# Multiplication of Signed Integers
# Booth's Algorithm

**Prepared by Dr. Kanan Bala Ray**

# Booth's Algorithm

It treats both +ve and -ve multiplier in the uniform way.

M X 0011110[30]=M X ( $2^5$ - $2^1$) [30=32-2=$2^5$ -$2^1$]

We know that, multiplying something by $2^i$ is equivalent to shifting the number **left** by **i times**.

We know that, multiplying something by **-**$2^i$ is equivalent to shifting the **2's complement** of the number **left** by **i times**.

# Booth’s Algorithm

In general, in the Booth scheme,

-1 times the shifted multiplicand is selected when moving from 0 to 1 in the multiplier.

+1 times the shifted multiplicand is selected when moving from 1 to 0, as the multiplier is scanned from right to left.

1 0 0 1 1 1 0 0 1 0 1 0 1 0 0

**-1 0 +1 0 0 -1 0 +1 -1 +1 -1 +1 -1 0 0**

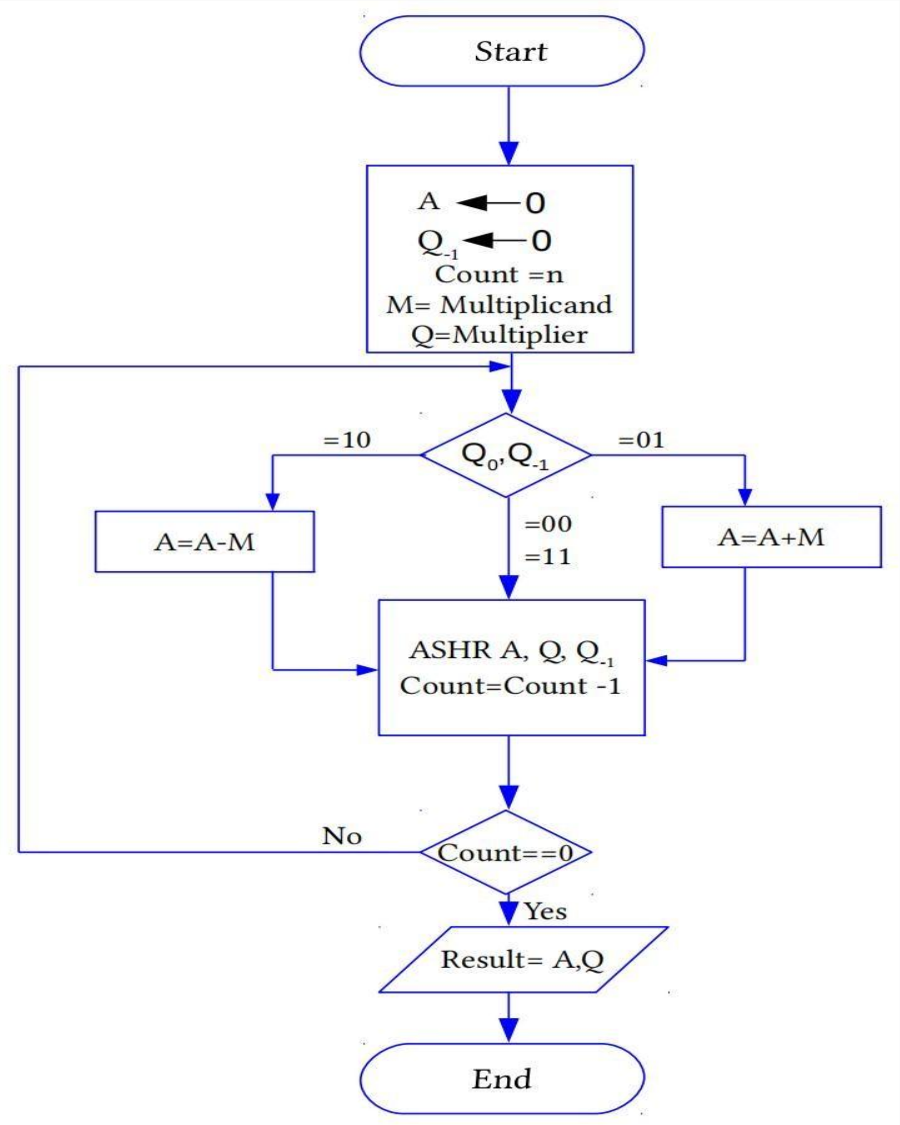
# Booth's Algorithm

| Multiplier | | Version of multiplicand selected by $i^{th}$ bit |
|---|---|---|
| Bit $_i$ | Bit $_{i-1}$ | |
| 0 | 0 | 0 X M |
| 0 | 1 | +1 X M |
| 1 | 0 | -1 X M |
| 1 | 1 | 0 X M |

Booth multiplier recoding table.

