IEEE754 code

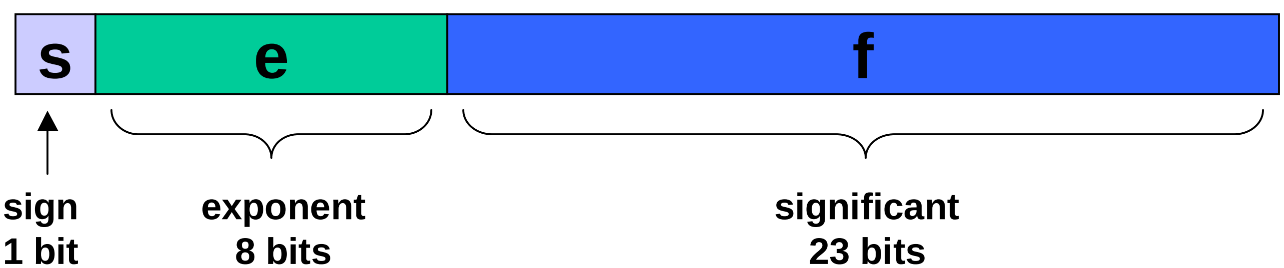
IEEE – Institute of Electrical and Electronics Engineers

IEEE754 (1985) - Binary floating point standard

FP binary number

$$(-1)^{s} * 1.f * 2^{e-127}$$

is coded as follows (32 bits):



IEEE754

$$(-1)^{s} * 1.f * 2^{e-127}$$

Significant is stored without the leading 1.

Exponent is stored in a biased form,
i.e. biased by 127 (for 32-bit FP),
in order to avoid coding negative values and
facilitate the comaparison of FP numbers.

$$1\,10000001\,010000...000_b =$$

$$-1 * 1.01_b * 2^{129-127} = -1.01_b * 2^2 = -5$$

$$0\,01100001\,000000...000_b =$$

$$+1 * 1.0_b * 2^{96-127} = 1 * 2^{-31} \approx 4.65e-10$$

Dec → IEEE754

-7.25

↓

-111.01_b

↓

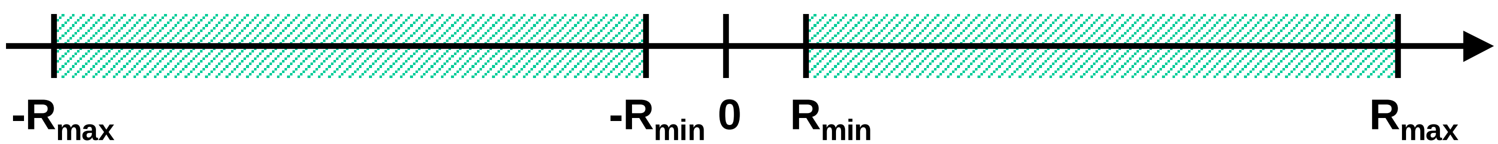
$-1.1101_b * 2^2$

↓

1 10000001 11010000000000000000000

Limitations of IEEE754

Range limitation:
due to limited exponent bit-field



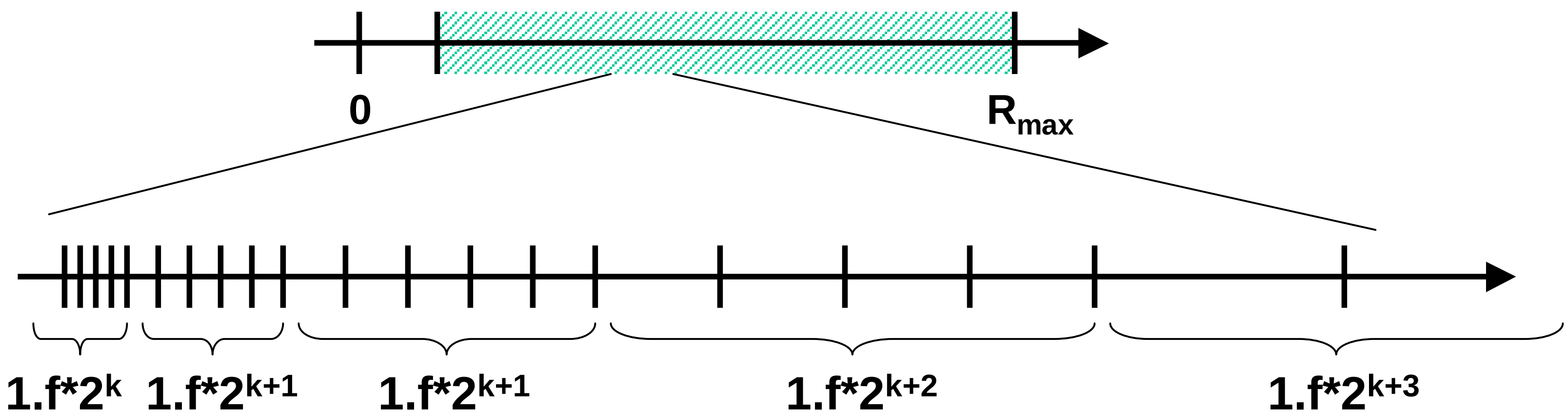
IEEE754 single precision (32 bits)

