Chapter 6

Computer Arithmetic

Contents

Data Types

Data Types:

- Binary information is stored in ***memory*** or ***processor registers***
- Registers contain either ***data*** or ***control information***
  - ***Data*** are numbers and other binary-coded information
  - ***Control information*** is a bit or a group of bits used to specify the sequence of command signals
- Data types found in the registers of digital computers
  - ***Numbers*** used in arithmetic computations
  - ***Letters*** of the alphabet used in data processing
  - ***Other discrete symbols*** used for specific purpose
- The binary number system is the most natural system to use in a digital computer
- Number Systems
  - ***Base*** or ***Radix r system*** : uses distinct symbols for ***r digits***
  - Most common number system :Decimal, Binary, Octal, Hexadecimal
  - Positional-value(weight) System : $\mathbf{r^2\ r^1 r^0 . r^{-1}\ r^{-2}\ r^{-3}}$
    - Multiply each digit by an integer power of r and then form the sum of all weighted digits

Data Types

- Decimal System/Base-10 System
  - Composed of 10 symbols or numerals(0, 1, 2, 3, 4, 5, 6, 7, 8, 9)
- Binary System/Base-2 System
  - Composed of 2 symbols or numerals(0, 1)
  - Bit = Binary digit
- Octal System/Base-8 System
  - Composed of 8 symbols or numerals(0, 1, 2, 3, 4, 5, 6, 7)
  - Bit = Binary digit
- Hexadecimal System/Base-16 System :
  - Composed of 16 symbols or numerals(0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F)
- Binary-to-Decimal Conversions

$1011.101_2 = (1 \times 2^3) + (0 \times 2^2) + (1 \times 2^1) + (1 \times 2^o) + (1 \times 2^{-1}) + (0 \times 2^{-2}) + (1 \times 2^{-3})$

$= 8_{10} + 0 + 2_{10} + 1_{10} + 0.5_{10} + 0 + 0.125_{10}$

$= 11.625_{10}$

- Octal-to-Decimal Conversions

$(736.4)_8 = 7 \times 8^2 + 3 \times 8^1 + 6 \times 8^0 + 4 \times 8^{-1}$

$= 7 \times 64 + 3 \times 8 + 6 \times 1 + 4/8 = (478.5)_{10}$

- Hexadecimal-to-Decimal Conversions

$(F3)_{16} = F \times 16 + 3 = 15 \times 16 + 3 = (243)_{10}$

Data Types

- Conversion of decimal 41.6875 into binary

  Repeated division

Integer = 41

41 / 2 = 20 remainder 1

20 / 2 = 10 remainder 0

10 / 2 = 5 remainder 0

5 / 2 = 2 remainder 1

2 / 2 = 1 remainder 0

1 / 2 = 0 remainder 1

(binary number will end with 1) : LSB

(binary number will start with 1) : MSB

Read the result upward to give an answer of $(41)_{10} = (101001)_2$

Fraction = 0.6875

0.6875 x 2 = 1.3750 integer 1

1.3750 x 2 = 0.7500 integer 0

0.7500 x 2 = 1.5000 integer 1

1.5000 x 2 = 1.0000 integer 1

MSB

LSB

(ignore the integral part and multiply only the fractional part until you get 0's in fractional part)

Read the result downward $(0.6875)_{10} = (0.1011)_2$

$(41.6875)_{10} = (101001.1011)_2$

Data Types

Hex-to-Decimal Conversion

$2AF_{16} = (2 \times 16^2) + (10 \times 16^1) + (15 \times 16^o)$

$= 512_{10} + 160_{10} + 15_{10}$

$= 687_{10}$

Decimal-to-Hex Conversion

423_{10} / 16 = 26 remainder 7 (Hex number will end with 7) : **LSB**

26_{10} / 16 = 1 remainder 10

1_{10} / 16 = 0 remainder 1 (Hex number will start with 1) : **MSB**

Read the result upward to give an answer of $423_{10} = 1A7_{16}$

Hex	Binary	Decimal
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
A	1010	10
B	1011	11
C	1100	12
D	1101	13
E	1110	14
F	1111	15
14	0001 0100	20
F8	1111 1000	248

Hex-to-Binary Conversion

$$9F2_{16} = \begin{array}{ccc} 9 & F & 2 \\ \downarrow & \downarrow & \downarrow \end{array}$$

$= 1001 \quad 1111 \quad 0010$

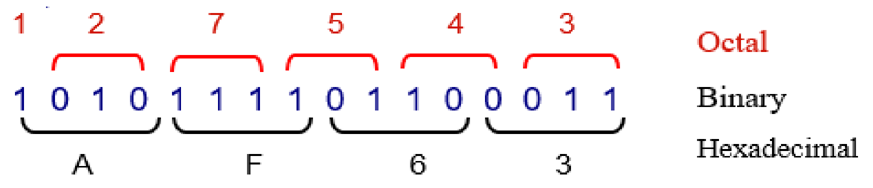
$= 100111110010_2$

Binary-to-Hex Conversion

$$1\,1\,1\,0\,1\,0\,0\,1\,1\,0_2 = \underbrace{0\,0\,1\,1}_{3}\ \underbrace{1\,0\,1\,0}_{A}\ \underbrace{0\,1\,1\,0}_{6}$$