# Multiplication of Signed Integers Booth's Algorithm[Method 2]

# Flowchart for Booth's Algorithm



# Booth's Algorithm (-7 x 3) [+7=0111 -7=1001]

| Register | Multiplier Register | | Multiplicand Register | Operation | Remark |
|---|---|---|---|---|---|
| A | Q | $Q_{-1}$ | M | | |
| 0000 | 0011 | 0 | 1001 | Initially | |
| | | | | | |
| 0111 | 0011 | 0 | | A=A-M=A+ 2s(M) | 1st Cycle |
| 0011 | 1001 | 1 | | Shift A, Q, $Q_{-1}$ right | |
| | | | | | |
| 0011 | 1001 | 1 | | No add/sub | 2nd Cycle |
| 0001 | 1100 | 1 | | Shift A, Q, $Q_{-1}$ right | |
| | | | | | |
| 1010 | 1100 | 1 | | A=A+M | 3rd Cycle |
| 1101 | 0110 | 0 | | Shift A, Q, $Q_{-1}$ right | |
| | | | | | |
| 1101 | 0110 | 0 | | No add/sub | 4th Cycle |
| 1110 | 1011 | 0 | | Shift A, Q, $Q_{-1}$ right | |

# Booth's Algorithm (14 x -7) [+14=01110 -14=10010]

| Register | Multiplier Register | | Multiplicand Register | Operation | Remark |
|---|---|---|---|---|---|
| A | Q | $Q_{-1}$ | M | | |
| 00000 | 11001 | 0 | 01110 | Initially | Count=5 |
| | | | | | |
| 10010 | 11001 | 0 | | A=A-M=A+ 2s(M) | Count=4 |
| 11001 | 01100 | 1 | | AShift A, Q, $Q_{-1}$ right | |
| | | | | | |
| 100111 | 01100 | 1 | | A=A+M | Count=3 |
| 00011 | 10110 | 0 | | AShift A, Q, $Q_{-1}$ right | |
| | | | | | |
| 00011 | 10110 | 0 | | No add/sub | Count=2 |
| 00001 | 11011 | 0 | | AShift A, Q, $Q_{-1}$ right | |
| | | | | | |
| 10011 | 11011 | 0 | | A=A-M=A+ 2s(M) | Count=1 |
| 11001 | 11101 | 1 | | AShift A, Q, $Q_{-1}$ right | |

# Booth's Algorithm (14 x -7) [+14=01110 -14=10010]

| Register | Multiplier Register | | Multiplicand Register | Operation | Remark |
|---|---|---|---|---|---|
| A | Q | $Q_{-1}$ | M | | |
| 11001 | 11101 | 1 | | No add/sub | Count=0 |
| 11100 | 11110 | 1 | | AShift A, Q, $Q_{-1}$ right | |

# Booth's Algorithm (-15 x -5) [+15=01111 -15=10001]

| Register | Multiplier Register | | Multiplicand Register | Operation | Remark |
|---|---|---|---|---|---|
| A | Q | $Q_{-1}$ | M | +5= 00101(-5= 11011) | |
| 00000 | 11011 | 0 | 10001 | Initially | Count=5 |
| | | | | | |
| 01111 | 11011 | 0 | | A=A-M=A+ 2s(M) | Count=4 |
| 00111 | 11101 | 1 | | AShift A, Q, $Q_{-1}$ right | |
| | | | | | |
| 00111 | 11101 | 1 | | No add/sub | Count=3 |
| 00011 | 11110 | 1 | | AShift A, Q, $Q_{-1}$ right | |
| | | | | | |
| 10100 | 11110 | 1 | | A=A+ M | Count=2 |
| 11010 | 01111 | 0 | | AShift A, Q, $Q_{-1}$ right | |
| | | | | | |
| 101001 | 01111 | 0 | | A=A-M=A+ 2s(M) | Count=1 |
| 00100 | 10111 | 1 | | AShift A, Q, $Q_{-1}$ right | |

