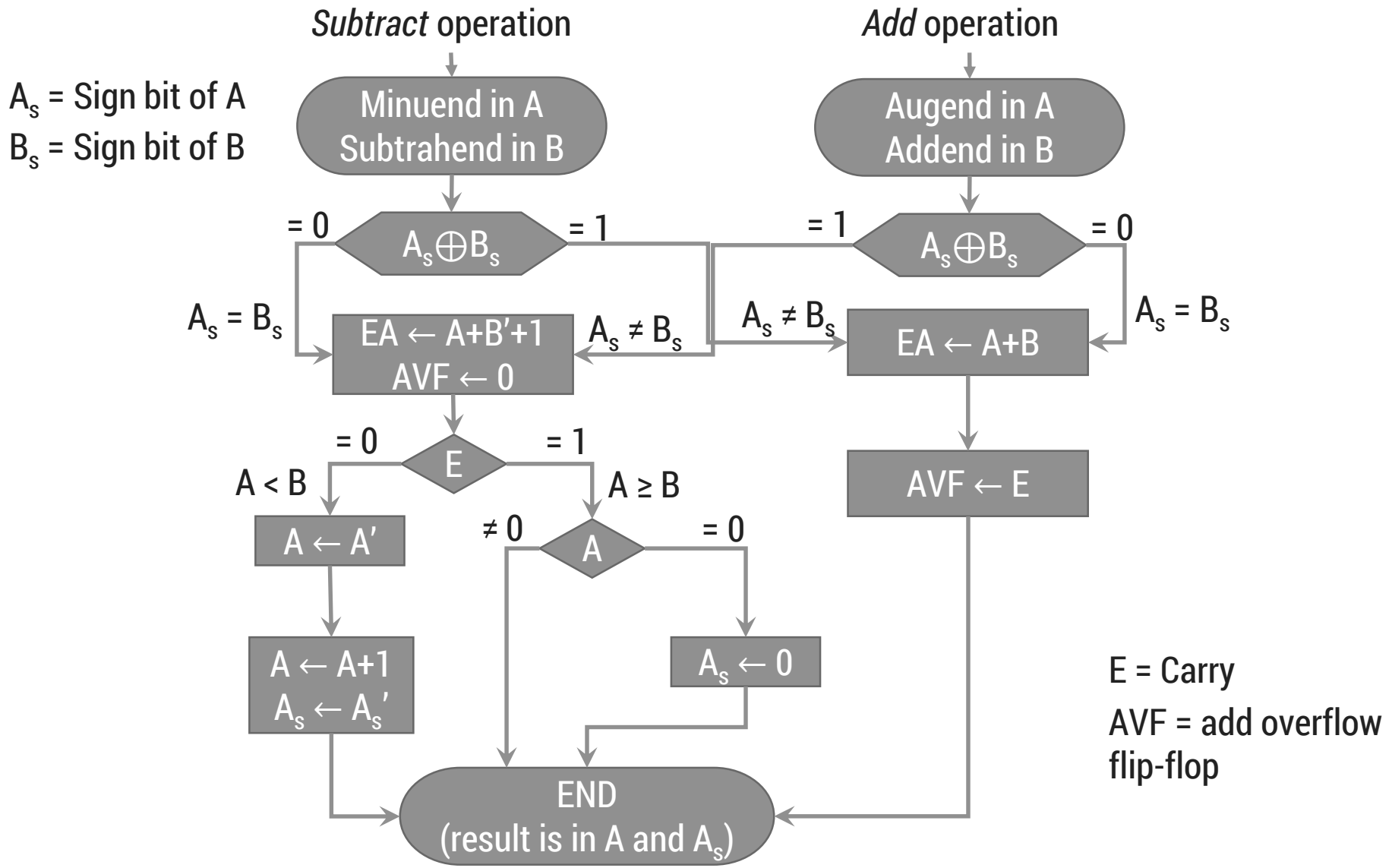
Unit VII
Computer Arithmetic

Addition and Subtraction

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

Flowchart for Addition & Subtraction



Hardware for signed magnitude addition and subtraction

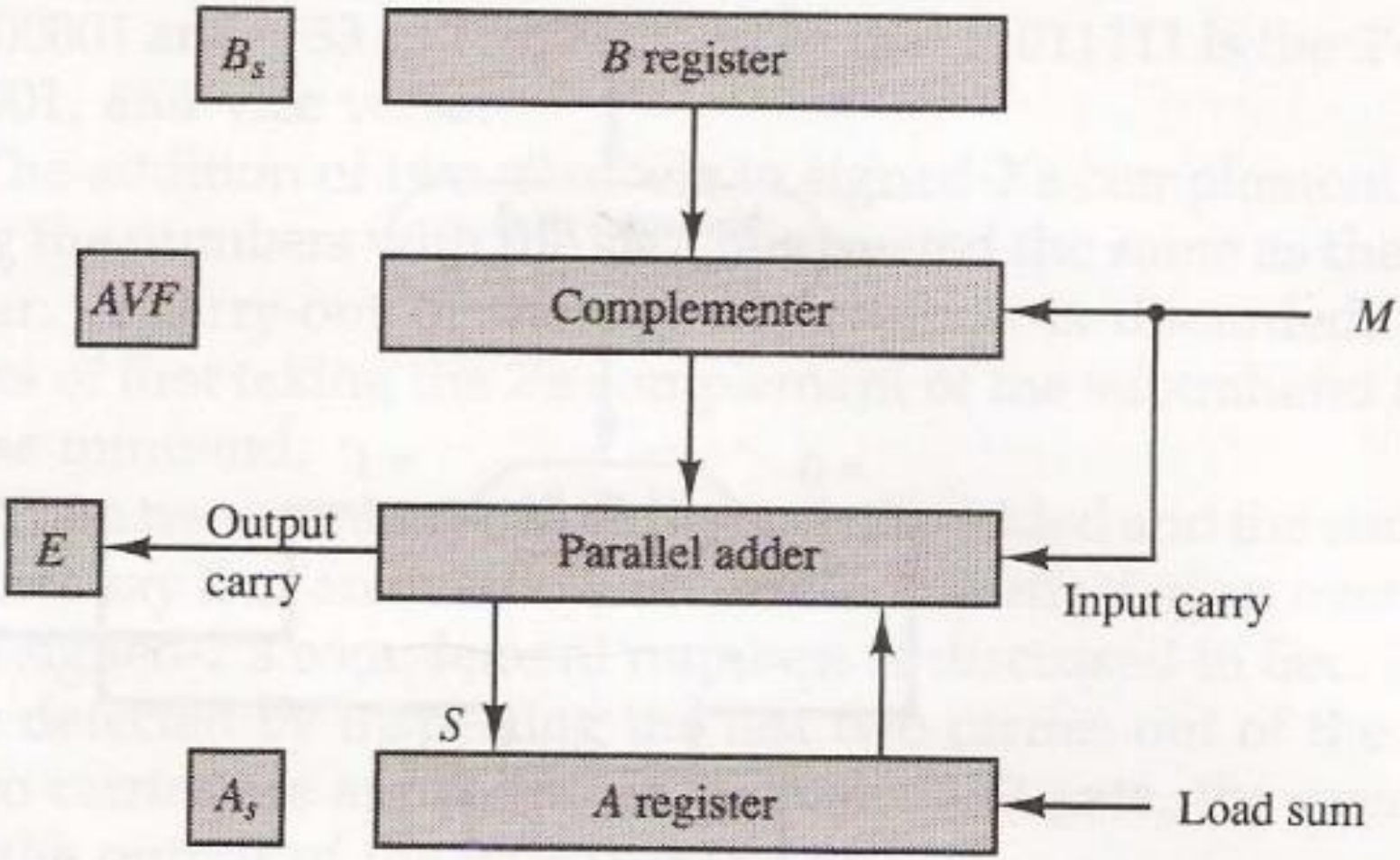
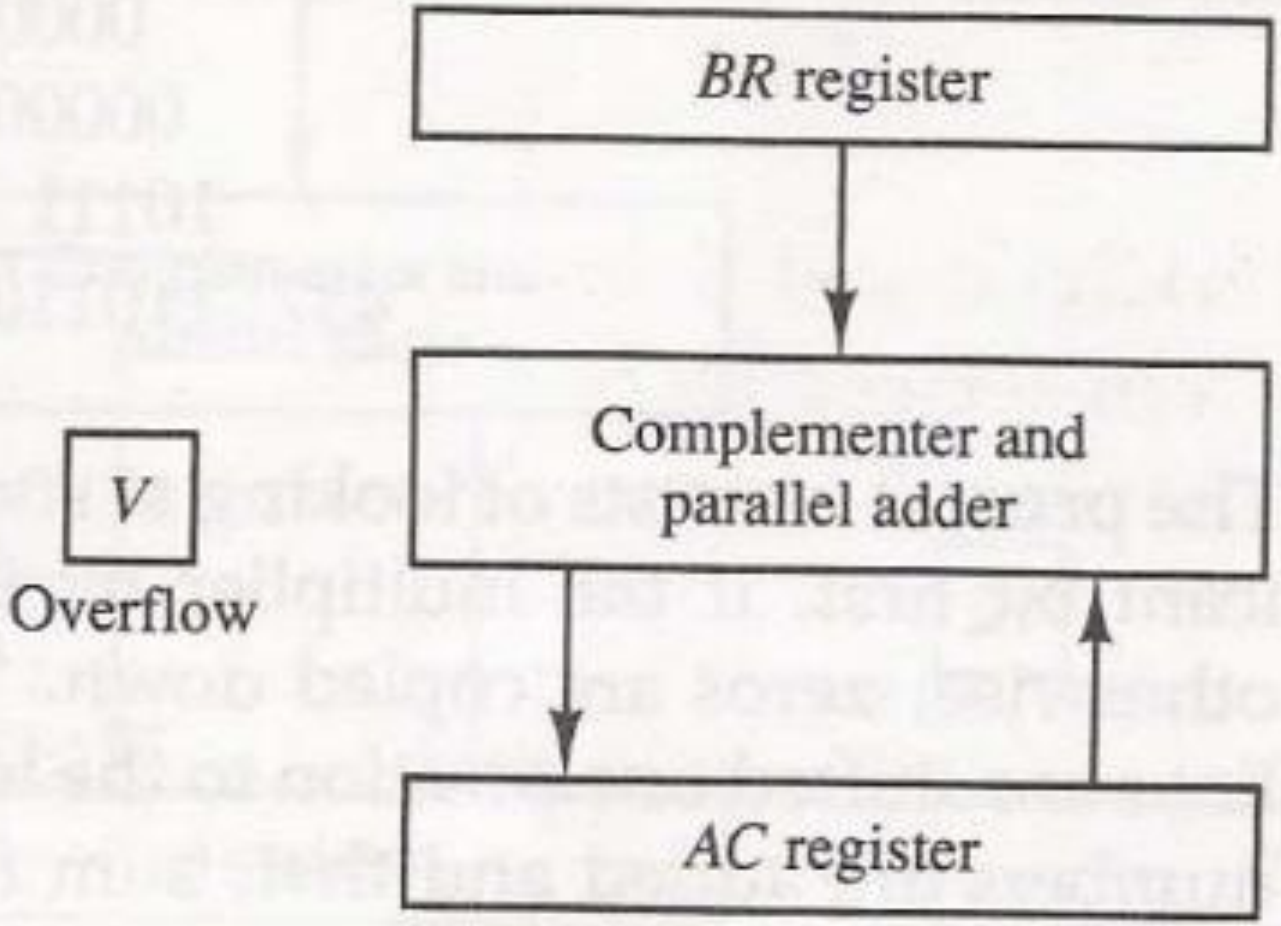


Figure 10-1 Hardware for signed-magnitude addition and subtraction.

