Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

08 – Floating Point Numbers

CS1022 – Introduction to Computing II

Dr Jonathan Dukes / jdukes@scss.tcd.ie
School of Computer Science and Statistics

Binary Number Representation

32-bits 2^{32} unique values

e.g. unsigned integers

  $0 \ldots 2^{32}-1 = 0 \ldots 4{,}294{,}967{,}295$

e.g. signed integers using 2's complement

  $-2^{31} \ldots 0 \ldots +2^{31}-1 = -2{,}147{,}483{,}648 \ldots 0 \ldots +2{,}147{,}483{,}647$

How do we represent values like 3.14 or 2½?

How do we represent values with really large magnitudes?

  e.g. > 2,147,483,647

Think about (normalised) scientific notation …

Binary Floating-Point Numbers

Convert the following binary numbers to decimal numbers with fractions

10010101

1.1

101000.01

Convert the following decimal numbers to binary floating point numbers

10½

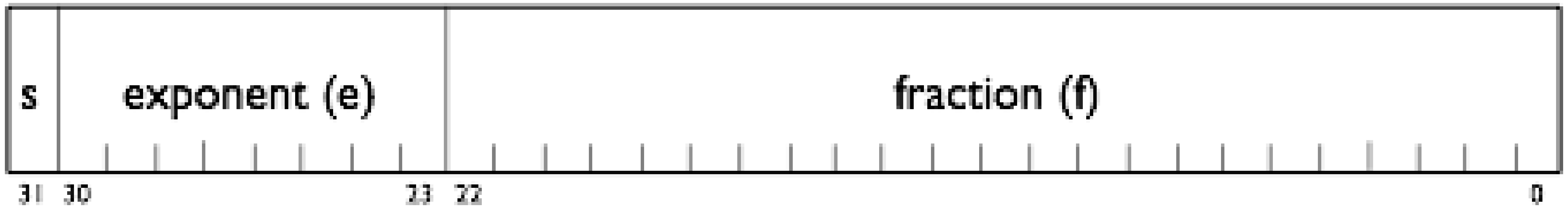
5¼

7.75

2.1

IEEE 754 Floating-Point Representation

Use a different interpretation of a 32-bit value to represent floating point numbers, e.g. IEEE 754



$$(-1)^s \times f \times 2^e$$

How can we represent …

… positive and negative values?

… values with positive and negative exponents?

Where is the radix point?

Sign bit?

0 ⇒ positive floating-point number

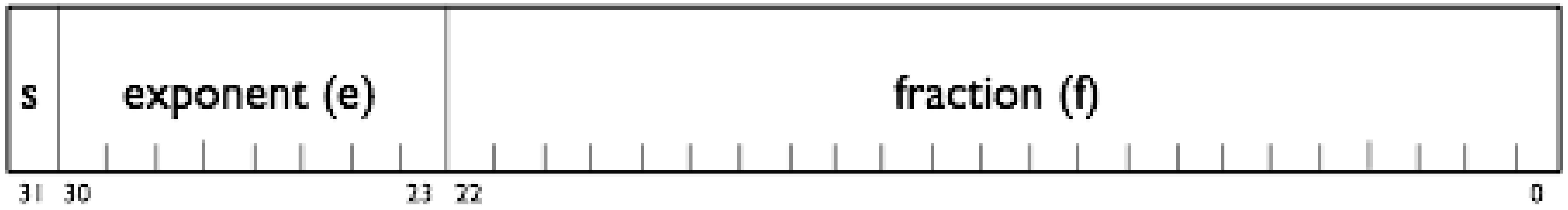
1 ⇒ negative floating-point number

Positive and negative exponents?

Option 1: 2’s Complement exponents

Option 2: Biased exponents

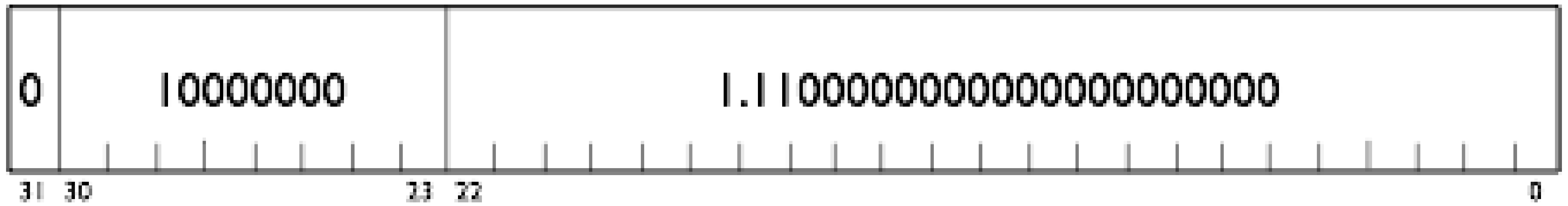
Subtract a constant bias (b) from stored exponent to obtain signed exponent



