(1.2) Round-off Errors and Computer Arithmetic

MATH 4701 Numerical Analysis

Decimal Representation of Real Numbers

We normally refer to numbers by their **decimal representation**s.

Decimal Representation of Real Numbers

We normally refer to numbers by their **decimal representation**s.

A real number is represented by a code of the form $a_N a_{N-1} \ldots a_1 a_0 . a_{-1} a_{-2} \ldots$, where each a_i is a digit chosen out of the ten possible digits 0, 1, 2, ... , 8, and 9.

Decimal Representation of Real Numbers

We normally refer to numbers by their **decimal representations**.

A real number is represented by a code of the form $a_N a_{N-1} \ldots a_1 a_0 . a_{-1} a_{-2} \ldots$, where each a_i is a digit chosen out of the ten possible digits 0, 1, 2, ... , 8, and 9.

The code $a_N a_{N-1} \ldots a_1 a_0 . a_{-1} a_{-2} \ldots$ represents the value of the series

$$a_N 10^N + a_{N-1} 10^{N-1} + \ldots + a_1 10 + a_0 + a_{-1} 10^{-1} + a_{-2} 10^{-2} + a_{-3} 10^{-3} + \ldots$$

Decimal Representation of Real Numbers

We normally refer to numbers by their **decimal representation**s.

A real number is represented by a code of the form $a_N a_{N-1} \ldots a_1 a_0 . a_{-1} a_{-2} \ldots$, where each a_i is a digit chosen out of the ten possible digits 0, 1, 2, ... , 8, and 9.

The code $a_N a_{N-1} \ldots a_1 a_0 . a_{-1} a_{-2} \ldots$ represents the value of the series

$$a_N 10^N + a_{N-1} 10^{N-1} + \ldots + a_1 10 + a_0 + a_{-1} 10^{-1} + a_{-2} 10^{-2} + a_{-3} 10^{-3} + \ldots$$

For example

$$621.38888\ldots = 621.3\bar{8} = 6 \times 100 + 2 \times 10 + 1 + \frac{3}{10} + \frac{8}{100} + \frac{8}{1000} + \frac{8}{10000} + \ldots.$$

Decimal Representation of Real Numbers

We normally refer to numbers by their **decimal representation**s.

A real number is represented by a code of the form $a_N a_{N-1} \ldots a_1 a_0 . a_{-1} a_{-2} \ldots$, where each a_i is a digit chosen out of the ten possible digits 0, 1, 2, ... , 8, and 9.

The code $a_N a_{N-1} \ldots a_1 a_0 . a_{-1} a_{-2} \ldots$ represents the value of the series

$$a_N 10^N + a_{N-1} 10^{N-1} + \ldots + a_1 10 + a_0 + a_{-1} 10^{-1} + a_{-2} 10^{-2} + a_{-3} 10^{-3} + \ldots$$

When $a_i = 0$ for all $i \leq -k - 1$, we omit all the zero digits after the kth decimal place and we write the code as finite sum

$$a_N a_{N-1} \ldots a_1 a_0 . a_{-1} a_{-2} \ldots a_{-k}.$$

Decimal Representation of Real Numbers

The code $a_N a_{N-1} \ldots a_1 a_0 . a_{-1} a_{-2} \ldots$ represents the value of the series

$$a_N 10^N + a_{N-1} 10^{N-1} + \ldots + a_1 10 + a_0 + a_{-1} 10^{-1} + a_{-2} 10^{-2} + a_{-3} 10^{-3} + \ldots$$

When $a_i = 0$ for all $i \leq -k - 1$, we omit all the zero digits after the kth decimal place and we write the code as finite sum

$$a_N a_{N-1} \ldots a_1 a_0 . a_{-1} a_{-2} \ldots a_{-k} .$$

The only way two different decimal representations can describe the same number is

$$a_N a_{N-1} \ldots a_1 a_0 . a_{-1} a_{-2} \ldots a_{-k} \bar{0} = a_N a_{N-1} \ldots a_1 a_0 . a_{-1} a_{-2} \ldots \hat{a_{-k}} \bar{9}$$

where $a_{-k} \neq 0$ and $\hat{a_{-k}} = a_{-k} - 1$.

Decimal Representation of Real Numbers

When $a_i = 0$ for all $i \leq -k-1$, we omit all the zero digits after the kth decimal place and we write the code as finite sum

$$a_N a_{N-1} \ldots a_1 a_0 . a_{-1} a_{-2} \ldots a_{-k}.$$

The only way two different decimal representations can describe the same number is

$$a_N a_{N-1} \ldots a_1 a_0 . a_{-1} a_{-2} \ldots a_{-k} \bar{0} = a_N a_{N-1} \ldots a_1 a_0 . a_{-1} a_{-2} \ldots \hat{a_{-k}} \bar{9}$$

where $a_{-k} \neq 0$ and $\hat{a_{-k}} = a_{-k} - 1$.

For example $2437.923 = 2437.923\bar{0} = 2437.922\bar{9}$.

Decimal Representation of Real Numbers

The only way two different decimal representations can describe the same number is

$$a_N a_{N-1} \ldots a_1 a_0 . a_{-1} a_{-2} \ldots a_{-k} \bar{0} = a_N a_{N-1} \ldots a_1 a_0 . a_{-1} a_{-2} \ldots \hat{a_{-k}} \bar{9}$$

where $a_{-k} \neq 0$ and $\hat{a_{-k}} = a_{-k} - 1$.

For example $2437.923 = 2437.923\bar{0} = 2437.922\bar{9}$.

For any positive integer n when a number is multiplied by 10^n, the decimal point in its decimal code is shifted n places to the right,

Decimal Representation of Real Numbers

The only way two different decimal representations can describe the same number is

$$a_N a_{N-1} \ldots a_1 a_0 . a_{-1} a_{-2} \ldots a_{-k} \bar{0} = a_N a_{N-1} \ldots a_1 a_0 . a_{-1} a_{-2} \ldots \hat{a_{-k}} \bar{9}$$

where $a_{-k} \neq 0$ and $\hat{a_{-k}} = a_{-k} - 1$.

For example $2437.923 = 2437.923\bar{0} = 2437.922\bar{9}$.

For any positive integer n when a number is multiplied by 10^n, the decimal point in its decimal code is shifted n places to the right,

and when a number is multiplied by 10^{-n}, the decimal point in its decimal code is shifted n places to the left.

Binary Representation of Real Numbers

The decimal representation of numbers is not suitable for computing devices since it requires a machine to generate and understand ten different states for the ten needed digits.

Binary Representation of Real Numbers

The decimal representation of numbers is not suitable for computing devices since it requires a machine to generate and understand ten different states for the ten needed digits.

It is also very difficult to design logic gates that describe the basic arithmetic (addition and multiplication) between pairs of these ten digits.

Binary Representation of Real Numbers

The decimal representation of numbers is not suitable for computing devices since it requires a machine to generate and understand ten different states for the ten needed digits.

It is also very difficult to design logic gates that describe the basic arithmetic (addition and multiplication) between pairs of these ten digits.

As a result, most computing devices use a representation for numbers which is based only on two codes 0 and 1.

Binary Representation of Real Numbers

The decimal representation of numbers is not suitable for computing devices since it requires a machine to generate and understand ten different states for the ten needed digits.

It is also very difficult to design logic gates that describe the basic arithmetic (addition and multiplication) between pairs of these ten digits.

As a result, most computing devices use a representation for numbers which is based only on two codes 0 and 1.

A real number A is represented by a **binary** code of the form $a_N a_{N-1} \ldots a_1 a_0 . a_{-1} a_{-2} \ldots$, where each a_i is 0 or 1.

Binary Representation of Real Numbers

The decimal representation of numbers is not suitable for computing devices since it requires a machine to generate and understand ten different states for the ten needed digits.

It is also very difficult to design logic gates that describe the basic arithmetic (addition and multiplication) between pairs of these ten digits.

As a result, most computing devices use a representation for numbers which is based only on two codes 0 and 1.

A real number A is represented by a **binary** code of the form $a_N a_{N-1} \ldots a_1 a_0 . a_{-1} a_{-2} \ldots$, where each a_i is 0 or 1.

The code $a_N a_{N-1} \ldots a_1 a_0 . a_{-1} a_{-2} \ldots$ represents the value of the series

$$a_N 2^N + a_{N-1} 2^{N-1} + \ldots + a_1 2 + a_0 + a_{-1} 2^{-1} + a_{-2} 2^{-2} + a_{-3} 2^{-3} + \ldots$$

Binary Representation of Real Numbers

A real number A is represented by a **binary** code of the form $a_N a_{N-1} \ldots a_1 a_0 . a_{-1} a_{-2} \ldots$, where each a_i is 0 or 1.

The code $a_N a_{N-1} \ldots a_1 a_0 . a_{-1} a_{-2} \ldots$ represents the value of the series

$$a_N 2^N + a_{N-1} 2^{N-1} + \ldots + a_1 2 + a_0 + a_{-1} 2^{-1} + a_{-2} 2^{-2} + a_{-3} 2^{-3} + \ldots$$

For example

$110101.110111111\ldots = 110101.110\bar{1} =$

Binary Representation of Real Numbers

A real number A is represented by a **binary** code of the form $a_N a_{N-1} \ldots a_1 a_0 . a_{-1} a_{-2} \ldots$, where each a_i is 0 or 1.

The code $a_N a_{N-1} \ldots a_1 a_0 . a_{-1} a_{-2} \ldots$ represents the value of the series

$$a_N 2^N + a_{N-1} 2^{N-1} + \ldots + a_1 2 + a_0 + a_{-1} 2^{-1} + a_{-2} 2^{-2} + a_{-3} 2^{-3} + \ldots$$

For example

$$110101.110111111\ldots = 110101.110\bar{1} =$$

$$= 1\times 2^5 + 1\times 2^4 + 0\times 2^3 + 1\times 2^2 + 0\times 2 + 1 + \frac{1}{2} + \frac{1}{2^2} + \frac{0}{2^3} + \frac{1}{2^4} + \frac{1}{2^5} + \frac{1}{2^6} + \ldots =$$

Binary Representation of Real Numbers

A real number A is represented by a **binary** code of the form $a_N a_{N-1} \ldots a_1 a_0 . a_{-1} a_{-2} \ldots$, where each a_i is 0 or 1.

