CS10003:
Programming & Data Structures

Dept. of Computer Science & Engineering
Indian Institute of Technology Kharagpur

Autumn 2020

Floating Point Representation

Number System : *The Basics*

- We are accustomed to using the so-called **decimal number system**
  - Ten digits :: 0,1,2,3,4,5,6,7,8,9
  - Every digit position has a weight which is a power of 10
  - **Base** or **radix** is 10

- Example:

  $234 = 2 \times 10^2 + 3 \times 10^1 + 4 \times 10^0$

  $250.67 = 2 \times 10^2 + 5 \times 10^1 + 0 \times 10^0 + 6 \times 10^{-1} + 7 \times 10^{-2}$

Floating-point Numbers

- The representations discussed so far applies only to integers.
  - Cannot represent numbers with fractional parts.
- We can assume a decimal point before a 2's complement number.
  - In that case, pure fractions (without integer parts) can be represented.
- We can also assume the decimal point somewhere in between.
  - This lacks flexibility.
  - Very large and very small numbers cannot be represented.

Floating Point Numbers (reals)

- To represent numbers like 0.5, 3.1415926, etc, we need to do something else. First, we need to represent them in binary, as

E.g. 11.00110 for 2+1+1/8+1/16=3.1875

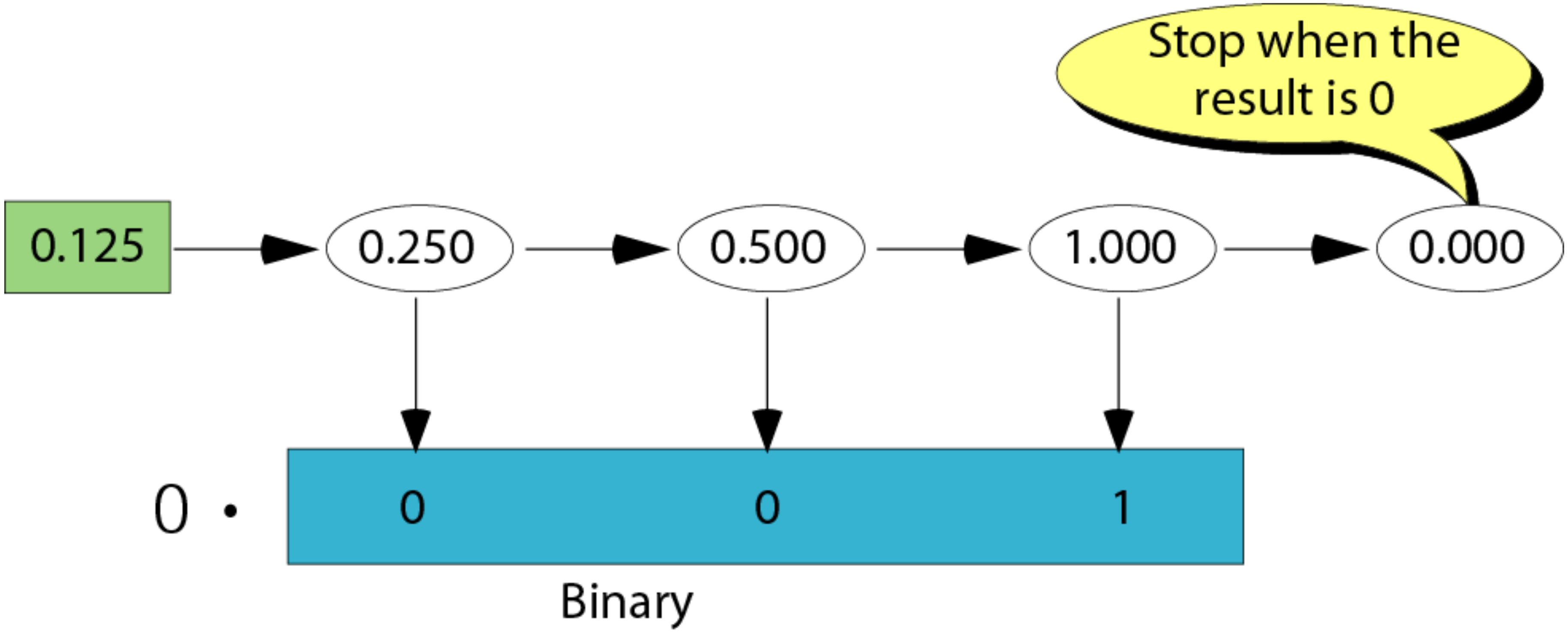
$$n = \cdots + a_m 2^m \cdots + a_2 2^2 + a_1 2 + a_0 + a_{-1} \times \frac{1}{2} + a_{-2} \times 2^{-2} + a_{-3} \times 2^{-3} + \cdots a_{-k} 2^{-k} + \cdots$$

- Next, we need to rewrite in scientific notation, as 1.100110×2^1. That is, the number will be written in the form:

  $1.xxxxxx\ldots \times 2^e$ x = 0 or 1

Changing fractions to binary

- Multiply the fraction by 2,...



