Arithmetic and Logical Unit

Multiplication algorithm

- **Booth's algorithm** **(signed Number Multiplication):**
- **Treats both positive and negative operands uniformly.**

Algorithm :

Let A—Accumalator

Q – Multiplier register

M- Multiplicand

n iterations to be performed where n- no. of bits in the multiplier.

Initialization :

Initialize A to 0

Append a 0-bit in Q_{-1} position.

Do n times

1. Examine bits Q_0Q_{-1}
2. If Q_0Q_{-1}=01 then

   a) Perform A=A+M

b) RS(A.Q)

   If Q_0Q_{-1}=10 then

   a) Perform A=A-M

   b) RS(A.Q)

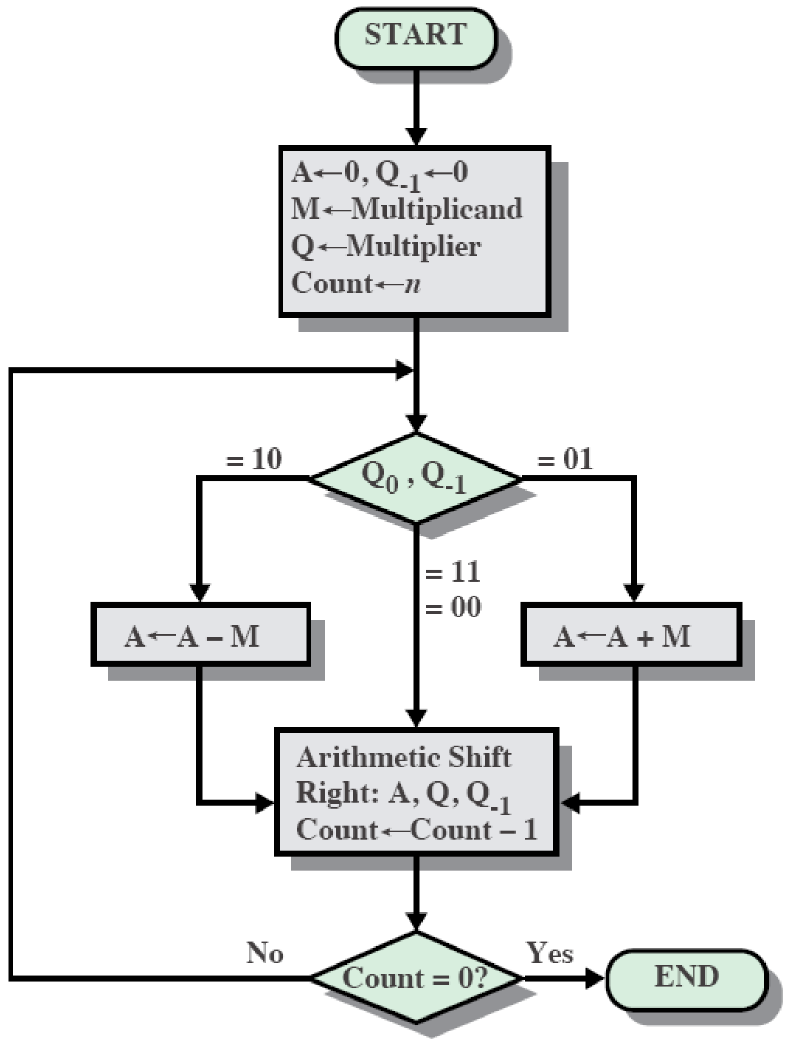
   If Q_0Q_{-1}=00/11 then

   a) RS(A.Q)

After n iterations discard the LSB of Q register.

Product-A.Q

Flowchart for Booth's algorithm



Example of Booth's Algorithm

1. 5 x 2

(0101) X (010)

	M	A	Q
	0101	0000	0100
1. RS		0000	0010
2. A=A-M		1011	0010
RS		1101	1001
3. A=A+M		0010	1001
RS		0001	0100 --→ Discard

Product 0001 010 = (10)

Example of Booth's Algorithm

2. 5 x 5

(0101) (0101)

	M	A	Q
	0101	0000	0101<u>10</u>
1. A=A-M		1011	01010
2. A=A+M		0010	10101
RS		1101	1001
3. A=A-M		1100	01010
RS		1110	00101
4. A=A+M		0011	00101
RS		0001	10010 --□ Discard

Product 00011001

Example of Booth's Algorithm

3. -5 x 4

(1011) (0100)

	M	A	Q
	1011	0000	01000
1. RS		0000	00100
2. RS		0000	00010
3. A=A-M		0101	00010
RS		0010	10001
4. A=A+M		1101	10001
RS		1110	11000 --□ Discard