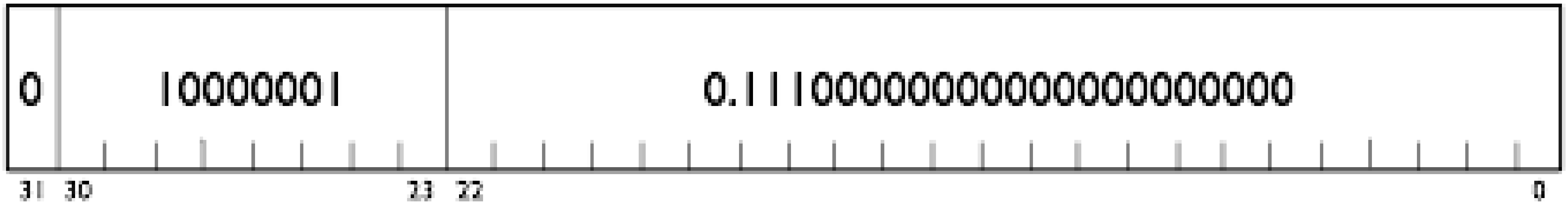
$$(-1)^s \times f \times 2^{e+(-b)}$$

Problem: Multiple Representations

Assume that the radix point is immediately after the LSB



$$+1.11 \times 2^{(+1)} = 1.11_2 = 1.75_{10}$$



$+0.111 \times 2^{(+2)} = 1.11_2 = 1.75_{10}$ (same value!)

Don’t want multiple representations of the same value! *(if (a == b) …)*

Store floating-point numbers in normalised form

$$1.ddd \ldots d$$

Hidden Bit

Normalisation

$$0.0101 \times 2^{-4}$$

... becomes ...

$$1.0100 \times 2^{-6}$$

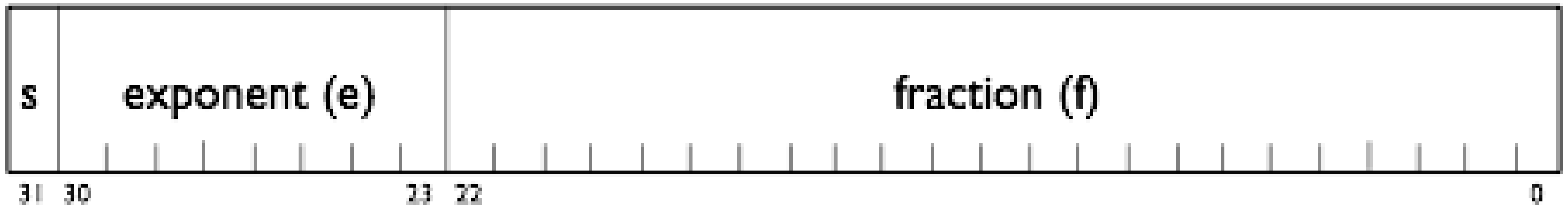
adjust fraction so there is a single 1 to left of radix point

compensate by adjusting exponent accordingly

If there is always going to be a 1 to the left of the radix point, we don't need to store it!

Increases precision (by one bit) – like not storing the 2 LSBs of a branch target offset!

