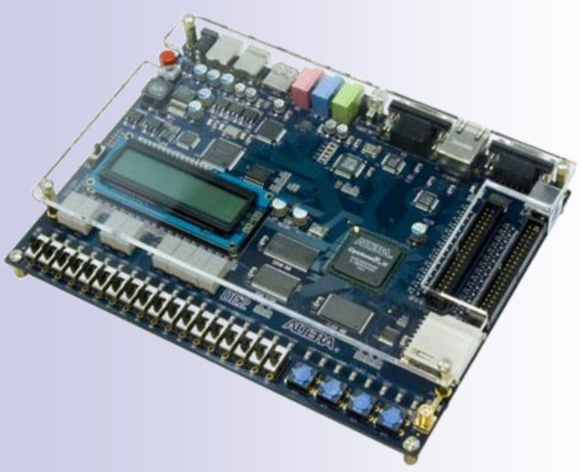
# Selection from

## Susta:Computer System Structures
## *& John Loomis: Computer organization*
## *& M.Mudawar:*Computer Architecture & Assembly Language

*Version: 1.0*

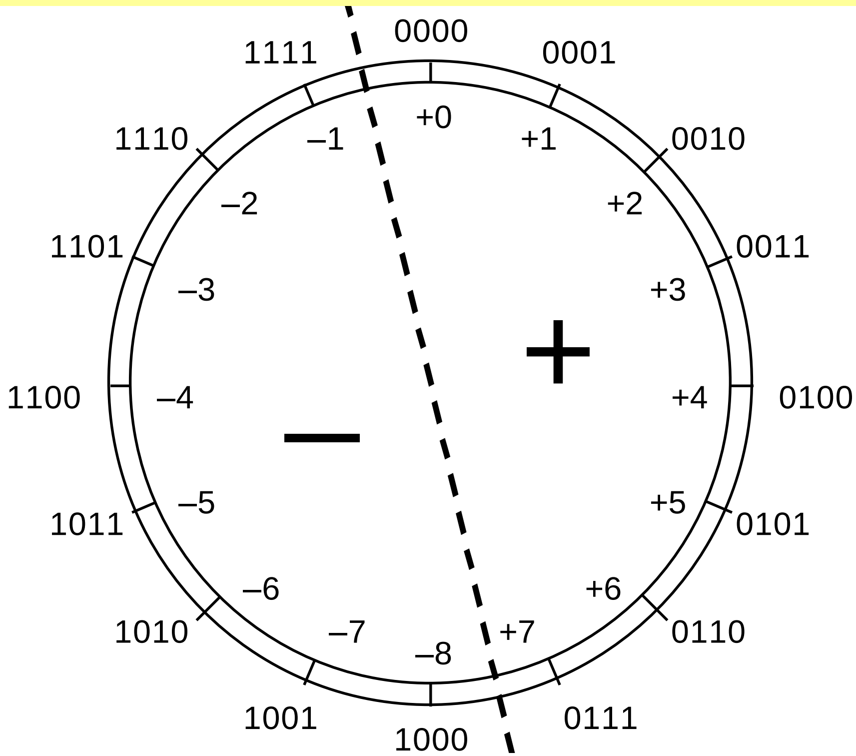


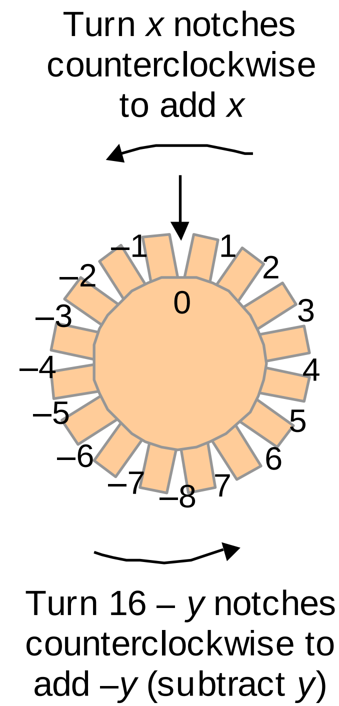
Practical Exercise 2nd

ČVUT-FEL in Prague,

# Two's-Complement Representation

With $k$ bits, numbers in the range $[-2^{k-1}, 2^{k-1} - 1]$ represented.
Negation is performed by inverting all bits and adding 1.

