# Chapter 10: Computer Arithmetic


| | |
|---|---|
| 10-2 | Addition and Subtraction |
| 10-3 | Multiplication Algorithms |
| 10-4 | Division Algorithms |
| 10-5 | Floating-Point Arithmetic Operations |
| 10-6 | Decimal Arithmetic Unit |
| 10-7 | Decimal Arithmetic Operations |

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:

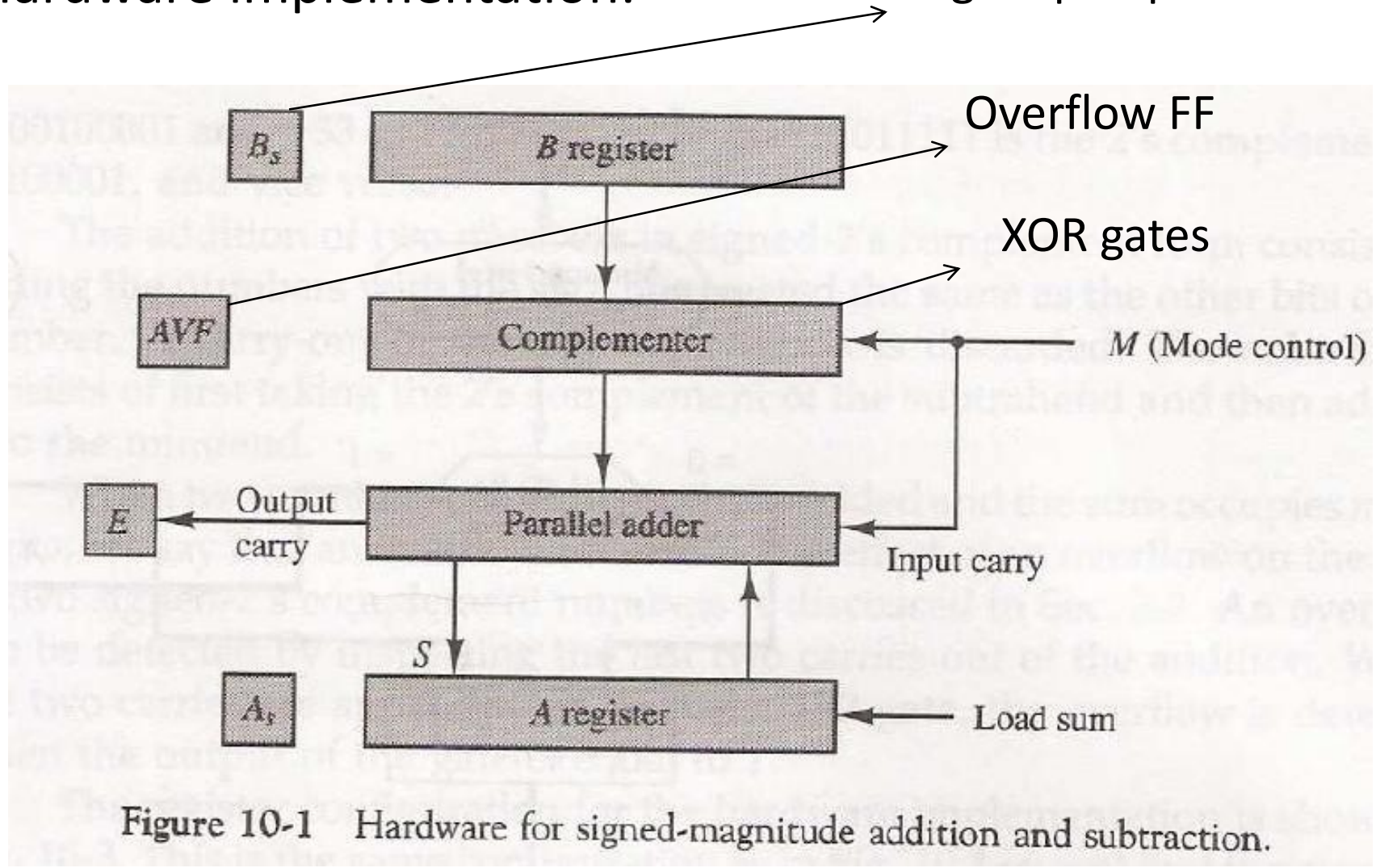
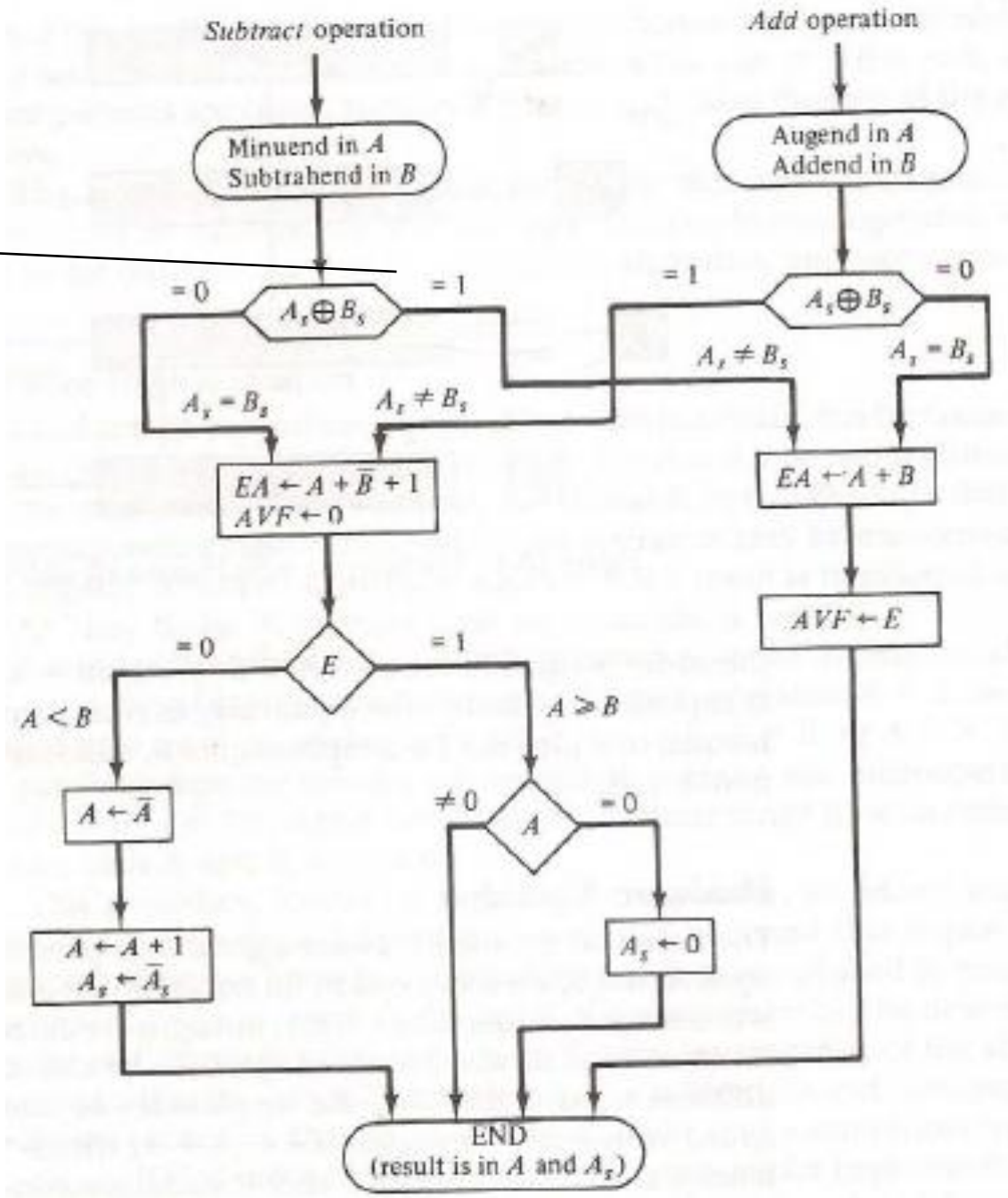



Figure 10-1 Hardware for signed-magnitude addition and subtraction.

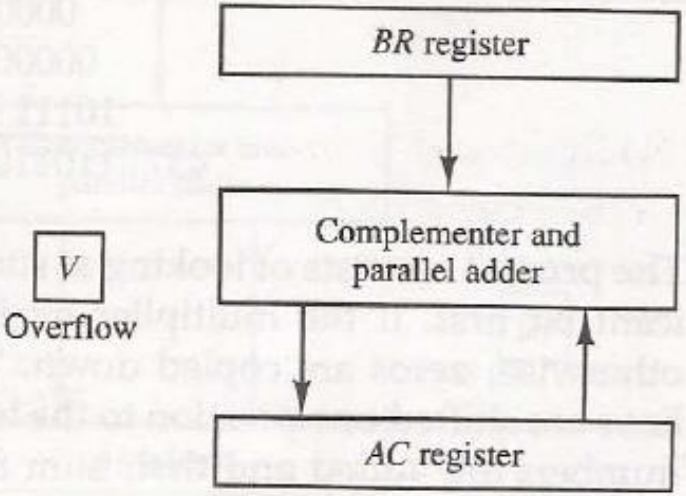
Algorithm:

Like as: A= -2 , B=5



# 2's complement addition and subtraction:

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



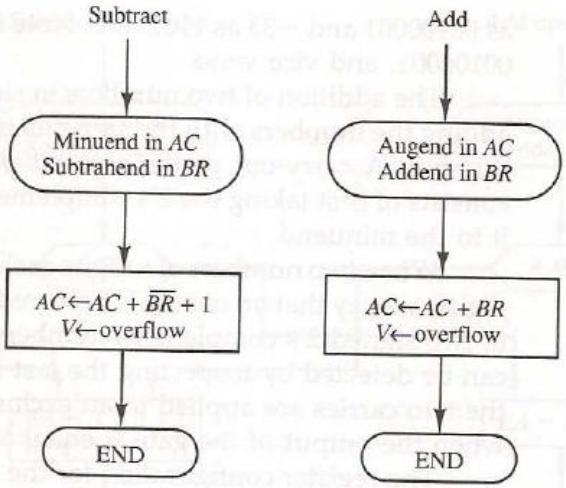
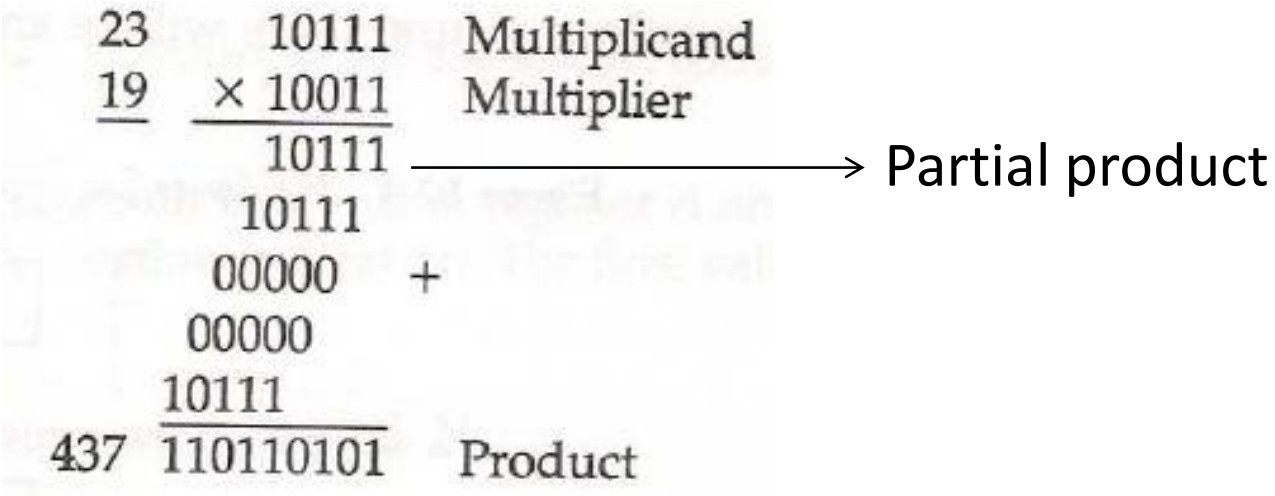




**Figure 10-4** Algorithm for adding and subtracting numbers in signed-2's complement representation.

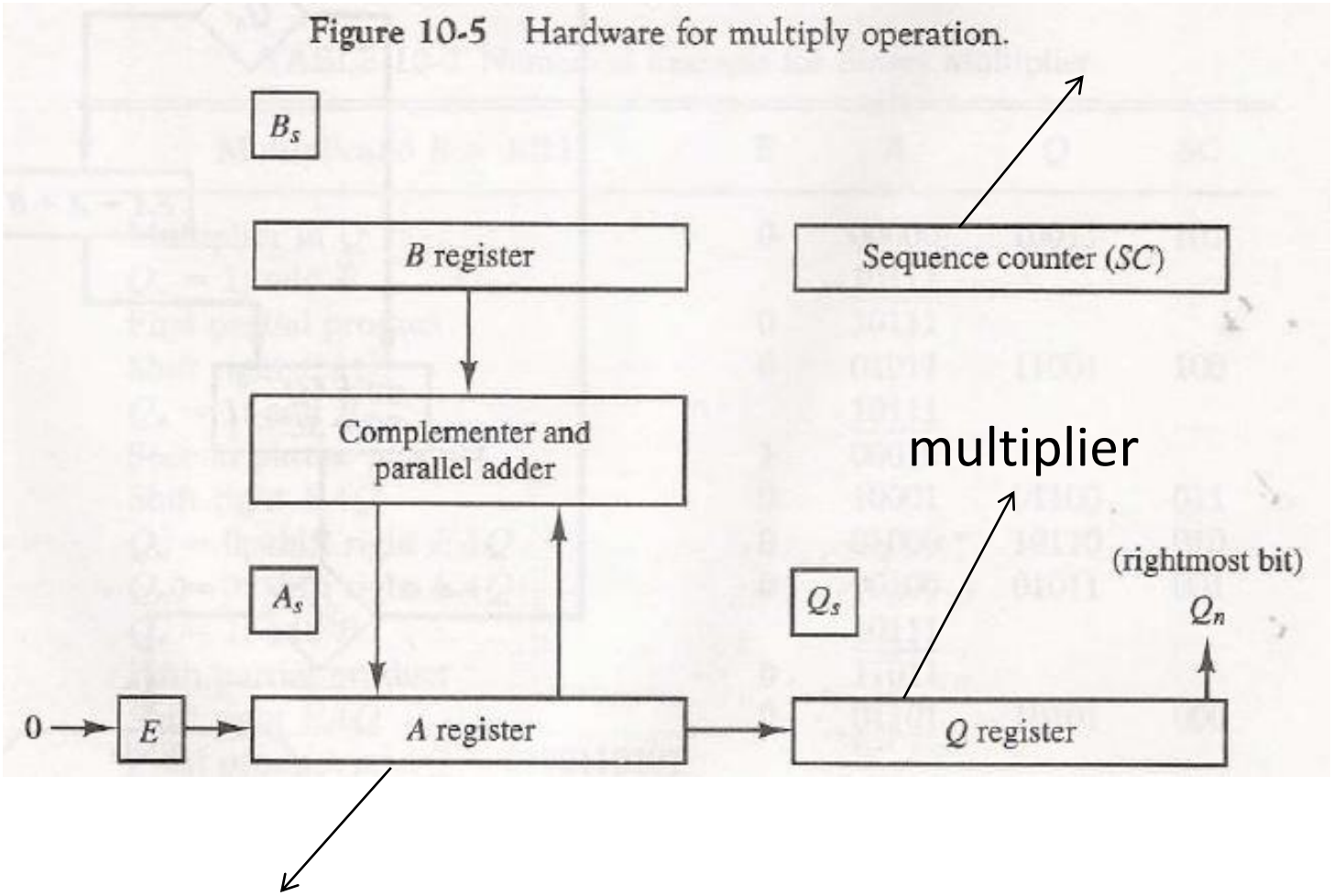
# Multiplication algorithms:

A binary example:



# Hardware implementation

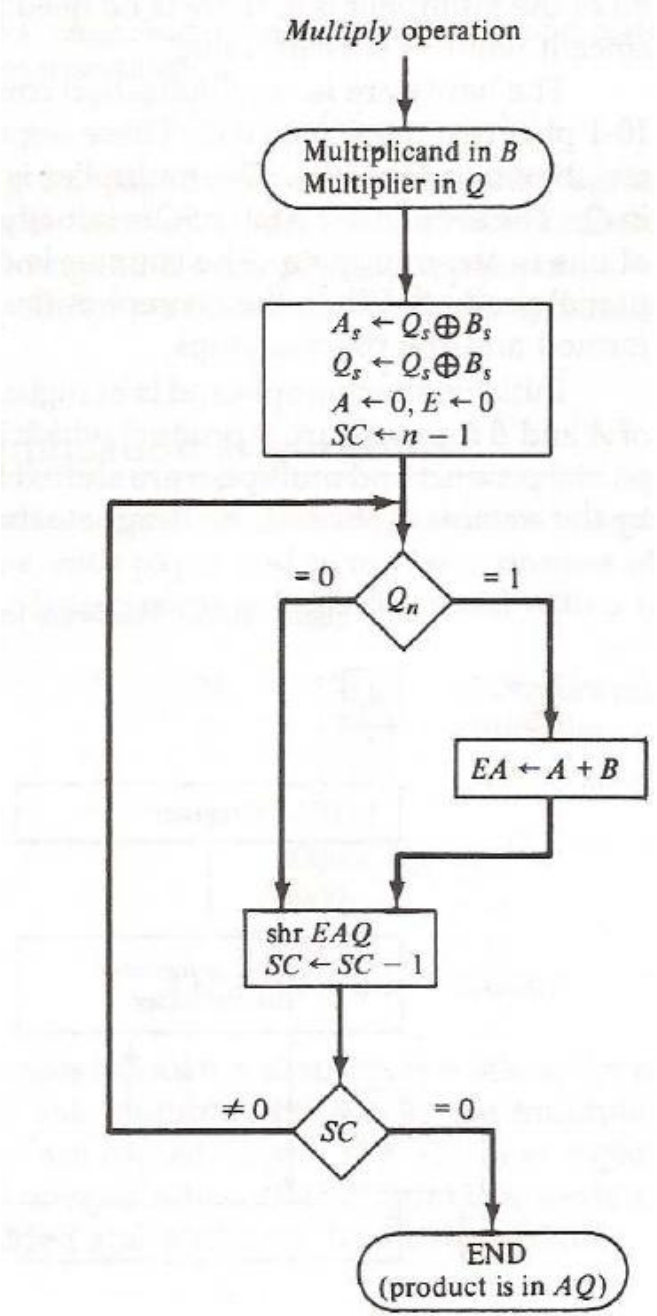
#of bit in multiplier




Partial product

Algorithm:

Figure 10-6 Flowchart for multiply operatic




| | B=11011 Q=00111 |
|---|---|
| 4 | Q4=1,A=0,Qs=1<br>EA=A+B=1011<br>EAQ= 0 1011 0111<br>Shr EAQ= 0 0101 1011 |
| 3 | Q3=1<br>EA = 1 0000<br>EAQ 1 0000 1011<br>Shr EAQ 0 1000 0101 |
| 2 | Q2=1<br>EA= 1 0011<br>EAQ 1 0011 01 01<br>shr EAQ 0 1001 101 0 |
| 1 | Q1=0<br>Shr EAQ 0 0100 1101=77 |
| 0 | |

# Booth multiplication algorithm

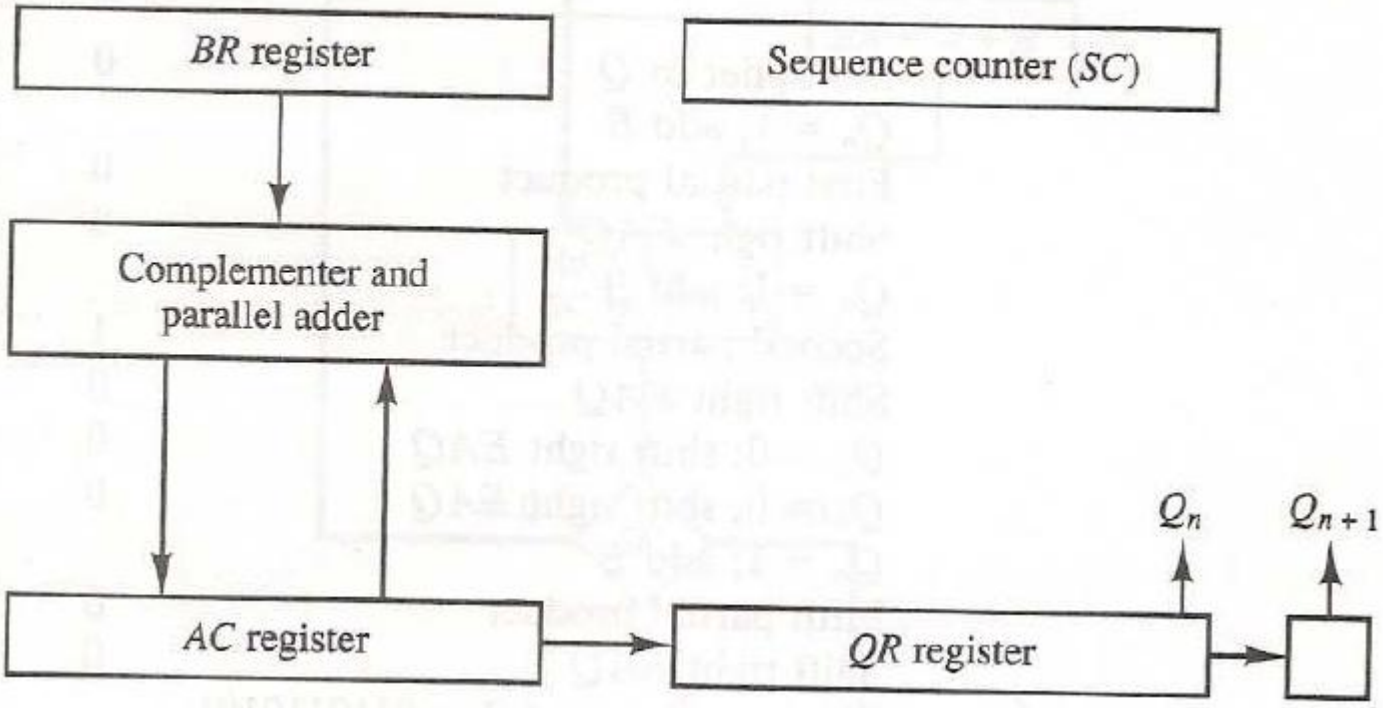
A= 00011 B= 00111 => A*B= A*(7)=A* (8-1)=A*8-A*1

1. The multiplicand is subtracted from the partial product upon encountering the first least significant 1 in a string of 1's in the multiplier.
2. The multiplicand is added to the partial product upon encountering the first 0 (provided that there was a previous 1) in a string of 0's in the multiplier.
3. The partial product does not change when the multiplier bit is identical to the previous multiplier bit.

In 2's compl. representation, we can use Booth alg. without change.

Hardware

Figure 10-7 Hardware for Booth algorithm.



# Algorithm

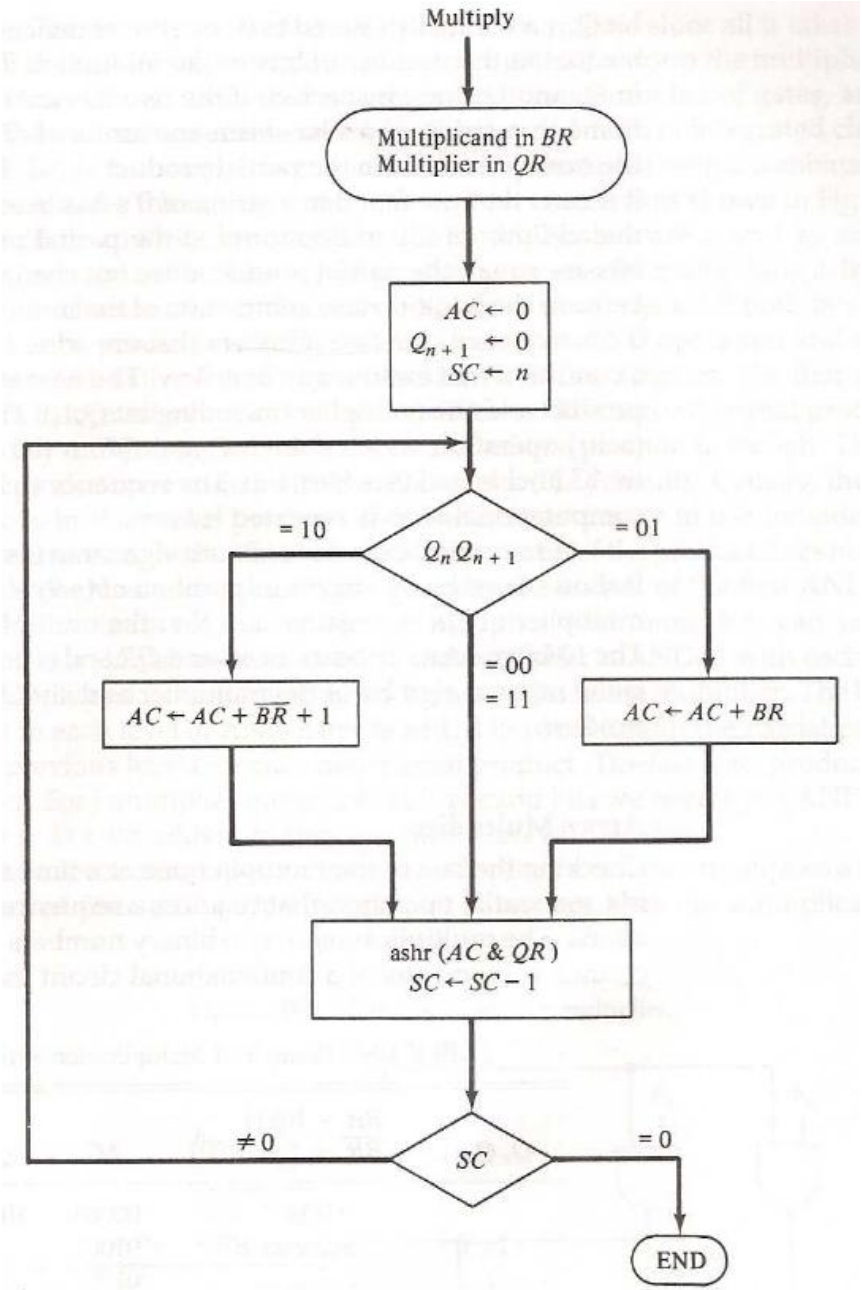



**Figure 10-8** Booth algorithm for multiplication of signed-2's complement numbers.

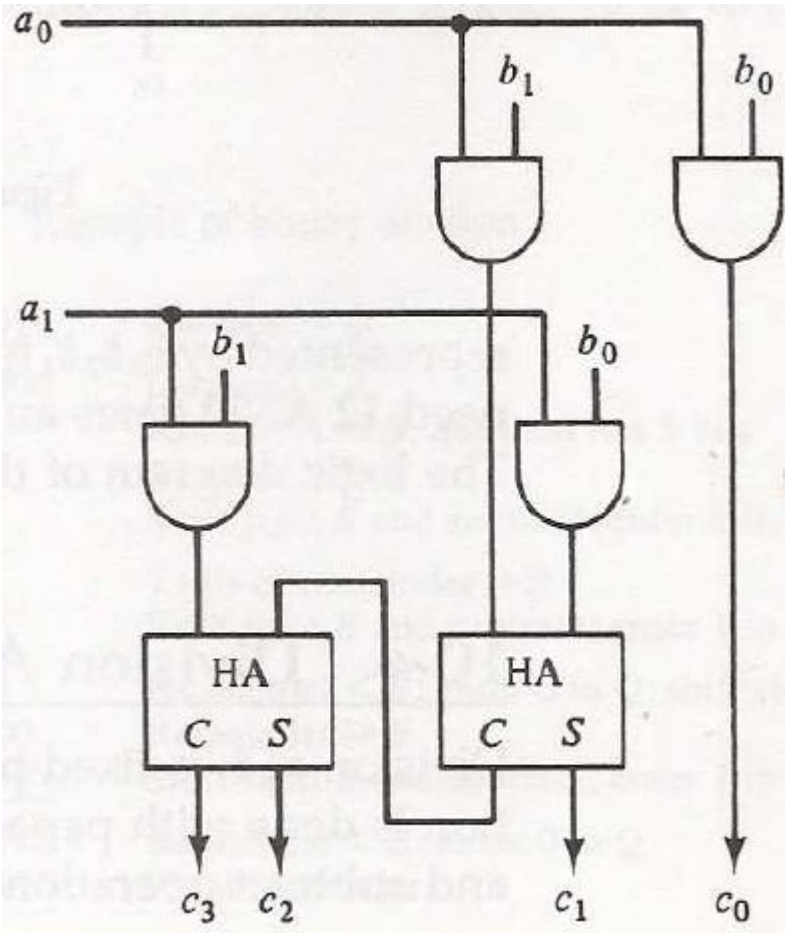
An example:

TABLE 10-3 Example of Multiplication with Booth Algorithm

| $Q_n Q_{n+1}$ | $BR = 10111$<br>$\overline{BR} + 1 = 01001$ | $AC$ | $QR$ | $Q_{n+1}$ | $SC$ |
|---|---|---|---|---|---|
| | 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

| | | | |
|---|---|---|---|
| | | $b_1$ | $b_0$ |
| | | $a_1$ | $a_0$ |
| | | $a_0b_1$ | $a_0b_0$ |
| | $a_1b_1$ | $a_1b_0$ | |
| $c_3$ | $c_2$ | $c_1$ | $c_0$ |



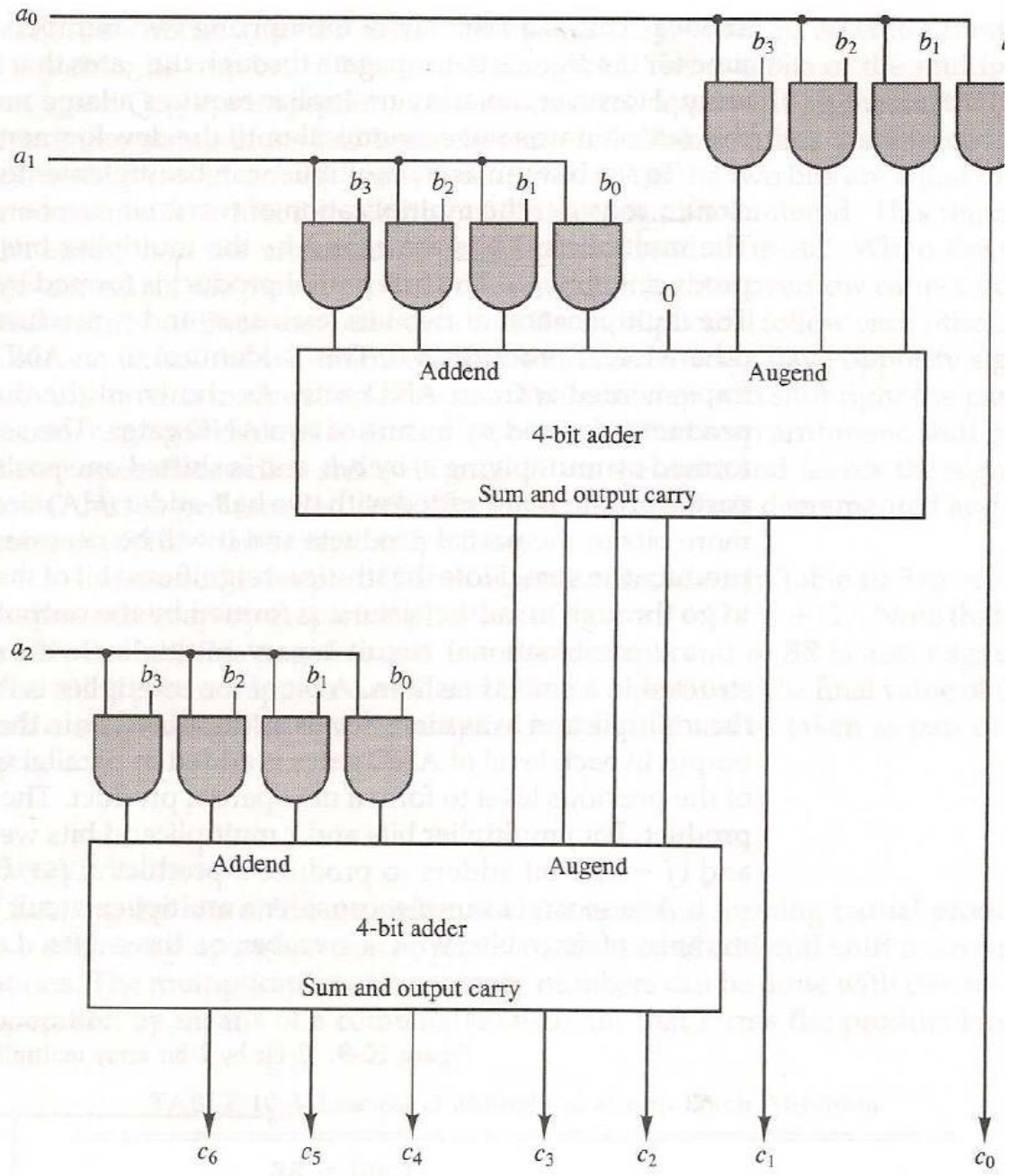
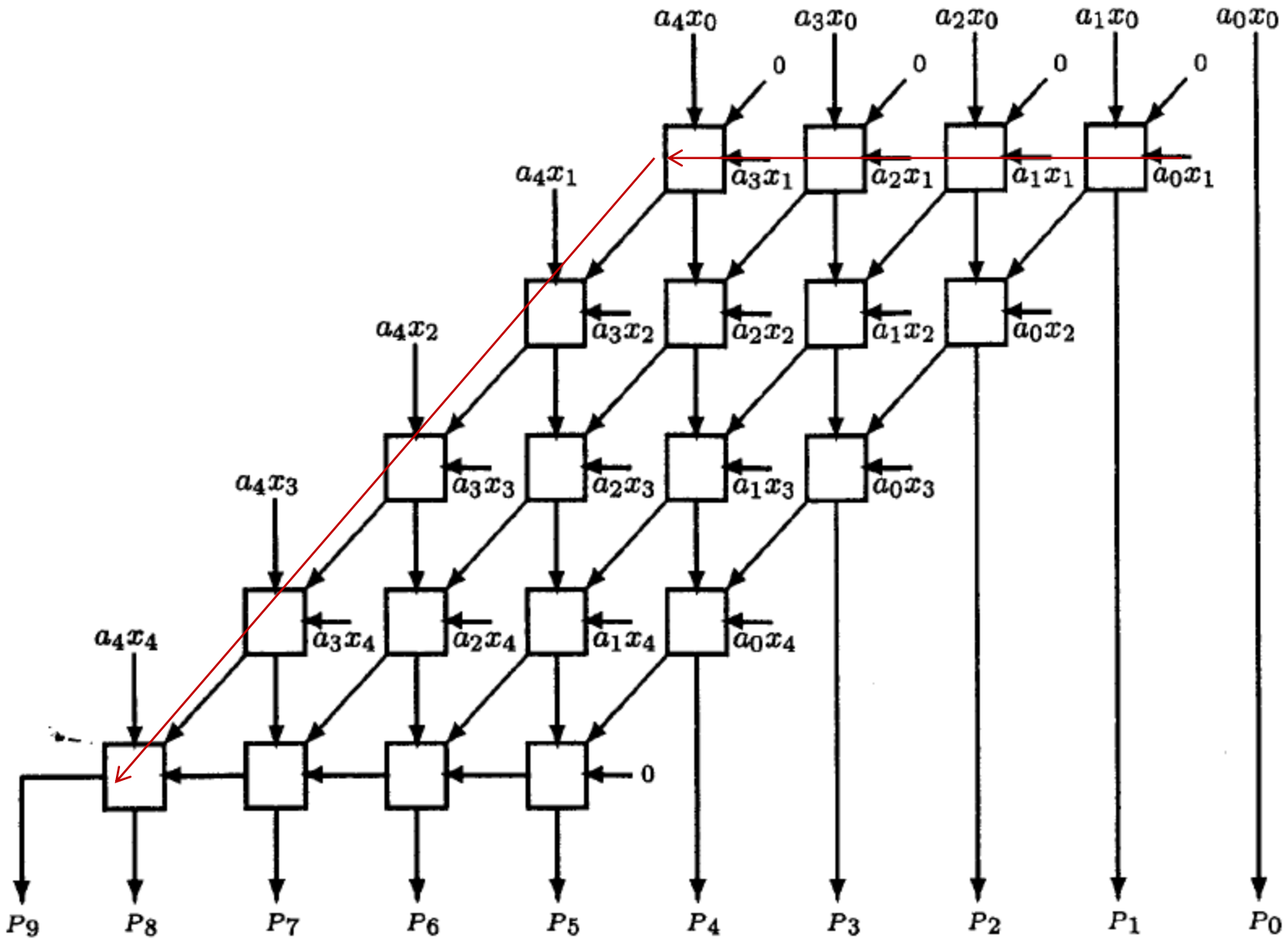