Transform the fraction 0.875 to binary

Solution

Write the fraction at the left corner. Multiply the number continuously by 2 and extract the integer part as the binary digit. Stop when the number is 0.0.

0.875 ➔ 1.750 ➔ 1.5 ➔ 1.0 ➔ 0.0

0 . 1 1 1

Transform the fraction 0.4 to a binary of 6 bits.

Solution

Write the fraction at the left cornet. Multiply the number continuously by 2 and extract the integer part as the binary digit. You can never get the exact binary representation. Stop when you have 6 bits.

0.4 →	0.8 →	1.6 →	1.2 →	0.4 →	0.8 →	1.6
0 .	0	1	1	0	0	1

Normalization

- Sign, exponent, and mantissa

Example of normalization

Original Number	Move	Normalized
+1010001.1101	← 6	$+2^{6}$ x 1.01000111001
–111.000011	← 2	-2^{2} x 1.11000011
+0.00000111001	6 →	$+2^{-6}$ x 1.11001
–0.001110011	3 →	-2^{-3} x 1.110011

Fixed Point Representation

- Consists of a whole or integral part and a fractional part.
- The two parts are separated by a binary point.
- Suppose, there are k whole digits and l fractional digits, the value obtained is:
- $x = \sum_{i=-l}^{k-1} x_i 2^i = (x_{k-1} x_{k-2} \cdots x_0 x_{-1} x_{-2} \cdots x_{-l})_2$
- In a $(k + l)$ $-$bit representation, numbers from 0 to $2^k - 2^{-l}$ can be represented.
- Hence, k decides the range, and l decides the precision.
- As $k + l$ is constant, we have a tradeoff!

Floating Point

- Fixed point representations are hence not good for applications dealing with very large (needing a larger range), and extremely small numbers (and hence need precision) at the same time.
- Consider, the (8+8)-bit fixed point numbers:

$x = (0000\ 0000.0000\ 1001)_2$-- Small Number

$y = (1001\ 0000.0000\ 0000)_2$-- Large Number

The relative representation error due to truncation or rounding of digits beyond the 8th position is significant for x, but it is less severe for y.

On, the other hand, neither y^2 nor $\frac{y}{x}$ is representable in this format!

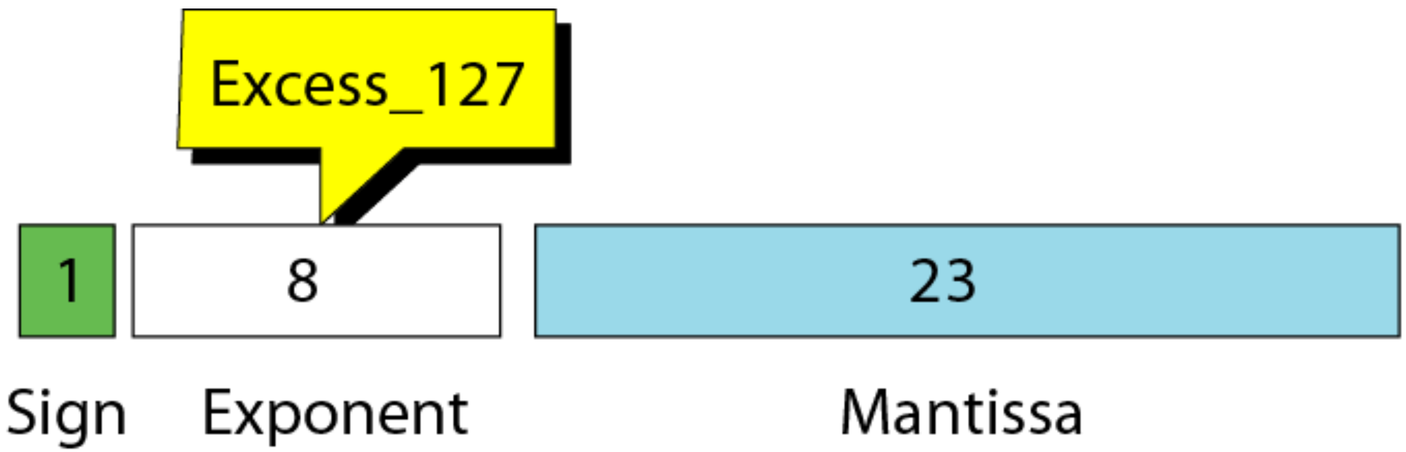
Floating point numbers address this issue, and is made of fixed point signed-magnitude number and an accompanying scale factor.

