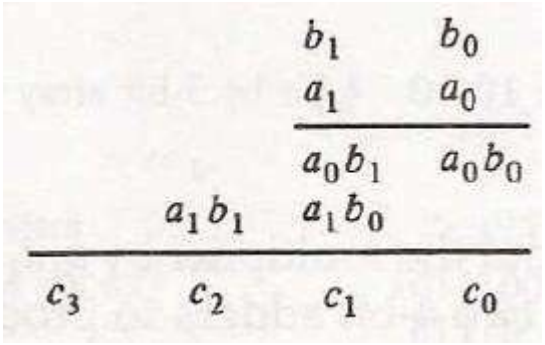
Figure 10-8 Booth algorithm for multiplication of signed-2's complement numbers.

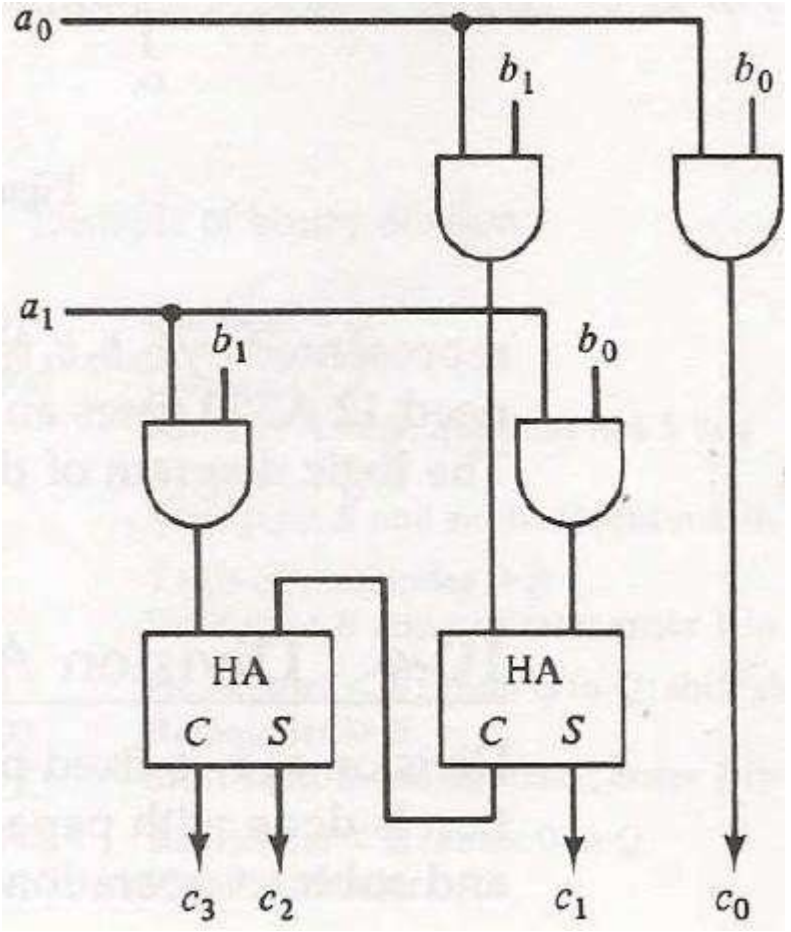
Multiplicand BR = 10111(-9) and Multiplier QR = 10011(-13)

TABLE 10-3 Example of Multiplication with Booth Algorithm

$Q_n Q_{n+1}$	$BR = 10111$ $\overline{BR} + 1 = 01001$	AC	QR	Q_{n+1}	SC
	Initial	00000	10011	0	101
1 0	Subtract BR	01001			
		01001			
	ashr	00100	11001	1	100
1 1	ashr	00010	01100	1	011
0 1	Add BR	10111			
		11001			
	ashr	11100	10110	0	010
0 0	ashr	11110	01011	0	001
1 0	Subtract BR	01001			
		00111			
	ashr	00011	10101	1	000

