Chapter 10.

Computer Arithmetic

- Arithmetic Instruction:

ADD, SUB, MUL, DIV.

- Arithmetic Processor:

Part of processor unit that executes arithmetic operation.

- Data type(decimal or binary) specified in Arithmetic instruction is of fixed point or floating point form.
- Fixed point → integer or fraction.
- Negative number → signed magnitude or unsigned magnitude.

Addition and Subtraction

- We consider ADD, SUB, MUL and DIV for the following type of data

1. Fixed point binary data in signed magnitude representation.
2. Fixed point binary data in signed 2’s complement representation.
3. Floating point binary data
4. Binary code decimal data.

ADD and SUB with signed-magnitude data

- In this section we develop the addition and subtraction algorithm for data representation in signed magnitude and signed 2's complement.

- Procedure for writing +ve number using Signed magnitude:

Leftmost position bit is used as a sign bit.

If sign bit =0 , represent positive integer binary number

If sign Bit = 1, represent negative integer binary number.

In addition to sign bit, number may have binary(or decimal) point.

Position of the binary point is needed to represent fractions, integers , or mixed integer- fraction number.

- Based on the position of binary point in register it is characterize in two ways:

1. Fixed point: If the position of binary point is fixed in one position.

A) Extreme leftmost position to make the stored number a fraction,

B) Extreme rightmost position to make stored number a integer.

Floating point representation used second register to store a number that designates the position of the decimal point in the first register.

- Signed magnitude representation of a negative number consist of magnitude and negative sign.
- Ex: Consider a signed number 14 stored in a 8-bit register.

  +14 → 00001110 , 0 leftmost bit represent sign bit.

  There is 3 ways to represent – 14 with 8-bit:

In signed-magnitude representation	1 0001110
In signed-1's complement representation	1 1110001
In signed-2's complement representation	1 1110010

- Rules for addition:

Follows the common rule of arithmetic addition.

1. If sign are same +6 + +13 = +19 , we add magnitudes and give the common sign.
2. If sign are different -6 + +13 = +7 , we subtract the smaller magnitude from the larger one and give the result the sign of larger magnitude.

This process requires Comparison of signs and magnitude and then performing addition or subtraction.

Rules for addition in signed 2's complement:

Process required only addition and complementation.

1. Positive number is written same as its binary equivalent form.
2. For writing Negative number:

   step 1: Write its positive number in its equivalent binary representation

   step 2: Take its 2's complement.

3. Add two number including their sign bits and discard any carry out of the sign bit.

Note: if sum obtained after addition is negative, it is in 2's complement form.

+6	00000110	−6	11111010
+13	00001101	+13	00001101
+19	00010011	+7	00000111
+6	00000110	−6	11111010
−13	11110011	−13	11110011
−7	11111001	−19	11101101

- Rules for Subtraction:

Steps→

1. Takes the 2's complement of the subtrahend (include the sign bit),
2. Add it to minuend (include sign bit)
3. Discard the carry out bit from the sign bit.

Overflow: If two number with n digits each added and resulted sum occupies n+1 digits we can say overflow occurs.

Overflow is a problem in digital computer because of the finite width of the register.

Occurrence of Overflow:

1. When both numbers added are of same sign.

carries: 0 1		carries: 1 0	
+70	0 1000110	−70	1 0111010
+80	0 1010000	−80	1 0110000
+150	1 0010110	−150	0 1101010

- Overflow condition detection:

It is detected by observing the carry into the sign bit position and carry out of the sig bit position.

If the carry in != carry out bit ; overflow condition is occurred.

Following table listed some of the operation.

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

- To implement the SUB and Add arithmetic operation we required:

1. Two register to hold the magnitude of the number. Let Reg. A and Reg. B.
2. As and Bs be two flip flop that hold the sign bit.
3. Third register to save the result. We can save the space by using the destination register same as source register. Here, As and A itself.
4. Parallel- adder is need to perform the microoperation A+B.
5. Comparator ckt is need to establish If A>B, A<B or A=B.
6. Two parallel- subtraction ckt is needed to perform A-B and B-A.
7. XOR gate to obtain sign relationship using As and Bs bit.