The code $a_N a_{N-1} \ldots a_1 a_0 . a_{-1} a_{-2} \ldots$ represents the value of the series

$$a_N 2^N + a_{N-1} 2^{N-1} + \ldots + a_1 2 + a_0 + a_{-1} 2^{-1} + a_{-2} 2^{-2} + a_{-3} 2^{-3} + \ldots$$

For example

$$110101.110111111\ldots = 110101.110\bar{1} =$$

$$= 1\times 2^5 + 1\times 2^4 + 0\times 2^3 + 1\times 2^2 + 0\times 2 + 1 + \frac{1}{2} + \frac{1}{2^2} + \frac{0}{2^3} + \frac{1}{2^4} + \frac{1}{2^5} + \frac{1}{2^6} + \ldots =$$

$$= 32 + 16 + 4 + 1 + \frac{1}{2} + \frac{1}{4} + \sum_{i=0}^{\infty} \frac{1}{2^4} \left(\frac{1}{2}\right)^i =$$

Binary Representation of Real Numbers

A real number A is represented by a **binary** code of the form $a_N a_{N-1} \ldots a_1 a_0 . a_{-1} a_{-2} \ldots$, where each a_i is 0 or 1.

The code $a_N a_{N-1} \ldots a_1 a_0 . a_{-1} a_{-2} \ldots$ represents the value of the series

$$a_N 2^N + a_{N-1} 2^{N-1} + \ldots + a_1 2 + a_0 + a_{-1} 2^{-1} + a_{-2} 2^{-2} + a_{-3} 2^{-3} + \ldots$$

For example

$$110101.110111111\ldots = 110101.110\bar{1} =$$

$$= 1\times 2^5 + 1\times 2^4 + 0\times 2^3 + 1\times 2^2 + 0\times 2 + 1 + \frac{1}{2} + \frac{1}{2^2} + \frac{0}{2^3} + \frac{1}{2^4} + \frac{1}{2^5} + \frac{1}{2^6} + \ldots =$$

$$= 32 + 16 + 4 + 1 + \frac{1}{2} + \frac{1}{4} + \sum_{i=0}^{\infty} \frac{1}{2^4} \left(\frac{1}{2}\right)^i =$$

$$= 53.75 + \frac{1}{16} \sum_{i=0}^{\infty} \left(\frac{1}{2}\right)^i$$

Binary Representation of Real Numbers

A real number A is represented by a **binary** code of the form $a_N a_{N-1} \ldots a_1 a_0 . a_{-1} a_{-2} \ldots$, where each a_i is 0 or 1.

The code $a_N a_{N-1} \ldots a_1 a_0 . a_{-1} a_{-2} \ldots$ represents the value of the series

$$a_N 2^N + a_{N-1} 2^{N-1} + \ldots + a_1 2 + a_0 + a_{-1} 2^{-1} + a_{-2} 2^{-2} + a_{-3} 2^{-3} + \ldots$$

For example

$110101.110111111\ldots = 110101.110\bar{1} =$

$= 1\times 2^5 + 1\times 2^4 + 0\times 2^3 + 1\times 2^2 + 0\times 2 + 1 + \frac{1}{2} + \frac{1}{2^2} + \frac{0}{2^3} + \frac{1}{2^4} + \frac{1}{2^5} + \frac{1}{2^6} + \ldots =$

$= 32 + 16 + 4 + 1 + \frac{1}{2} + \frac{1}{4} + \sum_{i=0}^{\infty} \frac{1}{2^4} (\frac{1}{2})^i =$

$= 53.75 + \frac{1}{16} \sum_{i=0}^{\infty} (\frac{1}{2})^i = 53.75 + (\frac{1}{16})(2) = 53.875.$

Binary Representation of Real Numbers

A real number A is represented by a **binary** code of the form $a_N a_{N-1} \ldots a_1 a_0 . a_{-1} a_{-2} \ldots$, where each a_i is 0 or 1.

The code $a_N a_{N-1} \ldots a_1 a_0 . a_{-1} a_{-2} \ldots$ represents the value of the series

$$a_N 2^N + a_{N-1} 2^{N-1} + \ldots + a_1 2 + a_0 + a_{-1} 2^{-1} + a_{-2} 2^{-2} + a_{-3} 2^{-3} + \ldots$$

When $a_i = 0$ for all $i \leq -k - 1$, we omit all the zero digits after the kth binary place and we write the code as finite sum

$$a_N a_{N-1} \ldots a_1 a_0 . a_{-1} a_{-2} \ldots a_{-k} .$$

Binary Representation of Real Numbers

A real number A is represented by a **binary** code of the form $a_N a_{N-1} \ldots a_1 a_0 . a_{-1} a_{-2} \ldots$, where each a_i is 0 or 1.

The code $a_N a_{N-1} \ldots a_1 a_0 . a_{-1} a_{-2} \ldots$ represents the value of the series

$$a_N 2^N + a_{N-1} 2^{N-1} + \ldots + a_1 2 + a_0 + a_{-1} 2^{-1} + a_{-2} 2^{-2} + a_{-3} 2^{-3} + \ldots$$

When $a_i = 0$ for all $i \leq -k - 1$, we omit all the zero digits after the kth binary place and we write the code as finite sum

$$a_N a_{N-1} \ldots a_1 a_0 . a_{-1} a_{-2} \ldots a_{-k} .$$

The only way two different binary representations can describe the same number is

$$a_N a_{N-1} \ldots a_1 a_0 . a_{-1} a_{-2} \ldots a_{-k} 1\bar{0} = a_N a_{N-1} \ldots a_1 a_0 . a_{-1} a_{-2} \ldots \hat{a_{-k}} 0\bar{1}$$

Binary Representation of Real Numbers

A real number A is represented by a **binary** code of the form $a_N a_{N-1} \ldots a_1 a_0 . a_{-1} a_{-2} \ldots$, where each a_i is 0 or 1.

The code $a_N a_{N-1} \ldots a_1 a_0 . a_{-1} a_{-2} \ldots$ represents the value of the series

$$a_N 2^N + a_{N-1} 2^{N-1} + \ldots + a_1 2 + a_0 + a_{-1} 2^{-1} + a_{-2} 2^{-2} + a_{-3} 2^{-3} + \ldots$$

The only way two different binary representations can describe the same number is

$$a_N a_{N-1} \ldots a_1 a_0 . a_{-1} a_{-2} \ldots a_{-k} 1\bar{0} = a_N a_{N-1} \ldots a_1 a_0 . a_{-1} a_{-2} \ldots \hat{a_{-k}} 0\bar{1}$$

For example $110101.110111111\ldots = 110101.110\bar{1} = 110101.111$.

Binary Representation of Real Numbers

The only way two different binary representations can describe the same number is

$$a_N a_{N-1} \ldots a_1 a_0 . a_{-1} a_{-2} \ldots a_{-k} 1\bar{0} = a_N a_{N-1} \ldots a_1 a_0 . a_{-1} a_{-2} \ldots \hat{a_{-k}} 0\bar{1}$$

For example $110101.110111111\ldots = 110101.110\bar{1} = 110101.111$.

Or $10000 = 1111.\bar{1}$ both represent the integer 16.

Binary Representation of Real Numbers

The only way two different binary representations can describe the same number is

$$a_N a_{N-1} \ldots a_1 a_0 . a_{-1} a_{-2} \ldots a_{-k} 1\bar{0} = a_N a_{N-1} \ldots a_1 a_0 . a_{-1} a_{-2} \ldots \hat{a_{-k}} 0\bar{1}$$

For example $110101.110111111\ldots = 110101.110\bar{1} = 110101.111$.

Or $10000 = 1111.\bar{1}$ both represent the integer 16.

For any positive integer n when a number is multiplied by 2^n, the binary point in its binary code is shifted n places to the right,

Binary Representation of Real Numbers

The only way two different binary representations can describe the same number is

$$a_N a_{N-1} \ldots a_1 a_0 . a_{-1} a_{-2} \ldots a_{-k} 1\bar{0} = a_N a_{N-1} \ldots a_1 a_0 . a_{-1} a_{-2} \ldots \hat{a_{-k}} 0\bar{1}$$

For example $110101.110111111\ldots = 110101.110\bar{1} = 110101.111$.

Or $10000 = 1111.\bar{1}$ both represent the integer 16.

For any positive integer n when a number is multiplied by 2^n, the binary point in its binary code is shifted n places to the right,

and when a number is multiplied by 2^{-n}, the binary point in its binary code is shifted n places to the left.

Binary Representation of Real Numbers

For any positive integer n when a number is multiplied by 2^n, the binary point in its binary code is shifted n places to the right,

and when a number is multiplied by 2^{-n}, the binary point in its binary code is shifted n places to the left.

Given any **positive** number A, we can find the digits in the binary representation of A from the left to the right, one by one, in the following way.

Binary Representation of Real Numbers

Given any **positive** number A, we can find the digits in the binary representation of A from the left to the right, one by one, in the following way.

First we find the largest integer N such that $2^N < A$.

Binary Representation of Real Numbers

Given any **positive** number A, we can find the digits in the binary representation of A from the left to the right, one by one, in the following way.

First we find the largest integer N such that $2^N < A$. The **binary code** for A, then will starts with $a_N = 1$ on the left.

Binary Representation of Real Numbers

Given any **positive** number A, we can find the digits in the binary representation of A from the left to the right, one by one, in the following way.

First we find the largest integer N such that $2^N < A$. The **binary code** for A, then will starts with $a_N = 1$ on the left.

