Outline of Lecture

- **Floating point numbers**

Floating Point

- **We need a way to represent** real numbers
- **For example, we should be able to include:**

$3.14159256 = \pi$ (fraction)

$2.71828 = e$ (fraction)

1.0×10^{-9} (very small)

3.15576×10^{25} (very large)

- **Computer arithmetic that supports such numbers is called *floating point*, because it represents numbers in which the decimal point is not fixed, as it is for integers.**

Representation:

- **In general, floating point numbers are of the form:**

$$(-1)^S \times F \times 2^E$$

- **Increasing the number of bits in E means increasing the range of numbers that can be represented.**

- **Increasing the number of bits in F means increasing the accuracy of numbers.**

Thus, we should make a compromise between these two values

- **IEEE 754 floating point standard:**

  - single precision: 8 bit exponent, 23 bit significand

  - double precision: 11 bit exponent, 52 bit significand

Normalized Form

- A number in scientific notation which has no leading 0's is called a ***normalized*** number.
- In binary, the form is given as follows:

$$1.xxxxxx_2 \times 2^{yyyy}$$

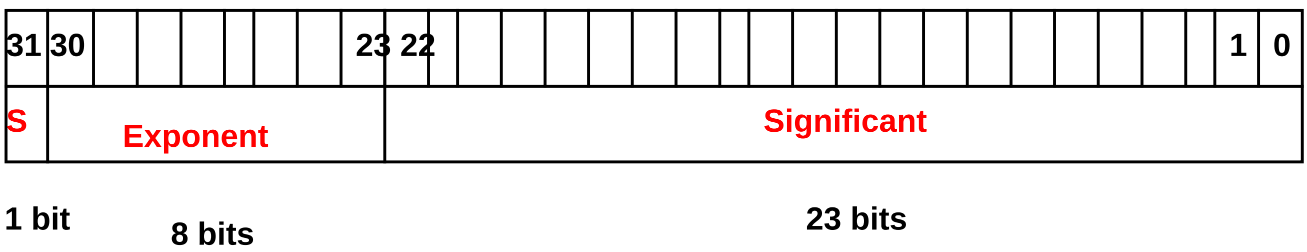
- Practicality dictates that floating-point numbers be compatible with the size of a word.

Representation in MIPS

- The MIPS, and almost any computer, floating point format, follows the *IEEE 754 floating point standard*. We need a standard so that programs can be easily ported across different computers.

- E has 8 bits and F has 23 bits. As a result, numbers as small as 2.0×10^{-38} and numbers as large as 2.0×10^{38} can be represented in a MIPS computer.

The MIPS floating point representation is shown below:



- Any numbers ou*tside that range will create an *overflow* or *underflow*.

Representation in MIPS

- **Most computers, including MIPS, also have a *dou-ble precision* floating-point arithmetic so that the range and the accuracy of numbers is increased.**

- **In this case, 64 bits (two MIPS words) are used to represent floating point numbers -**

from 2.0×10^{-308} to 2.0×10^{-308} .

S = 1 bit

E = 11 bits

F = 52 bits

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:

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

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:

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

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

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

$$= - 1.25 \times 4$$

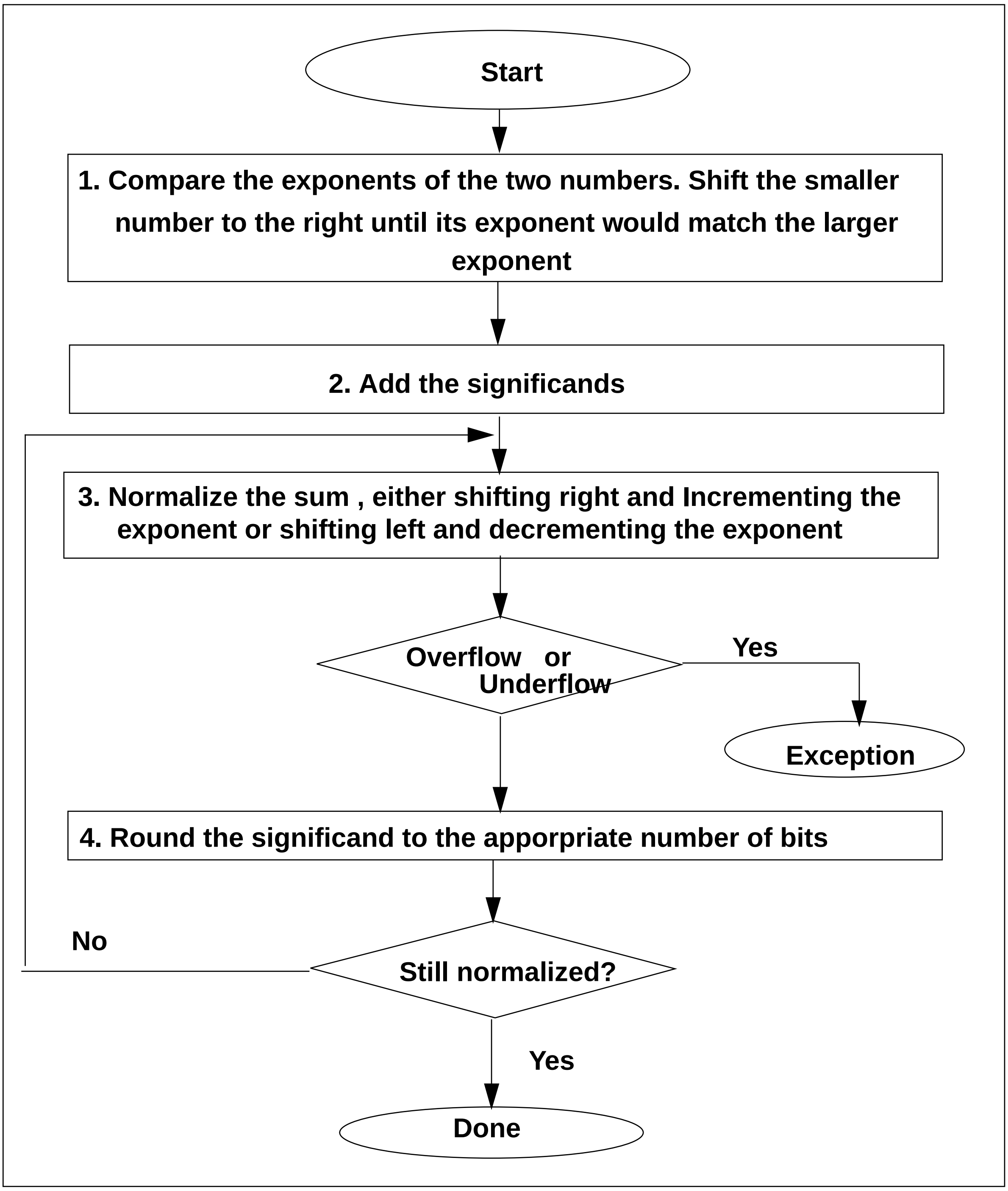
$$= - 5.0$$

Basic Floating point Addition

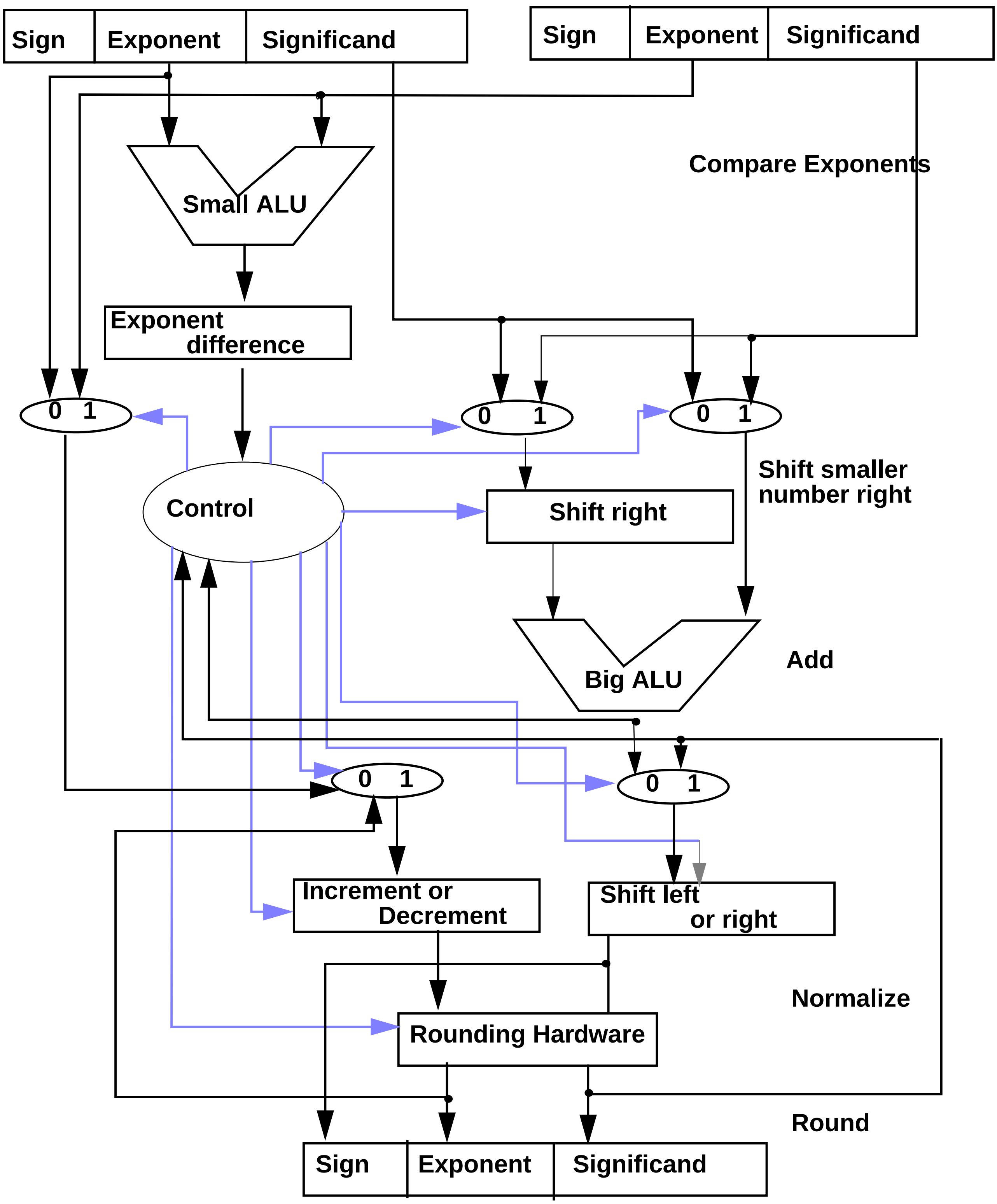
- **Add 2.01 * 10^{20} to 3.11 * 10^{23}**
  - **Adjust exponent so that 2.01 * 10^{20} becomes 0.00201 * 10^{23}**
  - **-Then add 0.00201 to 3.11 to form 3.11201**
  - **Result is 3.11201 * 10^{23}**
  - **Normalization may be needed if number is in IEEE standard format. (Recall hidden 1.)**
  - **Also 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

- Exactly what you do in primary school.



Floating Point Addition



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

Chapter Four Summary

- **Computer arthimetic is constrained by limited precision**

- **Bit patterns have no inherent meaning but standards do exist**

  - two's compliment
  - IEEE 754 floating point

- **Computer instructions determine meaning of the bit patterns**

- **Performance and accuracy are important so there are many complexities in real machine (i .e ., algorithms and implementations).**