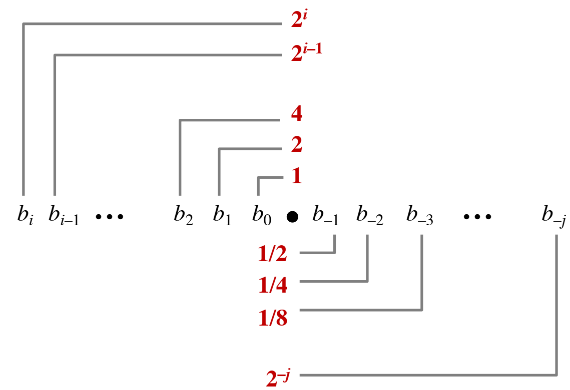
Floating-point numbers

Fractional binary numbers
IEEE floating-point standard
Floating-point operations and rounding
Lessons for programmers

Many more details we will skip (it's a 58-page standard…)
See CSAPP 2.4 for more detail.

Fractional Binary Numbers



$$\sum_{k=-j}^{i} b_k \cdot 2^k$$

Fractional Binary Numbers

Value **Representation**

5 and 3/4
2 and 7/8
47/64

Observations

Shift left =
Shift right =
Numbers of the form `0.111111`…$_2$ are…?

Limitations:

Exact representation possible when?

1/3 = 0.333333…$_{10}$ =

Fixed-Point Representation

Implied binary point.

b_7 b_6 b_5 b_4 b_3 [.] b_2 b_1 b_0
b_7 b_6 b_5 b_4 b_3 b_2 b_1 b_0 [.]

range: difference between largest and smallest representable numbers
precision: smallest difference between any two representable numbers

fixed point = fixed range, fixed precision

IEEE **Floating Point** Standard 754

IEEE = Institute of Electrical and Electronics Engineers

Numerical form:

$$V_{10} = (-1)^{s} * M * 2^{E}$$

Sign bit *s* determines whether number is negative or positive

Significand (mantissa) *M* usually a fractional value in range [1.0,2.0)

Exponent *E* weights value by a (-/+) power of two

Analogous to scientific notation

Representation:

MSB **s** = sign bit *s*

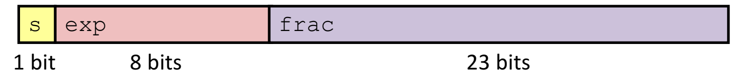
exp field encodes *E* (but is *not equal* to E)

frac field encodes *M* (but is *not equal* to M)

Numerically well-behaved, but hard to make fast in hardware

Precisions

Single precision `(float)`**: 32 bits**

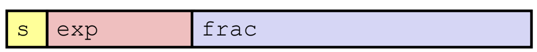


Double precision `(double)`**: 64 bits**

Finite representation of infinite range…

Three kinds of values

$V = (-1)^{s} * M * 2^{E}$



1. Normalized: *M* = 1.xxxxx…

As in scientific notation: $0.011 \times 2^{5} = 1.1 \times 2^{3}$

Representation advantage?

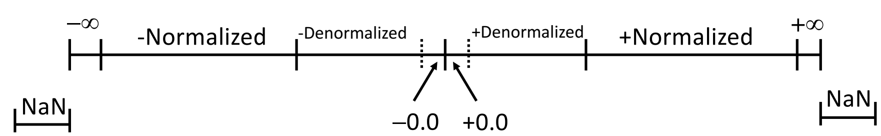
2. Denormalized, near zero: *M* = 0.xxxxx…, smallest *E*

Evenly space near zero.

3. Special values:

0.0:	**s** = 0	**exp** = 00…0	**frac** = 00…0
+inf, -inf: division by 0.0		**exp** = 11…1	**frac** = 00…0
NaN ("Not a Number"): sqrt(-1), $\infty - \infty$, $\infty * 0$, etc.		**exp** = 11…1	**frac** ≠ 00…0

Value distribution



Normalized values, with `float` example

$V = (-1)^s * M * 2^E$

s	exp	frac
	k=8	n=23

Value: `float f = 12345.0;`

12345_{10} = 11000000111001_2

= $1.1000000111001_2 \times 2^{13}$ (normalized form)

Significand:

M = `1.1000000111001`$_2$

`frac`= `10000001110010000000000`$_2$

Exponent: E = **exp** – Bias → **exp** = E + Bias

E = 13

Bias = 127 = $2^7 - 1$ = $\mathbf{2^{k-1} - 1}$ Splits exponents roughly -/+

exp = 140 = `10001100`$_2$

Result:

0	10001100	10000001110010000000000
s	exp	frac

2. Denormalized Values: near zero

"Near zero": exp = 000…0

Exponent:

E = **1 + exp** – Bias = 1 - Bias not: **exp** – Bias

Significand: leading zero

M = **0 . xxx…x$_2$**

frac = xxx…x

Cases:

exp = 000…0, **frac** = 000…0 0.0, -0.0

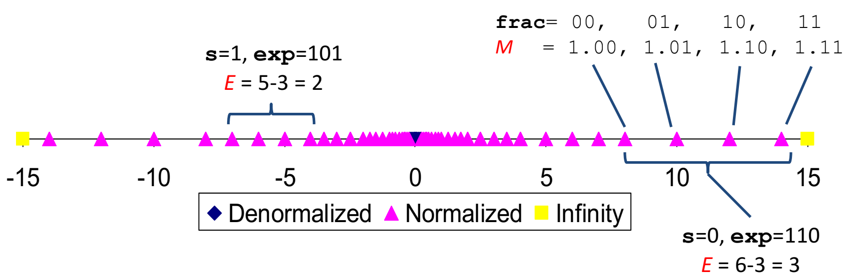
exp = 000…0, **frac** ≠ 000…0

Value distribution example

6-bit IEEE-like format

Bias = $2^{3-1} - 1 = 3$

s	exp	frac
1	3	2

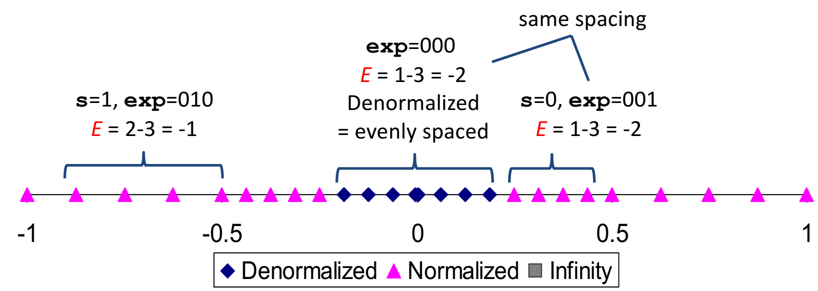


Value distribution example (zoom in on 0)

6-bit IEEE-like format

Bias = $2^{3-1} - 1 = 3$

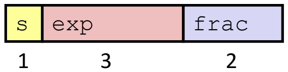
s	exp	frac
1	3	2



Try to represent 3.14, 6-bit example

6-bit IEEE-like format

Bias = $2^{3-1} - 1 = 3$



Value: `3.14;`

3.14 = 11.0010 0011 1101 0111 0000 1010 000…

= $1.1001\ 0001\ 1110\ 1011\ 1000\ 0101\ 0000\ldots_2 \times 2^1$ (normalized form)

Significand:

M = `1.10010001111010111011100001010000…` $_2$

`frac`= $\underline{10}_2$

Exponent:

E = 1 Bias = 3 **`exp`** = 4 = 100_2

Result:

`0 100 10` = $1.10_2 \times 2^1 = 3$ next highest?

Floating Point Arithmetic*

$V = (-1)^S * M * 2^E$


```
double x = ..., y = ...;
double z = x + y;
```

1. **Compute exact result.**
2. **Fix/Round**, roughly:

   Adjust *M* to fit in [1.0, 2.0)…

   If *M* >= 2.0: shift *M* right, increment *E*

   If *M* < 1.0: shift *M* left by k, decrement *E* by k

   Overflow to infinity if *E* is too wide for **`exp`**

   Round* *M* if too wide for **`frac`**.

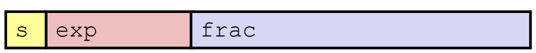
   Underflow if nearest representable value is 0.

   …

*complicated…

Lessons for programmers

$V = (-1)^S * M * 2^E$



`float` ≠ **real number** ≠ `double`

Rounding breaks associativity and other properties.

```
double a = ..., b = ...;
...
if (a == b) ...

if (abs(a – b) < epsilon) ...
```