Computer Arithmetic

Contents

- Introduction
- Addition and Subtraction
- Multiplication Algorithms
- Division Algorithms

Introduction

- Arithmetic instructions in digital computers manipulate data to produce results necessary for the solutions of computational problems. These instructions perform arithmetic calculations and are responsible for the bulk of activity involved in processing data in a computer.
- The four basic arithmetic operations are addition,subtraction,multiplication and division.
- From these four basic operations , it is possible to formulate other arithmetic functions and solve problems by means of numerical analysis methods.
- An arithmetic processor is the part of a processor unit that executes arithmetic operations.
- An arithmetic instruction may specify binary or decimal data, and in each case the data may be in fixed-point or floating point form.
- Negative numbers may be in signed magnitude or signed compliment representation.
- Fixed point numbers may represents integers or fractions.

Addition and Subtraction

- The addition and subtraction algorithm for data represented in signed magnitude and again data represented in signed 2’s complement.
- It is important to realize that the adopted representation for negative numbers refers to the representation of numbers in the register before and after the execution of the arithmetic operations.

Addition and Subtraction with Signed-magnitude Data:

- The representation of numbers in signed-magnitude is familiar because it is used in everyday arithmetic calculations. The procedure for adding or subtracting two signed binary numbers with paper and pencils simple and straight-forward. A review of this procedure will be helpful for deriving the hardware algorithm.
- When two numbers A and B are signed numbers are added or subtracted, we find that there are eight different conditions to consider, depending on the sign of the numbers and the operation performed.

- These conditions are listed in the first column of the table below. The other column in the table show the actual operation to be performed with the magnitude of the numbers. The last column is needed to prevent negative zero.
- In other words ,when two equal numbers are subtracted, the result should be +0 not -0.
- The algorithms for addition and subtraction are derived from the table and can be stated as follows (the words inside parentheses should be used for the subtraction algorithm) Addition (subtraction) algorithm:
    - when the signs of A and B are identical (different), add the two magnitude and attach the sign of A to the result.
    - When the sign of A and B are different (identical),compare the magnitudes and subtract the smaller number from the larger .

Table : 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

- To implement the two arithmetic operations with hardware, it is first necessary that the two numbers be stored in registers.
- Let A and B be two registers that hold the magnitude of the numbers, and A_s and B_s be two flip-flops that hold the corresponding signs. The results of the operation may be transferred to a third register however, a saving achieved if the result is transferred into A and A_s . thus A and A_s together from an accumulator register.
- Consider now the hardware implementation of the algorithms above.
  - First, a parallel adder is needed to perform the micro operation A+B.
  - second, comparator circuit is needed to establish if $A>B$, $A=B$, or $A<B$.
  - third, two parallel subtractor circuits are needed to perform the micro operation A-B and B-A.
  - The sign relationship can be determined from an exclusive-OR gate with As and B_s as inputs.

- The output carry is transferred to flip-flop E.
- The complementer consists of exclusive-OR gates and the parallel adder consists of full adder circuit.

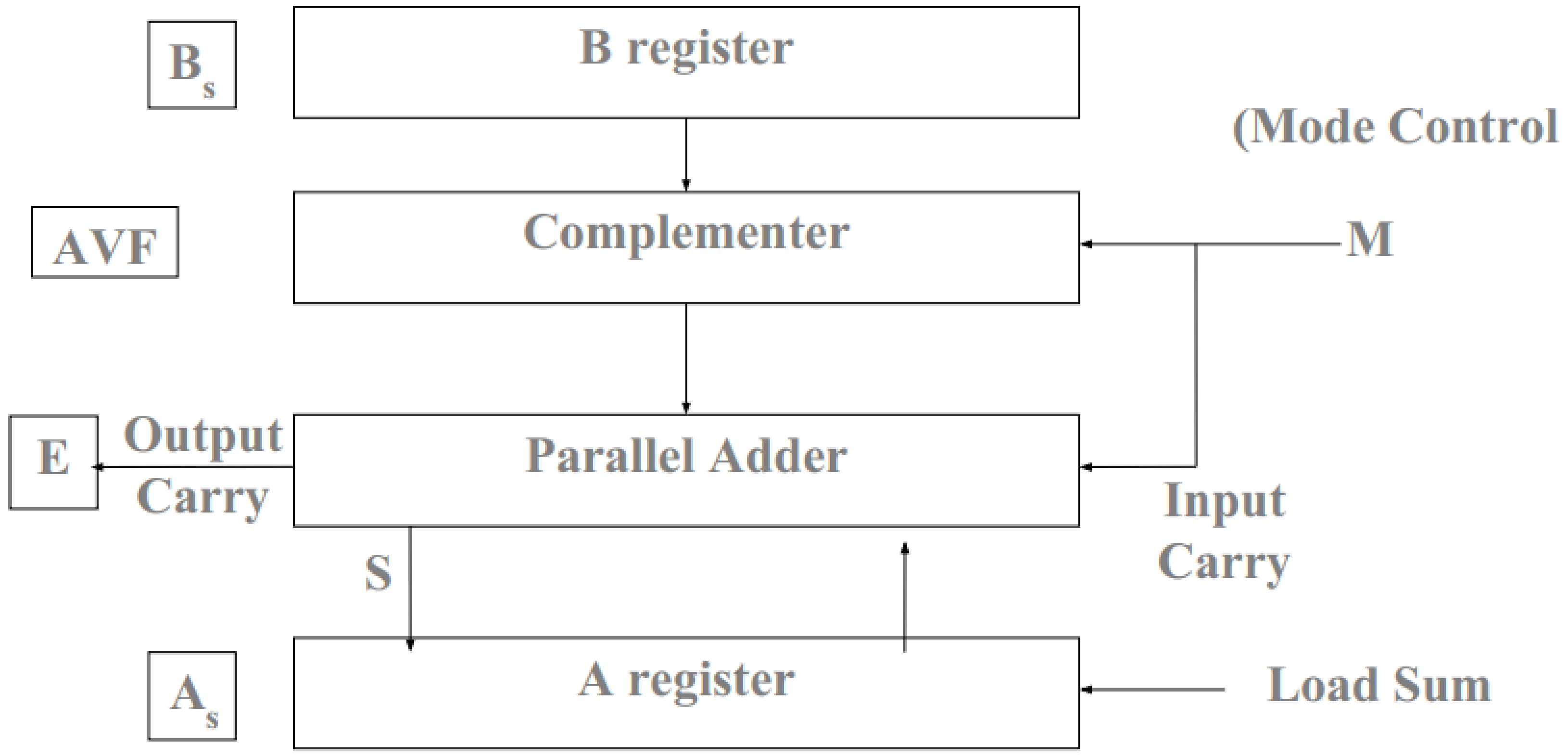


Figure : Hardware for signed magnitude addition and subtraction

For addition: A=3 and B=2, add A+B

```
+3          0011

+2          0010

……………………

  +5        0101
```

For subtraction: add A+B’ +1

```
+3          0011

-2          1110

……………………..

  +1        00 01
```