We then let $A_1 = A - 2^N$ and we let $a_{N-1} = 1$ if $A_1 \geq 2^{N-1}$ and we let $a_{N-1} = 0$ if $A_1 < 2^{N-1}$. In both cases, we now let $A_2 = A_1 - a_{N-1}2^{N-1}$.

Binary Representation of Real Numbers

Given any **positive** number A, we can find the digits in the binary representation of A from the left to the right, one by one, in the following way.

First we find the largest integer N such that $2^N < A$. The **binary code** for A, then will starts with $a_N = 1$ on the left.

We then let $A_1 = A - 2^N$ and we let $a_{N-1} = 1$ if $A_1 \geq 2^{N-1}$ and we let $a_{N-1} = 0$ if $A_1 < 2^{N-1}$. In both cases, we now let $A_2 = A_1 - a_{N-1}2^{N-1}$.

At each step, if a_N, a_{N-1}, ..., a_{N-k} are found and the value of A_{N-k} is determined, we let $a_{N-k-1} = 1$ if $A_{N-k} \geq 2^{N-k-1}$ and we let $a_{N-k-1} = 0$ if $A_{N-k} < 2^{N-k-1}$. In both cases, we now let $A_{N-k-1} = A_{N-k} - a_{N-k-1}2^{N-k-1}$.

Binary Representation of Real Numbers

First we find the largest integer N such that $2^N < A$. The **binary code** for A, then will starts with $a_N = 1$ on the left.

We then let $A_1 = A - 2^N$ and we let $a_{N-1} = 1$ if $A_1 \geq 2^{N-1}$ and we let $a_{N-1} = 0$ if $A_1 < 2^{N-1}$. In both cases, we now let $A_2 = A_1 - a_{N-1}2^{N-1}$.

At each step, if a_N, a_{N-1}, ..., a_{N-k} are found and the value of A_{N-k} is determined, we let $a_{N-k-1} = 1$ if $A_{N-k} \geq 2^{N-k-1}$ and we let $a_{N-k-1} = 0$ if $A_{N-k} < 2^{N-k-1}$. In both cases, we now let $A_{N-k-1} = A_{N-k} - a_{N-k-1}2^{N-k-1}$.

Example: Find the binary expansion of 37.4 up to the third binary place.

Binary Representation of Real Numbers

First we find the largest integer N such that $2^N < A$. The **binary code** for A, then will starts with $a_N = 1$ on the left.

We then let $A_1 = A - 2^N$ and we let $a_{N-1} = 1$ if $A_1 \geq 2^{N-1}$ and we let $a_{N-1} = 0$ if $A_1 < 2^{N-1}$. In both cases, we now let $A_2 = A_1 - a_{N-1}2^{N-1}$.

At each step, if a_N, a_{N-1}, ..., a_{N-k} are found and the value of A_{N-k} is determined, we let $a_{N-k-1} = 1$ if $A_{N-k} \geq 2^{N-k-1}$ and we let $a_{N-k-1} = 0$ if $A_{N-k} < 2^{N-k-1}$. In both cases, we now let $A_{N-k-1} = A_{N-k} - a_{N-k-1}2^{N-k-1}$.

Example: Find the binary expansion of 37.4 up to the third binary place.

Solution: 100101.011.

Floating Point Representation of Numbers

In computing devices, a fixed number of bits are dedicated to representing a number.

Floating Point Representation of Numbers

In computing devices, a fixed number of bits are dedicated to representing a number.

One bit is dedicated to specify the sign of the number (0 for positive and 1 for negative).

Floating Point Representation of Numbers

In computing devices, a fixed number of bits are dedicated to representing a number.

One bit is dedicated to specify the sign of the number (0 for positive and 1 for negative).

We use the rest of the bits based on the floating point notation.

Floating Point Representation of Numbers

In computing devices, a fixed number of bits are dedicated to representing a number.

One bit is dedicated to specify the sign of the number (0 for positive and 1 for negative).

We use the rest of the bits based on the floating point notation. The decimal version of representing a decimal code $a_N a_{N-1} \ldots a_1 a_0 . a_{-1} a_{-2} \ldots$ using the floating point notation is $0.a_N a_{N-1} \ldots a_1 a_0 a_{-1} a_{-2} \ldots \times 10^{N+1}$.

Floating Point Representation of Numbers

In computing devices, a fixed number of bits are dedicated to representing a number.

One bit is dedicated to specify the sign of the number (0 for positive and 1 for negative).

We use the rest of the bits based on the floating point notation. The decimal version of representing a decimal code $a_N a_{N-1} \ldots a_1 a_0 . a_{-1} a_{-2} \ldots$ using the floating point notation is $0.a_N a_{N-1} \ldots a_1 a_0 a_{-1} a_{-2} \ldots \times 10^{N+1}$.

For example, 7302.346 is represented as 0.7302346×10^4 and 0.00316 is represented as 0.316×10^{-2}.

Floating Point Representation of Numbers

In computing devices, a fixed number of bits are dedicated to representing a number.

One bit is dedicated to specify the sign of the number (0 for positive and 1 for negative).

We use the rest of the bits based on the floating point notation. The decimal version of representing a decimal code $a_N a_{N-1} \ldots a_1 a_0 . a_{-1} a_{-2} \ldots$ using the floating point notation is $0.a_N a_{N-1} \ldots a_1 a_0 a_{-1} a_{-2} \ldots \times 10^{N+1}$.

For example, 7302.346 is represented as 0.7302346×10^4 and 0.00316 is represented as 0.316×10^{-2}.

The number $m = 0.a_N a_{N-1} \ldots a_1 a_0 a_{-1} a_{-2} \ldots$ is called **mantissa** and the number $N + 1$ is called the **exponent**.

Floating Point Representation of Numbers

In computing devices, a fixed number of bits are dedicated to representing a number.

One bit is dedicated to specify the sign of the number (0 for positive and 1 for negative).

We use the rest of the bits based on the floating point notation. The decimal version of representing a decimal code $a_N a_{N-1} \ldots a_1 a_0 . a_{-1} a_{-2} \ldots$ using the floating point notation is $0.a_N a_{N-1} \ldots a_1 a_0 a_{-1} a_{-2} \ldots \times 10^{N+1}$.

For example, 7302.346 is represented as 0.7302346×10^4 and 0.00316 is represented as 0.316×10^{-2}.

The number $m = 0.a_N a_{N-1} \ldots a_1 a_0 a_{-1} a_{-2} \ldots$ is called **mantissa** and the number $N+1$ is called the **exponent**. Mantissa always satisfies $0 \leq m < 1$ and the exponent is always an integer (positive, negative, or zero).

Floating Point Representation of Numbers

In computing devices, a fixed number of bits are dedicated to representing a number.

One bit is dedicated to specify the sign of the number (0 for positive and 1 for negative).

We use the rest of the bits based on the floating point notation. Representing a binary code $a_N a_{N-1} \ldots a_1 a_0 . a_{-1} a_{-2} \ldots$ using the floating point notation is
$a_N . a_{N-1} \ldots a_1 a_0 a_{-1} a_{-2} \ldots \times 2^N = 1 . a_{N-1} \ldots a_1 a_0 a_{-1} a_{-2} \ldots \times 2^N$.

Floating Point Representation of Numbers

In computing devices, a fixed number of bits are dedicated to representing a number.

One bit is dedicated to specify the sign of the number (0 for positive and 1 for negative).

We use the rest of the bits based on the floating point notation. Representing a binary code $a_N a_{N-1} \ldots a_1 a_0 . a_{-1} a_{-2} \ldots$ using the floating point notation is $a_N . a_{N-1} \ldots a_1 a_0 a_{-1} a_{-2} \ldots \times 2^N = 1 . a_{N-1} \ldots a_1 a_0 a_{-1} a_{-2} \ldots \times 2^N$. The reason we move a_N to the left of the binary point is that we know it is always 1 (unlike the decimal case where it can be any of the nine non-zero digits).

Floating Point Representation of Numbers

In computing devices, a fixed number of bits are dedicated to representing a number.

One bit is dedicated to specify the sign of the number (0 for positive and 1 for negative).

The remaining bits are split into r bits for **mantissa** m, and s bits for the **characteristic** or **shifted exponent** f.

Floating Point Representation of Numbers

In computing devices, a fixed number of bits are dedicated to representing a number.

One bit is dedicated to specify the sign of the number (0 for positive and 1 for negative).

The remaining bits are split into r bits for **mantissa** m, and s bits for the **characteristic** or **shifted exponent** f.

The r bits for mantissa record the r binary places after the binary point of mantissa (since 1 before the point is always the same value).

Floating Point Representation of Numbers

In computing devices, a fixed number of bits are dedicated to representing a number.

One bit is dedicated to specify the sign of the number (0 for positive and 1 for negative).

The remaining bits are split into r bits for **mantissa** m, and s bits for the **characteristic** or **shifted exponent** f.

The r bits for mantissa record the r binary places after the binary point of mantissa (since 1 before the point is always the same value).

The s bits dedicated to the shifted exponent represent 2^s possible integer values ranging from 0 to $2^s - 1$.

Floating Point Representation of Numbers

In computing devices, a fixed number of bits are dedicated to representing a number.

One bit is dedicated to specify the sign of the number (0 for positive and 1 for negative).

The remaining bits are split into r bits for **mantissa** m, and s bits for the **characteristic** or **shifted exponent** f.

The r bits for mantissa record the r binary places after the binary point of mantissa (since 1 before the point is always the same value).

The s bits dedicated to the shifted exponent represent 2^s possible integer values ranging from 0 to $2^s - 1$. Since we also need negative values for the exponent in the floating point notation, the actual exponent e of the number is $f - 2^{s-1} + 1$.

Floating Point Representation of Numbers

In computing devices, a fixed number of bits are dedicated to representing a number.

One bit is dedicated to specify the sign of the number (0 for positive and 1 for negative).