Hardware Implementation of Add and SUB

AVF: add-overflow flipflop holds the overflow bit when A and B are added
E: Flip-flop hold the output carry(used to determine the relative magnitude of the two numbers)
Mode M= 0, A + B operation be perform with input carry =0
M = 1, A+ B' +1 operation be perform with input carry = 1

Complementor used to perform 1's complement(XOR-gate).

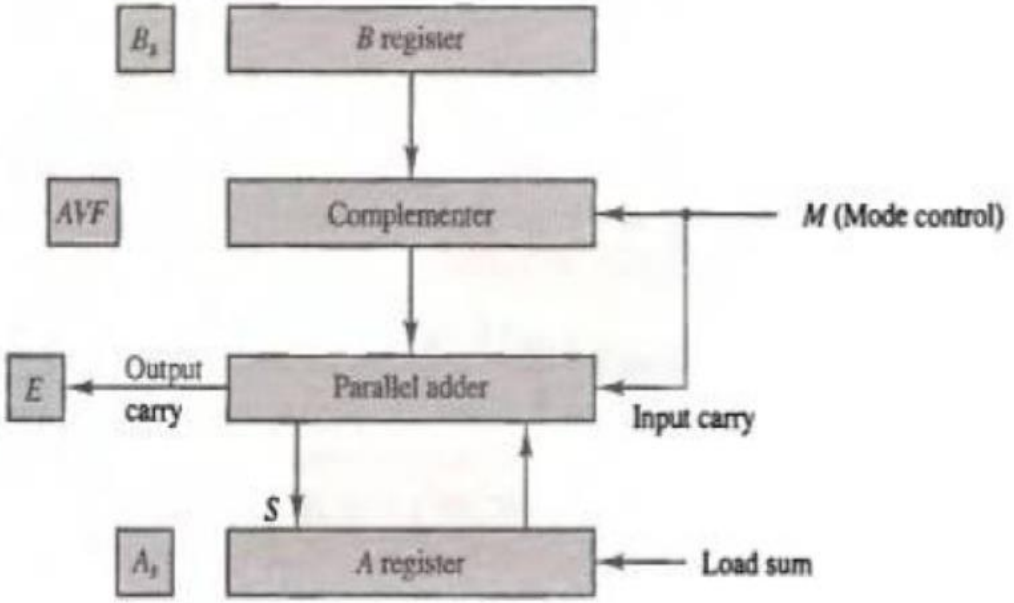


Figure 10-1 Hardware for signed-magnitude addition and subtraction.