M=0 then no complement
M=1 then complementer perform 1s complement and in parallel adder 1 is added so 2s complement is performed

Addition and subtraction with Signed 2's Complement data

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



Algoritham

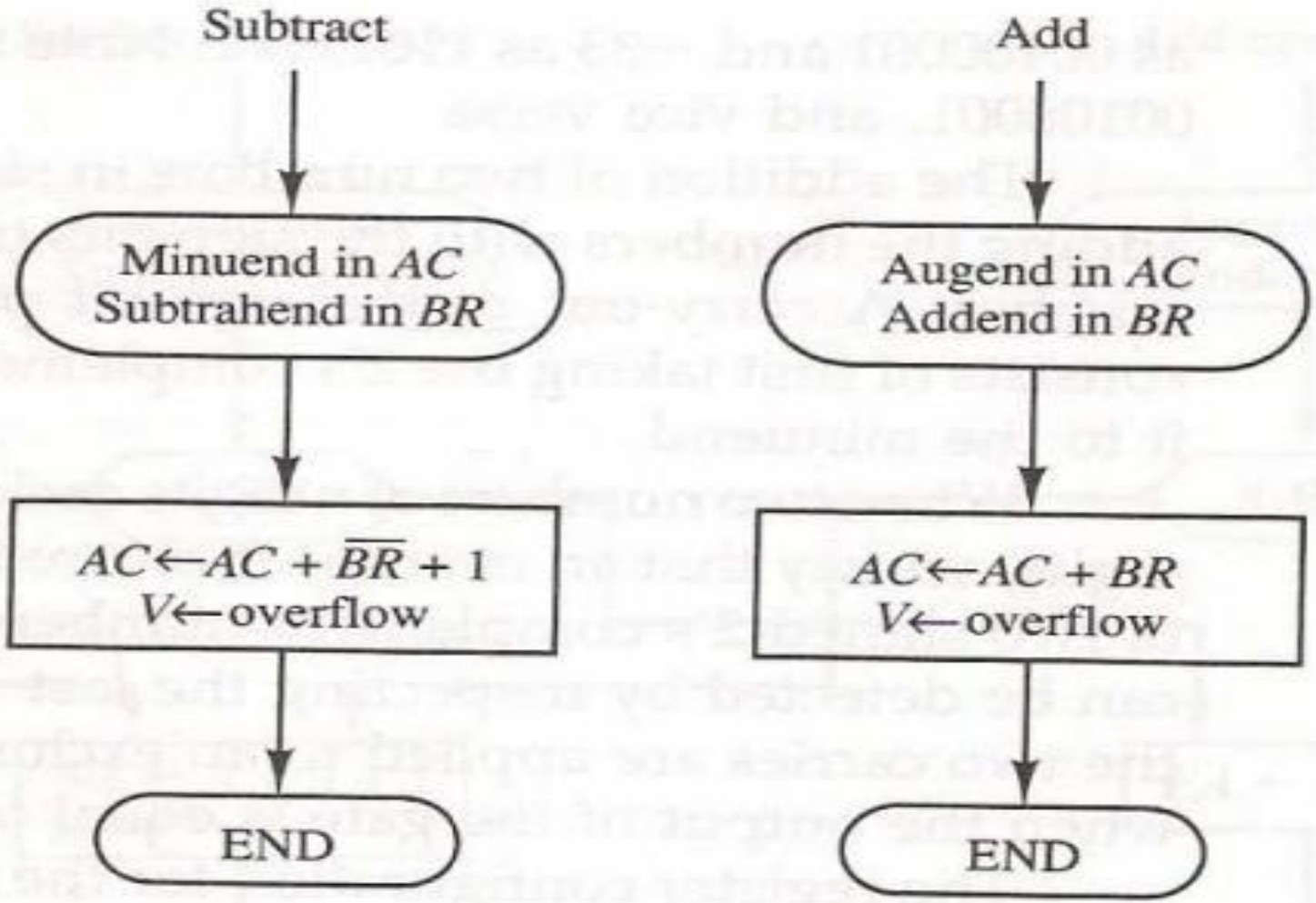
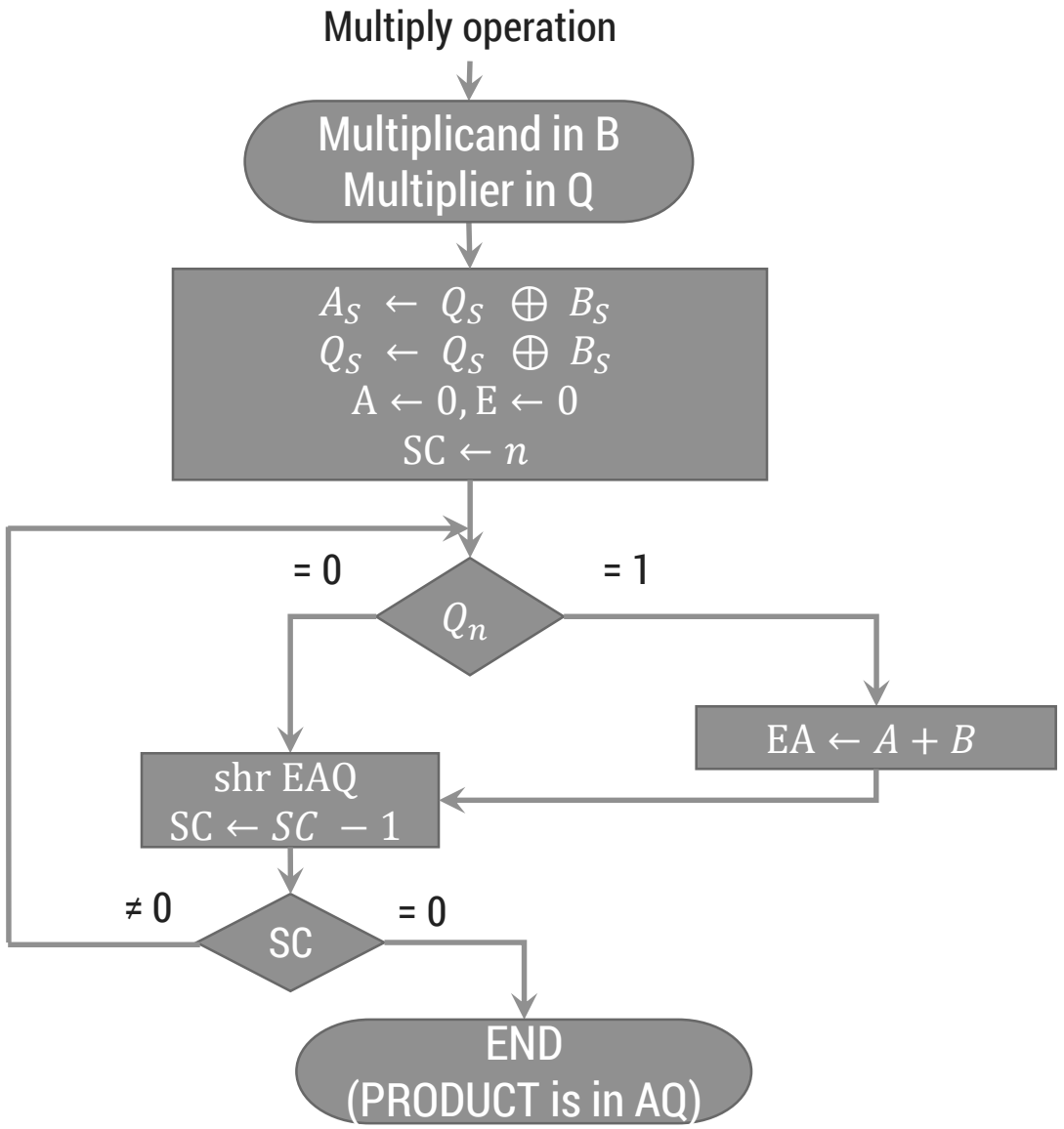


Figure 10-4 Algorithm for adding and subtracting numbers in signed-2's complement representation.