# Booth's Algorithm (-15 x -5) [+15=01111 -15=10001]

| Register | Multiplier Register | | Multiplicand Register | Operation | Remark |
|---|---|---|---|---|---|
| A | Q | $Q_{-1}$ | M | | |
| 00100 | 10111 | 1 | | No add/sub | Count=0 |
| 00010 | 01011 | 1 | | AShift A, Q, $Q_{-1}$ right | |

# Analysis of Booth's Algorithm

1 0 0 1 1 1 0 0 1 0 1 0 1 0 0

**-1 0 +1 0 0 -1 0 +1 -1 +1 -1 +1 -1 0 0**

The general case.

1 0 1 0 1 0 1 0 1 0 1 0 1 0 1

**-1 +1 -1+1 -1 +1 -1 +1 -1 +1 -1 +1 -1 +1-1**

The **Worst** case.

1 1 1 1 1 0 0 0 0 0 1 1 1 1 1

**0 0 0 0 -1 0 0 0 0 +1 0 0 0 0 -1**

The **Best** case.

# Bit Pair Recoding of Multipliers

**1** 1 1 0 1 0 **0**

**0 0 -1 +1 -1 0**

**0 -1 -2**

Bit Pair recoding halves the maximum number of summands .

# Bit-Pair Recoding of Multipliers

| Multiplier bit pair | | Multiplier bit on the right i-1 | Version of multiplicand selected by $i^{th}$ bit |
|---|---|---|---|
| Bit $_{i+1}$ | Bit $_{i}$ | | |
| 0 | 0 | 0 | 0 X M |
| 0 | 0 | 1 | +1 X M |
| 0 | 1 | 0 | +1 X M |
| 0 | 1 | 1 | +2 X M |
| 1 | 0 | 0 | -2 X M |
| 1 | 0 | 1 | -1 X M |
| 1 | 1 | 0 | -1 X M |
| 1 | 1 | 1 | 0 X M |

# Bit-Pair Recoding of Multipliers

+11 x -9

+11 = 01011, -11= 10101

+9 = 01001

-9 = 10111

Bit pair recorded multiplier will be

**1** 1 0 1 1 1 **0**

-1 +2 -1

```
                0 1 0 1 1
               -1  +2  -1
  ------------------------------
      1 1 1 1 1 1 0 1 0 1
      0 0 0 1 0 1 1 0
      1 1 0 1 0 1
  ------------------------------
    1 1 1 1 0 0 1 1 1 0 1
```

**Ans. 2's comp(**1 1 1 0 0 1 1 1 0 1 )
= 0 0 0 1 1 0 0 0 1 1
= -99

X (-2)= Take the 2's comp n then shift left by 1 position

X (-1)= Take the 2's comp of the operand

X (2)= Shift the operand by 1 bit position to the left

# Bit-Pair Recoding of Multipliers

+18 x -11

+18 = 010010, -18= 101110

+11 = 001011

-11 = 110101

Bit pair recorded multiplier will be

1 1 0 1 0 1 **0**

-1 +1 +1

```
              0 1 0 0   1 0
                -1  +1    +1
-----------------------------
  0 0 0 0 0 0 0  1 0 0 1  0
  0 0 0 0 0  1 0  0 1 0
  1 1 1 0 1  1 1  0
-----------------------------
  1 1 1 1 0 0 1  1 1 0 1  0
```

**Ans. 2's comp(**1 1 1 1 0 0 1 1 1 0 1 0)

= 0 0 0 0 1 1 0 0 0 1 1 0

= 198

# Division of Positive Integers
# Restoring Algorithm

# Restoring Division

Initialization:

A=0

Q= Dividend

M= Divisor

n= No of bits in the operand

Note: Both Q and M are positive integers represented using equal no of bits.
In A one extra bit is considered to keep track of the sign of the result of subtraction operation.