Hardware Algorithm

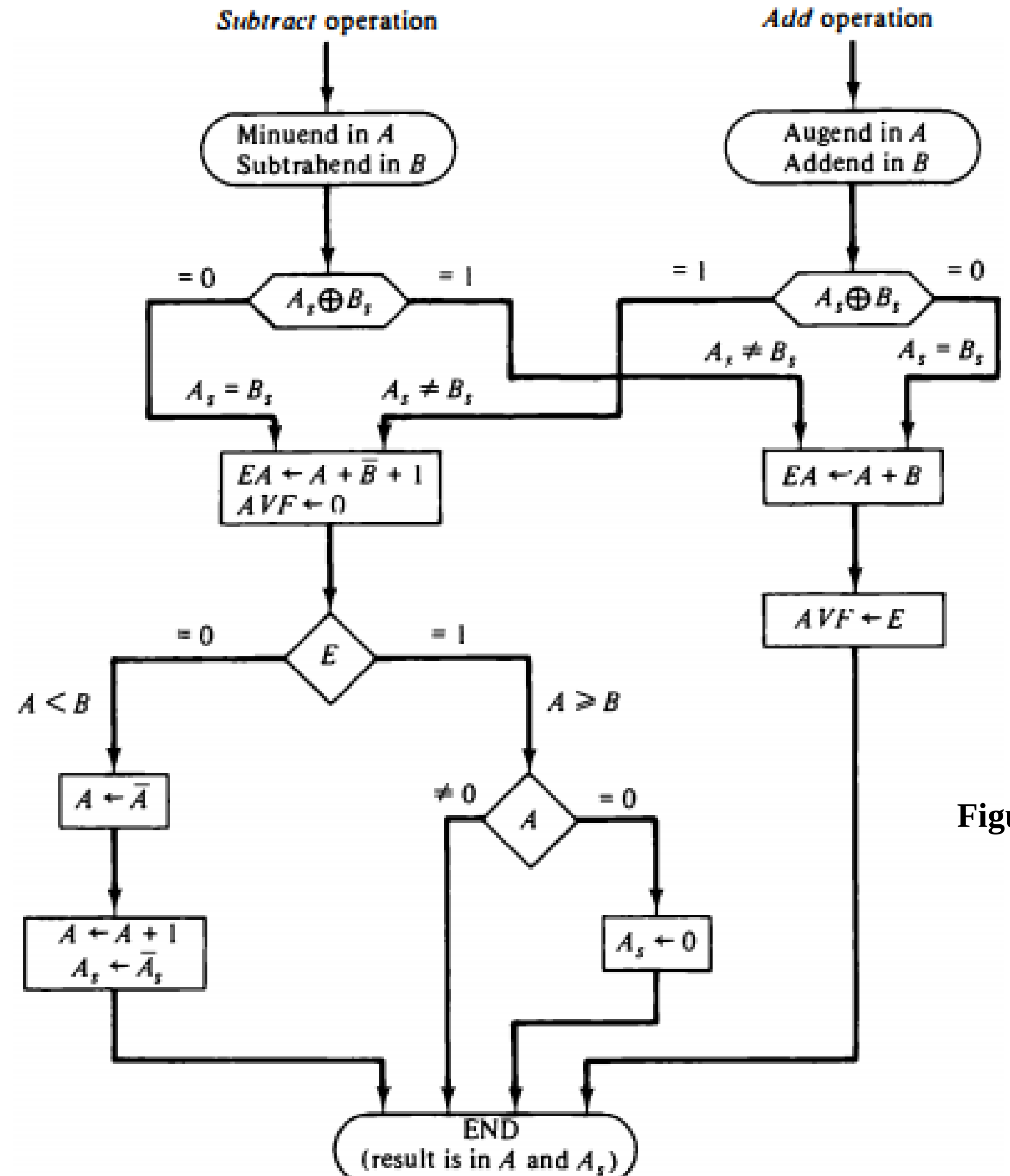


Figure : Flowchart for add and subtract operations

- The two signs A_s and B_s are compared by an exclusive-OR gate . If the output of the gate is 0, the signs are identical if it is 1, the signs are different.
- The magnitudes are added with a micro operation $EA \leftarrow A+B$.. Where EA is a register that combines E and A . The carry in E after the addition constitutes an overflow if it is equal to 1 and transferred into the add overflow flip-flop AVF.
- The two magnitudes are subtracted if the signs are different for an add operation or identical for a subtract operation.
- The magnitudes are subtracted by adding A to the 2’s complement of B.
- No overflow can occur if the numbers are subtracted to AVF is cleared to 0. A 1 in E, indicates that $A \geq B$ and the number in A is the correct result.
- If this number is zero, the sign A, must be made positive to avoid a negative zero. A 0 in E indicates that $A < B$.
- Examples : (+8) + (-3) = +5, (-8) +(-3)= -11, (+8) - (-3) = +11, (-8) - (-3) = -5

Addition and Subtraction with Signed 2’s Complement Data

- The left most bit of binary number represents the sign bit 0 for positive and 1 for negative. If the sign bit is 1, the entire the entire number is represented in 2’s compliment form (+33 is represented as 00100001 and -33 as 11011111. Note that 11011111 is the 2's complement of 00100001, and vice versa).
- The addition of two numbers in signed 2’s complement form consists of adding the number with the sign bits treated the same as the other bits of the number . A carry out of the sign bit position is discarded .
- The subtraction consists of first taking the 2’s compliment of the subtrahend and then adding it to the minuend
- When two numbers of n digits each are added and the sum occupies n+1 Digits, we say that an overflow occurred.
- When the two carries are applied to an exclusive-OR gate, the overflow is detected when the output of the gate is equal to 1.

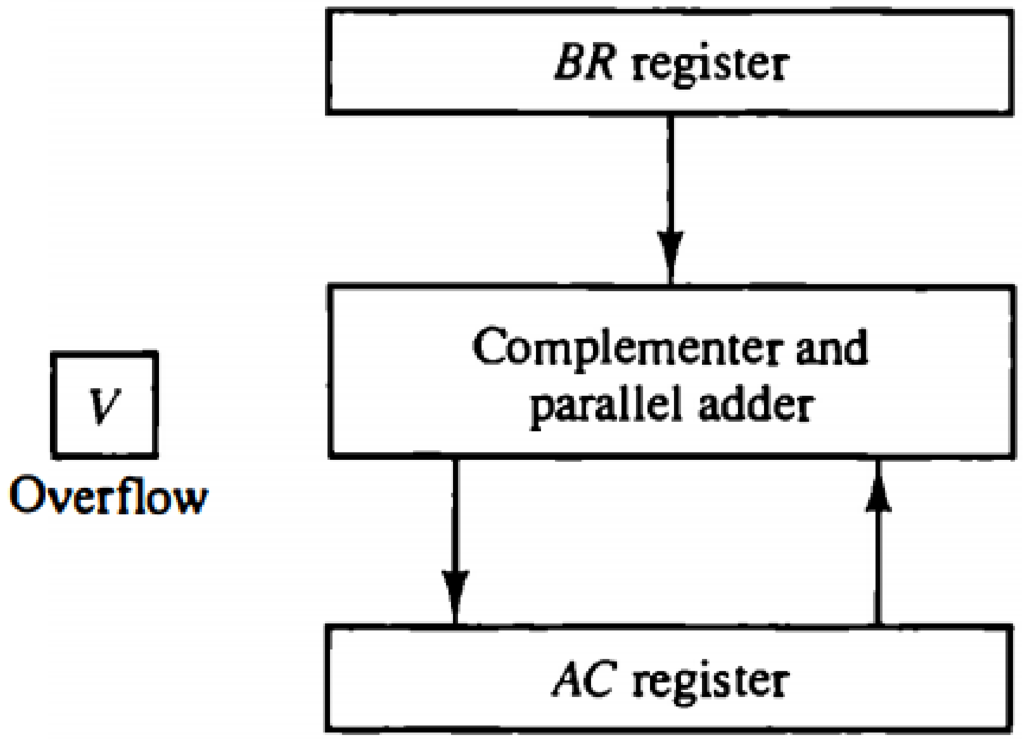


Figure : Hardware for signed 2's complement addition and subtraction.

- AC and BR are the registers to hold the numbers.
- The leftmost bit in AC and BR represent the sign bits of the numbers.
- The two sign bits are added or subtracted together with the other bits in the complementer and parallel adder.
- The overflow flip flop V is set to 1 if there is an overflow. The output carry in this case is discarded.