$R_{min} \approx 1.2e\text{-}38$

$R_{max} \approx 3.4e\text{+}38$

Limitations of IEEE754

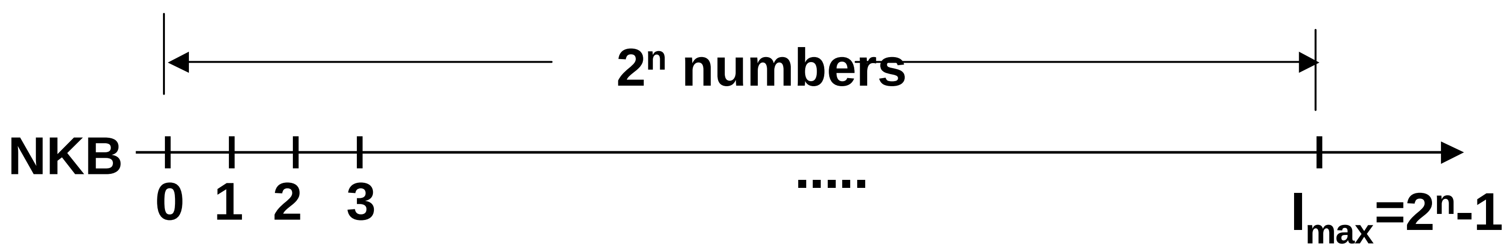
Accuracy limitation: due to limited significant bit-field



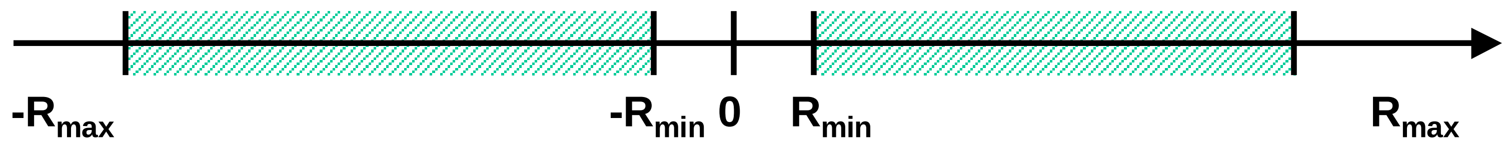
Only selected FP numbers can be represented.
Numbers "density" is not constant and depends on the exponent value.
In each $<2^i , 2^{i+1}>$ interval, there is 2^{n+1} equally distributed numbers, where n is number of bits of the significant.
For numbers close to R_{min}, accuracy is the best, but range is the shortest, for numbers close to R_{max}, accuracy is the worst, but range the widest.

FP numbers vs integer

2^n integer numbers can be represented with n-bits.



Less than 2^n FP numbers can be represented with n-bits.



n-bits gives max. 2^n different bit patterns.
The meaning of those patterns is just the matter of interpretation.
In case of FP, all available bit patterns are just differently distributed over the x-axis, but the total number of all possible number representation is constant.