Normalized numbers in Single Precision Format

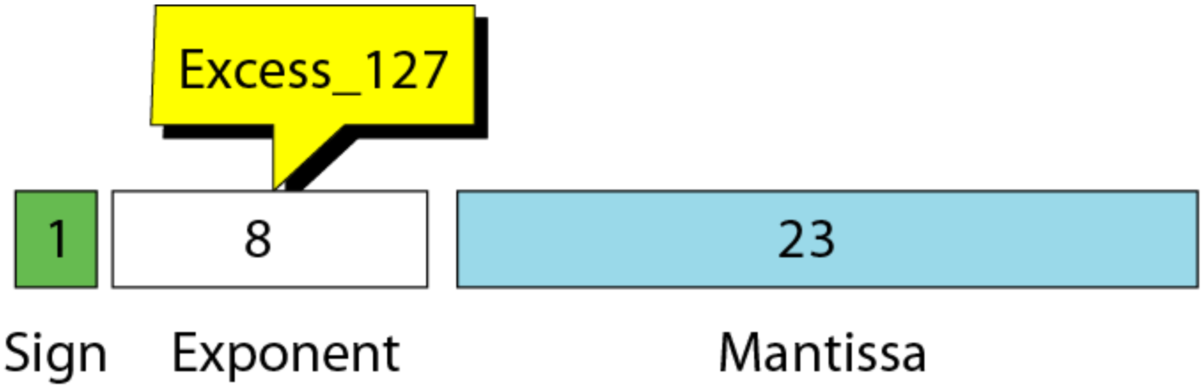
- The normalized numbers are:

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

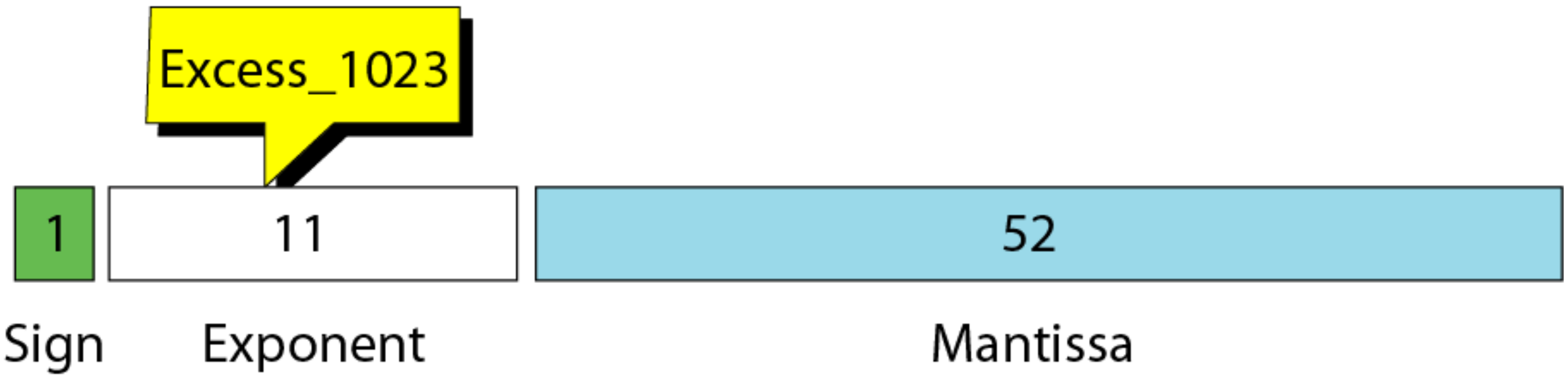
Here S is the sign bit, f is the Mantissa and E is the exponent.



IEEE standards for floating-point representation



a. Single Precision



b. Double Precision

Show the representation of the normalized number $+ 2^6 \times 1.01000111001$

Solution

The sign is positive. The Excess_127 representation of the exponent is 133. You add extra 0s on the right to make it 23 bits. The number in memory is stored as:

0 10000101 01000111001000000000000

Example of floating-point representation

Number	*Sign*	*Exponent*	Mantissa
$-2^{2} \times 1.11000011$	1	10000001	11000011000000000000000
$+2^{-6} \times 1.11001$	0	01111001	11001000000000000000000
$-2^{-3} \times 1.110011$	1	01111100	11001100000000000000000