The remaining bits are split into r bits for **mantissa** m, and s bits for the **characteristic** or **shifted exponent** f.

The r bits for mantissa record the r binary places after the binary point of mantissa (since 1 before the point is always the same value).

The s bits dedicated to the shifted exponent represent 2^s possible integer values ranging from 0 to $2^s - 1$. Since we also need negative values for the exponent in the floating point notation, the actual exponent e of the number is $f - 2^{s-1} + 1$. This allows for exponents between $-2^{s-1} + 1$ and 2^{s-1} to be covered as possible exponents in the floating point representation.

A 19 Bit Binary Number System Example

A calculator uses 19 bit binary numbers with one bit for the sign of the number, followed by 11 bits for mantissa m, followed by 7 bits for the characteristic f.

A 19 Bit Binary Number System Example

A calculator uses 19 bit binary numbers with one bit for the sign of the number, followed by 11 bits for mantissa m, followed by 7 bits for the characteristic f.

(1) How does this calculator represent 570.3?

A 19 Bit Binary Number System Example

A calculator uses 19 bit binary numbers with one bit for the sign of the number, followed by 11 bits for mantissa m, followed by 7 bits for the characteristic f.

(1) How does this calculator represent 570.3?

$570 = 512 + 32 + 16 + 8 + 2 = 2^9 + 2^5 + 2^4 + 2^3 + 2^1$

A 19 Bit Binary Number System Example

A calculator uses 19 bit binary numbers with one bit for the sign of the number, followed by 11 bits for mantissa m, followed by 7 bits for the characteristic f.

(1) How does this calculator represent 570.3?

$570 = 512 + 32 + 16 + 8 + 2 = 2^9 + 2^5 + 2^4 + 2^3 + 2^1$

570.3 has binary code 1000111010.01 up to 12 digits used.

A 19 Bit Binary Number System Example

A calculator uses 19 bit binary numbers with one bit for the sign of the number, followed by 11 bits for mantissa m, followed by 7 bits for the characteristic f.

(1) How does this calculator represent 570.3?

$570 = 512 + 32 + 16 + 8 + 2 = 2^9 + 2^5 + 2^4 + 2^3 + 2^1$

570.3 has binary code 1000111010.01 up to 12 digits used.

In floating point notation we write 570.3 as 1.00011101001×2^9.

A 19 Bit Binary Number System Example

A calculator uses 19 bit binary numbers with one bit for the sign of the number, followed by 11 bits for mantissa m, followed by 7 bits for the characteristic f.

(1) How does this calculator represent 570.3?

$570 = 512 + 32 + 16 + 8 + 2 = 2^9 + 2^5 + 2^4 + 2^3 + 2^1$

570.3 has binary code 1000111010.01 up to 12 digits used.

In floating point notation we write 570.3 as 1.00011101001×2^9.

With 7 bits for exponent we can generate integer codes between $0 = 0000000$ and $127 = 2^7 - 1 = 1111111$.

A 19 Bit Binary Number System Example

A calculator uses 19 bit binary numbers with one bit for the sign of the number, followed by 11 bits for mantissa m, followed by 7 bits for the characteristic f.

(1) How does this calculator represent 570.3?

$570 = 512 + 32 + 16 + 8 + 2 = 2^9 + 2^5 + 2^4 + 2^3 + 2^1$

570.3 has binary code 1000111010.01 up to 12 digits used.

In floating point notation we write 570.3 as 1.00011101001×2^9.

With 7 bits for exponent we can generate integer codes between $0 = 0000000$ and $127 = 2^7 - 1 = 1111111$.

To almost equally cover the negative and positive exponents we shift the integer value of the exponent code by $2^6 - 1 = 63$ which results in exponent values between -63 and 64.

A 19 Bit Binary Number System Example

A calculator uses 19 bit binary numbers with one bit for the sign of the number, followed by 11 bits for mantissa m, followed by 7 bits for the characteristic f.

(1) How does this calculator represent 570.3?

In floating point notation we write 570.3 as 1.00011101001×2^9.

With 7 bits for exponent we can generate integer codes between $0 = 0000000$ and $127 = 2^7 - 1 = 1111111$.

To almost equally cover the negative and positive exponents we shift the integer value of the exponent code by $2^6 - 1 = 63$ which results in exponent values between -63 and 64.

This means the actual exponent 9 should be represented by the binary code for $9 + 63 = 72$.

A 19 Bit Binary Number System Example

A calculator uses 19 bit binary numbers with one bit for the sign of the number, followed by 11 bits for mantissa m, followed by 7 bits for the characteristic f.

(1) How does this calculator represent 570.3?

In floating point notation we write 570.3 as 1.00011101001×2^9.

With 7 bits for exponent we can generate integer codes between $0 = 0000000$ and $127 = 2^7 - 1 = 1111111$.

To almost equally cover the negative and positive exponents we shift the integer value of the exponent code by $2^6 - 1 = 63$ which results in exponent values between -63 and 64.

This means the actual exponent 9 should be represented by the binary code for $9 + 63 = 72$. $72 = 64 + 8 = 2^6 + 2^3$ is represented by the 7 digit binary code 1001000.

A 19 Bit Binary Number System Example

A calculator uses 19 bit binary numbers with one bit for the sign of the number, followed by 11 bits for mantissa m, followed by 7 bits for the characteristic f.

(1) How does this calculator represent 570.3?

In floating point notation we write 570.3 as 1.00011101001×2^9.

To almost equally cover the negative and positive exponents we shift the integer value of the exponent code by $2^6 - 1 = 63$ which results in exponent values between -63 and 64.

This means the actual exponent 9 should be represented by the binary code for $9 + 63 = 72$. $72 = 64 + 8 = 2^6 + 2^3$ is represented by the 7 digit binary code 1001000.

570.3 is positive, making the first code take the value 0, we found the 11 digit code for mantissa to be 00011101001, and the 7 digit code for the characteristic to be 1001000.

A 19 Bit Binary Number System Example

A calculator uses 19 bit binary numbers with one bit for the sign of the number, followed by 11 bits for mantissa m, followed by 7 bits for the characteristic f.

(1) How does this calculator represent 570.3?

In floating point notation we write 570.3 as 1.00011101001×2^9.

This means the actual exponent 9 should be represented by the binary code for $9 + 63 = 72$. $72 = 64 + 8 = 2^6 + 2^3$ is represented by the 7 digit binary code 1001000.

570.3 is positive, making the first code take the value 0, we found the 11 digit code for mantissa to be 00011101001, and the 7 digit code for the characteristic to be 1001000.

Therefore, 570.3 is represented by the following 19 bit code.

$$0|00011101001|1001000$$

A 19 Bit Binary Number System Example

A calculator uses 19 bit binary numbers with one bit for the sign of the number, followed by 11 bits for mantissa m, followed by 7 bits for the characteristic f.

(2) How does this calculator represent $-\frac{1}{3}$?

A 19 Bit Binary Number System Example

A calculator uses 19 bit binary numbers with one bit for the sign of the number, followed by 11 bits for mantissa m, followed by 7 bits for the characteristic f.

(2) How does this calculator represent $-\frac{1}{3}$?

The binary representation of $\frac{1}{3}$ is $0.01010101\ldots = 0.\overline{01}$

A 19 Bit Binary Number System Example

A calculator uses 19 bit binary numbers with one bit for the sign of the number, followed by 11 bits for mantissa m, followed by 7 bits for the characteristic f.

(2) How does this calculator represent $-\frac{1}{3}$?

The binary representation of $\frac{1}{3}$ is $0.01010101... = 0.\overline{01}$

$$\frac{1}{4} + \frac{1}{16} + \frac{1}{64} + \frac{1}{256} + ... = \frac{\frac{1}{4}}{1 - \frac{1}{4}} = \frac{1}{3}$$

(**geometric series** with the first term $\frac{1}{4}$ and geometric factor $\frac{1}{4}$.)

A 19 Bit Binary Number System Example

A calculator uses 19 bit binary numbers with one bit for the sign of the number, followed by 11 bits for mantissa m, followed by 7 bits for the characteristic f.

(2) How does this calculator represent $-\frac{1}{3}$?

The binary representation of $\frac{1}{3}$ is $0.01010101\ldots = 0.\overline{01}$

The binary floating point representation of $\frac{1}{3}$ is $1.010101\ldots \times 2^{-2} = 0.\overline{01} \times 2^{-2}$

A 19 Bit Binary Number System Example

A calculator uses 19 bit binary numbers with one bit for the sign of the number, followed by 11 bits for mantissa m, followed by 7 bits for the characteristic f.

(2) How does this calculator represent $-\frac{1}{3}$?

The binary representation of $\frac{1}{3}$ is $0.01010101\ldots = 0.\overline{01}$

The binary floating point representation of $\frac{1}{3}$ is $1.010101\ldots \times 2^{-2} = 0.\overline{01} \times 2^{-2}$

The actual exponent -2 should be represented by the binary code for $-2 + 63 = 61$.

A 19 Bit Binary Number System Example

A calculator uses 19 bit binary numbers with one bit for the sign of the number, followed by 11 bits for mantissa m, followed by 7 bits for the characteristic f.

(2) How does this calculator represent $-\frac{1}{3}$?

The binary representation of $\frac{1}{3}$ is $0.01010101\ldots = 0.\overline{01}$

The binary floating point representation of $\frac{1}{3}$ is $1.010101\ldots \times 2^{-2} = 0.\overline{01} \times 2^{-2}$

The actual exponent -2 should be represented by the binary code for $-2 + 63 = 61$. $61 = 32 + 16 + 8 + 4 + 1 = 2^5 + 2^4 + 2^3 + 2^2 + 1$ is represented by the 7 digit binary code 0111101.

A 19 Bit Binary Number System Example

A calculator uses 19 bit binary numbers with one bit for the sign of the number, followed by 11 bits for mantissa m, followed by 7 bits for the characteristic f.

(2) How does this calculator represent $-\frac{1}{3}$?

