

**CS537**

**Introduction to Numerical  
Methods**

**Lecture 1**

**Number Representations and Errors**

Professor Jun Zhang

Department of Computer Science  
University of Kentucky  
Lexington, KY 40206-0046

August 28, 2008

## Number in Different Bases

Humans use number base **10** for reasons of the number of fingers

Computers use number base **2** for reasons of on-off switch

Number based on **8** can be used to facilitate conversion between the base **2** and the base **10**

**Base 10 Expansion:**

$$\begin{aligned}76321 &= 1 + 20 + 300 + 6000 + 70000 \\ &= 1 \times 10^0 + 2 \times 10^1 + 3 \times 10^2 \\ &\quad + 6 \times 10^3 + 7 \times 10^4\end{aligned}$$

The general formula for a base **10** integer is

$$\begin{aligned}a_n a_{n-1} \dots a_2 a_1 a_0 &= a_0 \times 10^0 + a_1 \times 10^1 \\ &+ a_2 \times 10^2 + \dots + a_{n-1} \times 10^{n-1} + a_n \times 10^n\end{aligned}$$

## Nonintegers and Fractions

An example of a fractional number

$$\begin{aligned}0.6321 &= \frac{6}{10} + \frac{3}{100} + \frac{2}{1000} + \frac{1}{10000} \\ &= 6 \times 10^{-1} + 3 \times 10^{-2} + 2 \times 10^{-3} \\ &\quad + 1 \times 10^{-4}\end{aligned}$$

The general formula for a decimal fraction

$$0.b_1b_2b_3 \dots = b_1 \times 10^{-1} + b_2 \times 10^{-2} + b_3 \times 10^{-3} + \dots$$

Decimal fractional numbers can be repeating or nonrepeating (rational or irrational numbers)

$$\begin{aligned}(a_n a_{n-1} \dots a_1 a_0 . b_1 b_2 b_3 \dots)_{10} = \\ \sum_{k=0}^n a_k \times 10^k + \sum_{k=1}^{\infty} b_k \times 10^{-k}\end{aligned}$$

## Base $\beta$ Numbers

Convert an integer number from base **8** system to base **10** system

$$\begin{aligned}(7632)_8 &= 2 \times 8^0 + 3 \times 8^1 + 6 \times 8^2 + 7 \times 8^3 \\ &= 2 + 8(3 + 8(6 + (8(7)))) \\ &= 3994\end{aligned}$$

Convert a fractional number between **0** and **1**

$$\begin{aligned}(0.763)_8 &= 7 \times 8^{-1} + 6 \times 8^{-2} + 3 \times 8^{-3} \\ &= 8^{-3} (3 + 8(6 + 8(7))) \\ &= \frac{499}{512} \\ &= 0.9746\end{aligned}$$

General formula

$$\begin{aligned}(a_n a_{n-1} \dots a_1 a_0 . b_1 b_2 b_3 \dots)_\beta &= \\ \sum_{k=0}^n a_k \times \beta^k + \sum_{k=1}^{\infty} b_k \times \beta^{-k}\end{aligned}$$

## Conversion between Systems

From system  $\alpha$  to  $\beta$  with  $\alpha < \beta$

- express  $(N)_\alpha$  in nested form using powers of  $\alpha$
- replace each digit by corresponding base  $\beta$  number
- carry out indicated arithmetic in base  $\beta$

Two examples were given in the previous page

Note that one needs to convert the integer and the fractional parts separately

The method is in theory can be used to convert a number between any two bases. But the third step is not easy for humans if the base is not **10**

## Division Algorithm

Use remainder-quotient-split method to convert an integer, best if  $\alpha > \beta$

$$\begin{aligned}(N)_{\alpha} &= (c_m c_{m-1} \dots c_1 c_0)_{\beta} \\ &= c_0 + \beta(c_1 + \beta(c_2 + \dots + \beta(c_m) \dots))\end{aligned}$$

1. Divide the representation by  $\beta$ ;
2. Take the remainder as the next digit in the base  $\beta$  representation;
3. Repeat the procedure on the quotient.

This procedure is easier to carry out by hand

Note that the first digit obtained is the one closest to the radix point

## Multiplication Algorithm

Use integer-fraction-split process for converting a fractional part

$$x = \sum_{k=1}^{\infty} c_k \times \beta^{-k} = (0.c_1c_2c_3\dots)_{\beta}$$

- 1.) multiply the (fractional) number by  $\beta$ ;
- 2.) take the integer part as the first (next) digit;
- 3.) repeat the process on the remaining *fractional* part.

Again, the first digit obtained is the one closest to the radix point

Terminating fractional number may become non-terminating in different base systems, and vice versa

## **Use Intermediate Base**

We can convert a base **10** number to a base **8** (octal) number, then to base **2** (binary) number, and vice versa

Binary-octal table

000	001	010	011	100	101	110	111
0	1	2	3	4	5	6	7

Groups of three binary digits can be translated to octal using the binary-octal table

$$(110\ 101\ 001.011\ 100\ 111)_2 \\ = (651.347)_8 = (?)_{10}$$

$$(57.321)_{10} = (?)_8 = (?)_2$$

## Base 16 Numbers

Certain computers with word length being multiples of 4 use hexadecimal system, need A, B, C, D, E, and F to represent 10, 11, 12, 13, 14, and 15.