Multiplication

- Traditional way of binary multiplication

23	10111
x19	x 10011
	10111
	10111
	00000
	00000
	10111
437	110110101



Hardware for multiply operation

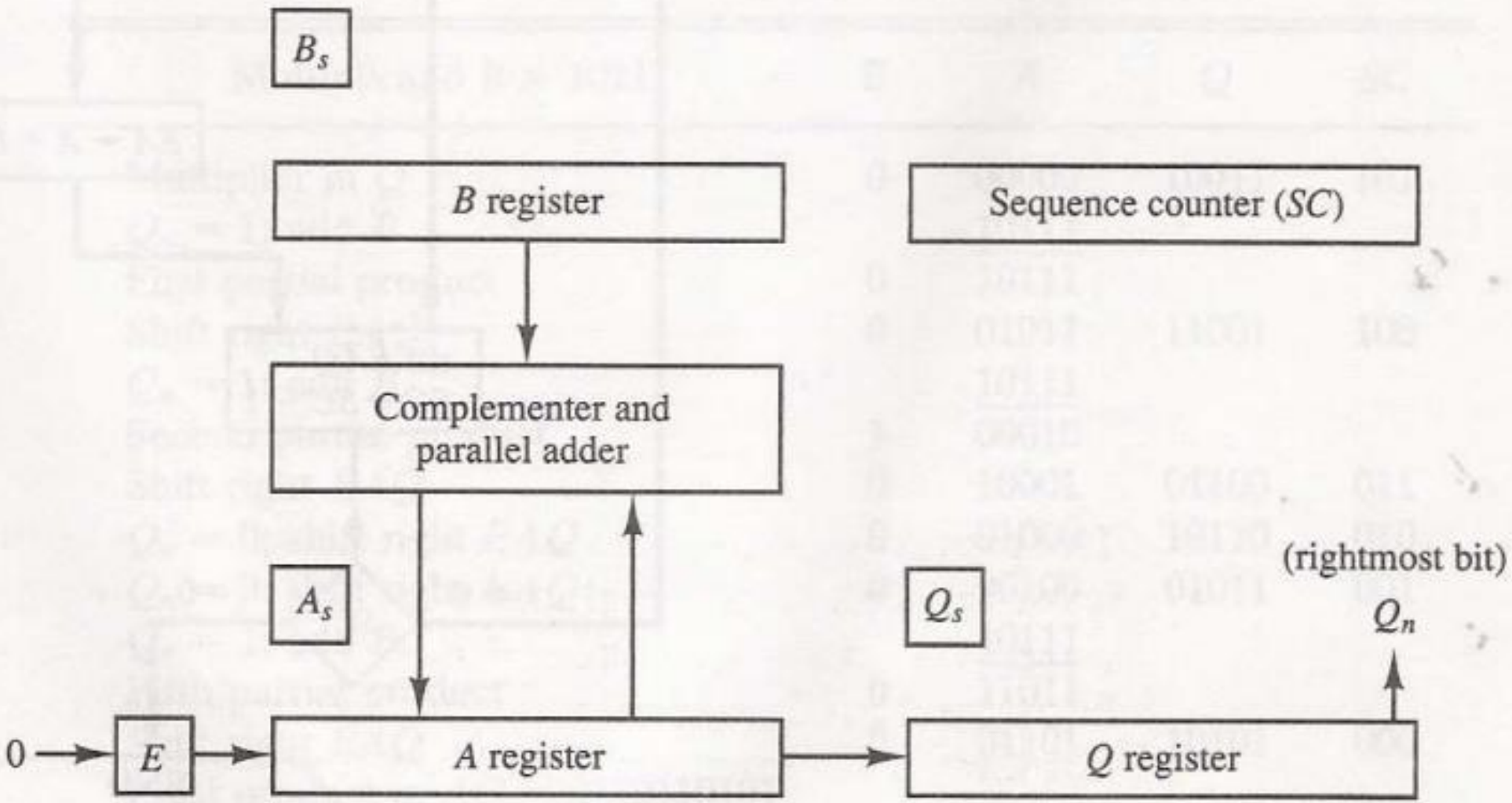


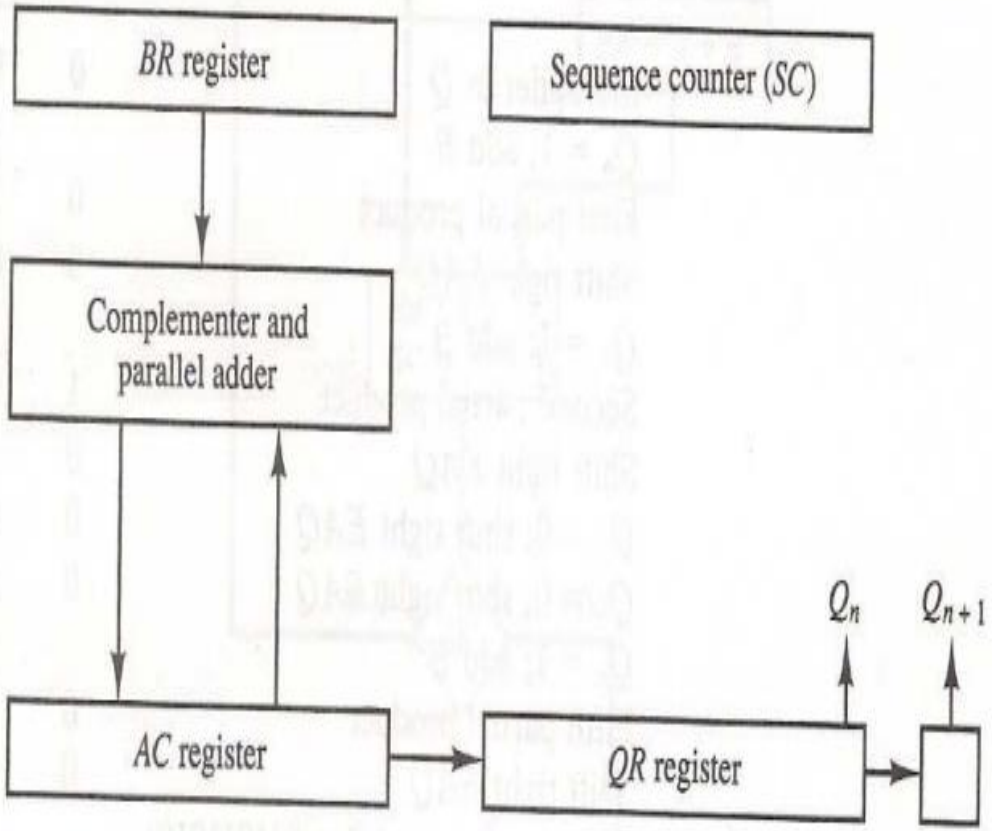
Figure 10-5 Hardware for multiply operation.

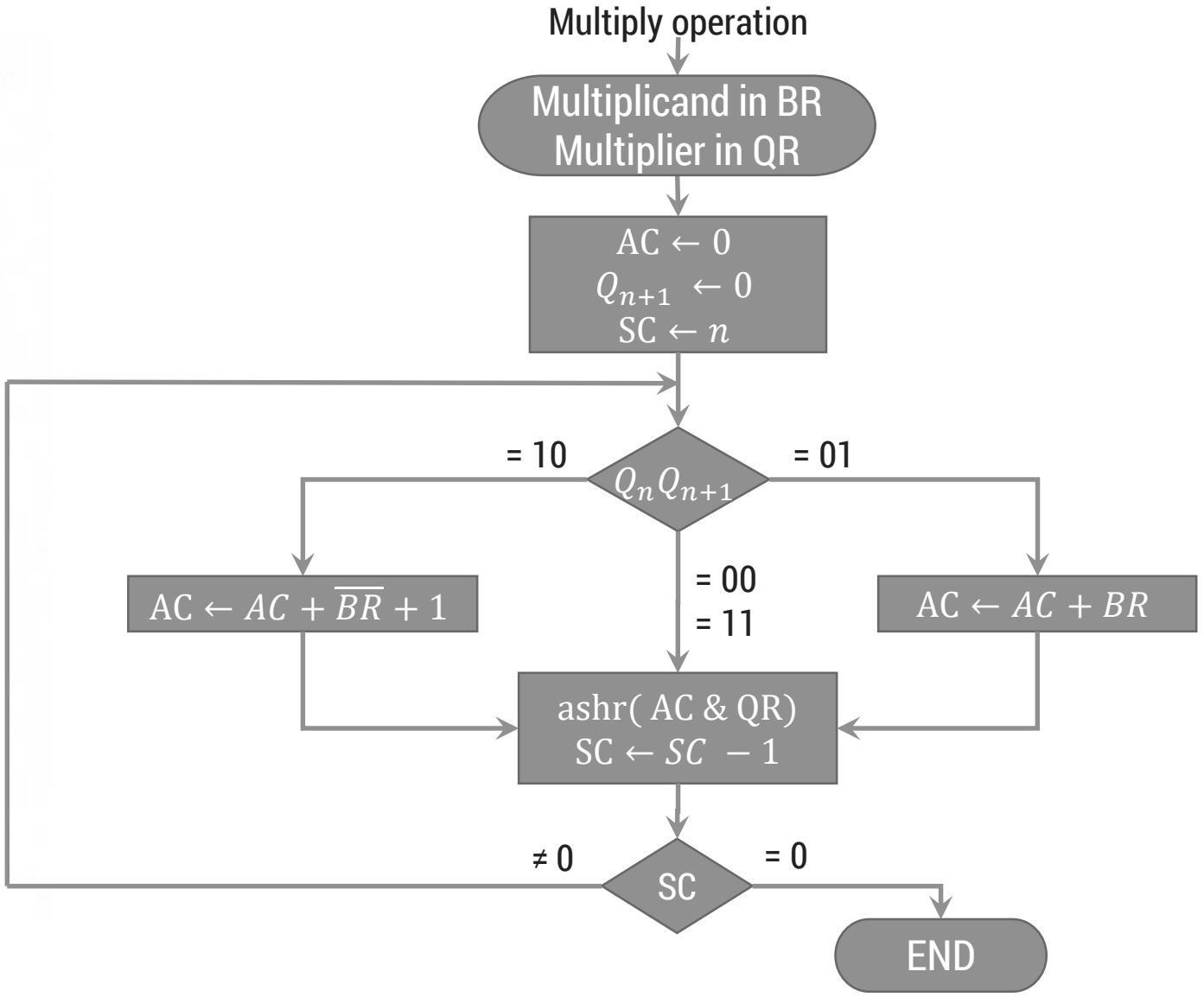
Perform 23 x 19

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

Figure 10-7 Hardware for Booth algorithm.



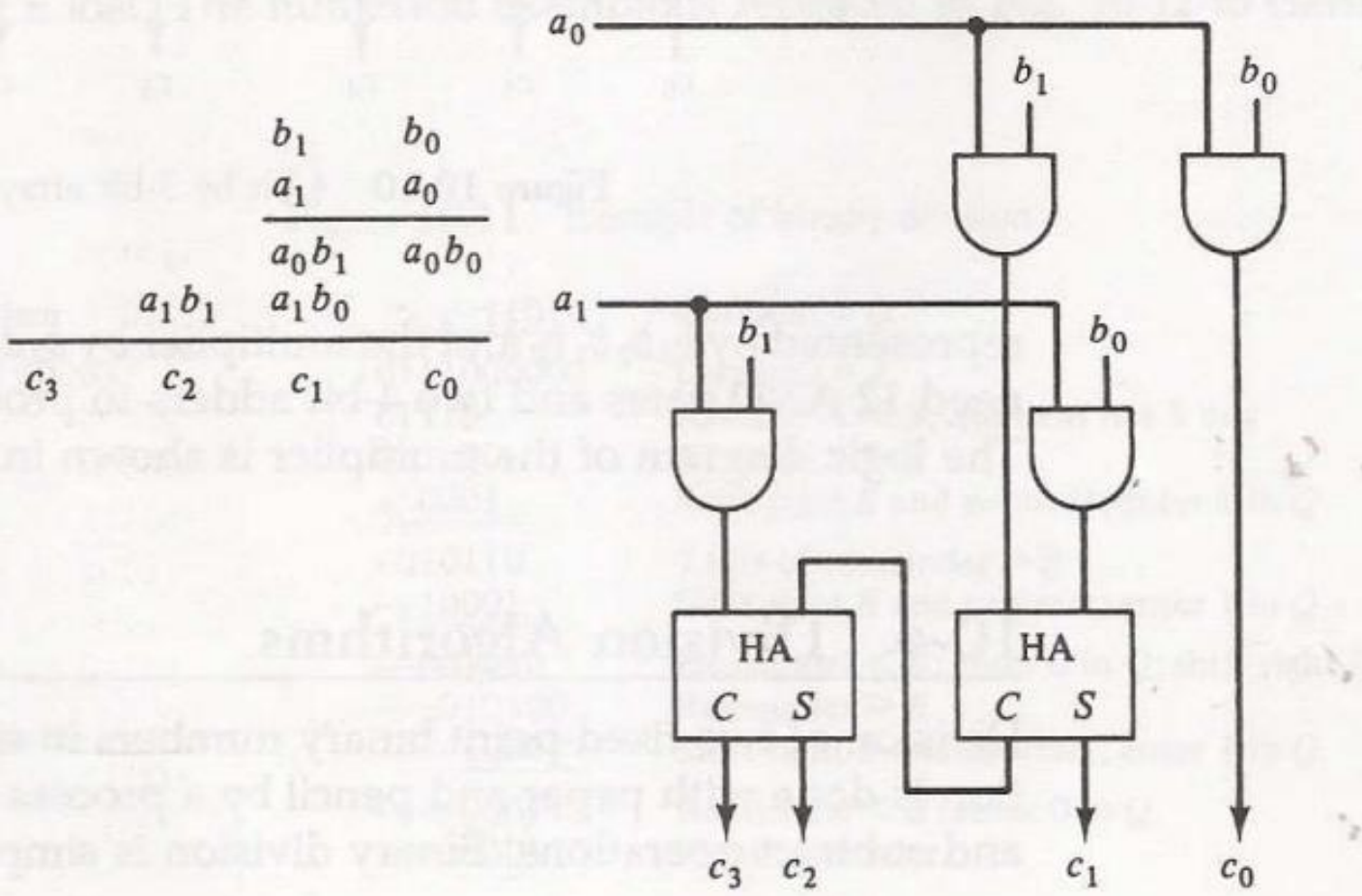


Multiply (-9) x (-13) using Booth Algorithm

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

Figure 10-9 2-bit by 2-bit array multiplier.