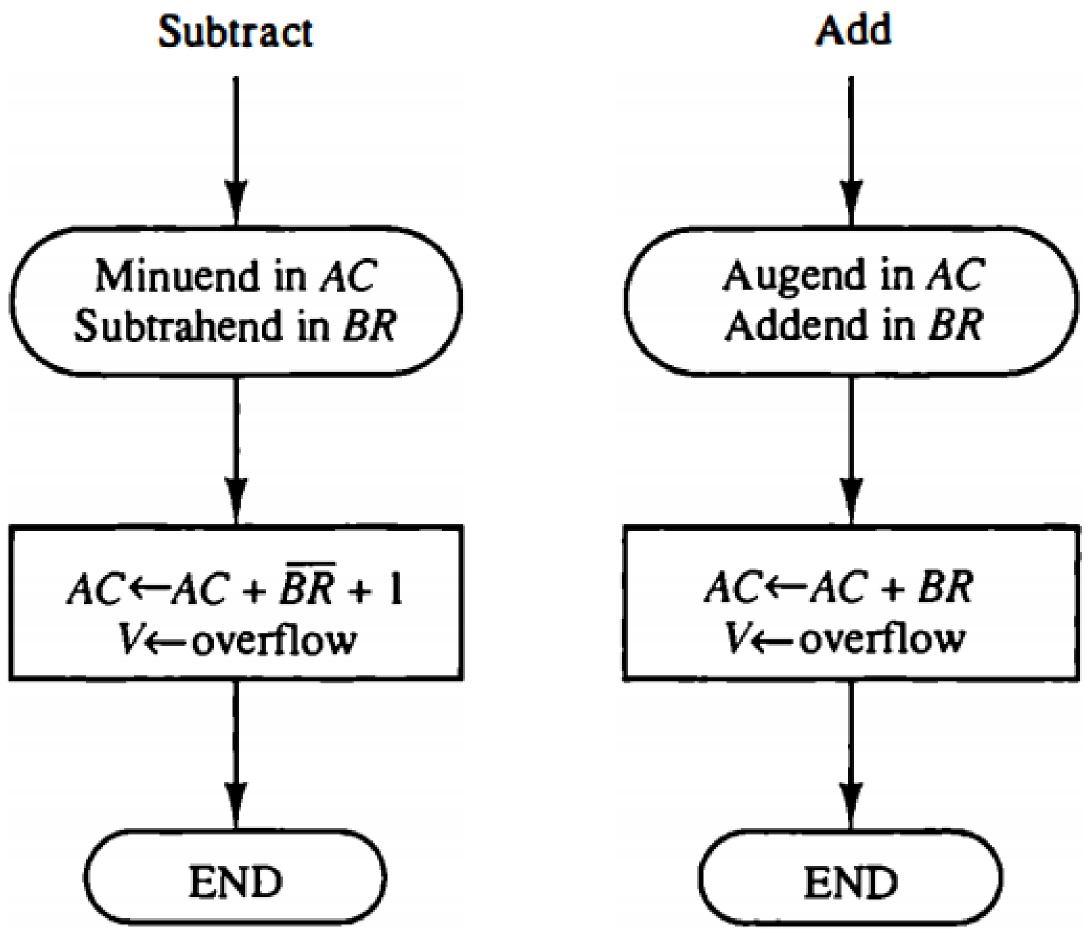


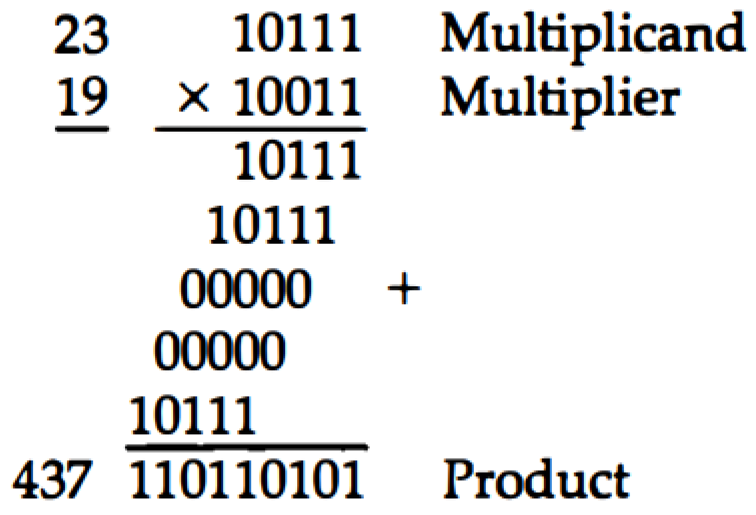
Figure : Algorithm for adding and subtracting numbers in signed-2's complement representation.

- The sum is obtained by adding the contents of AC and BR.
- The overflow bit V is set to 1 if the exclusive OR of the last two carries is 1, and it is cleared to 0 otherwise.
- The subtraction operation is accomplished by adding the content of AC to the 2's complement of BR. Taking the 2's complement of BR has the effect of changing a positive number to negative, and vice versa.

Multiplication Algorithms

Signed Magnitude representation

- Multiplication of two fixed point binary numbers in signed magnitude representation is done with paper and pencil of successive shift and add operation.
- If the multiplier bit is a 1, the multiplicand is copied down; otherwise, zeros are copied down.



```
 23        10111     Multiplicand
 19      × 10011     Multiplier
           10111
          10111
         00000    +
        00000
       10111
437    110110101     Product
```

Hardware Implementation for Signed Magnitude data

- When multiplication is implemented in a digital computer, it is convenient to change the process slightly. First instead of providing register to store and add simultaneously as many binary numbers as there are bits in the multiplier , as it is convenient to provide an adder for the summation of only two binary numbers and successively accumulate the partial products in a register. Second instead of shifting the multiplicand to the left , the partial product is shifted to the right
- The hardware for multiplication consists of the equipment shown in fig. plus two are more registers.
- These registers are together with registers A and B.
- The multiplier stored in the Q register and its sign in Qs The sequence counter SC is initially set to a number equal to the number of bits in the multiplier. The counter is decremented by 1 after forming each partial product
- The sum of A and B forms a partial product which is transferred to the EA register .

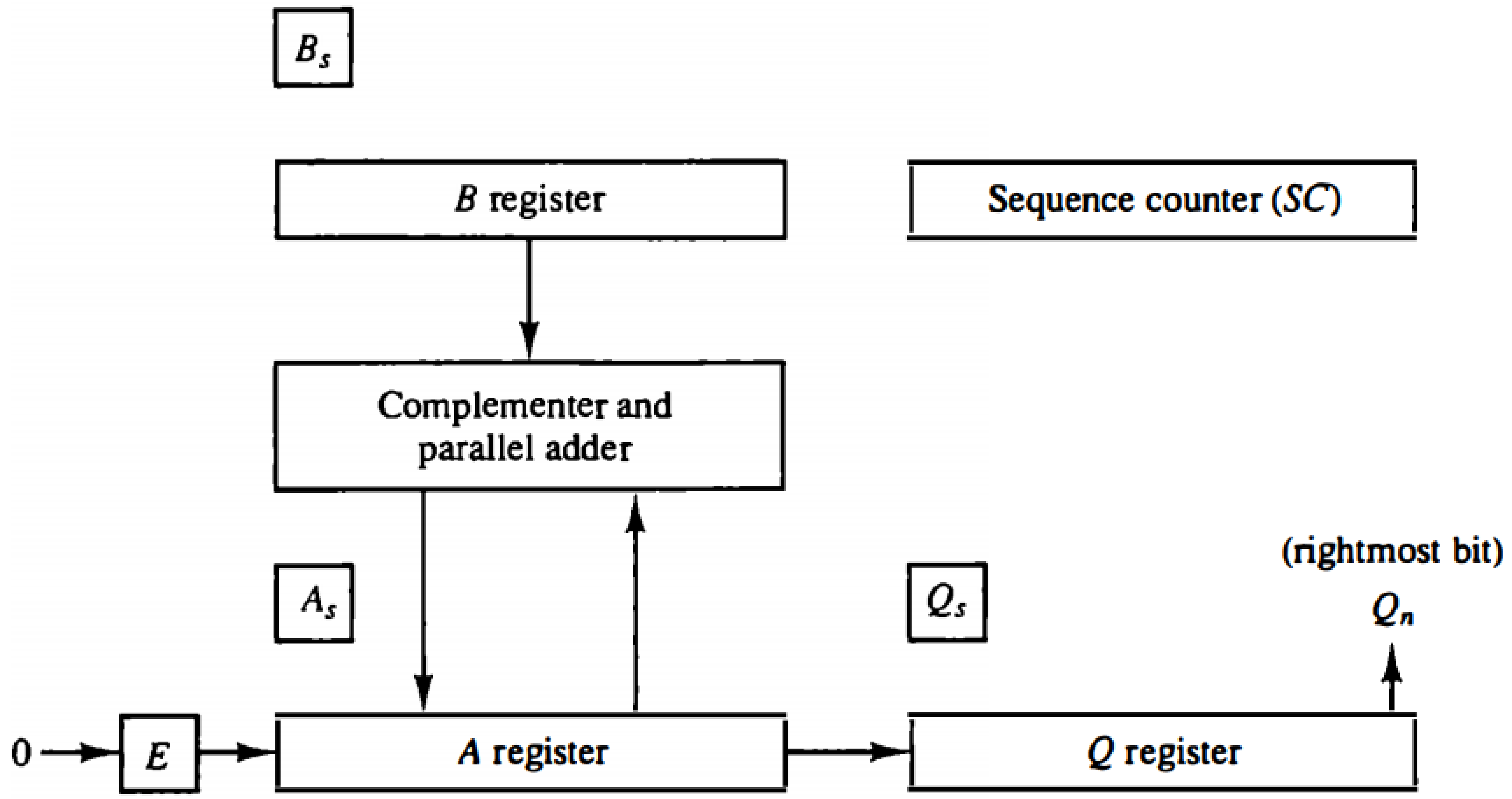


Figure : Hardware for multiply operation.

- The shift will be denoted by the statement shr EAQ to designate the right shift depicted .
- The least significant bit of A is shifted into the most significant position of Q.

Hardware Algorithm

- Initially the multiplicand is in B and the multiplier in Q there corresponding signs are in B_s and Q_s .,respectively.
- Register A and E are cleared and the sequence counter SC is set to a number equal to the number of bits of the multiplier.
- After the initialization , the low order bit of the multiplier is in Q_n is tested .if it is 1,the multiplicand In B is added to the present partial product in A . If it is 0, nothing is done .
- Register EAQ shifted once to the right to form the new partial product.
- The process stops when SC=0.
- Note that the partial product formed in A is shifted into Q one bit at a time and eventually replaces multiplier. The final product is available in both A and Q,with A holding the most significant bits and Q holding the least significant bits.