Binary-hexadecimal table

0000	0001	0010	0011	0100	0101
0	1	2	3	4	5
0110	0111	1000	1001	1010	1011
6	7	8	9	A	B
1100	1101	1110	1111		
C	D	E	F		

$$\begin{aligned} & (010\ 111\ 100.110\ 100\ 101\ 111)_2 \\ &= (1011\ 1100.1101\ 0010\ 1111)_2 \\ &= (\mathbf{BC.D2F})_{16} \end{aligned}$$

## Normalized Scientific Notation

Write a number in the form of

$$728.359 = 0.728359 \times 10^3$$

A normalized floating-point number has the form

$$x = \pm 0.d_1 d_2 d_3 \dots \times 10^n$$

where  $d_1 \neq 0$ ,  $n$  is an integer

In a simple notation

$$x = \pm r \times 10^n \quad \left( \frac{1}{10} \leq r < 1 \right)$$

$r$  is called normalized mantissa and  $n$  is the exponent. Also for binary representation

$$x = \pm q \times 2^m \quad \left( \frac{1}{2} \leq q < 1 \right)$$

## Machine Numbers

The real numbers that can be represented exactly in a computer are called the *machine numbers* for this computer

Most real numbers are not machine numbers

If a computer has word length of the form  $0.d_1d_2d_3d_4$ , then **0.1011** is a machine number, but **0.10101** is not

Machine numbers are machine dependent. The use of normalized floating-point numbers creates a phenomenon of *hole at zero*, a bunch of numbers close to **0** are not representable

This is mainly caused by the under-flow problem, i.e., small numbers close to zero will be treated as zero in a computer

## A 32-bit Machine

Single precision floating-point numbers with 32-bit word length

How to store a floating-point number

$$\pm q \times 2^m$$

with 32 bits?

Sign of  $q$  needs 1 bit

Integer  $|m|$  needs 8 bits

Number  $q$  needs 23 bits

Single precision IEEE standard floating-point

$$(-1)^s \times 2^{c-127} \times (1.f)_2$$

## **32-bit Representation**

The exponent number  $c$  is actually stored, so  $m = c - 127$ , this is an *excess-127 code* (make sure only “positive numbers” are stored)

With normalized representation, the first bit is always  $1$  and needs not be stored. The mantissa actually contains  $24$  binary digits with a *hidden bit*

With a mantissa of  $23$  bit, a machine can have about six significant decimal digits of accuracy, since  $2^{-23} \approx 1.2 \times 10^{-7}$

The largest number  $\epsilon$  such that  $1 + \epsilon = 1$  is called the *machine epsilon* or *unit roundoff error*

For single precision,  $\epsilon = 1.2 \times 10^{-7}$

For double precision,  $\epsilon = 2.2 \times 10^{-16}$

## Representation Procedure

How to represent a real number  $x$ ?

- 1.) if  $x$  is zero, stored by a full word of zero bits, (a possible sign bit)
- 2.) for nonzero  $x$ , first consider sign bit and then consider  $|x|$
- 3.) convert both integer and fractional parts of  $|x|$  from decimal to octal, then to binary
- 4.) one-plus normalize  $(|x|)_2$  by shifting the binary point
- 5.) find the 24 bit 1-plus normalized mantissa
- 6.) find the exponent of 2 by setting it equal to  $c-127$  and determine  $c$
- 7.) denote the 32-bit representation as 8 hexadecimal digits

## Summary and Examples

A 32-bit single-precision pattern

$$b_1 b_2 b_3 \dots b_9 b_{10} b_{11} \dots b_{32}$$

can be interpreted as the real number

$$(-1)^{b_1} \times 2^{(b_2 b_3 \dots b_9)_2} \times 2^{-127} \times (1.b_{10} b_{11} \dots b_{32})_2$$

### Examples

Find the 32-bit representation of **-52.234375**.

Integer part

$$(52.)_{10} = (64.)_8 = (110\ 100.)_2$$

Fractional part

$$(.234375)_{10} = (.17)_8 = (.001\ 111)_2$$

## Examples – cont.

$$\begin{aligned}(52.234375)_{10} &= (110\ 100.001111)_2 \\ &= (1.101\ 000\ 011110)_2 \times 2^5\end{aligned}$$

The exponent is  $(5)_{10}$ , we need to write it as  $c - 127 = 5$  so  $c = 132$

The stored exponent is

$$(132)_{10} = (204)_8 = (10\ 000\ 100)_2$$

The representation of  $-52.234375$  is

$$\begin{aligned}&(1\ 10\ 000\ 100\ 101\ 000\ 011\ 110\ 000\ 000\ 000\ 00)_2 \\ &= (1100\ 0010\ 0101\ 0000\ 1111\ 0000\ 0000\ 0000)_2 \\ &= (C250F000)_{16}\end{aligned}$$