Single vs Double Precision IEEE754

Single Precision: 32 bits

8b exponent + 23b significant

$R_{min} \approx 10^{-38}$

$R_{max} \approx 10^{+38}$

7 digit accuracy

Double Precision: 64 bits

11b exponent + 52b significant

$R_{min} \approx 10^{-308}$

$R_{max} \approx 10^{+308}$

16 digit accuracy

FP Arithmetics

Rules of FP notation (IEEE754):

1. All numbers cannot be represented
2. Arithmetic operarators (+,-,*,/) return rounded results
3. Range error (underflow & overflow) can rise an exception - special IEEE754 code
4. Rounding error is never signalled
5. Basic arithmetic operations require complex hardware & algorithms

Reserved IEEE754 codes

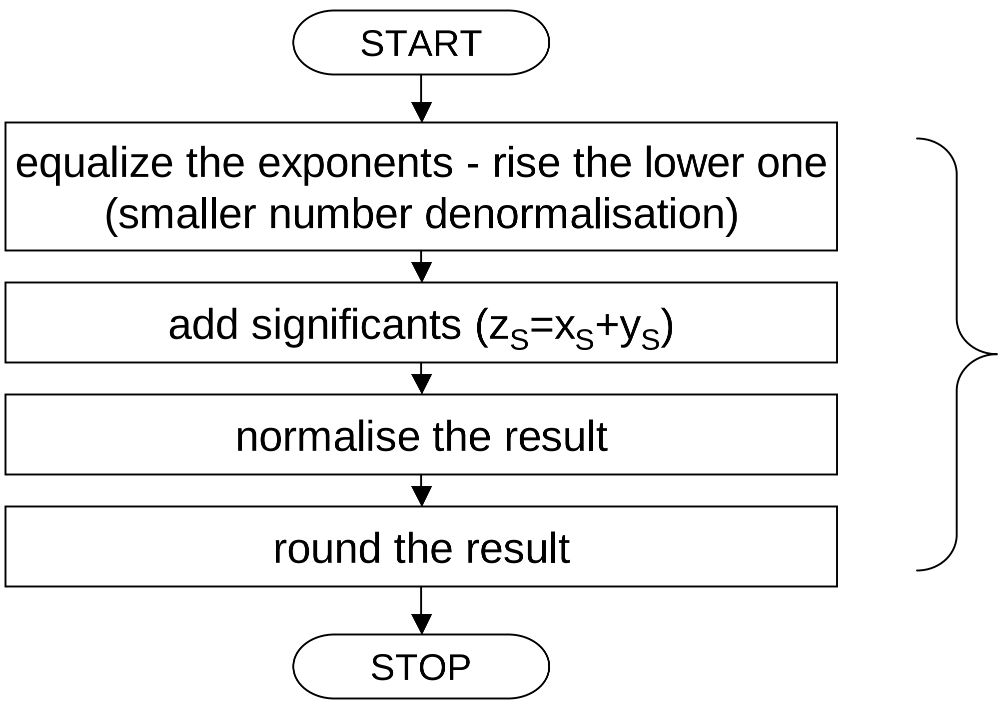
	sign	exponent	significant
positive number	0	1-254	significant
negative number	1	1-254	significant
number zero+ (0+)	0	0	0
number zero- (0-)	1	0	0
denormalised number	0/1	0	significant
+ infinity	0	255	0
- infinity	1	255	0
NaN (Not a Number)	0/1	255	≠0 (error code)

FP addition

$$x = x_S \cdot 2^{x_E}$$

$$y = y_S \cdot 2^{y_E}$$

$$\Rightarrow \quad z = x + y \quad \Rightarrow \quad z = z_S \cdot 2^{z_E}$$



Exponent
Overflow or Underflow ?

Significant
Overflow or Underflow ?

