WIKIPEDIA

IEEE 754

維基百科，自由的百科全書

IEEE二進位浮點數算術標準（**IEEE 754**）是20世紀80年代以來最廣泛使用的浮點數運算標準，為許多CPU與浮點運算器所採用。這個標準定義了表示浮點數的格式（包括負零-0）與反常值（denormal number）），一些特殊數值（無窮（Inf）與非數值（NaN）），以及這些數值的「浮點數運算子」；它也指明了四種數值修約規則和五種例外狀況（包括例外發生的時機與處理方式）。

IEEE 754規定了四種表示浮點數值的方式：單精確度（32位元）、雙精確度（64位元）、延伸單精確度（43位元以上，很少使用）與延伸雙精確度（79位元以上，通常以80位元實做）。只有32位元模式有強制要求，其他都是選擇性的。大部分程式語言都有提供IEEE浮點數格式與算術，但有些將其列為非必需的。例如，IEEE 754問世之前就有的C語言，現在有包括IEEE算術，但不算作強制要求（C語言的`float`通常是指IEEE單精確度，而`double`是指雙精確度）。

該標準的全稱為**IEEE二進位浮點數算術標準（ANSI/IEEE Std 754-1985）**，又稱**IEC 60559:1989，微處理器系統的二進位浮點數算術**（本來的編號是IEC 559:1989）[1]。後來還有「與基數無關的浮點數」的「IEEE 854-1987標準」，有規定基數為2跟10的狀況。現在最新標準是「ISO/IEC/IEEE FDIS 60559:2010」。

在六、七十年代，各家電腦公司的各個型號的電腦，有著千差萬別的浮點數表示，卻沒有一個業界通用的標準。這給資料交換、電腦協同工作造成了極大不便。IEEE的浮點數專業小組於七十年代末期開始醞釀浮點數的標準。在1980年，英特爾公司就推出了單片的8087浮點數協處理器，其浮點數表示法及定義的運算具有足夠的合理性、先進性，被IEEE採用作為浮點數的標準，於1985年發布。而在此前，這一標準的內容已在八十年代初期被各電腦公司廣泛採用，成了事實上的業界工業標準。加州大學伯克利分校的數值計算與電腦科學教授威廉·卡韓被譽為「浮點數之父」。

目錄

浮點數剖析

一個浮點數 (Value) 的表示其實可以這樣表示：

$$Value = sign \times exponent \times fraction$$

也就是浮點數的**實際值**，等於符號位（sign bit）乘以**指數偏移值**(exponent bias)再乘以**分數值**(fraction)。

以下內文是IEEE 754對浮點數格式的描述。

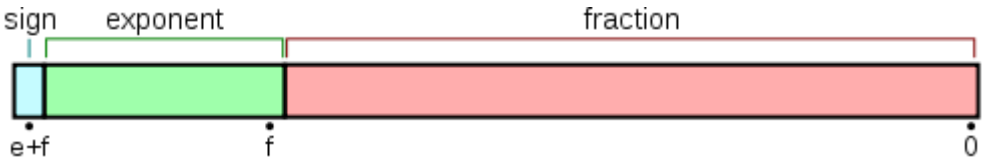
本文表示位元的約定

把W個位元（bit）的資料，從記憶體位址低端到高端，以0到W−1編碼。通常將記憶體位址低端的位元寫在最右邊，稱作最低有效位（Least Significant Bit,LSB），代表最小的位元，改變時對整體數值影響最小的位元。聲明這一點的必要性在於X86體系架構是小端序的資料儲存。

對於十進位整數N，必要時表示為N_{10}以與二進位的數的表示N_2相區分。

對於一個數，其二進位科學計數法表示下的指數的值，下文稱之為**指數的實際值**；而根據IEEE 754標準對指數部分的編碼的值，稱之為浮點數表示法**指數域的編碼值**。

