COMPUTER SCIENCE

Computer Organization and Architecture

Floating Point Representation

Lecture_03

Vijay Agarwal sir

01 Floating Point Representation

02 IEEE 754 Floating Point Representation

Floating Point Representation.

S: Sign < 0 (+ve)
1 (-ve)

E or BE = e + bias

BE = AE + bias

S	E	m.

$bias = 2^{k-1}$

BE = AE + bias

$E = e + bias$

$e = E - bias$

Explicit

$0.1\ldots \times 2^{e}$

$(-1)^{S}\ 0.m \times 2^{e}$

$(-1)^{S}\ 0.m \times 2^{E-bias}$

Implicit

$1.nnn \times 2^{e}$

$(-1)^{S}\ 1.m \times 2^{e}$

$(-1)^{S}\ 1.m \times 2^{E-bias}$

$0.111 \Rightarrow 1 - \frac{1}{2^3}$

$0.1111 \Rightarrow 1 - \frac{1}{2^4}$

GP series
yesterday class.

(9) Max +ve value using Explicit

S	E	M
1bit	6bit	9bit

Conventional & Explicit then Max +ve Value.

$bias = 2^{6-1} = 2^5$

$bias = 32$

$E = 111111 \Rightarrow E = 63$

$(-1)^S \; 0.m \times 2^E$

$(-1)^S \; 0.m \times 2^{E-bias}$

$63 - 32$

$(-1)^0 \; 0.111111111 \times 2^{31}$

$+0.\underbrace{111111111}_{9\text{ times}} \times 2^{31}$

$111111111.0 \times 2^{-9} \times 2^{31}$

Left Alignment

m: 111111111

111111111×2^{22}

$(511)\,(2^9 - 1) \times 2^{22}$

$2^{9+22} - 2^{22}$

$+2^{31} - 2^{22}$

$= +2^{31}$ Any

Alternate

$$0.\underbrace{111111111}_{\Downarrow} \times 2^{31}$$

$$\left(1-\frac{1}{2^9}\right) \times 2^{31}$$

$$2^{31} - 2^{31} \times 2^{-9}$$

$$2^{31} - x = +2^{31}$$ Ans

Normalized Mantissa

Explicit Normalized

Syntax

$0.1\underline{\quad\quad} \times 2^{\underline{e}}$

M

Formula to get number

[value formula]

$(-1)^S \times 0.M \times 2^e$

$(-1)^S \times 0.M \times 2^{E\text{-bias}}$

Implicit Normalized

Syntax

$1.\underline{\quad\quad} \times 2^{\underline{e}}$

M

Formula to get number

[value formula]

$(-1)^S \times 1.M \times 2^e$

$(-1)^S \times 1.M \times 2^{E\text{-bias}}$

Explicit

0.1 After the point,

Immediate first bit should be 1

Example

(101.11)

0.10111×2^3

M = 10111,

e = 3

E = e + bias

Implicit

Before the point 1 means 1. .....

Example

(101.11)

1.0111×2^2

M = 0111,

e = 2

E = e + bias

Floating-Point Representation

S	E	M

S: sign bit
- 0 +ve
- 1 –ve

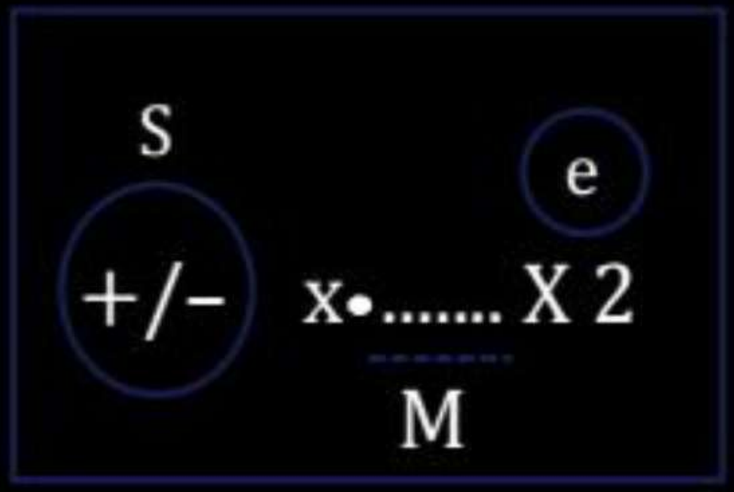
E:Biased exponent

M: Mantissa

$E = e + bias$

or

$BE = AE + bias$



$. ..... X2^{e}$

Q. $+21.75$

Implicit?

1 bit	7 bit	8 bit
S	E	M

10101.11

$1.010111 \times 2^{+4}$

$M = 010111$

$e = 4$, bias $= 2^{7-1}$ bias = 64

$E = 4 + 64$

$E = 68 = (1000100)_2$

Value Formula:

$(-1)^S \times 1.M \times 2^e$

$(-1)^0 \times 1.010111 \times 2^{68-64}$

1.010111×2^4

$10101.11 = (21.75)_{10}$

Ans

S(1bit)	E(7bit)	M(8 bit)
0	1000100	01011100

Hexadecimal $= (445C)_{16}$ Hexa Value

Q.

Consider a 16 bit register used to store floating point number. Mantissa is Explicit normalized signed fraction number. Exponent is in Excess-32 form then what is 16-bit for $-(29.75)_{10}$ in the register?

S(1bit) E(6bit) M(9bit

1	100101	1110111	00

-29.75

-11101.11×2^0

$0.1110111 \times 2^{+5}$

Sign = 1

m: 1110111

e = 5

bias = 32

$E = 5 + 32 = 37 \Rightarrow 100101$

$2^{k-1} = 2^5$

$k-1 = 5$

$k = 6$

Solution

1 bit	6 bit	9 bit
S	E	M

bias = 32.

-29.75

-11101.11

0.1110111×2^5

M: 1110111

e = 5

bias $= 2^{6-1}$

bias = 32

$E = 5 + 32 = 37 = (100101)_2$

S(1 bit)	E(6 bit)	M(9 bit)
1	100101	111011100

$(-1)^S \; 0.M \times 2^{E-bias} \Rightarrow (-1)^S \; 0.M \times 2^{37-32}$

$(-1)^1 \; 0.111011100 \times 2$

$-0.111011100 \times 2^{+5}$

-11101.1100

(-29.75) Ans

Q. $+21.75$

Implicit ?