Array multiplier: Fast approach





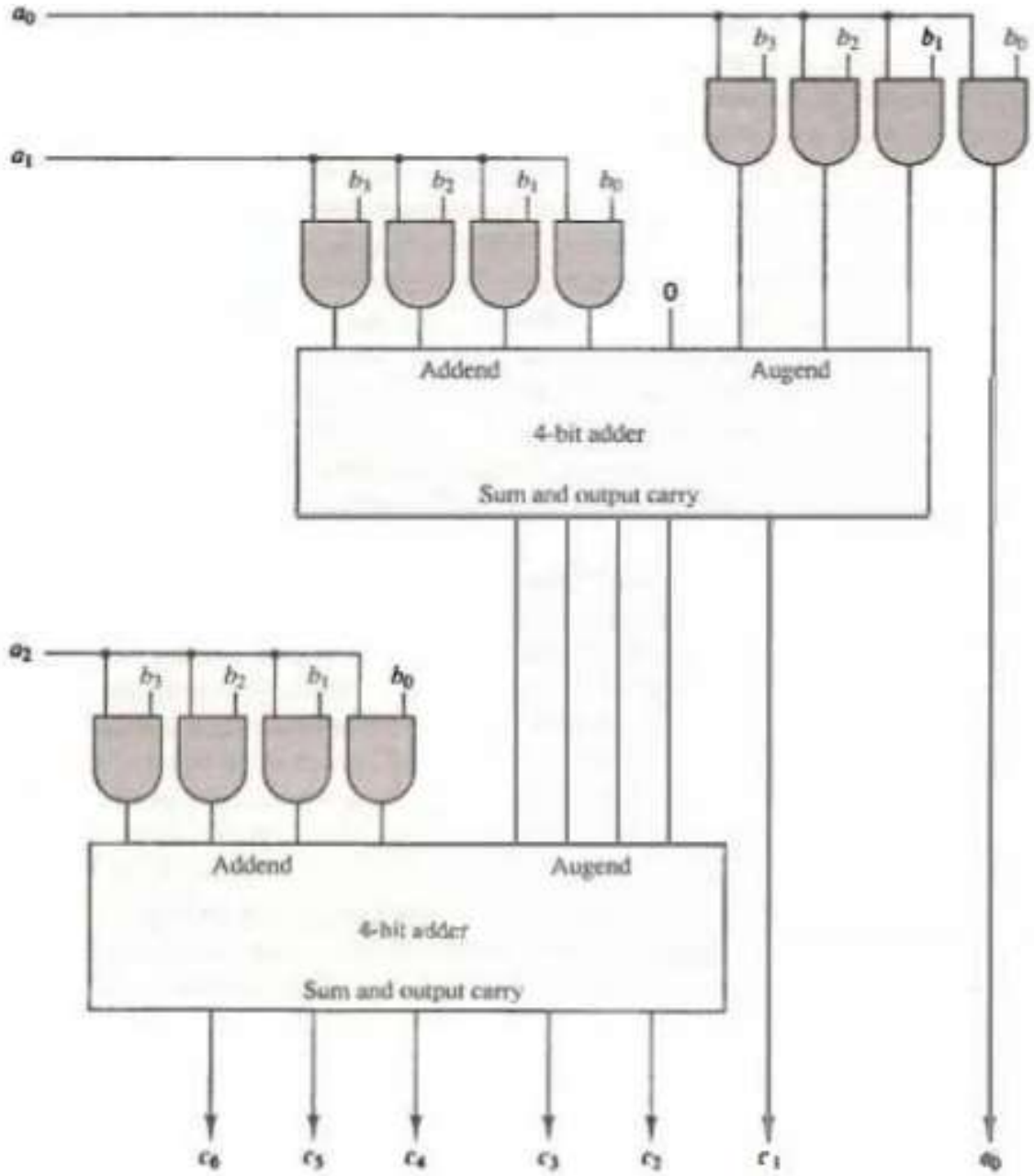
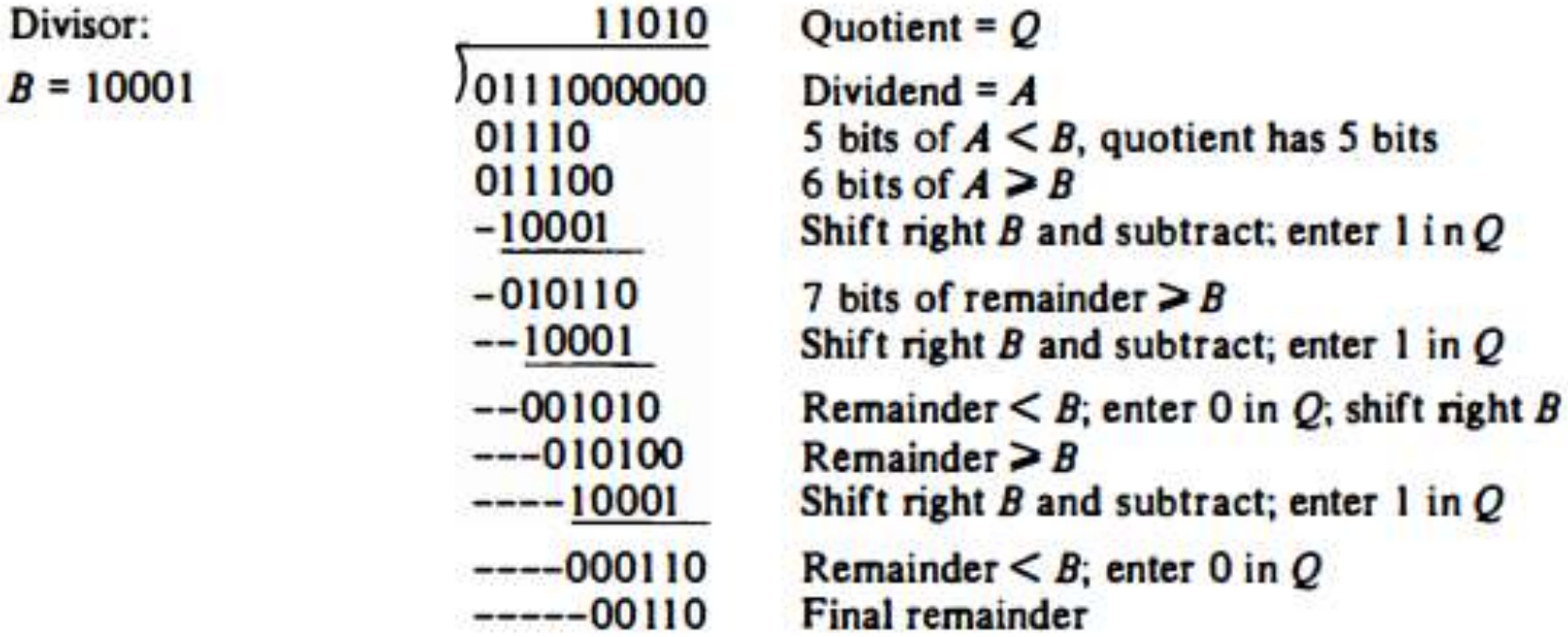