整體呈現



IEEE 754浮點數的三個域

二進位浮點數是以符號數值表示法的格式儲存——最高有效位被指定為符號位（sign bit）；「指數部份」，即次高有效的e個位元，儲存指數部分；最後剩下的f個低有效位的位元，儲存「有效數」（significand）的小數部份（在非規約形式下整數部份默認為0，其他情況下一律默認為1）。

指數偏移值

指數偏移值(exponent bias），是指浮點數表示法中的指數域的編碼值為指數的實際值加上某個固定的值，IEEE 754標準規定該固定值為$2^{e-1}-1$[2]，其中的e為儲存指數的位元的長度。

以單精度浮點數為例，它的指數域是8個位元，固定偏移值是$2^{8-1}-1=128-1=127$。此為有號數的表示方式，單精度浮點數的指數部分實際取值是從-128到127。例如指數實際值為17_{10}，在單精度浮點數中的指數域編碼值為144_{10}，即$144_{10}=17_{10}+127_{10}$。

採用指數的實際值加上固定的偏移值的辦法表示浮點數的指數，好處是可以用長度為e個位元的無符號整數來表示所有的指數取值，這使得兩個浮點數的指數大小的比較更為容易，實際上可以按照字典序比較兩個浮點表示的大小。

這種**移碼**表示的指數部分，中文稱作**階碼**。

正規形式的浮點數

如果浮點數中指數部分的編碼值在$0<exponent<2^e-2$之間，且在科學表示法的表示方式下，分數 (fraction) 部分**最高有效位**（即整數位）是**1**，那麼這個浮點數將被稱為**正規形式的浮點數**。「規約」是指用唯一確定的浮點形式去表示一個值。

由於這種表示下的尾數有一位隱含的二進位有效數字，為了與二進位科學計數法的尾數（mantissa）相區別，IEEE754稱之為**有效數**（significant）。

舉例來說，雙精度 (64-bit) 的規約形式浮點數在指數偏移值的值域為**00000000001** (11-bit) 到**11111111110**，在分數部分則是**000.....000**到**111.....111** (52-bit)

非正規形式的浮點數

如果浮點數的指數部分的編碼值是0，分數部分非零，那麼這個浮點數將被稱為**非正規形式的浮點數**。一般是某個數字**相當**接近零時才會使用非規約型式來表示。 IEEE 754標準規定：**非正規形式的浮點數的指數偏移值比正規形式的浮點數的指數偏移值小1**。例如，最小的正規形式的單精度浮點數的指數部分編碼值為1，指數的實際值為-126；而非規約的單精度浮點數的指數域編碼值為0，對應的指數實際值也是-126而不是-127。實際上非正規形式的浮點數仍然是有效可以使用的，只是它們的絕對值已經小於所有的規約浮點數的絕對值；即所有的非規約浮點數比規約浮點數更接近0。規約浮點數的尾數大於等於1且小於2，而非規約浮點數的尾數小於1且大於0。

除了規約浮點數，IEEE754-1985標準採用非規約浮點數，用來解決填補絕對值意義下最小規格數與零的距離。（舉例說，正數下，最大的非規格數等於最小的規格數。而一個浮點數編碼中，如果exponent=0，且尾數部分不為零，那麼就按照非規約浮點數來解析）非規約浮點數源於70年代末IEEE浮點數標準化專業技術委員會醞釀浮點數二進位標準時，Intel公司對**漸進式下限溢位出**（gradual underflow）的力薦。當時十分流行的DEC VAX機的浮點數表示採用了**突然式下限溢位出**（abrupt underflow）。如果沒有漸進式下限溢位出，那麼0與絕對值最小的浮點數之間的距離（gap）將大於相鄰的小浮點數之間的距離。例如單精度浮點數的絕對值最小的規約浮點數是1.0×2^{-126},它與絕對值次小的規約浮點數之間的距離為$2^{-126}\times2^{-23}=2^{-149}$。如果不採用漸進式下限溢位出，那麼絕對值最小的規約浮點數與0的距離是相鄰的小浮點數之間距離的2^{23}倍！可以說是非常突然的下限溢位出到0。這種情況的一種糟糕後果是：兩個不等的小浮點數X與Y相減，結果將是0.訓練有素的數值分析人員可能會適應這種限制情況，但對於普通的程式設計師就很容易