Product 1110 1100

Efficiency of Booth's algorithm

- **Booth's algorithm is efficient when the multiplier Q contains a series of 1's or 0's since it minimizes on addition and subtraction operations.**

Booth’s recoding algorithm

- -1 times the shifted multiplicand is selected when moving from 0 to 1
- +1 times the shifted multiplicand is selected when moving from 1 to 0

as the multiplier is scanned from right to left

1. +7 x +8

(00111) (01000)

Multiplicand Multiplier

+8----→ 0 1 0 0 0 0 Implied zero

+1 -1 0 0 0 (Recoded multiplier)

2’s complement of +7 -----→ 11001

0 0 1 1 1

+1 -1 0 0 0

0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 x

0 0 0 0 0 0 0 0 x x

1 1 1 1 0 0 1 x x x

0 0 0 1 1 1 x x x x

1 0 0 0 0 1 1 1 0 0 0 (+56)

Discard

Product

2. +5 x +10

(00101) (01010)

Multiplicand Multiplier

+10----□ 0 1 0 1 0 0 Implied zero

+1 -1 +1 -1 0 (Recoded multiplier)

2's complement of +5 -----□ 11011

0 0 1 0 1

+1 -1 +1 -1 0

0 0 0 0 0 0 0 0 0 0

1 1 1 1 1 1 0 1 1 x

0 0 0 0 0 1 0 1 x x

1 1 1 1 0 1 1 x x x

0 0 0 1 0 1 x x x x

11 0 0 0 0 1 1 0 0 1 0 (+50)

Discard

Product

3. +13 x -6

(01101) (11010)

Multiplicand Multiplier

-6---□ 1 1 0 1 0 0 Implied zero

0 -1 +1 -1 0 (Recoded multiplier)

2's complement of +13 -----□ 10011

0 1 1 0 1

0 -1 +1 -1 0

0 0 0 0 0 0 0 0 0 0

1 1 1 1 1 0 0 1 1 x

0 0 0 0 1 1 0 1 x x

1 1 1 0 0 1 1 x x x

0 0 0 0 0 0 x x x x

1 1 1 1 0 1 1 0 0 1 0 (-78)

Discard

Product

4. -5 x -5

(1011) (1011)

Multiplicand Multiplier

-5---→ 1 0 1 1 0 Implied zero

-1 +1 0 -1 (Recoded multiplier)

2's complement of -5 -----→ 0101

1 0 1 1

-1 +1 0 -1

---

0 0 0 0 0 1 0 1

0 0 0 0 0 0 0 x

1 1 1 0 1 1 x x

0 0 1 0 1 x x x

---

1 0 0 0 1 1 0 0 1 (25)

Discard

Product

Unsigned number Multiplication Example

1011 Multiplicand (11)

x 1101 Multiplier (13) Partial

1011 product(PP)

0000 PP

1011 PP

1011 PP

10001111 Product (143)

- Note: if multiplier bit is 1 copy multiplicand otherwise zero
- Note: need double length result

Unsigned Number multiplication

Algorithm : M= 1011 Q= 1101

Let a variable x be initialized to 0000[4 bit binary number]

1. Since the bit q0 of multiplier is 1

   x=x+ 1011= 0000 + 1011 = 1011

   Right shift x ---- 0 1 0 1 | 1

2. Since the bit q1 of multiplier is 0

   x= x+ 0000 = 0 1 0 1 | 1

   Right shift x ---- 0 0 1 0 | 1 1

3. Since the bit q2 of multiplier is 1

   x=x+ 1011= 0 0 1 0 | 1 1

   1 0 1 1

   1 1 0 1 | 1 1

Unsigned number Multiplication

Right shift x ---- 0 1 1 0 | 1 1 1

4. Since the bit q3 of multiplier is 1

x=x+ 1011= 0 1 1 0 | 1 1 1

1 0 1 1

10 0 0 1 | 1 1 1

Right shift x---- 1 0 0 0 | 1 1 1 1

Product - 1 0 0 0 | 1 1 1 1

Unsigned number Multiplication

2. 10 x 5

1010 101

(M) (Q)