Figure 10.10 4-bit by 3-bit array multiplier.

Division algorithms:

Dividend=449, Divisor=17, Quotient=26 and Remainder=6

Figure 10-11 Example of binary division.



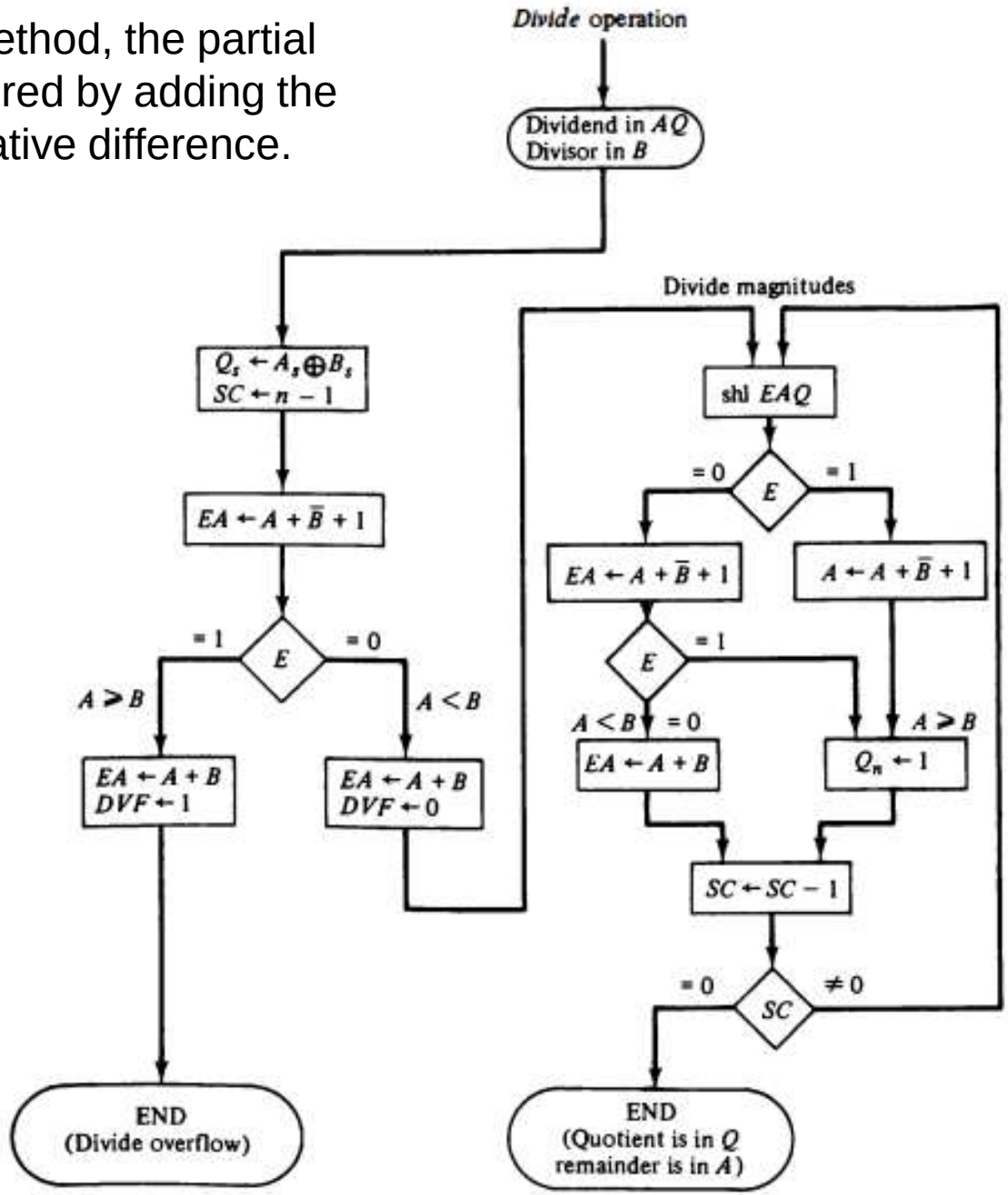
Divisor $B = 10001$, $\quad \overline{B} + 1 = 01111$