What number has the representation

$$(45DE4000)_{16}?$$

## Errors in Representing Numbers

Non-machine numbers are represented by a nearest machine number in computer

Additional digits will be truncated

**Correctly rounding** and **roundoff error**

$$\frac{1}{2} \times 2^{-23}$$

For a 32-bit single-precision machine with 23 bits for mantissa, the relative error in correct rounding is

The unit round error is  $\varepsilon = 2^{-23}$

For a computer with a 4 binary digit word length, the number 0.11101011 will be stored as 0.1110 for rounding off, and 0.1111 for correctly rounding

## **Notation $\mathbf{fl}(x)$**

Most computers use double-length arithmetic operations. Numbers are extended to double length, arithmetic operations are performed, and the result is rounded to a single length number

Use  $\mathbf{fl}(x)$  to denote the floating-point machine number corresponding to a real number  $x$

For the 32-bit machine, we have

$$\mathbf{fl}(x) = x(1 + \delta), \quad \left( |\delta| \leq \frac{1}{2} \epsilon \right)$$

It is easy to see that if  $\epsilon < 2^{-23}$ , then

$$\mathbf{fl}(1 + \epsilon) = 1$$

## **Inverse-Error Analysis**

More generally known as **backward error analysis**

Denote  $\odot$  as one of the basic arithmetic operations, then

$$\mathbf{fl}(x \odot y) = (x \odot y)(1 + \delta), \quad (|\delta| \leq 2^{-24})$$

Two interpretations

$$\mathbf{fl}(x) = (x + y)(1 + \delta)$$

$$\mathbf{fl}(x) = x(1 + \delta) + y(1 + \delta)$$

Direct-error analysis and reverse-error analysis

Forward-error analysis and backward-error analysis

## Loss of Significance

Subtraction of two nearly equal numbers may result in loss of significant digits on a finite precision machine

Cure: reprogram or use higher precision arithmetic

Avoid using unnecessary higher precision arithmetic

### Loss of Precision Theorem

Let  $x$  and  $y$  be normalized floating-point machine numbers with  $x > y > 0$ . If  $2^{-p} \leq 1 - y/x \leq 2^{-q}$  for some positive integers  $p$  and  $q$ , then at most  $p$  and at least  $q$  significant digits are lost in the subtraction  $x - y$

## An Example from Book

$$x = 37.593621 \text{ and } y = 37.584216$$

$$1 - \frac{y}{x} = 0.0002501754,$$

which is between  $2^{-12} = 0.000244$  and  $2^{-11} = 0.000488$ . At least **11** and at most 12 binary digits are lost when computing  $x - y$

Exactly how many digits lost depends on a computer

$$x = 0.37593612 \times 10^2 \text{ and } y = 0.37584216 \times 10^2$$

Suppose a machine has 5 decimal digits of accuracy

$$\text{We have } \tilde{x} = 0.37594 \times 10^2 \text{ and } \tilde{y} = 0.37584 \times 10^2,$$

The machine computes

$$\tilde{x} - \tilde{y} = 0.00010 = 0.10000 \times 10^{-3}$$

## An Example Cont.

Exact computation

$$x - y = 0.00009396 = 0.9396 \times 10^{-4}$$

The relative error is

$$\frac{|(x - y) - (\tilde{x} - \tilde{y})|}{|x - y|} = \frac{0.604 \times 10^{-5}}{0.9396 \times 10^{-4}} = 0.06428$$

This relative error is considered large since the machine has 5 decimal digits of accuracy

Note that the double length computing operations are carried out after the numbers are stored

If we use a machine with at least 8 decimal digits of accuracy, we can have the exact value as  $0.93960000 \times 10^{-4}$

## Avoiding Loss of Significance

Analyze possible loss of significance, reschedule computation to avoid subtraction of two nearly equal numbers, modify algorithm

**Example.** Evaluating

$$f(x) = \sqrt{x^2 + 1} - 1 \quad \text{at} \quad x \approx 0$$

Using 5-decimal-digit arithmetic for  $x = 10^{-3}$ , we have  $f(x) = 0$

Rationalizing the numerator, we have

$$f(x) = \frac{x^2}{\sqrt{x^2 + 1} + 1}$$

computing  $\sqrt{(10^{-3})^2 + 1} + 1 = 0.2 \times 10^1$ , and  $f(x) = 0.5 \times 10^{-6}$ .

## More Examples

Evaluating  $f(x) = x - \sin x$  at  $x \approx 0$

Using Taylor series for  $\sin x$

$$\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \dots$$

Then

$$f(x) = \frac{x^3}{3!} - \frac{x^5}{5!} + \frac{x^7}{7!} - \dots$$

Compute  $x = 0.1$  with four-decimal-digit arithmetic,  $\sin(0.1) = 0.9983 \times 10^{-1}$ . So  $x - \sin x = 0.17 \times 10^{-3}$ . But  $x^3/3! = 0.1667 \times 10^{-3}$ . This strategy is not good for large  $x$ . For  $x = \pi$ ,  $x - \sin \pi = \pi$ , but  $x^3/3! = 5.1677$ .

Range reduction for periodic functions

$$\sin(12532.14) \approx \sin(3.47 + 1994 \times 2\pi) = \sin(3.47)$$