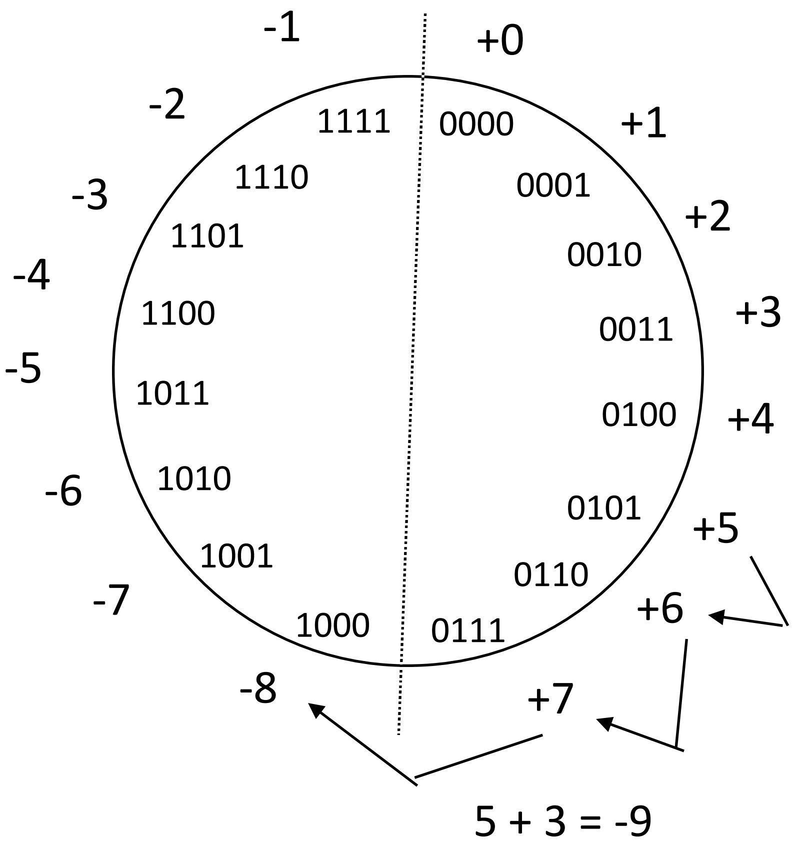


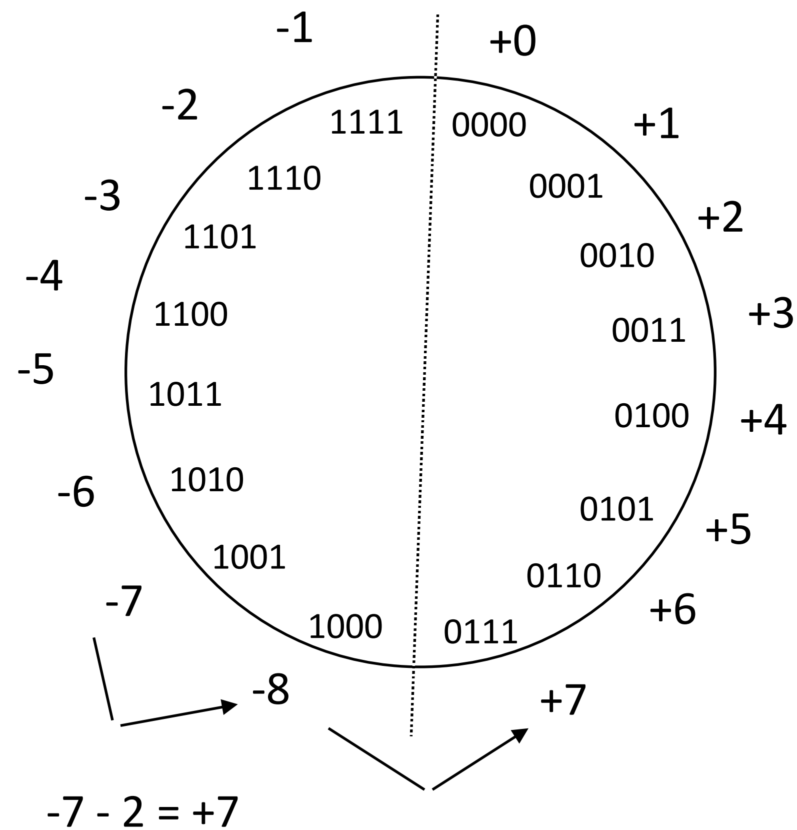