10101.11

1.010111×2^4

$M = 010111$

$e = 4$, bias $= 2^{7-1}$

$E = 4 + 64$

$E = 68 = (1000100)_2$

1 bit	7 bit	8 bit
S	E	M

Value Formula:

$(-1)^S \times 1.M \times 2^e$

$(-1)^0 \times 1.010111 \times 2^{68-64}$

1.010111×2^4

$10101.11 = (21.75)_{10}$

Ans

S(1bit)	E(7bit)	M(8 bit)
0	1000100	01011100

Hexadecimal $= (445C)_{16}$

Convential Representation.

Can not Represent '0' & ∞.

$$\text{bias} = 2^{k-1} - 1$$

Implicit ⇒ 1.Something ⇒ 1.x ⇒ Not Represent '0'.

Explicit ⇒ 0.1..... 0.1 ⇒ Not Represent '0'.

In CN

IEEE 802.3 Ethernet

IEEE 802.5 Token Ring

IEEE 802.11 WiFi

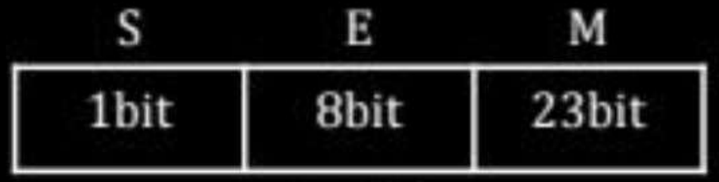
Disadvantage of Convential Representation

It can not store '0'
Explicit = 0.1
Implicit 1.0
It cannot Represent '∞'
[Infinity]
It can not represent number which is not normalized

Default is Implicit.

IEEE 754 Floating Point Representation

Single Precision (32 bit) Excess 127

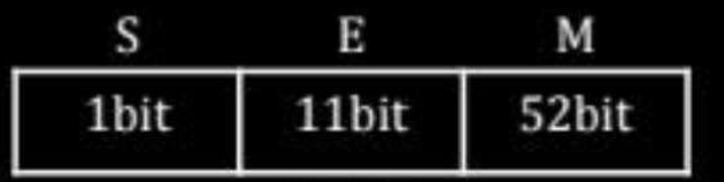


$bias = 2^{K-1} - 1$

$= 2^8 - 1$

Bias = 127

Double Precision (64 bit) Excess 1023

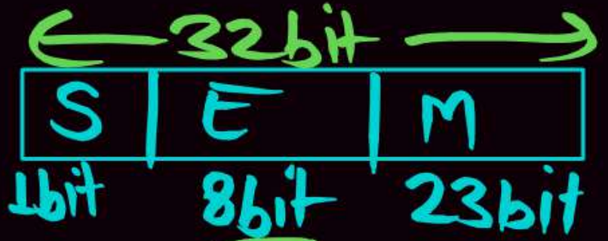


$bias = 2^{11-1} - 1$

Bias = 1023

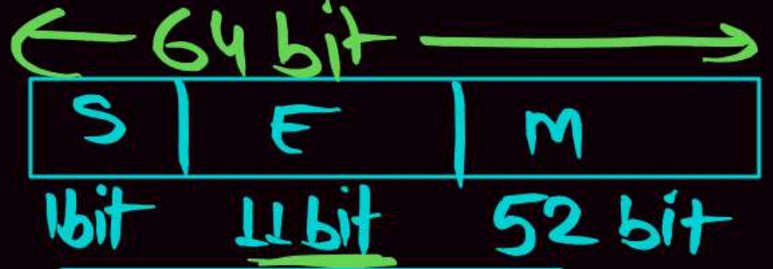
IEEE 754

Single Precision (32 bit)



$$bias = 2^{k-1} - 1$$

Double Precision (64 bit)



$$bias = 2^{k-1} - 1$$

S	E	m

1bit 8bit 23bit

8-1

Single Precision

$bias = 2^{8-1} - 1$

$bias = 127$

Excess − 127

S	E	m

1bit 11bit 52bit

Double Precision

$bias = 2^{11-1} - 1$

$bias = 1023$

Excess − 1023.

111

2^7 2^6 2^5 2^4 2^3 2^2 2^1 2^0

128 64 32 16 8 4 2 1

133: 1 0 0 0 0 1 0 1

A = 10

B = 11

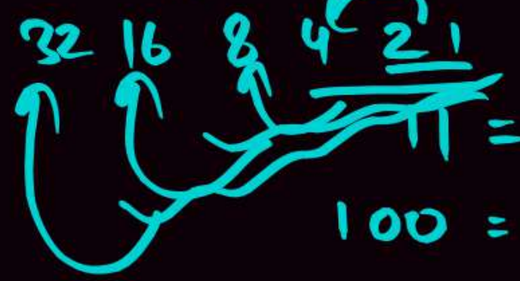
C = 12

D = 13

E = 14

F: 15

Next Term is
Add up all Previous term +1



2^7 2^6 2^5 2^4 2^3 2^2 2^1 2^0

128 64 32 16 8 4 2 1

133: 1 0 0 0 0 1 0 1

= 3

100 = 4

111 — 7

1000 — 8

1111 — 15

10000 — 16

11111 → 31

100000 → 32

Q. 1

How to represent +(119) into IEEE 754 single precision & Double precision floating Point Representation?

Implicit

Single Precision.

S	E	M

1bit 8bit 23bit

$bias = 2^{8-1} - 1 \Rightarrow bias = 127$

(+119)

$+1110111 \times 2^{0}$

$+1.1101111 \times 2^{+6}$

S=0

M=110 1111

e=+6

bias = 127

E=e+bias = 6+127

E=133

10000101

S(1bit)	E (8bit)	Mantissa (23 bit)
0	10000101	11011100 0000 0000 0000 000
	4 2	E E 0 0 0 0 0

(42EE0000)F Ans

Value formula

$(-1)^S \, 1.M \times 2^{e} \Rightarrow (-1)^S \, 1.M \times 2^{E-bias}$

E=133

bias=127

133−127

$(-1)^0 \, 1.1101111000000000000000000 \times 2^{+6}$

$1.1101111000000000000000000 \times 2^{6}$

$1110111.0000000000000...$

(119) Ans

1029

E ⇒ 11 bit

2^{10}	2^9	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
1024	512	256	128	64	32	16	8	4	2	1
1029 ⇒	1	0	0	0	0	0	0	0	1	0	1

A : 10
B : 11
C : 12
D : 13
E : 14
F : 15

Q. 1 How to represent +(119) into IEEE 754 single precision & Double precision floating Point Representation?

Double Precision.