	E	A	Q	SC
Dividend:		01110	00000	5
shl EAQ	0	11100	00000	
add $\overline{B} + 1$		01111		
$E = 1$	1	01011		
Set $Q_n = 1$	1	01011	00001	4
shl EAQ	0	10110	00010	
Add $\overline{B} + 1$		01111		
$E = 1$	1	00101		
Set $Q_n = 1$	1	00101	00011	3
shl EAQ	0	01010	00110	
Add $\overline{B} + 1$		01111		
$E = 0$; leave $Q_n = 0$	0	11001	00110	
Add B		10001		2
Restore remainder	1	01010		
shl EAQ	0	10100	01100	
Add $\overline{B} + 1$		01111		
$E = 1$	1	00011		
Set $Q_n = 1$	1	00011	01101	1
shl EAQ	0	00110	11010	
Add $\overline{B} + 1$		01111		
$E = 0$; leave $Q_n = 0$	0	10101	11010	
Add B		10001		
Restore remainder	1	00110	11010	0
Neglect E				
Remainder in A:		00110		
Quotient in Q:			11010	

Figure 10-12 Example of binary division with digital hardware.

Figure 10-13 Flowchart for divide operation.

In the restoring method, the partial remainder is restored by adding the divisor to the negative difference.



Other Algorithms:

- Comparison: In the comparison method A and B are compared prior to the subtraction operation.
- Then if A >=B, B is subtracted from A. If A < B nothing is done.
- The partial remainder is shifted left and the numbers are compared again. The comparison can be determined prior to the subtraction by inspecting the end-carry out of the parallel-adder prior to its transfer to register E.

- Nonrestoring method :In the nonrestoring method, B is not added if the difference is negative but instead, the negative difference is shifted left and then B is added.
- In the nonrestoring method, B is subtracted if the previous value of Q, was a 1, but B is added if the previous value of Q. was a 0 and no restoring of the partial remainder is required.
- This process saves the step of adding the divisor if A is less than B, but it requires special control logic to remember the previous result.
- The first time the dividend is shifted, B must be subtracted. Also, if the last bit of the quotient is 0, the partial remainder must be restored to obtain the correct final remainder.

Floating point Arithmetic operations

- A floating point number in computer registers consists of two parts: a mantissa m and an exponent e. The two parts represent a number obtained from multiplying m times a radix r raised to the value of e; thus

$$m \times r^e$$

- The mantissa may be a fraction or an integer. The location of the radix point and the value of the radix r are assumed and are not included in the registers.

- For example, assume a fraction representation and a radix 10. The decimal number 537.25 is represented in a register with m = 53725 and e = 3 and is interpreted to represent the floating-point number .53725 X 10^3

- A floating-point number is normalized if the most significant digit of the mantissa is nonzero. A zero cannot be normalized because it does not have a nonzero digit. Floating-point representation increases the range of numbers that can be accommodated in a given register.

- Consider the addition of the following floating-point numbers:

$$.5372400 \times 10^{2}$$
$$+ .1580000 \times 10^{-1}$$

- It is necessary that the two exponents be equal before the mantissas can be added. We can either shift the first number three positions to the left, or shift the second number three positions to the right. When the mantissas are stored in registers, shifting to the left causes a loss of most significant digits. Shifting to the right causes a loss of least significant digits.