$= 3A6_{16}$

Binary, octal, and hexadecimal Conversion



Data Types

- Binary-Coded-Decimal Code
  - Each digit of a decimal number is represented by its binary equivalent

$$\begin{tikzcd}
8 \arrow[d] & 7 \arrow[d] & 4 \arrow[d] & \textbf{(Decimal)} \\
1000 & 0111 & 0100 & \textbf{(BCD)}
\end{tikzcd}$$

  - Only the four bit binary numbers from 0000 through 1001 are used
  - Comparison of BCD and Binary

$137_{10} = 10001001_2$ **(Binary) -** *require 8 bits*

$137_{10} = 0001\ 0011\ 0111_{BCD}$ **(BCD) -** *require 12 bits*

- Alphanumeric Representation
  - Alphanumeric character set
    - 10 decimal digits, 26 letters, special character($, +, =,….)
    - ASCII(American Standard Code for Information Interchange)
    - Standard alphanumeric binary code uses seven bits to code 128 characters

ASCII Table

Character	Binary code	Character	Binary code	Character	Binary code	Character	Binary code
A	100 0001	U	101 0101	0	011 0000	/	010 1111
B	100 0010	V	101 0110	1	011 0001	,	010 1100
C	100 0011	W	101 0111	2	011 0010	=	011 1101
D	100 0100	X	101 1000	3	011 0011		
E	100 0101	Z	101 1010	4	011 0100		
F	100 0110			5	011 0101		
G	100 0111			6	011 0110		
H	100 1000			7	011 0111		
I	100 1001			8	011 1000		
J	100 1010			9	011 1001		
K	100 1011						
L	100 1100						
M	100 1101			space	010 0000		
N	100 1110			.	010 1110		
O	100 1111			(	010 1000		
P	101 0000			+	010 1011		
Q	101 0001			$	010 0100		
R	101 0010			*	010 1010		
S	101 0011			)	010 1001		
T	101 0100			-	010 1101		

Complements

- ***Complements*** are used in digital computers for simplifying the ***subtraction operation*** and for logical manipulation
- There are two types of complements for base r system
  - 1) r's complement 2) (r-1)'s complement
    - Binary number : 2's or 1's complement
    - Decimal number : 10's or 9's complement
  - (r-1)'s Complement
    - (r-1)'s Complement of N = $(r^n-1)-N$
      - 9's complement of N=***546700***

        (10^6-1)-546700= (1000000-1)-546700= 999999-546700

        = ***453299***
      - 1's complement of N=***101101***

        (2^6-1)-101101= (1000000-1)-101101= 111111-101101

        = ***010010***

N : given number
r : base
n : no of digits in the given number

546700(N) + 453299(9's com)
=999999

101101(N) + 010010(1's com)
=111111

  - r's Complement
    - r's Complement of N = r^n-N for N != 0 and 0 for N=0
      - 10's complement of ***2389***= $(10^4$ -2389)= (10000-2389) =***7611***
      - 2's complement of ***1101100***= 0010011+1= ***0010100***

* ***r's Complement***
(r-1)'s Complement +1 =(r^n-1)-N+1= r^n-N

Subtraction of Unsigned Numbers

- Subtraction of Unsigned Numbers
  - 1) M + (r^n-N)
  - 2) M $\geq$ N : Discard end carry, Result = M-N
  - 3) M < N : No end carry, Result = - r's complement of (N-M)

M $\geq$ N

- ***Decimal Example***

72532(M) - 13250(N) = 59282

72532

Discard End Carry

+ 86750 (10's complement of 13250)

1 59282

Result = ***59282***

X $\geq$ Y

- ***Binary Example***

1010100(X) - 1000011(Y) = 0010001

1010100

Discard End Carry

+ 0111101 (2's complement of 1000011)

1 0010001

Result = ***0010001***

M < N ***13250(M) - 72532(N) = -59282***

13250

+ 27468 (10's complement of 72532)

No End Carry

0 40718

Result = -(10's complement of 40718)

= -(59281+1) = ***-59282***

X < Y ***1000011(X) - 1010100(Y) = -0010001***

1000011

+ 0101100 (2's complement of 1010100)

No End Carry

0 1101111

Result = -(2's complement of 1101111)

= -(0010000+1) = ***-0010001***

Practice Examples

1. Convert the following binary numbers to decimal :
   a. 101110 b. 1110101 c. 110110100

2. Convert the following numbers with the indicated bases to decimal :
   b. $(12121)_3$ b. $(4310)_5$ c. $(50)_7$

3. Obtain 9's & 10's complement of the following eight-digit decimal numbers:
   c. 12349876 b. 00980100 c. 90009951

4. Perform the subtraction with the following unsigned decimal numbers by taking the 10's complement of the subtrahend
   d. 5250-1321
   e. 1753-8640
   f. 20-100
   g. 1200-250

5. Perform the subtraction with the following unsigned binary numbers by taking the 2's complement of the subtrahend
   h. 11010-10000
   i. 11010-1101
   j. 100-110000
   k. 1010100-1010100

Thank You.