S	E	m
1bit	11bit	52bit

$bias = 2^{11-1} - 1 \Rightarrow 1023$

bias = 1023

$+119$

$+1110111 \times 2^0$

$1.1101111 \times 2^{+6}$

S = 0

M = 1101111

e = +6

bias = 1023

E = 6 + 1023 = 1029

E = 10000000101

S	E (11 bit)	Mantissa (52 bit)
0	10000000101	1011100000000000000000000000000000000000000000000000

4 0 5 D C 0 0 0 0 0 0 0 0 0 0 0 0

(405DC0000000000000)$_{16}$ Ans

Q. 1 How to represent +(119) into IEEE 754 single precision & Double precision floating Point Representation?

bias = $2^{11-1} - 1 \Rightarrow 1023$

Double Precision

S	E	m
1bit	11bit	52bit

bias = 1023

$(-1)^S \; 1.m \times 2^{E}$ (E = 1029)

$E - bias$

$(-1)^S \; 1.m \times 2$

1024

1029 − 1023

S	E (11 bit)	Mantissa (52 bit)
0	10000000101	11011100000000000000000000000000000000000000000000

$(-1)^0 \; 1.11011100000000000000000 \times 2^{6}$

$+1.11011100000000000000000 \times 2^{6}$

$1110111.0000000000000 \times 2^{0}$

(119) Ans

4 0 5 D C 0 0 0 0 0 0 0 0 0 0 0 0

(405DC0000000000000)$_{16}$ Ans

E = 1029

Q. Consider a 32 bit register which stores floating numbers in IEEE single precision format(Implicit). The value of the number, f 32 bit are given below is _____

E=132 bias = 127

Sign(1bit)	Exponent(8bit)	Mantissa(23bit)
0	10000100 (128, 4)	11000000000000000000000

E-bias

$(-1)^S \ 1.M \times 2$

$(-1)^0 \ 1.11000000000000 \times 2^{132-127}$

$\Rightarrow +1.111000000000000 \times 2^{+5}$

$\Rightarrow 111000.0$

$= (56)$

A 48

B 56

C 64

D 60

Ans (B)

Solution

Sign(1bit)	Exponent(8bit)	Mantissa(23bit)
0	10000100	11000000000000000000000

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

$(-1)^0 (1.110.....) \times 2^{132-127}$

$(1.11) \times 2^5$

$= 111000$

$= 56$ Ans

$E = e + bias$

bias = 127

$E - bias$

$E = 10000100$

$E = 132$

Q. The value of float type variable is represented using the single precision 32 bit floating point IEEE 754 standard use 1 bit sign, 8bit for E and 23 bit mantissa. A float type variable X is assigned the decimal value (–14.25). Then representation of X in Hexa decimal notation is?

(A) 416C0000H

(B) 41640000H

(C) C16C0000H

(D) C1640000H

−14.25

−1110.01

$1.11001 \times 2^{+3}$

$e = +3$

bias = 127

$E = 130$

$3 + 127$

Ans (D)

Solution

1 bit	8 bit	23 bit
S	E	M

-14.25

1110.01

1.11001×2^3

$S = 1$

M: 11001

$e = 3$

$Bias = 2^{8-1} - 1 = 127$

$E = e + bias$

$= 3 + 127$

$E = 130 \Rightarrow 10000010$

S	E	M
1	10000010	11001 00000000000000

C 1 6 4 0 0 0 0

$Hexadecimal = (C1640000)_H$

IEEE 754 Floating Point Representation

Represent
① '0'
&
② ∞
&
③ NaN.
④ & etc

IEEE 754 Floating Point Representation

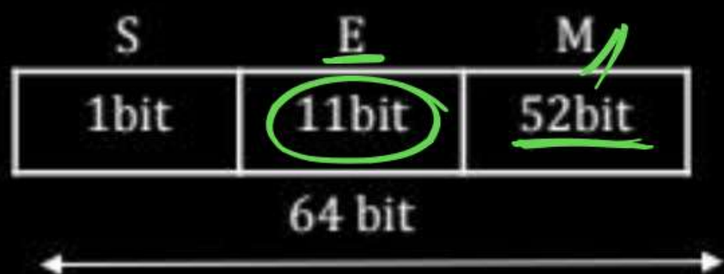
Single Precision
(32 bit)
Excess 127

$bias = 2^{K-1} - 1$

$= 2^{8} - 1$

Bias = 127

Double Precision
(64 bit)
Excess 1023



$bias = 2^{11-1} - 1$

Bias = 1023

In Convential Representation

$$bias = 2^{k-1}$$

But In IEEEE 754 Representation

$$bias = 2^{k-1} - 1$$

WHY? (P.T.O)

S	E	M

1bit 8bit 23bit

WHY bias $= 2^{k-1} - 1$. ?

(e) In Single Precision $= 2^{8-1} - 1 =$ bias $= 127$

Excess = 127

In Single Precision

Because If we take bias = 128 (2^{8-1}) (or) Excess 128, then there is a Chance of getting 'E' is 255. (When E=255 then either ∞ (or) Nan)

So Cannot Represent.

Bcz 8bit 2's Complement Range $= -2^{8-1}$ to $+2^{8-1}$

$\Rightarrow$ −128 to +127.

Assume if we have e=127

$E = e + \text{bias}$ (if bias 128)

$= 127 + 128 = 255$ ✗

So bias will be $2^{k-1} - 1$

1111 1111 ⇒ 255.

