Computer Arithmetic

Addition and Subtraction

Dr. Mohammed Abdulridha Hussain

Signed Magnitude

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

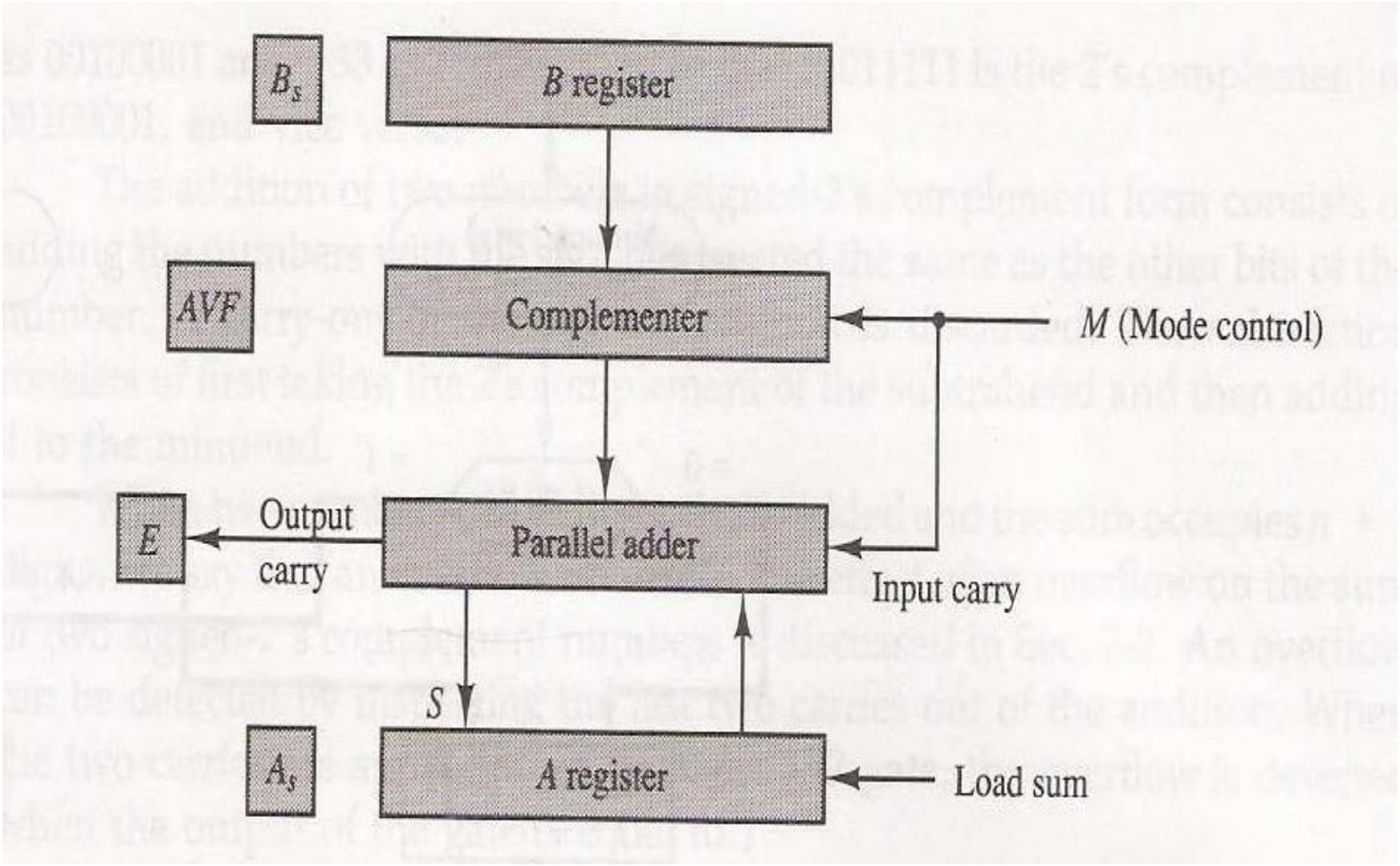
Signed Magnitude

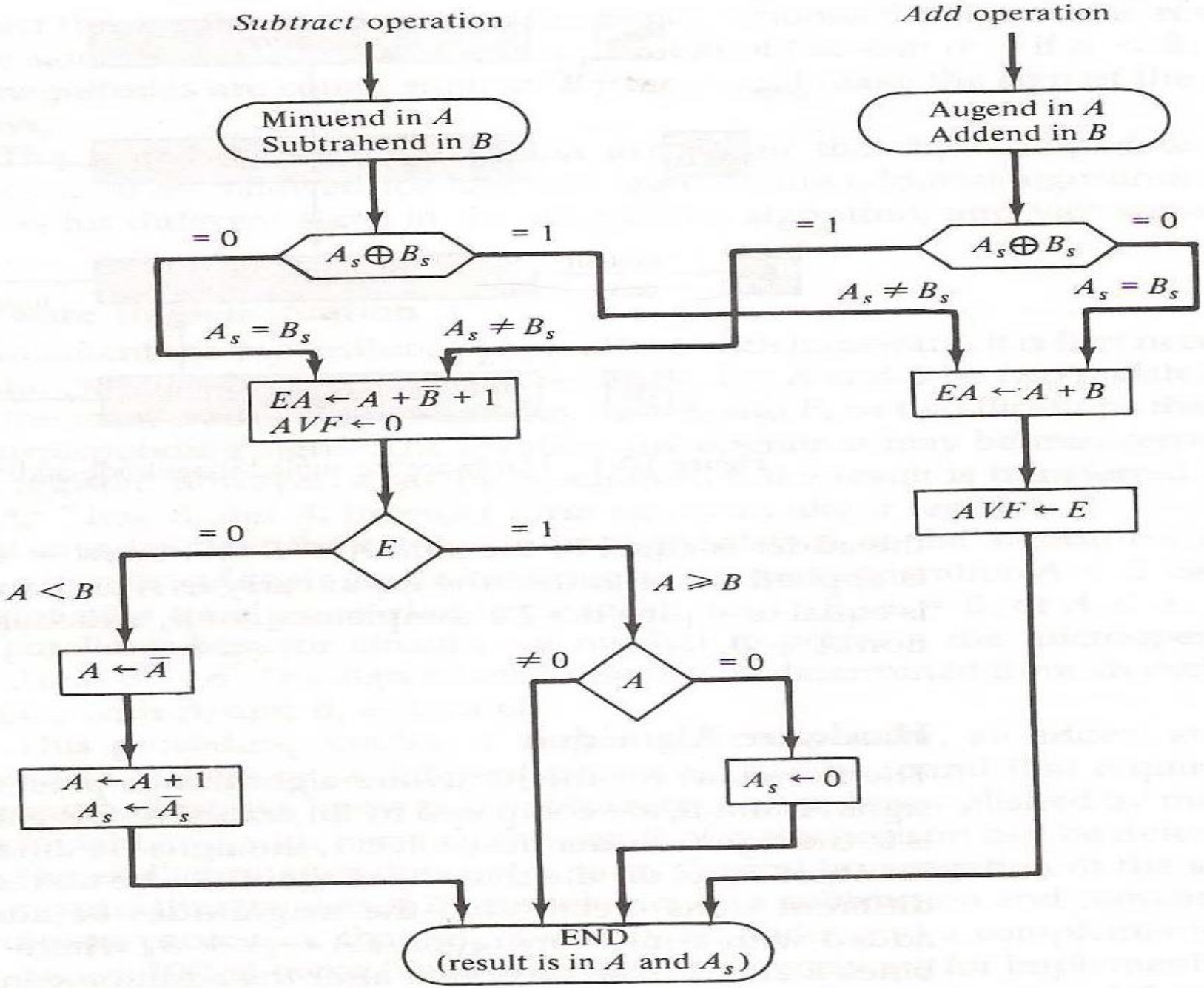
- Addition(Subtraction) algorithm:
- When the signs of A and B are identical . Add the two magnitudes and attach the sign of A to the result.
- When the signs of A and B are different , compare the magnitudes and subtract the smaller from the larger. Choose the sign of the result to be the same as A if A>B or the complement of the sign of A if A<B.
- If the two magnitudes are equal, subtract B from A and make the sign of the result positive.