**Figure 10-10** 4-bit by 3-bit array multiplier.


$a_4x_0$
$a_3x_0$
$a_2x_0$
$a_1x_0$
$a_0x_0$
0
$a_4x_1$
$a_3x_1$
$a_2x_1$
$a_1x_1$
$a_0x_1$
$a_4x_2$
$a_3x_2$
$a_2x_2$
$a_1x_2$
$a_0x_2$
$a_4x_3$
$a_3x_3$
$a_2x_3$
$a_1x_3$
$a_0x_3$
$a_4x_4$
$a_3x_4$
$a_2x_4$
$a_1x_4$
$a_0x_4$
$P_9$
$P_8$
$P_7$
$P_6$
$P_5$
$P_4$
$P_3$
$P_2$
$P_1$
$P_0$

# Division algorithms:

A/B = q , r ; A: Dividend , B: Divisor, q: quotient, r: remainder

Restoring method:

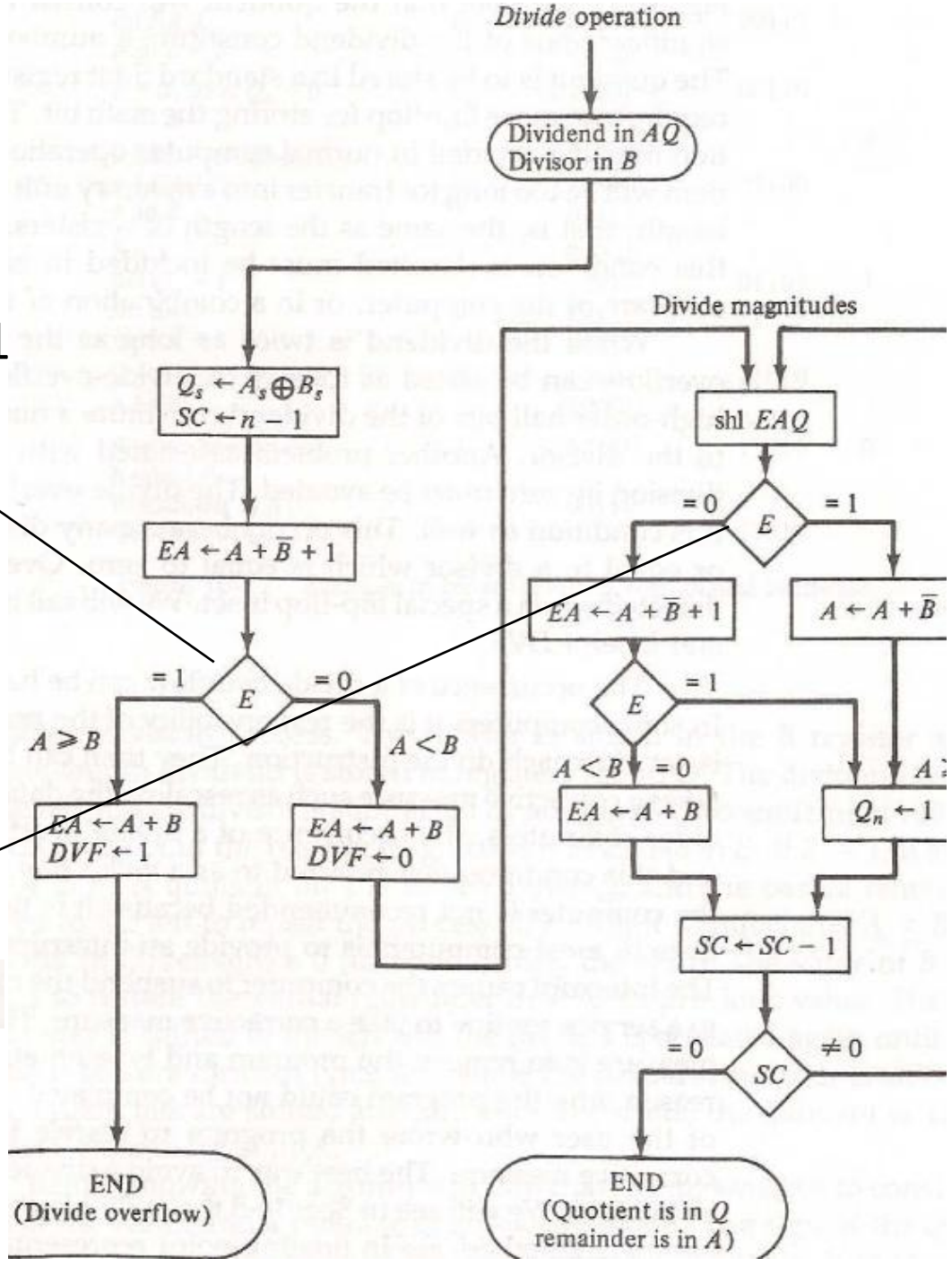
Divisor $B = 10001$, $\overline{B} + 1 = 01111$

| | $E$ | $A$ | $Q$ | $SC$ |
|---|---|---|---|---|
| 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.

# Algorithm:

If A-B>= 0 then DVF←1




$$(EA - 2^{n-1}) + (2^{n-1} - B) = EA - B$$

# Other methods:

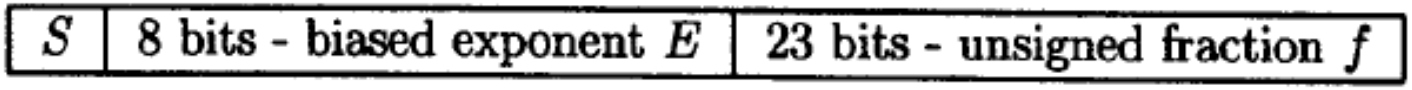
- Comparison
- Non-restoring: in the restoring method when A<B → 2(A-B+B)-B=2A-B,

  A<B → 2(A-B)+B=2A-B

# Floating point operations

- The standard format:

➢ IEEE 754 single precision

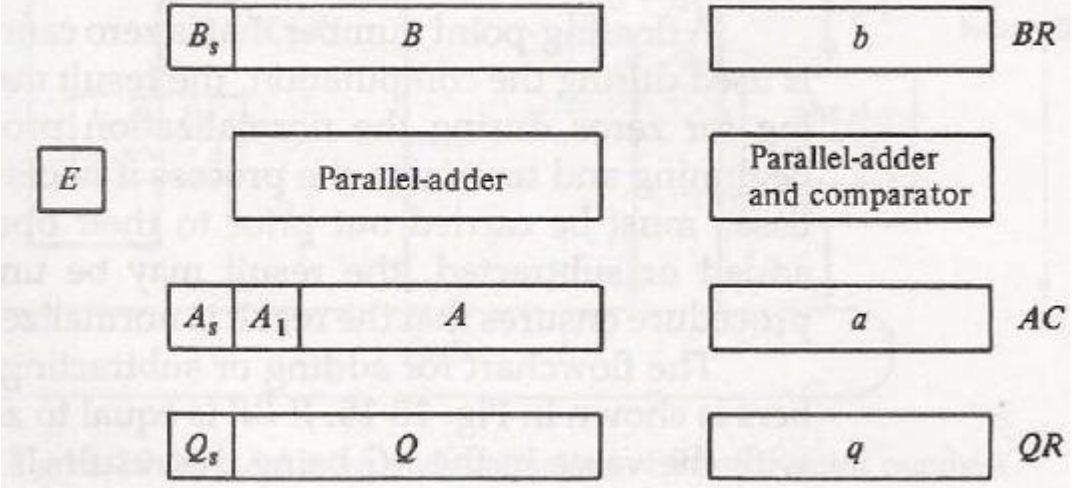



$$F = (-1)^S \, 1.f \, 2^{E-127}$$

- F=0 : E=0 & $f$=0
- F=denormalized : E=0 & $f \neq 0$
- F= $\pm\infty$ : E=255 & f=0
- F=NAN: E=255 & f ≠0

Hardware:

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



# Addition and subtraction

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

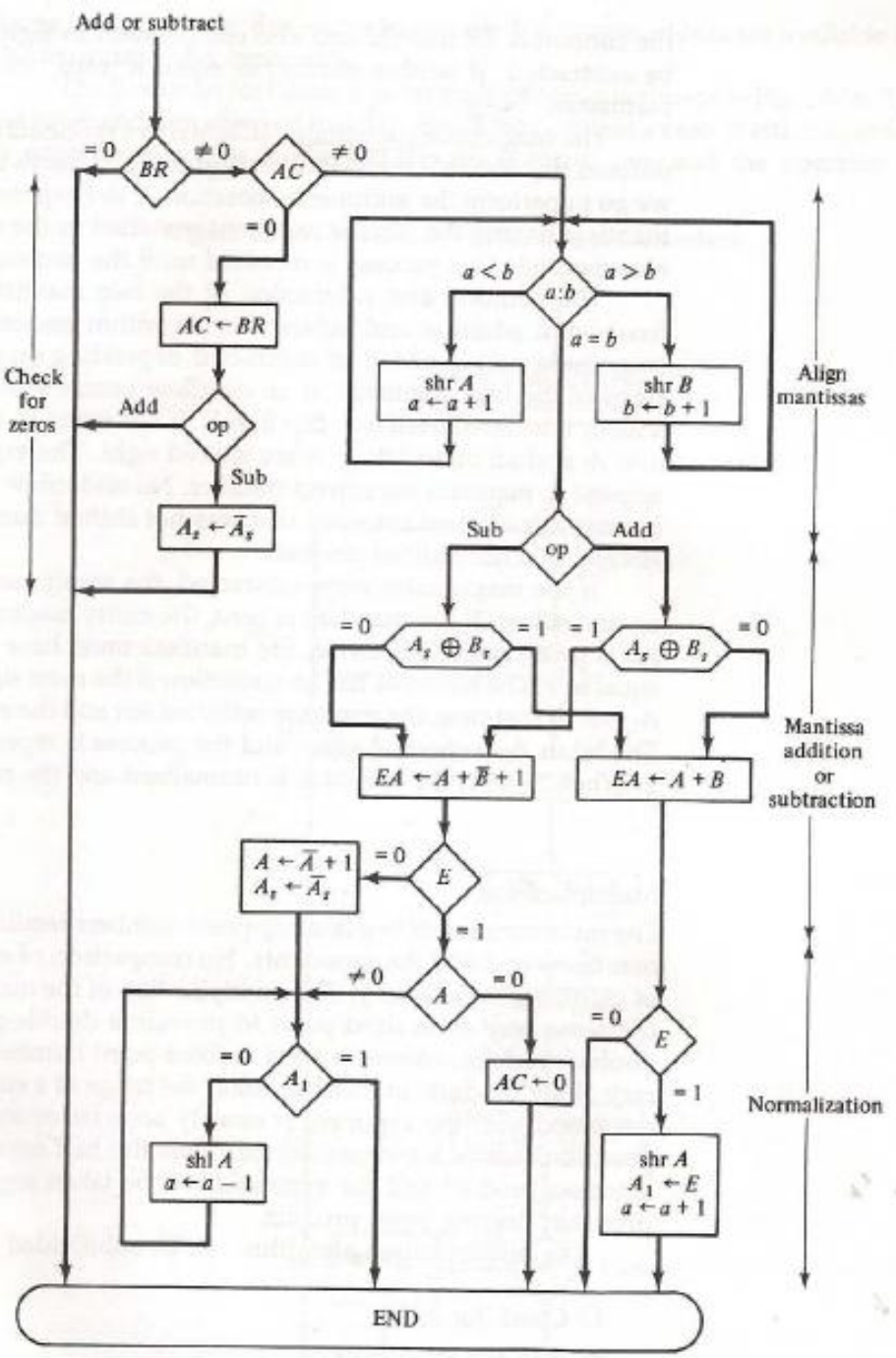
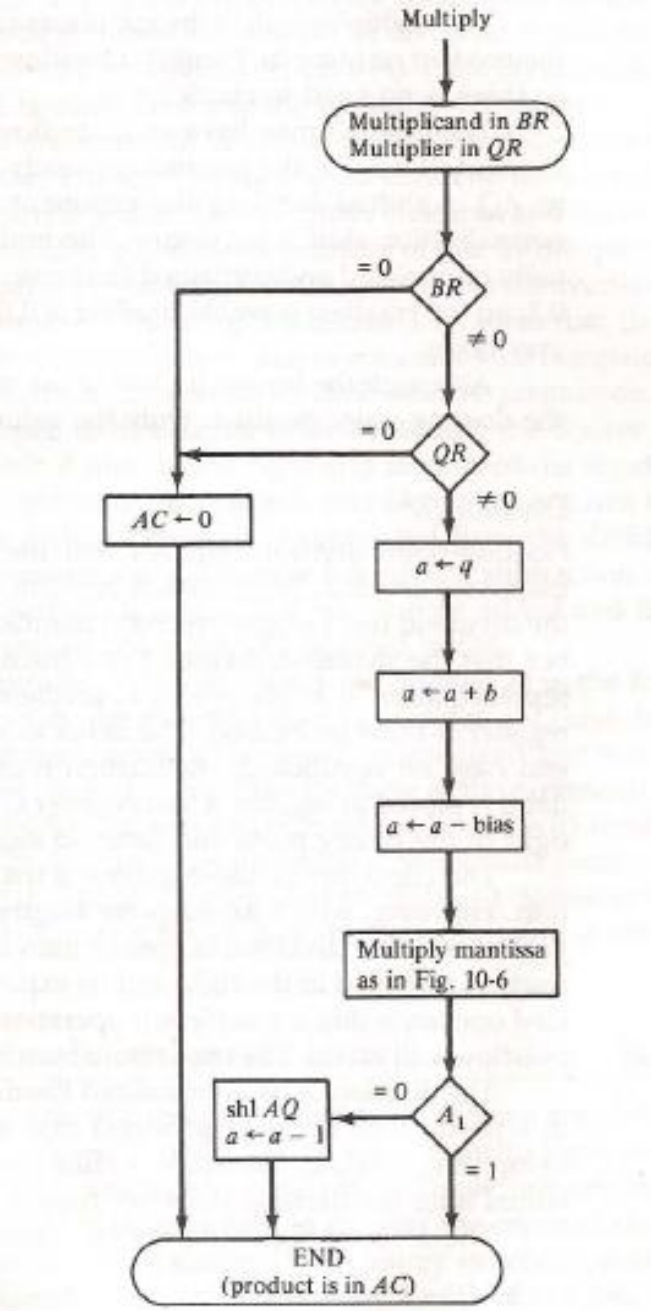



Figure 10-15 Addition and subtraction of floating-point numbers.

# Multiplication:

Figure 10-16 Multiplication of floating-point numbers.



# Division:

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

# Decimal Arithmetic Unit

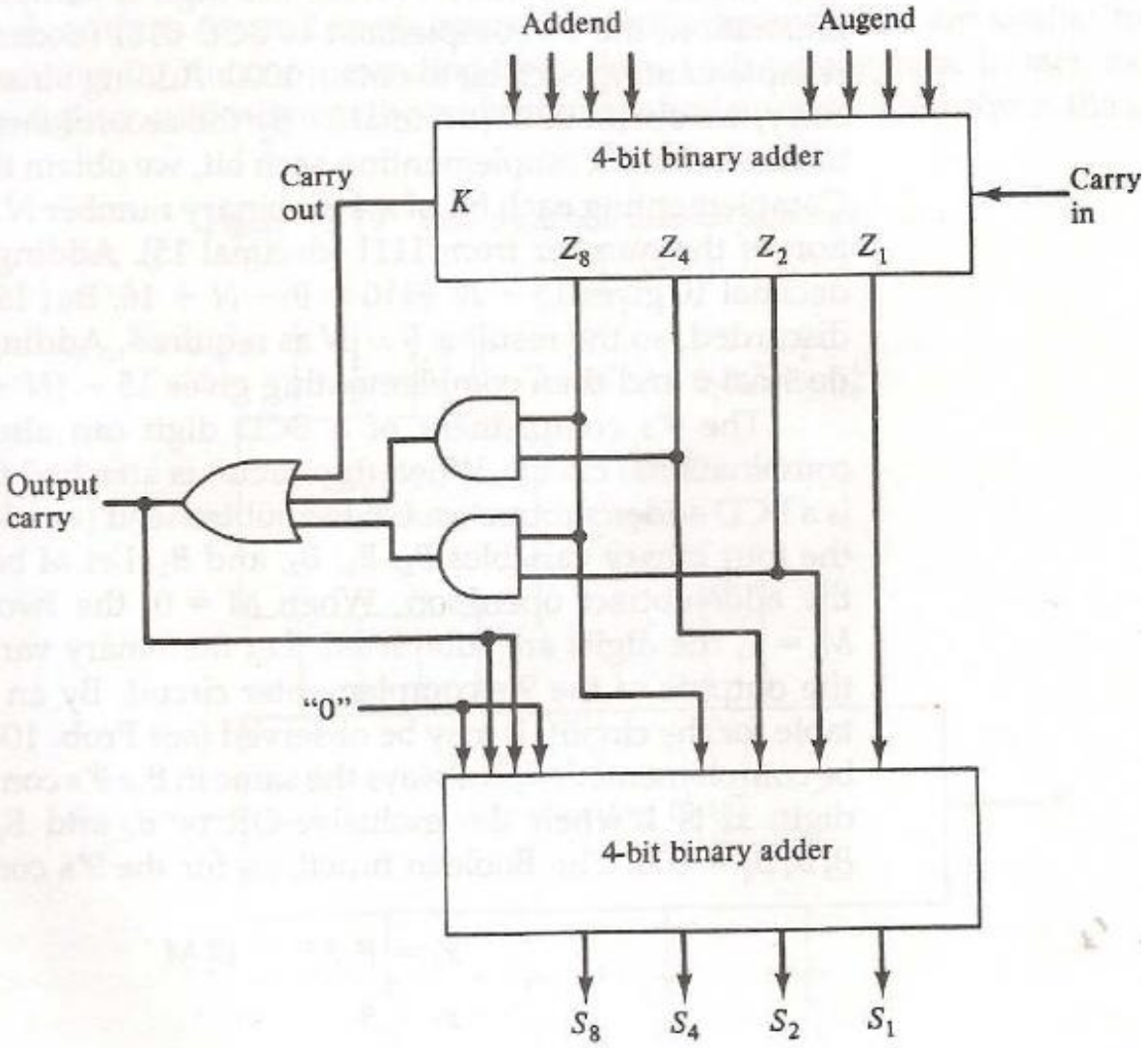
BCD Adder

TABLE 10-4 Derivation of 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 |
| 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 |

Hardware:

Figure 10-18 Block diagram of BCD adder.