Single Precision (32 bit)

S	E	M
1 bit	8 bit	23 bit

$bias = 2^{8-1} - 1 \Rightarrow bias = 127$

V.V.V.V.Imp

	Sign(1 bit)	E(1 bit)	M(23 bit)	Value
①	0 or 1	00000000 E = 0	00000000000000000000000 M = 0	±0
②	0 or 1	11111111 E = 255	00000000000000000000000 M = 0	±∞ Infinity
③	0 or 1	1 ≤ E ≤ 254 1 to 254	M = Any	Implicit Normalized form $(-1)^S \times 1.M \times 2^e$ $(-1)^S \times 1.M \times 2^{E-127\ bias}$
	0 or 1	E = 0	M ≠ 0	Denormalized number/Fractional form $(-1)^S \times 0.M \times 2^{E-127\ bias}$
	0 or 1	E = 255	M ≠ 0	'Not a Number' (NAN)

Result

When E = 0

- M = 0 → (± 0)
- M ≠ 0 → (Fraction / Denormalized form)

When E = 255

- M = 0 → Infinite
- M ≠ 0 → NaN (Not a Number)

Double Precision

Excess - 1023

11111111111 = 2047

1-bit	11-bit	52-bit
S	E	M

$bias = 2^{11-1} - 1$

$bias \Rightarrow 1023$

Sign (1 bit)	E(11 bit)	M(52 bit)	Value
0 or 1	0000 0000 000 E = 0	00000000000.. M = 0	± 0
0 or 1	1111 1111 111 E = 2047	0000000000.. M = 0	± ∞ (Infinity)
0 or 1	$1 \le E \le 2046$	M = _ _ _ _ _ _ _ _ _ _	Implicit Normalization $(-1)^S \cdot M \times 2^e$ $(-1)^S \times 1 \cdot M \times 2^{E-1023}$
0 or 1	E = o	M ≠ 0	Denormalized number/ Fractional Form $(-1)^S 0 \cdot M \times 2^{E-1023}$
	E = 2047	M ≠ 0	Not a number

NOTE: When $E = 0$ then Value 0 when $M = 0$ or fractional form when $M \neq 0$

NOTE: When $E = 2047$ then Value ∞ when $M = 0$ or Not a Number(NAN) when $M \neq 0$

Note

Features IEEE 754 are special symbols to represent unusual events....

For example instead of interrupting on a divide by 0,softwarecan set the result to a bit pattern representing $+\infty$,- ∞ ; The largest exponent is reserved for these special symbols.

IEEE 754 has a symbols for the result of invalid operations, such 0/0, or subtract infinity from infinity . This symbols is NaN, for Not a Number.

The purpose of NaN is to allow programmer to postpone some test and decisions to a later time in this program. (when it is convenient)

Q. How to represent +(1.0) into IEEE 754 single precision floating Point Repartition?

1.0×2^0

$e = 0$

$bias = 127$

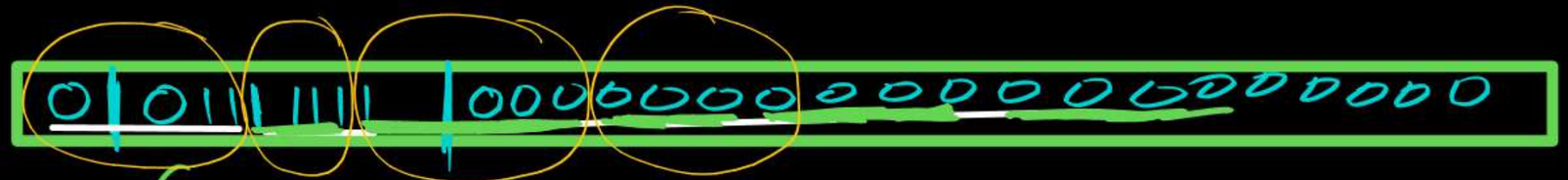
$E = e + bias = 0 + 127$

$E = 127$

$E = 01111111$

$M = 0$

$S = 0$



$(3F800000)_H$ Ans

$(-1)^S \; 1.M \times 2^{E-bias}$

$(-1)^0 \; 1.0000000 \times 2^{127-127} \Rightarrow +1.0000000 \times 2^0$

$= (1.0)$ Ans

Sol. To represent +(1.0) into IEEE 754 single precision floating Point Repartition..

$+ 1.0 \times 2^0$

$M = 0$

$e = 0$

$E = 127$

S	E	M
1	8	23

$Bias = 2^{8-1}-1 = 127$

$E = e + bias$

$E = 0 + 127$

0	0 1111111	0000 0000 0000 ....

3 F 8 0 0

(3F80 0000)

$2^3 - 1$

'7'

$\underline{111} \Rightarrow 1 - \frac{1}{2^3}$

1111 (4 bit) then $2^4 - 1$

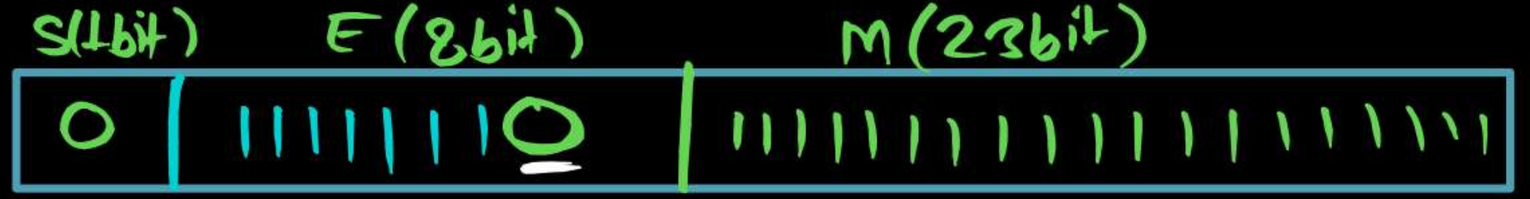
If 24 bit then $2^{24} - 1$

$\underbrace{111111111111111111111111}_{\text{24 times 1's}} \times 2^{104}$

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

$+ 2^{128}$ Ans

Q. What will be maximum positive vale(+ve) in IEEE 754 single precision floating Point Representation?



$e = E - bias$

$E - bias$

$S = 0$

$E = 11111110 = 254$

$bias = 127$

$Mantissa = 11111111111111111111111$

$(-1)^S \; 1 \cdot M \times 2$

$(-1)^0 \; 1.(11111111111111111111111) \times 2^{254-127}$

23 times

$1.(11111111111111111111111) \times 2^{127}$

23 times

$(2^{24}-1) \times 2^{104} \times 2^{-23} \times 2^{127}$

$2^{24+104} - 2^{104}$

$+2^{128}$ Ans

$127 - 23 = 104$

Q. 1

Consider a 64 bit Register which store floating point Number in IEEE 754 Double Precision Format.
What is the value of the Number if 64 bit are given as
1100 0000 0011 1101 1000 0000000000....