Equalization of exponents - denormalisation

$$1.\boxed{000000}\,2^{+4} + 1.\boxed{000000}\,2^{-4}$$

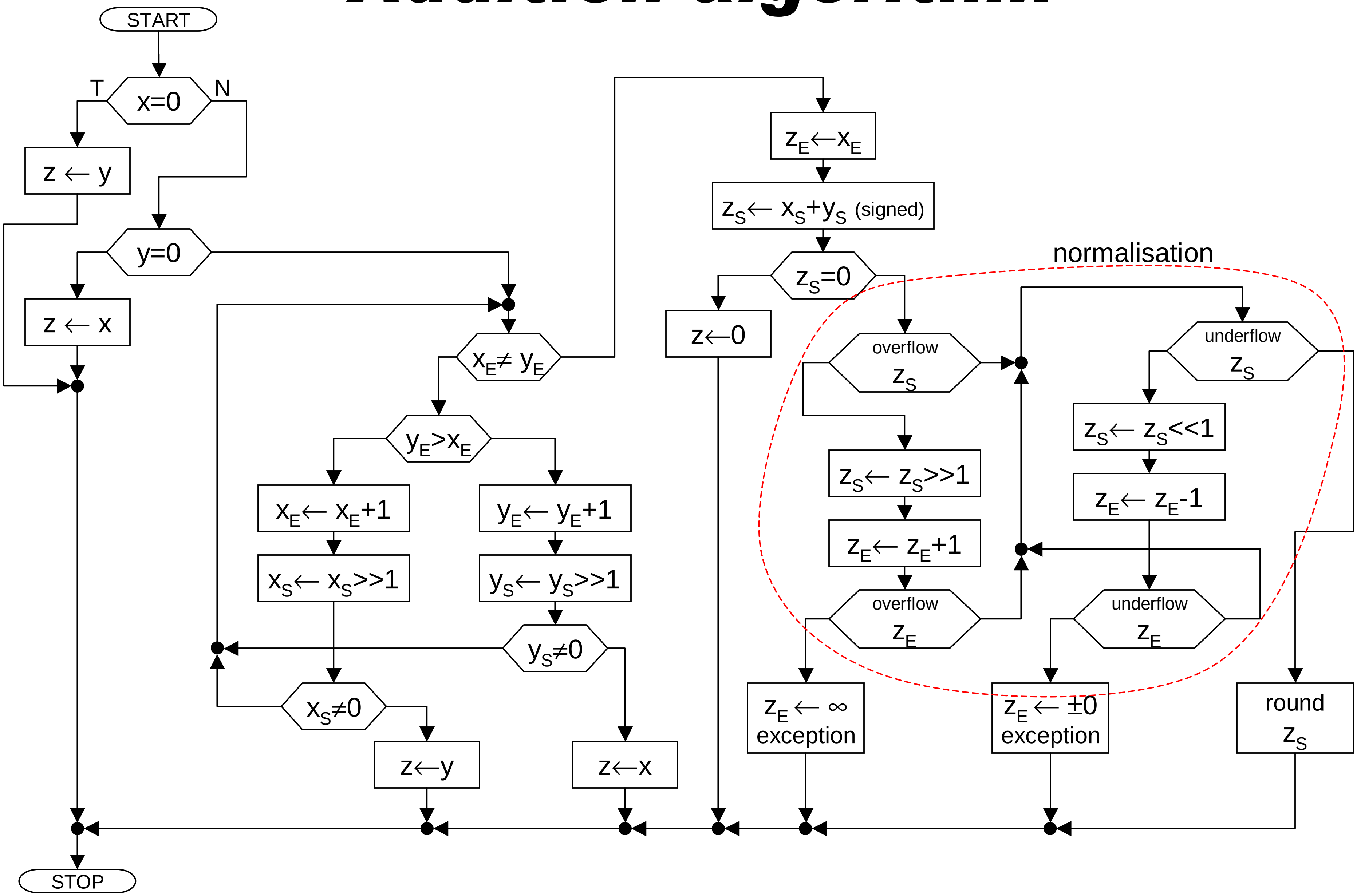
$$1.\boxed{000000}\,2^{+4} + 0.\boxed{000000}01\;2^{+4}$$

Denormalisation: move significant right & increment exponent

!

Addition of numbers with a big difference in order of magnitude has no influence on the results - the smaller number is neglected.