Hardware Implementation

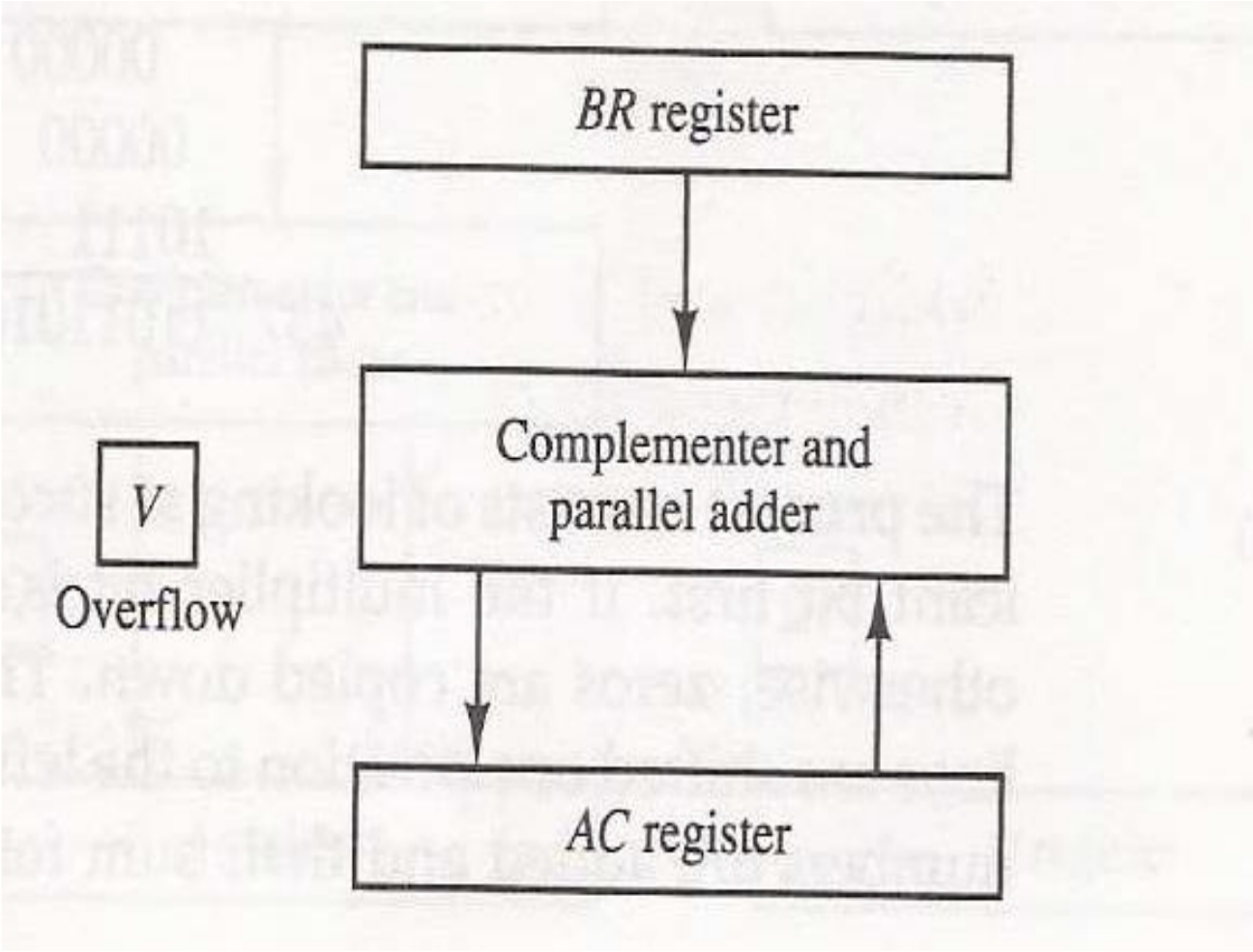
- Let A & B two registers that hold the magnitudes of the numbers and As & Bs be two flip-flop that hold the corresponding signs.
- Complementer = XOR
- Adder = Full Adder
- E = Carry; AVF = overflow register
- If M = 0 ; Transfer B & Add
- If M = 1; $s = A + \bar{B} + 1 = A - B$