The binary floating point representation of $\frac{1}{3}$ is $1.010101\ldots \times 2^{-2} = 0.\overline{01} \times 2^{-2}$

The actual exponent -2 should be represented by the binary code for $-2 + 63 = 61$. $61 = 32 + 16 + 8 + 4 + 1 = 2^5 + 2^4 + 2^3 + 2^2 + 1$ is represented by the 7 digit binary code 0111101.

$-\frac{1}{3}$ is negative, making the first code take the value 1, we found the 11 digit code for mantissa to be 01010101010, and the 7 digit code for the characteristic to be 0111101.

A 19 Bit Binary Number System Example

A calculator uses 19 bit binary numbers with one bit for the sign of the number, followed by 11 bits for mantissa m, followed by 7 bits for the characteristic f.

(2) How does this calculator represent $-\frac{1}{3}$?

The binary floating point representation of $\frac{1}{3}$ is $1.010101\ldots \times 2^{-2} = 0.\overline{01} \times 2^{-2}$

$-\frac{1}{3}$ is negative, making the first code take the value 1, we found the 11 digit code for mantissa to be 01010101010, and the 7 digit code for the characteristic to be 0111101.

Therefore, $-\frac{1}{3}$ is represented by the following 19 bit code.

$$1|01010101010|0111101$$

A 19 Bit Binary Number System Example

A calculator uses 19 bit binary numbers with one bit for the sign of the number, followed by 11 bits for mantissa m, followed by 7 bits for the characteristic f.

(3) What is the largest number this calculator can express exactly?

A 19 Bit Binary Number System Example

A calculator uses 19 bit binary numbers with one bit for the sign of the number, followed by 11 bits for mantissa m, followed by 7 bits for the characteristic f.

(3) What is the largest number this calculator can express exactly?

We get the largest possible value using the code

$$0|11111111111|1111111$$

A 19 Bit Binary Number System Example

A calculator uses 19 bit binary numbers with one bit for the sign of the number, followed by 11 bits for mantissa m, followed by 7 bits for the characteristic f.

(3) What is the largest number this calculator can express exactly?

We get the largest possible value using the code

$$0|11111111111|1111111$$

Binary code 1111111 for $1 + 2 + 4 + 8 + 16 + 32 + 64 = 127$ corresponds with the exponent 64.

A 19 Bit Binary Number System Example

A calculator uses 19 bit binary numbers with one bit for the sign of the number, followed by 11 bits for mantissa m, followed by 7 bits for the characteristic f.

(3) What is the largest number this calculator can express exactly?

We get the largest possible value using the code

$$0|11111111111|1111111$$

Binary code 1111111 for $1 + 2 + 4 + 8 + 16 + 32 + 64 = 127$ corresponds with the exponent 64.

The code 01111111111111111111 corresponds with the binary floating point representation for $1.11111111111 \times 2^{64}$

A 19 Bit Binary Number System Example

A calculator uses 19 bit binary numbers with one bit for the sign of the number, followed by 11 bits for mantissa m, followed by 7 bits for the characteristic f.

(3) What is the largest number this calculator can express exactly?

We get the largest possible value using the code

$$0|11111111111|1111111$$

Binary code 1111111 for $1+2+4+8+16+32+64=127$ corresponds with the exponent 64.

The code 01111111111111111 corresponds with the binary floating point representation for $1.11111111111 \times 2^{64}$ corresponding with

$$2^{53}(1+2+2^2+2^3+...+2^9+2^{10}+2^{11})$$

A 19 Bit Binary Number System Example

A calculator uses 19 bit binary numbers with one bit for the sign of the number, followed by 11 bits for mantissa m, followed by 7 bits for the characteristic f.

(3) What is the largest number this calculator can express exactly?

We get the largest possible value using the code

$$0|11111111111|1111111$$

Binary code 1111111 for $1+2+4+8+16+32+64=127$ corresponds with the exponent 64.

The code 01111111111111111 corresponds with the binary floating point representation for $1.11111111111 \times 2^{64}$ corresponding with

$$2^{53}(1+2+2^2+2^3+\ldots+2^9+2^{10}+2^{11}) = 2^{53}(2^{12}-1)$$

A 19 Bit Binary Number System Example

A calculator uses 19 bit binary numbers with one bit for the sign of the number, followed by 11 bits for mantissa m, followed by 7 bits for the characteristic f.

(3) What is the largest number this calculator can express exactly?

We get the largest possible value using the code

$$0|11111111111|1111111$$

Binary code 1111111 for $1+2+4+8+16+32+64=127$ corresponds with the exponent 64.

The code 01111111111111111 corresponds with the binary floating point representation for $1.11111111111 \times 2^{64}$ corresponding with

$$2^{53}(1+2+2^2+2^3+...+2^9+2^{10}+2^{11}) = 2^{53}(2^{12}-1) = 36884480948164362240$$

A 19 Bit Binary Number System Example

A calculator uses 19 bit binary numbers with one bit for the sign of the number, followed by 11 bits for mantissa m, followed by 7 bits for the characteristic f.

(4) What is the smallest **positive** number this calculator can express as a non-zero code?

A 19 Bit Binary Number System Example

A calculator uses 19 bit binary numbers with one bit for the sign of the number, followed by 11 bits for mantissa m, followed by 7 bits for the characteristic f.

(4) What is the smallest **positive** number this calculator can express as a non-zero code?

We get the smallest possible positive value using the code

$$0|00000000000|0000000$$

A 19 Bit Binary Number System Example

A calculator uses 19 bit binary numbers with one bit for the sign of the number, followed by 11 bits for mantissa m, followed by 7 bits for the characteristic f.

(4) What is the smallest **positive** number this calculator can express as a non-zero code?

We get the smallest possible positive value using the code

$$0|00000000000|0000000$$

which corresponds with the floating point representation

$$1.00000000000 \times 2^{-63}$$

A 19 Bit Binary Number System Example

A calculator uses 19 bit binary numbers with one bit for the sign of the number, followed by 11 bits for mantissa m, followed by 7 bits for the characteristic f.

(4) What is the smallest **positive** number this calculator can express as a non-zero code?

We get the smallest possible positive value using the code

$$0|00000000000|0000000$$

which corresponds with the floating point representation

$$1.00000000000 \times 2^{-63}$$

Therefore, the smallest positive number this calculator can represent is 2^{-63}.

A 19 Bit Binary Number System Example

A calculator uses 19 bit binary numbers with one bit for the sign of the number, followed by 11 bits for mantissa m, followed by 7 bits for the characteristic f.

(5) What interval of numbers are represented by this calculator with the code 0|00000000110|0010101?

A 19 Bit Binary Number System Example

A calculator uses 19 bit binary numbers with one bit for the sign of the number, followed by 11 bits for mantissa m, followed by 7 bits for the characteristic f.

(5) What interval of numbers are represented by this calculator with the code 0|00000000110|0010101?

A 19 Bit Binary Number System Example

A calculator uses 19 bit binary numbers with one bit for the sign of the number, followed by 11 bits for mantissa m, followed by 7 bits for the characteristic f.

(5) What interval of numbers are represented by this calculator with the code 0|00000000110|0010101?

The smallest number represented by this code is $1.000000001100 \times 2^{-42} = 2^{-42} + 2^{-51} + 2^{-52}$.

A 19 Bit Binary Number System Example

A calculator uses 19 bit binary numbers with one bit for the sign of the number, followed by 11 bits for mantissa m, followed by 7 bits for the characteristic f.

(5) What interval of numbers are represented by this calculator with the code $0|00000000110|0010101$?

The smallest number represented by this code is $1.000000001100 \times 2^{-42} = 2^{-42} + 2^{-51} + 2^{-52}$.

The upper end point of numbers represented by this code is $1.00000000111 \times 2^{-42} = 1.000000001101 \times 2^{-42} = 2^{-42} + 2^{-51} + 2^{-52} + 2^{-53}$. This number itself, however, is not represented by this code.

A 19 Bit Binary Number System Example

A calculator uses 19 bit binary numbers with one bit for the sign of the number, followed by 11 bits for mantissa m, followed by 7 bits for the characteristic f.

(5) What interval of numbers are represented by this calculator with the code $0|00000000110|0010101$?

The smallest number represented by this code is $1.000000001100 \times 2^{-42} = 2^{-42} + 2^{-51} + 2^{-52}$.

The upper end point of numbers represented by this code is $1.00000000111 \times 2^{-42} = 1.000000001101 \times 2^{-42} = 2^{-42} + 2^{-51} + 2^{-52} + 2^{-53}$. This number itself, however, is not represented by this code.

Therefore the interval of numbers $[\frac{1027}{4503599627370496}, \frac{2055}{9007199254740992})$ of size $2^{-53} = \frac{1}{9007199254740992}$ are all represented by this same code.

A 19 Bit Binary Number System Example

A calculator uses 19 bit binary numbers with one bit for the sign of the number, followed by 11 bits for mantissa m, followed by 7 bits for the characteristic f.

(6) What interval of numbers are represented by this calculator with the code 0|11000000000|1011101?

A 19 Bit Binary Number System Example

A calculator uses 19 bit binary numbers with one bit for the sign of the number, followed by 11 bits for mantissa m, followed by 7 bits for the characteristic f.

(6) What interval of numbers are represented by this calculator with the code 0|11000000000|1011101?

The code 1011101 for characteristic corresponds with the exponent $1 + 4 + 8 + 16 + 64 - 63 = 30$.

A 19 Bit Binary Number System Example

A calculator uses 19 bit binary numbers with one bit for the sign of the number, followed by 11 bits for mantissa m, followed by 7 bits for the characteristic f.

(6) What interval of numbers are represented by this calculator with the code $0|11000000000|1011101$?

The code 1011101 for characteristic corresponds with the exponent $1 + 4 + 8 + 16 + 64 - 63 = 30$.