- The second method is preferable because it only reduces the accuracy, while the first method may cause an error. The usual alignment procedure is to shift the mantissa that has the smaller exponent to the right by a number of places equal to the difference between the exponents. After this is done, the mantissas can be added:

$$
\begin{array}{r}
.5372400 \times 10^{2} \\
+.0001580 \times 10^{2} \\
\hline
.5373980 \times 10^{2}
\end{array}
$$

- When two normalized mantissas are added, the sum may contain an overflow digit. An overflow can be corrected easily by shifting the sum once to the right and incrementing the exponent.

- When two numbers are subtracted, the result may contain most significant zero's as shown in the following example:

$$\begin{array}{r} .56780 \times 10^5 \\ -\ .56430 \times 10^5 \\ \hline .00350 \times 10^5 \end{array}$$

- A floating-point number that has a 0 in the most significant position of the mantissa is said to have an underflow. To normalize a number that contains an underflow, it is necessary to shift the mantissa to the left and decrement the exponent until a nonzero digit appears in the first position. In the example above, it is necessary to shift left twice to obtain $.35000 \times 10^3$

- Floating-point multiplication and division do not require an alignment of the mantissas. The product can be formed by multiplying the two mantissas and adding the exponents. Division is accomplished by dividing the mantissas and subtracting the exponents.

- The operations performed with the mantissas are the same as in fixed point numbers, so the two can share the same registers and circuits. The operations performed with the exponents are compare and increment (for aligning the mantissas), add and subtract (for multiplication and division), and decrement (to normalize the result).

- The exponent may be represented in any one of the three representations: signed-magnitude, signed-2's complement, or signed-1's complement.

- A fourth representation employed in many computers is known as a biased exponent. In this representation, the sign bit is removed from being a separate entity.

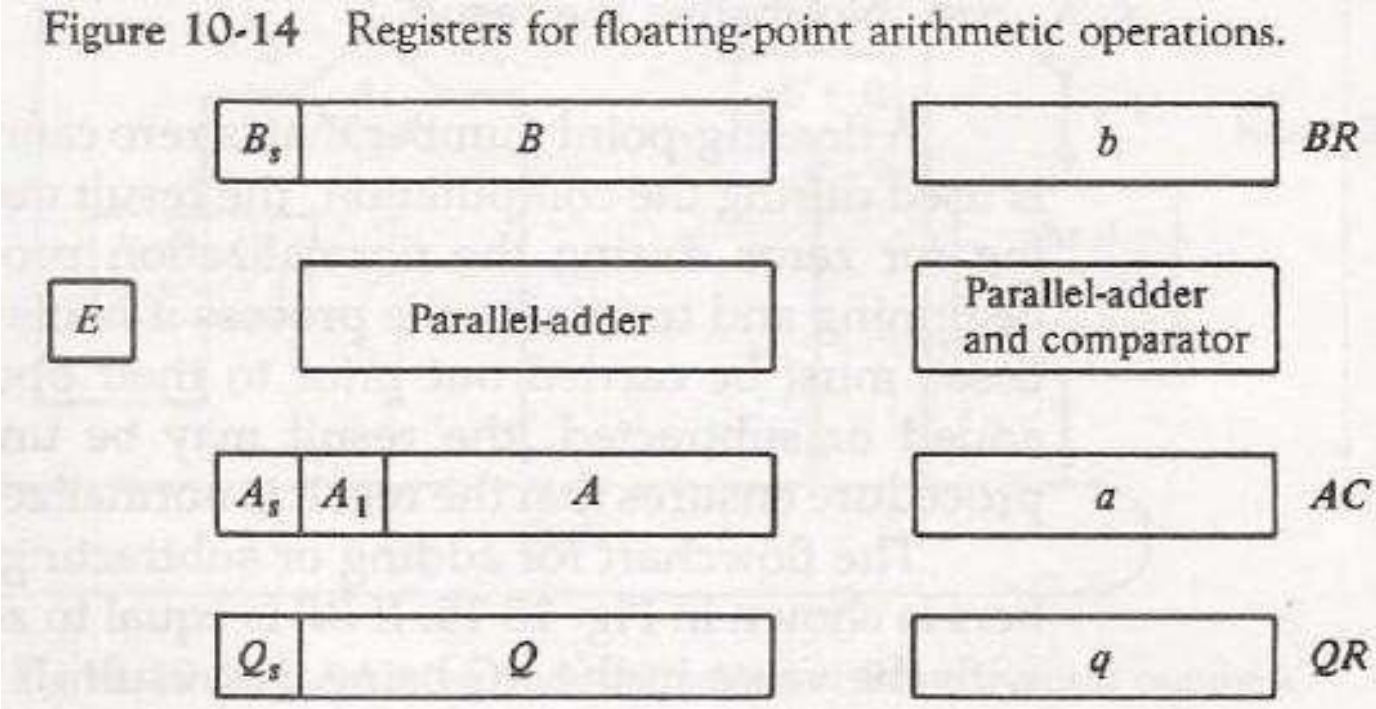
- The bias is a positive number that is added to each exponent as the floating-point number is formed, so that internally all exponents are positive.