Hardware Implementation

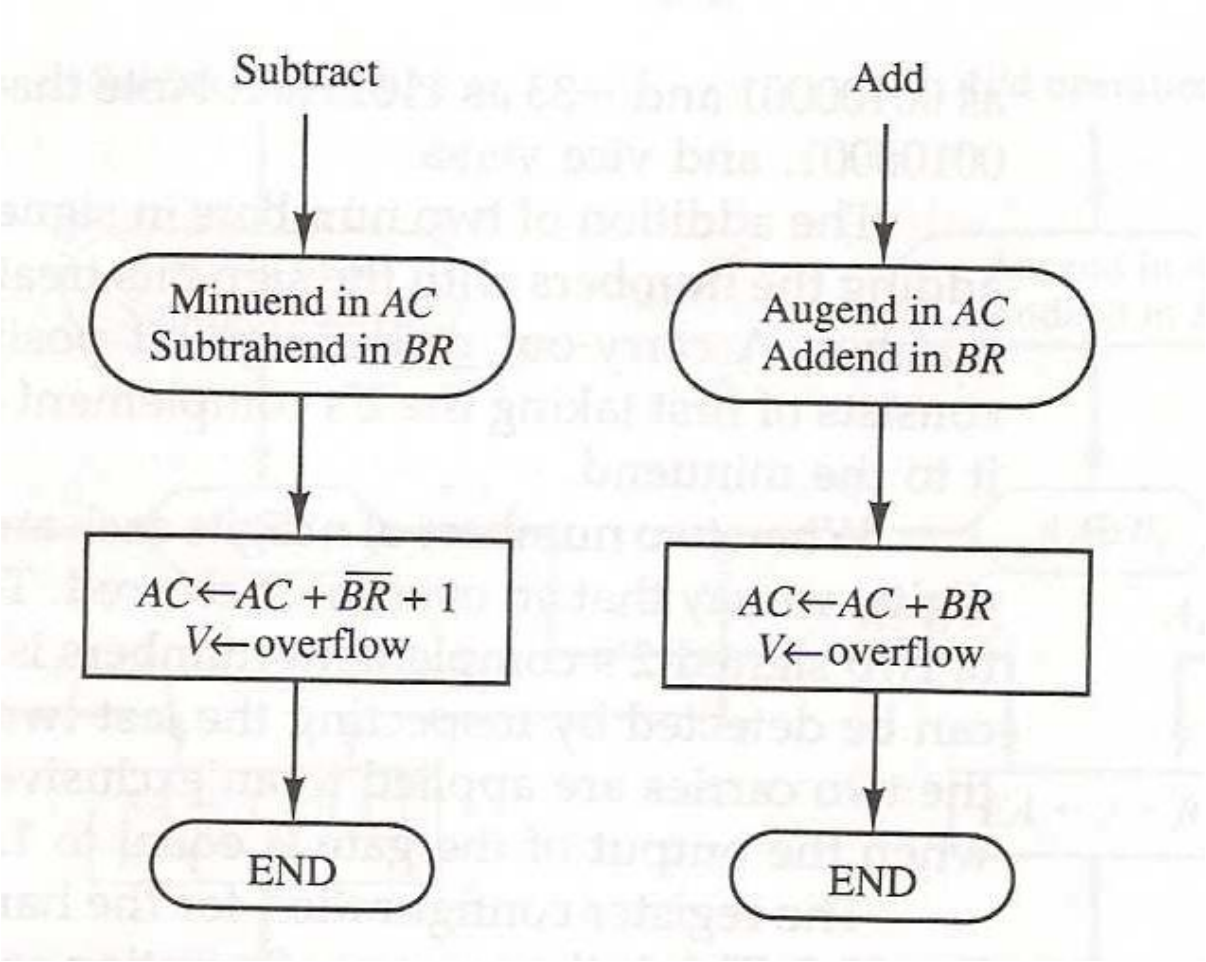



Subtract operation
Minuend in A
Subtrahend in B
$A_s \oplus B_s$
= 0
= 1
$A_s = B_s$
$A_s \neq B_s$
$EA \leftarrow A + \overline{B} + 1$
$AVF \leftarrow 0$
E
= 0
= 1
$A < B$
$A \geqslant B$
$A \leftarrow \overline{A}$
$A \leftarrow A + 1$
$A_s \leftarrow \overline{A}_s$
A
$\neq 0$
= 0
$A_s \leftarrow 0$
Add operation
Augend in A
Addend in B
$A_s \oplus B_s$
= 1
= 0
$A_s \neq B_s$
$A_s = B_s$
$EA \leftarrow A + B$
$AVF \leftarrow E$
END
(result is in A and A_s)

With signed-2's Complement data: Hardware implementation



Hardware Algorithm



Computer Arithmetic

Multiplication Algorithms

Dr. Mohammed Abdulridh Hussain

Introduction

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

Introduction

- The sign of the product is determined from the signs of the multiplicand and multiplier. If they are alike, the sign of the product is positive. If they are unlike, the sign of the product is negative.