Addition ad Subtraction

- All the arithmetic operations can be performed by addition,subtraction,multiplication and division
- Addition and Subtraction operations are performed on
  - Signed magnitude data: 1 bit is used to represent the sign and other bits represent the magnitude.
  - 2'complement data

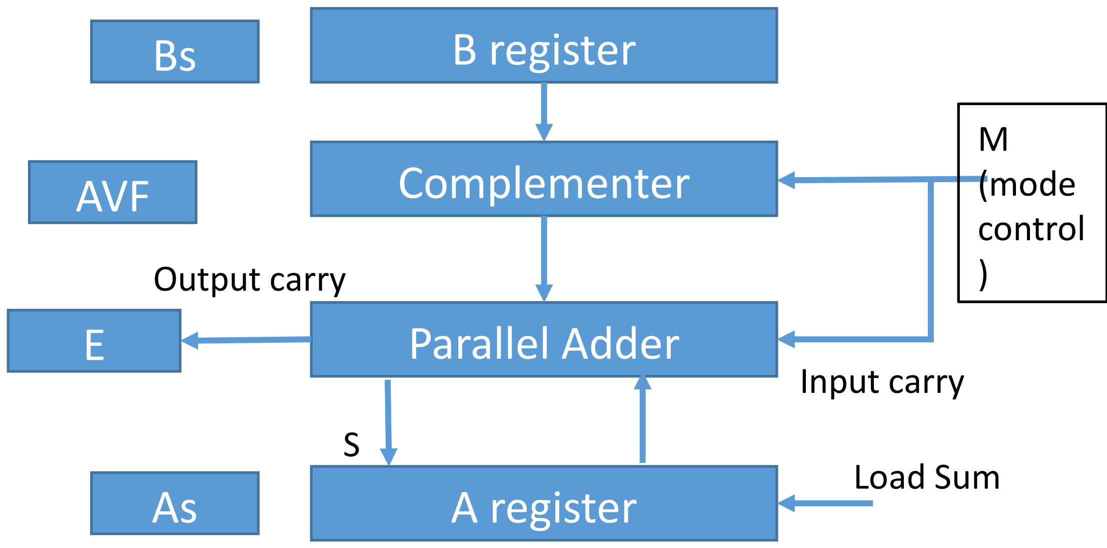
Addition and Subtraction with Signed-Magnitude Data

- Addition Algorithm:
  - When the signs of A and B are identical ,add the 2 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 number from larger.
- Subtraction Algorithm:
  - When the signs of A and B are different ,add the 2 magnitudes and attach the sign of A to the result
  - When the signs of A and B are identical, compare the magnitudes and subtract the smaller number from larger.
- Choose the sign of the result to be same as A if A>B or complement of A if A<B

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)

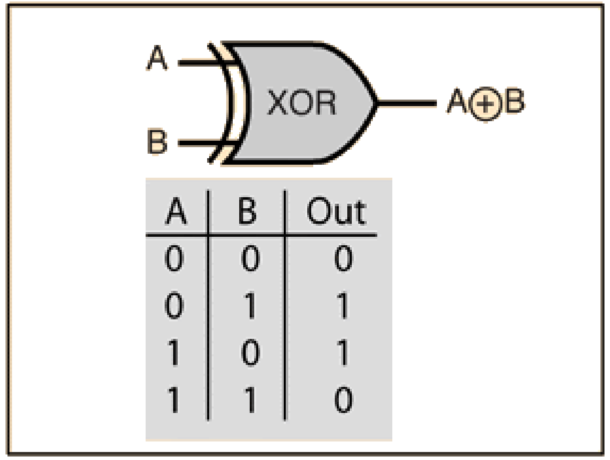
Hardware implementation

- A and B be two registers that hold the magnitudes of No
- As and Bs be two flip-flops that hold the corresponding signs
- The Result is transferred into A and As.
- Parallel adder is needed to perform the micro operation A + B.
- Output carry are transferred to E flip-flop
  - Where it can be checked to determine the relative magnitude of the Nos.
- Add overflow flip-flop (AVF) holds the overflow bit when A and B are added.