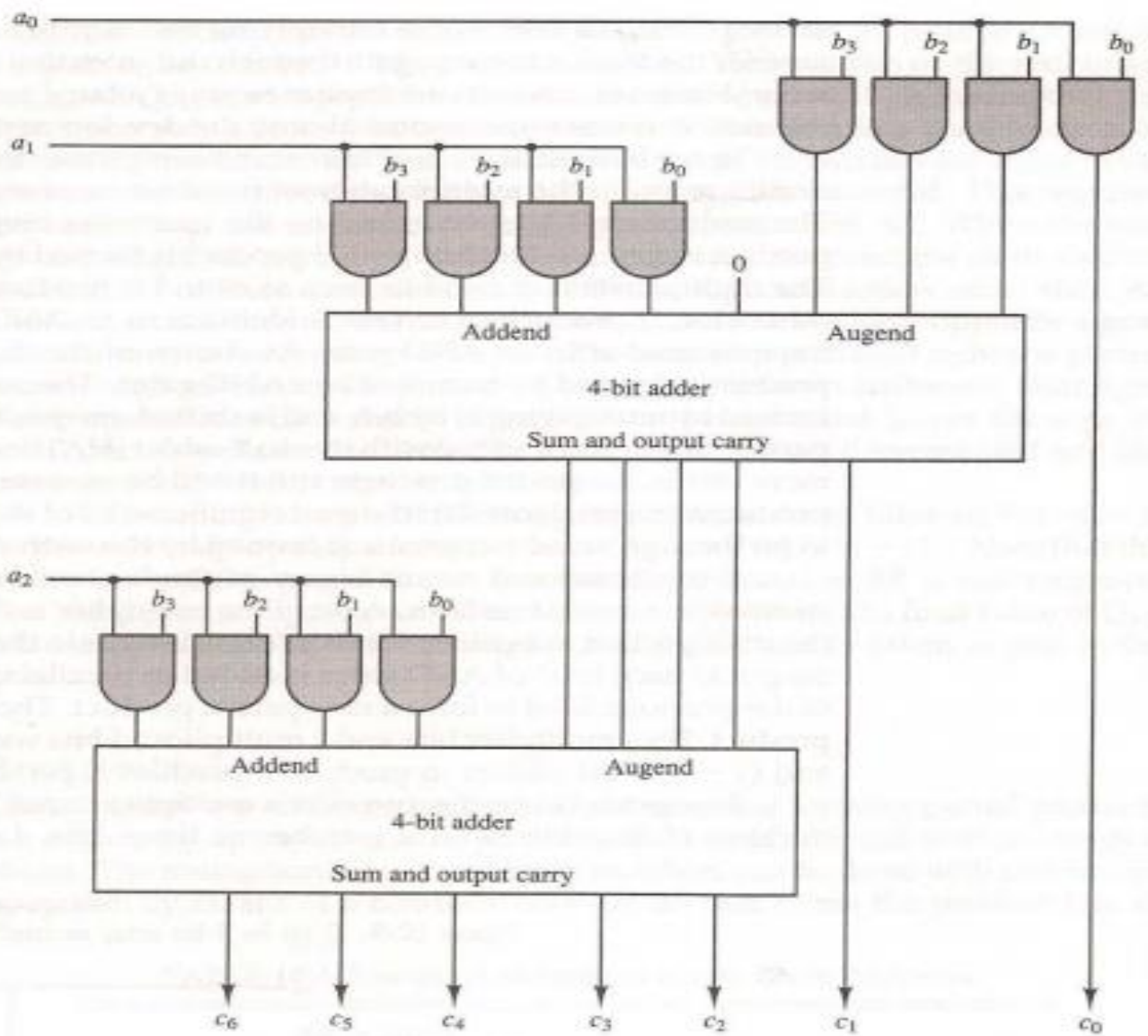
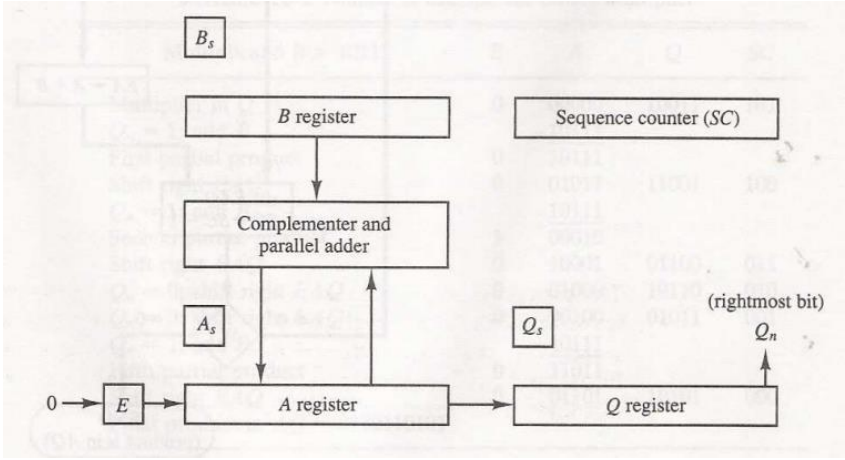


Figure 10-10 4-bit by 3-bit array multiplier.

Division Algoritham

Figure 10-11 Example of binary division.

Divisor:		
$B = 10001$	11010	Quotient = Q
	0111000000	Dividend = A
	01110	5 bits of $A < B$, quotient has 5 bits
	011100	6 bits of $A \geq B$
	-10001	Shift right B and subtract; enter 1 in Q
	-010110	7 bits of remainder $\geq B$
	--10001	Shift right B and subtract; enter 1 in Q
	--001010	Remainder $< B$; enter 0 in Q; shift right B
	---010100	Remainder $\geq B$
	----10001	Shift right B and subtract; enter 1 in Q
	----000110	Remainder $< B$; enter 0 in Q
	-----00110	Final remainder

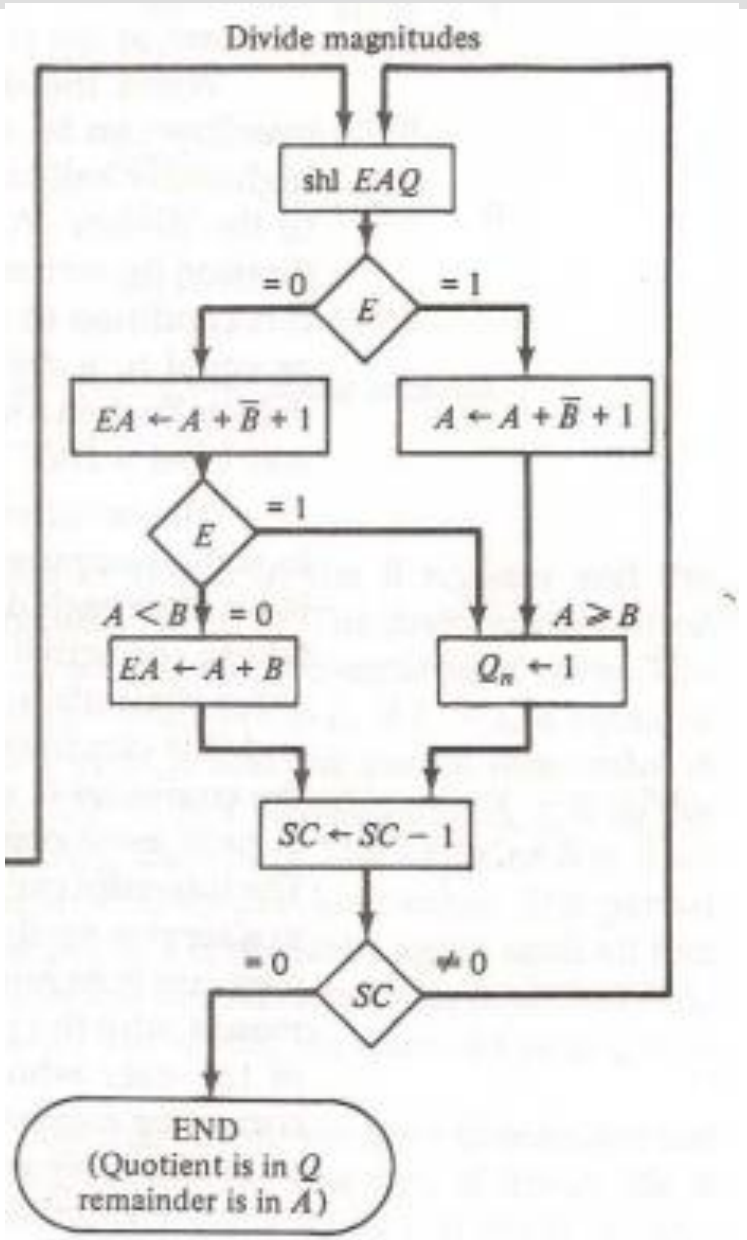


- Initially, the dividend is in A & Q and the divisor is in B
- Sign of result is transferred into Q, to be the part of quotient
- Then a constant is set into the SC to specify the number of bits in the quotient
- Since an operand must be saved with its sign, one bit of the word will be inhabited by the sign, and the magnitude will be composed of n -1 bits
- The condition of divide-overflow is checked by subtracting the divisor in B from the half of bits of the dividend stored in A
- If A ≥ B, DVF is set and the operation is terminated before time
- If A < B, no overflow condition occurs and so the value of the dividend is reestablished by adding B to A
- The division of the magnitudes starts by shl dividend in AQ to left in the high-order bit shifted into E. *(Note: If shifted a bit into E is equal to 1, and we know that EA > B as EA comprises a 1 followed by n -1 bits whereas B comprises only n -1 bits). In this case, B must be subtracted from EA, and 1 should insert into Q, for the quotient bit*
- If the shift-left operation (shl) inserts a 0 into E, the divisor is subtracted by adding its 2's complement value and the carry is moved into E. If E = 1, it means that A ≥ B; thus, Q, is set to 1. If E = 0, it means that A < B and the original number is reimposed by adding B into A