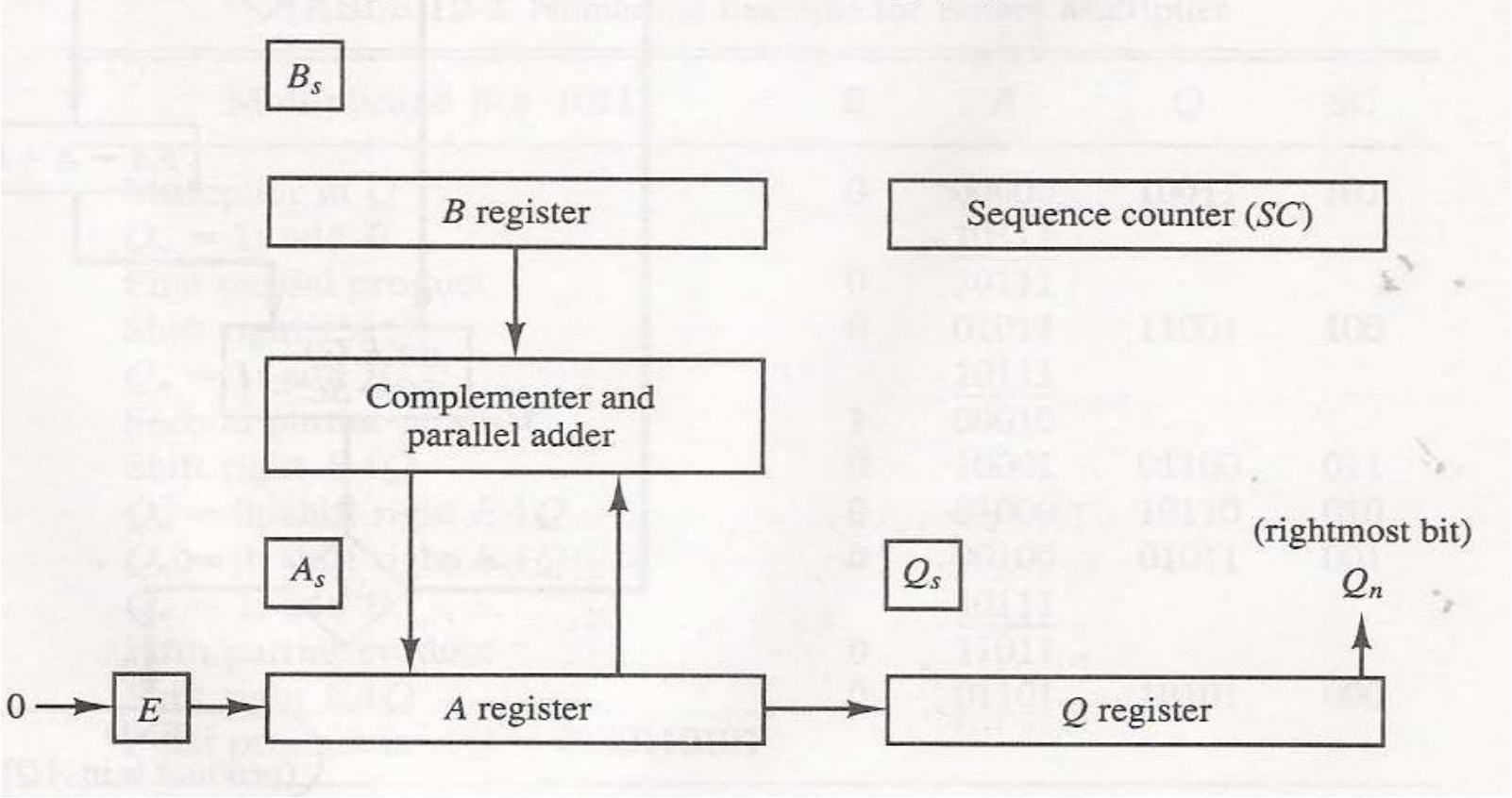
Hardware Implementation for Signed-Magnitude Data

- First, Instead of providing registers to store and add simultaneously as many binary numbers as there are bits in the multiplier, it is convenient to provide an adder for summation of only two binary numbers and successively accumulate the partial products in a register.

Hardware Implementation for Signed-Magnitude Data

- Second, instead of shifting the multiplicand to the left, the partial product is shifted to the right, which results in leaving the partial product and the multiplicand in the required relative positions.
- Third, when the corresponding bit of the multiplier is 0, there is no need to add all zeros to the partial product since it will not alter its value.

Hardware Implementation for Signed-Magnitude Data



Hardware Implementation for Signed-Magnitude Data

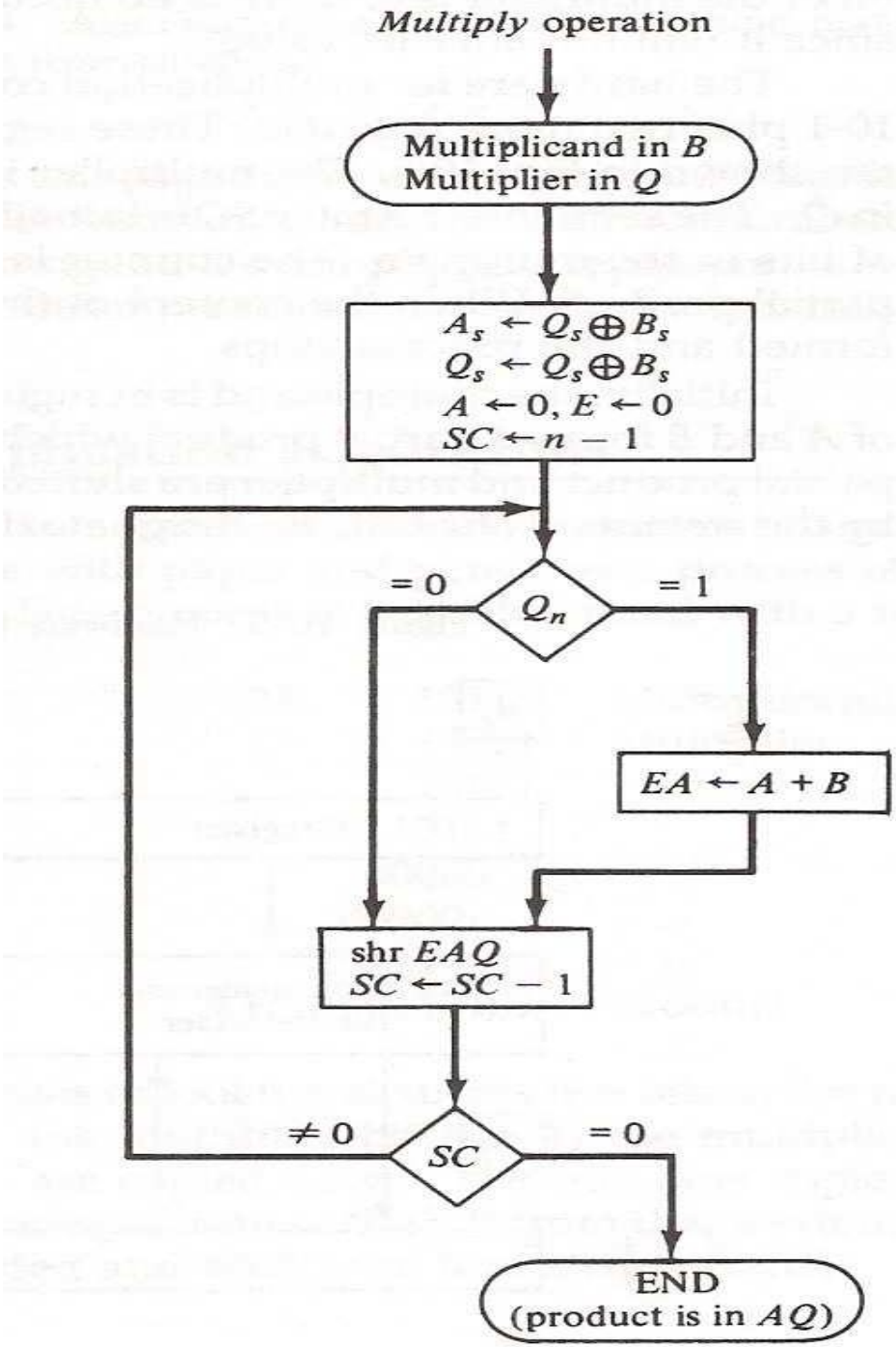
- Initially, the multiplicand is in register B and the multiplier in Q. The sum of A and B forms a partial product which is transferred to the EA register. Both partial product and multiplier are shifted to the right.