M=0,Input Carry=0 ▭ A+B
M=1,Input Carry=1 ▭ A+B’+1

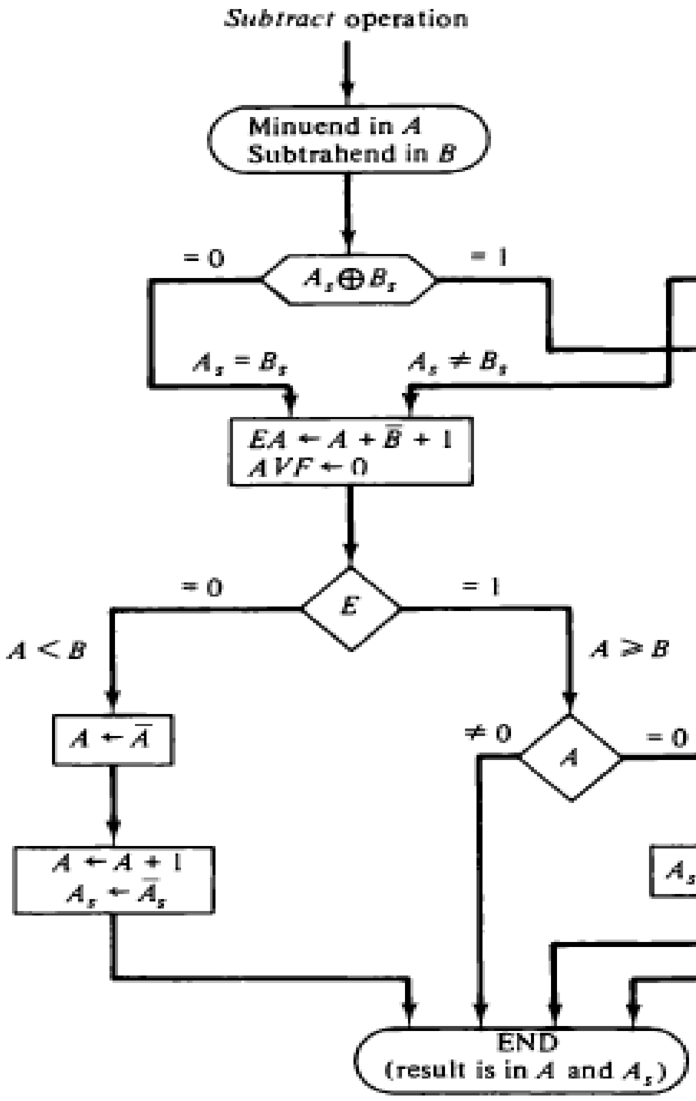
Hardware Algorithm



A	B	Out
0	0	0
0	1	1
1	0	1
1	1	0

Addition:
Signs are same ->add
Signs are different->subtract

Subtraction:
Signs are same ->subtract
Signs are different->add



Addition and Subtraction

- 2's Complement data
- Left most bit of a binary number represents sign-bit. (0 for =ve and 1 for –ve)
- If sign bit=1, then the entire number is represented in 2'complement form.
- Example +33   00100001

  -33   11011111
- Addition:
  - Add the numbers and treat the sign bits same as other bits
  - Carry out of the sign bit position is discarded
- Subtraction:
  - Take the 2'complement of the subtrahend and add it to minuend

Addition and Subtraction

- Overflow: When two n-bit numbers are added and the sum occupies n+1 bits

- The overflow can be detected by applying the last two carries out of the addition to an XOR gate. A ‘1’ at the output indicates an overflow.

Hardware and Algorithm for Signed 2's complement Data

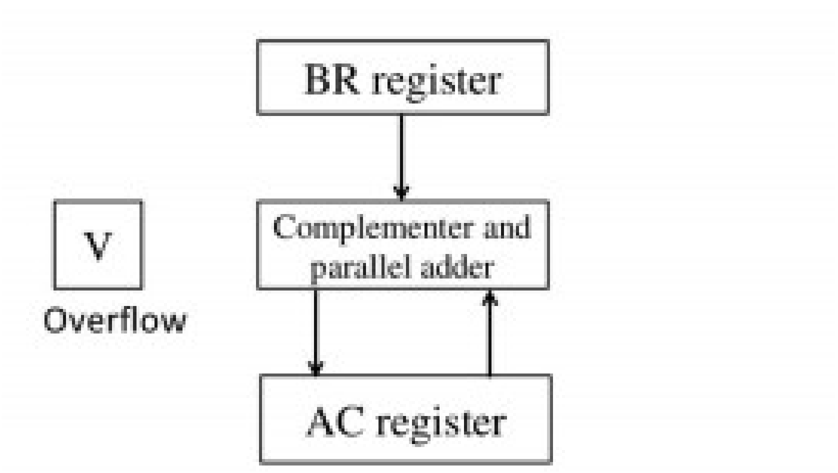


Figure: Hardware for signed-2's complement addition and subtraction.