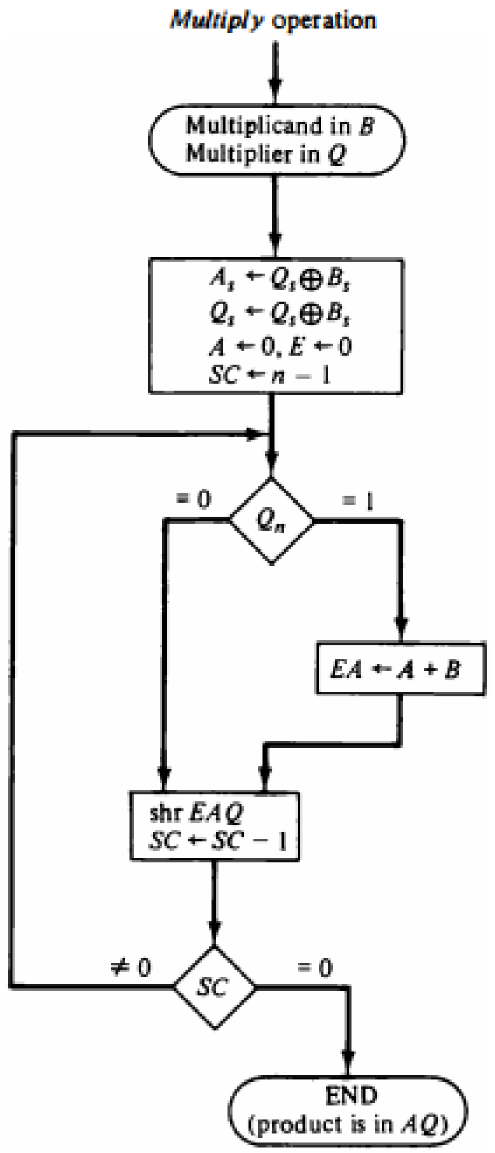
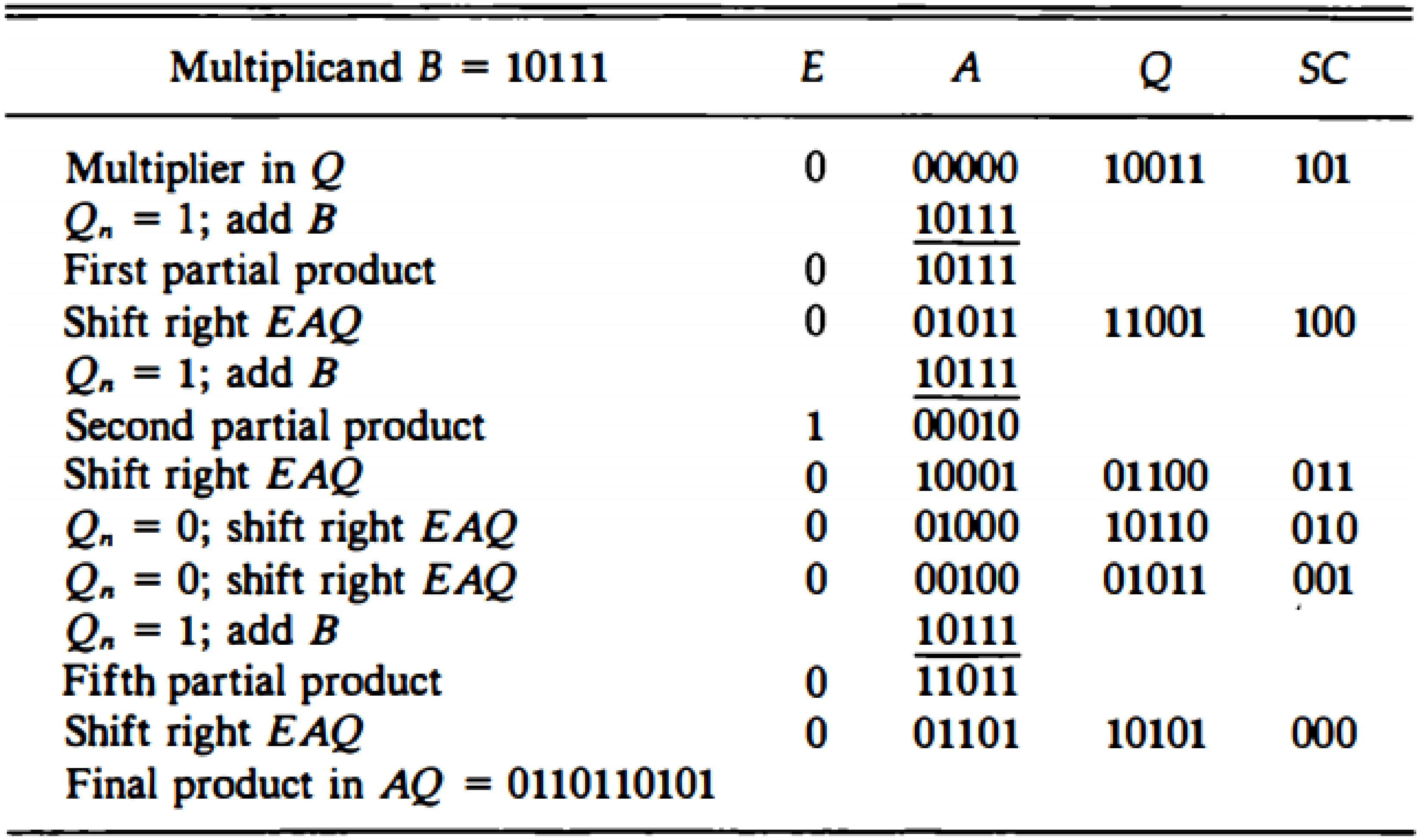


Figure : Flowchart for multiply operation.

Booth Multiplication Algorithm

Example: 23 x 19 = 437, B=10111 and Q=10011

TABLE : 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 $AQ = 0110110101$				

Signed 2’s complement representation

- Booth algorithm gives a procedure for multiplying binary integers in signed 2’s complement representation.
- It operates on the fact that strings of 0's in the multiplier require no addition but just shifting, and a string of 1's in the multiplier from bit weight 2^k to weight 2^m can be treated as $2^{k+1} – 2^m$.
- For example, the binary number 001110 (+ 14) has a string of 1's from 2^3 to 2^1 ($k = 3, m = 1$). The number can be represented as $2^{k-1} - 2^m = 2^4 – 2^1 = 16-2 = 14$.
- Therefore, the multiplication M x 14, where M is the multiplicand and 14 the multiplier, can be done as $M \times 2^4 - M X 2^1$ Thus the product can be obtained by shifting the binary multiplicand M four times to the left and subtracting M shifted left once.

Hardware for booth algorithm

- The two bits of multiplier in Q_n and Q_{n+1} are inspected . If the two bits are equal to 10 it means that the first 1 in a string of 1's has been encountered.
- The final value of Q_{n+1} is the original sign bit of the multiplier and should not be taken as part of the product.

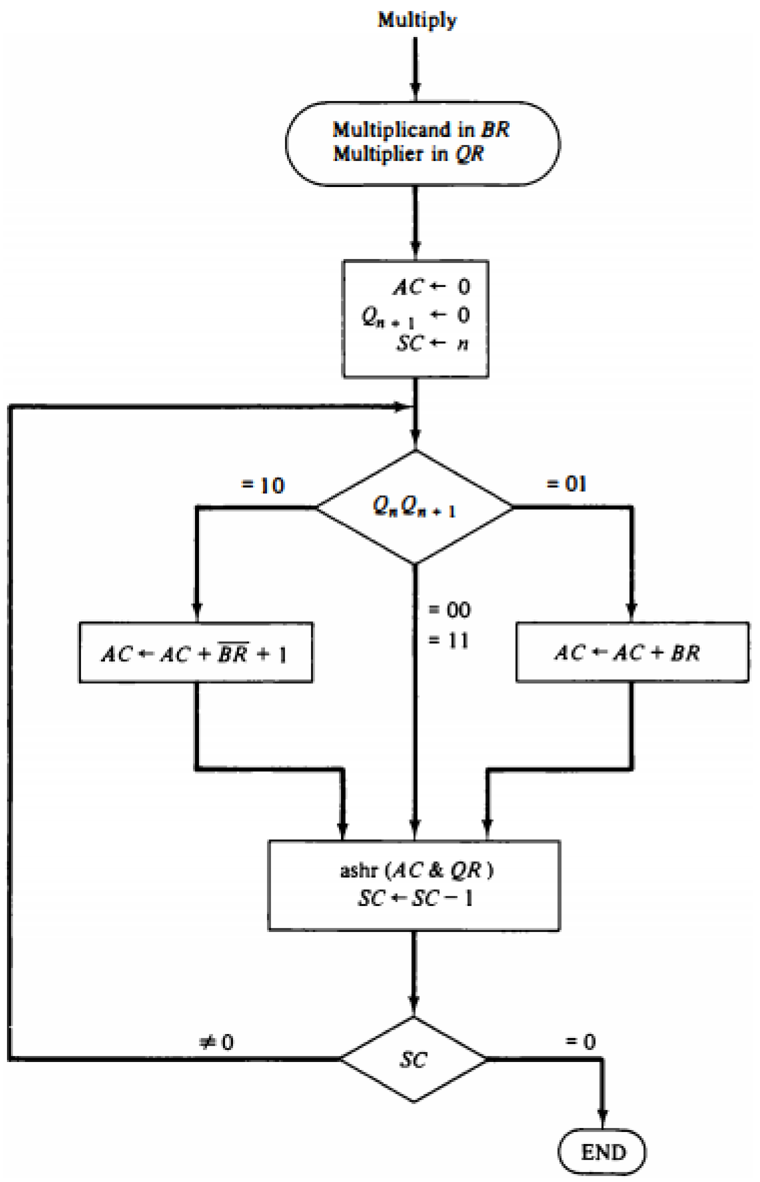


Figure : Booth algorithm for multiplication of signed 2's complement numbers.