Hardware Algorithm

- The signs are compared, both A and Q are set to correspond to the sign of the product since a double-length product will be stored in registers A and Q.
- 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. Since an operand must be stored with its sign. One bit of the word will be occupied by the sign and the magnitude will consist of n-1 bits.

Hardware Algorithm

- After the initialization, the low-order bit of the multiplier in Q_n is tested. If it is a 1, the multiplicand in B is added to the present partial product in A. If it is a 0, nothing is done. Register EAQ is then shifted once to the right to form the new partial product. The sequence counter is decremented by 1 and its new value checked.
- If it is not equal to zero, the process is repeated an a new partial product is formed. The process stops when SC = 0.


Multiply operation
Multiplicand in B
Multiplier in Q
$A_s \leftarrow Q_s \oplus B_s$
$Q_s \leftarrow Q_s \oplus B_s$
$A \leftarrow 0, E \leftarrow 0$
$SC \leftarrow n - 1$
Q_n
$= 0$
$= 1$
$EA \leftarrow A + B$
shr EAQ
$SC \leftarrow SC - 1$
SC
$\neq 0$
$= 0$
END
(product is in AQ)

Example

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 $AQ = 0110110101$				

Computer Arithmetic

Booth Multiplication Algorithm

Dr. Mohammed Abdulridha Hussain

Introduction

- 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
- 001110 (+14) has 2^4 to 2^1 (k = 4, m = 1).

Introduction

- $2^4 – 2^1 = 16 – 2 = 14$, therefore, the multiplication M x 14, where M is the multiplicand and 14 the multiplier, can be done as M x 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.