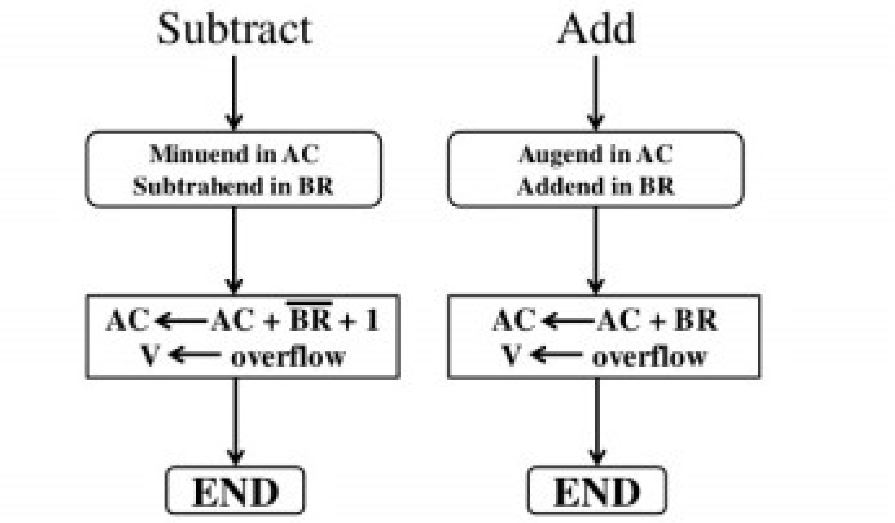
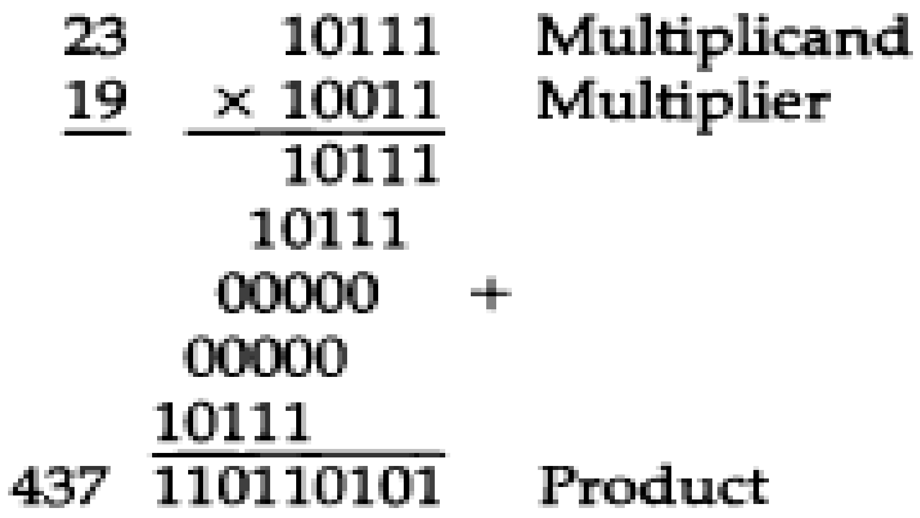


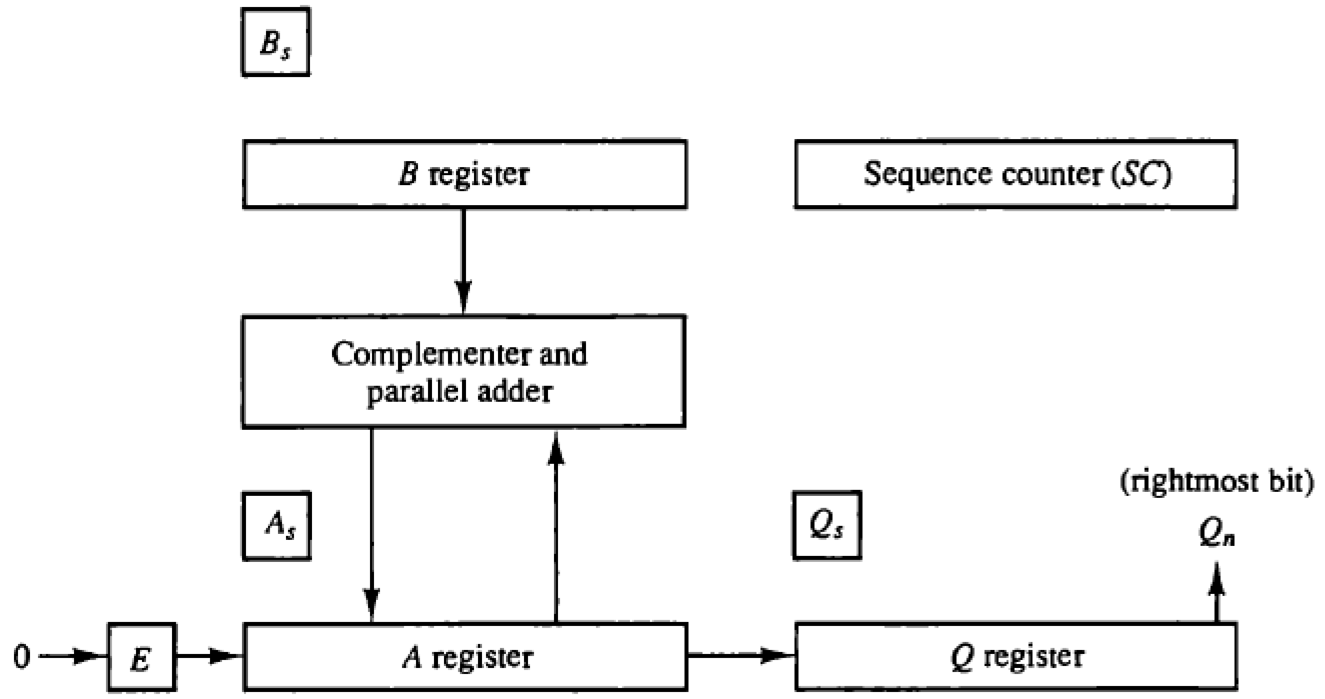
Figure: Algorithm for adding and subtracting numbers in signed-2's complement representation.