Final IEEE 754 Floating-Point Representation

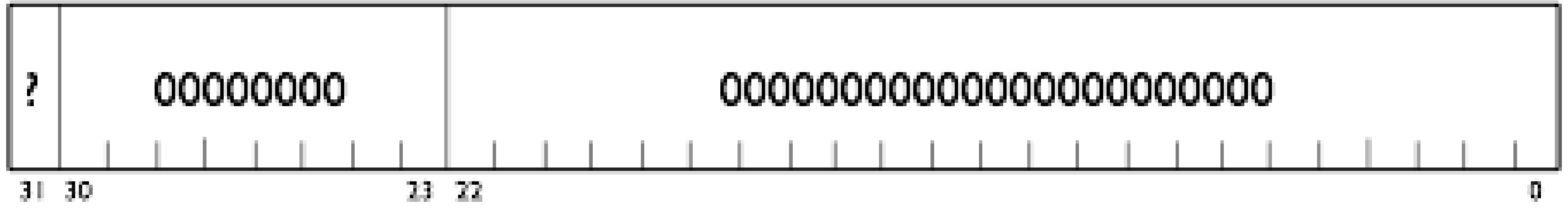


$$(-1)^s \times 1.f \times 2^{(e+b)}$$

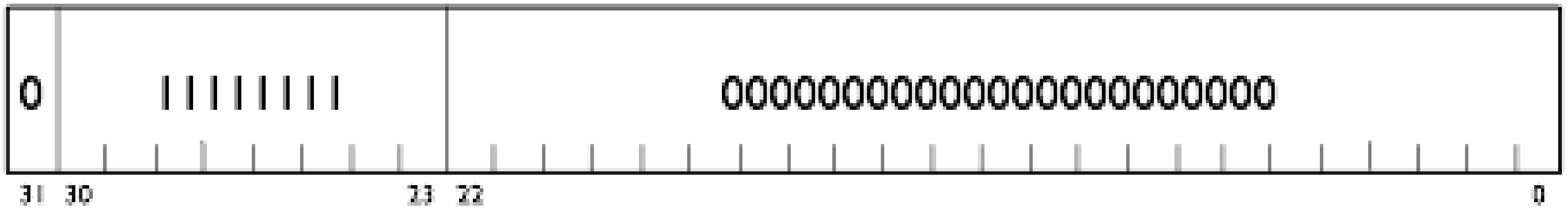
Examples?

Special bit patterns, e.g.

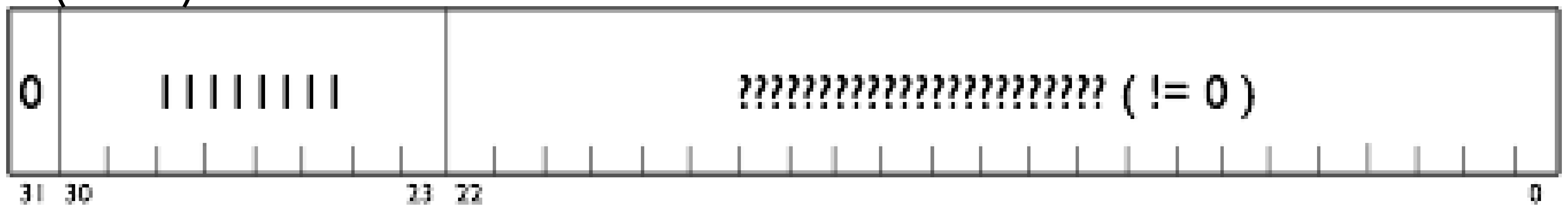
Zero (±)



Infinity (±)

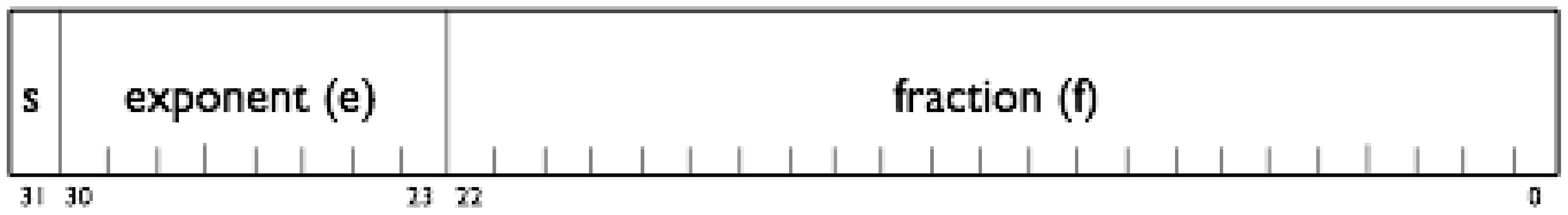


Not a Number (NaN)