- Consider an exponent that ranges from -50 to 49. Internally, it is represented by two digits (without a sign) by adding to it a bias of 50.

- The exponent register contains the number e + 50, where e is the actual exponent. This way, the exponents are represented in registers as positive numbers in the range of 00 to 99.

- The advantage of biased exponents is that they contain only positive numbers. It is then simpler to compare their relative magnitude without being concerned with their signs.

Register Configuration



It is assumed that each floating-point number has a mantissa in signed magnitude representation and a biased exponent.

Addition and subtraction

1. Check for zeros.
2. Align the mantissas.
3. Add or subtract the mantissas.
4. Normalize the result.

- The addition and subtraction of the two mantissas is identical to the fixed-point addition and subtraction algorithm. The magnitude part is added or subtracted depending on the operation and the signs of the two mantissas. If an overflow occurs when the magnitudes are added, it is transferred into flip-flop E.

- If E is equal to 1, the bit is transferred into A1 and all other bits of A are shifted right. The exponent must be incremented to maintain the correct number.

- No underflow may occur in this case because the original mantissa that was not shifted during the alignment was already in a normalized position.

- If the magnitudes were subtracted, the result may be zero or may have an underflow. If the mantissa is zero, the entire floating-point number in the AC is made zero.

- Otherwise, the mantissa must have at least one bit that is equal to 1. The mantissa has an underflow if the most significant bit in position A1 is 0.

- In that case, the mantissa is shifted left and the exponent decremented. The bit in A1 is checked again and the process is repeated until it is equal to 1. When A1 = 1, the mantissa is normalized and the operation is completed.

