Outline of Lecture

- **Floating Point Addition**
- **Floating Point Multiplication**

IEEE 754 floating-point standard

- In order to pack more bits into the significant, IEEE 754 makes the leading 1 bit of normalized binary numbers ***implicit***.

- In this case the significant will be 24 bits long in single precision (implied 1 and 23-bit fraction), and 53 bits long in double precision (1 + 52).

- In this case, numbers are represented as follows:

$$(-1)^S \times (1 + \text{significant}) \times 2^E$$

- The bits of the significant represent the fraction between 0 and 1 and E specifies the value in the exponent field.

- If the bits in the significant from left to right are s1, s2, ..., then the value is:

$$(-1)^S \times (1 + (s1 \times 2^{-1}) + (s2 \times 2^{-2}) + (s2 \times 2^{-3}) + \ldots) \times 2^E$$

Example

Show the IEEE 754 representation of the number -0.75 in single precision and double precision.

Answer

$- 0.75_{ten} = - 0.11_{two}$

In scientific notation the value is $-0.11_{two} \times 2^{0}$ and in normalized scientific notation it is $-1.1_{two} \times 2^{-1}$.

The general representation for single precision is:

$$(-1)^{S} \times (1 + \text{significant}) \times 2^{(\text{exponent} - 127)}$$

Thus $-1.1_{two} \times 2^{-1}$ is represented as follows:

$$(-1)^{S} \times (1 + .1000\ 0000\ 0000\ 0000\ 0000\ 000_{two}) \times 2^{(126 - 127)}$$

1   01111110   1000 0000 0000 0000 0000 000 = 32 bits

The double precision representation is:

$$(-1)^{S} \times (1 + .1000\ 0000\ 0000\ ....\ 0000\ 000_{two}) \times 2^{(1022 - 1023)}$$

1   01111111110   0000000000000 ... 000 = 64 bits

Example

What decimal number is represented by this word?

1 10000001 010000000000 ... 0000 = 32 bits

Answer

The sign bit = 1, the exponent field contains 129, and the significant field contains 1 x 2^{-2} = 0.25.

Using the equation:

$$(-1)^S \times (1 + \text{significant}) \times 2^{(\text{exponent} - 127)}$$

$$= (-1)^1 \times (1 + 0.25) \times 2^{(129 - 127)}$$

$$= (-1)^1 \times 1.25 \times 2^2$$

$$= -1.25 \times 4$$

$$= -5.0$$

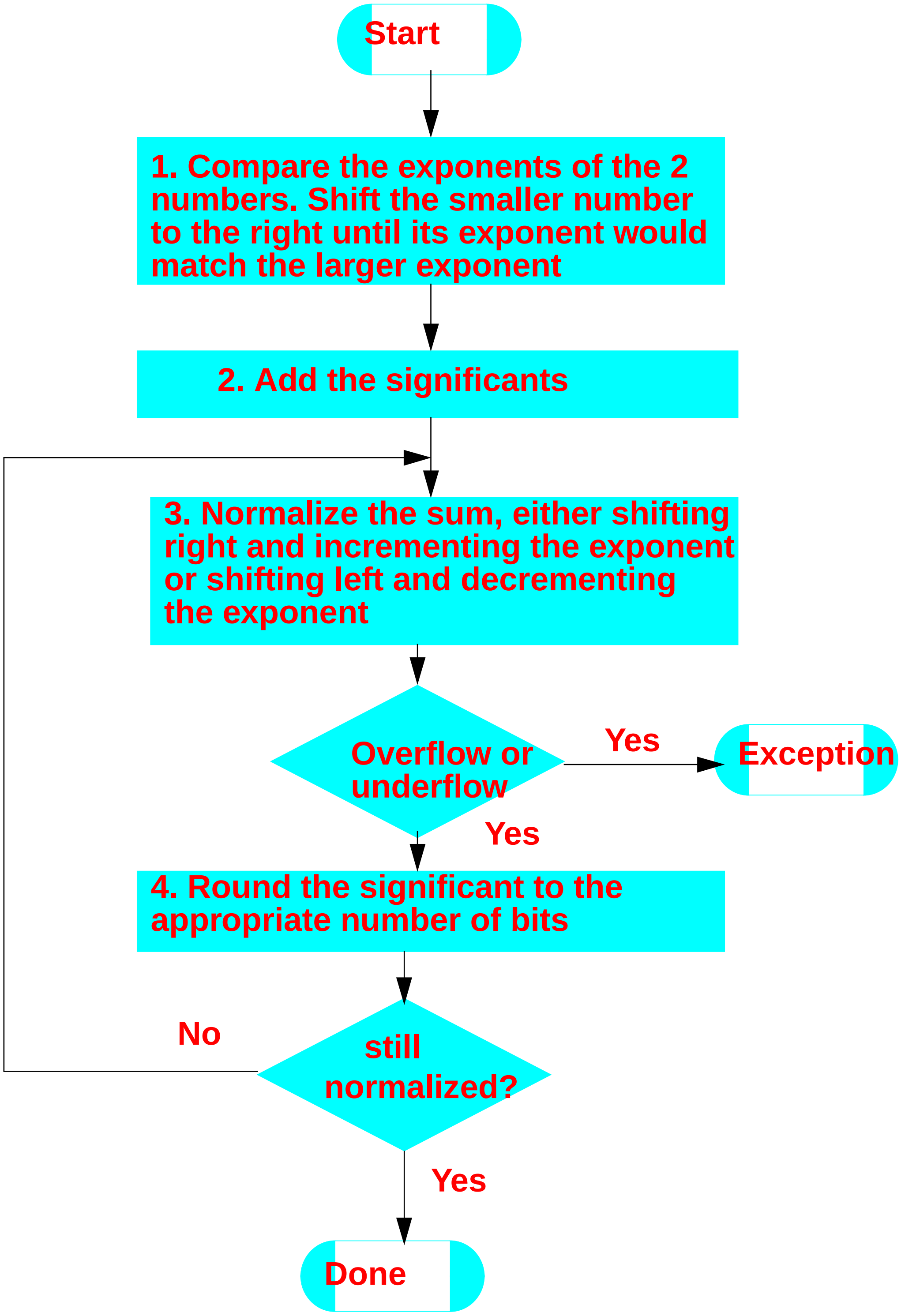
Basic Floating point Addition

- **Add 2.01 * 10^{20} to 3.11 * 10^{23}**
  - **- Adjust exponent so that 2.01 * 10^{20} becomes 0.00201 * 10^{23}**
  - **- Then add 0.00201 to 3.11 to form 3.11201**
  - **- Result is 3.11201 * 10^{23}**
  - **- Normalization may be needed if number is in IEEE standard format. (Recall hidden 1.)**
  - **- Also need special handling if result = ZERO or is too small/ too large to represent. (These are some floating point representation complexities to be discussed later)**