Step 1: Shift A and Q left one binary position.

Step 2: Subtract M from A, and place the answer back in A

If the sign of A is 1, set $q_0$ to 0 and add M back to A (restore A); otherwise, set $q_0$ to 1

Repeat these steps *n* times.

Result: Quotient: Q

Remainder : A

# Restoring Division Example (15 / 5)

| | | A | Q |
|---|---|---|---|
| Initially | 0 | 0 0 0 0 | 1 1 1 1 |
| M | 0 | 0 1 0 1 | -M= 1 1 0 1 1 |
| ShiftL | 0 | 0 0 0 1 | |
| Subtract | 1 | 1 0 1 1 | |
| | 1 | 1 1 0 0 | |
| Restore | 0 | 0 1 0 1 | |
| | 1 0 | 0 0 0 1 | |
| ShiftL | 0 | 0 0 1 1 | |
| Subtract | 1 | 1 0 1 1 | |
| | 1 | 1 1 1 0 | |
| Restore | 0 | 0 1 0 1 | |
| | 10 | 0 0 1 1 | |
| ShiftL | 0 | 0 1 1 1 | |

# Restoring Division Example (15 / 5)

| | | A | Q |
|---|---|---|---|
| | 0 | 0 1 1 1 | 1 0 0 |
| Subtract | 1 | 1 0 1 1 | 1 0 0 1 |
| | 1 0 | 0 0 1 0 | |
| ShiftL | 0 | 0 1 0 1 | 0 0 1 |
| Subtract | 1 | 1 0 1 1 | |
| | 10 | 0 0 0 0 | 0 0 1 1 |

Result:
Quotient: 0011
Remainder: 0000

# Non restoring Division

In case of restoring division, after unsuccessful division, M is added to A then shifted to left and M is subtracted from it.

A+M

2(A+M)

2(A+M) - M

2A+2M - M = 2A +M

i.e., Shift A to the left and ADD M directly to it.

# Non restoring Division

Initialization:
A=0
Q= Dividend
M= Divisor
n= No of bits in the operand

Note: Both Q and M are positive integers represented using equal no of bits.
In A one extra bit is considered to keep track of the sign of the result of subtraction operation.

Repeat these step <1> *n* times:

Step 1: If the sign of A is 0, shift A and Q left one bit position and subtract M from A; otherwise, shift A and Q left and add M to A.

Now, if the sign of A is 0, set $q_0$ to 1; otherwise, set $q_0$ to 0.

Step 2:If the sign of A is 1, add M to A

# Non restoring Division Example(15 / 5)

| | | A | Q |
|---|---|---|---|
| Initially | 0 | 0 0 0 0 | 1 1 1 1 |
| M | 0 | 0 1 0 1 | -M= 1 1 0 1 1 |
| ShiftL | 0 | 0 0 0 1 | 1 1 1 |
| Subtract | 1 | 1 0 1 1 | 1 1 1 |
| | 1 | 1 1 0 0 | 1 1 1 0 |
| ShiftL | 1 | 1 0 0 1 | 1 1 0 |
| Add | 0 | 0 1 0 1 | |
| | 1 | 1 1 1 0 | 1 1 0 0 |
| ShiftL | 1 | 1 1 0 1 | 1 0 0 |
| Add | 0 | 0 1 0 1 | |
| | 10 | 0 0 1 0 | 1 0 0 1 |
| ShiftL | 0 | 0 1 0 1 | 0 0 1 |
| Subtract | 1 | 1 0 1 1 | |
| | 10 | 0 0 0 0 | 0 0 1 1 |

Quotient= 0011=3

Remainder= 0000=0

# Floating Point Numbers IEEE Number Representation

# Why Floating Point Numbers?

The maximum value that can be represented using 32 bits is 4,294,967,295, is an integer equal to $2^{32} - 1$.

For scientific calculations, sometimes, we need very big number like $6.0247 \times 10^{23}$ or a very small nubmer like $9.1 \times 10^{-31}$

But for representing such numbers 32 bits are not sufficient enough, we need larger number of bits in our conventional methods.