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

	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		
Restore remainder	1	01010		2
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	

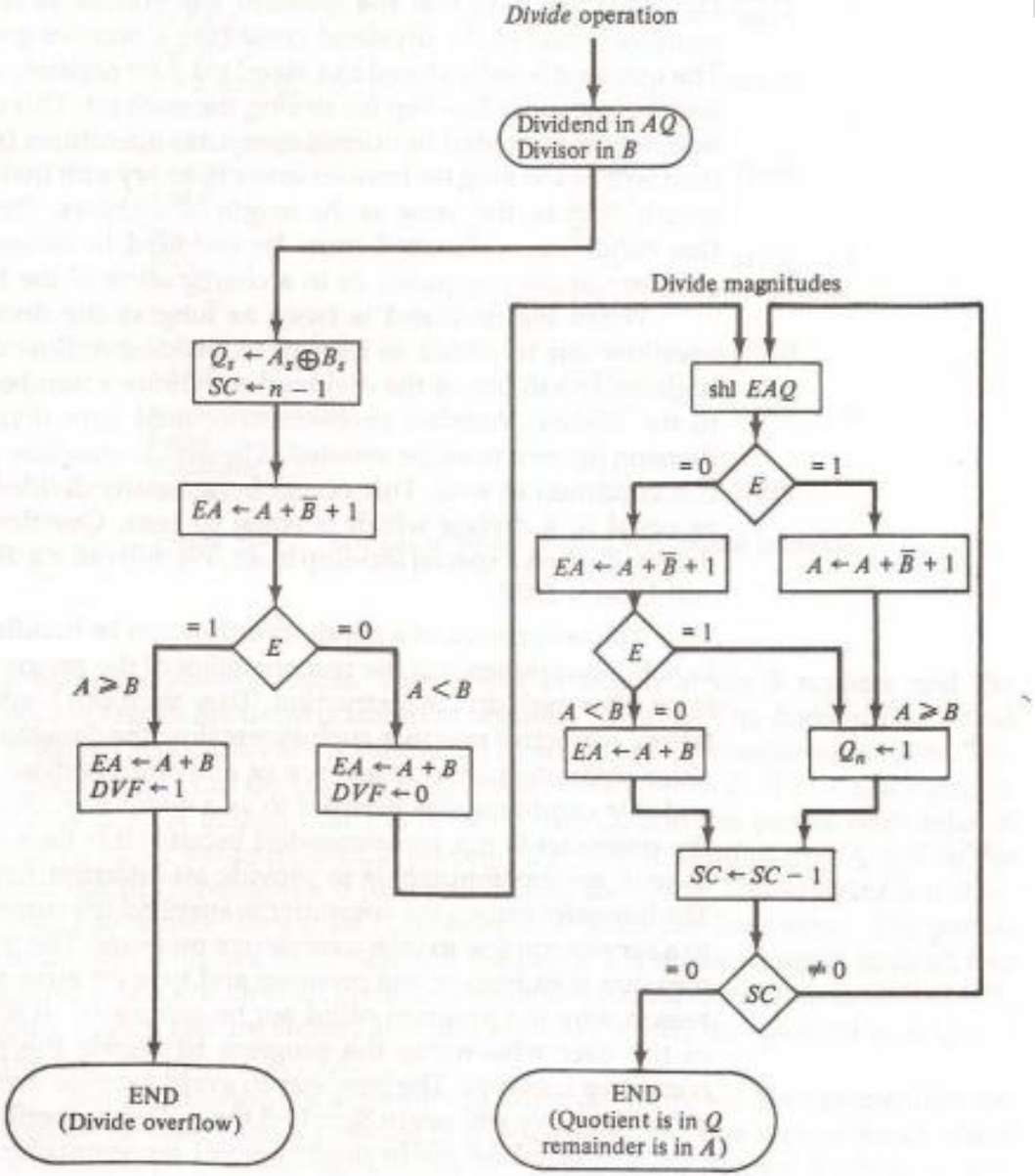
Figure 10-12 Example of binary division with digital hardware.



Divide Overflow

When the dividend is twice as long as the divisor, the condition for overflow can be stated as follows: A divide-overflow condition occurs if the high-order half bits of the dividend constitute a number greater than or equal to the divisor. Another problem associated with division is the fact that a division by zero must be avoided. The divide-overflow condition takes care of this condition as well. This occurs because any dividend will be greater than or equal to a divisor which is equal to zero. Overflow condition is usually detected when a special flip-flop is set. We will call it a divide-overflow flip-flop and label it *DVF*.

Figure 10-13 Flowchart for divide operation.



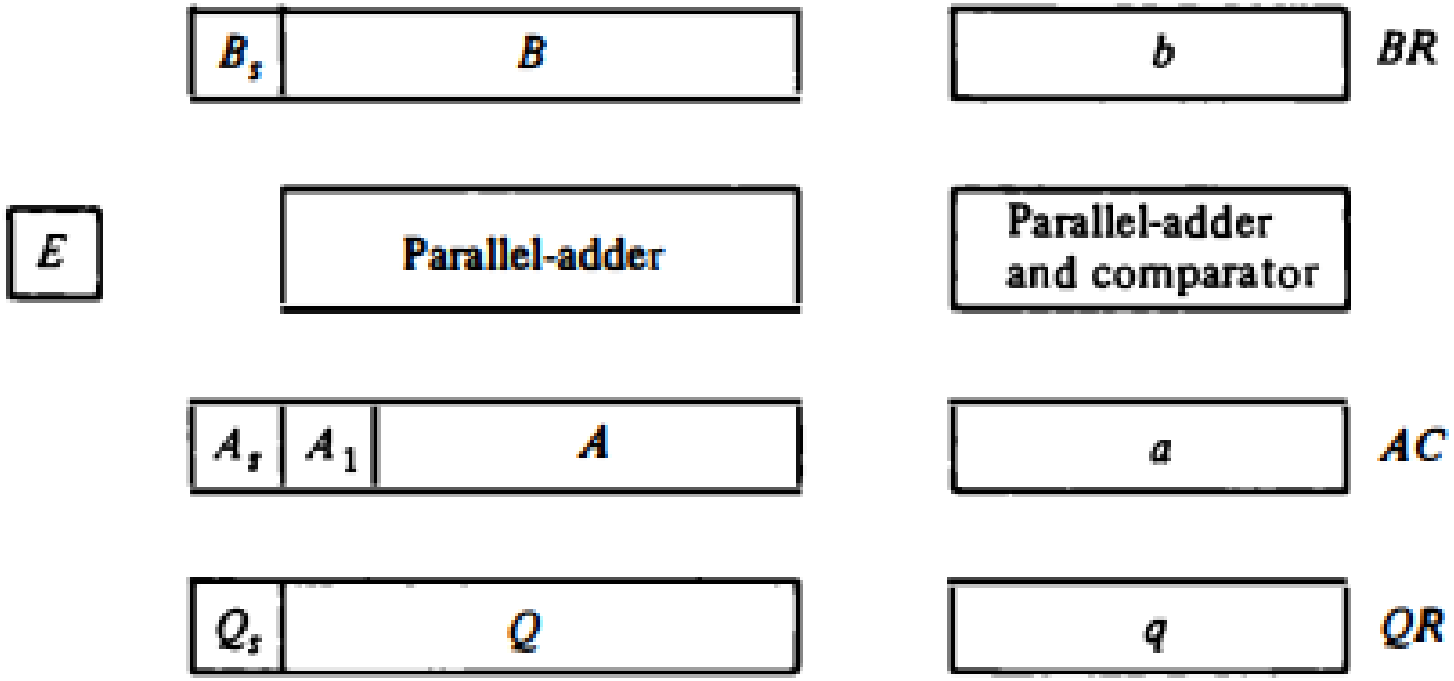
Floating Point Arithmatic

- Many high-level programming languages have a facility for specifying floating point numbers. The most common way is to specify them by a real declaration statement as opposed to fixed-point numbers, which are specified by an integer declaration statement.
- 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
- mx(r^e)
- 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. In this way the mantissa contains the maximum possible number of significant digits. A zero cannot be normalized because it does not have a nonzero digit. It is represented in floating-point by all 0' sin the mantissa and exponent.

Registers for floating,point arithmetic operations.

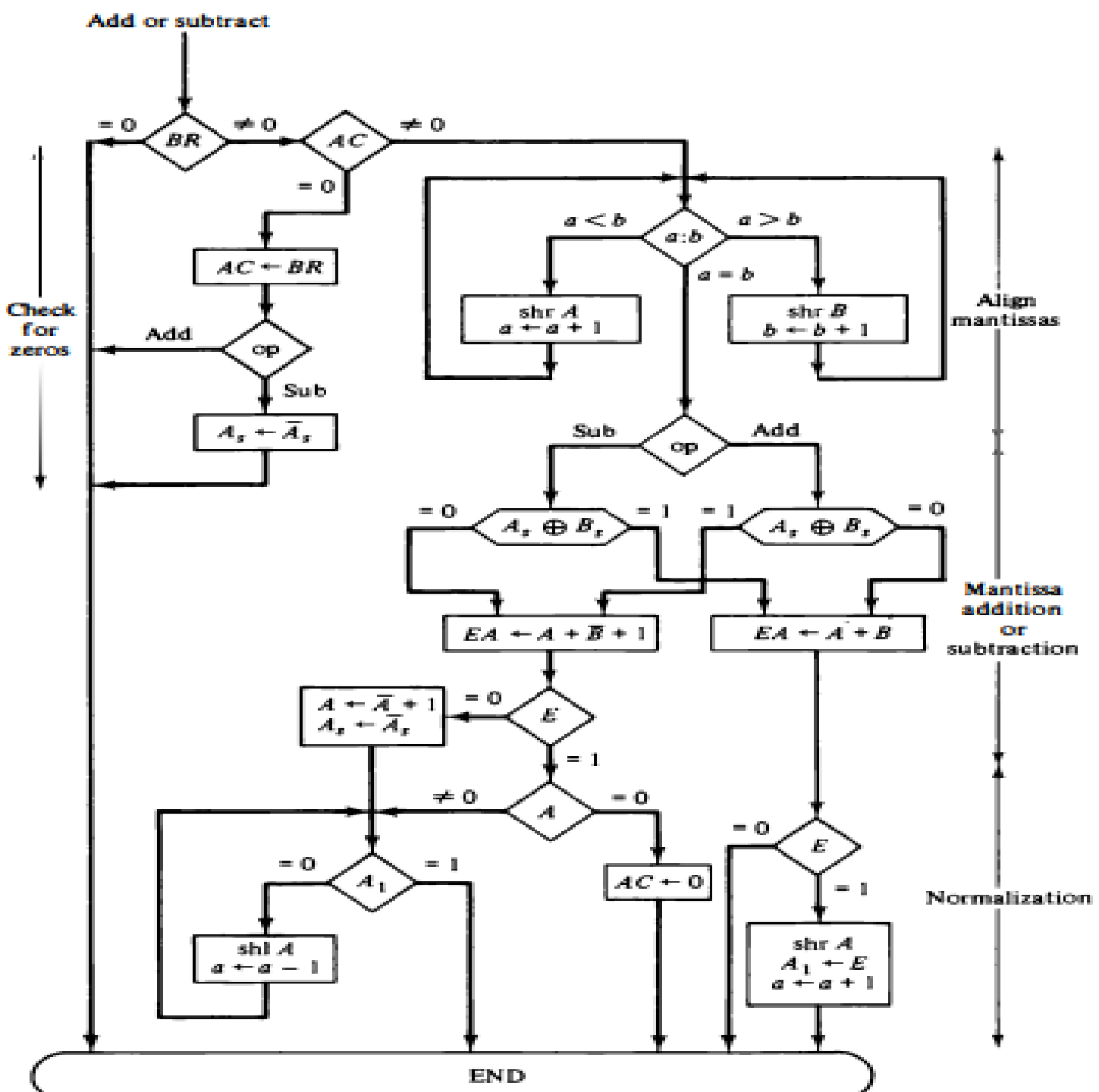
- There are three registers, BR, AC, and QR. Each register is subdivided into two parts. The mantissa part has the same uppercase letter symbols as in fixed-point representation. The exponent part uses the corresponding lower case letter symbol.