As XOR Bs = 0, sign are identical

As XOR Bs = 1,

For add: Identical sign indicates magnitude be added

For Sub: different sign indicates magnitude be added

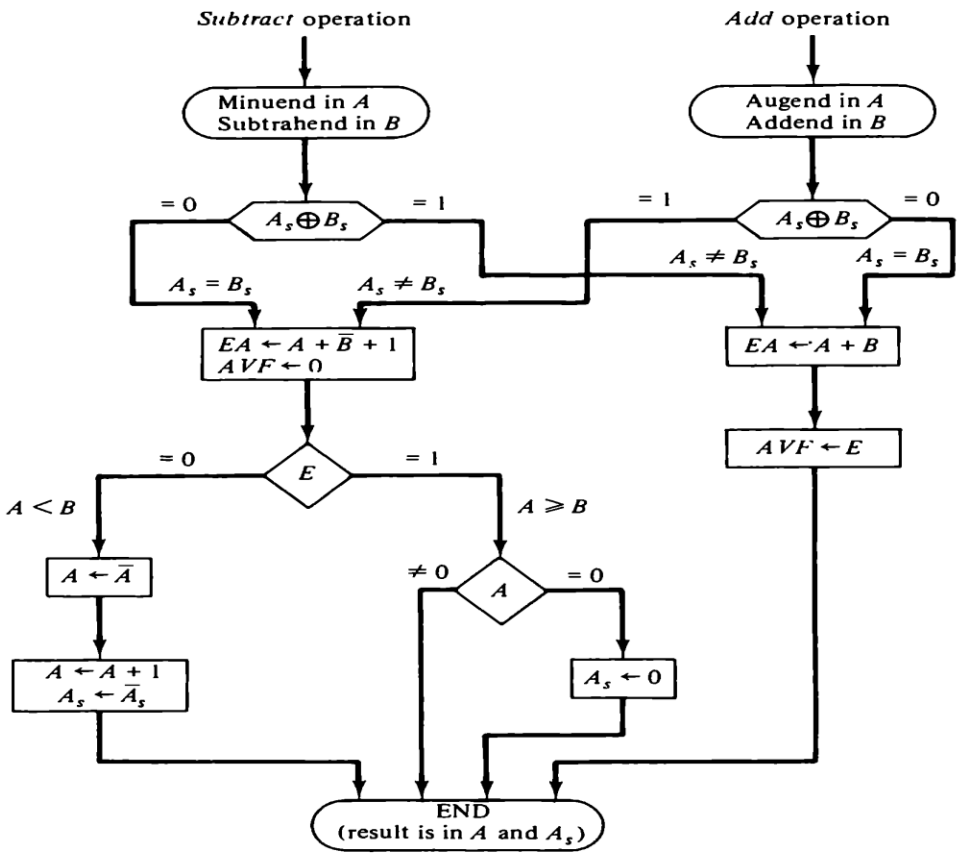


Figure 10-2 Flowchart for add and subtract operations.

Multiplication algorithm:

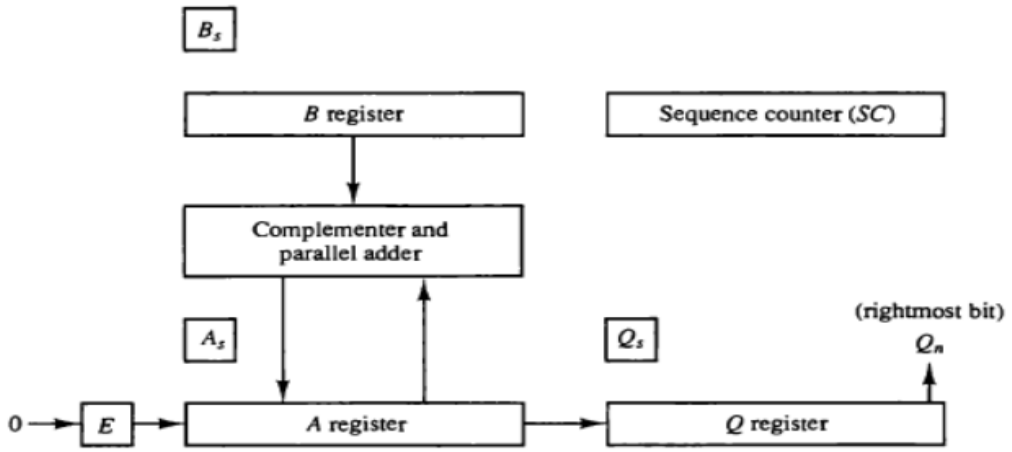
- Multiplication of 2 signed magnitude fixed binary number

Process involve→ successive shift + add operation.

For H/W implementation :

1. Adder for doing addition of 2 binary number.
2. Register Q to hold the multiplier and Qs to hold sign bit.
3. SC successive counter , initially set to number equals to number of bits in multiplier and it is decremented by 1 after every successive partial product.
4. Register B holds the Multiplicand.
5. Initially set A←0, E←0.
6. Register EA ← A+B (partial product).

Figure 10-5 Hardware for multiply operation.