**Solution:**

**IEEE 754 Standard: Single Precision(32 bits)**

**Double Precision(64 bits)**

# Introduction to Basic Terms

For a number $6.0247 \times 10^{23}$

It consists of 3 components:

**a. Mantissa: [Significant Digits] 6.0247**

**b. base: 10**

**c. Exponent : 23**

**d. Scale factor :$10^{23}$**

# IEEE Representation

IEEE Floating Point notation is the standard representation in use. There are two representations:

**- Single precision.**

**- Double precision.**


Fig. IEEE Single Precision


Fig. IEEE Double Precision

# Biased Exponent

Using 8 bits we can have exponent value in the range $-2^7$ to $+(2^7-1)$

Biased exponent says that we donot want our exponent as a signed number, we want it only as a positive number.

So to get only +ve values, we are using biased exponent, where we will add bias to the actual exponent and the result will be stored as the exponent of the number.

Say the exponent is -5, it will be stored as -5+ bias
Bias is represented as $2^{k-1}$ , if exponent is represented using k no of bits.[In general]

# Biased Exponent

In the IEEE Single precision, using 8 bits exponent can be from 0 to 255(using 8bits)

But 0 and 255 is used for representing special numbers.

So, we are going to store exponent in the range 1 to 254.

If if exponent is represented using k(8) no of bits, we are taking bias as $2^{k-1}-1$ , ($2^7-1=127$)

| Original Exponent | Biased (Excess-127) Exponent |
|---|---|
| -126 | 1 |
| -125 | 2 |
| .... | |
| ....... | |
| 127 | 254 |

# Implicit Normalized Number

Say a number is 1001.01

Then to perform implicit normalization we need to bring the decimal point to the left of 1st one in the number.

$1001.01 = 1.00101 \times 2^3$

# Steps to convert a given decimal number into IEEE Format

Step1: Convert the number into Binary.

Step 2: Normalize (Implicit) the number.
if decimal point is taken to the left by i positions, then multiply the number by $\mathbf{2^{i}}$
if decimal point is taken to the right by i positions, then multiply the number by $\mathbf{2^{-i}}$

Step 3: If the number is +ve S=0, Else S=1

Step 4: Add 127 to the exponent , then write the binary of it for E’

Step 5: Mantissa is whatever is to the right of the decimal point in the normalized number. If it is not of 23 bits, then append zeros to the right of the actual mantissa to make it of length 23 bits.

# Example : Convert -13.25 into IEEE Single Precision Format

Step1: Convert the number into Binary.

13.25 = 1101.01

Step 2: Normalize (Implicit) the number.

$1101.01 = 1.10101 \times 2^3$

Step 3: If the number is +ve S=0, Else S=1

Here, S=1

Step 4: Add 127 to the exponent , then write the binary of it for E'

E'=127+3=130= 10000010

Step 5: Mantissa is whatever is to the right of the decimal point in the normalized number. If it is not of 23 bits, then append zeros to the right of the actual mantissa to make it of length 23 bits.

M= 1010 1000 0000 0000 0000 000

Solution: S=1
E'=1000 0010
M= 1010 1000 0000 0000 0000 000

# Example : Convert -13.25 into IEEE Double Precision Format

Step1: Convert the number into Binary.

13.25 = 1101.01

Step 2: Normalize (Implicit) the number.

$1101.01 = 1.10101 \times 2^3$

Step 3: If the number is +ve S=0, Else S=1

Here, S=1

Step 4: Add 1023 to the exponent , then write the binary of it for E'

E'=1023+3=1026= 100 0000 0010

Step 5: Mantissa is whatever is to the right of the decimal point in the normalized number. If it is not of 52 bits, then append zeros to the right of the actual mantissa to make it of length 23 bits.

M= 1010 1000 0000 0000 0000 000.......0

Solution: S=1

E'=100 0000 0010

M= 1010 1000 0000 0000 0000 000.......0

# Example : Convert -17 into IEEE Format

Step1: Convert the number into Binary.

17 = 10001

Step 2: Normalize (Implicit) the number.