Figure 10-14 Registers for floating-point arithmetic operations.



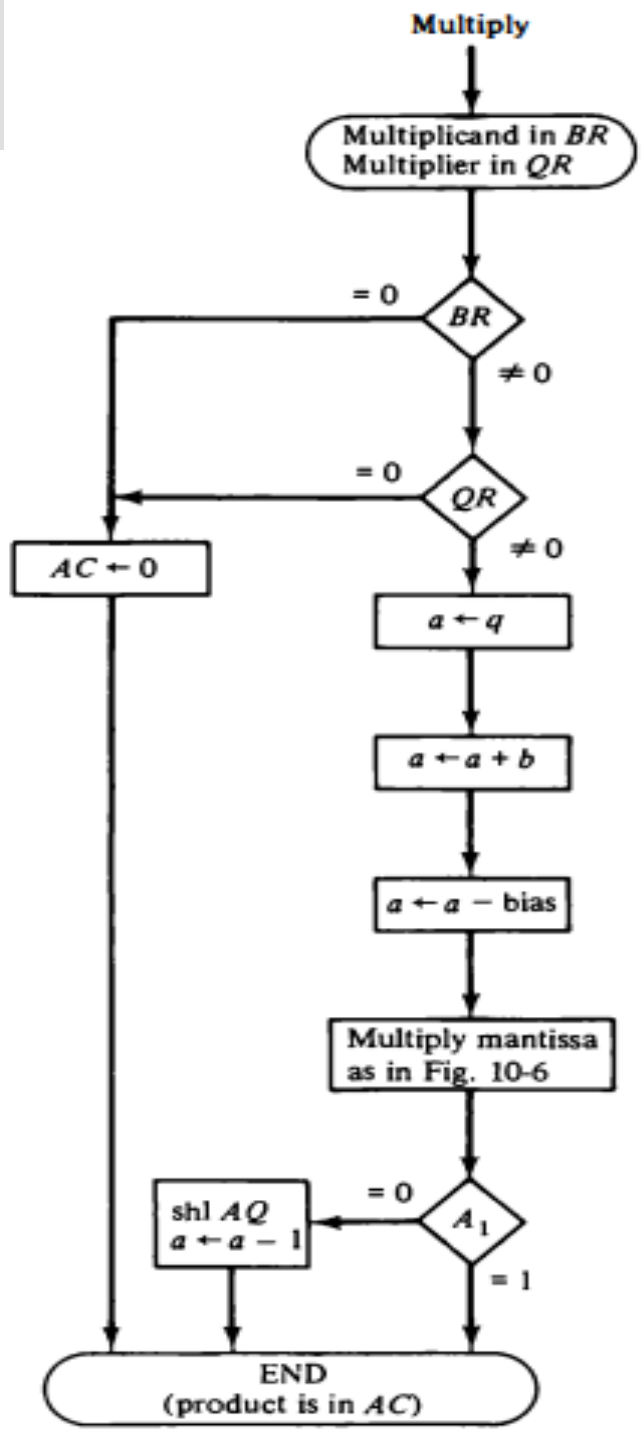
Floating point addition and subtraction

- During addition or subtraction, the two floating-point operands are in AC and BR. The sum or difference is formed in the AC. The algorithm can be divided into four consecutive parts:
- 1. Check for zeros.
- 2. Align the mantissas.
- 3. Add or subtract the mantissas.
- 4. Normalize the result.


Add or subtract
BR
AC
= 0
≠ 0
AC ← BR
Add
op
Sub
$A_s \leftarrow \overline{A}_s$
Check for zeros
a < b
a : b
a > b
a = b
shr A
a ← a + 1
shr B
b ← b + 1
Align mantissas
Sub
op
Add
$A_s \oplus B_s$
= 0
= 1
$EA \leftarrow A + \overline{B} + 1$
$EA \leftarrow A + B$
Mantissa addition or subtraction
$A \leftarrow \overline{A} + 1$
$A_s \leftarrow \overline{A}_s$
E
A
A_1
shl A
a ← a − 1
AC ← 0
shr A
$A_1 \leftarrow E$
a ← a + 1
Normalization
END

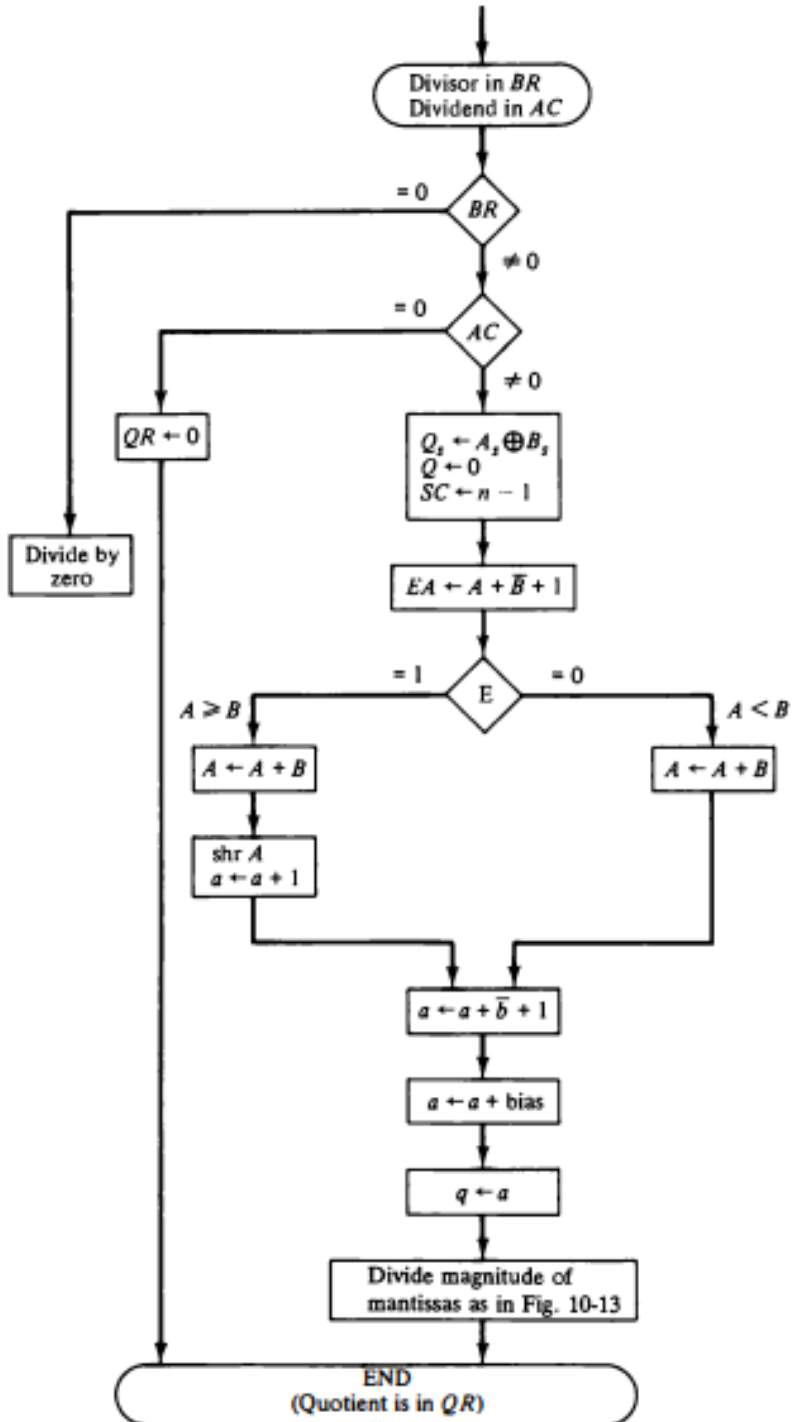
multiplication of two floating-point no.s

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

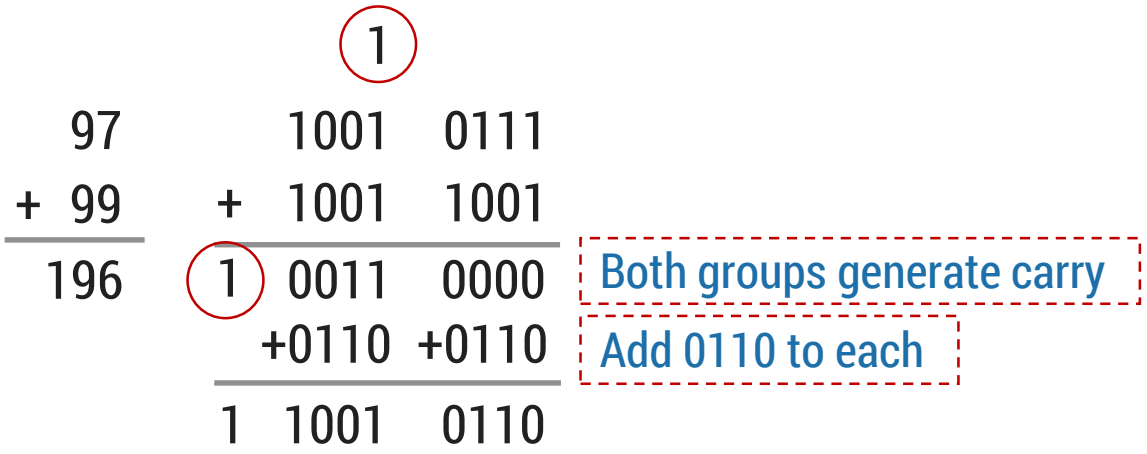


Division of Floating point no.s

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



BCD Adder



If codes are illegal or carry is generated in the group then we add 0110 to that particular group

- Two BCD digits are applied to 4-bit binary adder which produce result ranging from 0 to 19 i.e. 9 + 9 + 1 = 19
- Output sum of two decimal numbers must be represented in BCD.
- Problem is to find rule by which binary number is to be converted to correct BCD