# Overflow Conditions

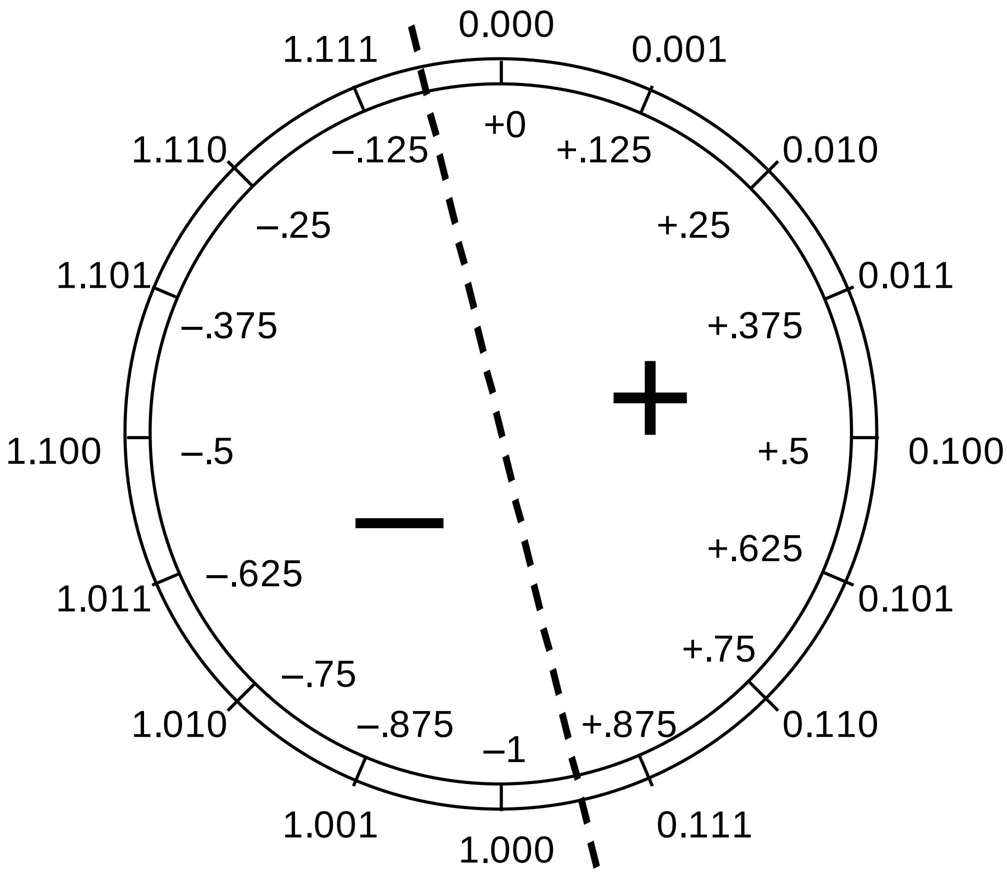
Add two positive numbers to get a negative number