$10001.0 = 1.0001 \times 2^4$

Step 3: If the number is +ve S=0, Else S=1

Here, S=1

Step 4: Add 127 to the exponent , then write the binary of it for E'

E'=127+4=131= 10000011

Step 5: Mantissa is whatever is to the right of the decimal point in the normalized number. If it is not of 23 bits, then append zeros to the right of the actual mantissa to make it of length 23 bits.

M= 0001 0000 0000 0000 0000 000

Solution: S=1

E'=1000 0011

M= 0001 0000 0000 0000 0000 000

# Example : Convert -0.35 into IEEE Format

Step1: Convert the number into Binary.

0.35 = 0.01011

Step 2: Normalize (Implicit) the number.

0.01011 = 1.011 x $2^{-2}$

Step 3: If the number is +ve S=0, Else S=1

Here, S=1

Step 4: Add 127 to the exponent , then write the binary of it for E'

E'=127+-2=125= 0111 1101

Step 5: Mantissa is whatever is to the right of the decimal point in the normalized number. If it is not of 23 bits, then append zeros to the right of the actual mantissa to make it of length 23 bits.

M= 0110 0000 0000 0000 0000 000

Solution: S=1
E'= 0111 1101
M= 0110 0000 0000 0000 0000 000

# Value Represented

Value represented $=(-1)^{s} \times 1.M \times 2^{E'-127}$

What is the value represented by the following 32 bits in IEEE -754 REPRESENTATION?

S=0

E'= 10000011

M=11000....................00

Value represented $=(-1)^{0} X 1.11 \text{ X } 2^{131-127}$ [10000011=131]

$=+1.11 \text{ X } 2^{4}$

$=(11100)_{2}$

$=+(28)_{10}$

# Value Represented

Value represented $=(-1)^{s} X 1.M \; X \; 2^{E'-127}$

What is the value represented by the following 32 bits in IEEE -754 REPRESENTATION?

S=0

E'= 10000000

M=11000....................00

Value represented $=(-1)^{0} X 1.11 \; X \; 2^{128-127}$ [10000000=128]

$=+1.11 \; X \; 2^{1}$

$=(11.1)_{2}$

$=+(3.5)_{10}$

# Value Represented

Value represented $=(-1)^s X 1.M \text{ X } 2^{E'-127}$

What can be the maximum value represented by the 32 bits in IEEE -754 REPRESENTATION?

S=0

E'= 11111110

M=11111..................11

Value represented $=(-1)^0 \text{ x } 01.11..................1 \text{ x } 2^{254-127}$

$=1111................1 \text{ x } 2^{-23} \text{ x } 2^{127}$

$=(2^{24}-1) \text{ x } 2^{104}$

$=2^{128}$

# Value Represented

Value represented = $(-1)^{s} X 1.M X 2^{E'-127}$

What can be the minimum value represented by the 32 bits in IEEE -754 REPRESENTATION?

S=1

E'= 11111110

M=11111..................11

Value represented $=(-1)^{1} \times 01.11..................1 \times 2^{254-127}$

$=-1111................1 \times 2^{-23} \times 2^{127}$

$=-(2^{24}-1) \times 2^{104}$

$=-2^{128}$

# Value Represented

Value represented $=(-1)^{s} X 1.M \ X \ 2^{E'-127}$

What can be the minimum positive value represented by the 32 bits in IEEE - 754 REPRESENTATION?

S=0

E'= 00000001

M=0000.....0000

Value represented $=(-1)^{0} \times 1.00.................0 \times 2^{1-127}$

$=1 \times 2^{-126}$

$=2^{-126}$

# Value Represented

Value represented $=(-1)^{s} X 1.M \text{ X } 2^{E'-127}$

What can be the maximum negative value represented by the 32 bits in IEEE -754 REPRESENTATION?

S=1

E'= 00000001

M=0000.....0000

Value represented $= (-1)^{1} \text{ x } 1.00..................0 \text{ x } 2^{1-127}$

$= -1 \text{ x } 2^{-126}$

$= -2^{-126}$

# Denormalized Number

Denormalized Number: is a very very small number which cannot be normalized( implicit) [For single precision ]

Let say a number be, 0.0000000...............................11 [After decimal 131 places are there

After Implicit normalization =>

$1.1 \times 2^{-130}$

M=1

E'=-130+127=-3

In conventional metod this number cannot be stored as the exponent is -ve. But , in IEEE Standard, this will be treated as a special number.