Multiplication Algorithm

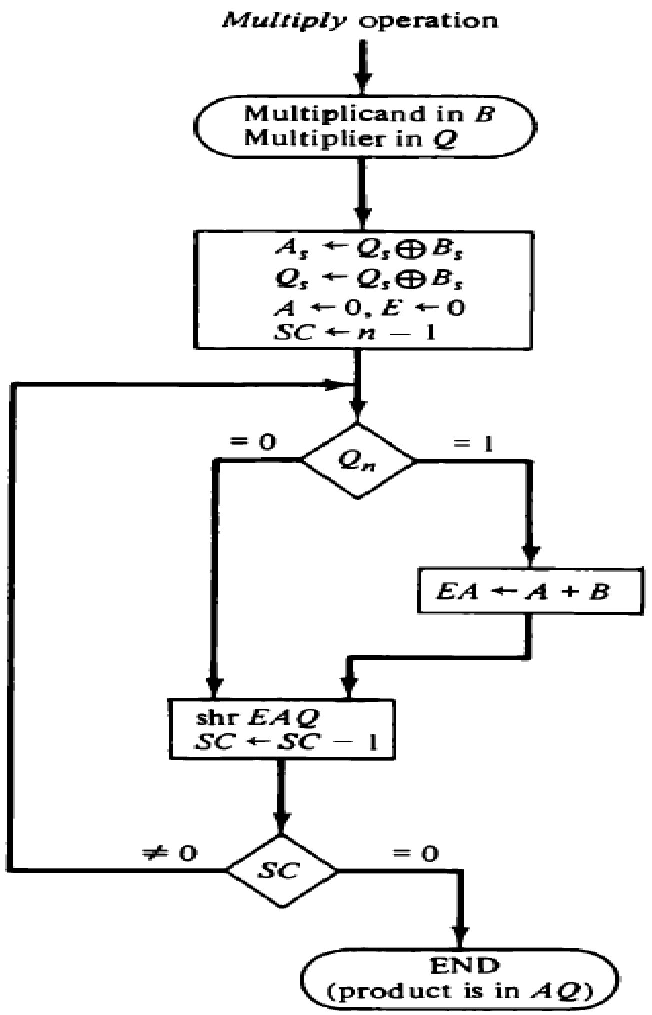


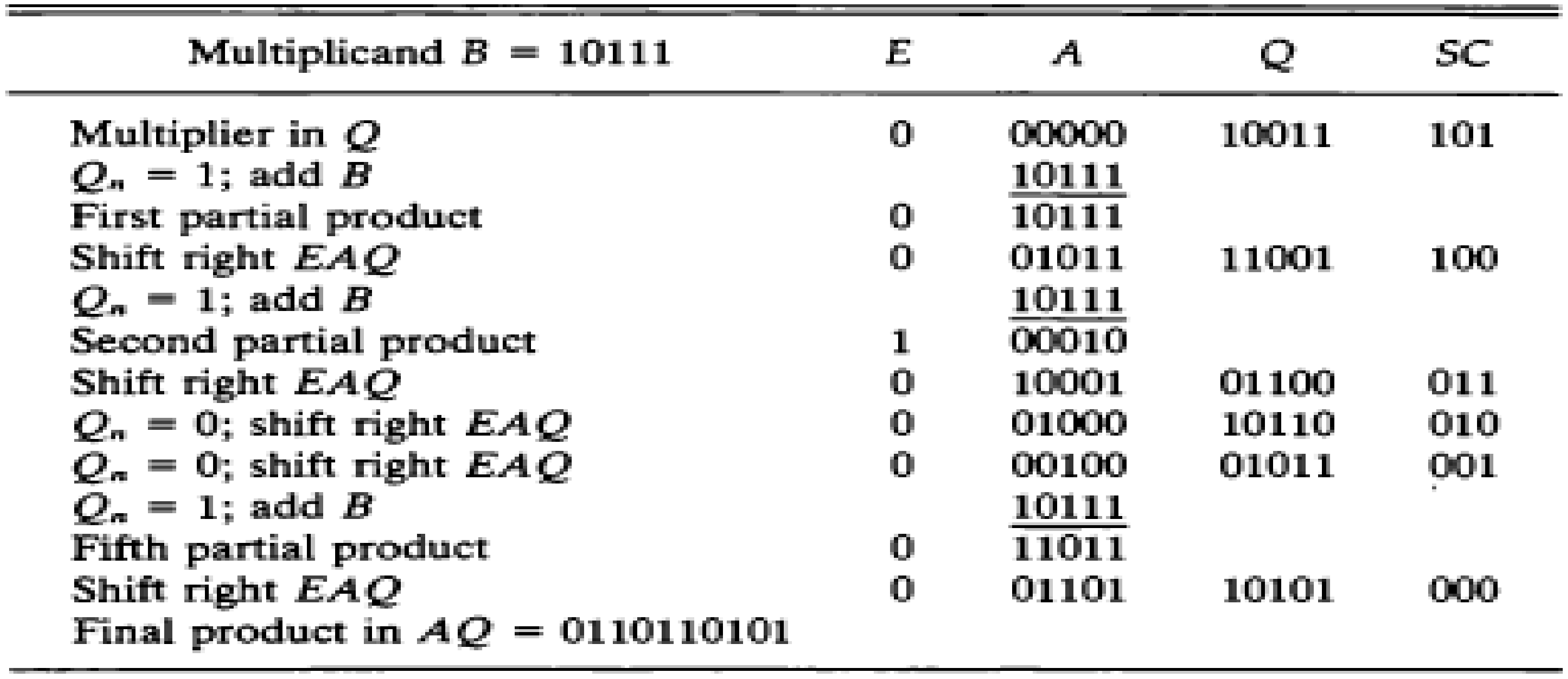
```
 23          10111    Multiplicand
 19        × 10011    Multiplier
          ---------
             10111
            10111
           00000    +
          00000
         10111
         ---------
437      110110101    Product
```