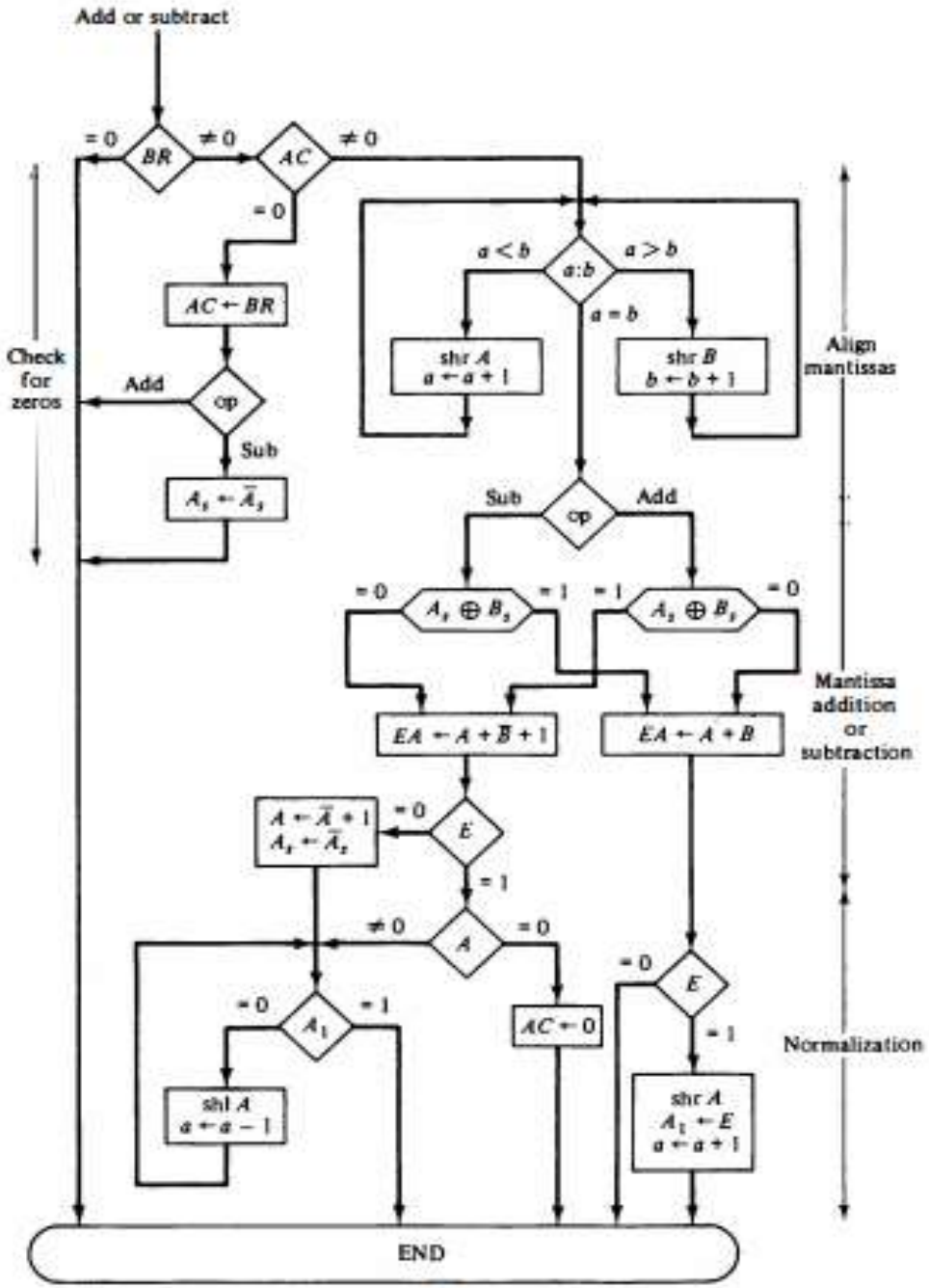


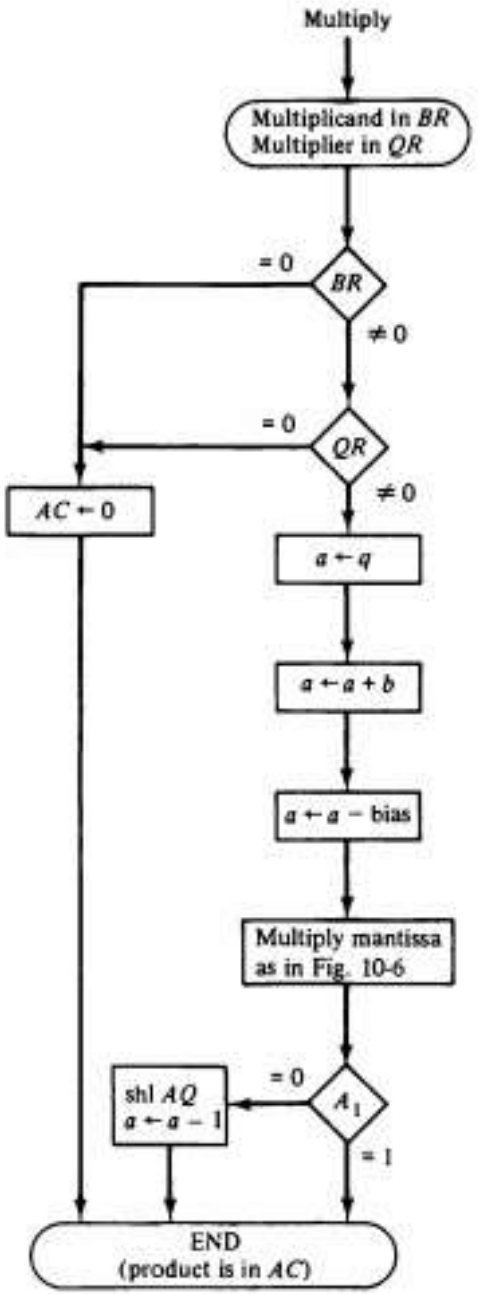
Figure 10-15 Addition and subtraction of floating-point numbers.

Multiplication

The multiplication algorithm can be subdivided into four parts:

1. Check for zeros.
2. Add the exponents.
3. Multiply the mantissas.
4. Normalize the product.

Figure 10-16 Multiplication of floating-point numbers.



- Overflow cannot occur during multiplication, so there is no need to check for it.

- The product may have an underflow, so the most significant bit in A is checked. If it is a 1, the product is already normalized.

- If it is a 0, the mantissa in AQ is shifted left and the exponent decremented. Note that only one normalization shift is necessary. The multiplier and multiplicand were originally normalized and contained fractions.

- The smallest normalized operand is 0.1, so the smallest possible product is 0.01.

- Therefore, only one leading zero may occur. Although the low-order half of the mantissa is in Q, we do not use it for the floating-point product. Only the value in the AC is taken as the product.

Division:

1. Check for zeros.
2. Initialize registers and evaluate the sign.
3. Align the dividend.
4. Subtract the exponents.
5. Divide the mantissas.