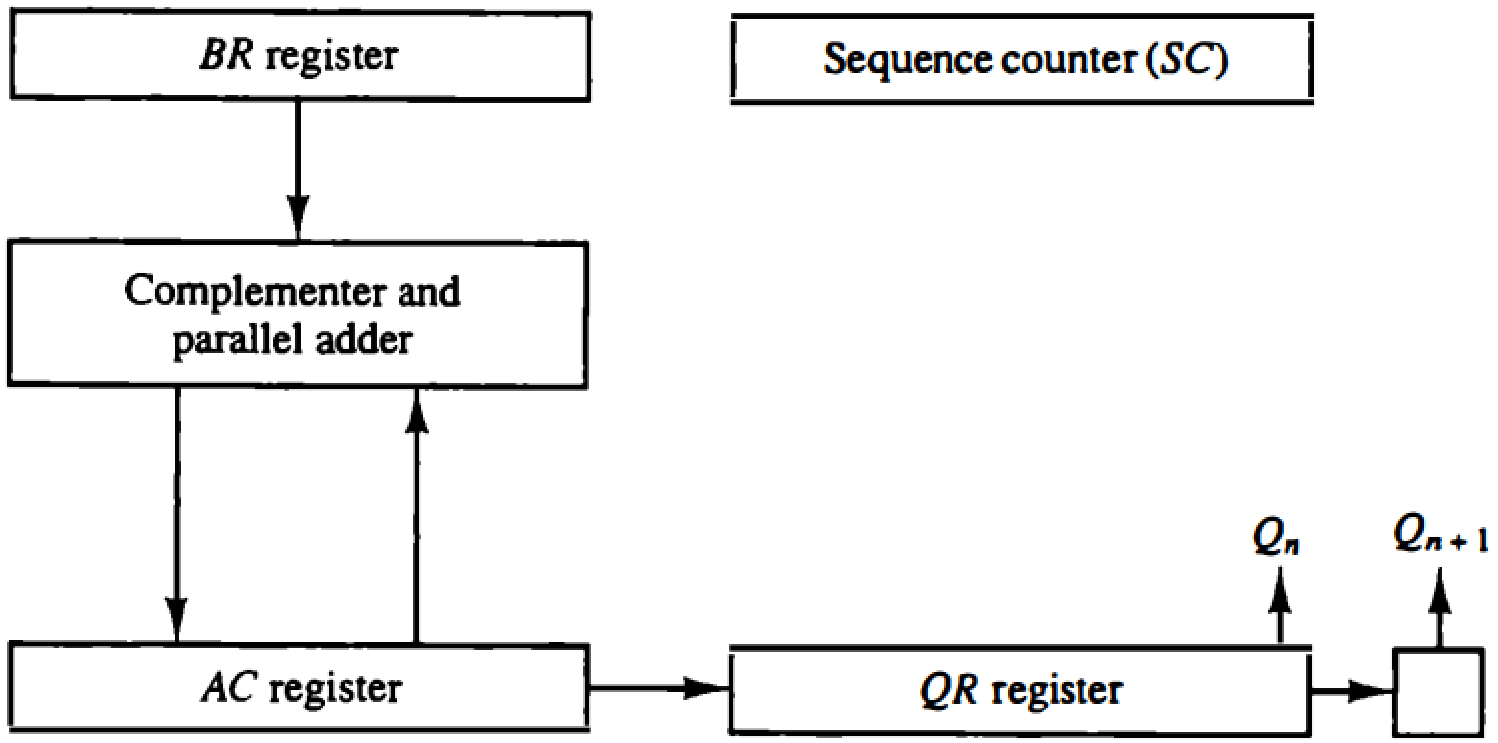
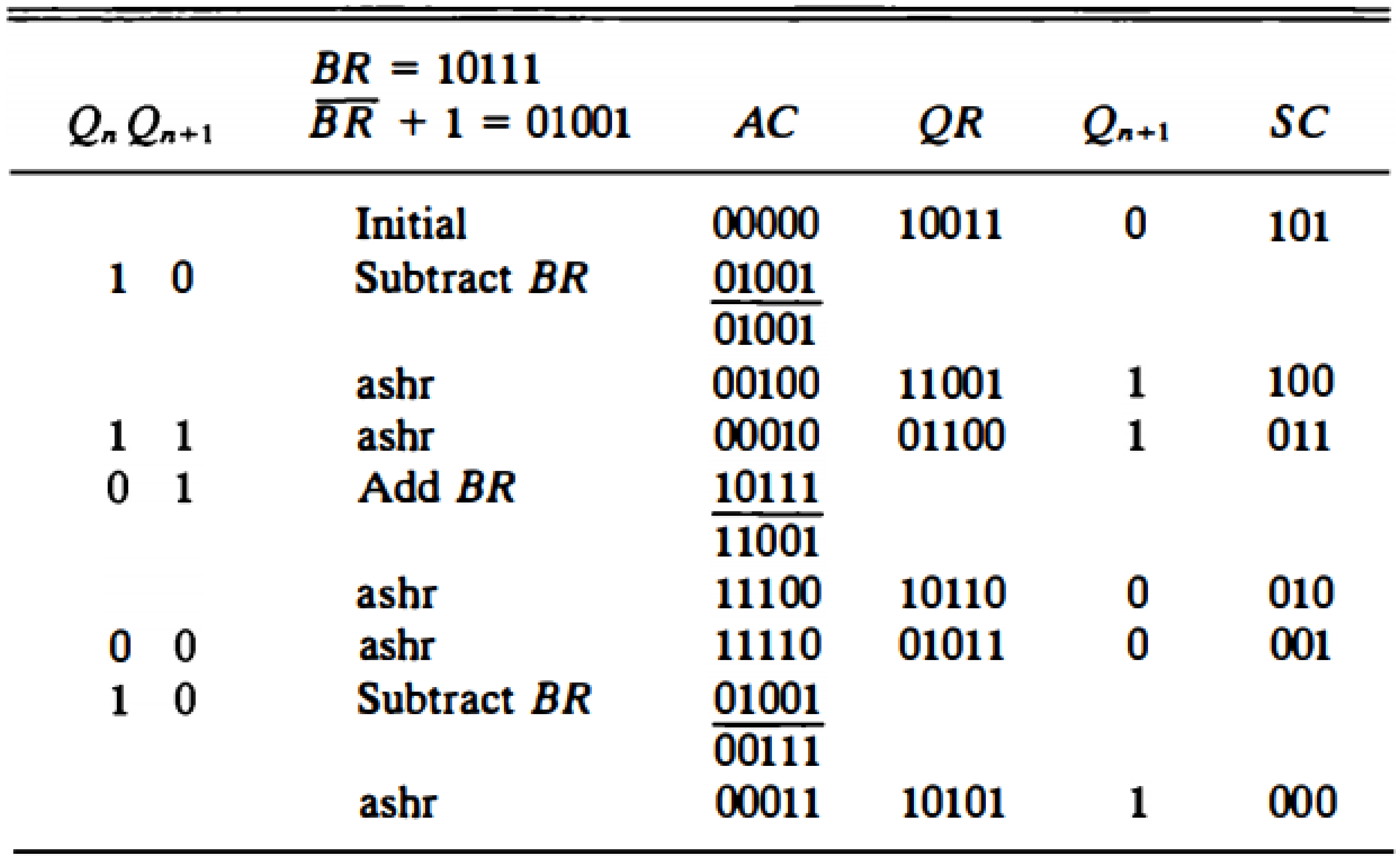


Figure : Hardware for Booth algorithm

Example : (-9) x (- 13) = + 117
BR=10111 (multiplicand) and QR=10011 (multiplier)



$Q_n \, Q_{n+1}$	$BR = 10111$ $\overline{BR} + 1 = 01001$	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

Q. Multiply the two numbers 23 and -9 by using the Booth's multiplication algorithm.

Here, BR = 23 = (010111) and QR= -9 = (110111)

Q_n	**Q_{n+1}**	**BR= 0 1 0 1 1 1** **BR' + 1 = 1 0 1 0 0 1** **Operation**	**AC**	**QR**	**Q_{n+1}**	**SC**
		Initially	000000	110111	0	6
1	0	Subtract BR	101001			
			101001			
		ashr	110100	111011	1	5
1	1	ashr	111010	011101	1	4
1	1	ashr	111101	001110	1	3
0	1	Addition BR	010111			
			010100			
		ashr	001010	000111	0	2
1	0	Subtract BR	101001			
			110011			
		ashr	111001	100011	1	1
1	1	ashr	111100	110001	1	0

$\mathbf{Q_{n+1}}$ = 1, it means the output is negative.
Hence, 23 * -9 = 2's complement of 111100110001 => (00001100111)

Solve

Q. Multiply 3 x 5 using booth algorithm.

Q. Multiply -5 x 4 using booth algorithm.

Q. Multiply (-10) *(-4) using Booth's algorithm.

Array multiplier

- The multiplication of the two binary numbers can be done with one micro-operation by means of a combinational circuit that forms the product bits all at once. This is a fast way of multiplying two numbers since all it takes is the time for the signals to propagate through the gate that form the multiplication array.

$$\begin{array}{cccc}
 & & b_1 & b_0 \\
 & & a_1 & a_0 \\
\cline{3-4}
 & & a_0b_1 & a_0b_0 \\
 & a_1b_1 & a_1b_0 & \\
\hline
c_3 & c_2 & c_1 & c_0
\end{array}$$