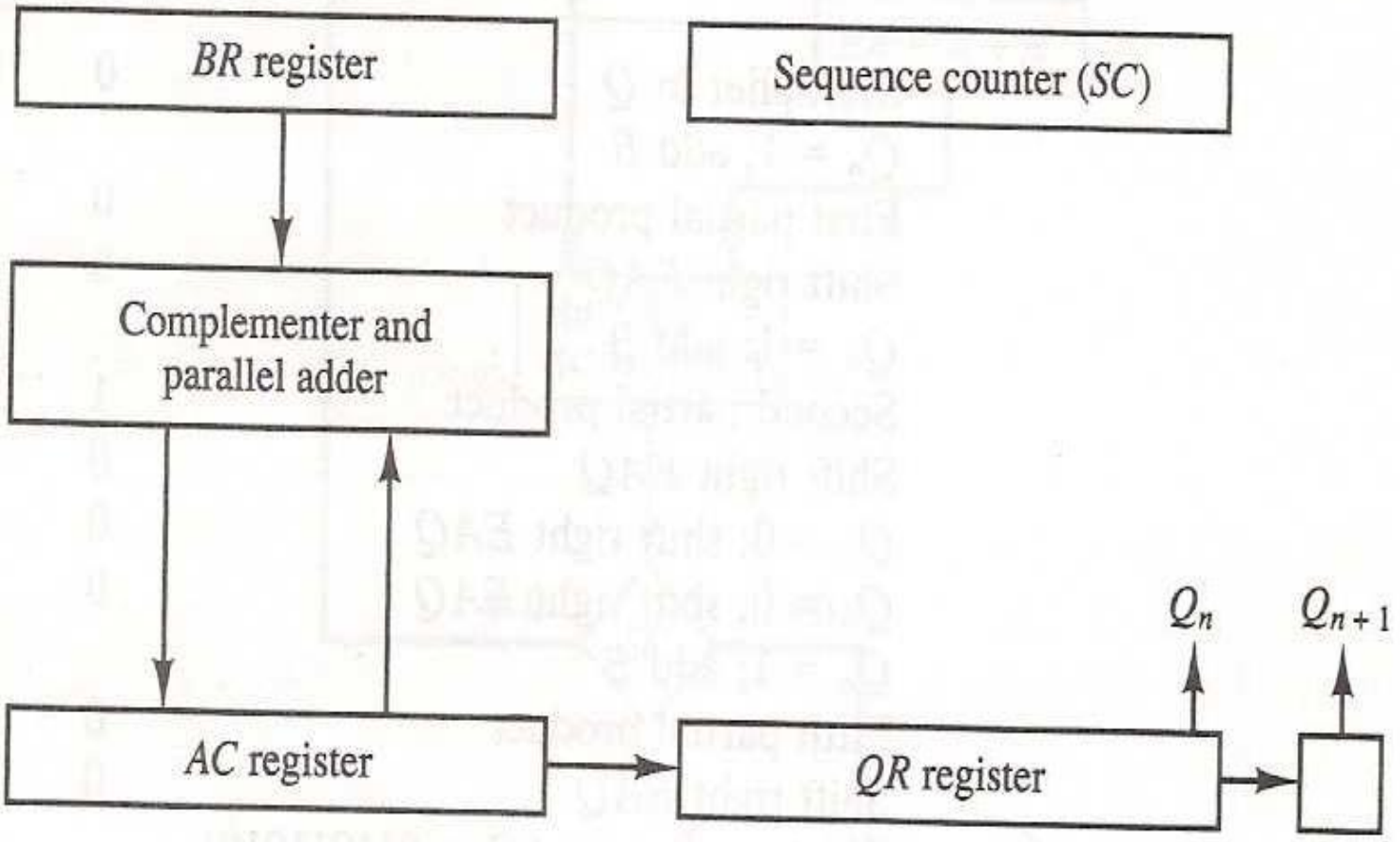
Introduction

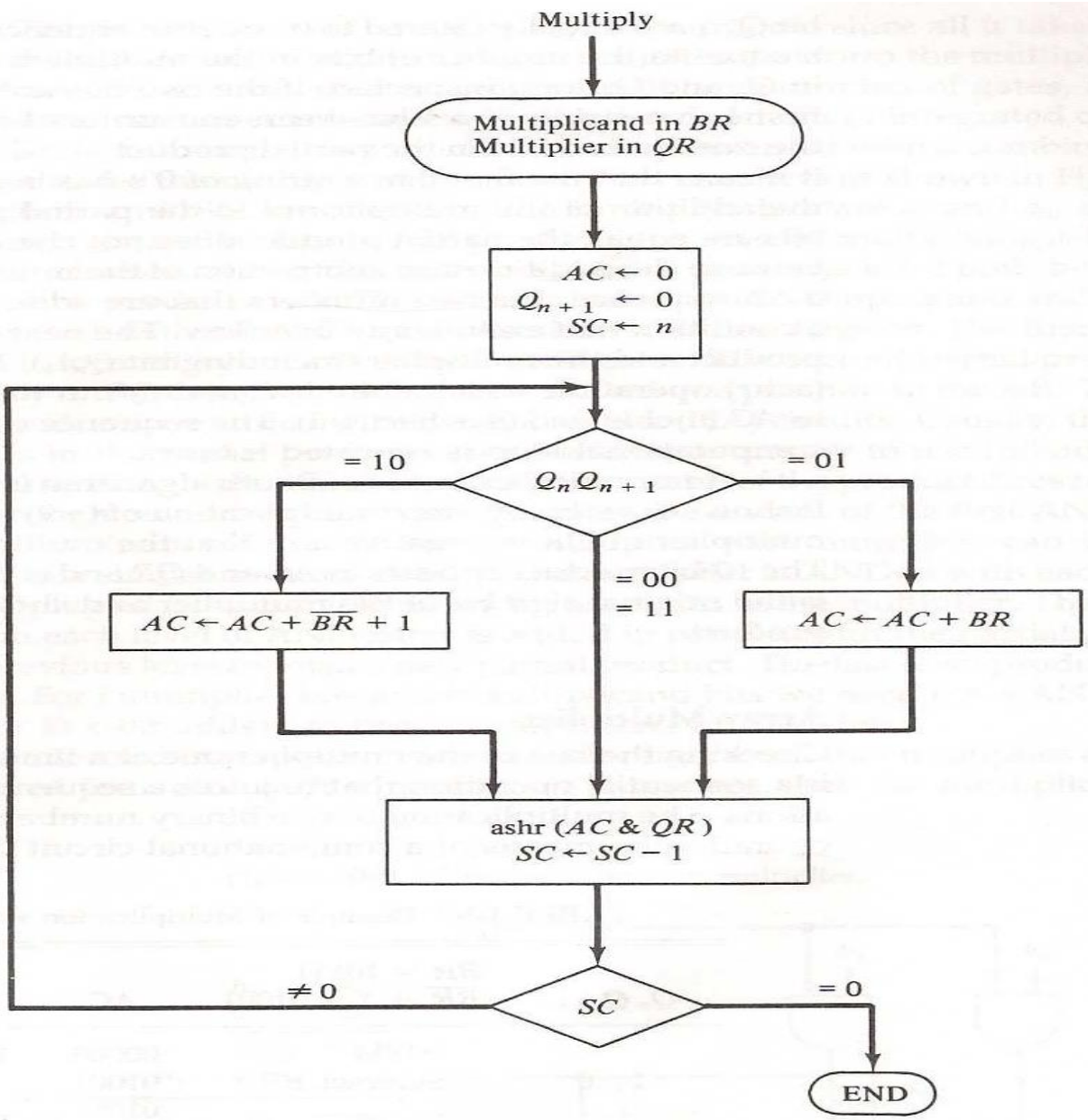
- Booth algorithm requires examination of the multiplier bits and shifting of the partial product. Prior to the shifting, the multiplicand may be added to the partial product, subtracted from the partial product, or left unchanged according to the following rules:

Introduction

- The multiplicand is subtracted from the partial product upon encountering the first least significant 1 in a string of 1’s in the multiplier.
- 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.
- The partial product does not change when the multiplier bit is identical to the previous multiplier bit.

Hardware


BR register
Sequence counter (SC)
Complementer and parallel adder
Q_n
Q_{n+1}
AC register
QR register


Multiply
Multiplicand in BR
Multiplier in QR
AC ← 0
Q_{n+1} ← 0
SC ← n
Q_n Q_{n+1}
= 10
= 01
= 00
= 11
AC ← AC + BR̄ + 1
AC ← AC + BR
ashr (AC & QR)
SC ← SC − 1
SC
≠ 0
= 0
END