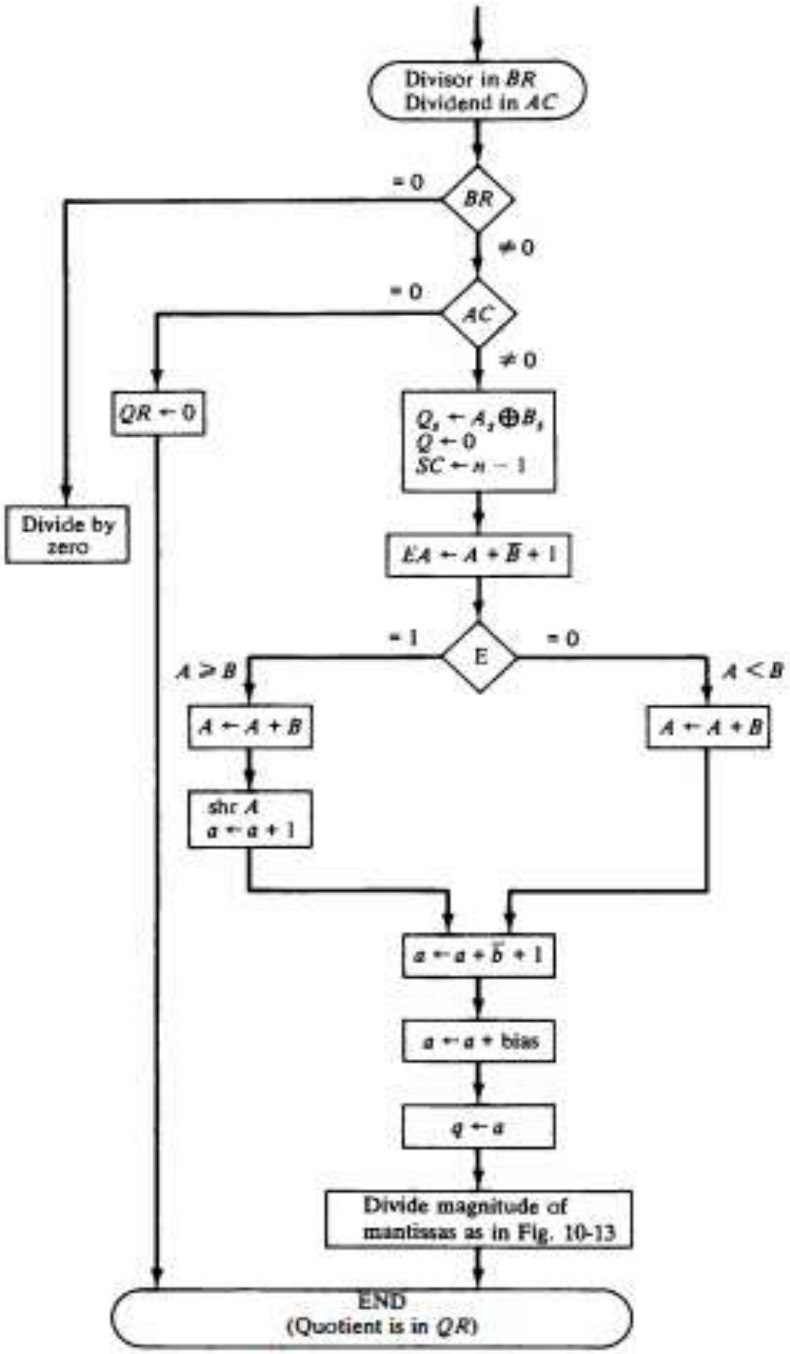


Figure 10-17 Division of floating-point numbers.

- The check for divide-overflow is the same as in fixed-point representation. However, with floating-point numbers the divide-overflow imposes no problems.

- If the dividend is greater than or equal to the divisor, the dividend fraction is shifted to the right and its exponent incremented by 1.

- For normalized operands this is a sufficient operation to ensure that no mantissa divide overflow will occur.

- The operation above is referred to as a dividend alignment. The division of two normalized floating-point numbers will always result in a normalized quotient provided that a dividend alignment is carried out before the division.

- Therefore, unlike the other operations, the quotient obtained after the division does not require a normalization

- The dividend alignment is similar to the divide-overflow check in the fixed-point operation. The proper alignment requires that the fraction dividend be smaller than the divisor.

- The two fractions are compared by a subtraction test. The carry in E determines their relative magnitude. The dividend fraction is restored to its original value by adding the divisor.

- If A >=B, it is necessary to shift A once to the right and increment the dividend exponent. Since both operands are normalized, this alignment ensures that A < B.