Interpret the following 32-bit floating-point number

1 01111100 11001100000000000000000

Solution

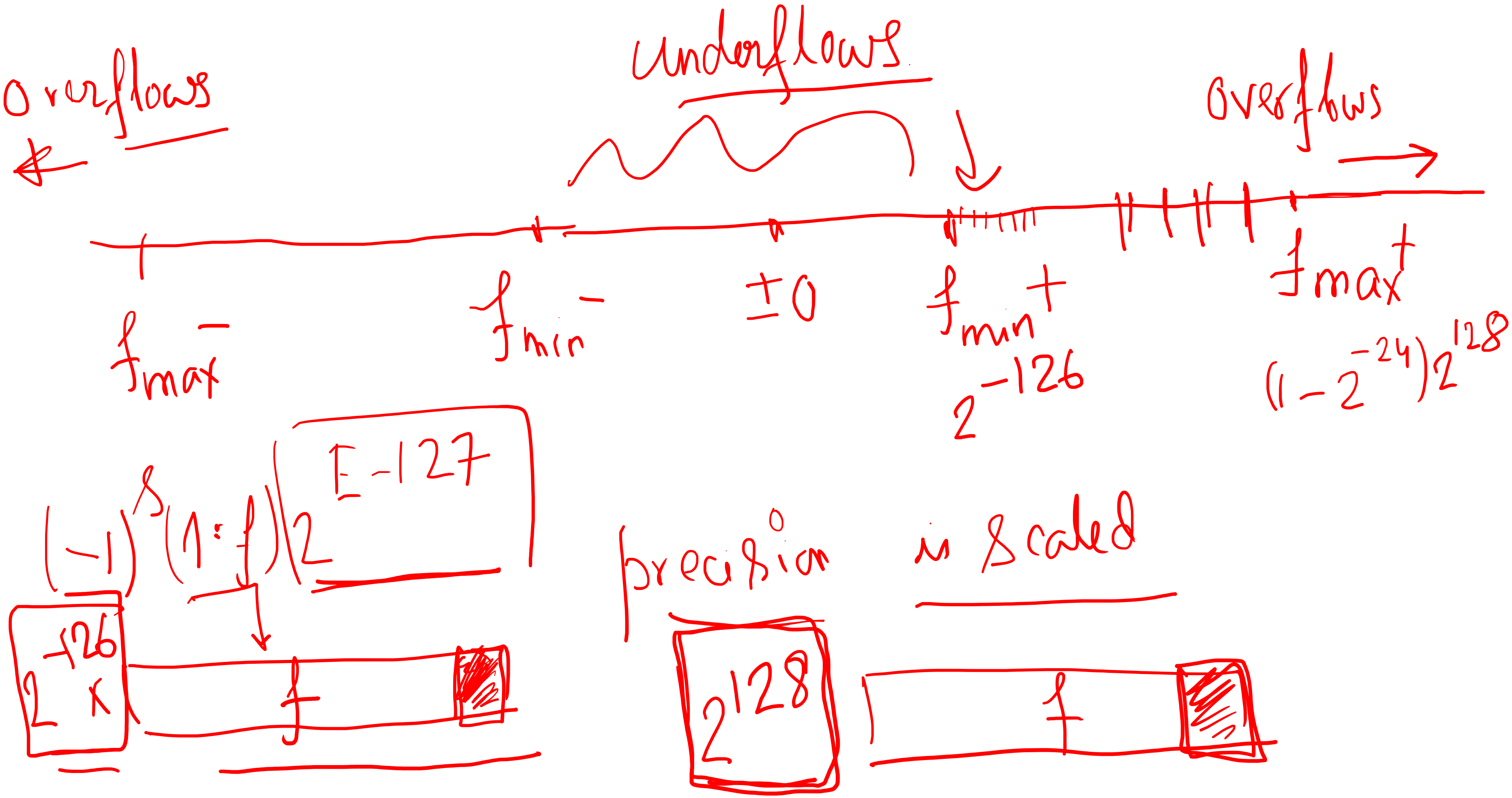
The sign is negative. The exponent is –3 (124 – 127). The number after normalization is

$$-2^{-3} \times 1.110011$$

Range of normalized numbers

- $f_{max}^{+} = (1.111\ldots1)2^{254-127}$
  - E=0 is reserved for zero (with f=0) and denormalized numbers (with f≠0).
  - E=255 is reserved for ±∞ (with f=0) and for NaN (Not a Number) (with f≠0).
- Thus, $f_{max}^{+}=(2-2^{-23})2^{127}=(1-2^{-24})2^{128}$.
- Similarly, $f_{min}^{+}=(1.0)2^{1-127}=2^{-126}$.
- The exponent bias and significand range were selected so that the reciprocal of all normalized numbers can be represented without overflow. (in particular f_{min}^{+}).

Floating Point Number Line



Thank You!