Q.4

Consider a 64 bit Register which store floating point Number in IEEE 754 Double Precision Format.
What is the value of the Number if 64 bit are given as
0111 1111 1111 0000 0000 0000000000....

Sol.२

Consider a 64 bit Register which store floating point Number in IEEE 754 Double Precision Format.

What is the value of the Number if 64 bit are given as
0111 1111 1111 0000 0000 0000000000....

S(1 bit)	E(11 bit)	M(52 bit)
0	1111 1111 111	0000000000000

Here all exponent bits is 1

E = 2047 → Infinity (when M = 0)

E = 2047 → Not a number (when M ≠ 0)

Here M = 0 so its $+\infty$

MCQ Q.3

The value of a *float* type variable is represented using the single-precision 32-bit floating point format of IEEE-754 standard that uses 1 bit for sign, 8 bits for biased exponent and 23 bits for mantissa. A *float* type variable X is assigned the decimal value of −14.25. The representation of X in hexadecimal notation is

[GATE-2014-Set2-CS: 2M]

A. C1640000H

B. 416C0000H

C. 41640000H

D. C16C0000H

MCQ Q.4

Consider the IEEE-754 single precision floating point numbers
P = 0xC1800000 and Q = 0x3F5C2EF4.
Which one of the following corresponds to the product of these numbers (i.e., P × Q), represented in the IEEE-754 single precision format?

[GATE-2023-CS: 2M]

A 0x404C2EF4

B 0x405C2EF4

C 0xC15C2EF4

D 0xC14C2EF4

Add/Subtract Rule

1. Choose the number with the smaller exponent and shift its mantissa right a number of steps equal to the difference in exponents.
2. Set the exponent of the result equal to the larger exponent.
3. Perform addition/subtraction on the mantissas and determine the sign of the result.

   Normalize the resulting value, if necessary.

Multiplication and division are somewhat easier than addition and subtraction, in that no alignment of mantissas is needed.

Multiply Rule

1. Add the exponents and subtract 127.
2. Multiply the mantissas and determine the sign of the result.
3. Normalize the resulting value, if necessary.

Divide Rule

1. Subtract the exponents and add 127.
2. Divide the mantissas and determine the sign of the result.
3. Normalize the resulting value, if necessary.

The addition or subtraction of 127 in the multiply and divide rules results from using the excess –127 notation for exponents.

1.1

1+(.1)

eg 1.11

(1.75)

eg 10.11

(2.75)

① 1.11 ⇒ 1+(.11)

1+(0.5 0.25)

(1.75)

② 10.11

10 + 2^{-1} 2^{-2}

2 + .5 .25

(2.75)

Floating-Point Representation

Example:

Show the IEEE 754 binary representation of the number -0.75_{ten} in single and double precision.

Answer:

-0.75

$-(0.11)$

The number -0.75_{ten} is also.

$$-3/4_{ten} \text{ or } -3/2^2{}_{ten}$$

Single Precision

It is also represented by the binary fraction.

$$-0.11_{two}/2^2{}_{ten} \text{ or } -0.11_{two}$$

In scientific notation, the value is.

$$-0.11_{two} \times 2^0$$

And in normalized scientific notation, it is.

$$-1.1_{two} \times 2^{-1}$$

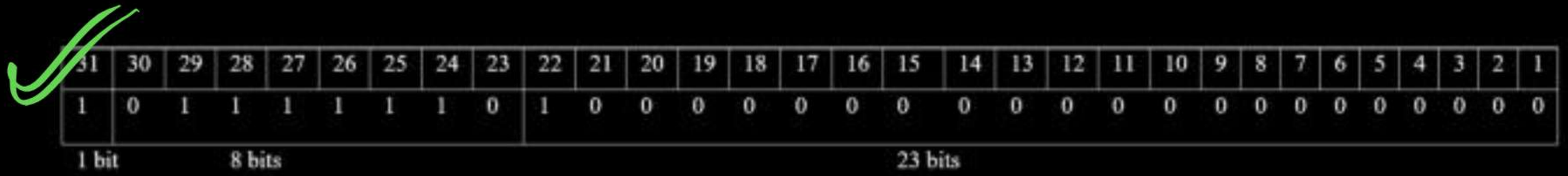
The general representation for a single precision number is.

$$(-1)^{S} \times (1 + \text{Fraction}) \times 2^{(\text{Exponent} - 127)}$$

When we subtract the bias 127 from the exponent of $-1.1_{two} \times 2^{-1}$, the result.

$$(-1)^{S} \times (1 + .1000\ 0000\ 0000\ 0000\ 0000\ 000_{two}) \times 2^{(126 - 127)}$$

The single precision binary representation of -0.75_{ten} is then



31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
1	0	1	1	1	1	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

1 bit | 8 bits | 23 bits

Floating-Point Addition

Let's add numbers in scientific notation by hand to illustrate the problems in floating-point addition:

$$9.999_{ten} \times 10^{1} + 1.610_{ten} \times 10^{-1}.$$

Assume that we can store only four decimal digits of the significand and two decimal digits of the exponent.

Step 1. To be able to add these numbers properly, we must align the decimal point of the number that has the smaller exponent. Hence, we need a form of the smaller number, $1.610_{ten} \times 10^{-1}$, that matches the larger exponent.

We obtain this by observing that there are multiple representations of an unnormalized floating-point number in scientific notation:

$$1.610_{ten} \times 10^{-1} = 0.1610_{ten} \times 10^{0} = 0.01610_{ten} \times 10^{1}$$

The number on the right is the version we desire, since its exponent matches the exponent of the larger number, $9.999_{ten} \times 10^1$. Thus, the first step shifts the significand of the smaller number to the right until its corrected exponent matches that of the larger number. But we can represent only four decimal digits so, after shifting, the number is really:

$$0.016_{ten} \times 10^1$$

Step 2. Next comes the addition of the significands:

$$\begin{array}{r} 9.999_{ten} \\ + \underline{0.016_{ten}} \\ \underline{10.015_{ten}} \end{array}$$

The sum is $10.015_{ten} \times 10^1$.

Step 3.

:This sum is not in normalized scientific notation, so we need to adjust it

$$10.015_{ten} \times 10^{1} = 1.0015_{ten} \times 10^{2}$$

Thus, after the addition we may have to shift the sum to put it into normalized form, adjusting the exponent appropriately. This example shows shifting to the right, but if one number were positive and the other were negative, it would be possible for the sum to have many leading 0s, requiring left shifts. Whenever the exponent is increased or decreased, we must check for overflow or underflow—that is, we must make sure that the exponent still fits in its field.