X= 0000

1. Since q0 is 1

X= X + 1010 = 1010

RS ---- 0101 | 0

2. Since q0 is 0

X=X + 0000 = 0101 | 0

RS ---- 0010 | 10

3. Since q0 is 1

X= X+ 1010

0010 | 10

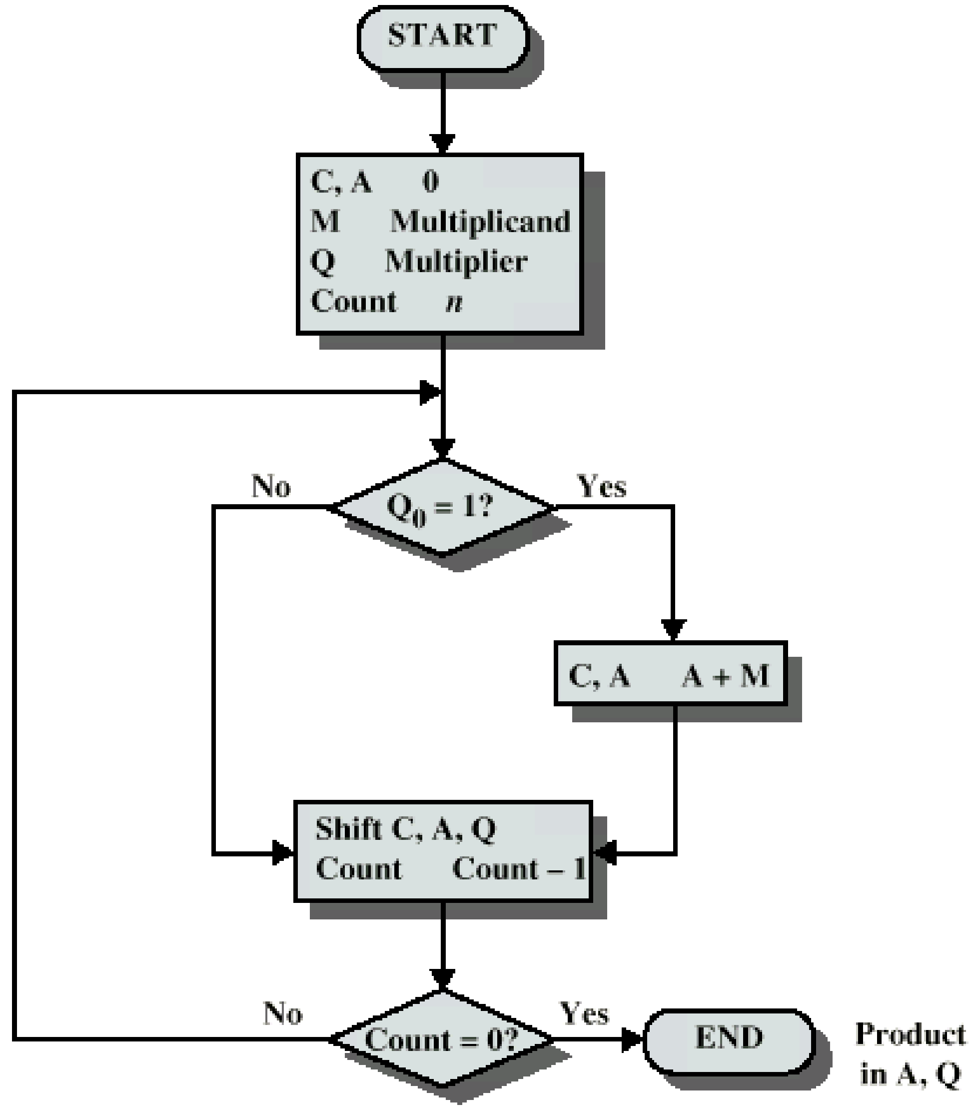
1010

1100|10

RS----- 0110 | 010

Product – 0110010 (50)

Flowchart for Unsigned Binary Multiplication



Division algorithms

```
          1 0 1  0
1010  | 110 10 0  1
       -  110 1
     0110
   - 1010
                    1 101
               +    1 010
                      1100
                  -  1010
                       0101
                   -  1 0 10
                       1011
                      +1 0 10
                       0  1  0  1   ----- Remainder
```

Restoring Division Algorithm

- Positive divisor ---→ Reg. M
- Positive dividend -→ Reg Q
- Reg A--→0
- After division is complete

  Quotient ----- Q

  Remainder ----- A

A 0 bit is added at the left end of both A and M to serve as a sign bit for subtraction.

Algorithm :

Do n times (n--→ no. of bits in Q)

1. Shift A and Q left one binary position.
2. Subtract M from A, placing the answer back in A (A=A-M)
3. If the sign of A is 1 set qo to 0 and add M back to A (restore A) , otherwise set q0 to 1.