or two negative numbers to get a positive number



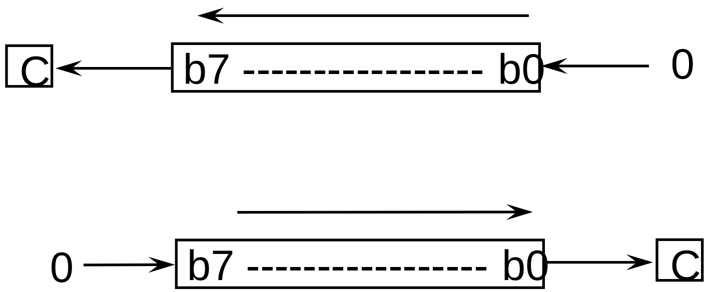



# Fixed-Point 2's-Complement Numbers

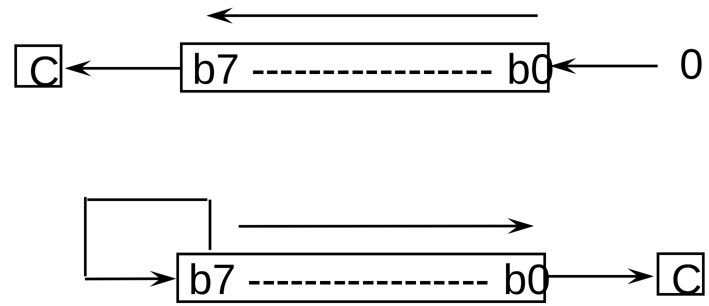



Schematic representation of 4-bit 2's-complement encoding for (1 + 3)-bit fixed-point numbers in the range [–1, +7/8].

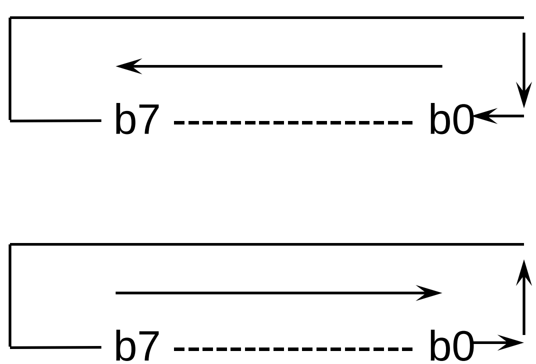
# Logical Shift




# Arithmetic Shift




# Rotation (Cyclic Shift)



# Mutliplication of signed numbers

Multiplication in Two's complement cannot be accomplished with

the standard technique since, as far as the machine itself is concerned, for Y[n]:

$-Y = 0 - Y = 2^n - Y$

since, when subtracting from zero, need to "borrow" from next column leftwards.

**Consider $X \times (-Y)$**

Internal manipulation of $-Y$ is as $2^n - Y$

Therefore $X \times (-Y) = X \times (2^n - Y) = 2^n \times X - X \times Y$,

it is correct as n-bit result, but it is wrong as 2*n bit result.

*A standard product of two n-bit numbers is 2*n-bit number,*

*thus we must calculate the result as 2*n-bit numbers!*

However as expected 2*n-bit result should be $2^{2n} - (X \times Y)$

# Mutliplication of signed numbers

**Consider (-X) × (−Y)**

Internal manipulation of −X is as $2^n - X$ and −Y is as $2^n - Y$

Therefore (-X) × (−Y) = $(2^n - X) \times (2^n - Y) = 2^{2n} - 2^n \times X - 2^n \times Y + X \times Y$,

The expected 2*n-bit result should be $2^{2n} + (X \times Y)$

We must calculate as 2*n-bit result and add a correction to obtain positive number.

*Note: Because negative numbers have many bit 1, computers usually utilize special algorithms for negative sign number multiplications., e.g. Booth's multiplication algorithm to increase speed.*

# Signed Multiplication

- Case 1: Positive Multiplier - we add as 8 bit numbers!

```
Multiplicand            1100₂ = -4
Multiplier         ×    0101₂ = +5
                   ---------------
Sign-extension      11111100
                    11110000
                   ---------------
Product             11101100₂ = -20
```

- Case 2: Negative Multiplier

```
Multiplicand            1100₂ = -4
Multiplier         ×    1101₂ = -3
                   ---------------
Sign-extension      11111100     -4          11111100   -4
                    11110000    -16          11110000  -16
(+4<<3)             00100000    +32          11100000  -32
                   ---------------          ---------------
Product          10|00001100₂ = +12       10|11001100 = -52
```

# Unsigned Division

$$
\begin{array}{rrll}
 & 10011_2 = 19 & & \text{Quotient} \\
\text{Divisor} \quad 1011_2 & \big) \; 11011001_2 = 217 & & \text{Dividend} \\
 & -1011 & & \\
\hline
 & 10 & & \\
 & 101 & & \\
 & 1010 & & \\
 & 10100 & & \\
 & -1011 & & \\
\hline
 & 1001 & & \\
 & 10011 & & \\
 & -1011 & & \\
\hline
 & 1000_2 = 8 & & \text{Remainder}
\end{array}
$$

Try to see how big a number can be subtracted, creating a digit of the quotient on each attempt