陷入錯誤了。採用了漸進式下限溢位出後將不會出現這種情況。例如對於單精度浮點數，指數部分實際最小值是（-126），對應的尾數部分從$1.1111\ldots11$, $1.1111\ldots10$一直到$0.0000\ldots10$, $0.0000\ldots01$，$0.0000\ldots00$相鄰兩小浮點數之間的距離（gap）都是$2^{-126}\times 2^{-23}=2^{-149}$；而與0最近的浮點數（即最小的非正規數）也是$2^{-126}\times 2^{-23}=2^{-149}$。

特殊值

這里有三個特殊值需要指出：

1. 如果*指數*是0並且尾數的*小數部分*是0，這個數±0（和符號位相關）
2. 如果*指數* = 2^e-1並且尾數的*小數部分*是0，這個數是±∞（同樣和符號位相關）
3. 如果*指數* = 2^e-1並且尾數的*小數部分*非0，這個數表示為不是一個數（NaN）。

以上規則，總結如下：

形式	指數	小數部分
零	0	0
非正規形式	0	非0
正規形式	1到2^e-2	任意
無窮	2^e-1	0
NaN	2^e-1	非零

32位元單精度

單精度二進制小數，使用32個位元存儲。

1	8	23位長
S	Exp	Fraction
31	30至23 偏正值（實際的指數大小+127）	22至0位編號（從右邊開始為0）

S為符號位，Exp為指數位，Fraction為有效數位。 指數部分即使用所謂的*偏正值*形式表示，偏正值為實際的指數大小與一個固定值（32位元的情況是127）的和。採用這種方式表示的目的是簡化比較。因為，指數的值可能為正也可能為負，如果採用補碼表示的話，全體符號位S和Exp自身的符號位將導致不能簡單的進行大小比較。正因為如此，指數部分通常採用一個無符號的正數值存儲。單精度的指數部分是−126～+127加上偏移值127，指數值的大小從1～254（0和255是特殊值）。浮點小數計算時，指數值減去偏正值將是實際的指數大小。

單精度浮點數各種極值情況：

類別	正負號	實際指數	有偏移指數	指數域	尾數域	數值
零	0	-127	0	0000 0000	000 0000 0000 0000 0000 0000	0.0
負零	1	-127	0	0000 0000	000 0000 0000 0000 0000 0000	−0.0
1	0	0	127	0111 1111	000 0000 0000 0000 0000 0000	1.0
-1	1	0	127	0111 1111	000 0000 0000 0000 0000 0000	−1.0
最小的非正規數	*	-126	0	0000 0000	000 0000 0000 0000 0000 0001	$\pm 2^{-23} \times 2^{-126} = \pm 2^{-149} \approx \pm 1.4 \times 10^{-45}$
中間大小的非正規數	*	-126	0	0000 0000	100 0000 0000 0000 0000 0000	$\pm 2^{-1} \times 2^{-126} = \pm 2^{-127} \approx \pm 5.88 \times 10^{-39}$
最大的非正規數	*	-126	0	0000 0000	111 1111 1111 1111 1111 1111	$\pm(1-2^{-23}) \times 2^{-126} \approx \pm 1.18 \times 10^{-38}$
最小的正規數	*	-126	1	0000 0001	000 0000 0000 0000 0000 0000	$\pm 2^{-126} \approx \pm 1.18 \times 10^{-38}$
最大的正規數	*	127	254	1111 1110	111 1111 1111 1111 1111 1111	$\pm(2-2^{-23}) \times 2^{127} \approx \pm 3.4 \times 10^{38}$
正無窮	0	128	255	1111 1111	000 0000 0000 0000 0000 0000	+∞
負無窮	1	128	255	1111 1111	000 0000 0000 0000 0000 0000	−∞
NaN	*	128	255	1111 1111	non zero	NaN
* 符號位可以為0或1 .						