Addition algorithm



Rounding the significant

rounding „up”

$$z_S = \ldots\ 00\ |\ 1..01$$

$$z_S = \ldots\ 01\ |$$

rounding „down”

$$z_S = \ldots\ 00\ |\ 0..01$$

$$z_S = \ldots\ 00\ |$$

rounding to the nearest even number

$$z_S = \ldots\ 00\ |\ 100..$$

$$z_S = \ldots\ 00\ |$$

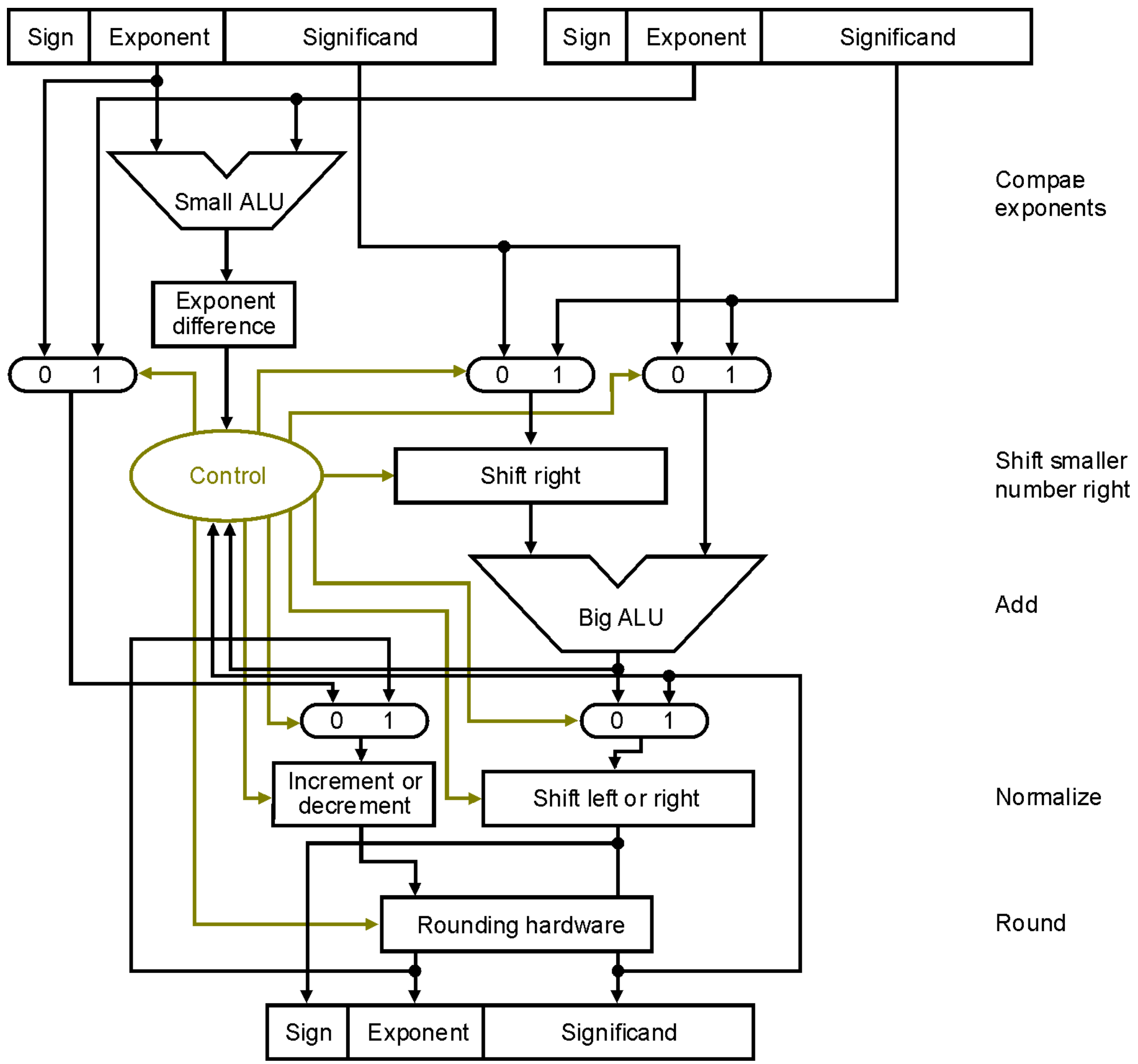