The smallest number represented by this code is $1.110000000000 \times 2^{30} = 2^{30} + 2^{29} + 2^{28}$.

A 19 Bit Binary Number System Example

A calculator uses 19 bit binary numbers with one bit for the sign of the number, followed by 11 bits for mantissa m, followed by 7 bits for the characteristic f.

(6) What interval of numbers are represented by this calculator with the code $0|11000000000|1011101$?

The code 1011101 for characteristic corresponds with the exponent $1 + 4 + 8 + 16 + 64 - 63 = 30$.

The smallest number represented by this code is $1.110000000000\bar{0} \times 2^{30} = 2^{30} + 2^{29} + 2^{28}$.

The upper end point of numbers represented by this code is $1.110000000001\bar{1} \times 2^{30} = 1.11000000001 \times 2^{30} = 2^{30} + 2^{29} + 2^{28} + 2^{19}$. This number itself, however, is not represented by this code.

A 19 Bit Binary Number System Example

A calculator uses 19 bit binary numbers with one bit for the sign of the number, followed by 11 bits for mantissa m, followed by 7 bits for the characteristic f.

(6) What interval of numbers are represented by this calculator with the code $0|11000000000|1011101$?

The code 1011101 for characteristic corresponds with the exponent $1 + 4 + 8 + 16 + 64 - 63 = 30$.

The smallest number represented by this code is $1.110000000000\bar{0} \times 2^{30} = 2^{30} + 2^{29} + 2^{28}$.

The upper end point of numbers represented by this code is $1.110000000001\bar{1} \times 2^{30} = 1.11000000001 \times 2^{30} = 2^{30} + 2^{29} + 2^{28} + 2^{19}$. This number itself, however, is not represented by this code.

The interval of number represented by this code is $[1879048192, 1879572480)$.

A 19 Bit Binary Number System Example

A calculator uses 19 bit binary numbers with one bit for the sign of the number, followed by 11 bits for mantissa m, followed by 7 bits for the characteristic f.

(6) What interval of numbers are represented by this calculator with the code $0|11000000000|1011101$?

The smallest number represented by this code is $1.110000000000\bar{0} \times 2^{30} = 2^{30} + 2^{29} + 2^{28}$.

The upper end point of numbers represented by this code is $1.110000000001\bar{1} \times 2^{30} = 1.11000000001 \times 2^{30} = 2^{30} + 2^{29} + 2^{28} + 2^{19}$. This number itself, however, is not represented by this code.

The interval of number represented by this code is $[1879048192, 1879572480)$.

In particular, an interval of size $2^{19} = 524288$ of numbers are all represented by this same code.

A 19 Bit Binary Number System Example

A calculator uses 19 bit binary numbers with one bit for the sign of the number, followed by 11 bits for mantissa m, followed by 7 bits for the characteristic f.

(7) How many codes in this calculator's number system are assigned to the numbers between 8 and 32?

A 19 Bit Binary Number System Example

A calculator uses 19 bit binary numbers with one bit for the sign of the number, followed by 11 bits for mantissa m, followed by 7 bits for the characteristic f.

(7) How many codes in this calculator's number system are assigned to the numbers between 8 and 32?

Since mantissa m satisfies $0 \leq m < 1$, we have $1 \leq 1 + m < 2$ and $2^e \leq (1 + m) \times 2^e < 2^{e+1}$.

A 19 Bit Binary Number System Example

A calculator uses 19 bit binary numbers with one bit for the sign of the number, followed by 11 bits for mantissa m, followed by 7 bits for the characteristic f.

(7) How many codes in this calculator's number system are assigned to the numbers between 8 and 32?

Since mantissa m satisfies $0 \leq m < 1$, we have $1 \leq 1 + m < 2$ and $2^e \leq (1 + m) \times 2^e < 2^{e+1}$.

In order for the number to satisfy $2^3 = 8 \leq (1 + m) \times 2^e < 32 = 2^5$ we must have $e = 3$ or $e = 4$.

A 19 Bit Binary Number System Example

A calculator uses 19 bit binary numbers with one bit for the sign of the number, followed by 11 bits for mantissa m, followed by 7 bits for the characteristic f.

(7) How many codes in this calculator's number system are assigned to the numbers between 8 and 32?

Since mantissa m satisfies $0 \leq m < 1$, we have $1 \leq 1 + m < 2$ and $2^e \leq (1 + m) \times 2^e < 2^{e+1}$.

In order for the number to satisfy $2^3 = 8 \leq (1 + m) \times 2^e < 32 = 2^5$ we must have $e = 3$ or $e = 4$.

A 19 Bit Binary Number System Example

A calculator uses 19 bit binary numbers with one bit for the sign of the number, followed by 11 bits for mantissa m, followed by 7 bits for the characteristic f.

(7) How many codes in this calculator's number system are assigned to the numbers between 8 and 32?

Since mantissa m satisfies $0 \leq m < 1$, we have $1 \leq 1 + m < 2$ and $2^e \leq (1 + m) \times 2^e < 2^{e+1}$.

In order for the number to satisfy $2^3 = 8 \leq (1 + m) \times 2^e < 32 = 2^5$ we must have $e = 3$ or $e = 4$.

So only the two specific choices for the 7 characteristic bits corresponding to exponents 3 and 4, with any possible combination of the 11 mantissa bits, and the sign bit equal to 0 will correspond to the numbers between 8 and 32 (but not equal to 32) in this calculator's number system.

A 19 Bit Binary Number System Example

A calculator uses 19 bit binary numbers with one bit for the sign of the number, followed by 11 bits for mantissa m, followed by 7 bits for the characteristic f.

(7) How many codes in this calculator's number system are assigned to the numbers between 8 and 32?

So only the two specific choices for the 7 characteristic bits corresponding to exponents 3 and 4, with any possible combination of the 11 mantissa bits, and the sign bit equal to 0 will correspond to the numbers between 8 and 32 (but not equal to 32) in this calculator's number system.

Therefore, there are $2^{11} \times 2 + 1 = 4097$ numbers between 8 and 32 in this calculator's number system.

A 19 Bit Binary Number System Example

A calculator uses 19 bit binary numbers with one bit for the sign of the number, followed by 11 bits for mantissa m, followed by 7 bits for the characteristic f.

(7) How many codes in this calculator's number system are assigned to the numbers between 8 and 32?

So only the two specific choices for the 7 characteristic bits corresponding to exponents 3 and 4, with any possible combination of the 11 mantissa bits, and the sign bit equal to 0 will correspond to the numbers between 8 and 32 (but not equal to 32) in this calculator's number system.

Therefore, there are $2^{11} \times 2 + 1 = 4097$ numbers between 8 and 32 in this calculator's number system.

It is important to notice that the answer would bee the same if we were asked to count how many numbers in this calculator's number system are between $1024 = 2^{10}$ and $4096 = 2^{12}$.

Chopping and Rounding Choices for Termination

When we use decimal floating form of numbers and we have k digits available for the mantissa, we have two choices to terminate a possibly infinite sequence of digits in that number's mantissa.

Chopping and Rounding Choices for Termination

When we use decimal floating form of numbers and we have k digits available for the mantissa, we have two choices to terminate a possibly infinite sequence of digits in that number's mantissa.

A number $x = 0.d_1 d_2 \ldots d_k d_{k+1} \ldots \times 10^n$ can be represented within the k digit limitation of the computing device by one of the following two ways:

Chopping and Rounding Choices for Termination

When we use decimal floating form of numbers and we have k digits available for the mantissa, we have two choices to terminate a possibly infinite sequence of digits in that number's mantissa.

A number $x = 0.d_1 d_2 ... d_k d_{k+1} ... \times 10^n$ can be represented within the k digit limitation of the computing device by one of the following two ways:

Chopping: We terminate the number by chopping by omitting all the digits d_{k+1}, d_{k+2}, ...

Chopping and Rounding Choices for Termination

When we use decimal floating form of numbers and we have k digits available for the mantissa, we have two choices to terminate a possibly infinite sequence of digits in that number's mantissa.

A number $x = 0.d_1 d_2 ... d_k d_{k+1} ... \times 10^n$ can be represented within the k digit limitation of the computing device by one of the following two ways:

Chopping: We terminate the number by chopping by omitting all the digits d_{k+1}, d_{k+2}, ... In this case the floating form of the number x represented by $fl(x)$ is equal to $0.d_1 d_2 ... d_k \times 10^n$.

Chopping and Rounding Choices for Termination

When we use decimal floating form of numbers and we have k digits available for the mantissa, we have two choices to terminate a possibly infinite sequence of digits in that number's mantissa.

A number $x = 0.d_1d_2...d_kd_{k+1}... \times 10^n$ can be represented within the k digit limitation of the computing device by one of the following two ways:

Chopping: We terminate the number by chopping by omitting all the digits d_{k+1}, d_{k+2}, ... In this case the floating form of the number x represented by $fl(x)$ is equal to $0.d_1d_2...d_k \times 10^n$.

Rounding: We terminate the number by rounding by first adding $5 \times 10^{n-(k+1)}$ to the number and then chopping it.

Chopping and Rounding Choices for Termination

When we use decimal floating form of numbers and we have k digits available for the mantissa, we have two choices to terminate a possibly infinite sequence of digits in that number's mantissa.

A number $x = 0.d_1 d_2 ... d_k d_{k+1} ... \times 10^n$ can be represented within the k digit limitation of the computing device by one of the following two ways:

Chopping: We terminate the number by chopping by omitting all the digits d_{k+1}, d_{k+2}, ... In this case the floating form of the number x represented by $fl(x)$ is equal to $0.d_1 d_2 ... d_k \times 10^n$.

Rounding: We terminate the number by rounding by first adding $5 \times 10^{n-(k+1)}$ to the number and then chopping it. When $d_{k+1} < 5$ we simply chop the number and $fl(x) = 0.d_1 d_2 ... d_k \times 10^n$.