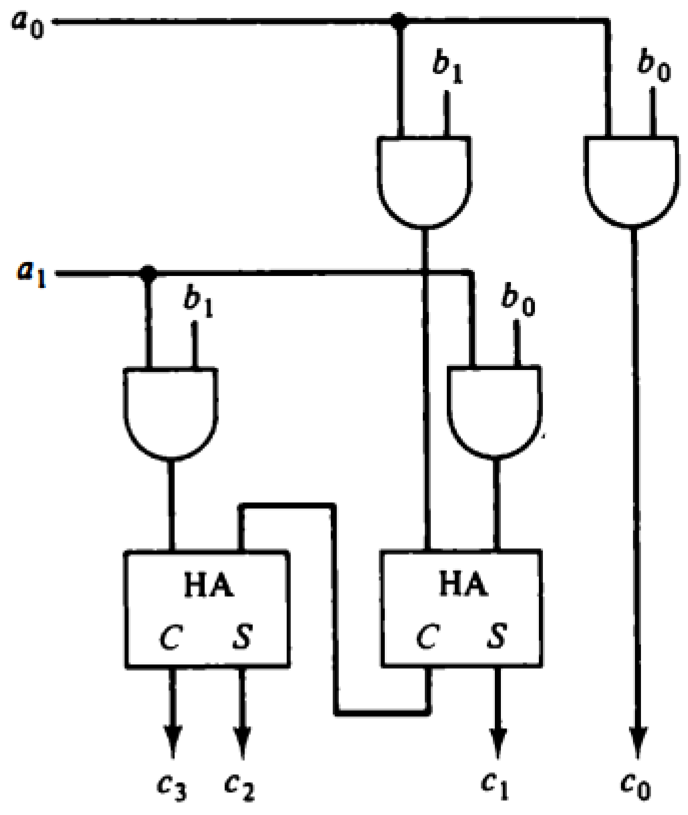


Figure : 2-bit by 2-bit array multiplier

- Consider the multiplication of two 2-bit numbers as shown in Figure. The multiplicand bits are b1 and b0, the multiplier bits are a1 and a0, and the product is c3 c2 c1 c0.
- This is identical to an AND operation and can be implemented with an AND gate. the first partial product is formed by means of two AND gates.
- The second partial product is formed by multiplying a1 by b1 b0 and is shifted one position to the left. The two partial products are added with two half-adder (HA) circuits.
- A combinational circuit binary multiplier with more bits can be constructed in a similar fashion.
- For j multiplier bits and k multiplicand bits we need j x k AND gates and (j - 1) k-bit adders to produce a product of j + k bits.

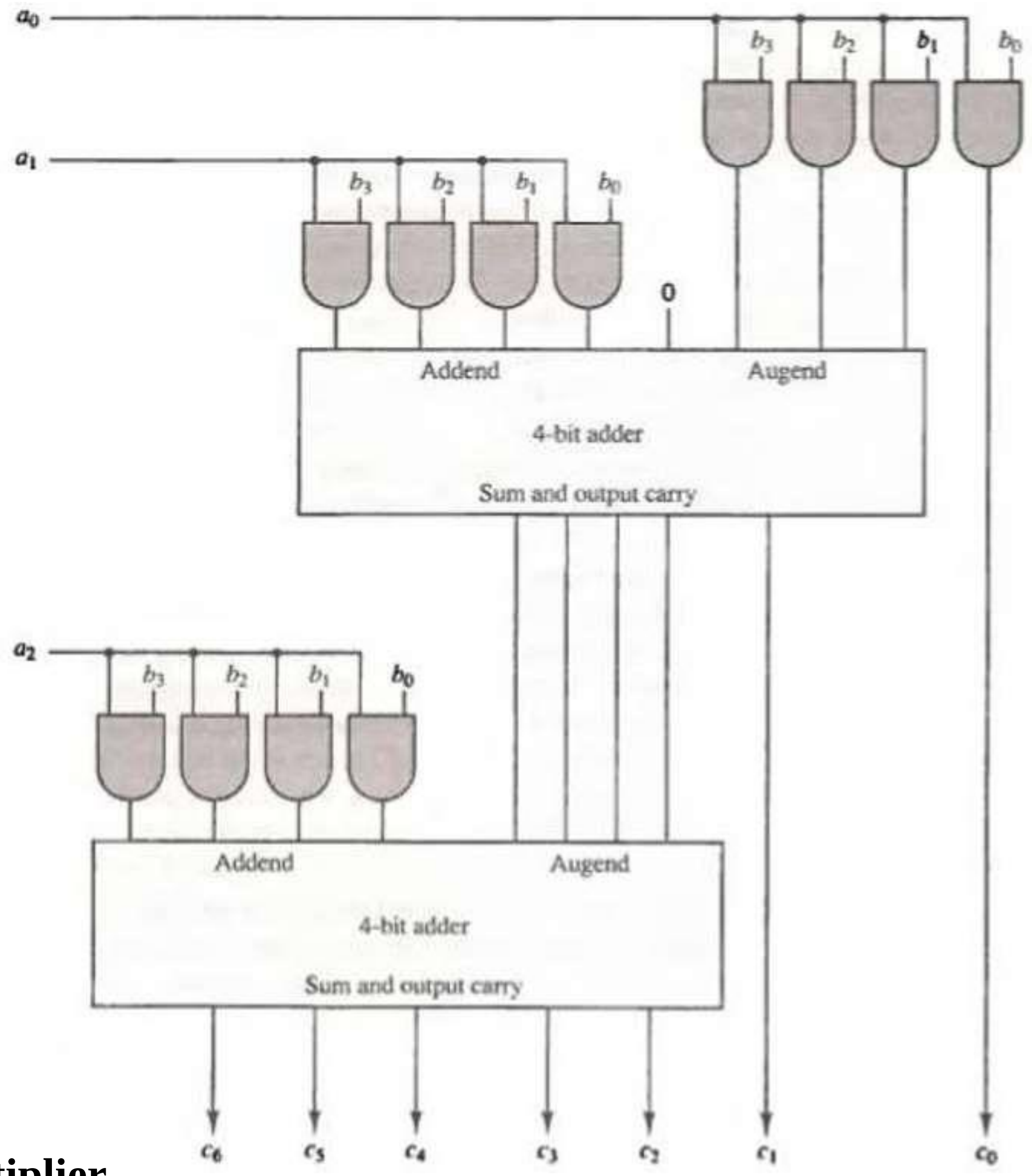
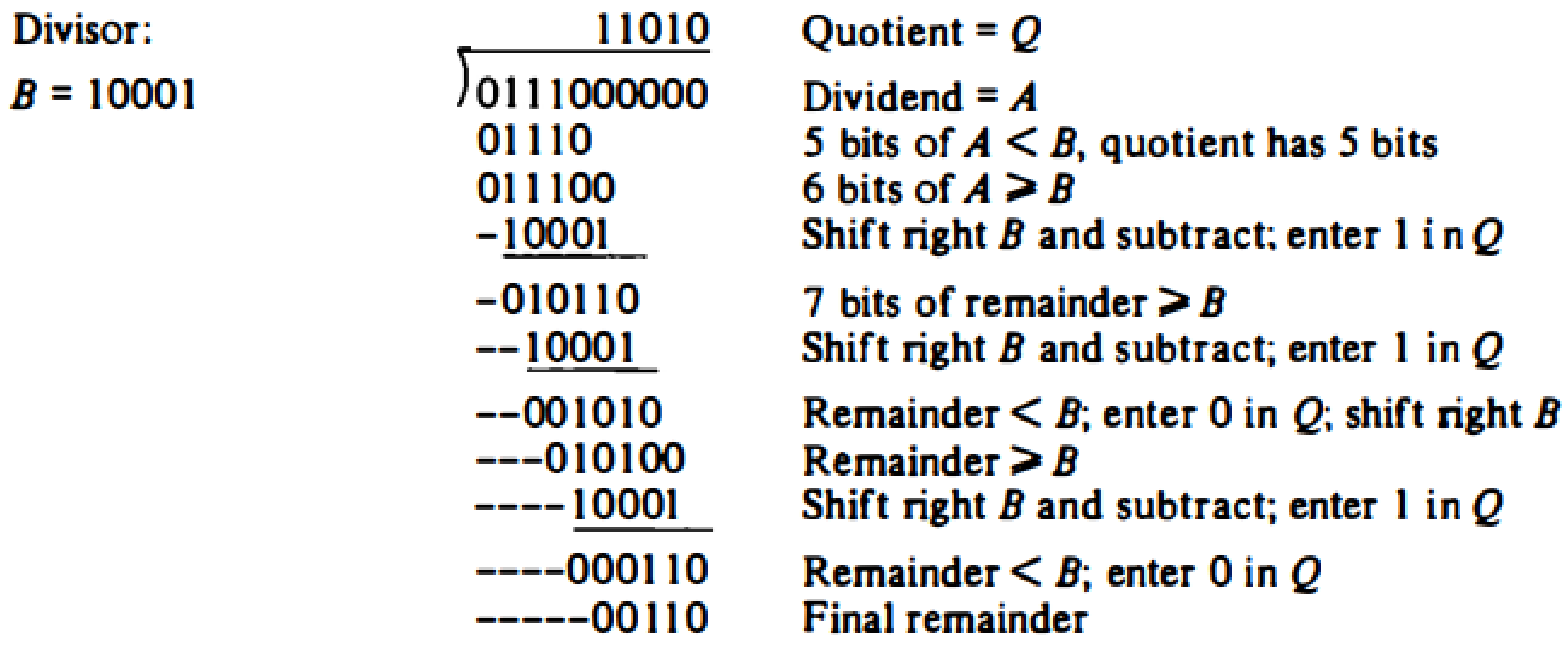


Figure : 4-bit by 3-bit array multiplier

Division Algorithm

- Division of two fixed-point binary numbers in signed magnitude representation is done with paper and pencil by a process of successive compare ,shift ,and subtract operations.