BCD Adder

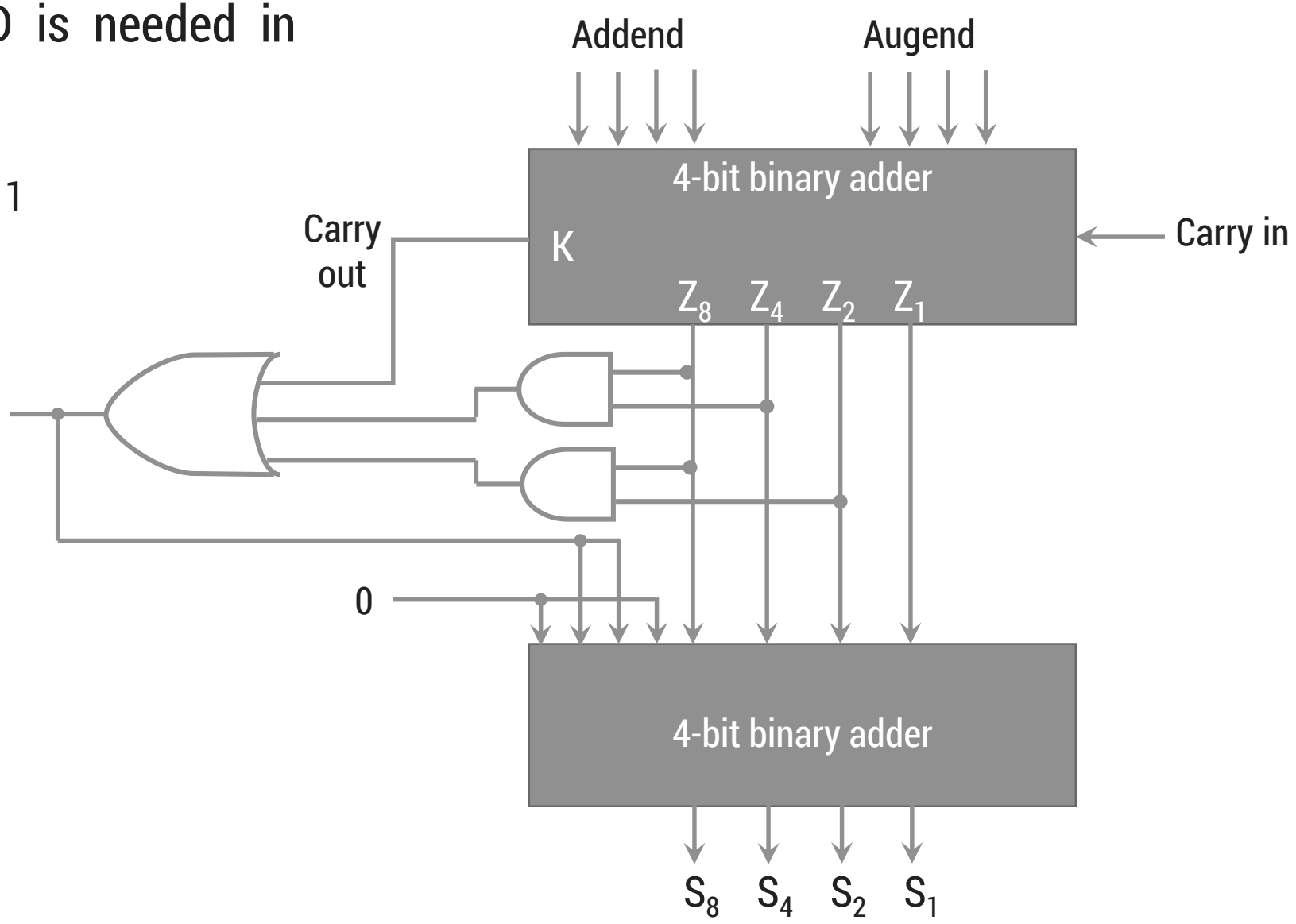
Binary Sum					BCD Sum					
K	Z_8	Z_4	Z_2	Z_1	C	S_8	S_4	S_2	S_1	Decimal
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	2
0	0	0	1	1	0	0	0	1	1	3
0	0	1	0	0	0	0	1	0	0	4
0	0	1	0	1	0	0	1	0	1	5
0	0	1	1	0	0	0	1	1	0	6
0	0	1	1	1	0	0	1	1	1	7
0	1	0	0	0	0	1	0	0	0	8
0	1	0	0	1	0	1	0	0	1	9

Binary Sum					BCD Sum					
K	Z_8	Z_4	Z_2	Z_1	C	S_8	S_4	S_2	S_1	Decimal
0	1	0	1	0	1	0	0	0	0	10
0	1	0	1	1	1	0	0	0	1	11
0	1	1	0	0	1	0	0	1	0	12
0	1	1	0	1	1	0	0	1	1	13
0	1	1	1	0	1	0	1	0	0	14
0	1	1	1	1	1	0	1	0	1	15
1	0	0	0	0	1	0	1	1	0	16
1	0	0	0	1	1	0	1	1	1	17
1	0	0	1	0	1	1	0	0	0	18
1	0	0	1	1	1	1	0	0	1	19

BCD Adder

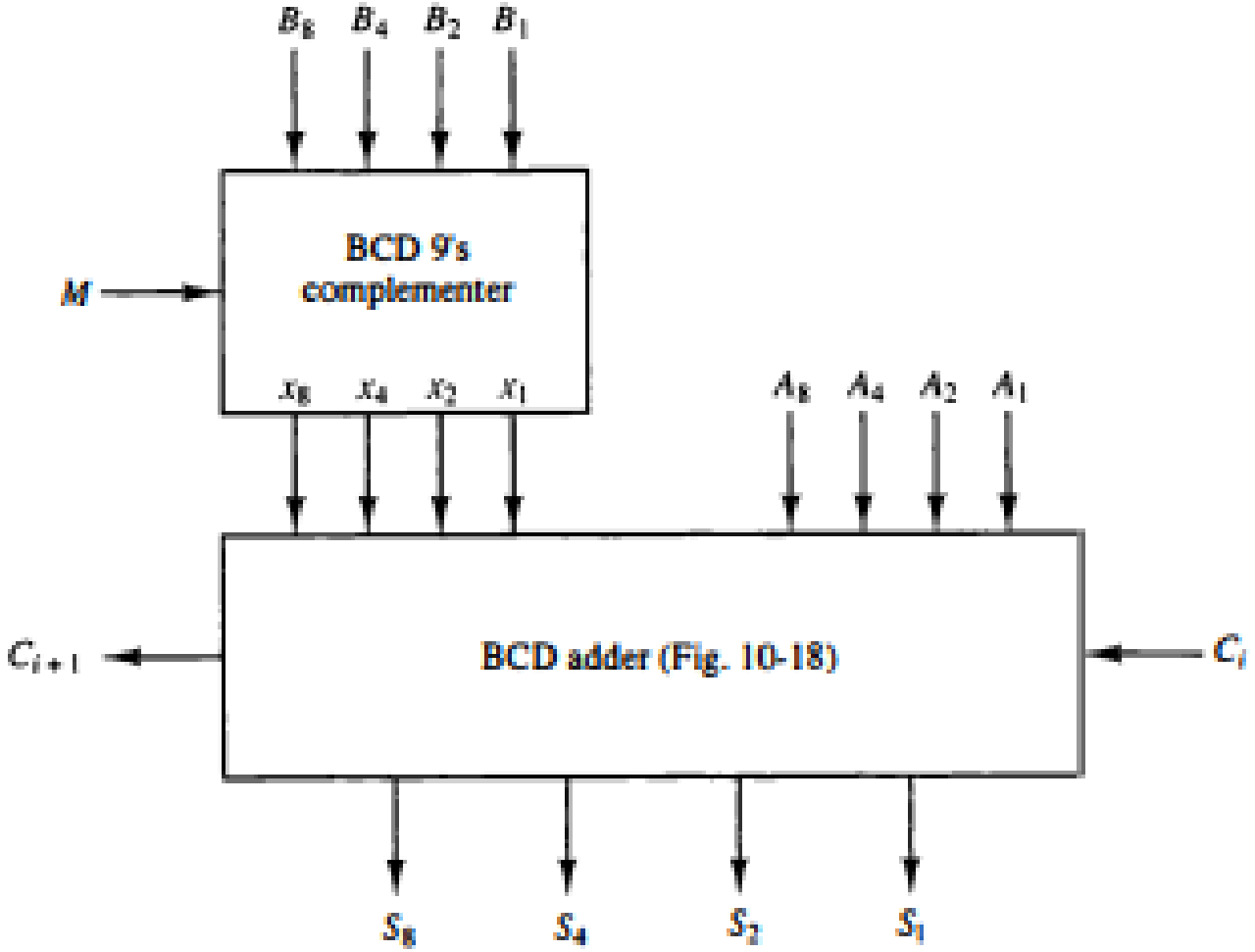
- Correction from binary to BCD is needed in following conditions
  - K = 1
  - Z_8 and Z_4 or Z_8 and Z_2 must have 1

$$C = K + Z_8Z_4 + Z_8Z_2$$

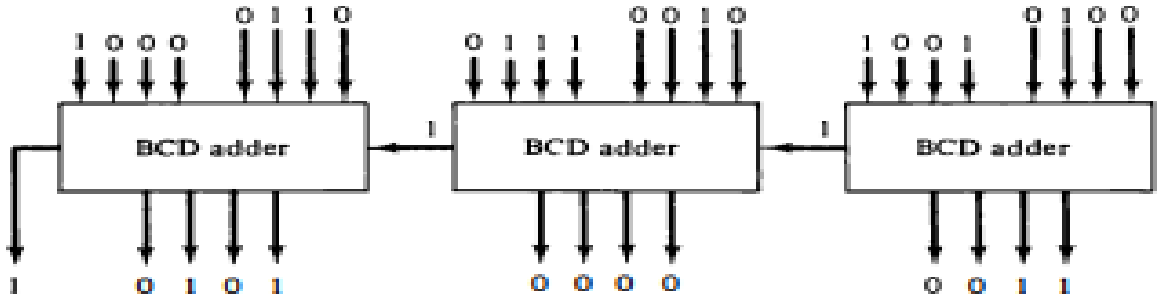


Decimal Arithmetic Circuit

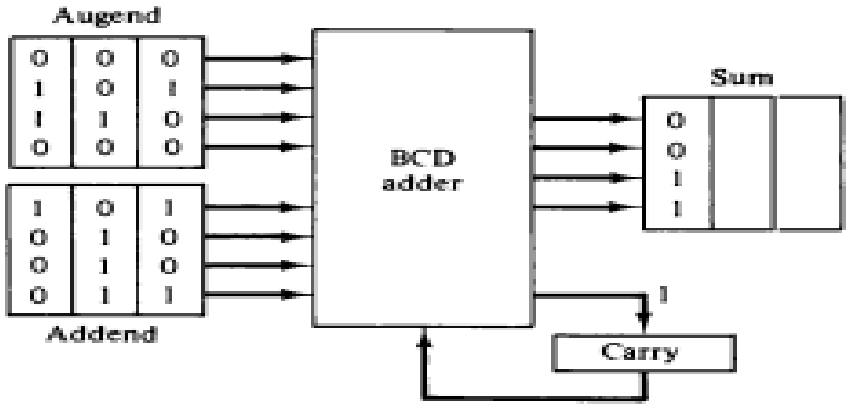
Figure 10-19 One stage of a decimal arithmetic unit.