Dividend =

Quotient × Divisor

+ Remainder

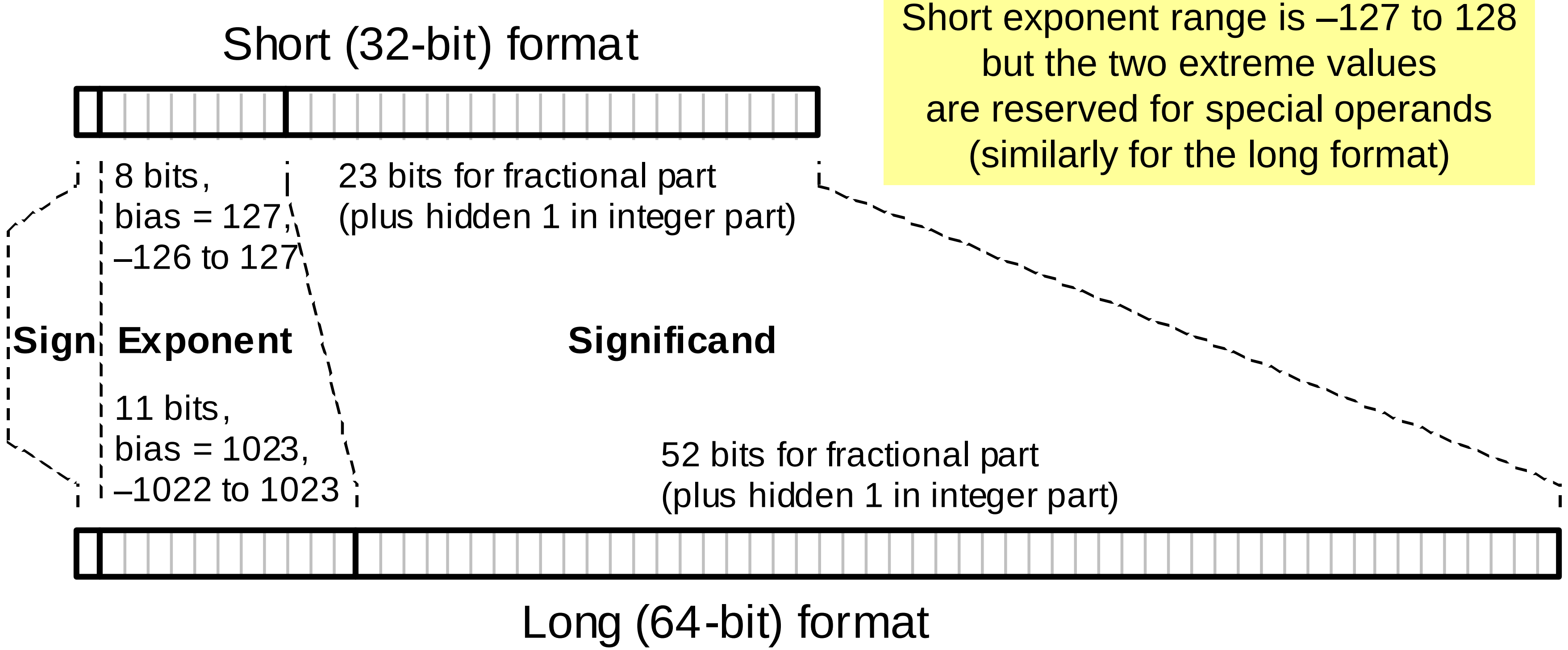
217 = 19 × 11 + 8

Binary division is accomplished via shifting and subtraction

# ANSI/IEEE Standard Floating-Point Format (IEEE 754)

Revision (IEEE 754R) is being considered by a committee

Short exponent range is –127 to 128
but the two extreme values
are reserved for special operands
(similarly for the long format)




The two ANSI/IEEE standard floating-point formats.