Restoring Division Algorithm

1. 9 / 5

1001 101

M		A	Q
0101		0000	1001
	1. Sh	0001	001
	Sub (A=A-M)	1100	0010
		+0101	
		0001	
	2. Sh	0010	010
	Sub (A=A-M)	1101	0100
		+0101	
		0010	

Restoring Division Algorithm

M		A	Q
0101			
	3. Sh	0100	100
	Sub (A=A-M)	1111	1000
		+0101	
		0100	
	4. Sh	1001	000
	Sub (A=A-M)	0100	0001
		Remainder	Quotient

Restoring Division Algorithm

2. 12 / 10

1100 1010

M		A	Q
01010		00000	1100
	1. Sh	00001	100
	Sub (A=A-M)	10111	1000
	Restore	00001	
	2. Sh	00011	000
	Sub (A=A-M)	11001	0000
	Restore	00011	

Restoring Division Algorithm

M		A	Q
01010			
	3. Sh	00110	000
	Sub (A=A-M)	11100	0000
	Restore	00110	
	4. Sh	01100	000
	Sub (A=A-M)	00010	0001
		Remainder	Quotient

Non Restoring division algorithm

- If A is (+) we shift left and subtract M from A
  i.e 2A-M
- If A is (-)
  1. Restore A---- A+M
  2. Left Shift 2(A+M)
  3. Subtract M from A
     2(A+M)-M= 2A+M

Algorithm :

1. Do n times (n—no. of bits in Q)

   If the sign of A is 0, shift A and Q left one binary position and subtract M from A, otherwise shift A and Q left and add M to A.

   If the sign of A is 0 , set q0 to 1 else set q0 to 0.

2. After n iterations , if the sign of A is 1, add M to A

Non Restoring division algorithm

- **Eg : 8/3**

M		A	Q
011		000	1000
	1. Sh	001	000
	Sub	110	0000
	2. Sh	100	000
	Add	111	0000
	3. Sh	110	000
	Add	001	0001
	4. Sh	010	001
	Sub	111	0010
		111	Quotient
		+011	
		010—Remainder	

Non Restoring division algorithm

- **Eg : 15/3**

M		A	Q
011		000	1111
	1. Sh	001	111
	Sub	110	1110
	2. Sh	101	110
	Add	000	1101
	3. Sh	001	101
	Sub	110	1010
	4. Sh	101	010
	Add	000	0101
		Remainder	Quotient

Floating point number representation

2 ways of representing decimal nos :

1. **Fixed point representation**

   The usual way of representating numbers is to write the number with the decimal point fixed in it's correct position between the 2 appropriate digits eg : 13.75 or 345.78

   This is called fixed point representation.

   Drawback:

   This representation becomes cumbersome when dealing with several very large or very small numbers.

   eg: 0.00000001375

   or 1375000000000

2. Floating point representation

Three numbers associated with a floating point number are :

a. Mantissa M

b. Exponent E

c. Base B

These three numbers together represent the real number $M \times B^E$

Eg : 1.0×10^{18}

1.0- Mantissa

18- Exponent

10- Base

Base B – constant,

Therefore a floating point number is stored as (M,E)

An Example

Suppose that M and E are both 3-bit sign and magnitude integers and B=2

- **M and E each can assume the values +/-0 , +/-1, +/-2 and +/-3.**
- **All binary words of the form (M,E) = (x00,xxx) represent zero where x denotes either 0/1.**
- **The smallest non zero positive number is :**

  (001,111)= 1 x2^{3}= 0.125

The smallest non zero negative number is :

(101,111)= -1 x2^{3}= -0.125

The largest representable positive number is :

(011,011)= 3 x2^{3}=24

The largest representable negative number is :

(111,011)= -3 x 2^{3}= -24

Note : The left most bit which is the sign of the mantissa is also the sign of the floating point number

Normalization of floating point numbers

1.0×10^{18}

0.1×10^{19}

1000000×10^{12}

0.000001×10^{24}

----- All represent the same number

- Floating point representation is redundant i.e. the same no can be represented in more than one way.
- It is generally desirable to have a unique/normal form for each representable number in a floating point system.
- The mantissa is said to be normalized if the digit to the right of the binary point is non zero.

  eg: $0.1bbbbbb \times 2^{+/E}$

Hence 0.1×10^{19} is the unique normal form of 1.0×10^{18}

Signed number representations

- **Basically, there are two common signed number representations:**

1. Sign and magnitude

000	**+0**
001	**+1**
010	**+2**
011	**+3**
100	**-0**
101	**-1**
110	**-2**
111	**-3**

Drawback : 2 representations for number 0 (+0 and -0)

Signed number representations

2. Two's complement representation

000	+0
001	+1
010	+2
011	+3
100	-4 / +4
101	-3
110	-2
111	-1

Hence for a 3-bit number (+3 to -4) 0r (-3 to +4)

$2^{n-1}-1$ **to** -2^{n-1} $-2^{n-1}+1$ **to** 2^{n-1}

An 8 bit register holds numbers in 2's complement form with leftmost bit as the sign bit.