# BCD subtractor:

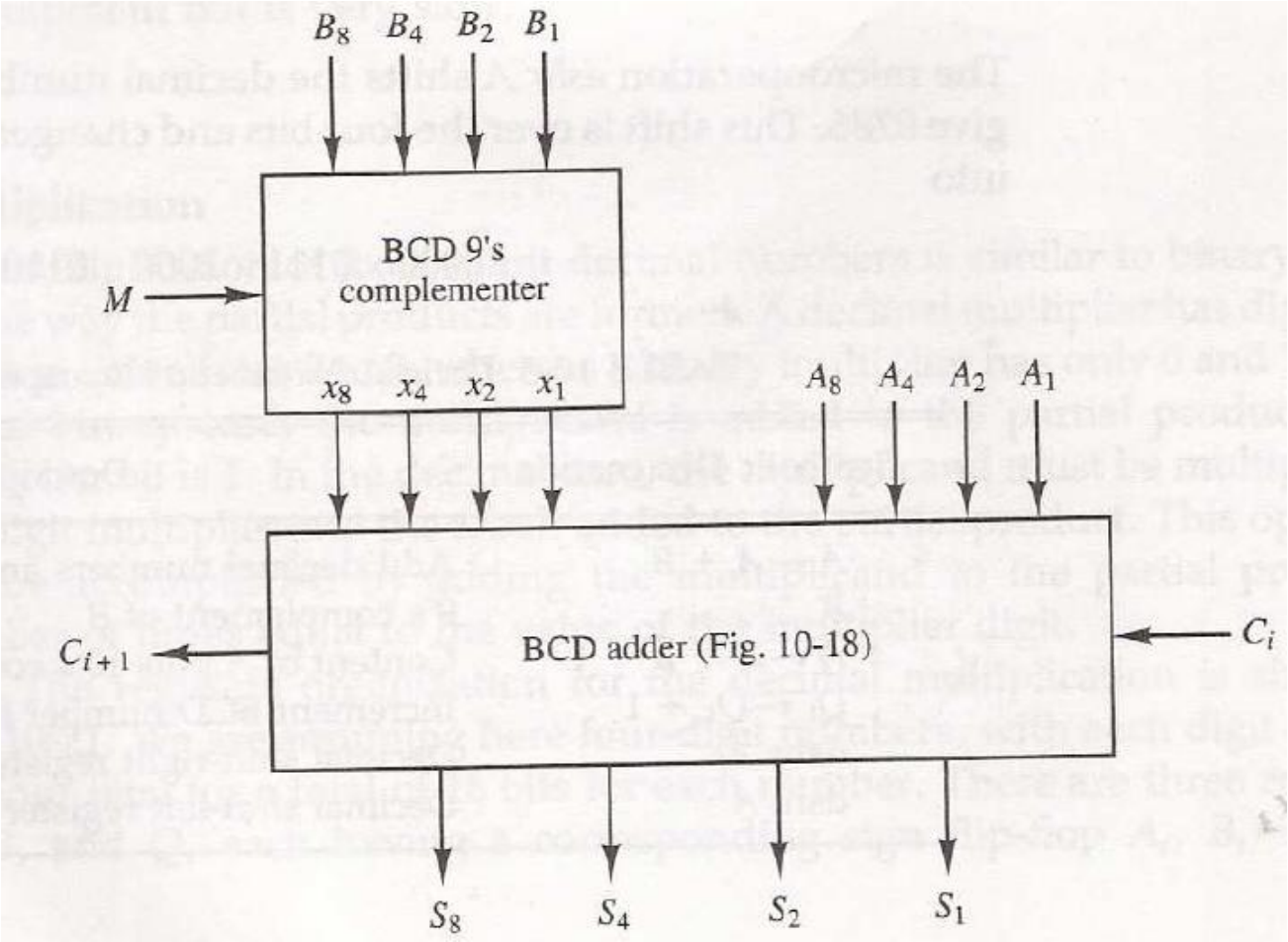
A-B = A+ 9’s compl. of B

9’ compl.(B) =
- B’+ 1010= 15-B+1010=9-B+16
- (B+ 0110)’ = 15 – B- 0110 =9-B
- logic circuit

An example: -(0111) =

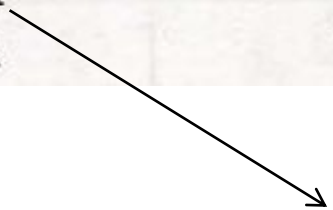
1000+1010=(0111+0110)’=0010

# Decimal arithmetic operation



# Decimal arithmetic operation

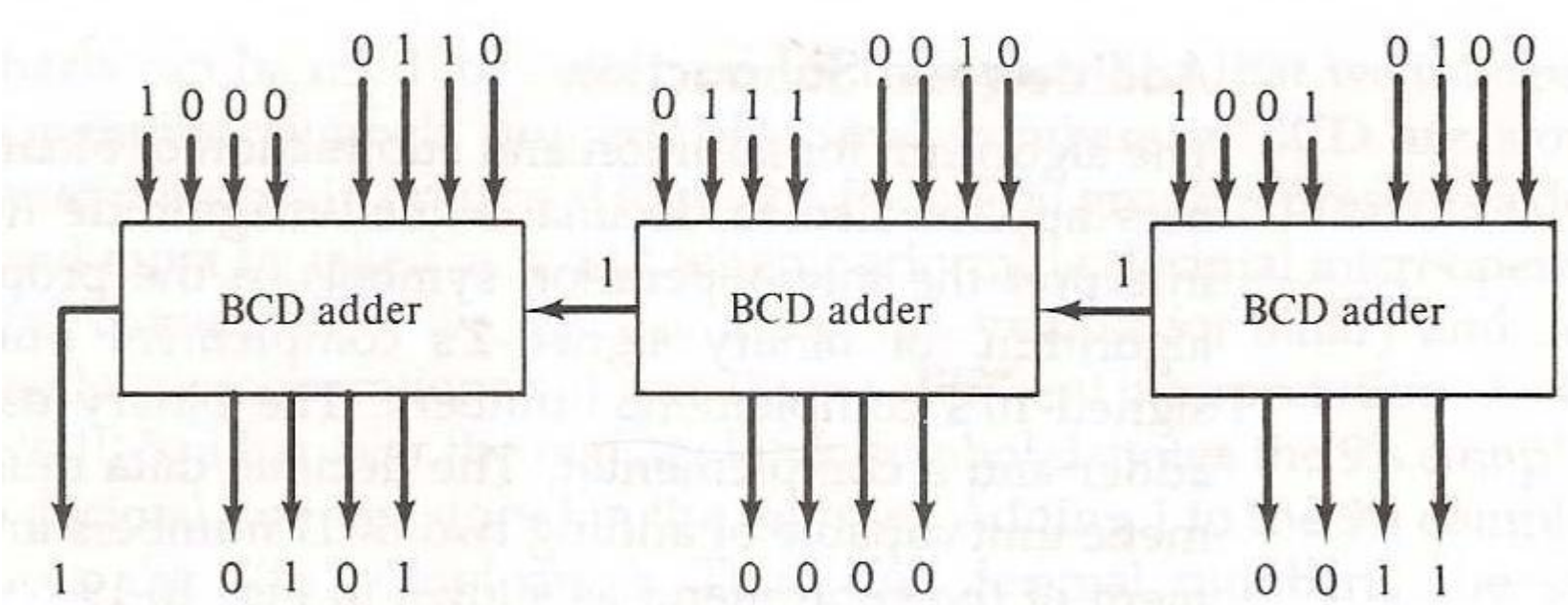
| | |
|---|---|
| $A \leftarrow A + B$ | Add decimal numbers and transfer sum into $A$ |
| $\overline{B}$ | 9's complement of $B$ |
| $A \leftarrow A + \overline{B} + 1$ | Content of $A$ plus 10's complement of $B$ into $A$ |
| $Q_L \leftarrow Q_L + 1$ | Increment BCD number in $Q_L$ |
| dshr $A$ | Decimal shift-right register $A$ |
| dshl $A$ | Decimal shift-left register $A$ |