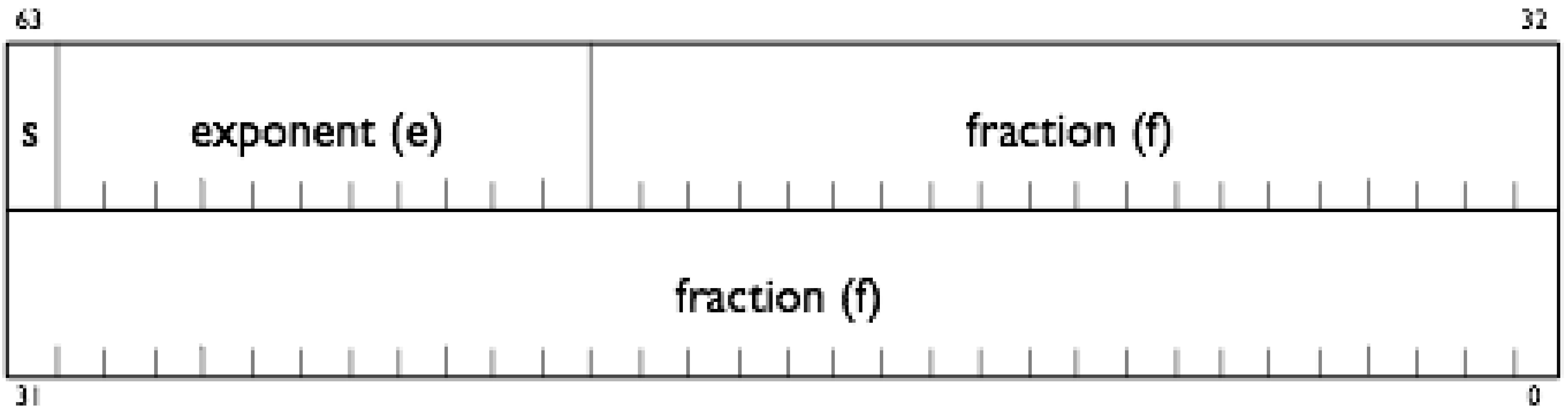


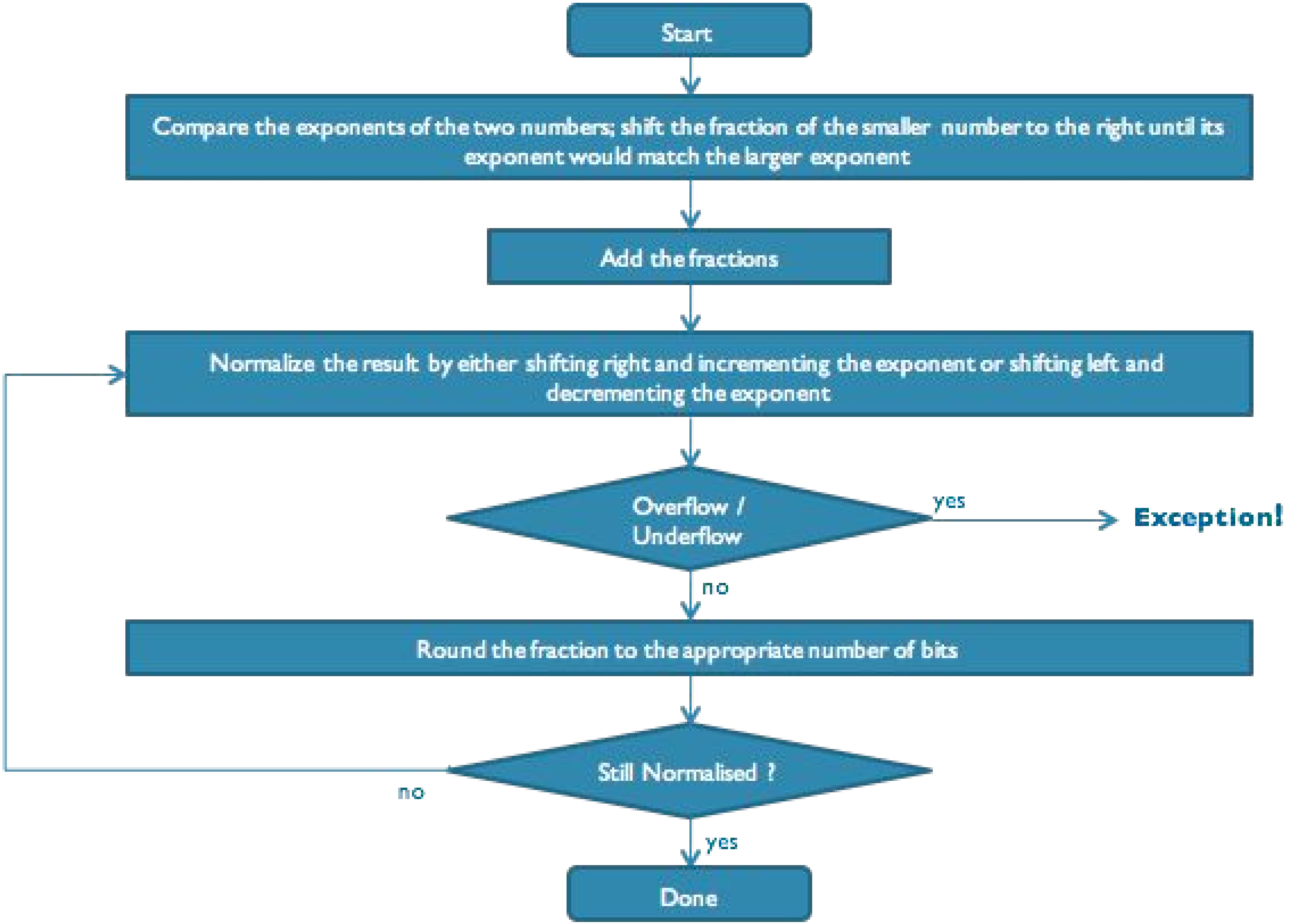
Single and Double Precision

32-Bit Single Precision



64-Bit Double Precision




Start
Compare the exponents of the two numbers; shift the fraction of the smaller number to the right until its exponent would match the larger exponent
Add the fractions
Normalize the result by either shifting right and incrementing the exponent or shifting left and decrementing the exponent
Overflow / Underflow
yes
Exception!
no
Round the fraction to the appropriate number of bits
Still Normalised ?
no
yes
Done