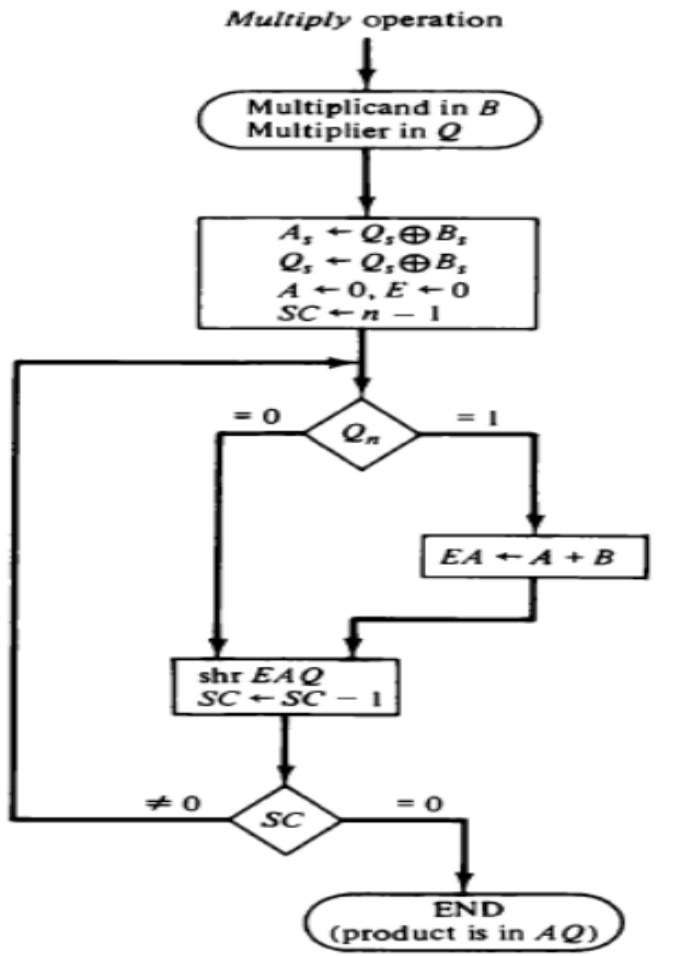
23	10111		Multiplicand
19	× 10011		Multiplier
	10111		
	10111		
	00000	+	
	00000		
	10111		
437	110110101		Product

Multiplicand → B,
Multiplier →Q,
SC holds the value equal to n-1 because 1 bit (MSB) of Q is occupied by sign bit.

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 $AQ = 0110110101$				

Figure 10-6 Flowchart for multiply operation.



Booth Multiplication Algorithm

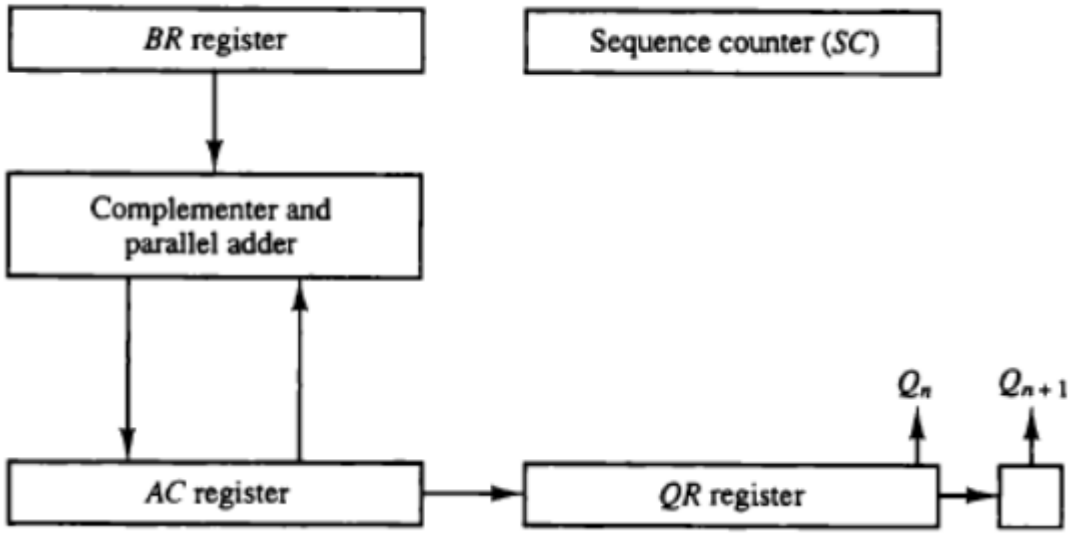
It works on binary integer in signed 2's complement form.

It also examine multiplier bit and do the shifting. But prior to shifting, multiplicand added to partial product or subtracted to partial product or remain unchanged be decided by the following rules:

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.

Register AC, BR, QR (sign bit define in register itself)
Qn+1 appended to QR to facilitate the double bit inspection.

Figure 10-7 Hardware for Booth algorithm.



Flow chart of Booth Algo.