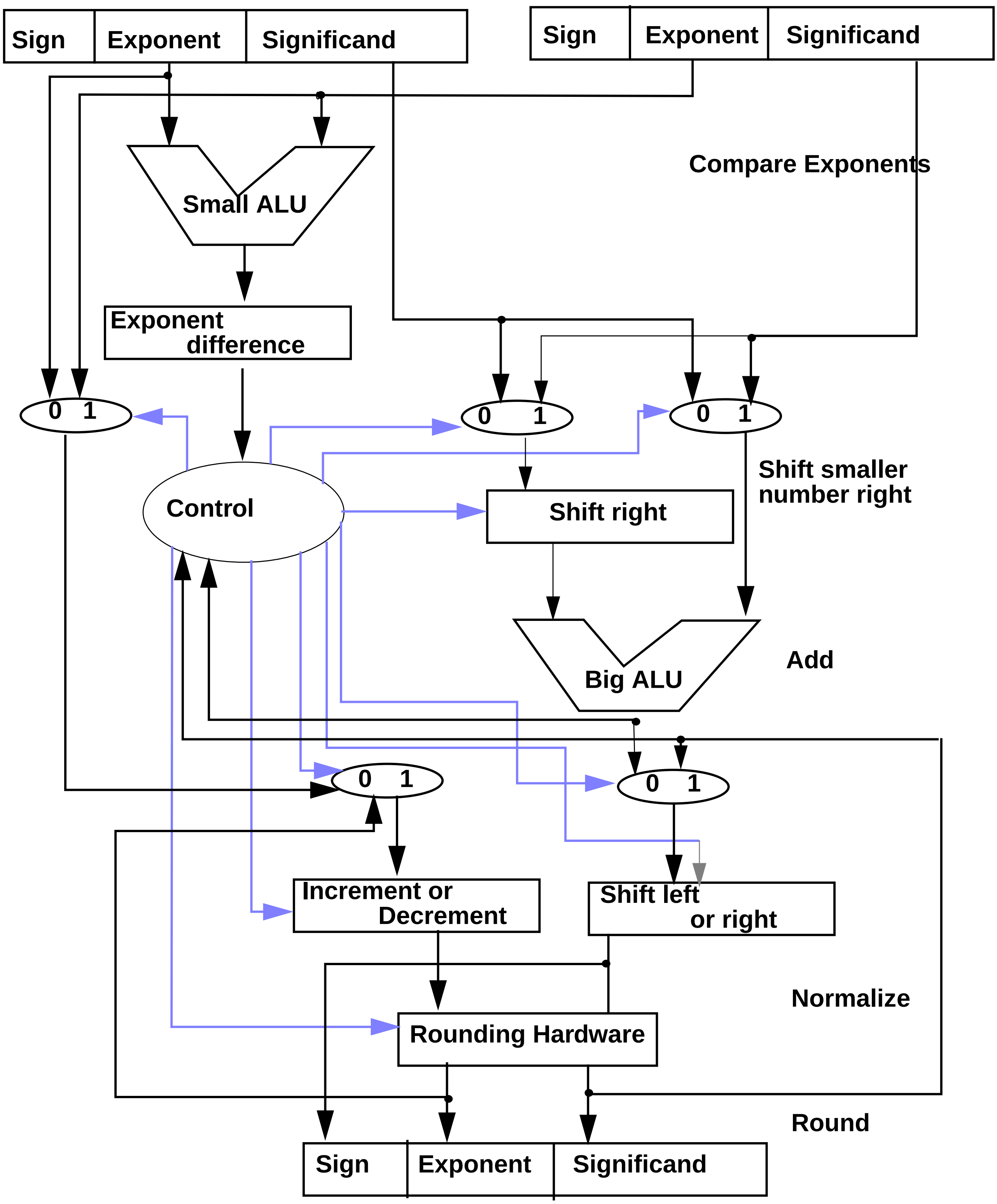
Floating Point Addition

- When we add numbers, for example 9.999×10^1 + 1.610×10^{-1}, in scientific notation, we typically follow the steps below:
  - ➔ We must ***align*** the decimal point of the number with the smaller exponent - we make 1.610×10^{-1} into 0.016×10^1
  - ➔ Then, we add the significants of the two numbers together (e.g., $9.999 \times 10^1 + 0.016 \times 10^1 = 10.015 \times 10^1$).
  - ➔ We normalize the result of the addition - 10.015×10^1 becomes 1.0015×10^2.
  - ➔ The significant can only be represented using a fixed number of digits - thus, we must round the result so that it can fit into those digits (e.g., 1.002×10^2) if we have only 4 digits to represent the significant

Floating Point Addition



Floating Point Hardware



Example

Add 0.5 to -0.4375 using the IEEE 754 floating point.