1. What is the largest (+) number that can be stored? Express the answer in decimal and binary format.
2. What is the largest (-) number that can be stored? Express the answer in decimal and binary format.

$2^{n-1}-1$ to -2^{n-1}

127 to -128

01111111 10000000

Or

$-2^{n-1}+1$ to 2^{n-1}

-127 to +128

10000001 to 10000000

An Example

- **An example of a floating point number in binary format (32 bit)**

Assumptions :

1. **Implied base 2.**
2. **7-bit signed exponent is expressed as a 2's complement integer.**
3. **Left most bit represents sign of the number (0 means + ,1 means -)**

S bit	7 –bit signed exponent	24-bit fractional mantissa

32 bit

The 24 bit mantissa is considered to be a fraction with the binary point at it's left end .

To retain as many significant bits as possible, the fraction al mantissa is kept in a normalized form i.e it's leftmost bit is always 1.

An Example

Eg:

0	0 0 0 1 0 0 1	001000000000000000000000

+ 0.001000000000000000000000 x 2^9

Unnormalized value

0	0 0 0 0 1 1 1	100000000000000000000000

+ 0.100000000000000000000000 x 2^7

Normalized value

Binary Normalization example

An Example

- Hence a 7-bit 2's complement exponent has a range of --

  $2^{n-1}-1$ to -2^{n-1}

  2^6-1 to -2^6

  +63 to -64
- Some computers employ a different type of representation for exponents i.e. biased exponent representation
- Here the n-bit exponent is expressed in excess $2^{(n-1)}$ format.
- Hence the 7-bit exponent is expressed in excess 64 format.
- In this representation the sign bit is removed from being a separate entity.
- A (+) no. called bias (64) is added to each exponent as the floating point number is formed, so that internally all exponents are positive.

An Example

- E' —new exponent
- E' = E + 64
- Since -64 ≤ E ≤ +63
- 0 ≤ E' ≤ +127

-64 is represented as ------- 0000000

-63 is represented as ------- 0000001

-62 is represented as ------- 0000010

1 is represented as ------- 1000000

2 is represented as ------- 1000001

+63 is represented as ------- 1111111

IEEE 754 Floating point number format

- The IEEE Standard for Floating- Point Arithmetic (IEEE 754) is a technical standard established by the Institute of Electrical and Electronics Engineers (IEEE) and the most widely used standard for floating-point computation, followed by many CPU.
- IEEE 754 defines 2 standards

  1. Single precision no. (32 bits)
  2. Double precision no. (64 bits)

IEEE 754 Floating point number format(Single precision)

- **The standard for a 32 bit floating point number is ----**

1. Sign bit S

2. 8 bit exponents field E'

(in excess 127 representation) i.e the bias added here is $2^{n-1}-1$

3. 23-bit mantissa field M

4. Base 2

S	E'	M

Sign of the number
0 ----- +
1 ----- -

8 bit signed exponent in excess 127 representation

23 bit fractional mantissa

IEEE 754 Floating point number format(Single precision)

- Since E varies from
- $-2^{n-1}+1$ to 2^{n-1}
- i.e $-127 \leq E \leq +128$
- $E' = E + 127$
- $0 \leq E' \leq +255$

E	E'	
-127	00000000	represents the floating point no. zero
-126	00000001	
0	01111111	
1	10000000	
+128	11111111 represents ∞	

IEEE 754 Floating point number format(Single precision)

Normalization in IEEE 754

- **In IEEE 754 format the mantissa is actually of the form 1.M where 1 is to the left of the binary point.**
- **This I is implicit/hidden and is not stored along with the number but assumed to be there.**
- **Use of hidden 1 means that the precision of a normalized number is effectively increased by 1 bit.**
- Eg :

0	0 0 1 0 1 0 0 0	001010………0

- **Value represented ---** **M= 1.001010**

E'= 40

E=E'-127 = 40 – 127 = -87

1.001010… X 2^{-87}

Hence in IEEE 754 format the actual value of N

$N= (-1)^S\ 2^{E'-127} \times (1.M)$

IEEE 754 Floating point number format(Double precision)

S	E’	M

Sign of the number
0 ----- +
1 ----- -

11 bit signed exponent in excess 1023 representation

52bit fractional mantissa

- **$-1023 \leq E \leq +1024$**
- **$E' = E + 1023$**
- **$0 \leq E' \leq +2047$**
- **Here the actual value of N is**

$$N = (-1)^S 2^{E'-1023} X (1.M)$$

Numericals on IEEE 754

Convert the following nos. in IEEE- 754 single precision format.