64位元雙精度

雙精度二進制小數，使用64個位元存儲。

1	11	52位長
S	Exp	Fraction
63	62至52 偏正值（實際的指數大小+1023）	51至0位編號（從右邊開始為0）

S為符號位，Exp為指數位，Fraction為有效數位。指數部分即使用所謂的偏正值形式表示，偏正值為實際的指數大小與一個固定值（64位元的情況是1023）的和。採用這種方式表示的目的是簡化比較。因為，指數的值可能為正也可能為負，如果採用補碼表示的話，全體符號位S和Exp自身的符號位將導致不能簡單的進行大小比較。正因為如此，指數部分通常採用一個無符號的正數值存儲。雙精度的指數部分是−1022～+1023加上1023，指數值的大小從1～2046（0（2進位全為0）和2047（2進位全為1）是特殊值）。浮點小數計算時，指數值減去偏正值將是實際的指數大小。

浮點數的比較

浮點數基本上可以按照符號位、指數域、尾數域的順序作字典比較。顯然，所有正數大於負數；正負號相同時，指數的二進位表示法更大的其浮點數值更大。

浮點數的捨入

任何有效數上的運算結果，通常都存放在較長的暫存器中，當結果被放回浮點格式時，必須將多出來的位元丟棄。 有多種方法可以用來執行捨入作業，實際上IEEE標準列出4種不同的方法：

- **捨入到最接近**：修約到最接近，在一樣接近的情況下偶數優先（Ties To Even，這是預設的修約方式）：會將結果修約為最接近且可以表示的值，但是當存在兩個數一樣接近的時候，則取其中的偶數（在二進位中式以0結尾的）。
- **朝+∞方向捨入**：會將結果朝正無限大的方向捨入。
- **朝-∞方向捨入**：會將結果朝負無限大的方向捨入。
- **朝0方向捨入**：會將結果朝0的方向捨入。

浮點數的運算與函式

標準運算

下述函式必須提供:

- 加減乘除（Add、subtract、multiply、divide）。在加減運算中負零與零相等：$-0.0 = 0.0$
- 平方根（Square root）：$\sqrt{x} \geq 0 (x \geq 0)$，另規定$\sqrt{-0.0} = -0.0$
- 浮點餘數。返回值$x - (round(x/y) * y)$。
- 近似到最近的整數$round(x)$。如果恰好在兩個相鄰整數之間，則近似到偶數。
- 比較運算. -Inf <負的規約浮點數數<負的非規約浮點數< -0.0 = 0.0 <正的非規約浮點數<正的規約浮點數< Inf；
- 特殊比較： -Inf = -Inf, Inf = Inf, NaN與任何浮點數（包括自身）的比較結果都為假，即 (NaN ≠ x) = false.

建議的函式與謂詞

- `copysign(x, y)`: `copysign(x, y)`返回的值由x的不帶符號的部份和y的符號組成。因此`abs(x)`等於`copysign(x, 1.0)`。`copysign`可以對NaN正確操作，這是少有的幾個可以對NaN像普通算術一樣操作有效的函式之一。C99新增了`copysign`函式。
- −x:從涵義上指將x的符號反轉。當x是±0或者NaN時，其涵義可能不同於0-x.
- `scalb(y, N)`:計算$y \times 2^N$（N是整數），無需再計算2^N。C99中對應的函式名是`scalbn`.
- `logb(x)`:計算$x = 1.a \times 2^n$（x ≠ 0, a ∈[0, 1)）中的n. C99新增了`logb`和`ilogb`函式。
- `nextafter(x,y)`:沿y方向找最鄰近x的可表達浮點數。比如`nextafter(0, 1)`得到的是最小可表達的正數。C99新增了`nextafter`函式。
- `finite(x)`:判斷x是否有限，即−Inf < x < Inf. C99新增了`isfinite`函式。
- `isnan(x)`:判斷x是否是一個NaN，這等價於"x ≠ x". C99新增了`isnan`函式。
- `x <> y`:僅當x < y或者x > y時才為True，其涵義是NOT（x = y）。注意這不同於"x ≠ y"。
- `unordered(x, y)`:當x與y無法比較大小時為True，比如說x或者y是一個NaN. C99中對應的函式名是`isunordered`.
- `class(x)`:區分x的浮點數類屬：信號NaN、靜默NaN、-Inf、負的規約浮點數，負的非規約浮點數，-0.0,0.0，正的非規約浮點數，正的規約浮點數，Inf。