Answer

Change the two numbers in normalized scientific notation.

$0.5_{ten} = 1.000_{two} \times 2^{-1}$

$-0.4375_{ten} = -1.110_{two} \times 2^{-2}$

Step1: The significant of the smaller number is shifted right until its exponent matches the larger number:

$$-1.110_{two} \times 2^{-2} = -0.111_{two} \times 2^{-1}$$

Step 2: Add the significants

$$1.000_{two} \times 2^{-1} + (-0.111_{two} \times 2^{-1}) = 0.001 \times 2^{-1}$$

Step 3: Normalize the sum, and check the overflow and underflow

$$0.001_{two} \times 2^{-1} = 1.000_{two} \times 2^{-4}$$

$127 \geq -4 \geq -126$, thus there is no overflow or underflow.

Step 4: Round the sum (assume we have 4 bits of precision)

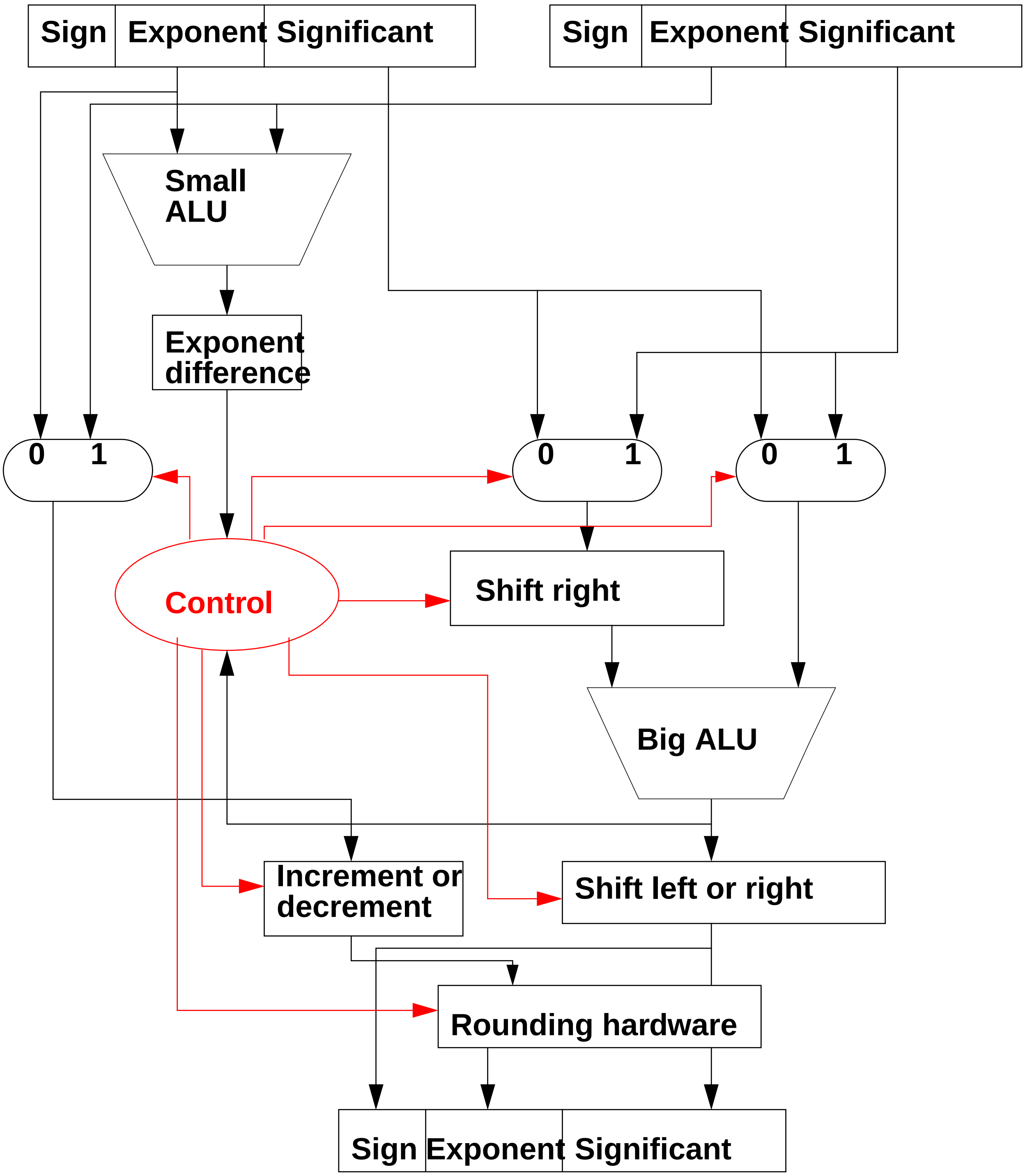
$$1.000_{two} \times 2^{-4}$$

The sum fits in 4 bits, so there is no need for rounding.

$$1.000_{two} \times 2^{-4} = 0.0001_{two} = 0.0625_{ten}$$

Using the IEEE 754 format, $1.000_{two} \times 2^{-4}$ would be represented as:

0 01111011 000000 0000


Sign
Exponent
Significant
Sign
Exponent
Significant
Small ALU
Exponent difference
0 1
0 1
0 1
Control
Shift right
Big ALU
Increment or decrement
Shift left or right
Rounding hardware
Sign
Exponent
Significant

Floating-Point Multiplication

Given two decimal numbers in scientific notation, we try to multiply them (e.g., $1.110_{ten} \times 10^{10} \times 9.200_{ten} \times 10^{-5}$):

Step 1: We find the exponent of the product by adding the exponents of the products together

New exponent = 10 + (-5) = 5

Step 2: We perform the multiplication of the significants

New significant: $1.110 \times 9.200 = 10.21200$

The product is: 10.212×10^5

Step 3: We normalize the product.