# Floating-Point Representation 1

- Convert the decimal number to binary:
  - $228_{10} = 11100100_2 = 1.11001 \times 2^7$
- Fill in each field of the 32-bit number:
  - The sign bit is positive (0)
  - The 8 exponent bits represent the value 7
  - The remaining 23 bits are the mantissa

| 1 bit | 8 bits | 23 bits |
|---|---|---|
| 0 | 00000111 | 111 0010 0000 0000 0000 0000 |
| **Sign** | **Exponent** | **Mantissa** |

# Floating-Point Representation 2

- First bit of the mantissa is always 1:
  - $228_{10} = 11100100_2 = \mathbf{1}.\mathbf{11001} \times 2^7$
- Thus, storing the most significant 1, also called the *implicit leading 1,* is redundant information.
- Instead, store just the fraction bits in the 23-bit field. The leading 1 is implied.

| 1 bit | 8 bits | 23 bits |
|---|---|---|
| 0 | 00000111 | 110 0100 0000 0000 0000 0000 |
| **Sign** | **Exponent** | **Fraction** |

# Floating-Point Representation 3

- *Biased exponent*: bias = 127 ($01111111_2$)
  - Biased exponent = bias + exponent
  - Exponent of 7 is stored as:

$$127 + 7 = 134 = 0x10000110_2$$

- The **IEEE 754 32-bit floating-point representation** of $228_{10}$

| 1 bit | 8 bits | 23 bits |
|---|---|---|
| 0 | 10000110 | 110 0100 0000 0000 0000 0000 |
| **Sign** | **Biased Exponent** | **Fraction** |

# Normalized and denormalized numbers

If the exponent is between 1 and 254, a normal real number is represented.

If the exponent is 0:

- if fraction is 0, then value = 0.

- if fraction is not zero, it represents a denormalized number.

$b_1\ b_2\ \ldots\ b_{23}$ represents $0.\ b_1\ b_2\ \ldots\ b_{23}$ rather than $1.b_1b_2\ \ldots\ b_{23}$

Why? To reduce the chance of underflow.

# Denormalized numbers

- No hidden 1
- Allows numbers very close to 0
- E = 00…0 ➔ Different interpretation applies
- Denormalization rule: number represented is

  $(-1)^S \times 0.F \times 2^{-126}$ (single-precision)

  $(-1)^S \times 0.F \times 2^{-1022}$ (double-precision)
- Note: zeroes also follow this rule

# Special-case numbers

- Zeroes:

| | | | |
|---|---|---|---|
| 0 | 00…0 | 00…0 | ➔ +0 |
| 1 | 00…0 | 00…0 | ➔ -0 |

- Infinities:

| | | | |
|---|---|---|---|
| 0 | 11…1 | 00…0 | ➔ +∞ |
| 1 | 11…1 | 00…0 | ➔ -∞ |

- Not a Number (NaN): E = 11…1; F != 00…0

| | | |
|---|---|---|
| 1 | 11…1 | F != 00…0 |

# Short and Long IEEE 754 Formats: Features

Some features of ANSI/IEEE standard floating-point formats

| **Feature** | **Single/Short** | **Double/Long** |
|---|---|---|
| Word width in bits | 32 | 64 |
| Significand in bits | 23 + 1 hidden | 52 + 1 hidden |
| Significand range | $[1, 2 - 2^{-23}]$ | $[1, 2 - 2^{-52}]$ |
| Exponent bits | 8 | 11 |
| Exponent bias | 127 | 1023 |
| Zero (±0) | $e$ + bias = 0, $f = 0$ | $e$ + bias = 0, $f = 0$ |
| Denormal | $e$ + bias = 0, $f \neq 0$<br>represents $\pm 0.f \times 2^{-126}$ | $e$ + bias = 0, $f \neq 0$<br>represents $\pm 0.f \times 2^{-1022}$ |
| Infinity (±∞) | $e$ + bias = 255, $f = 0$ | $e$ + bias = 2047, $f = 0$ |
| Not-a-number (NaN) | $e$ + bias = 255, $f \neq 0$ | $e$ + bias = 2047, $f \neq 0$ |
| Ordinary number | $e$ + bias $\in [1, 254]$<br>$e \in [-126, 127]$<br>represents $1.f \times 2^{e}$ | $e$ + bias $\in [1, 2046]$<br>$e \in [-1022, 1023]$<br>represents $1.f \times 2^{e}$ |
| *min* | $2^{-126} \cong 1.2 \times 10^{-38}$ | $2^{-1022} \cong 2.2 \times 10^{-308}$ |
| *max* | $\cong 2^{128} \cong 3.4 \times 10^{38}$ | $\cong 2^{1024} \cong 1.8 \times 10^{308}$ |