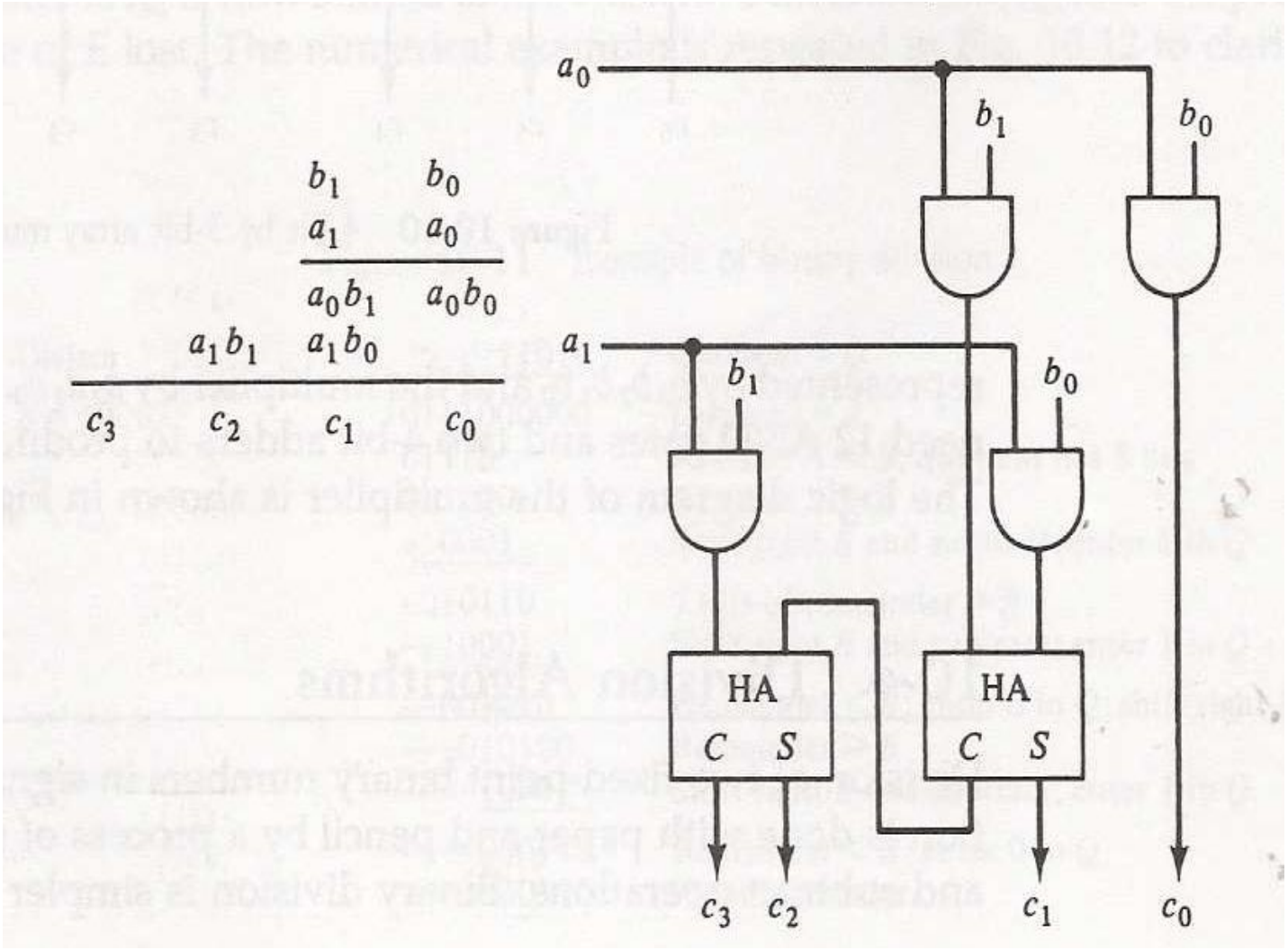
Algorithm

- If the two bits are equal to 10, it means that the first 1 in a string of 1's has been encountered. This is requires a subtraction of the multiplicand from the partial product in AC. If the two bits are equal to 01, it means that the first 0 in a string of 0's has been encountered. This requires the addition of the multiplicand to the partial product in AC. When the two bits are equal, the partial product does not change.