Hardware for Multiply operation



Flow Chart for Multiply operation





Multiplicand $B = 10111$	E	A	Q	SC
Multiplier in Q	0	00000	10011	101
$Q_n = 1$; add B		$\underline{10111}$		
First partial product	0	10111		
Shift right EAQ	0	01011	11001	100
$Q_n = 1$; add B		$\underline{10111}$		
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		$\underline{10111}$		
Fifth partial product	0	11011		
Shift right EAQ	0	01101	10101	000
Final product in $AQ = 0110110101$				

Booth Multiplication Algorithm

- Booth algorithm gives a procedure for multiplying binary integers in signed 2's complement representation in efficient way, i.e., less number of additions/subtractions required.
- 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) to 2^m.
- Example: (+14) is represented as 001110 has string of 1's from 2^3 to 2^1.

Here K=3,m=1

(+14) can be represented as 2^{k+1}-$2^{m=}2^4$-2^1=16-2=14.

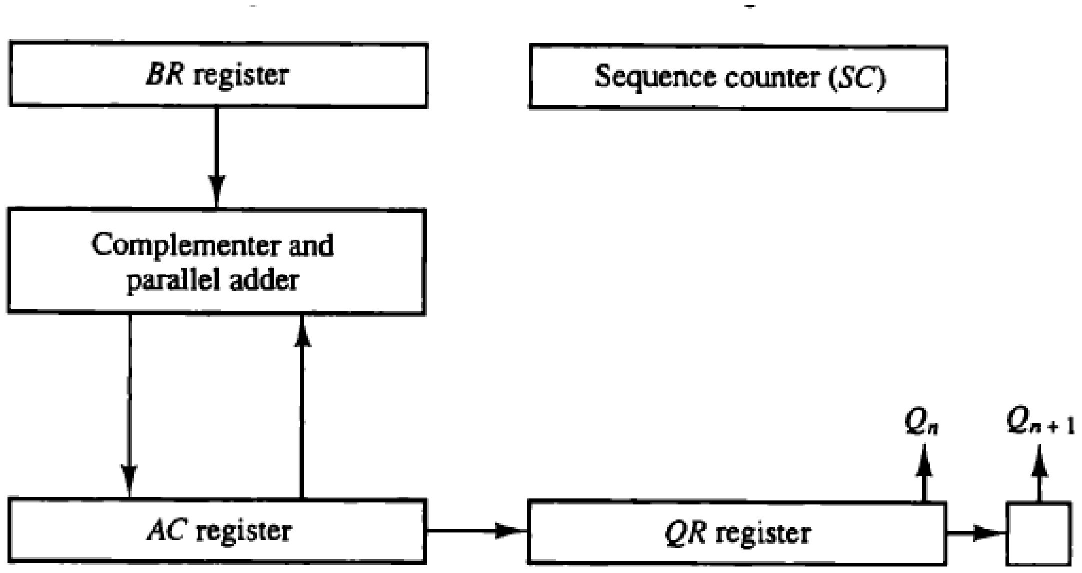
MX14 = MX2^4-MX2^1

Booth Multiplication Algorithm

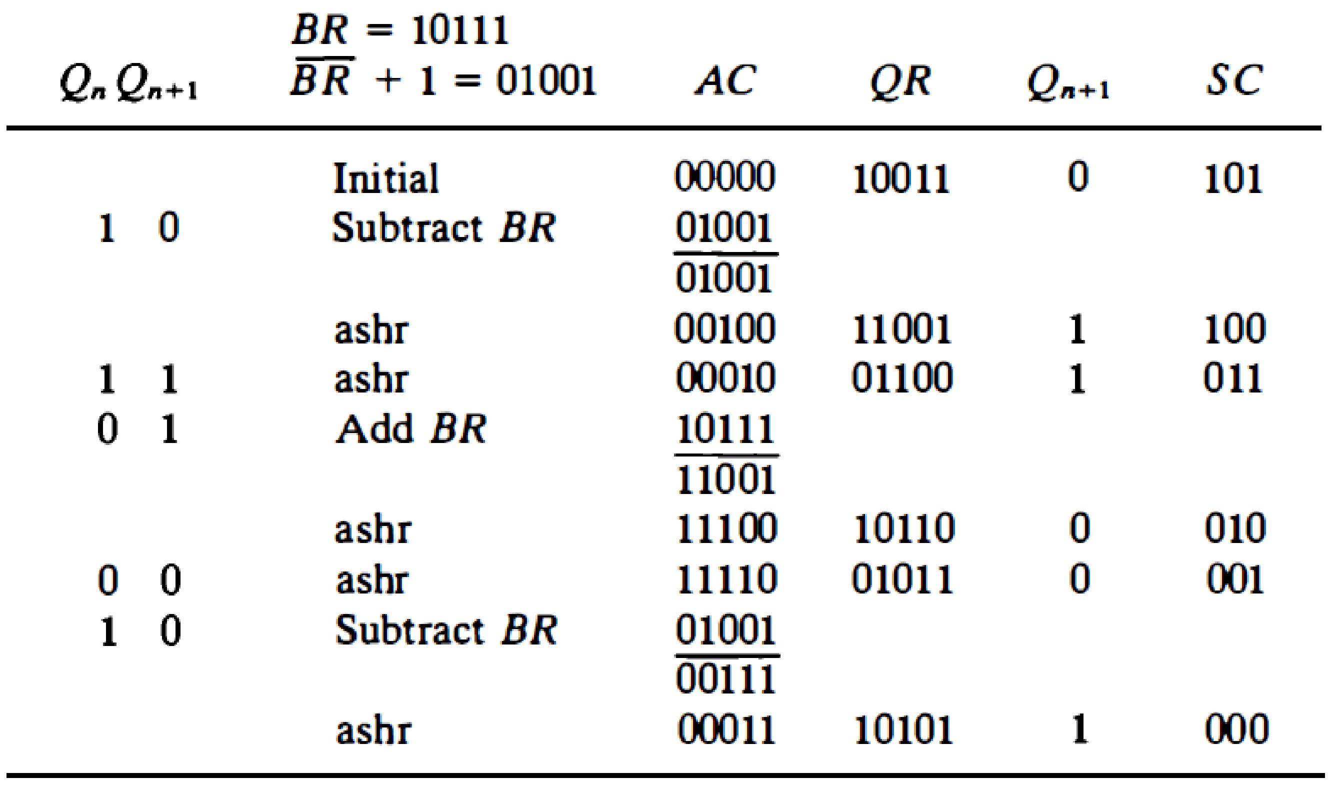