-9 x -13 = + 117 (0001110101)

+9 → Binary representation is 01001

2's complement is 10111 = -9

+13→ Binary representation is 01101

2's complement is 10011 = -13

BR = 10111, QR=10011

1←Qn, Initially Qn+1 =0.

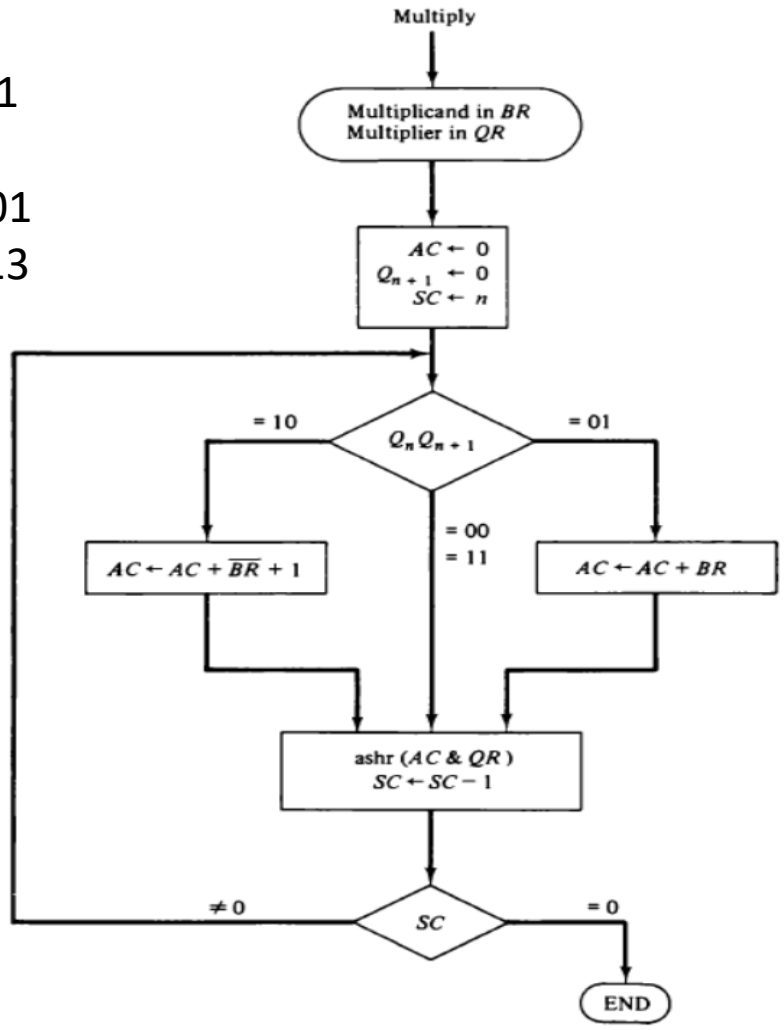


Figure 10-8 Booth algorithm for multiplication of signed-2's complement numbers.

Figure 10-7 Hardware for Booth algorithm.

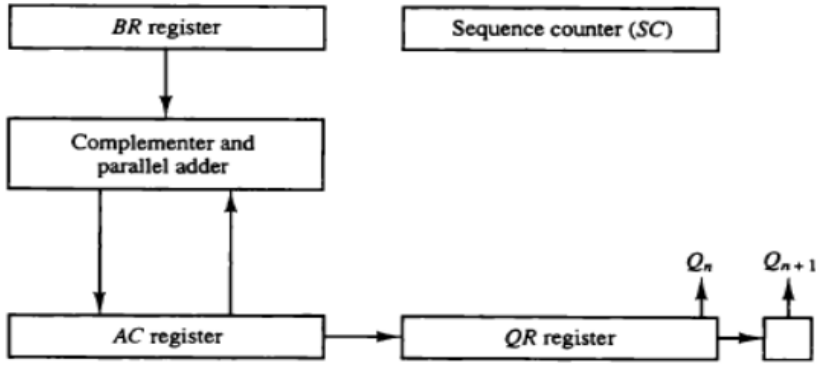


TABLE 10-3 Example of Multiplication with Booth Algorithm

Q_nQ_{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

- Use combinational circuits.
- Fastest way of multiplication as it use gates that form the multiplication array.
- Not economical because use large number of gates.
- j → multiplier bit, k multiplicand bit requires j x k AND gates and (j-1)k bit adder to produce a product of j+k bits.