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

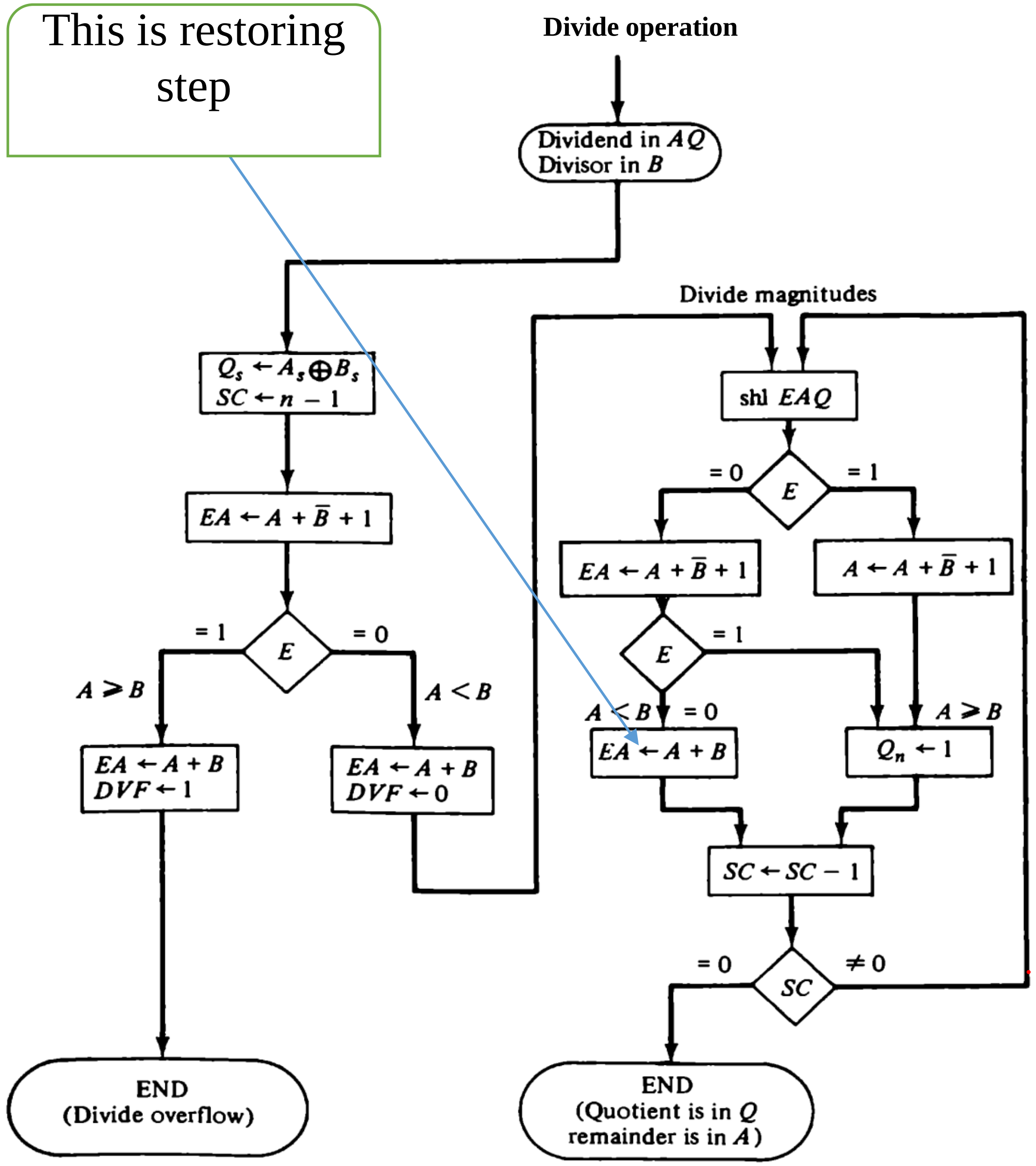
Figure : Example of binary division, 448/17

Hardware Implementation for Signed-Magnitude Data

- When the division is implemented in a digital computer, it is convenient to change the process slightly. Instead of shifting the divisor to the right, the dividend, or partial remainder, is shifted to the left, thus leaving the two numbers in the required relative position. Subtraction may be achieved by adding A to the 2's complement of B. The information about the relative magnitudes is then available from the end-carry.
- Register EAQ is now shifted to the left with 0 inserted into Q, and the previous value of E lost.
- **Restoring Method:** The reason for the name is that the partial remainder is restored by adding the divisor to the negative difference. Two other methods are available for dividing numbers, the Comparision method and the Nonrestoring method.
- No restoring method B is not added if the difference is negative but instead, the negative difference is shifted left and then B is added.

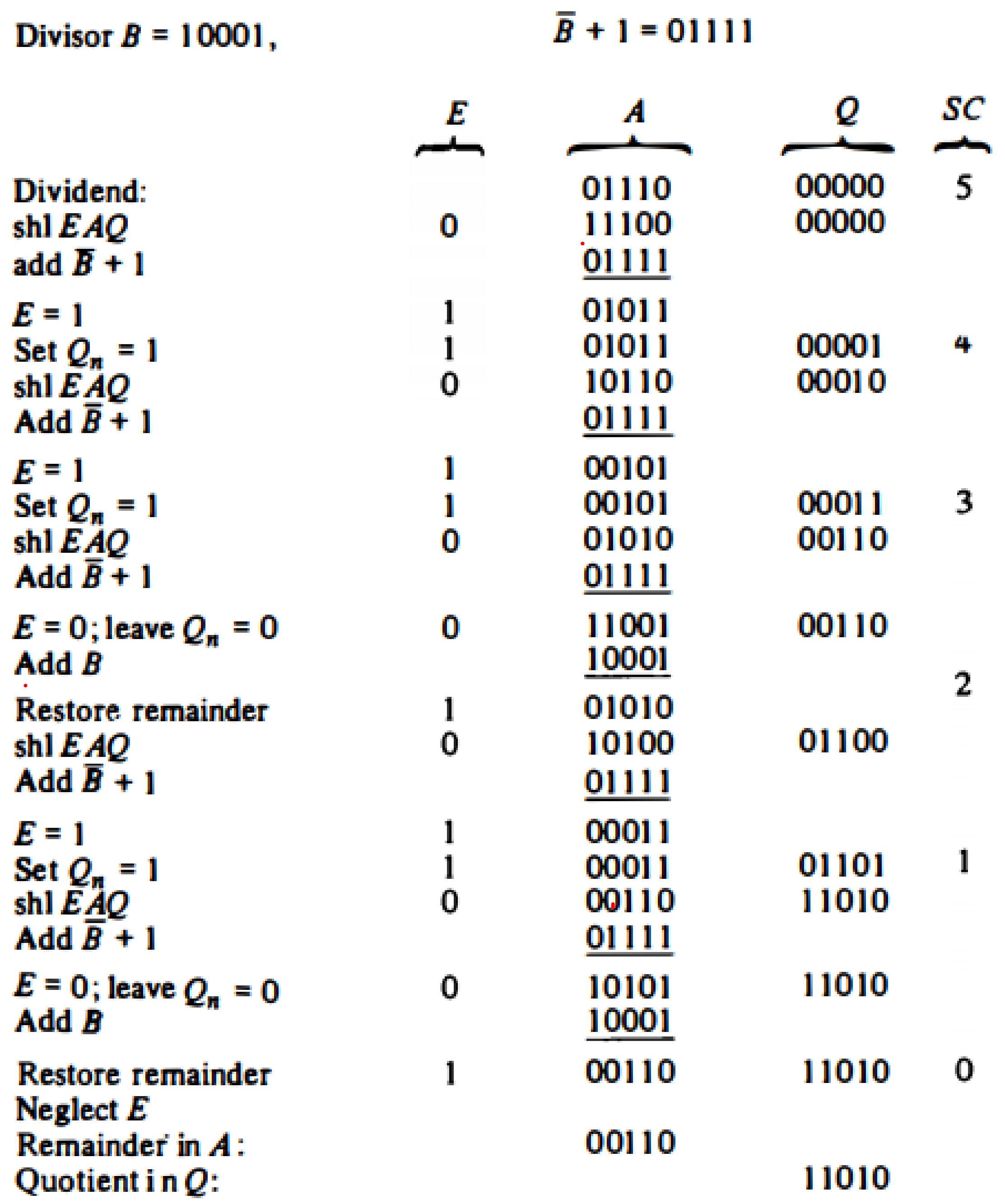
Hardware Algorithm (Restoring Algorithm)

- The dividend AQ and the divisor is B . The sign of the result is transferred into Q_s to be part of the quotient.
- The sequence counter SC to specify the number of bits in the quotient.
- If A >= B, the divide-overflow flip-flop DVF is set and the operation is terminated prematurely. If A < B, no divide overflow occurs so the value of the dividend is restored by adding B to A .