$$z_S = \ldots\ 01\ |\ 100..$$

$$z_S = \ldots\ 10\ |$$

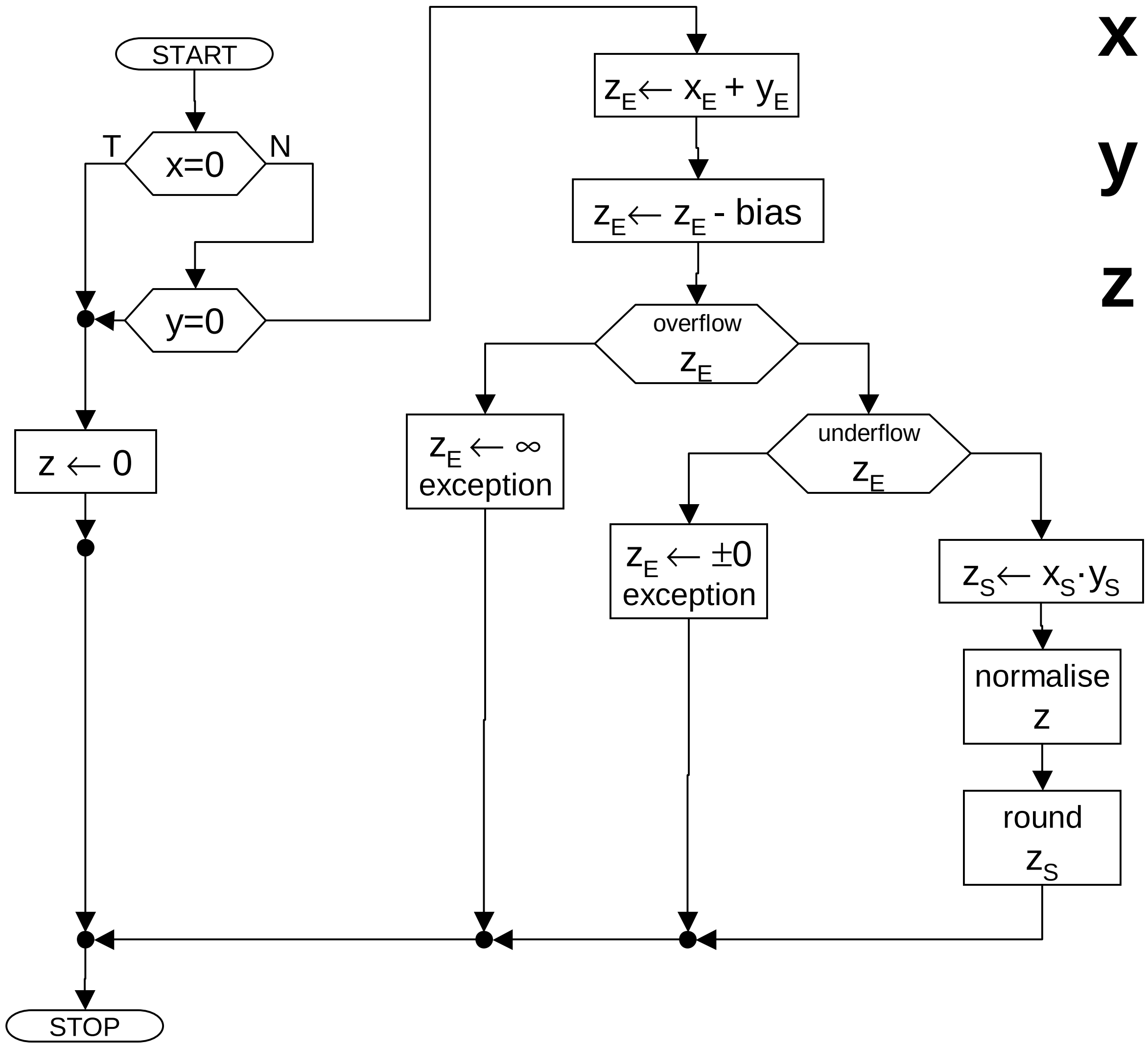
FP working registers must be longer then the nominal size
to provide higher accuracy as long as possible, before rounding.

Rounding rules must assure deterministic results
on various computer architectures.