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

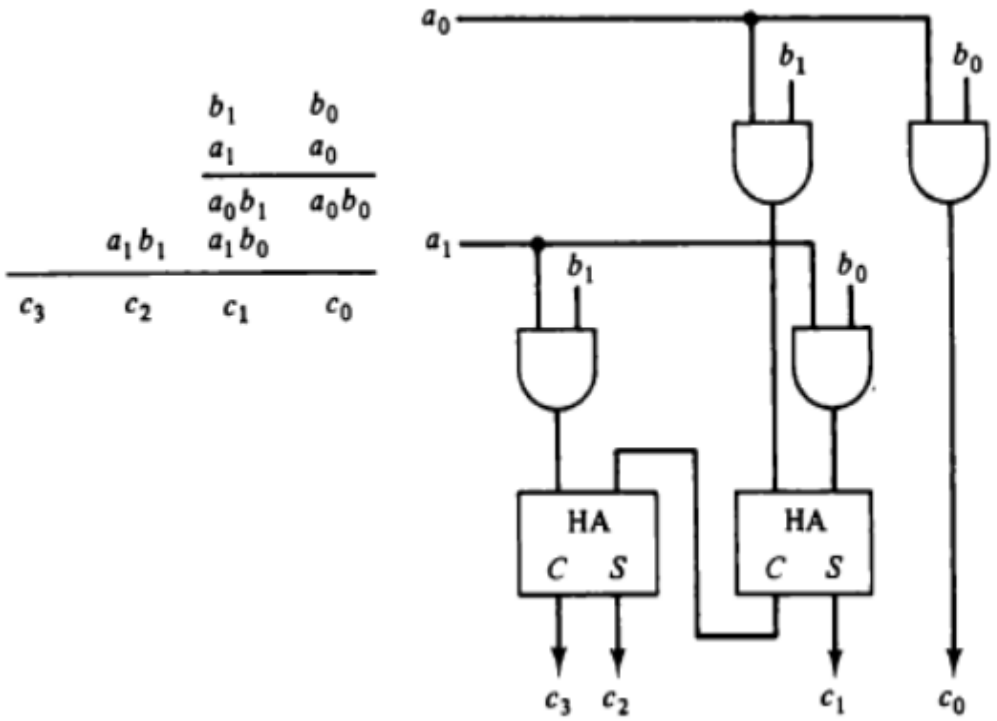


Fig. shows the multiplication of 3 bit number
With 4 bit number.
J = 3, k=4

- AND gate = j x k = 3 x 4 = 12
- Two 4-bit adder used
- Product = j + k = 3 + 4 = 7
  7 bit product. (C0 – C6)