1. 10.58

$(1010.100101)_2$

1.010100101×2^3

E'=127+3= 130=10000010

S=0

M=010100101

0	10000010	0 1 0 1 0 0 1 0 1

Numericals on IEEE 754

2. $-8.08 = -(1000.000101)$

$-(1.000000101 \times 2^3)_2$

$E' = 127 + 3 = 130$

$S = 1$

$M = 000000101$

1	10000010	0 0 0 0 0 0 1 0 1………

3. 1.5

$(1.1)_2$

1.1×2^0

$E' = 0 + 127$

0	01111111	10……0

Numericals on IEEE 754

4. 21

$21 = (10101)_2$

$= 1.0101 \times 2^4$

E' = 127 + 4 =131

M= 0101....

S=0

0	10000011	0 1 0 1 0.........0

5.

0.021=

0.00000011001

$= 1.1001 \times 2^{-7}$

E'= 127-7= 120

M= 1001

S= 0

Numericals on IEEE 754

- **Give the value represented by the following IEEE single precision no.**

1	10000001	0 1 0.........0

E'= 10000001 = 129

E= E' -127 = 129-127 = 2

$(1.01)_2 = (1.25)_{10}$

$X = -1.25 \times 2^2 = -5$

Numericals on IEEE 754

Convert the following no. in IEEE- 754 double precision format.

$(0.0625)_{10}$ =

0.0625 x 2 =0.125 ----□ 0

0.125 x 2 = 0.25 ----□ 0

0.25 x 2 = 0.5 ----□ 0

0.5 x 2 =1 ----□ 1

$(0.0625)_{10} = (0.0001)_2 = 1.0 \times 2^{-4}$

E’ = E + 1023 = $(1019)_{10}$

M = 00000……

S =0

Floating point Addition/subtraction

- **$X = 0.3 \times 10^{2}$**
- **$Y = 0.2 \times 10^{3}$**
- **$X+Y = (0.03 \times 10^{3}) + (0.2 \times 10^{3}) = 0.23 \times 10^{3}$**
- **$X-Y = (0.03 \times 10^{3}) - (0.2 \times 10^{3}) = -0.17 \times 10^{3}$**

Floating point Addition/subtraction

- **Floating point arithmetic with binary numbers:**

1) $1.10101 \times 2^4 + 1.00101 \times 2^6$

$(0.0110101 \times 2^6 + 1.00101 \times 2^6)$

$= 1.1001001 \times 2^6$

2) $1.10011 \times 2^4 + 1.00101 \times 2^4$

$= 10.11000 \times 2^4$ (significand/mantissa overflow)

Normalize the result 1.011000×2^5

3) $1.00110 \times 2^{254} + 1.10001 \times 2^{254}$

10.10111×2^{254}

1.010111×2^{255}

exponent =255 signifies exponent overflow

Floating point Addition/subtraction

4) $1.11011 \times 2^{6} - 1.11001 \times 2^{6}$

$= 0.00011 \times 2^{6}$

Normalize the result – 1.1×2^{2}

When decrementing the exponent check for exponent underflow

5) $1.00111 \times 2^{10} + 1.10111 \times 2^{35}$

$0.00000000000000000000000000100111 \times 2^{35} +$

1.10111×2^{35}

$=1.101110000000000000000000100111 \times 2^{35}$

(significand underflow)