- The division o f the magnitudes starts by shifting the dividend in A Q to the left with the high-order bit shifted into E. If the bit shifted into E =1, EA > B so B is subtracted from EA and Q_n is set to 1.
- If E=0, the divisor is subtracted by adding its 2's complement value and the carry is transferred into E .If E = 1, it signifies that A >=B; therefore, Q_n is set to 1. If E = 0, it signifies that A < B and the original number is restored by adding B to A and we leave a 0 in Q_n .
- This process is repeated again with register A holding the partial remainder. After n - 1 times, the quotient magnitude is formed in register Q and the remainder is found in register A . The quotient sign is in Q_s and the sign of the remainder in A_s

Example : 448/17

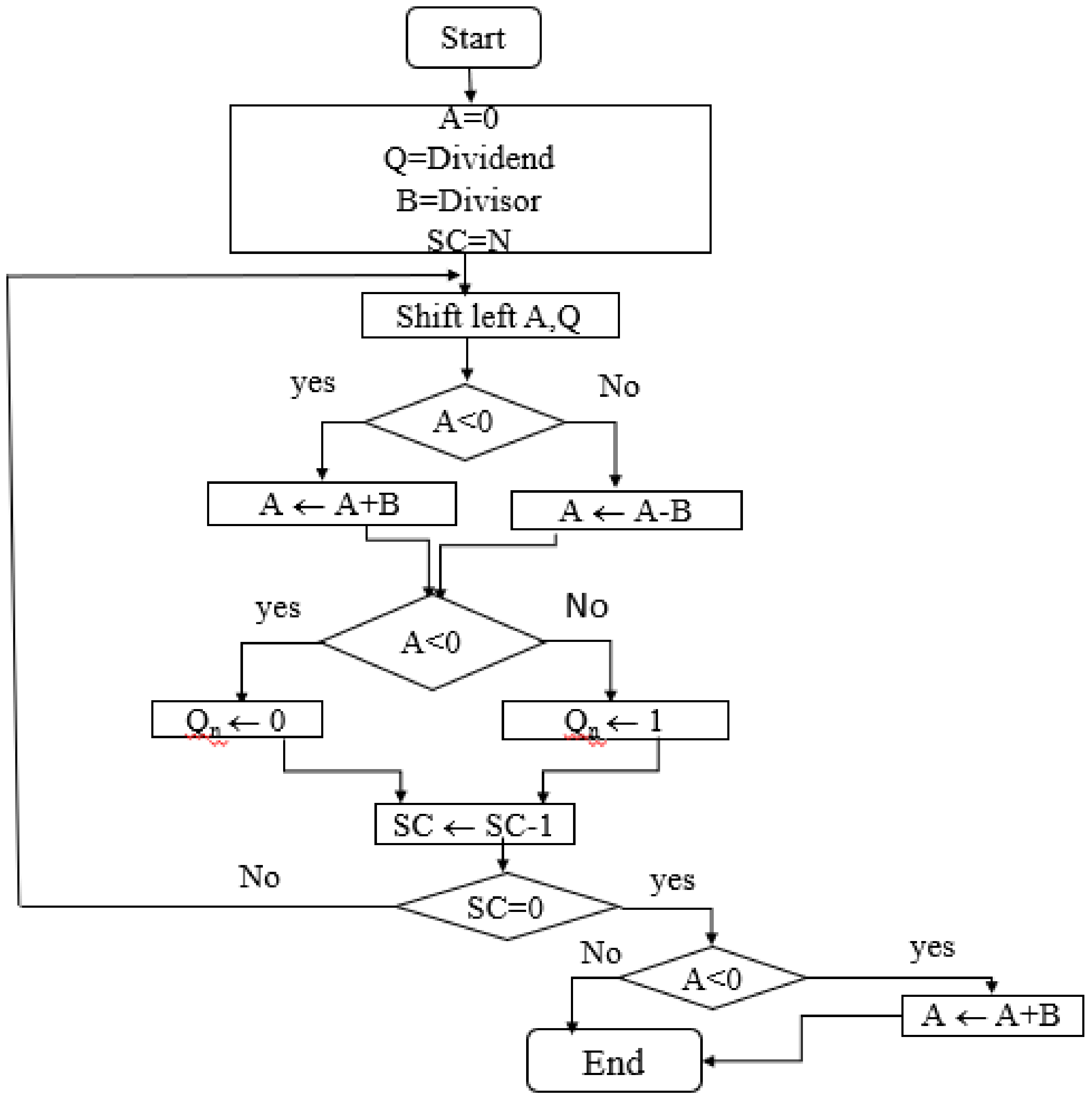


Divisor $B = 10001$, $\overline{B} + 1 = 01111$

	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	

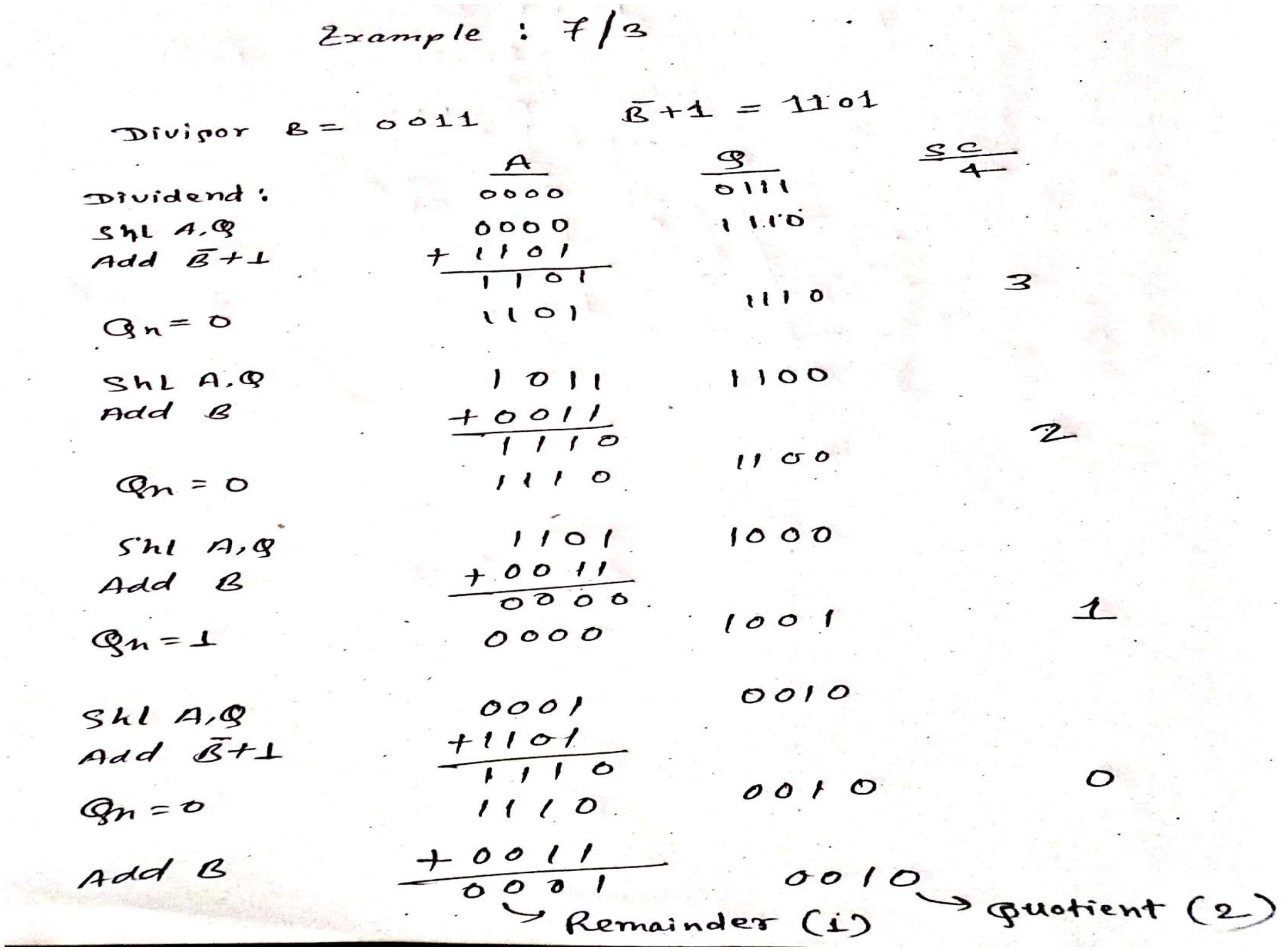
Figure :Example of binary division with digital hardware.

Non-restoring division Algorithm



- First the registers are initialized with corresponding values (Q = Dividend, B= Divisor, A = 0, SC = number of bits in dividend)
- Step2: Check the sign bit of register A
- If it is 1 shift left content of AQ and perform A = A+B, otherwise shift left AQ and perform A = A-B (means add 2's complement of M to A and store it to A)
- Again the sign bit of register A
- If sign bit is 1 Q_n become 0 otherwise Q_n become 1 (Q_n means least significant bit of register Q)
- Decrements value of SC by 1
- If SC is not equal to zero go to **Step 2** otherwise go to next step
- If sign bit of A is 1 then perform A = A+B
- Register Q contain quotient and A contain remainder

Example: 7/3



Example : 7/3

Divisor B = 0011 $\bar{B}+1$ = 1101

	A	Q	SC
Dividend :	0000	0111	4
ShL A,Q	0000	1110	
Add $\bar{B}+1$	+ 1101		
	1101	1110	3
$Q_n = 0$	1101		
ShL A,Q	1011	1100	
Add B	+ 0011		
	1110	1100	2
$Q_n = 0$	1110		
ShL A,Q	1101	1000	
Add B	+ 0011		
	0000	1001	1
$Q_n = 1$	0000		
ShL A,Q	0001	0010	
Add $\bar{B}+1$	+ 1101		
	1110	0010	0
$Q_n = 0$	1110		
Add B	+ 0011		
	0001 → Remainder (1)	0010 → Quotient (2)	

Q. Divide 11/3 using non-restoring division

Q. Divide 5/3 using non-restoring division