Step 4.

Since we assumed that the significand can be only four digits long (excluding the sign), we must round the number. In our grammar school algorithm, the rules truncate the number if the digit to the right of the desired point is between 0 and 4 and add 1 to the digit if the number to the right is between 5 and 9. The number

$$1.0015_{ten} \times 10^2$$

Is rounded to four digits in the significant to

$$1.002_{ten} \times 10^2$$

since the fourth digit to the right of the decimal point was between 5 and 9. Notice that if we have bad luck on rounding, such as adding 1 to a string of 9s, the sum may no longer be normalized, and we would need to perform step 3 again.

Steps 1 and 2 are similar to the example just discussed: adjust the significand of the number with the smaller exponent and then add the two significands. Step 3 normalizes the results, forcing a check for overflow or underflow. The test for overflow and underflow in step 3 depends on the precision of the operands. Recall that the pattern of all zero bits in the exponent is reserved and used for the floating-point representation of zero. Also, the pattern of all one bits in the exponent is reserved for indicating values and situations outside the scope of normal floating-point numbers . Thus, for single precision, the maximum exponent is 127, and the minimum exponent is –126. The limits for double precision are 1023 and –1022.

Decimal Floating-Point Addition

Example:

Try adding the numbers 0.5_{ten} and -0.4375_{ten} in binary using the algorithm in Figure:

Answer:

Let's first look at the binary version of the two numbers in normalized scientific notation, assuming that we keep 4 bits of precision:

$$0.5_{ten} = 1/2_{ten} = 1/2^{1}_{ten}$$

$$= 0.1_{two} = 0.1_{two} \times 2^{0} = 1.000_{two} \times 2^{-1}$$

$$-0.4375_{ten} = -7/16_{ten} = -7/2^{4}_{ten}$$

$$= -0.0111_{two} = -0.0111_{two} \times 2^{0} = -1.110_{two} \times 2^{-2}$$

Now we follow the algorithm:

Step 1. The significand of the number with the lesser exponent ($-1.11_{two} \times 2^{-2}$) is shifted right until its exponent matches the larger number:

$$-1.110_{two} \times 2^{-2} = -0.111_{two} \times 2^{-1}$$

Step 2. Add the significands:

$$1.000_{two} \times 2^{-1} + (-0.111_{two} \times 2^{-1}) = 0.001_{two} \times 2^{-1}$$

Step 3. Normalization the sum, checking for overflow or underflow:

$$0.001_{two} \times 2^{-1} = 0.010_{two} \times 2^{-2} = 0.100_{two} \times 2^{-3} = 1.000_{two} \times 2^{-4}$$

Since $127 \geq -4 \geq -126$, there is no overflow or underflow. (The biased exponent would be $-4 + 127$, or 123, which is between 1 and 254, the smallest and largest unreserved biased exponents.

Step 4. Round the sum:

$$1.000_{two} \times 2^{-4}$$

The sum already fits exactly in 4 bits, so there is no change to the bits due to rounding.