- As in all multiplication schemes, booth algorithm requires examination of the multiplier bits and shifting of the partial product.
- Prior to the shifting, the multiplicand may be added to the partial product, subtracted from the partial product, or left unchanged according to following rules:
  - 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 Implementation of Booths Algorithm

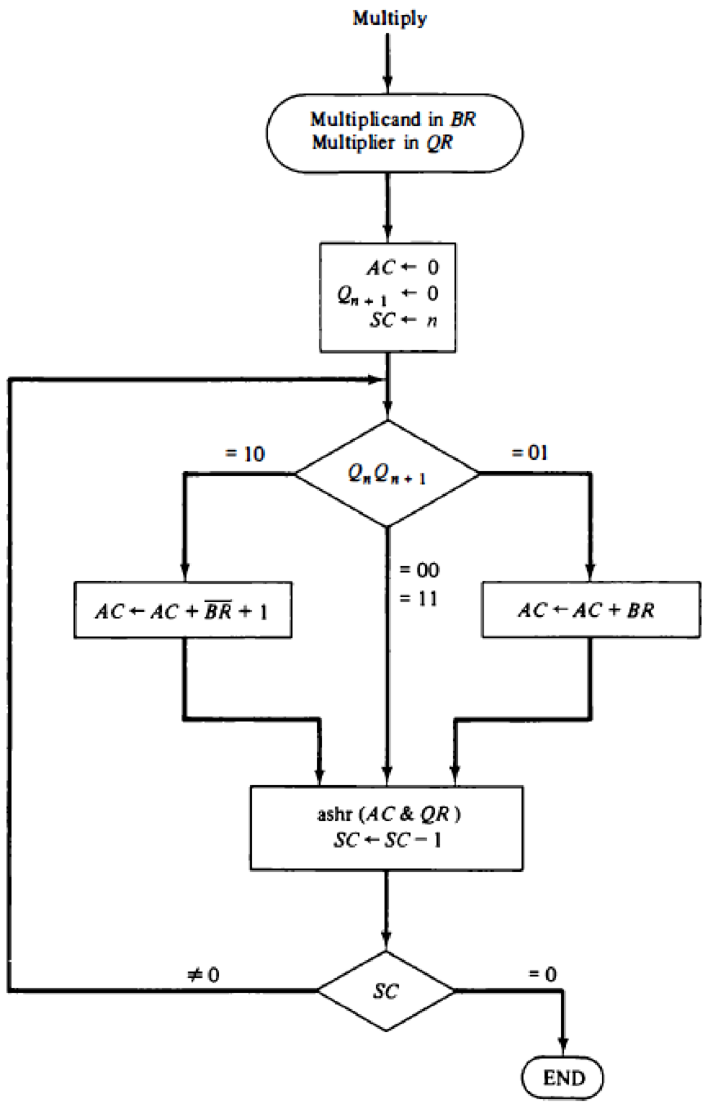
We name the register as A, B and Q, AC, BR and QR respectively.
Qn designates the least significant bit of multiplier in the register QR.
An extra flip-flop Qn+1is appended to QR to facilitate a double inspection of the multiplier.



Flowchart



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



Practice Examples

- Show the contents of E,A,Q and SC during the process of multiplication of two binary numbers, 11111 (multiplicand) and 10101(multiplier). The signs are not included.
- Show the step-by-step multiplication process using Booth algorithm when the following binary numbers are multiplied. Assume 5-bit registers that hold signed numbers.
  a. (+15) X (+13)
  b. (+15) X (-13)