# Denormalized Number

IEEE- 754 standard says that, for a small number normalize it till -126 bits.

Let say a number be, 0.0000000............................011 [After decimal 130 places are there

After Implicit normalization =>

$0.0011 \times 2^{-126}$

M=0011
E'=00000000

This is an example of denormalized number

Value formula for denormalized number : $\mathbf{(-1)^{s} \times 0.M \times 2^{-126(bias-1)}}$

# SPECIAL VALUES

| S | E’ | M | Number |
|---|---|---|---|
| 0 | 00000000 | 000.................0 | +0 |
| 1 | 00000000 | 000.................0 | -0 |
| 0 | 11111111 | 000.................0 | +∞ |
| 1 | 11111111 | 000.................0 | -∞ |
| 0/1 | 11111111 | M≠0 | NaN |
| 0/1 | 00000000 | M≠0 | Denormalized Number |
| **0/1** | **E’≠00000000 E’≠11111111** | **M=XXX......X** | **Implicit Normalized Number** |

# Floating point arithmetic: ADD/SUB rule

Choose the number with the smaller exponent.

Shift its mantissa right until the exponents of both the numbers are equal.

Add or subtract the mantissas.

Determine the sign of the result.

Normalize the result if necessary and truncate/round to the number of mantissa bits.

# Floating point arithmetic: MUL rule

Add the exponents.

Subtract the bias.

Multiply the mantissas and determine the sign of the result.

Normalize the result  (if necessary).

Truncate/round the mantissa of the result.

# Floating point arithmetic: DIV rule

Subtract the exponents

Add the bias.

Divide the mantissas and determine the sign of the result.

Normalize the result if necessary.

Truncate/round the mantissa of the result.

# Floating Point Arithmetic: ADD/SUB rule

Number 1:

S=0

E’= 1000 0100

M=000111100.....0

Number 2:

S=0

E’= 1000 0011

M=0100100...00000

Number 1:

$1.0001111 \times 2^5$

Number 2:

$1.01001 \times 2^4$

Choose the number with the smaller exponent. and shift its mantissa right until the exponents of both the numbers are equal. So number 2 will become

$0.1010010 \times 2^5$ and now perform the addition

$$
\begin{array}{l}
1.0\ 0\ 0\ 1\ 1\ 1\ 1 \times 2^5 \\
0.1\ 0\ 1\ 0\ 0\ 1\ 0 \times 2^5 \\
\hline
1.1\ 1\ 0\ 0\ 0\ 0\ 1 \times 2^5
\end{array}
$$

# Value Represented

Value represented $=(-1)^{s} X 0.M \ X \ 2^{-126}$

What is the value represented by the following 32 bits in IEEE -754 REPRESENTATION?

S=0

E'= 00000000

M=11000....................00≠0=>Denormalized number

Value represented $=(-1)^{0} X 0.11 \ X \ 2^{-126}$

$=11.0 \times 2^{-2} \times 2^{-126}$

$=3 \times 2^{-128}$

# GATE QUESTIONS

Consider three registers R1, R2, and R3 that store numbers in IEEE−754 single precision floating point format. Assume that R1 and R2 contain the values (in hexadecimal notation) 0x42200000 and 0xC1200000, respectively.

If R3 = R1 / R2, what is the value stored in R3 ?

(A) 0x40800000

(B) 0xC0800000

(C) 0x83400000

(D) 0xC8500000

R1= 0x42200000

R1= 0**100 0010 0010 0000 ........................0000**

R1, S=0

E'=10000100 = 132-127=5

M= 1.010000000......0

R1, $1.0100 \times 2^5$ = 101000=40

R2= 0xC1200000

R2= 1**100 0001 0010 0000 ........................0000**

R2, S=1

E'=1000 0010 = 130-127=3

M= 1.010000000......0

R2, $1.0100 \times 2^3$ = 1010=-10

So, R1/R2= 40/-10= -4

-4= $100.0 \times 2^0 = 1.00 \times 2^2$, S=1, E'=1000 0001, M=0000000.......0

1**100 0000 1**00.......... .....0= C0800000

# THANK YOU