Addition Hardware



Multiplication algorithm

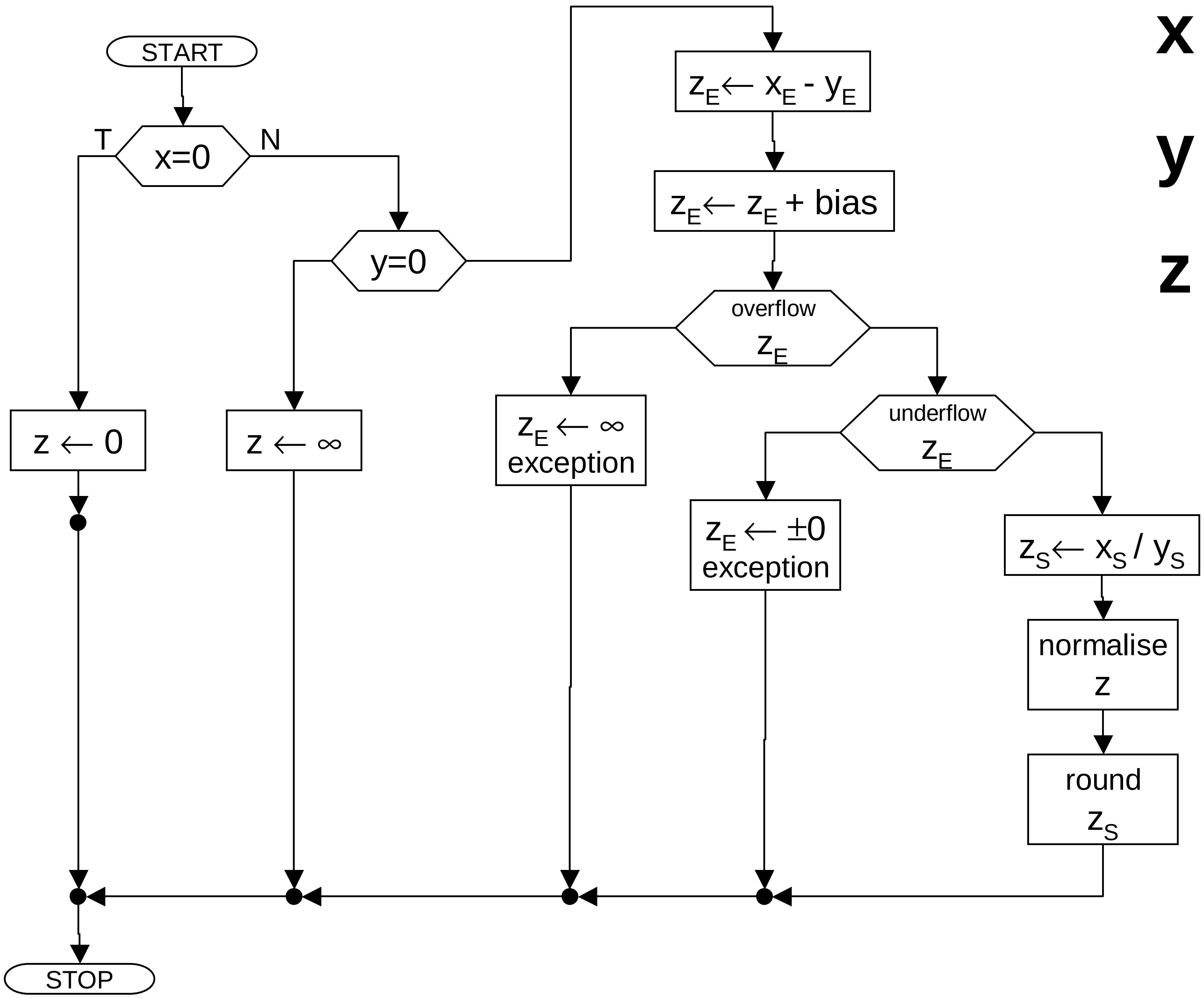


$$x = x_S \cdot 2^{x_E}$$

$$y = y_S \cdot 2^{y_E}$$

$$z = x_S \cdot y_S \cdot 2^{x_E + y_E}$$

Division algorithm



$$x = x_S \cdot 2^{x_E}$$

$$y = y_S \cdot 2^{y_E}$$

$$z = x_S / y_S \cdot 2^{x_E - y_E}$$