精度

在二進制，第一個有效數字必定是「1」，因此這個「1」並不會儲存。

討論一

單精和雙精浮點數的有效數字分別是有儲存的23和52個位，加上最左手邊沒有儲存的第1個位，即是24和53個位。

$$\log 2^{24} = 7.22$$

$$\log 2^{53} = 15.95$$

由以上的計算，單精和雙精浮點數可以保證7位和15位十進制有效數字。

討論二

C++語言標準定義的浮點數的**十進位精度**(decimal precision)：十進位數字的位數，可被(浮點數)表示而值不發生變化[3]。C語言標準定義的浮點數的十進位精度為：十進位數字的位數q，使得任何具有q位十進位數字的浮點數可近似表示為b進位的p位數字並且能近似回十進位表示而不改變這q位十進位數字[4]

但由於相對近似誤差不均勻，有的7位十進位浮點數不能保證近似轉化為32位元浮點再近似轉化回7位十進位浮點後保持值不變：例如8.589973e9將變成8.589974e9。這種近似誤差不會超過1位元的表示能力，因此(24-1)*std::log10(2)等於6.92，下取整為6，成為std::numeric_limits<float>::digits10以及FLT_DIG的值。std::numeric_limits<float>::max_digits10的值為9，含義是必須9位十進位數字才能區分float的所有值；也即float的最大表示區分度。

類似的，std::numeric_limits<double>::digits10或DBL_DIG是15， std::numeric_limits<double>::max_digits10是17

例子

以下的C++程式，概略地展示了單精和雙精浮點數的精度。

```
#include <iostream>

int main () {
    std::cout.precision(20);
    float a=123.45678901234567890;
    double b=123.45678901234567890;
    std::cout << a << std::endl;
    std::cout << b << std::endl;
    return 0;
}

// Xcode 5.1
// Output:
// 123.456787109375
// 123.45678901234568059
// Program ended with exit code: 0
```

相關條目

- 浮點數
- 單精度浮點數
- 雙精度浮點數

外部連結

- IEEE 754 references (https://web.archive.org/web/20070505021348/http://babbage.cs.qc.edu/courses/cs341/IEEE-754references.html)
- Let's Get To The (Floating) Point by Chris Hecker (http://www.d6.com/users/checker/pdfs/gdmfp.pdf)
- What Every Computer Scientist Should Know About Floating-Point Arithmetic by David Goldberg (http://docs.sun.com/source/806-3568/ncg_goldberg.html) - a good introduction and explanation.
- IEEE 854-1987 (http://www2.hursley.ibm.com/decimal/854mins.html) History and minutes
- Converter (https://web.archive.org/web/20070424205950/http://www.h-schmidt.net/FloatApplet/IEEE754.html)
- Another Converter (https://web.archive.org/web/20060901165225/http://babbage.cs.qc.edu/courses/cs341/IEEE-754.html)

- Converter as MS-Windows program (http://www.61131.com/download.htm)
- Comparing doubles in C++ (https://web.archive.org/web/20070928213715/http://metasharp.net/index.php?title=How_to_compare_double_or_float_in_Cpp)
- An Interview with the Old Man of Floating-Point (http://www.cs.berkeley.edu/~wkahan/ieee754status/754story.html)
- Coprocessor.info : x87 FPU pictures, development and manufacturer information (http://www.coprocessor.info/)

參考文獻

1. Codes（英語）.
2. 參見有符號數處理的Excess-N
3. 原文：Number of base 10 digits that can be represented without change
4. 原文：number of decimal digits, q, such that any floating-point number with q decimal digits can be rounded into a floating-point number with p radix b digits and back again without change to the q decimal digits.