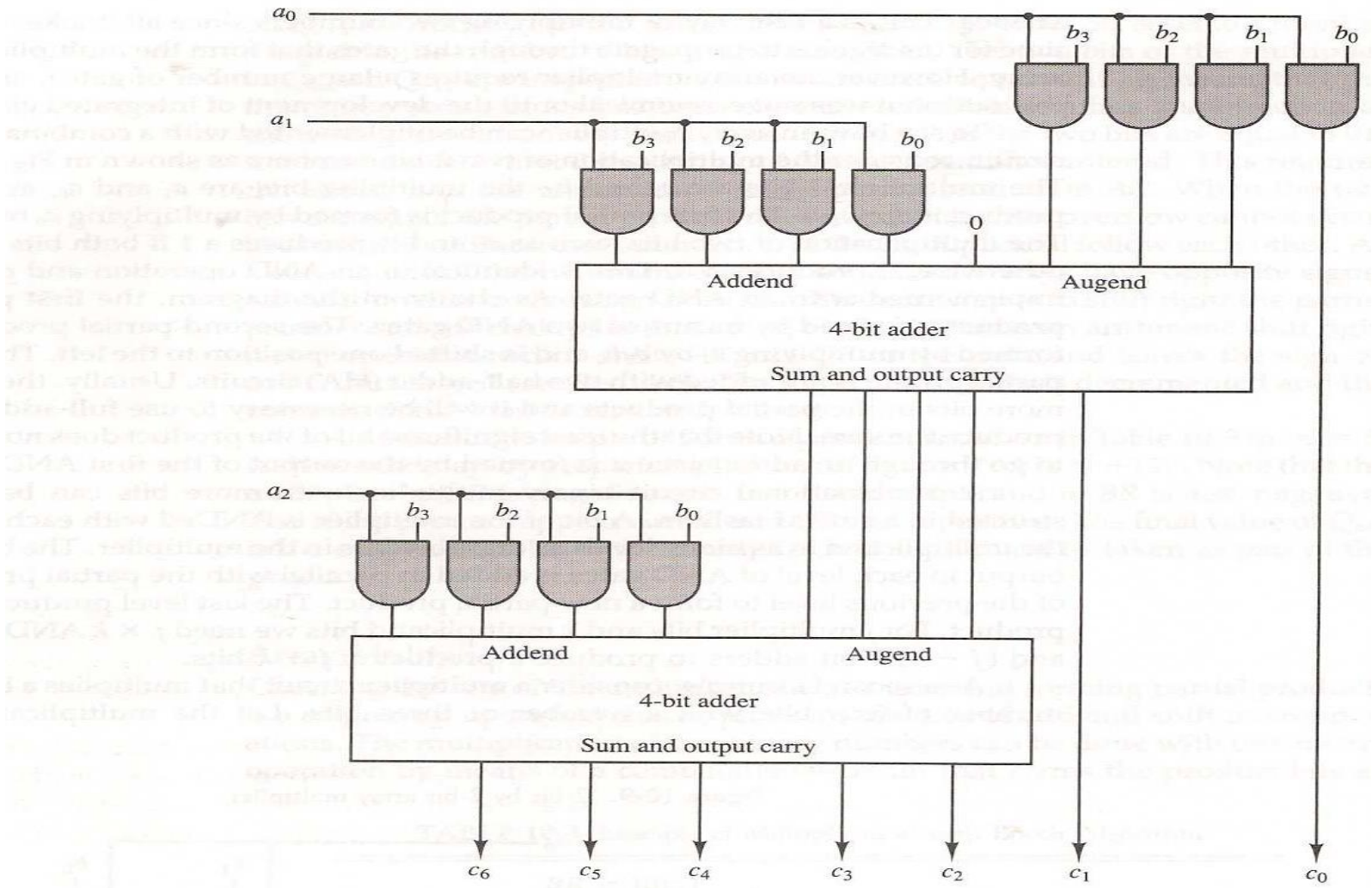
Example (-9) X (-13) = 117

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



4-bit by 3 bit array multiplier



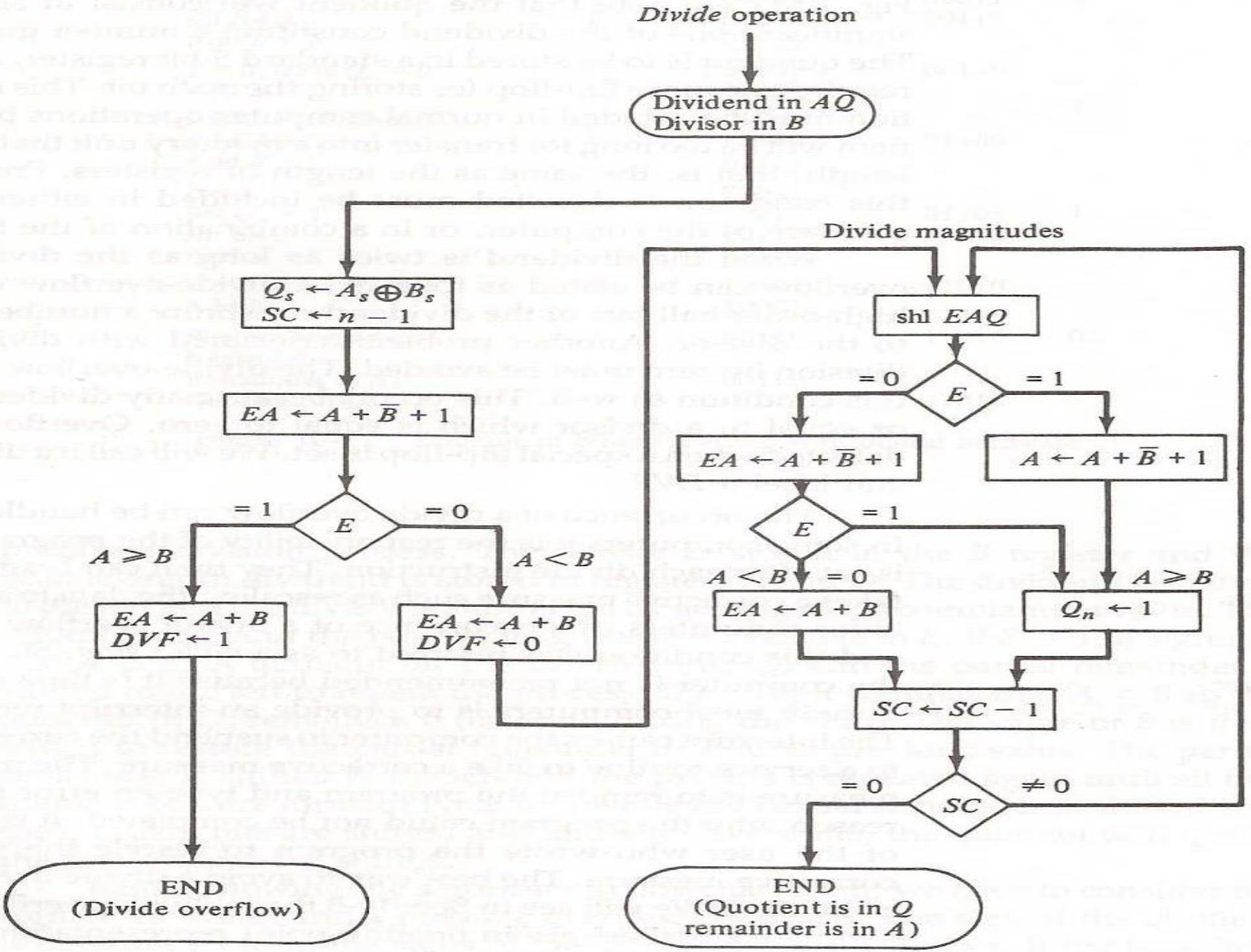
Computer Arithmetic

Division Algorithms

Dr. Mohammed Abdulridha Hussain

Introduction

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


Divide operation
Dividend in AQ
Divisor in B
$Q_s \leftarrow A_s \oplus B_s$
$SC \leftarrow n - 1$
$EA \leftarrow A + \overline{B} + 1$
E
= 1
= 0
$A \geqslant B$
$A < B$
$EA \leftarrow A + B$
$DVF \leftarrow 1$
$EA \leftarrow A + B$
$DVF \leftarrow 0$
END
(Divide overflow)
Divide magnitudes
shl EAQ
E
= 0
= 1
$EA \leftarrow A + \overline{B} + 1$
$A \leftarrow A + \overline{B} + 1$
E
= 1
$A < B$
= 0
$EA \leftarrow A + B$
$Q_n \leftarrow 1$
$A \geqslant B$
$SC \leftarrow SC - 1$
SC
= 0
$\neq 0$
END
(Quotient is in Q
remainder is in 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	

Examples

- By using Addition & Subtraction algorithms solves the following:

(13 + 9) , (-7 + 5), (5 – 2), (-7 – 5), (-2 – 3)

(-4 – (-6)) , (-5 + 4)

- Multiplication

(21 x 31)

- Booth multiplication

(15 x 13), (15 x -13)

- Division

(15 / 3) , (163/11)