Result= 1.10111×2^{35}

Floating point Addition/subtraction

- **Floating-point subtraction example:**

  E=10001010; S=1.11100000000000000000000

- **E=10001010; S=1.11000000000000000000000**

  E=10001010; S=0.00100000000000000000000

  Difference

  E=10000111; S=1.00000000000000000000000

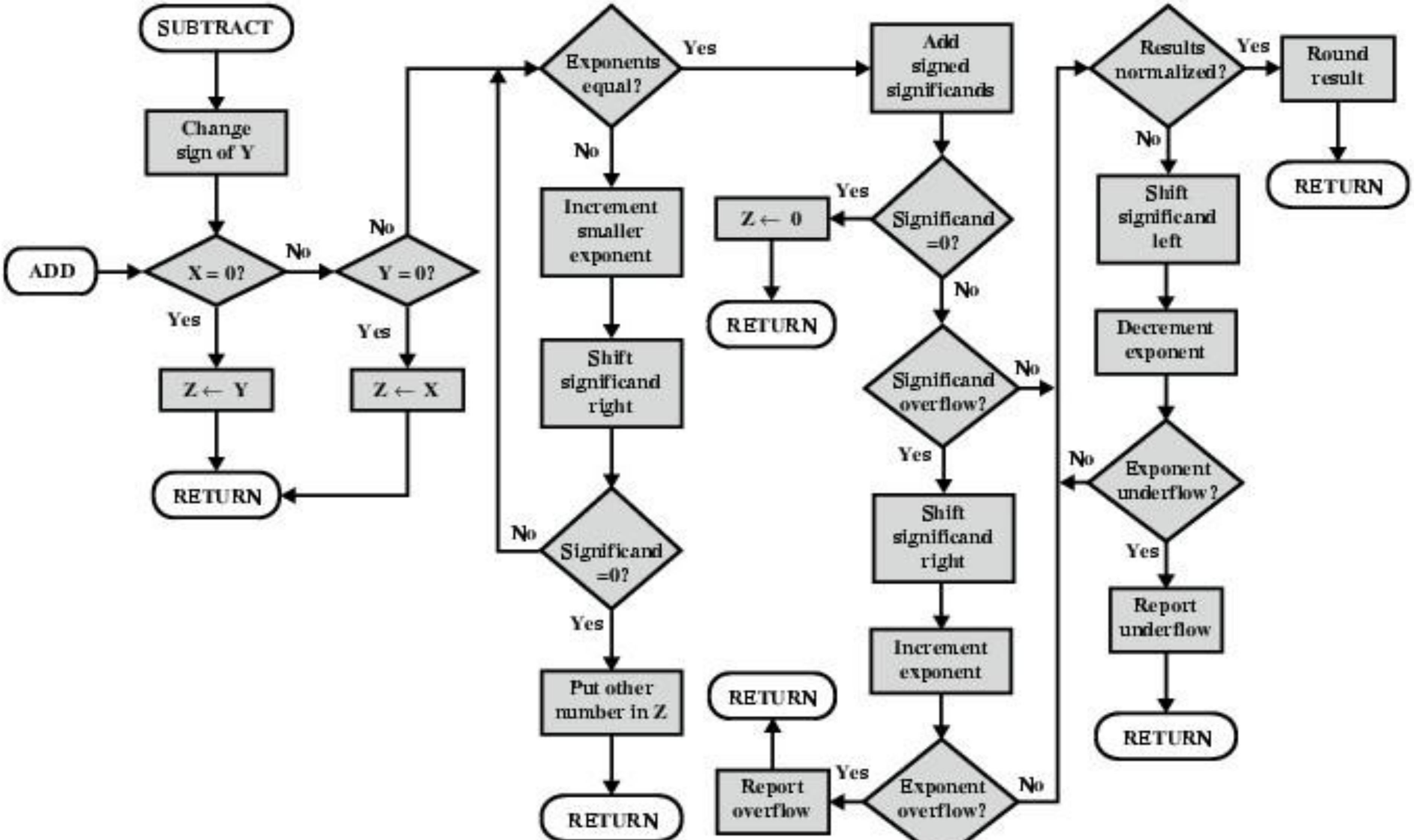
  Normalization

Floating point Addition/subtraction

Floating point arithmetic (+/-)

- **Check for zeros**
- **Align significands (adjusting exponents)**
- **Add or subtract significands**
- **Normalize result**

Flowchart for floating point addition and subtraction



FP Arithmetic x/÷

- Check for zero
- Add/subtract exponents
- Multiply/divide significands (watch sign)
- Normalize
- Round
- All intermediate results should be in double length storage

Floating point multiplication

- Eg: Multiply 21.44 by 7.24
- Convert into IEEE 754 format
- 21.44 = 10101.0111 x 2^0

  = 1.01010111 x2^4

  M= 01010111 E'= 131
- 7.24=111.0011 x 2^0

  =1.110011 x 2^2

  M= 110011 E'= 129

	sign	exponent	significand
register A 21.44	0	10000011	01010111000010101010001
register B 7.24	0	10000001	11001111010111000010100

Floating point multiplication

- For multiplication- Add the exponents and multiply the corresponding mantissas

Step 1 : Add exponents

- 129+131= 260
- When the exponents are added the resulting exponents is doubly biased.
- Hence correct the resulting sum by subtracting one bias from the resulting exponent.
- 260-127=133.

Floating point multiplication

Step 2 : Multiply the mantissa.