Chopping and Rounding Choices for Termination

When we use decimal floating form of numbers and we have k digits available for the mantissa, we have two choices to terminate a possibly infinite sequence of digits in that number's mantissa.

A number $x = 0.d_1 d_2 ... d_k d_{k+1} ... \times 10^n$ can be represented within the k digit limitation of the computing device by one of the following two ways:

Chopping: We terminate the number by chopping by omitting all the digits d_{k+1}, d_{k+2}, ... In this case the floating form of the number x represented by $fl(x)$ is equal to $0.d_1 d_2 ... d_k \times 10^n$.

Rounding: We terminate the number by rounding by first adding $5 \times 10^{n-(k+1)}$ to the number and then chopping it. When $d_{k+1} < 5$ we simply chop the number and $fl(x) = 0.d_1 d_2 ... d_k \times 10^n$. When $d_{k+1} \geq 5$ we add one to d_k and that may have an impact on all the digits or even the exponent.

Chopping and Rounding Choices for Termination

A number $x = 0.d_1d_2...d_kd_{k+1}... \times 10^n$ can be represented within the k digit limitation of the computing device by one of the following two ways:

Chopping: We terminate the number by chopping by omitting all the digits d_{k+1}, d_{k+2}, ... In this case the floating form of the number x represented by $fl(x)$ is equal to $0.d_1d_2...d_k \times 10^n$.

Rounding: We terminate the number by rounding by first adding $5 \times 10^{n-(k+1)}$ to the number and then chopping it. When $d_{k+1} < 5$ we simply chop the number and $fl(x) = 0.d_1d_2...d_k \times 10^n$. When $d_{k+1} \geq 5$ we add one to d_k and that may have an impact on all the digits or even the exponent.

For example $x = 9997.46 = 0.999746 \times 10^4$, when we use three digit chopping, $fl(x) =$

Chopping and Rounding Choices for Termination

A number $x = 0.d_1d_2...d_kd_{k+1}... \times 10^n$ can be represented within the k digit limitation of the computing device by one of the following two ways:

Chopping: We terminate the number by chopping by omitting all the digits d_{k+1}, d_{k+2}, ... In this case the floating form of the number x represented by $fl(x)$ is equal to $0.d_1d_2...d_k \times 10^n$.

Rounding: We terminate the number by rounding by first adding $5 \times 10^{n-(k+1)}$ to the number and then chopping it. When $d_{k+1} < 5$ we simply chop the number and $fl(x) = 0.d_1d_2...d_k \times 10^n$. When $d_{k+1} \geq 5$ we add one to d_k and that may have an impact on all the digits or even the exponent.

For example $x = 9997.46 = 0.999746 \times 10^4$, when we use three digit chopping, $fl(x) = 9990 = 0.999 \times 10^4$

Chopping and Rounding Choices for Termination

A number $x = 0.d_1d_2...d_kd_{k+1}... \times 10^n$ can be represented within the k digit limitation of the computing device by one of the following two ways:

Chopping: We terminate the number by chopping by omitting all the digits d_{k+1}, d_{k+2}, ... In this case the floating form of the number x represented by $fl(x)$ is equal to $0.d_1d_2...d_k \times 10^n$.

Rounding: We terminate the number by rounding by first adding $5 \times 10^{n-(k+1)}$ to the number and then chopping it. When $d_{k+1} < 5$ we simply chop the number and $fl(x) = 0.d_1d_2...d_k \times 10^n$. When $d_{k+1} \geq 5$ we add one to d_k and that may have an impact on all the digits or even the exponent.

For example $x = 9997.46 = 0.999746 \times 10^4$, when we use three digit chopping, $fl(x) = 9990 = 0.999 \times 10^4$ and when we use three digit rounding, $fl(x) =$

Chopping and Rounding Choices for Termination

A number $x = 0.d_1d_2...d_kd_{k+1}... \times 10^n$ can be represented within the k digit limitation of the computing device by one of the following two ways:

Chopping: We terminate the number by chopping by omitting all the digits d_{k+1}, d_{k+2}, ... In this case the floating form of the number x represented by $fl(x)$ is equal to $0.d_1d_2...d_k \times 10^n$.

Rounding: We terminate the number by rounding by first adding $5 \times 10^{n-(k+1)}$ to the number and then chopping it. When $d_{k+1} < 5$ we simply chop the number and $fl(x) = 0.d_1d_2...d_k \times 10^n$. When $d_{k+1} \geq 5$ we add one to d_k and that may have an impact on all the digits or even the exponent.

For example $x = 9997.46 = 0.999746 \times 10^4$, when we use three digit chopping, $fl(x) = 9990 = 0.999 \times 10^4$ and when we use three digit rounding, $fl(x) = 10000 = 0.100 \times 10^5$.

Error Measurement

If p^* is an approximation of an actual number p, the following terminology is used to refer to various measurements of error.

Error Measurement

If p^* is an approximation of an actual number p, the following terminology is used to refer to various measurements of error.

- The **actual error** is $p - p^*$

Error Measurement

If p^* is an approximation of an actual number p, the following terminology is used to refer to various measurements of error.

- The **actual error** is $p - p^*$
- The **absolute error** is $|p - p^*|$

Error Measurement

If p^* is an approximation of an actual number p, the following terminology is used to refer to various measurements of error.

- The **actual error** is $p - p^*$
- The **absolute error** is $|p - p^*|$
- The **relative error** is $\frac{|p-p^*|}{|p|}$

Error Measurement

If p^* is an approximation of an actual number p, the following terminology is used to refer to various measurements of error.

- The **actual error** is $p - p^*$
- The **absolute error** is $|p - p^*|$
- The **relative error** is $\frac{|p-p^*|}{|p|}$

When we use the relative error, we tolerate larger values of error for larger numbers.

Error Measurement

If p^* is an approximation of an actual number p, the following terminology is used to refer to various measurements of error.

- The **actual error** is $p - p^*$
- The **absolute error** is $|p - p^*|$
- The **relative error** is $\frac{|p-p^*|}{|p|}$

When we use the relative error, we tolerate larger values of error for larger numbers. When we use a fixed number of binary bits to estimate numbers, the chopped number $p^* = 1.d_1d_2...d_k \times 2^e$ is within a fixed relative error 2^{-k} of the actual number $p = 1.d_1d_2...d_kd_{k+1}... \times 2^e$, where k is the number of bits used for the mantissa.

Number of Significant Digits

We say p^* approximates p to k **significant digits** (or **significant figures**) if k is the largest non negative integer for which

$$\text{relative error} = \frac{|p - p^*|}{|p|} \leq 5 \times 10^{-k}$$

Number of Significant Digits

We say p^* approximates p to k **significant digits** (or **significant figures**) if k is the largest non negative integer for which

$$\text{relative error} = \frac{|p - p^*|}{|p|} \leq 5 \times 10^{-k}$$

If a number agrees with 0.1 to four significant digits, it means the number is within $[0.09995, 0.10005]$.

Number of Significant Digits

We say p^* approximates p to k **significant digits** (or **significant figures**) if k is the largest non negative integer for which

$$\text{relative error} = \frac{|p - p^*|}{|p|} \leq 5 \times 10^{-k}$$

If a number agrees with 0.1 to four significant digits, it means the number is within $[0.09995, 0.10005]$.

If a number agrees with 1000 to four significant digits, it means the number is within $[999.5, 1000.5]$.

Subtraction of Nearly Equal Numbers

Significant digits can be dropped when two nearly equal numbers are subtracted.

Subtraction of Nearly Equal Numbers

Significant digits can be dropped when two nearly equal numbers are subtracted.

As an example (**Example 5** p.21 in the book), suppose we use four digit rounding arithmetic.

Subtraction of Nearly Equal Numbers

Significant digits can be dropped when two nearly equal numbers are subtracted.

As an example (**Example 5** p.21 in the book), suppose we use four digit rounding arithmetic. With $p = 0.54617$ and $q = 0.54601$, $r = p - q = 0.00016$ we have $p^* = 0.5462$, $q^* = 0.5460$ and $r^* = 0.0002$.

Subtraction of Nearly Equal Numbers

Significant digits can be dropped when two nearly equal numbers are subtracted.

As an example (**Example 5** p.21 in the book), suppose we use four digit rounding arithmetic. With $p = 0.54617$ and $q = 0.54601$, $r = p - q = 0.00016$ we have $p^* = 0.5462$, $q^* = 0.5460$ and $r^* = 0.0002$.

$\frac{|p-p^*|}{|p|} = 0.000054...$ so p^* is accurate to four significant digits.

Subtraction of Nearly Equal Numbers

Significant digits can be dropped when two nearly equal numbers are subtracted.

As an example (**Example 5** p.21 in the book), suppose we use four digit rounding arithmetic. With $p = 0.54617$ and $q = 0.54601$, $r = p - q = 0.00016$ we have $p^* = 0.5462$, $q^* = 0.5460$ and $r^* = 0.0002$.

$\frac{|p-p^*|}{|p|} = 0.000054...$ so p^* is accurate to four significant digits.

$\frac{|q-q^*|}{|q|} = 0.000018...$ so q^* is accurate to five significant digits.

Subtraction of Nearly Equal Numbers

Significant digits can be dropped when two nearly equal numbers are subtracted.

As an example (**Example 5** p.21 in the book), suppose we use four digit rounding arithmetic. With $p = 0.54617$ and $q = 0.54601$, $r = p - q = 0.00016$ we have $p^* = 0.5462$, $q^* = 0.5460$ and $r^* = 0.0002$.

$\frac{|p-p^*|}{|p|} = 0.000054...$ so p^* is accurate to four significant digits.

$\frac{|q-q^*|}{|q|} = 0.000018...$ so q^* is accurate to five significant digits.

But $\frac{|r-r^*|}{|r|} = 0.25$ so r^* is accurate to only one significant digit.