$10.212 \times 10^5 = 1.0212 \times 10^6$

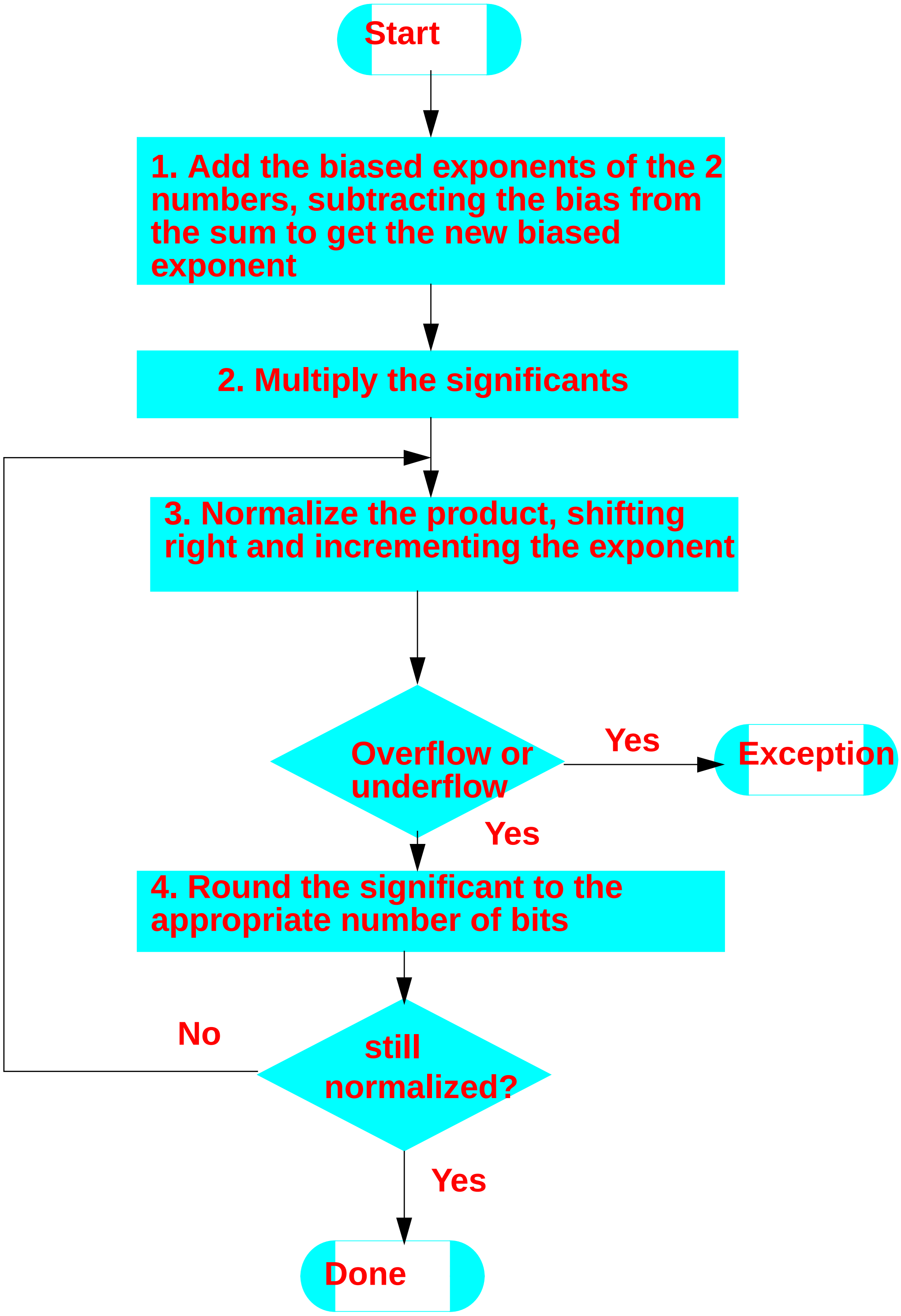
Step 4: We round the product (assume the significant is only 4 digits)

New products is: 1.021×10^6

Step 5: We find the sign of the product - it is positive unless the signs of the two numbers are different.

$+\ 1.021 \times 10^6$

Floating-Point Multiplication



Example

Multiply 0.5 and -0.4375 using floating point representation

Answer

$0.5 = 1.000_{two} \times 2^{-1}$

$-0.4375 = -1.110_{two} \times 2^{-2}$

Step 1: Add the exponents

New exponent= -1 + (-2) = -3

Step 2: Multiply the significants

New significant = $1.000_{two} \times 1.110_{two} = 1.110000_{two}$

New product = $1.110_{two} \times 2^{-3}$ (significant represented by 4 bits)

Step 3: Normalize the product and Check for overflow or under-flow

The product is normalized

$127 \geq -3 \geq -126$, thus there is no overflow or underflow

Step 4: Round the product

Product = $1.110_{two} \times 2^{-3}$

Step 5: The sign of the product is (-)

Product = $-1.110_{two} \times 2^{-3} = -0.21875_{ten}$

Using the IEEE floating point representation, the result is:

$1\ 01111100\ 11000 \ldots\ldots\ldots.\ 0000_{two}$

Floating Point Complexities

- **Operations are somewhat more complicated (See test)**
- **In addition to** overflow **we can have** underflow
- **Accuracy can be a big problem**
  - **rounding errors**
  - **positive divided by zero yields "Not a Number" (NaN)**
- **Implementing the standard can be tricky**
- **Not using the standard can be worse**
  - **see text for description of 80 x 86 and pentium bug!**
- **The MIPS processor supports the IEEE single and double precision formats:**
  - **Addition**

    add.s and add.d
  - **Subtraction**

    sub.s and sub.d