- **Considering only the first 4 digits of the mantissas**
- **1.010 x1.110 = 10.001100**

Step 3 : Normalize the mantissa

- **1.0001100 x 2 ^ 134**

Floating point multiplication

- **Points to Note :**

1. **Exponent overflow**

Xe = 120

Ye = 122

X’e= 120 + 127 = 247

Y’e= 122+127=249

X’e + Y’e =496

Subtract a bias : 496-127 = 369

Range of E’ from 0 to 255

Hence there is an exponent overflow

Floating point multiplication

2. Exponent underflow

Xe = -60

Ye = -74

X’e= -60 + 127 = 67

Y’e= -74 +127=53

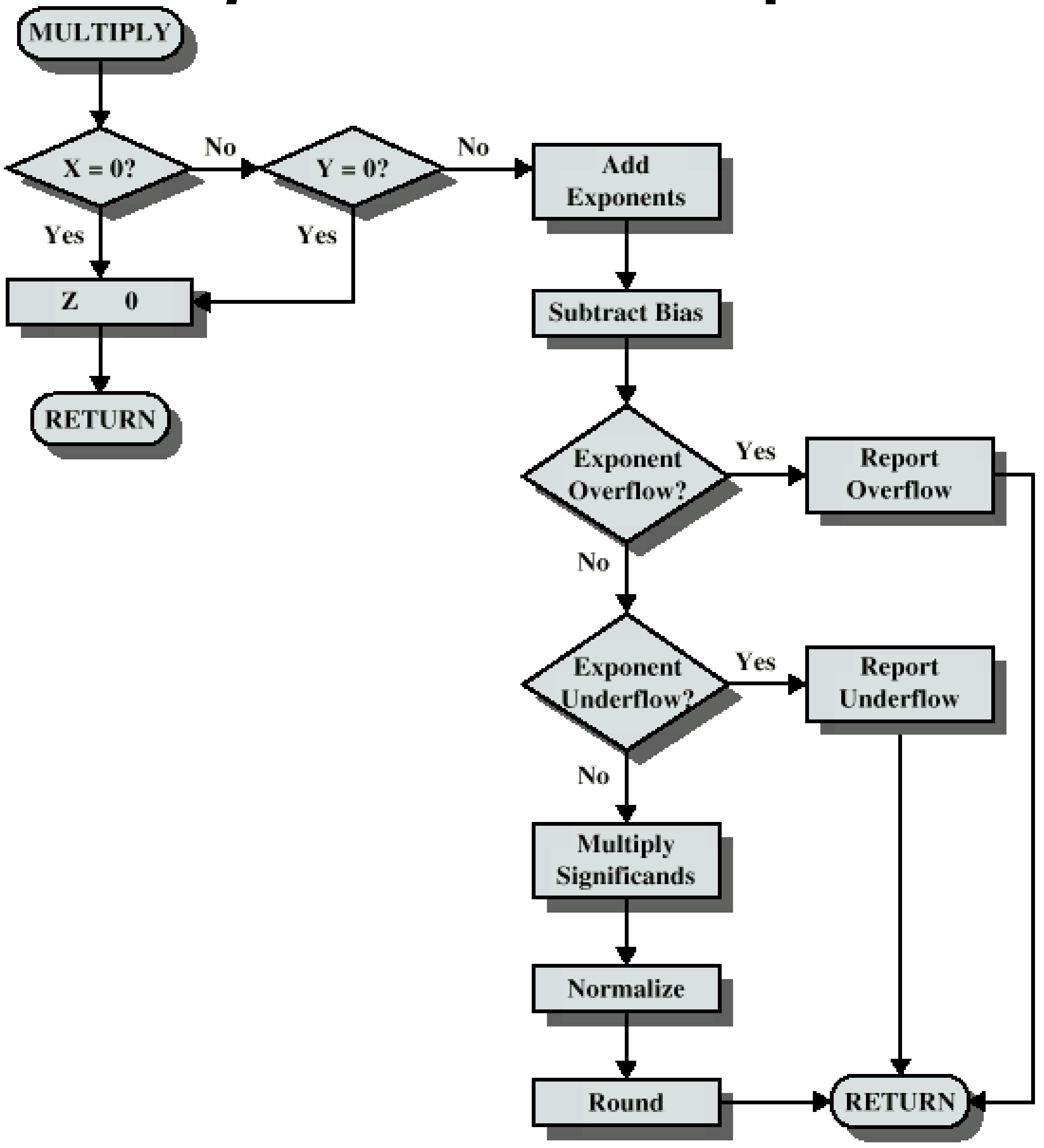
X’e + Y’e =120

Subtract a bias 120-127 = -7

Range of E’ from 0 to 255

Hence there is an exponent underflow

Floating Point Multiplication



- For division :
- X=Xm x2^Xe
- Y=Ym x 2^Ye

Z=(Xm/Ym) x 2^(Xe-Ye)

Note when the divisor exponent (Xe+127) is subtracted from dividend exponent (Ye+127), it removes the bias which must be added back in

i.e ((Xe+127) -(Ye+127)) = Xe-Ye

Hence correction– (Xe-Ye) +127

Case 1 : Assume Xe= 120 , Ye= - 2

Xe'= 147   Ye'= +125

Xe'-Ye'=272

Correction : 272 + 127 =399 ------- Exponent overflow

Floating Point Division