The Way to Evaluate an Expression Impacts Relative Error

The two roots of a quadratic equation $ax^2 + bx + c = 0$ are expressed as $x_1 = \frac{-b+\sqrt{b^2-4ac}}{2a}$ and $x_2 = \frac{-b-\sqrt{b^2-4ac}}{2a}$.

The Way to Evaluate an Expression Impacts Relative Error

The two roots of a quadratic equation $ax^2 + bx + c = 0$ are expressed as $x_1 = \frac{-b+\sqrt{b^2-4ac}}{2a}$ and $x_2 = \frac{-b-\sqrt{b^2-4ac}}{2a}$.

These two roots can also be expressed by

$$x_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a}\left(\frac{-b - \sqrt{b^2 - 4ac}}{-b - \sqrt{b^2 - 4ac}}\right)$$

The Way to Evaluate an Expression Impacts Relative Error

The two roots of a quadratic equation $ax^2 + bx + c = 0$ are expressed as $x_1 = \frac{-b+\sqrt{b^2-4ac}}{2a}$ and $x_2 = \frac{-b-\sqrt{b^2-4ac}}{2a}$.

These two roots can also be expressed by

$$x_1 = \frac{-b+\sqrt{b^2-4ac}}{2a}\left(\frac{-b-\sqrt{b^2-4ac}}{-b-\sqrt{b^2-4ac}}\right) = -\frac{b^2-(b^2-4ac)}{-2a(b+\sqrt{b^2-4ac})}$$

The Way to Evaluate an Expression Impacts Relative Error

The two roots of a quadratic equation $ax^2 + bx + c = 0$ are expressed as $x_1 = \frac{-b+\sqrt{b^2-4ac}}{2a}$ and $x_2 = \frac{-b-\sqrt{b^2-4ac}}{2a}$.

These two roots can also be expressed by

$$x_1 = -\frac{b^2 - (b^2 - 4ac)}{-2a(b + \sqrt{b^2 - 4ac})} = \frac{4ac}{-2a(b + \sqrt{b^2 - 4ac})}$$

The Way to Evaluate an Expression Impacts Relative Error

The two roots of a quadratic equation $ax^2 + bx + c = 0$ are expressed as $x_1 = \frac{-b+\sqrt{b^2-4ac}}{2a}$ and $x_2 = \frac{-b-\sqrt{b^2-4ac}}{2a}$.

These two roots can also be expressed by

$$x_1 = -\frac{b^2 - (b^2 - 4ac)}{-2a(b + \sqrt{b^2 - 4ac})} = \frac{4ac}{-2a(b + \sqrt{b^2 - 4ac})} = -\frac{2c}{b + \sqrt{b^2 - 4ac}}$$

The Way to Evaluate an Expression Impacts Relative Error

The two roots of a quadratic equation $ax^2 + bx + c = 0$ are expressed as $x_1 = \frac{-b+\sqrt{b^2-4ac}}{2a}$ and $x_2 = \frac{-b-\sqrt{b^2-4ac}}{2a}$.

These two roots can also be expressed by

$$x_1 = -\frac{b^2 - (b^2 - 4ac)}{-2a(b + \sqrt{b^2 - 4ac})} = \frac{4ac}{-2a(b + \sqrt{b^2 - 4ac})} = -\frac{2c}{b + \sqrt{b^2 - 4ac}}$$

and $x_2 = -\frac{2c}{b-\sqrt{b^2-4ac}}$

The Way to Evaluate an Expression Impacts Relative Error

The two roots of a quadratic equation $ax^2 + bx + c = 0$ are expressed as $x_1 = \frac{-b+\sqrt{b^2-4ac}}{2a}$ and $x_2 = \frac{-b-\sqrt{b^2-4ac}}{2a}$.

These two roots can also be expressed by $x_1 = -\frac{2c}{b+\sqrt{b^2-4ac}}$ and $x_2 = -\frac{2c}{b-\sqrt{b^2-4ac}}$.

On pages 22 and 23 of the book, the example $x^2 + 62.10x + 1 = 0$ is used with four-digit rounding arithmetic.

The Way to Evaluate an Expression Impacts Relative Error

The two roots of a quadratic equation $ax^2 + bx + c = 0$ are expressed as $x_1 = \frac{-b+\sqrt{b^2-4ac}}{2a}$ and $x_2 = \frac{-b-\sqrt{b^2-4ac}}{2a}$.

These two roots can also be expressed by $x_1 = -\frac{2c}{b+\sqrt{b^2-4ac}}$ and $x_2 = -\frac{2c}{b-\sqrt{b^2-4ac}}$.

On pages 22 and 23 of the book, the example $x^2 + 62.10x + 1 = 0$ is used with four-digit rounding arithmetic.

Find the relative error when we calculate each root x_1 or x_2 using the two different formulas above.

Counting the Number of Operations

In performing a calculation we also should be aware of the total number of operations involved.

Counting the Number of Operations

In performing a calculation we also should be aware of the total number of operations involved. Each addition and multiplication can cause additional error and contribute to the time needed to complete the computation.

Counting the Number of Operations

In performing a calculation we also should be aware of the total number of operations involved. Each addition and multiplication can cause additional error and contribute to the time needed to complete the computation. Multiplication is normally considered to be costlier addition.

Counting the Number of Operations

In performing a calculation we also should be aware of the total number of operations involved. Each addition and multiplication can cause additional error and contribute to the time needed to complete the computation. Multiplication is normally considered to be costlier addition.

As an example to evaluate a polynomial $P(x) = a_n x^n + a_{n-1} x^{n-1} + \ldots a_1 x + a_0$ at a number x, we may consider performing the computation normally by first finding all powers of x involved, then multiplying each by the respective coefficient, and finally adding all these numbers up.

Counting the Number of Operations

In performing a calculation we also should be aware of the total number of operations involved. Each addition and multiplication can cause additional error and contribute to the time needed to complete the computation. Multiplication is normally considered to be costlier addition.

As an example to evaluate a polynomial $P(x) = a_n x^n + a_{n-1}x^{n-1} + ...a_1 x + a_0$ at a number x, we may consider performing the computation normally by first finding all powers of x involved, then multiplying each by the respective coefficient, and finally adding all these numbers up.

Evaluating $P(x)$ this way will require $n-1$ multiplications for powers of x, n multiplications by the coefficients a_1,.., a_n (total of $2n-1$ multiplications), and n additions.

Counting the Number of Operations

As an example to evaluate a polynomial $P(x) = a_n x^n + a_{n-1} x^{n-1} + ... a_1 x + a_0$ at a number x, we may consider performing the computation normally by first finding all powers of x involved, then multiplying each by the respective coefficient, and finally adding all these numbers up.

Evaluating $P(x)$ this way will require $n-1$ multiplications for powers of x, n multiplications by the coefficients a_1,.., a_n (total of $2n-1$ multiplications), and n additions.

Alternatively, we can evaluate $P(x)$ in a **nested** manner.

Counting the Number of Operations

As an example to evaluate a polynomial $P(x) = a_n x^n + a_{n-1}x^{n-1} + ...a_1 x + a_0$ at a number x, we may consider performing the computation normally by first finding all powers of x involved, then multiplying each by the respective coefficient, and finally adding all these numbers up.

Evaluating $P(x)$ this way will require $n-1$ multiplications for powers of x, n multiplications by the coefficients a_1,.., a_n (total of $2n-1$ multiplications), and n additions.

Alternatively, we can evaluate $P(x)$ in a **nested** manner. This evaluation will be based on re writing $P(x)$ as

$$P(x) = a_0 + x(a_1 + x(a_2 + x(...x(a_{n-1} + x(a_n))...)))$$

Counting the Number of Operations

As an example to evaluate a polynomial $P(x) = a_n x^n + a_{n-1}x^{n-1} + ...a_1 x + a_0$ at a number x, we may consider performing the computation normally by first finding all powers of x involved, then multiplying each by the respective coefficient, and finally adding all these numbers up.

Evaluating $P(x)$ this way will require $n-1$ multiplications for powers of x, n multiplications by the coefficients a_1,.., a_n (total of $2n-1$ multiplications), and n additions.

Alternatively, we can evaluate $P(x)$ in a **nested** manner. This evaluation will be based on re writing $P(x)$ as

$$P(x) = a_0 + x(a_1 + x(a_2 + x(...x(a_{n-1} + x(a_n))...)))$$

For example $2x^3 - 5x^2 + 6x - 3 =$

Counting the Number of Operations

As an example to evaluate a polynomial $P(x) = a_n x^n + a_{n-1}x^{n-1} + ...a_1 x + a_0$ at a number x, we may consider performing the computation normally by first finding all powers of x involved, then multiplying each by the respective coefficient, and finally adding all these numbers up.

Evaluating $P(x)$ this way will require $n-1$ multiplications for powers of x, n multiplications by the coefficients a_1,.., a_n (total of $2n-1$ multiplications), and n additions.

Alternatively, we can evaluate $P(x)$ in a **nested** manner. This evaluation will be based on re writing $P(x)$ as

$$P(x) = a_0 + x(a_1 + x(a_2 + x(...x(a_{n-1} + x(a_n))...)))$$

For example $2x^3 - 5x^2 + 6x - 3 = -3 + x(6 + x(-5 + 2x))$

Counting the Number of Operations

As an example to evaluate a polynomial $P(x) = a_n x^n + a_{n-1}x^{n-1} + ...a_1 x + a_0$ at a number x, we may consider performing the computation normally by first finding all powers of x involved, then multiplying each by the respective coefficient, and finally adding all these numbers up.

Evaluating $P(x)$ this way will require $n-1$ multiplications for powers of x, n multiplications by the coefficients a_1,.., a_n (total of $2n-1$ multiplications), and n additions.

Alternatively, we can evaluate $P(x)$ in a **nested** manner. This evaluation will be based on re writing $P(x)$ as

$$P(x) = a_0 + x(a_1 + x(a_2 + x(...x(a_{n-1} + x(a_n))...)))$$

Nested evaluation of $P(x)$ will require n multiplications and n additions instead.