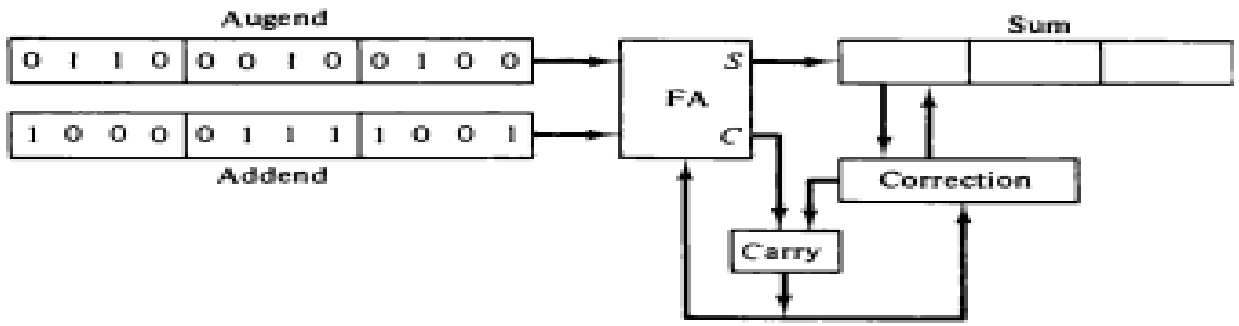
Three ways of decimal addition



(a) Parallel decimal addition: 624 + 879 = 1503



(b) Digit-serial, bit-parallel decimal addition



(c) All serial decimal addition

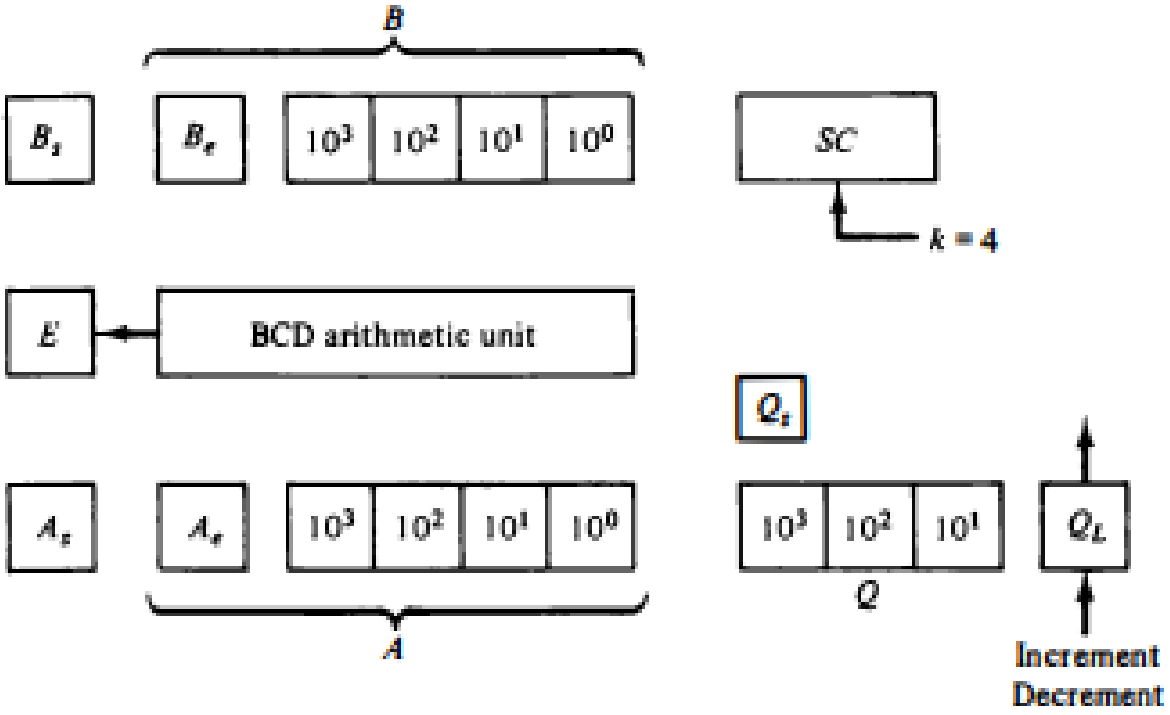


Figure 10-21 Registers for decimal arithmetic multiplication and division.

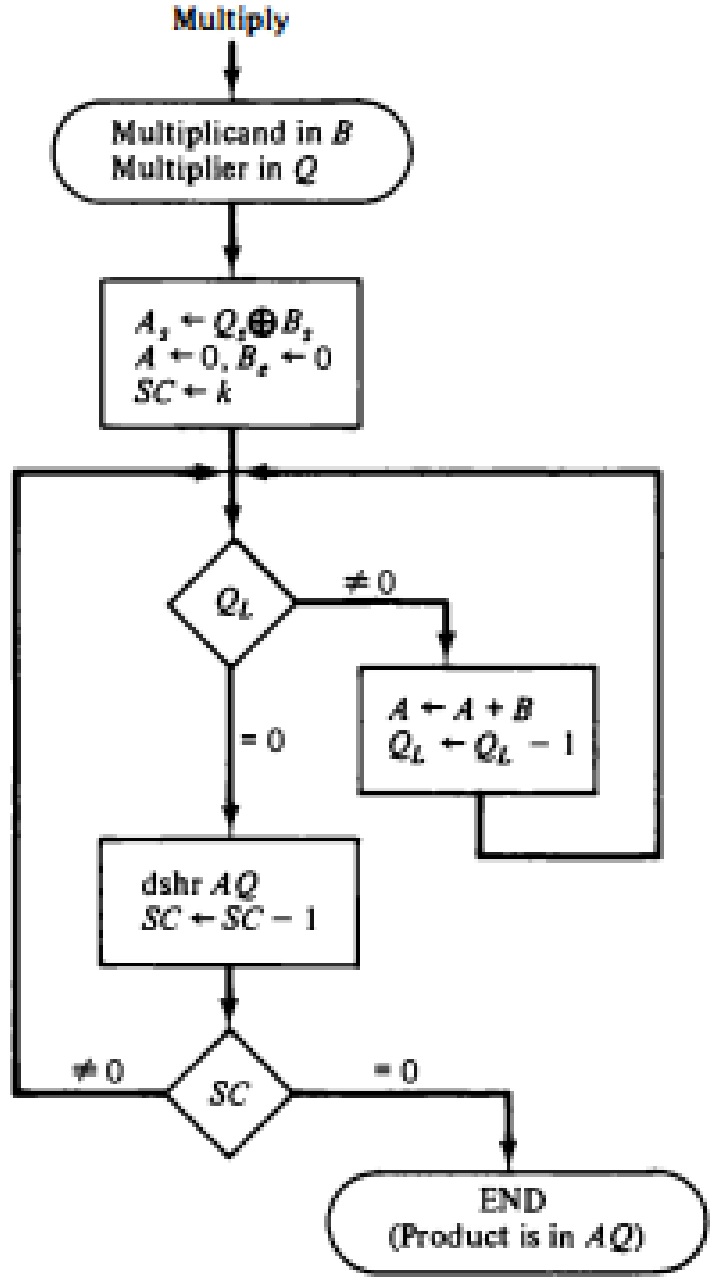


Figure 10-22 Flowchart for decimal multiplication.

Decimal Division

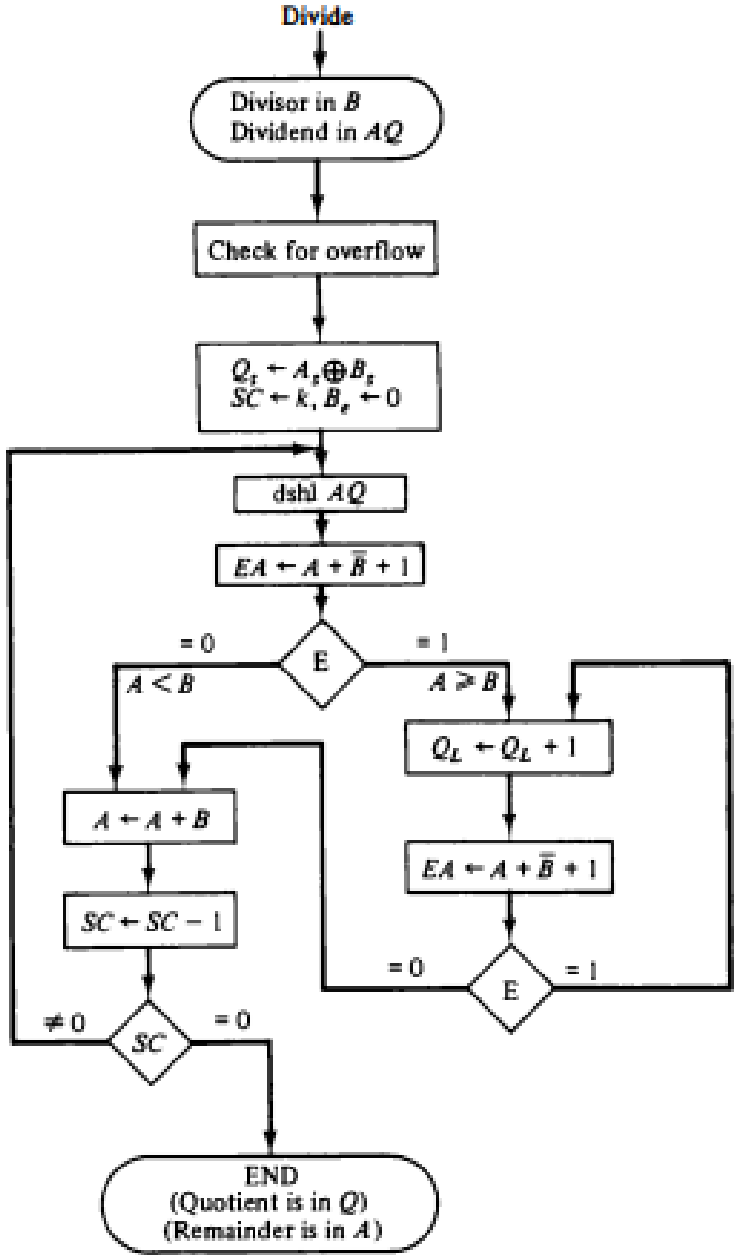


Figure 10-23 Flowchart for decimal division.

Questions asked in GTU exam

1. Explain Booth multiplication algorithm for multiplying binary integers in signed 2's complement representation.
2. Draw and explain flowchart for addition and subtraction operations with sign-magnitude data.
3. Explain BCD Adder with its block diagram.
4. Develop an algorithm for multiplication of two binary numbers, which are stored as per floating point representation.
5. Draw flowchart hardware multiplication algorithm and explain it.