dshr (7680) → 0768

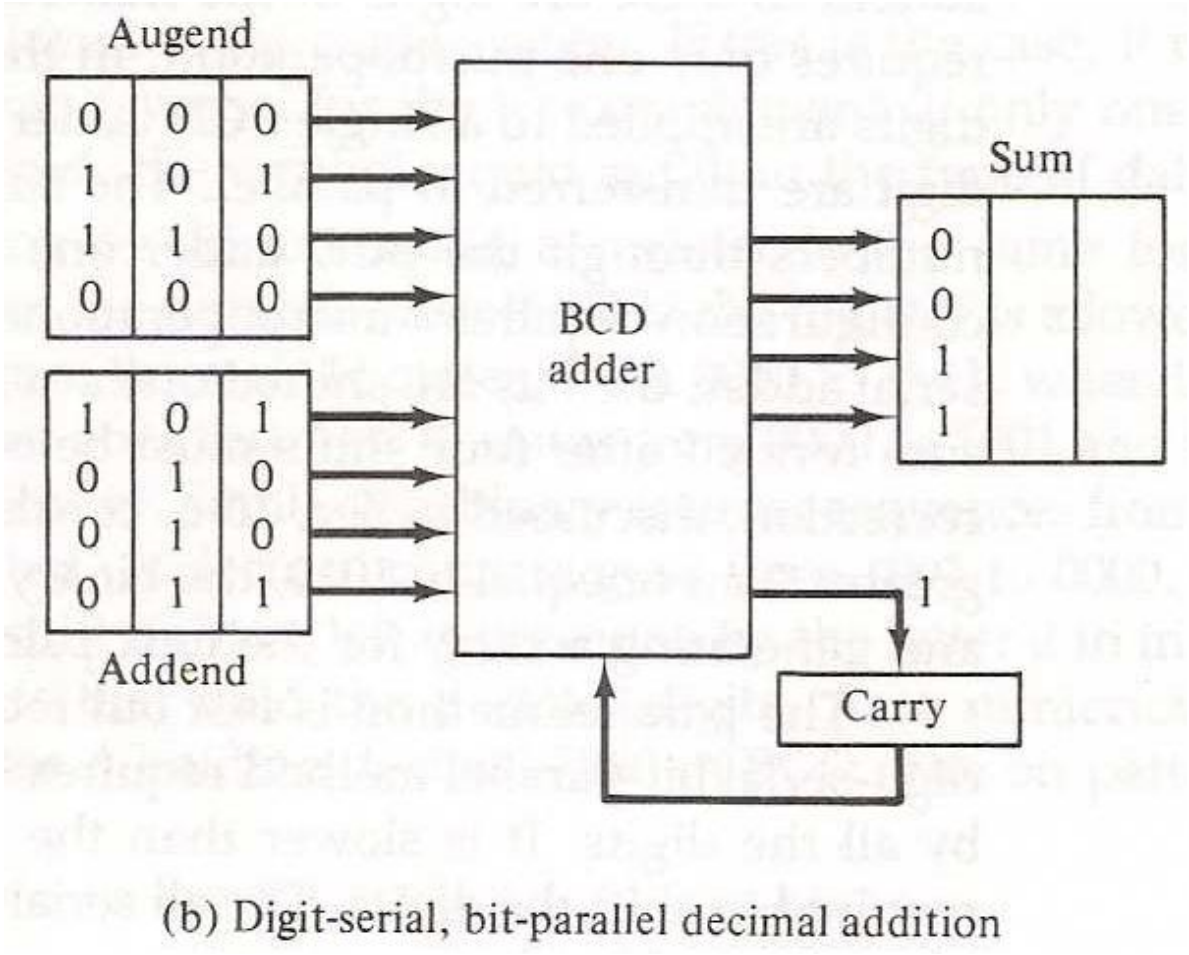
# Addition and Subtraction

Parallel adder:



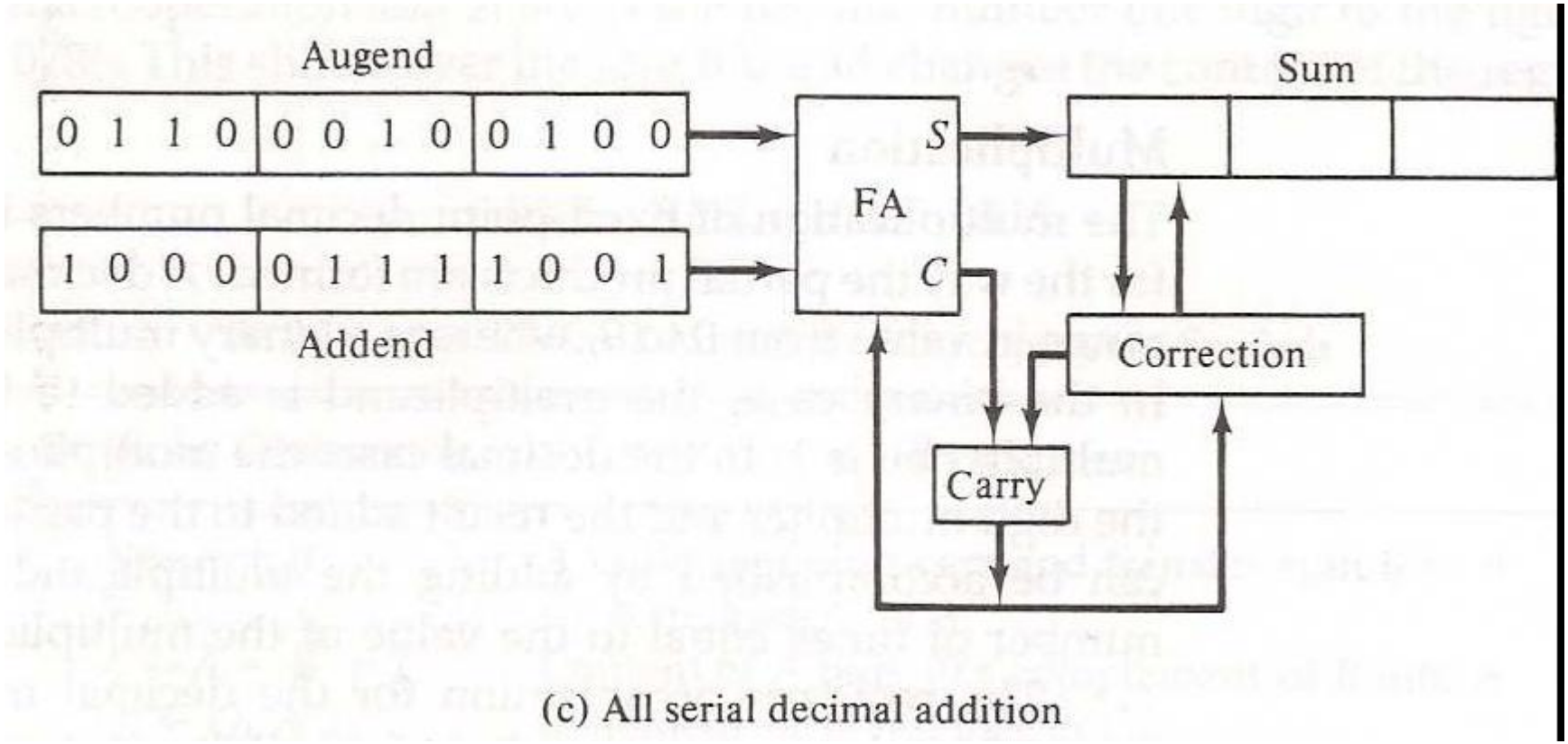

(a) Parallel decimal addition: 624 + 879 = 1503

# Digit serial bit parallel:




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

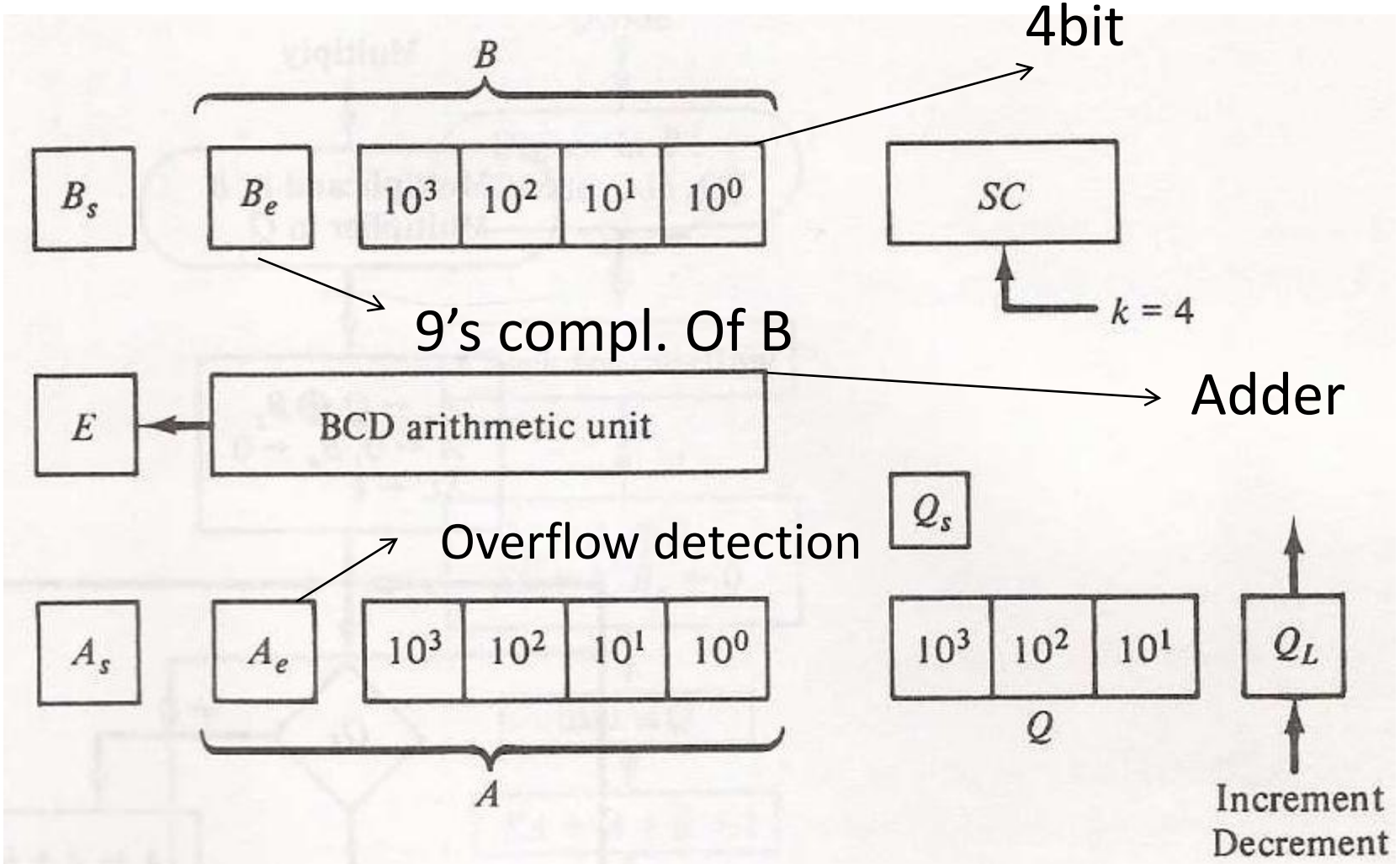
# All serial:




(c) All serial decimal addition

# Multiplication:

$A_i * B_i = [0,81]$!



# Algorithm:

K: # 0f digits