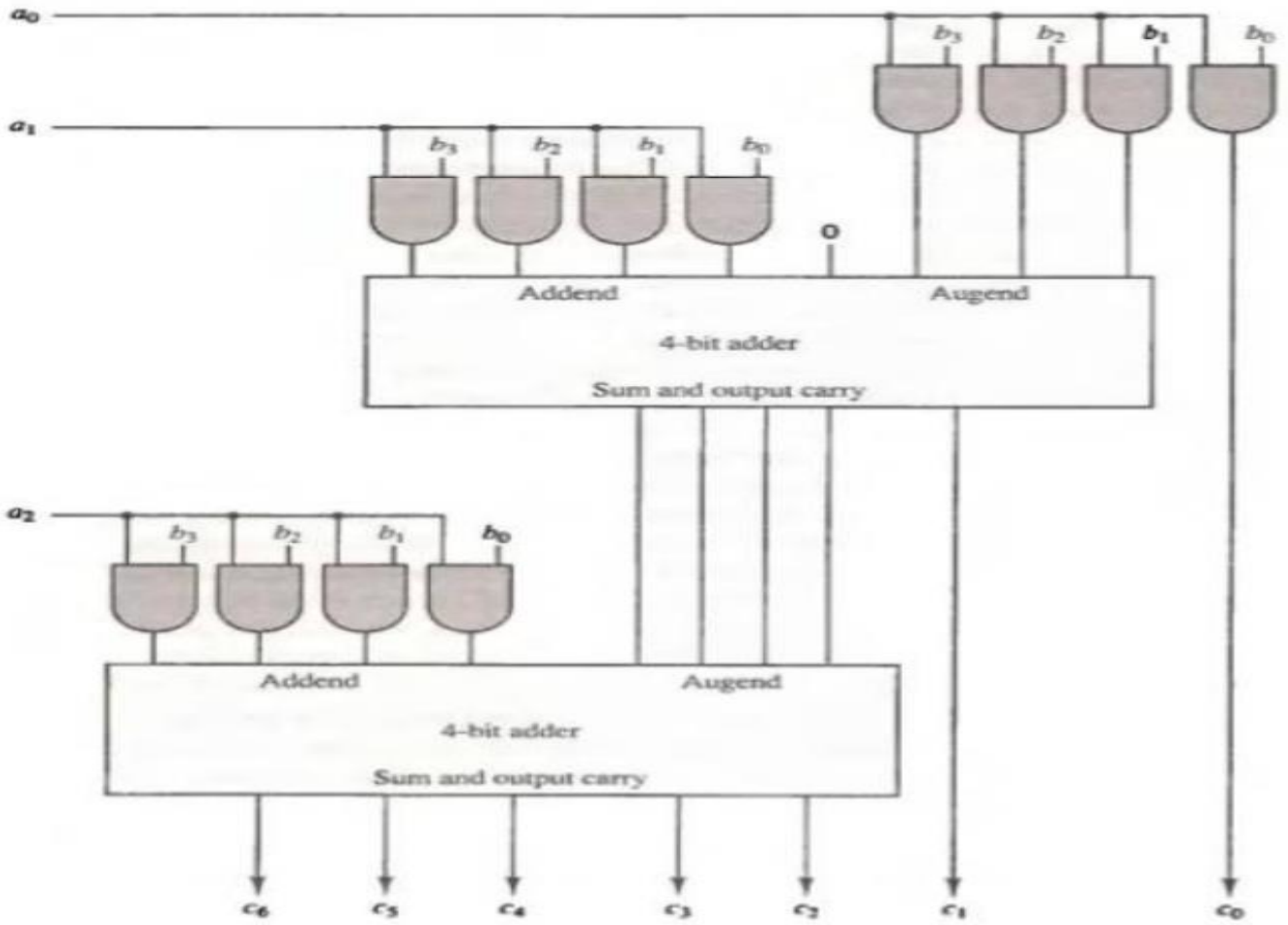


Figure 10-10 4-bit by 3-bit array multiplier.

Division Algorithm

In example below Divisor B consist of 5 bit and dividend A consist of 10 bit.

- 5-MSB bit of dividend compare with the Divisor.
- If 5 bit number of A< B then compare 6-MSB of A with B.
- If 6 bit MSB number of A > B , put 1 in quotient bit at sixth position above dividend and shift the divisor one position right and subtract it from the dividend.
- The resultant difference output is called partial reminder.
- Compare partial reminder with B.
- If reminder>= B, shift right the B and subtract it from reminder and put 1 in quotient
- If reminder < B, shift right B and put 0 to Q.

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

10001 = 17
01110= 14
011100=28
010110=22
001010=10
010100= 20
0110= 6

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

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

Hardware Implementation

- B stores the divisor, AQ used to store the double length dividend.
- E provides the relative magnitude.
- Shift divisor and partial product to left.
- Subtraction is achieved by adding A to 2's complement of B i.e. A-B= A+B'+1.
- Divide Overflow : May result me quotient overflow.

Condition of overflow→

1. If half bit MSB of dividend >= divisor.
2. If dividend / 0; it also check this condition.
3. Overflow condition is usually detected by special flip-flop known as divide-overflow flip flop (DVF).

Occurrence of overflow be handle in following ways:

1. To set check of DVF after every division instruction by programmer and branch to sub routine that take some corrective measures of data to avoid overflow.
2. To provide interrupt request when DVF is set which suspend the current program and branch to service routine for corrective measure.
3. Best way to use floating point data.

Figure 10-13 Flowchart for divide operation.