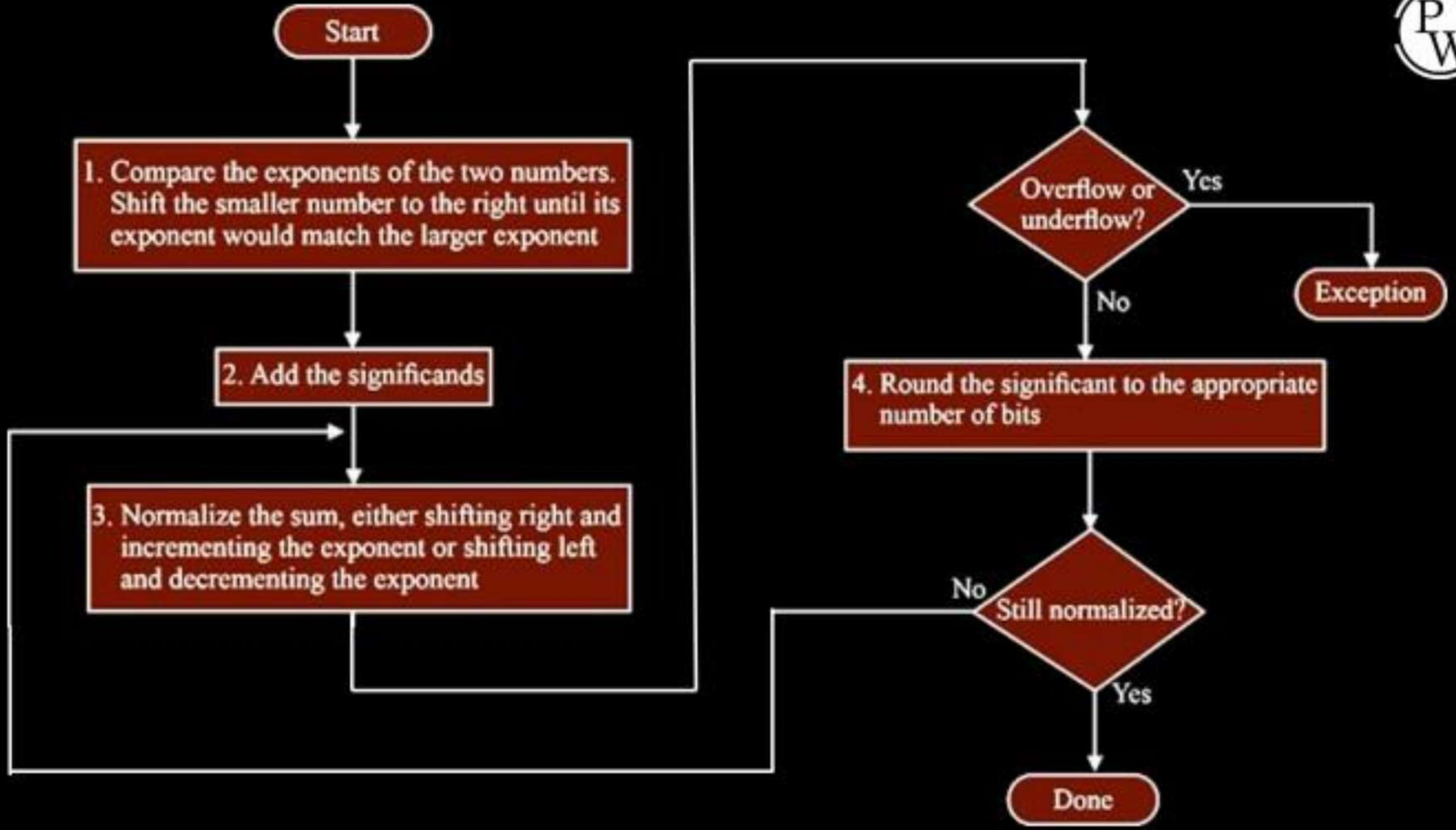
This sum is then

$$1.000_{two} \times 2^{-4} = 0.0001000_{two} = 0.0001_{two}$$

$$= 1/2^{4}_{ten} = 1/16_{ten} = 0.0625_{ten}$$

This sum is what we would expect from adding 0.5_{ten} to -0.4375_{ten}.

Many computers dedicate hardware to run floating-point operations as fast as possible. Figure sketches the basic organization of hardware for floating-point addition.


Start
1. Compare the exponents of the two numbers. Shift the smaller number to the right until its exponent would match the larger exponent
2. Add the significands
3. Normalize the sum, either shifting right and incrementing the exponent or shifting left and decrementing the exponent
Overflow or underflow?
Yes
Exception
No
4. Round the significant to the appropriate number of bits
No
Still normalized?
Yes
Done

Floating-point addition. The normal path is to execute steps 3 and 4 once, but if rounding causes the sum to be abnormalized, we must repeat step 3.

Floating-Point Multiplication

Now that we have explained floating-point addition, let's try floating-point multiplication. We start by multiplying decimal numbers in scientific notation by hand: $1.110_{ten} \times 10^{10} \times 9.200_{ten} \times 10^{-5}$. Assume that we can store only four digits of the significand and two digits of the exponent. Step 1. Unlike addition, we calculate the exponent of the product by simply adding the exponents of the operands together:

Step1. Unlike addition, we calculate the exponent of the product by simply adding the exponents of the operands together:

New exponent = 10 + (–5) = 5

Let's do this with the biased exponents as well to make sure we obtain the same result:

10 + 127 = 137, and –5 + 127 = 122, so

New exponent = 137 + 122 = 259

This result is too large for the 8-bit exponent field, so something is amiss! The problem is with the bias because we are adding the biases as well as the exponents:

New exponent = (10 + 127) + (–5 +127) = (5 + 2 × 127) = 259

Accordingly, to get the correct biased sum when we add biased numbers, we must subtract the bias from the sum:

New exponent = 137 + 122 – 127 = 259 – 127 = 132 = (5 + 127) and 5 is indeed the exponent we calculated initially.

Step 2. Next comes the multiplication of the significands:

$$
\begin{array}{r}
1.110_{ten} \\
\times \underline{9.200_{ten}} \\
0000 \\
0000 \\
2220 \\
\underline{9990} \\
10212000_{ten}
\end{array}
$$

There are three digits to the right of the decimal for each operand, so the decimal point is placed six digits from the right in the product significand:

$$10.212000_{ten}$$

Assuming that we can keep only three digits to the right of the decimal point, the product is 10.212×10^5.

Step 3. This product is unnormalized, so we need to normalize it:

$$10.212_{ten} \times 10^5 = 1.0212_{ten} \times 10^6$$

Thus, after multiplication, the product can be shifted right one digit to put it in normalized form, adding 1 to the exponent. At this point, we can check for overflow and underflow. Underflow may occur if both operands are small—that is, if both have large negative exponents.

Step 4. We assumed that the significand is only four digits long (excluding the sign), so we must round the number.

The number $1.0212_{ten} \times 10^6$

is rounded to four digits in the significand to

$1.021_{ten} \times 10^6$

Step 5. The sign of the product depends on the signs of the original operands. they are both the same, the sign is positive; otherwise, it's negative. Hence the product is

$$+1.021_{ten} \times 10^{6}$$

The sign of the sum in the addition algorithm was determined by addition of the significands, but in multiplication the sign of the product is determined by the signs of the operands.

Once again, as Figure shows, multiplication of binary floating-point numbers is quite similar to the steps we have just completed. We start with calculating the new exponent of the product by adding the biased exponents, being sure to subtract one bias to get the proper result. Next is multiplication of significands, followed by an optional normalization step. The size of the exponent is checked for overflow or underflow, and then the product is rounded. If rounding leads to further normalization, we once again check for exponent size. Finally, set the sign bit to 1 if the signs of the operands were different (negative product) or to 0 if they were the same (positive product).

Decimal Floating-Point Multiplication

Example:

Let's try multiplying the numbers 0.5_{ten} and -0.4375_{ten}, using the steps in Figure.

Answer:

In binary, the task is multiplying $1.000_{two} \times 2^{-1}$ by $-1.110_{two} \times 2^{-2}$.

Step 1. Adding the exponents without bias:

$$-1 + (-2) = -3$$

or, using the biased representation:

$$(-1 + 127) + (-2 + 127) - 127 = (-1 - 2) + (127 + 127 - 127)$$
$$= -3 + 127 = 124$$

Step 2. Multiplying the significands:

$$
\begin{array}{r}
1.000_{two} \\
\times\ \underline{1.110_{two}} \\
0000 \\
1000 \\
1000 \\
\underline{1000} \\
1110000_{two}
\end{array}
$$

The product is $1.1110000_{two} \times 2^{-3}$, but we need to keep it to 4 bits, so it is $1.110_{two} \times 2^{-3}$.

Step 3. Now we check the product to make sure it is normalized, and then check the exponent for overflow or underflow. The product is already normalized and, since $127 \geq -3 \geq -126$, there is no overflow or underflow. (Using the biased representation, $254 \geq 124 \geq 1$, so the exponent fits.)

Step 4. Rounding the product makes no change:

$$1.110_{two} \times 2^{-3}$$

Step 5. Since the signs of the original operands differ, make the sign of the product negative. Hence the product is

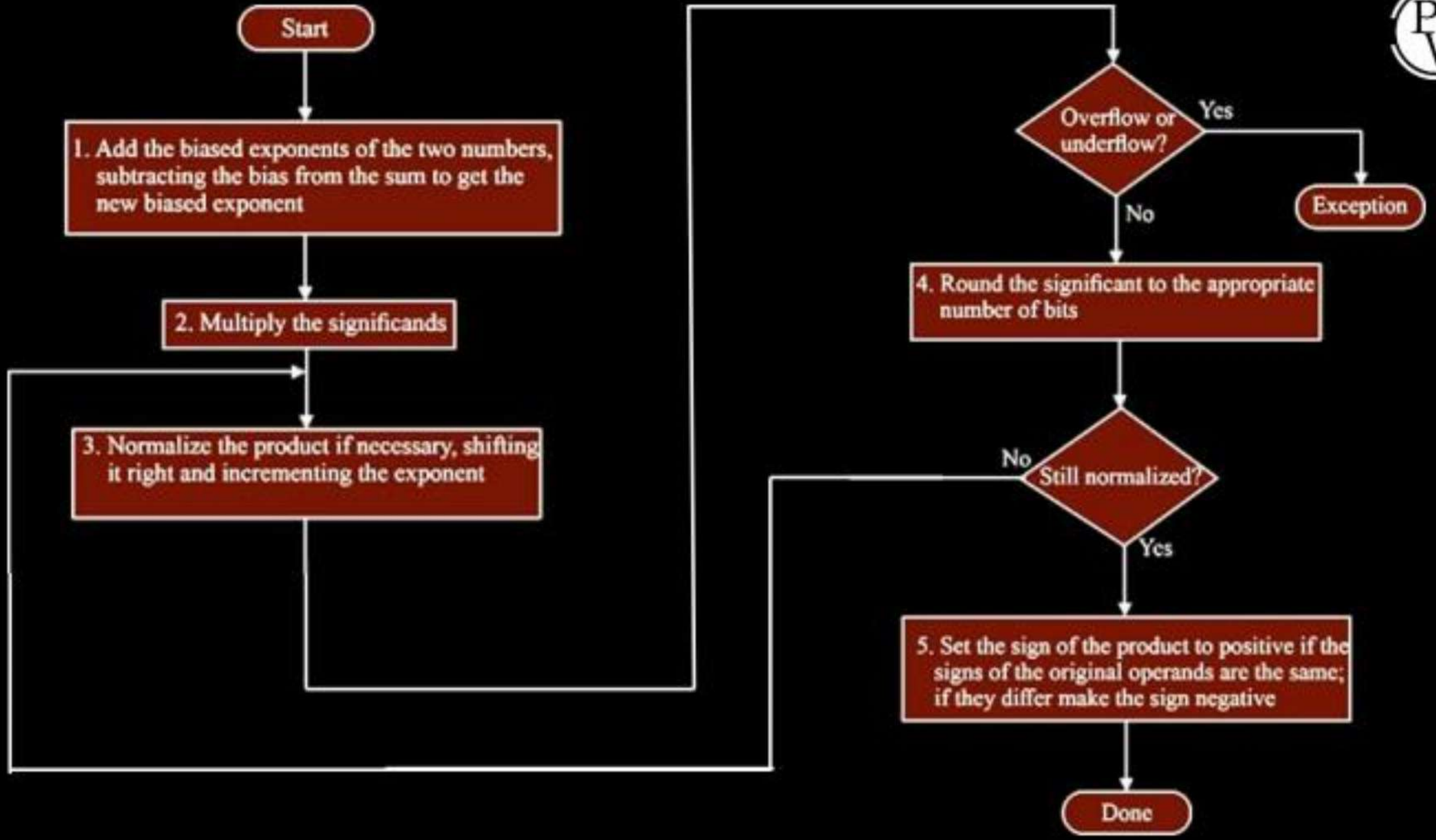
$$-1.110_{two} \times 2^{-3}$$

Converting to decimal to check our results:

$$-1.110_{two} \times 2^{-3} = -0.001110_{two} = -0.00111_{two}$$

$$= -7/2^{5}{}_{ten} = -7/32_{ten} = -0.21875_{ten}$$

The product of 0.5_{ten} and -0.4375_{ten} is indeed -0.21875_{ten}.


Start
1. Add the biased exponents of the two numbers, subtracting the bias from the sum to get the new biased exponent
2. Multiply the significands
3. Normalize the product if necessary, shifting it right and incrementing the exponent
Overflow or underflow?
Yes
Exception
No
4. Round the significant to the appropriate number of bits
No
Still normalized?
Yes
5. Set the sign of the product to positive if the signs of the original operands are the same; if they differ make the sign negative
Done

Floating multiplication:

The normal path is to execute steps 3 and 4 once, but if rounding causes the sum to be unnormalized, we must repeat step 3.

Arithmetic Operations on Floating-Point Number

In this section, we outing the general procedure for addition, subtraction, multiplication, and division of floating-point numbers. The rules we give apply to the single-precision IEEE standard format. These rules specify only the major steps needed to perform the four operations; for example, the possibility that overflow or underflow might occur is not discussed. Furthermore, intermediate results for both mantissas and exponents might require more than 24 and 8 bits, respectively. These and other aspects of the operations must be carefully considered in designing an arithmetic unit that meets the standard. Although we do not provide full details in specifying the rules, we consider some aspects of implementation, including rounding, in later sections.

If their exponents differ, the mantissas of floating-point numbers must be shifted with respect to each other before they are added or subtracted. Consider a decimal example in which we wish to add 2.9400×10^2 to 4.3100×10^4. We rewrite 2.9400×10^2 as 0.0294×10^4 and then perform additional of the mantissas to get 4.3394×10^4. The rule for addition and subtraction can be stated as follows:

Add/Subtract Rule

1. Choose the number with the smaller exponent and shift its mantissa right a number of steps equal to the difference in exponents.
2. Set the exponent of the result equal to the larger exponent.
3. Perform addition/subtraction on the mantissas and determine the sign of the result.

   Normalize the resulting value, if necessary.

Multiplication and division are somewhat easier than addition and subtraction, in that no alignment of mantissas is needed.

Multiply Rule

1. Add the exponents and subtract 127.
2. Multiply the mantissas and determine the sign of the result.
3. Normalize the resulting value, if necessary.

Divide Rule

1. Subtract the exponents and add 127.
2. Divide the mantissas and determine the sign of the result.
3. Normalize the resulting value, if necessary.

The addition or subtraction of 127 in the multiply and divide rules results from using the excess –127 notation for exponents.

THANK YOU!